AMIGA

LE LIVRE DU GFA BASIC 3.0

JUSQU'A LA VERSION 3.04

EDITIONS MICRO APPLICATION

LIVRE DATA BECKER

Remerciement

Début 2019, j'ai commencé à acheter tout le matériel Atari dont j'ai déjà été en possession dans les années 80/90. Il en est de même pour les livres qui portent sur la thématique de la programmation. Très tôt, je me suis rendu compte que la réalisation de ce projet pourrait être plus difficile que l'achat d'un Atari ST. Pour me procurer l'occasion de l'édition « Das große Basic 3.5 Buch », j'ai dû payer plus que le prix du nouveau livre à l'époque. !

Mais mon souhait était d'avoir à nouveau ce livre dans ma collection. Étant donné que le livre était en très bon état, j'ai souhaité le garder ainsi. C'est alors que j'ai décidé de le scanner.

Ainsi il m'était devenu possible de feuilleter la version digitale plus facilement, d'y faire des recherches spécifiques et d'y ajouter des signets. Plus tard, je me suis aperçu de la nécessité de mettre cette version numérique à la disponibilité du grand public.

Toutefois, une demande d'autorisation auprès de la maison d'édition « Data Becker Verlag » n'était plus possible dans la mesure où celle-ci a cessé toutes ses activités depuis le 31/03/2014 et qu'aucun successeur légal n'a été désigné. Je me suis donc lancé à la recherche de l'auteur, Uwe Litzkendorf, et je l'ai pu trouver.

Ce fut un grand plaisir pour moi de constater qu'il partageait mes points de vue. Il m'a, de surcroit, suggéré de digitaliser, en plus ce celui-ci, d'autres de ses livres sur le « GFA-Basic ». C'était avec un très grand plaisir que j'ai accepté de mettre en œuvre ce projet. C'est ainsi qu'à travers une simple demande par email, est née une merveilleuse opportunité de numériser ce livre pour le futur et de le mettre gratuitement à la disposition du public.

Une fois de plus, je tiens sincèrement à remercier Uwe de m'avoir accordé cette opportunité.

Thomas Werner
Statut de traitement: 19. mai 2020

Nouvel avant-propos sur la numérisation

Cela a pris beaucoup de temps ! Trente ans après la publication-papier de mes livres, Thomas Werner vient d'achever la digitalisation de la plupart de ceux-ci sous un format PDF interactif et de les publier en ligne gratuitement dans un domaine public. Après un long sommeil, mes livres, devraient désormais être mis à disposition du public.

A mon avis, la programmation orientée objet n'est pas adaptée à une utilisation de masse. Selon mes calculs, seulement 15 % environ de la population est capable d'obtenir des résultats significatifs avec la POO.

Pour être accessible « au plus grand nombre », un langage de programmation doit atteindre au moins 80 % de la population, afin que les enseignants et les élèves puissent se comprendre sans que des cours intensifs d'informatique préalables soient nécessaires. Dans le cas contraire, un fossé se creuse entre l'enseignant et l'élève. Il n'est donc pas étonnant que les langages de programmation Basics procédurales qui sont faciles à utiliser, tels que le populaire GFA-Basic, connaissent une certaine renaissance. En tant qu'ancien auteur de best-sellers, je ne peux que soutenir activement cette tendance. Avec plus de 4.000 pages publiées, et sous les yeux vigilants du public, je dispose de grandes connaissances relatives à la programmation de logiciels.

Non seulement ces connaissances ne sont absolument pas superflues et obsolète, même dans le monde d'aujourd'hui, mais constituent toujours la base des connaissances algorithmiques de base.

Et ce n'est pas tout ! J'ai par ailleurs décidé de développer un nouveau langage de programmation appelé "QS!X©". Il sera similaire au simple Basics-standard en de nombreux points.

"QS!X©" sera disponible en version open source (similaire à LINUX) sur la plateforme HTML5/Canvas 100% inter-compatible, et donc sur tous les systèmes d'exploitation ainsi que dans tous les navigateurs standards.

Vous trouverez de plus d'informations à ce sujet sous :
http://www.litzkendorf.net/invitation_info_e.pdf

Ceci va dans le même sens que la philosophie « classe pour la masse » de Frank Ostrowski, le père du GFA-Basics. Si tout fonctionne comme je l'imagine, la philosophie de Frank Ostrowski pourrait porter ses fruits sur un plan mondial même des années après son décès survenu beaucoup trop tôt. Dus moins dans le domaine de l'informatique, sa philosophie représente la base d'un « langage mondial ». Je suis sûr que cela lui plairait bien !

En souvenir d'un grand homme, avec lequel j'ai eu le plaisir de travailler ensemble et à qui, avec "QS!X©", un monument sera érigé!

Uwe Litzkendorf
Hildesheim, mai 2020

Bleek
Hecht
Litzkendorf

Le livre du GFA Basic AMIGA

Editions Micro Application

MICRO APPLICATION
58, Rue du Faubourg Poissonnière
75010 PARIS

ISBN : 2-86899-207-2

Auteurs : MM BLEEK, HECHT, LITZKENDORF

Traduction française assurée par Christian Frey

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

Table des Matières

CHAPITRE 1

Préliminaires

Le but de ce livre est la présentation détaillée des quelques 360 commandes et fonctions disponibles dans l'interpréteur GFA 3.0. Le souci de les présenter dans leur contexte d'application nous est resté à l'esprit au cours de cet ouvrage. En outre, nous nous sommes efforcés d'assurer une progression à leur exposition afin que leur acquisition soit plus aisée. Les règles de composition respectées dans la présente étude répondent à un souci d'exposition claire et précise.

Toute description de commande, d'instruction ou de fonction est introduite à l'aide d'un en-tête comprenant son nom, son abréviation possible et un bref résumé de son rôle.

Ensuite est présentée sa syntaxe. Puis suit l'explication de son rôle et de ses multiples formes d'insertion au sein d'un programme. Enfin elle est illustrée par une application ou une démonstration. Comme de nombreuses fonctions sont disséminées dans les multiples exemples du présent ouvrage, de brèves remarques concernant leur emploi en ces derniers sont annotées à la fin de leur description.

Tout au long du texte, nous nous sommes tenus aux conventions d'écriture suivantes :

<> Ces deux signes exposent distinctement les noms des touches dans le corps du texte (par exemple : <Shift>, <A>, <Enter>,<HELP>).

[] Le paramètre optionnel d'une instruction est représenté à l'aide de crochets (par exemple : ([:,'] ou [Longueur]). Cela signifie que l'énoncé ainsi distingué n'est d'usage que lorsque l'emploi de l'option recouverte est désiré ou nécessaire.

{} De nombreuses commandes, instructions et fonctions GFA-Basic possèdent une abréviation. L'interpréteur complète de lui-même l'ordre abrégé. Dès qu'une abréviation existe, elle est consignée

dans la ligne du titre à l'aide d'accolades, par exemple { SYS } ou { RET }. Ces accolades sont aussi utilisées lors de l'emploi de certaines commandes BASIC d'accès mémoire (CHAR{}, BYTE{}). Le risque de confusion entre ces deux modes d'usage est minime.

... Les points de suspension représentent des séquences d'instructions délimitant certaines structures (par exemple FOR...NEXT).

Dans la ligne de syntaxe, les noms de commandes, des instructions et des fonctions s'exposent en majuscules. Toutes les variables, les chaînes et les paramètres sont représentés en minuscules, leur initiale mise à part, (par exemple OPENW Handle). Leur introduction à l'aide d'une majuscule permet de les distinguer dans le corps du texte.

Les données de paramètres sont spécifiées à l'aide de désignations ou abréviations propres :

Adresse (ou Adr) :
Adresse désigne un élément physique support de l'information (elles sont spécifiées essentiellement sous forme d'entier de 32 bits)

Nombre (ou Nmb)
Un nombre quelconque

Argument (ou Arg)
Argument de fonction

Matrice
Une quelconque matrice d'éléments

Back=/Var=
Variables de retour d'informations propres aux fonctions

Expression/Expression$ (Expr/Expr$)
Expression numérique/alphanumérique

Indice
Indice d'éléments de matrice

Canal
Identificateur de fichiers

Numéro
Numéro de fenêtre GFA

Texte
Une quelconque chaîne de texte

Variable (ou Var/VAR$)
Un quelconque nom de variable (ne pas confondre avec l'instruction VAR !)

PositionX/PositionY (ou Xpos/Ypos)
Les coordonnées de l'écran

La notification du chemin d'accès d'un fichier, d'un programme ou d'un répertoire s'ordonne selon les règles usuelles de l'AmigaDOS.

Le concept d'expression usité au sein des paramètres, représente un ensemble de constantes, de formules, de textes, de fonctions ou de variables, amenant un résultat déterminé.

A titre d'illustration, une expression arithmétique :

```
A%=B%+(234^2/4.7)*12.97*C%)^2.1317+@fonc(Abc%)
```

une expression alphanumérique :

```
A$="Text"+STR$(A%*B%)+SPACE$(10)+@fonc$(Abc*)+B$
```

Veuillez noter que le résultat de toutes les fonctions (comme de celles définies par vos propres soins) est utilisable à diverses fins, entre autres:

d'affectation :

```
var%=@fonc      fonction personnelle
var%=FRE(0)     fonction BASIC FRE()
```

d'édition :

```
PRINT @fonc     fonction personnelle
PRINT FRE(0)    fonction BASIC FRE()
```

de test :

```
IF @fonc=X      fonction personnelle
IF FRE(0)       fonction BASIC FRE()
```

d'appel factice (sous forme d'argument muet (dummy)) :

```
VOID @fonc=X    fonction personnelle
VOID FRE(0)     fonction BASIC FRE()
```

La notification de cette possibilité n'est pas explicite dans toutes les descriptions de fonction car une telle répétition aurait alourdi l'ouvrage.

La ligne de la syntaxe des fonctions contient la forme d'affectation de variables qui leur est propre, par exemple : var=fonction(). Les appels de fonctions concernent toujours des valeurs ou des chaînes livrées par celles-ci. Leurs insertion et utilisation sont de ce fait analogues à celles des variables ou des chaînes. L'insertion du cardinal : #, lors des accès aux fichiers (OUT et IN mis à part) requérant la spécification d'un numéro de canal, est optionnelle. La version 3.0, celle qui nous intéresse, permet l'omission de l'énoncé GOSUB dans toutes les instructions ON...GOSUB Nom_de_la procédure ; l'interpréteur l'insère de lui-même. Il traduit ainsi par exemple : ON BREAK nom en ON BREAK GOSUB nom.

Chapitre 2

L'Amiga

Le micro-processeur 68000 de Motorola, le processeur principal équipant la plupart des nouvelles machines 16 bits telles le MacIntosh d'Apple et l'Amiga, offre à la différence de ses prédécesseurs de 8 bits, comme par exemple le 6502 du Commodore 64, des instructions machine qui exécutent des opérations que l'on devait programmer soi-même comme la division (on dit que la division est cablée bien que peu de fils soient réellement à l'intérieur du 68000). En plus de ces instructions, il en existe d'autres facilitant la programmation en multi-tâches comme la célèbre instruction TAS (malheureusement déconseillée par Commodore pour des raisons de compatibilité future) et les instructions du mode Superviseur. Sur le plan de la vitesse pure on peut dire que la vitesse de 1Mhz du C64 fait pâle figure à côté des 7.14 Mhz du 68000 sans parler des 25Mhz du 68030 pour les Amigas les plus costauds.

Que signifie une telle vitesse de 7.14 Megahertz ? Le hertz est une unité physique de fréquence par seconde. Les télévisions par exemple connaissent un rafraîchissement d'image d'une fréquence de 50 Hertz. Ce qui veut dire que chaque ligne d'écran est renouvelée 50 fois par seconde. En ce qui concerne notre micro-ordinateur, cela veut dire que les oscillations d'un quartz interne à la machine connaissent une fréquence de 7.14 millions de Hertz et qu'à chaque période (7.14 millions de périodes par secondes !) correspond un état de commutation déterminée. Grâce à cela, la combinaison de 16 voies des données et de 24 voies d'adresses propres du bus de l'Amiga permet de véhiculer, d'entreposer et de traiter en toute simultanéité une quantité d'informations proprement inimaginable.

On pourrait représenter cela tel un immense entrepôt dont les multiples étagères ne contiendraient pas moins de 16 millions de tiroirs. Dans chacun de ses tiroirs serait stockée une information, déterminant l'exploitation de l'entreprise. Et il faudrait maintenant que quelqu'un dans un court laps de temps, sache quelle information précise est

déposée en un tiroir déterminé et quel est son rôle parmi tous les contenus des tiroirs qui déterminent ensemble l'organisation de la marche de la société.

Une telle tâche est indubitablement surhumaine, mais une Amiga 2000, 1000 ou 500 l'accomplit aisément. L'analyse, le calcul et le traitement de cette masse de données lui requièrent certes un peu plus d'une seconde. Malgré cela, sa vitesse suffit amplement à traiter en moins d'un vingtième de seconde, à l'aide de l'interpréteur GFA BASIC, une boucle FOR...NEXT comprenant pas moins de 1000 instructions. Les instructions de cette boucle se décomposent chacune en de multiples opérations traitées par l'ordinateur. Il opère en même temps la vérification de la syntaxe du programme complet, interprète, analyse et exécute les instructions, commandes, fonctions insérées. De telles capacités sont proprement fantastiques et étonnantes. A la différence d'un ordinateur de 8 bits dont les possibilités de combinaisons se laissent encore imaginer, celles spécifiques à un 16 bits ne sont guère concevables.

Chapitre 3

Le GFA Basic

Dès l'origine, un langage de programmation fut livré avec l'Amiga. A la différence des autres ordinateurs dits familiaux où celui-ci était intégré à la machine même, une telle chose n'allait plus de soi pour l'Amiga. Et ses usagers ne pouvaient cacher leur contentement dès que l'on prononçait le vocable : AmigaBASIC.

Le nombre peu ordinaire de commandes, de programmes structurés et de bibliothèques ainsi offerts, amenèrent un nouvel âge de la programmation. Toutefois une goutte d'amertume entachait cette indéniable amélioration : la vitesse. L'AmigaBASIC ne cachait aucunement sa qualité d'interpréteur, on croyait d'ailleurs qu'il en était particulièrement fier.

Les utilisateurs de l'Amiga que vous êtes, ne disposent que depuis aujourd'hui d'un instrument propre à révéler toute la puissance de traitement de leur ordinateur. Frank Ostrowski développa, il y deux ans, la première version du GFA Basic. Il n'y a d'ailleurs pas d'autres langages de programmation qui puissent résoudre de manière si simple les problèmes les plus compliqués comme le fait le GFA Basic.

Cet interpréteur est à même d'assurer au plus grand nombre l'exploitation optimale du MC 68000 (les MC 68010/20/30 n'apporteront d'améliorations qu'en ce qui concerne soit leur vitesse d'horloge plus grande, soit les instructions s'effectuant plus rapidement) à l'aide d'un langage d'un apprentissage aisé. Il conteste de plus en plus la prééminence du langage C. Certes la programmation en C et en assembleur reste requise en certaines circonstances, toutefois il convient de noter que leurs adeptes ne sont pas légion. Peu nombreux sont ceux qui se servent d'un compilateur ou de l'Assembleur qui ne brillent guère par leur simplicité d'usage. Et leur nombre décroîtra puisque ce BASIC répond de loin aux demandes les plus exigeantes.

Il existe deux critiques que l'on peut faire à ce sujet. La première étant que le GFA Basic est plus lent que l'Assembleur. C'est tout à fait exact, mais il faut considérer que dans un programme, il est courant d'avoir 10% du code qui utilisent 90% du temps d'exécution d'où un programme écrit à 90% en GFA Basic et les 10% critiques en assembleur sera pratiquement aussi rapide que son équivalent en langage machine, mais aussi le temps de réalisation sera plus court et le code plus lisible donc plus efficace. La seconde critique beaucoup plus fine vient du fait que les BASIC en général et en particulier celui de GFA ne supportent pas les structures de données telles que les structures du langage C et les RECORDS du Pascal. Nous avons touché ici le gros point faible de ce langage, l'instruction INLINE peut toutefois y remédier élégamment, tout en offrant des possibilités uniques par rapport aux autres langages (Qui perd gagne !).

L'acquisition de ce langage, comme celle de tout autre d'ailleurs, nécessite son apprentissage. La maîtrise des structures essentielles et élémentaires de ce langage est requise, ne serait-ce qu'en vue de la simple communication d'une information à l'interlocuteur (l'interpréteur). Heureusement, ce BASIC, analogue en cela avec ses pareils, se montre d'une tolérance proprement humaine en ce qui concerne les règles syntaxiques.

Toutefois, le GFA Basic se veut heureusement le défenseur d'une programmation rigoureuse et structurée dont l'Editeur (orienté langage) est le fer de lance au grand dam de ceux qui apprécient les charmes de la programmation sauvage en BASIC. Tous ceux qui d'ordinaire utilisent les langages C, Pascal, Modula, n'y seront guère dépaysés, l'habitude de la programmation structurée leur étant acquise.

Toutefois en GFA Basic, chaque commande, instruction ou fonction est isolée en une ligne propre. D'autre part les lignes de programme sont ordonnées distinctement par l'interpréteur même, en considération de la structure logique de l'ensemble. Il est d'ailleurs fort agréable d'éditer le résultat d'une telle structuration graphique et logique. Un autre aspect non moins essentiel qui découle de cela est l'économie de temps réalisée lors de la recherche d'erreurs de programmation. Les lignes de commandes sont visibles d'emblée en l'absence de tout défilement horizontal ou Scroll. De plus les appartenances qu'observent ENDIF et IF ou NEXT et FOR, sont instantanément reconnaissables par l'indentation des blocs d'instructions qu'ils délimitent.

Un des avantages de la programmation structurée est de mettre en évidence les sous-routines qui se répètent en divers endroits du programme. Il vous est ainsi offert de constituer une bibliothèque d'outils recensant procédures et routines d'intérêt général. Il en va de même en principe dans d'autres langages de programmation, néanmoins les circonstances de leur emploi sont relativement rares. La possibilité de la transmission de paramètres aux procédures agrandit encore leur confort d'usage. Que demander de plus ?

La version 3.0 est le fruit des développements antérieurs, menés sur diverses machines. (Pour l'Historique complet, veuillez consulter la Préface de "Développer en GFA" par Frank Ostrowski aux Editions Micro Application). Les modifications ainsi apportées par ces multiples et diverses confrontations ont amené une bien meilleure organisation des variables (l'introduction des variables d'octet et de mot), une amélioration substantielle de l'éditeur et enfin de nombreuses possibilités bien plus rigoureuses de structuration (SELECT-CASE, ELSE IF, FUNCTION, etc.).

Une économie considérable du texte des programmes, donc du temps de développement et une plus grande vitesse d'exécution résultent de ces diverses améliorations. Quelques nouvelles instructions facilitent la programmation de procédures les plus complexes jusqu'à en faire un jeu d'enfant.

La première démonstration des qualités intrinsèques du GFA Basic est offerte par son éditeur. La vitesse fulgurante et l'aisance des recherches, des modifications, des éditions et des défilements de texte y sont des plus manifestes. Cet incomparable éditeur peut être qualifié tel un traitement de texte complet. Une procédure de recherche dans le texte complet ne requiert guère en moyenne plus d'une ou deux secondes.

Last but not least, les nouvelles fonctions de l'éditeur nécessitent leur présentation, en premier lieu <Control><U> qui restitue la dernière ligne effacée avec <Control><Y>, la nouvelle assignation des touches de fonctions la numérotation interne des lignes et la mémorisation chronologique des lignes de commandes en mode direct.

On pourrait me reprocher ma superbe ignorance de tout autre langage, chose en partie véridique, mais n'ayons pas peur de le dire : le GFA Basic est le meilleur langage de programmation existant sur Amiga. J'en suis particulièrement convaincu et je pense que sous peu, nos avis convergeront.

3.1 Quelques observations supplémentaires

Lors de la composition de l'ouvrage, nous ne disposions que d'une version bêta. Elle contenait d'énormes problèmes d'allocation mémoire et n'acceptait que des fichiers inférieurs à 64Ko ! A l'époque, la seule démo qui acceptait de tourner (et encore une fois sur deux...) était celle représentant les 4096 couleurs.

Puis la version 3.00 définitive arriva où l'on sentit des améliorations notables. Cette version fut commercialisée en langue allemande et anglaise. La conception du manuel français permit d'isoler un certain nombre de problèmes dont les plus importants furent corrigés dans la version 3.01 qui fut la première version distribuée en France.

Après cela, ce fut au tour des premiers utilisateurs d'émettre des critiques, des suggestions et de signaler les problèmes qu'ils avaient rencontrés. Ce qui monopolisa l'attention de la plupart fut le port série et la difficulté de l'interface car beaucoup de fonctions ne s'exécutaient pas correctement. Ces fonctions furent améliorées dans la version 3.02, ainsi que beaucoup d'autres dont personne en France ne s'était préoccupé (notamment INLINE). La version 3.03 actuelle offre vraiment beaucoup plus de sécurité et de nouvelles options dans sa barre de menus. Il est vrai que quelques problèmes majeurs persistent (instruction SOUND) mais on voit déjà pointer des solutions (la liste des instructions sonores a été enrichie, ce qui est un bon présage).

3.2 Le menu du GFA Basic

Juste avant la parution de la version 3.01, l'insertion d'un menu dans le GFA BASIC a été opérée. Son éditeur est utilisable à l'aide d'un menu d'Intuition recouvrant quelques commandes, et bénéficie ainsi du confort d'usage de cette interface graphique. L'activation de ce menu s'effectue par la pression continue du bouton droit de la souris, par le positionnement du pointeur de la souris dans la barre des menus de l'écran de l'éditeur. Neuf options dont le détail est brièvement exposé dans ce qui suit, apparaissent :

Load
Charge un programme GFA BASIC. Cette fonction correspond à l'option de la barre du bas et est aussi sélectionnable par la touche F1 ou <Amiga> L.

Save
Sauvegarde un programme BASIC en format GFA. Cette fonction est analogue à celle de la ligne au dessus de la barre et est aussi utilisable à l'aide de <Shift> F1.

New Names
Cette fonction active ou désactive le test des nouveaux noms de variables. D'ordinaire, l'utilisation d'une nouvelle variable amène l'édition d'une boîte de requête demandant confirmation. Celle-ci est maintenant évitée.

Toutefois, cette sélection ne vaut pas pour le mode direct, autrement toutes les anciennes variables s'en trouveraient supprimées.

Run
Lance un programme. La même fonction est assurée par <Amiga> R ou <Shift> F10.

TaskPri0
Règle la priorité de la tâche GFA sur 0 qui est la priorité standard des programme de l'amiga. Grâce à cela, un temps de calcul du micro-processeur plus élevé est réservé aux autres programmes exécutés simultanément. Le GFA BASIC s'en trouve ralentit ! Cette même fonction est assurée par <Amiga> 0.

TaskPri 1
Affecte une priorité privilégiée à la tâche GFA Basic. Les fonctions du programme, le traitement des données et l'éditionde priorité à seront accélérés. Toutefois les autres programmes s'en trouvent ralentis. La même fonction est assurée à l'aide de <Amiga> 1.

CleanUp
Cette option suspend les émissions sonores, la gestion des BOB, des lutins (SPRITE) et leur animation. Le même effet est obtenu à l'aide de <Amiga> C.

Save Icon
L'option Save Icon est apparue avec la version 3.02 du GFA Basic. Elle permet de créer une icône pour les fichiers sources et fonctionne comme un interrupteur.

NewCli
Lance un nouveau CLI sur l'écran du Workbench. Il n'a plus lieu ainsi d'éditer au préalable l'icône CLI. Toutefois, le Shell requiert alors son lancement manuel (version 1.3 du Workbench). Les chemins définis avec la commande Path pendant la Startup-Sequence sont oubliés, problème classique avec ce genre d'utilitaires.

3.3 Le runtime de l'interpréteur

Il est livré avec la version complète de l'interpréteur et permet l'exécution de vos programmes GFA Basic en l'absence du chargement de la version complète de l'interpréteur, l'éditeur compris.

La transmission d'un programme à des amis ou à des connaissances ne disposant pas de l'interpréteur est ainsi assurée. GFA vous autorise d'ailleurs à dupliquer le runtime. Il vous est ainsi possible d'écrire des applications en GFA Basic à des fins lucratives et de les commercialiser en fournissant gratuitement le runtime.

Après avoir lancé Workbench, apparaît la boîte de sélection, où il vous est possible de lancer le programme désiré en y spécifiant le nom du fichier qui le contient. Il vous est aussi possible d'adjoindre le nom du fichier à l'appel du programme dans le CLI ou d'indiquer GFA BASIC:GFABASRO dans la partie DEFAULT TOOL de l'icône du programme.

Chapitre 4

Les éléments de base du BASIC

Toutes les attentes ne peuvent être satisfaites en un seul ouvrage dont le motif est l'exposition d'un langage de programmation. Une initiation au langage informatique n'intéresserait ni les usagers avertis, ni les professionnels. Une étude technique dérouterait fort les débutants. Un certain compromis sous-tend notre étude. D'une part, nous nous sommes souciés d'assurer la présentation claire et détaillée des éléments de base de la programmation, en nous gardant par ailleurs de verser dans les lieux communs, d'autre part, nous nous sommes efforcés de mener des études techniques propres à satisfaire les plus exigeants sans pour autant verser dans un jargon technicien.

En vue d'assurer aux débutants une compréhension claire de la programmation en GFA Basic, nous décrirons d'abord les fonctions élémentaires de ce langage et en complément nous exposerons les termes essentiels du langage informatique. Tous ceux qui pensent en connaître assez sur l'architecture d'un ordinateur, sur l'algèbre booléenne, les divers systèmes numériques, etc., peuvent s'économiser la lecture de ce chapitre.

Toutefois nous conseillons aux débutants d'étudier avec un soin particulier les développements suivants. Les connaissances élémentaires qui y sont traitées, sont impérativement requises pour un usage informatique qui se voudrait tant soit peut probant.

4.1 L'abc de l'informatique

Un ordinateur se compose d'une unité centrale (ce qu'il y a dans la boîte) et de périphériques (lecteurs de disquette, écran clavier, souris, imprimante, etc.). Les périphériques assurent la liaison entre l'unité centrale et le monde extérieur. L'unité centrale se compose principalement d'un processeur, de mémoire et de bus. L'Amiga se distingue encore des autres à cause de Denise, Agnus et Paula (ainsi

que Gary) qui sont les doux noms de ces cicuits spécialisés. La mémoire est l'endroit où sont stockées les informations nécessaires au déroulement des programmes. On la divise en octets, soit 8 bits. La mémoire peut être en lecture seule (mémoire morte ou ROM) ou en lecture-écriture (mémoire vive ou RAM). Je ne ferais pas l'affront aux fiers possesseurs d'une Amiga 1000 de leur expliquer qu'ils possèdent de la WOM (pour les autres, sachez que ces valeureux pionniers possèdent de la mémoire que l'on ne peut écrire qu'une fois et qui est lue sur une disquette KickStart). Par exemple, l'A500 dispose au départ de 512 Ko c'est-à-dire 512 * 1024 octets. Le processeur MC 68000 et la mémoire communiquent à l'aide des bus qui transportent les informations par paquets de deux octets, donc 16 bits. L'Amiga est une machine 16/32 bits car elle peut manipuler également des paquets de quatre octets notamment par l'intermédiaire de ses registres (on peut considérer que ce sont des cases mémoire qui se trouvent à l'intérieur même du 68000 donc d'accès plus rapide). Une dernière particularité de l'Amiga est qu'elle partage une partie de sa mémoire (CHIP-RAM) avec les processeurs graphiques. La CHIP-RAM a une taille de 512Ko mais tout le monde voudrait qu'elle soit deux fois plus grande.

4.2 Bits et octets

C'est très intéressant de savoir qu'une Amiga 500 possède 512 Ko de mémoire mais comment s'en sert-on ? Pour des raisons pratiques, un ordinateur ne reconnaît que deux états que l'on peut identifier par exemple comme étant Ouvert et Fermé, Oui et Non, Branché et Débranché, 0 Volts et 5 Volts, moins de 5 Volts et plus de 5 Volts, D'accord et Pas d'accord ou bien 0 et 1. Encore une fois, pour des raisons pratiques, tout le monde s'est accordé sur le fait de les appeler 0 et 1.0 et 1 sont donc les deux états d'un bit qui est la plus petite unité de l'informatique.

Si vous savez à quoi ressemble un processeur vous avez déjà remarqué des petites pattes en métal que l'on appelle des broches. Par ces broches passe du courant. Imaginons que vous frappiez sur une touche du clavier. Celle-ci ayant eu très mal va se plaindre au circuit qui dirige le clavier, lequel va expédier un rapport au 68000 via un circuit qui s'appelle CIA. Et hop là ! Le courant va changer de sens sur trois des pattes du processeur qui est immédiatement averti. C'est une interruption. Le processeur finit son travail en cours puis va s'occuper de savoir quelle touche a été enfoncée. Vous avez donc compris qu'avec

trois fils, le 68000 sait qu'une entrée du clavier a été effectuée. Si l'on exprime cela en chiffres, on dira que le processeur a reçu une interruption de niveau 2. Le premier fil étant à 0, le second à 1 et le troisième à 0. Voilà pour les bits.

Imaginons maintenant 8 fils, par exemple ceux qui relient votre ordinateur à son imprimante. Les informations arrivent à l'imprimante par paquets de 8 bits (des octets) qui sont traduits en caractères.

4.3 L'arithmétique binaire

L'astuce de ce système numérique est d'attribuer à chaque état représenté par ces 8 fils, une valeur numérique. Ce système numérique été développé afin de représenter exclusivement les états sous-tension et hors-tension sous forme numérique. Il est analogue au système décimal à la seule différence que l'élévation à la puissance n'y obéit pas à la base dix mais à la base 2. L'ordre observé dans un chiffre binaire est identique à celui d'un nombre décimal ; le rang le moins élevé étant à droite, le plus élevé à gauche.

A titre d'illustration, nous prenons un octet quelconque.

```
                                01011101   =   93

En système décimal:

          (10 élevé à la puissance 0) x 3   =    3
        + (10 élevé à la puissance 1) x 9   =   90
                                            ------
                                Résultat:       93
                                            ======

En système binaire:

                        (2 puissance 0) x 1   =    1
                        (2 puissance 1) x 0   =    0
                        (2 puissance 2) x 1   =    4
                        (2 puissance 3) x 1   =    8
                        (2 puissance 4) x 1   =   16
                        (2 puissance 5) x 0   =    0
                        (2 puissance 6) x 1   =   64
                        (2 puissance 7) x 0   =    0
                                              ------
                                Résultat:       93
                                              ======
```

Après de longues recherches et des tests en laboratoire, les plus grands spécialistes mondiaux de l'informatique vous diront que ce code correspond au crochet fermé (]). Votre imprimante devrait en tenir compte si vous lui envoyez.

4.4 Le système hexadécimal

A quoi sert le système hexadécimal ? La représentation d'un octet en système binaire nécessite une chaîne de 8 caractères. Le système hexadécimal répond à cette contrainte en offrant une notation simplifiée des chiffres.

Ce système numérique possède toutes les propriétés des autres systèmes, à la seule différence que la base de l'élévation à la puissance n'y est ni 2, ni 10 mais 16. Son avantage est de permettre la représentation de la moitié d'un octet (4 bits), à l'aide d'un unique chiffre, le plus élevé étant 15 (comment 15 est-il un chiffre maintenant ?). Comme en système décimal, le chiffre du rang des unités le plus élevé est 9, les chiffres 10 à 15 y sont désignés à l'aide de lettres. Au chiffre 10 correspond la lettre A, au 11 la lettre B, au 12 la lettre C, au 13 la lettre D, au 14 la lettre E, au 15 la lettre F (alors F est un chiffre ? De mieux en mieux !).

Le nombre entier signé le plus grand que puisse traiter l'Amiga en GFA Basic : 2^31-1 peut ainsi être représenté à l'aide 8 chiffres seulement : 7FFFFFFF. Les notations de chiffres hexadécimaux sont en règle générale introduites à l'aide d'un dollar : $, par exemple $1AF7. En GFA Basic, cette notation est quelque peu différente puisque les chiffres hexadécimaux y sont précédés du préfixe : &H, par exemple &H1AF7. Les chiffres hexadécimaux économisent en premier lieu le temps et la place. Notre GFA Basic étant international, il comprend aussi la convention d'écriture qu'est le dollar ($) et la transforme selon sa propre syntaxe.

Veuillez aussi consulter à ce sujet les entrées BIN$, HEX$, OCT$.

4.5 Codes et Codes opératoires

En informatique les caractères, les mots, les commandes et les valeurs numériques, à vrai dire toutes les données, sont chiffrés à l'aide de codes. Ainsi en code ASCII (American Standard Code for Information Interchange : le code standard américain d'échange de données), la valeur 65 désigne la lettre "A", 66 la lettre "B", 67 la lettre "C" et ainsi de suite (la table des codes ASCII est consignée en fin d'ouvrage).

Supposons que la valeur 192 représente la commande "édite à la position du curseur, la valeur ASCII contenue dans le prochain octet sous la forme de son caractère correspondant". L'interpréteur serait ainsi informé, suite à la réception de la valeur 192, que l'exécution de l'ordre correspondant est à mener.

La commande pourrait s'énoncer ainsi :

```
décimal:      192         66
binaire:      11000000    10000010
hexadécimal:  C0          42
```

Le signe "B" serait ainsi édité sur l'écran. Dès que nous employons un code assurant une fonction d'initialisation, ordonnant l'exécution d'une procédure déterminée, nous avons affaire à un code opératoire. Un tel code opératoire est une commande représentée symboliquement à l'aide d'un chiffre. Le microprocesseur traite des chiffres transmis en des formats divers sous forme de bits.

Lors de la programmation du microprocesseur, des procédures de traitement que l'interpréteur est à même d'appeler à l'aide de tels codes opératoires ont été prédéfinies. Suite à l'insertion de la ligne :

```
PRINT "B"
```

l'énoncé est chiffré par l'interpréteur en binaire et transmis au microprocesseur afin de traitement. On aurait pu écrire :

```
PRINT CHR$(66) ! (66 est le code ASCII de la lettre "B")
```

mais cela aurait été bien moins commode.

4.6 Les mots et les mots longs

Nul gouffre ne sépare les octets des mots. Le terme mot désigne une unité d'information non pas de 8 bits (1 octet), mais de 16 bits. Le mot peut représenter bien plus de 256 états différents, très précisément 65 536.

Comme nous le mentionnons, le micro-processeur 68000, l'unité centrale de l'Amiga, traite les données selon un format de 16 bits. Il assure ainsi aisément la lecture, le traitement et la retransmission de valeurs comprises entre 0 et 65535. De plus, il est possible d'outrepasser cette dernière limite. Le micro-processeur 68000 est programmé de manière à ce qu'il puisse traiter consécutivement deux mots de 16 bits solidaires, comme s'il s'agissait d'une unité d'information isolée et complète de 32 bits, du moins dès qu'il en est averti. Toutefois, seuls 31 de ces 32 bits sont utilisables pour la représentation des valeurs ; le dernier bit, le plus élevé, étant réservé au signe.

Les chiffres ainsi exprimables suffisent amplement à tous les calculs courants :

```
2^31-1 = 2147483647
```

Cette unité de 32 bits est appelée double mot ou mot long. L'unité centrale a été programmée de manière à ce qu'elle puisse distinguer ces trois unités d'information. Peu lui importe l'unité que revêt l'information transmise, tant qu'est clairement signifiée l'unité dont il est question. A cette fin répondent les instructions : POKE/PEEK servant à écrire et à lire les octets, DPOKE/DPEEK assurant la même chose pour les mots et LPOKE/LPEEK pour les mots longs.

4.7 L'organisation de la mémoire vive

RAM
"Random Access Memory" est la désignation anglaise de la mémoire vive. Elle assure l'écriture en son sein de données et la lecture des informations qui y sont entreposées.

Elle se différencie en mémoire CHIP et mémoire FAST. La mémoire CHIP soutient l'unité centrale le 68000 lors de son travail et est sollicitée par exemple lors des tâches graphiques ou musicales. Il importe de réserver cette mémoire aux données requises par ces applications.

La mémoire FAST ne peut être sollicitée que par le 68000. Elle est de ce fait bien plus rapide, puisqu'aucun cycle d'horloge n'est à réserver à d'autres circuits. Mais ici ne devraient être mémorisés que des données internes et des programmes aucunement nécessités à des fins graphiques ou de gestion des Entrées/Sorties.

ROM
"Read Only Memory", mémoire à lecture seule. Elle est constituée de circuits intégrés dans lesquels sont enregistrées des données non modifiables, par exemple le système d'exploitation intégré à l'Amiga (500 ou 2000) : le Kickstart.

Le Kickstart, le système d'exploitation de l'Amiga
Afin d'assurer une exploitation multitâches, l'Amiga gère sa mémoire vive en fonction de chaque programme en cours. Comme cette gestion répond des programmes exécutés par l'utilisateur et puisque la taille de la mémoire vive d'une Amiga peut varier entre 256 et 9 728 Koctets (256 Koctets pour l'Amiga 1000 en l'absence de toute extension de mémoire et 9,5 Mégaoctets pour l'Amiga 2000 muni d'une carte de 8 Mo), il est impossible de déterminer avec précision quels sont les secteurs de mémoire mobilisés.

Néanmoins, nous sommes en mesure de vous renseigner sur certains faits. Puisque la gestion de la mémoire vive est chose des plus essentielles, peut-être voudriez-vous connaître la taille de la mémoire disponible suite au chargement du GFA Basic ?

C'est une chose des plus aisées. L'écran du Workbench de 2 Bitplanes et d'une résolution graphique de 640 x 256 pixels mobilise à lui seul une mémoire graphique de 40 Koctets. Il convient d'y ajouter pour chaque fenêtre active et les icônes qui y sont contenues, un nombre respectif de Koctets, qui occupent exclusivement de la mémoire CHIP, le Blitter requérant les informations graphiques ainsi contenues. De plus, le GFA Basic ouvre un nouvel écran, celui de l'éditeur. Pour éviter d'utiliser une nouvelle fois autant de place mémoire que le Workbench, il utilise seulement deux couleurs ce qui amène une économie de mémoire substantielle.

En conclusion, l'interpréteur mobilise bien plus de capacité de mémoire. Toutefois, la mémoire libre restante suffit amplement à toutes nos démonstrations, puisque nous nous sommes souciés d'assurer aux possesseurs d'Amiga 500 le bénéfice de nos programmes et démonstrations.

4.8 La logique booléenne

L'exécution d'une tâche par l'ordinateur, répondant à nos attentes personnelles, nécessite le développement préalable d'une procédure dont les modalités opératoires doivent correspondre tant soit peu à la logique de nos propres raisonnements. Laisser la machine résoudre des problèmes mathématiques, son domaine d'excellence, est de moindre utilité. Bien plus significatif serait de lui faire tirer des conclusions en lui proposant a priori un schéma de traitement. Sinon, le bel outil ne serait rien de plus qu'une meilleure machine à calculer.

Un mathématicien anglais du nom de George Boole, conçut une forme d'arithmétique qui porte maintenant son nom.

Certains schèmes de prise de décision y sont modélisés. Les décisions sont induites et déduites à l'aide de conditions isolées ou de combinaisons de conditions. Une situation étant présupposée vraie, on ordonne ses exigences en un modèle théorique dont les diverses alternatives sont strictement logiques. Une condition n'y peut être que vraie ou fausse. Les conjonctions : "ou" et "et" nous permettront de concrétiser cette modélisation de la prise de décision :

1. S'il fait chaud et si j'ai le temps, j'irai à la piscine.

2. S'il fait froid ou si je n'ai pas le temps, je n'irai pas à la piscine.

La prise de décision s'ordonne en cette illustration sur la vérification ou la non vérification des conditions posées a priori. La seconde ligne étant strictement l'inverse de la première, la vérification des conditions de la première induit l'insatisfaction des conditions de la seconde, et inversement. Divers opérateurs permettent de lier les conditions en des combinaisons logiques déterminées.

Ces opérateurs logiques sont :

```
AND, OR, NOT, XOR, IMP, EQV
```

Parmi ceux-ci le non "NOT" représente l'exception qui confirme la règle, puisque qu'il ne s'agit pas d'une combinaison mais d'une inversion, ou négation. Détaillons d'un peu plus près ces conjonctions logiques à l'aide de combinaisons concrètes de données. Les bits du premier opérande y sont combinés à ceux du second, selon leur rang respectif.

AND :

```
0 AND 0 -> 0
0 AND 1 -> 0
1 AND 0 -> 0
1 AND 1 -> 1

          11011001     (octet 1)
    AND   01101101     (octet 2)
    ---------------
          01001001     résultat


PRINT BIN$(&X11011001 AND &X01101101)
PRINT &X11011001 AND &X01101101
```

Le bit de l'octet résultant n'est activé que s'il est activé dans l'un et l'autre opérande.

OR :

```
0 OR 0 -> 0
0 OR 1 -> 1
1 OR 0 -> 1
1 OR 1 -> 1

          11011001   (octet 1)
    OR    01101101   (octet 2)
    ---------------
          11111101   résultat


PRINT BIN$(&X11011001 OR $X01101101)
PRINT &X11011001 OR &X01101101
```

Le bit de l'octet résultant est toujours activé dès qu'il est activé dans l'un ou l'autre opérande.

XOR:

```
0 XOR 0 -> 0
0 XOR 1 -> 1
1 XOR 0 -> 1
1 XOR 1 -> 0

          11011001  (octet 1)
    XOR   01101101  (octet 2)
   ----------------
          10110100  résultat


PRINT BIN$(&X11011001 XOR &X01101101)
PRINT &X11011001 XOR &X01101101
```

Le bit de l'octet résultant est activé s'il est activé dans l'un ou l'autre opérande et non pas dans les deux à la fois. XOR (Exclusif OR) est un ou exclusif. Veuillez observer que les trois opérations suivantes amènent un résultat sous forme de mot long. Cela est dû à l'assignation d'un signe (de négation) à l'énoncé obtenu (Cf. variables booléennes).

IMP:

```
0 IMP 0 -> 1
0 IMP 1 -> 1
1 IMP 0 -> 0
1 IMP 1 -> 1

              11011001            (octet 1)
    IMP       01101101            (octet 2)
   ------------------------------
11111111111111111111111101101111   résultat


PRINT BIN$(&X11011001 IMP $X01101101)
PRINT &X11011001 IMP $X01101101
```

Le bit résultant est toujours activé s'il est activé dans chacun des opérandes ou désactivé dans chacun des deux, ou activé dans le second opérande et désactivé dans le premier. Cela s'appelle une implication. Le résultat est faux (donc signifié par 0) si le bit est activé dans le premier octet et désactivé dans le second. Dans tous les autres cas, le résultat est vrai (donc signifié par 1).

EQV:

```
0 EQV 0 -> 1
0 EQV 1 -> 0
1 EQV 0 -> 0
1 EQV 1 -> 1

                11011001            (octet 1)
    EQV         01101101            (octet 2)
   ____________________________
11111111111111111111111101001011   résultat


PRINT BIN$(&X11011001 EQV $X01101101)
PRINT &X11011001 EQV $X01101101
```

Cet opérateur est l'inverse de XOR. Le bit résultant est activé s'il est activé dans chacun des deux opérandes ou désactivé dans tous les deux. Si le bit est activé dans l'un et non pas dans l'autre, le résultat est faux (donc signifié par un 0).

NOT:

```
NOT 0 -> 1
NOT 1 -> 0

        NOT     11011001
        ________________
11111111111111111111111100100110   résultat


PRINT BIN$(&X11011001 NOT $X01101101)
PRINT &X11011001 EQV $X01101101
```

Le bit résultant est activé s'il est désactivé dans l'octet initial. Le résultat est toujours le négatif de l'octet initial.

L'addition, la soustraction, la multiplication et la division s'effectuent en système binaire de manières analogues à celles observées dans le système décimal.

Addition:

```
                        11011001   (= 217 en système décimal)
                      + 01101101   (= 109 en système décimal)

                      ------------
1er  bit = 1+1      =          0   (reste 1)
2ème bit = 0+0+reste =        1    (reste 0)
3ème bit = 0+1      =       1      (reste 0)
4ème bit = 1+1      =      0       (reste 1)
5ème bit = 1+0+reste =    0        (reste 1)
6ème bit = 0+1+reste =   0         (reste 1)
7ème bit = 1+1+reste =  1          (reste 1)
8ème bit = 1+0+reste = 0           (reste 1)
9ème bit = reste     = 1

                      ------------
                     101000110     (= 326 en système décimal)
```

```
PRINT BIN$(&X11011001+$X01101101)
PRINT &X11011001+$X01101101
```

Soustraction:

```
                        11011001   (= 217 en système décimal)
                      - 01101101   (= 109 en système décimal)

                      ------------
1er  bit = 1-1      =          0   (reste 0)
2ème bit = 0-0      =         0    (reste 0)
3ème bit = 0-1      =        1     (reste 1)
4ème bit = 1-1-reste =      1      (reste 1)
5ème bit = 1-0-reste =     0       (reste 0)
6ème bit = 0-1      =    1         (reste 1)
7ème bit = 1-1-reste =   1         (reste 1)
8ème bit = 1-0-reste =  0          (reste 0)

                      ------------
                       1101100     (= 108 en système décimal)
```

```
PRINT BIN$(&X11011001-$X01101101)
PRINT &X11011001-$X01101101
```

Multiplication: 11011001 * 01101101 (= 217 x 109)

```
                      ----------------------
  1er  bit/OP1 * OP2              01101101   (rang 2^0)
+ 2ème bit/OP1 * OP2            + 00000000   (rang 2^1)
  somme intermédiaire =           001101101
+ 3ème bit/OP1 * OP2           + 00000000    (rang 2^2)
  somme intermédiaire =           0001101101
+ 4ème bit/OP1 * OP2          + 01101101     (rang 2^3)
  somme intermédiaire =         01111010101
+ 5ème bit/OP1 * OP2         + 01101101      (rang 2^4)
  somme intermédiaire =         101010100101
+ 6ème bit/OP1 * OP2        + 000000000      (rang 2^5)
  somme intermédiaire =       0101010100101
+ 7ème bit/OP1 * OP2       + 01101101       (rang 2^6)
```

```
   somme intermédiaire =      10010111100101
+ 8ème bit/OP1 *          + 01101101          (rang 2^7)

             Résultat:      101110001100101

PRINT BIN$(&X11011001*$X01101101)
PRINT &X11011001*$X01101101
```

La division de nombre entier

La division en système binaire est chose relativement fort complexe. Pour cette raison, nous ne vous proposerons qu'un exemple simple. Une description détaillée nous mènerait trop loin. Seul un résultat en nombre entier est envisagé. Certes, la division de nombres réels peut être entreprise de cette manière, mais cette unique opération nécessiterait quelques dizaines de pages à elle seule.

Le format de l'octet a été choisi intentionnellement pour ces divers exemples, car c'est le format de loin le plus usité. Evidemment, ces opérations peuvent être effectuées en un format différent.

Cette thématique n'est guère l'une des choses les plus simples de l'informatique. Certes l'utilisation des opérateurs logiques et des opérations binaires n'est nécessitée qu'à des stades avancés de la programmation. Si jamais vous désiriez approfondir vos connaissances en programmation, il pourrait se montrer fort utile que vous en ayez pris connaissance au préalable.

4.9 Conditions et conséquences

Les précédentes conclusions ne sont pas à prendre au pied de la lettre, puisque les opérateurs booléens AND et OR sont très importantes lors de l'utilisation de certaines commandes BASIC qui requièrent la combinaison de conditions particulières.

Prenons par exemple la structure de test IF...ELSE...ENDIF. Elle est indubitablement la plus usitée pour lier le déroulement du programme à la satisfaction préalable d'une condition ou d'une combinaison de conditions. Les deux prises de décision conditionnelle suivantes ont d'ailleurs déjà été exposées plus haut :

1. S'il fait chaud et si j'ai le temps, j'irai à la piscine.

2. S'il fait froid ou si je n'ai pas le temps, je n'irai pas à la piscine.

Deux conditions sont présentes à chacune des propositions. La conséquence qui y est portée est effectivement induite suite à la satisfaction des conditions énoncées.

Afin de traduire ces propositions en langage informatique, nous nous servirons de symboles numériques. Chaque expression affirmative est codée à l'aide d'un 1, chaque expression négative à l'aide d'un 0.
Nous codons ainsi :

```
faire = 1
avoir = 1
aller = 1
ne pas avoir = 0
ne pas aller = 0
```

Insérées dans une structure conditionnelle IF, les précédentes propositions s'énoncent de la manière suivante :

```
IF chaud = 1 And  temps = 1 ! S'il "fait" chaud ET si j'"ai" le temps
piscine = 1                 ! donc je "vais" à la piscine
ENDIF                       ! fin de la conséquence
IF froid = 1 OR temps = 0   ! S'il "fait" froid OU si je "n'ai pas" le temps
   piscine = 0              ! donc je "ne vais pas" à la piscine
ENDIF                       ! fin de la conséquence
```

Vous observez que les alternatives signifiées à l'aide des symboles représentant oui ou non sont des plus simples, mais que la prise de décision en devient bien plus complexe. L'adjonction d'une alternative plus complexe à l'aide d'une possibilité offerte par le GFA Basic nous permet d'affiner la prise de décision. Il est possible de chaîner plusieurs conditions à l'aide de leur mise entre parenthèses et donc de créer une alternative.

Une pareille alternative pourrait en notre exemple s'énoncer :

```
... ou si la piscine couverte est ouverte, ...
```

La proposition complète s'énoncerait alors ainsi :

```
S'il fait chaud ou si la piscine couverte est ouverte et si
j'ai le temps, j'irai à la piscine.
```

Nous représentons l'affirmation "la piscine est ouverte" à l'aide du chiffre 1.

L'instruction conditionnelle s'énonce ainsi :

```
IF (chaud=1 Or piscine=1) And temps=1! S'il "fait" chaud OU si la piscine couverte
"est" ouverte
                                    ! ET si j'ai le temps
piscine=1                           ! donc je vais à la
                                    ! piscine
ENDIF                               ! fin de la conséquence
```

Le facteur "temps" se rapporte aux deux alternatives. Si par exemple je décidais d'une autre alternative, aller à la piscine sans pour autant disposer du temps nécessaire, il faudrait placer les parenthèses différemment, pour que "temps" n'en réfère plus qu'à "chaud".

```
IF (chaud=1 And temps=1) OR temps=1! S'il "fait" chaud ET si
                                   ! "j'ai" le temps
                                   ! OU si la piscine
                                   ! ouverte est couverte
piscine=1                          ! donc je vais à la  piscine
ENDIF                              ! fin de la conséquence
```

L'alternative s'énonce maintenant ("chaud" et "temps") ou "ouverte", à la différence de l'exemple précédent où elle s'énonce "chaud" ou "ouverte".

Il convient de noter ici qu'une instruction IF se clôt toujours avec ENDIF pour que l'interpréteur sache quelle conséquence est liée aux conditions introduites par la première.

Il est aussi possible d'énoncer une conséquence alternative.

Lorsque je dis qu'à certaines conditions, je ferai une certaine chose, il s'ensuit implicitement que si les conditions requises sont insatisfaites, je ferai autre chose.

Cela pourrait se formuler ainsi :

S'il fait chaud ou si la piscine couverte est ouverte et si j'ai le temps, j'irai à la piscine. Sinon je n'irai pas à la piscine et je lirai.

Nous représentons l'affirmation "je lirai" de la manière habituelle avec un 1 et la négation "je ne lirai pas" avec un O.

L'expression ELSE veut dire "sinon", "autrement", "dans le cas contraire". ELSE est donc l'inversion conséquente des conditions introduites par IF. Les conséquences contenues entre ELSE et ENDIF ne sont induites que lors de l'insatisfaction de toutes les conditions portées dans IF.

Autrement dit, dès que le programme rencontre une instruction IF complétée par ELSE, les conséquences liées à IF ou celles liées à ELSE sont validées. Pour éviter cette alternative, il suffit de laisser l'instruction ELSE de côté. La non-satisfaction des conditions introduites par IF n'a alors d'autre conséquence que la poursuite du programme après l'instruction ENDIF afférente.

Pour éviter d'insérer toutes les conditions en une même ligne de programme ou pour définir des séquences de conditions multiples, il est possible d'imbriquer les instructions IF les unes dans les autres. Ce concept d'imbrication est fort usité dans les structures de boucle comme vous vous en rendrez compte par la suite. Sous ce terme est compris la possibilité d'adjoindre de nouvelles conditions à celles initiales, permettant ainsi des sous-branchements.

La prise de décision promue par l'exemple précédent s'ordonne en deux étapes. En premier lieu, le facteur temps n'est pris en considération que si les deux premières conditions "il fait chaud" et "la piscine est ouverte", sont satisfaites. S'il ne fait pas chaud et si la piscine n'est pas ouverte, le facteur temps est écarté et la décision "je lirai" s'ensuit.

Nous espérons qu'à l'aide de ces simples illustrations, certains ponts entre les formes habituelles de raisonnement et celles propres aux langages informatiques ont été posés. L'exercice des fonctions envisagées vous familiarisera avec le formalisme informatique et aiguisera votre curiosité quant à leurs multiples applications possibles. Il est parfois littéralement fascinant d'observer un traitement d'informations induit par une imbrication complexe de propositions conditionnelles aux conséquences multiples et fort diverses. On en retire l'impression d'une intelligence de la machine, bien que seul le génie créatif et la virtuosité du programmeur en soit la cause.

Et puis ces 1 et 0 deviennent ennuyeux. Vraiment pas lisibles ! Il serait tellement plus agréable de dire Vrai ou Faux. Et bien ils existent en GFA Basic. Faux s'écrit FALSE et correspond à 0 et Vrai s'écrit TRUE et correspond à -1. Si vous exécutez :

```
PRINT BIN$(TRUE)
```

vous verrez apparaître 32 signes 1 dans la fenêtre de sortie, on comprend donc que TRUE = NOT FALSE = NOT 0 (vrai=pas faux, logique non ?).

4.10 Flags

Un des principes essentiels de la programmation effective fut précédemment exposé. On le nomme flag ou en langue française drapeau. Son insertion se montre capitale en chaque prise de décision qui requiert la signification précise d'un état propre à influencer la décision finale. Sollicitons une nouvelle fois notre exemple :

```
IF chaud=TRUE OR piscine=TRUE
   flag=TRUE
ENDIF
... suite du programme...
IF flag=TRUE  AND temps=TRUE
   piscine=TRUE
ENDIF
```

Un état induit par des conditions précises est ainsi assigné au flag à des fins d'exploitation ultérieure. Les informations affectées au drapeau peuvent être restituées à n'importe quel endroit du programme.

Pour éviter de répéter à chacun de ces endroits la prise de décision et la définition de la proposition conditionnelle, s'il fait chaud ou si la piscine est ouverte, il suffit d'énoncer les termes de cette prise de décision au plus tôt et de mémoriser l'information résultante, que la décision obtenue soit positive ou négative, en une variable et d'ordonner la restitution du contenu de la variable à chaque endroit désiré.

Il est possible que l'on sache déjà que le temps sera demain au beau fixe mais que l'on ne sache pas aujourd'hui si demain, on aura le temps d'aller à la piscine. Ainsi est-on amené à réserver la seconde réponse à de plus amples informations. Nous avons nommé délibérément cette variable flag ; bien entendu, elle peut être nommée différemment.

Améliorons une dernière fois le dernier programme. Quand le GFABasic teste par exemple IF flag=TRUE, il soustrait flag et TRUE (qui contient 32 bits à 1, soit -1) et renvoie comme résultat TRUE s'il a obtenu 0 et FALSE si la différence n'est pas nulle. De plus, le IF ne teste

que la nullité (0 on ne branche pas et non nul on branche). Le programme fonctionnera tout aussi bien (et beaucoup plus vite) de la manière suivante :

```
IF chaud OR piscine
   flag=TRUE
ENDIF
...
IF flag AND temps
   piscine=TRUE
ENDIF
```

Et on remplacerait de même :

```
IF chaud  =FALSE
par
IF NOT chaud
```

4.11 Les variables

Les variables jouent dans un programme le même rôle qu'elles remplissent en mathématique. Elles sont insérées en tant que réceptacles de grandeurs ou d'expressions (numériques ou alphanumériques), dont les contenus ne sont communiqués, affectés et établis que lors du déroulement du programme et peuvent chager de valeur pendant l'exécution. Pendant le développement du programme, soit l'on sait uniquement QUE quelque chose sera affecté en cette variable, mais non pas ce QUI très précisément y sera contenu, soit on leur affecte dans le listing du programme des contenus que l'on pense nécessités aux lieux spécifiés.

Afin d'éviter tout malentendu, nous préciserons que d'une certaine manière, dans le premier cas, on sait déjà ce "que" cette variable mémorisera, on ne sait seulement pas quel contenu concret y sera affecté. La détermination du type de variable à insérer est requise au préalable. Il existe deux types élémentaires différents de variables : les variables numérique, contenant des valeur, et les variables alphanumériques contenant des chaînes de caractères ou de texte. Les variables numériques servent à mémoriser des valeurs :

```
Hypo=SQR(22^2+13^2) ! Hypo=racine carrée de 22 puissance 2
                                          + 13 puissance 2
```

Cet exemple en appelle à vos souvenirs scolaires. Les deux longueurs des côtés de l'angle droit d'un triangle rectangle sont élevées à la puissance 2 et leurs carrés sont additionnés. La somme ainsi obtenue est réduite à la racine carrée selon le théorème de Pythagore. La longueur de l'hypoténuse de notre triangle rectangle résultant de cette formule est placée dans la variable "Hypo". Tant qu'aucune autre valeur n'est transmise à cette variable, elle contient la longueur de l'hypothénuse du triangle rectangle de notre exemple. Cette valeur peut être appelée et restituée en n'importe quel lieu du programme ou aussi modifiée suite à de nouvelles indications (Note : On obtient le ^ avec l'accent circonflexe suivi d'un espace).

Il existe cinq types différents de variables numériques. La mémorisation d'un nombre réel, d'un nombre avec des rangs décimaux, demande simplement l'indication du nom de la variable, en l'absence de tout autre ajout. Ce type est présent dans notre exemple précédent. En règle générale, il nécessite pour la mémorisation de la valeur contenue, une place mémoire de 8 octets. Une précision jusqu'au treizième rang du chiffre est ainsi assurée. Les chiffres dépassant cette limite sont arrondis automatiquement.

```
A=123,125237667231 donne A=123,12523767
```

Les nombres de parties entières de plus de 13 rangs sont transformés automatiquement sous forme exponentielle :

```
A=643453017623.527 donne A=6.434530176235E+11
```

D'un autre côté, les indications de valeurs, pouvant être représentées en tant que chiffre "normal" et notifiées sous forme exponentielle, sont traduites dans le format normal.

```
A=1284,55E+7 donne A=128455000
```

Les notifications de chiffres sous forme exponentielle peuvent s'échelonner de 2.22507385807E-308 à 3.595386269725E+308. Une valeur exponentielle se lit de la manière suivante :

```
54,6341E+7 correspond à 54,6341 * (10^7)
```

Un autre type est représenté par la variable entière. Seuls des nombres entiers, autrement dit des valeurs ne contenant pas de décimales y sont affectés. A vrai dire, il est possible de leur affecter des nombres réels mais ceux-ci ne sont pas mémorisés en tant que tels, les chiffres après la virgule étant oubliés, oblitérés.

```
A%=149.523
PRINT A%
==========> 149
```

La signification à l'interpréteur de l'emploi d'une variable de type entière requiert l'adjonction du suffixe : % au nom de la variable, ce qui s'énonce en notre exemple : A%. Ce type nécessite une quantité de mémoire de 4 octets par variable, ce qui autorise la mémorisation de valeurs comprises entre -2147483648 et 2147483647. En outre, des variables entières de 1 et 2 octets sont utilisables.

Le troisième type numérique est la variable booléenne. Seules deux valeurs peuvent lui être affectées. Par la suite, lors de l'étude et de l'exercice des instructions isolées du BASIC, de nombreuses fonctions ne délivrant que deux valeurs différentes se présenteront. L'une de ces valeurs est le zéro. L'interpréteur la compte comme valeur de vérité 0. Bien qu'elle soit nommée valeur de vérité, cette valeur représente une constatation de fausseté. Elle signifie "faux". L'autre valeur de vérité est représentée par le nombre -1. Cette valeur représente une constatation de vérité et signifie "vrai".

Quant à savoir pourquoi la constatation de vérité est représentée à l'aide d'un -1, cela est dû au dit codage en complément à deux. Tous les bits d'un mot long sont activés, donc aussi le plus élevé dont résulte le signe négatif. Les plus attentifs se sont posés éventuellement lors de nos descriptions des mots et des mots longs, la question de la limitation de la valeur maximale à 2^31-1. Ceci tient à ce que le bit le plus élevé est réservé à l'assignation du signe du nombre. Dès que se présente un nombre négatif, le bit le plus élevé d'un mot long est activé et le nombre, le signe mis à part, est retranché à 2^31. Le motif de bits résultant est interprété en tant que nombre négatif codé en complément à deux et retourné sous la forme appropriée.

```
PRINT BIN$(1)
====>  00000000000000000000000000000001  ( 1 Bit)

PRINT BIN$(2^31-1)
====>  11111111111111111111111111111111  (31 Bits)

PRINT BIN$(-2^31-1)
====> 10000000000000000000000000000000  (32 Bits)

PRINT BIN$(-2^31+1)
====> 10000000000000000000000000000001  (32 Bits)
```

```
PRINT BIN$(-2^31+2)
====> 10000000000000000000000000000010  (32 Bits)

PRINT BIN$(-2^31+2^15)
====> 10000000000000001000000000000000  (32 Bits)

PRINT BIN$(2^31-2^15)
====>  11111111111111110000000000000000  (31 Bits)

A%=-1
PRINT BIN$(A%)
====> 11111111111111111111111111111111  (32 Bits)
```

Revenons aux valeurs de vérité et illustrons leur emploi à l'aide de l'ordre EXIST. Celui-ci vérifie l'existence ou l'inexistence d'un fichier spécifié sur une disquette. Le fichier étant disposé sur la disquette, EXIST délivre la valeur -1, sinon la valeur 0 est retournée. Cette fonction n'assure la délivrance d'aucune autre valeur, puisque seuls deux états peuvent se présenter : soit le fichier existe, soit il est inexistant. A travers cette illustration, la signification propre d'une variable booléenne apparaît en plein jour. La prise de décision y résulte de la vérification ou de la non vérification d'une donnée précise.

Une variable booléenne ne peut revêtir que ces deux valeurs. Même lors de l'attribution de valeurs différentes à celles-ci, l'interpréteur ne distinguera toujours qu'entre vrai et faux, -1 et 0. Toute valeur démentant les limites propres à ce type de variables et différente de 0, est interprétée et mémorisée automatiquement telle que vraie, donc -1. Ce type de variable a l'avantage de ne requérir qu'une quantité de mémoire de 2 octets par variable. La déclaration d'une variable booléenne s'ordonne avec l'adjonction du suffixe "!" au nom de la variable, par exemple VAR!.

A l'inverse des précédentes, les variables de texte ou de chaîne (string en langue anglaise) servent à entreposer non pas des valeurs, mais uniquement des caractères. A vrai dire, ces caractères sont représentés à l'aide de valeurs, comme nous l'avons vu précédemment lors de l'étude des codes ASCII. Mais lors de leur insertion, l'interpréteur sait que les valeurs contenues ne sont pas à traiter en tant que valeurs mais en tant que codes ASCII, du moins si la déclaration de type de variable de chaîne a été opérée au préalable. Cette dernière s'effectue par l'adjonction du suffixe : $ au nom de variable, par exemple VAR$.

Le nombre de caractères d'une variable de chaîne GFA BASIC s'échelonne entre 0 à 32767. Ceci ne signifie pas que chaque variable de chaîne requiert une quantité de mémoire de 32767 octets (1 caractère ASCII occupe un octet). La quantité de mémoire occupée par une telle variable dépend étroitement du nombre de caractères qui y sont affectés. Le nombre d'octets de mémoire occupés équivaut à celui des caractères contenus dans la variable. Six autres octets sont nécessités en complément par chaque variable de chaîne.

A chaque variable de chaîne est attribué un descripteur qui consigne l'adresse : la localisation de la variable dans la mémoire vive et sa taille (pour de plus amples informations, veuillez consulter ARRPTR et "Organisation des types de variables").

```
A$="BASIC"
PRINT "La chaîne contient ";LEN(A$); "caractères"
PRINT "Son adresse est ";VARPTR(A$)
PRINT "Le descripteur d'A$ est placé à l'adresse ";ARRPTR(A$)
PRINT "Le premier octet de la chaîne a la valeur
";PEEK(VARPTR(A$))
PRINT "Cette valeur ";PEEK(VARPTR(A$))" représente le caractère
";
PRINT CHR$(PEEK(VARPTR(A$)))
```

En premier lieu, la chaîne "BASIC" a été attribuée à la variable A$. Puis à l'aide de la fonction LEN est établit la longueur de la chaîne. L'adresse mémoire du premier caractère (octet) de cette expression nous est délivrée par la fonction VARPTR (abréviation qui signifie pointeur de variable). Grâce à la fonction de lecture de mémoire PEEK, la valeur ASCII du premier caractère de la chaîne précisément pointée par VARPTR est retournée. En conclusion, la fonction de texte CHR$ transforme la valeur précédemment lue en un caractère qui n'est autre que celui qui introduit la chaîne considérée.

L'addition d'un 1 à l'adresse renvoyée par VARPTR donne l'adresse du second caractère; l'addition d'un 2, celle du second caractère et ainsi de suite.

L'édition sur l'écran du contenu complet de la variable s'ordonne à l'aide de l'énoncé suivant :

```
PRINT A$
```

4.12 Matrices et vecteurs

Cela sonne presque comme un drame de Shakespeare. Notre propos n'est guère de vous ennuyer. Puisque ces termes opératoires se montrent d'une utilité certaine sinon requise, il convient d'étudier brièvement les présents réceptacles de variables. On les nomme matrices ou en langue anglaise arrays, ce qui signifie tableaux. Vous savez certainement que l'on nécessite, lors du calcul d'une courbe de fonction, au moins deux grandeurs. La plupart du temps, il devait s'agir des grandeurs "X" et "Y" représentant respectivement l'abscisse et l'ordonnée. Le calcul de l'ordonnée Y s'effectuait pour chaque valeur déterminée d'"X" à l'aide d'une équation de fonction. Les points d'intersection de ces deux grandeurs représentaient les points de la courbe.

Ces deux valeurs en étaient les coordonnées. Afin d'éviter la confusion possible avec d'autres valeurs mémorisées, traitées par l'ordinateur, elles sont assignées en des tableaux. Cette procédure est fort analogue à celle que certains nomment table de valeurs. Une telle table de deux dimensions où chaque dimension se représente à l'aide d'un vecteur est aussi appelée matrice.

Un vecteur (au sens de vecteur d'unité) est un champ ou une matrice unidimensionnelle, servant à contenir une certaine catégorie de données désignées à l'aide d'un même libellé ou nom de variable.

De plus amples informations sur l'application des matrices sont présentées sous DIM (création d'une matrice multidimensionnelle). La variable booléenne mise à part, tous les autres types de variables occupent dans une matrice la même quantité de mémoire que celle de leur forme isolée. Exceptionnellement, une variable booléenne ne nécessite dans un tableau qu'un bit.

4.13 Quelques conseils élémentaires

L'un des freins les plus puissants à l'apprentissage d'un langage informatique est la confusion par le débutant des instructions (appartenant au langage de programmation) et des noms (délibérément choisis) contenus dans un même listing de programme. Quelques règles simples permettent d'éviter cette confusion.La première consiste à

apprendre par coeur aussi rapidement que possible les noms des instructions. Tous les noms inconnus rencontrés ensuite ne sont que des concepts délibérément choisis. Cette déduction est certes d'une extrême banalité, mais elle est logiquement irréfutable.

Mais est-ce aussi simple alors que le GFA Basic ne recèle pas moins de 360 commandes, instructions, noms de variables réservés et matrices dont la connaissance est requise ?

Il est possible dans le GFA Basic d'Amiga, de dénommer sans aucun problème les variables avec les noms des instructions. Il n'y existe pas de mots réservés, les variables clés (TIMER, DATE$, etc.) étant comprises. Il était déjà possible dans les précédentes versions d'utiliser délibérément des noms d'instructions dans les procédures. Une procédure est toujours introduite par l'énoncé PROCEDURE, une fonction par DEFFN. Un label est toujours isolé ou éventuellement complété d'un commentaire (!commentaire), ponctué en fin par un double point :

```
Data_label:
```

ou

```
XYZ_label.1: ! Commentaire abc...xyz
```

Les appels de procédures se font simplement par leur nom (éventuellement précédé d'un arrobas), alors que les appels de fonctions débutent à l'aide de FN ou comme précédemment avec l'arrobas.

```
GOSUB Procédure1 ou @Procédure1
XY%=FN Fonction1 ou XY%=@Fonction1
```

Lors de l'emploi de labels, seuls des énoncés correctement déchiffrables par l'interpréteur sont utilisables. Par exemple, un label appelé save est interprété comme étant la commande SAVE, ou le nom fileselect sera traduit dans l'énoncé : FILES "elect".
L'emploi de PRINT comme nom de label est possible, mais nous vous le déconseillons en raison d'une meilleure lisibilité de vos programmes.

Comme nous le disions auparavant, les déclarations de types de variables s'opèrent à l'aide de l'adjonction d'un suffixe, les variables réelles mises à part, le suffixe # étant optionnel. Une variable entière est désignée à l'aide du suffixe "%" (VAR%), une variable octet à l'aide

d'un "|" (VAR|), une variable de mot à l'aide d'un "&" (VAR&), une variable booléenne d'un "!" (VAR!) et une variable de chaîne à l'aide d'un "$" (VAR$). Elles sont ainsi aisément reconnaissables.

Un nom peut être composé des caractères de texte habituels ([A-Z], [a-z], [0-9]); l'insertion du point et du souligné est aussi admise. Toutefois, les noms de variables et de fonctions doivent toujours commencer par un caractère.

```
N.om_de_vari_able.1
Titre_de_tableau.XYZ%(DIM1,DIM2,...)
1:          <--- est interprété en tant que label!
PROCEDURE 1724_de_A.à.Z
DEFFN hardcopy=X%Y%
```

L'éditeur BASIC écrit toutes les mots-clés en lettres majuscules et les variables en minuscules, assurant ainsi la lisibilité des listings de vos programmes, ce qui facilitera grandement la tâche des débutants.

Dans un futur proche, suite à quelques exercices en GFA BASIC, vous vous passerez aisément de cette aide, puisque la logique inhérente à la syntaxe des instructions trace d'elle-même la compréhension de leurs propres énonciations. Un nom suivi d'un opérateur logique, ne peut être qu'une variable. Un énoncé introduit par l'arrobas "@" et positionné en début de ligne indique un appel de procédure. A l'inverse, si l'énoncé précédé de l'arrobas "@" est précédé d'un opérateur logique, il ne peut s'agir que d'un appel de fonction et ainsi de suite ...

Comme nous le suggérions dans nos remarques, tenez-vous en au début à l'apprentissage des ordres élémentaires (PRINT, INPUT, READ, DATA, PEEK, POKE, GOSUB, etc...) et laissez les autres ordres et fonctions de côté. Dès que la maîtrise des instructions élémentaires est assurée, l'assimilation des autres se fera progressivement et aisément.

4.14 Les structures de boucle

Une boucle sert à exécuter une série d'instructions plusieurs fois. Le GFA BASIC propose quatre structures de boucles distinctes :

1. La boucle FOR...NEXT

2. La boucle DO...LOOP

3. La boucle REPEAT...UNTIL

4. La boucle WHILE...WEND

En premier lieu, nous détaillerons les deux structures utilisant des tests.

La boucle REPEAT...UNTIL (Répéter...Jusqu'à <condition>)

Le branchement dans la boucle est assuré par l'instruction REPEAT. Celle-ci est suivie d'une série d'instructions ou autrement dit d'un bloc de programme. L'originalité de cette boucle réside dans le fait que le test de condition n'est opéré qu'en fin de boucle. La condition insérée détermine le nombre d'exécutions de la boucle et dans quelles conditions l'abandon de la boucle est opérée. L'instruction UNTIL marque la fin de la boucle et la condition détermine sa reprise ou son abandon.

```
REPEAT
  INC A
  B=SQR(A)
  PRINT "Racine de ";A;" = ";B
UNTIL B=15 OR A=200
```

A l'intérieur de la boucle est incrémenté (augmenté d'un) un compteur "A". Ensuite on calcule la racine carrée de A. Le résultat est affiché avec la commande PRINT. La boucle s'arrête dès que B vaut 15 ou A 200. Cette boucle est tout à fait exemplaire de ce que sont les pièges de la programmation qui font que beaucoup de programmes se plantent sans raison apparente. D'après vous, quand la boucle va-t-elle s'arrêter? La réponse est : "Je n'en sais rien, peut-être jamais !". Cela vous surprend peut-être mais tout dépend de la valeur de A à l'entrée de la boucle. Comme le carré de 15 vaut 225, il apparaît logique que la boucle s'arrêtera dès que A vaudra 200. Mais si A valait 210 par exemple à l'entrée ? Pas de problème, la boucle s'arrêtera aussitôt que B vaudra 15, c'est-à-dire quand A vaudra 225. Mais il suffit que vous initialisiez A à 230 ou même 1.10 pour déclencher une catastrophe, la boucle ne s'arrêtera jamais ! Prenez donc dès maintenant des habitudes saines qui vous feront échapper à ce genre de problèmes. Voici comment la sécurité de la boucle peut être améliorée :

```
REPEAT
  INC A
  B=SQR(A)
  Print "Racine de ";A;" = ";B
UNTIL B>=15 OR A=200
```

Nous n'avons pas mis A>=200 dans la condition car le test B>=15 aurait été inutile (encore un piège !) et la boucle s'arrêtera quoi qu'il arrive. Les erreurs les plus dangereuses sont celles qui ne sont pas facilement visualisables. Si par exemple, on avait démarré la boucle avec un A négatif, l'interpréteur se serait chargé lui-même de la détection de l'erreur. La propriété essentielle de cette forme de boucle est que l'exécution de la série d'instructions qui y est contenue est assurée une fois au moins, puisque le test de condition ne s'y opère qu'en fin.

Il en va autrement avec la boucle WHILE...WEND (Tant que <condition> Faire ... Fin Faire). A l'inverse de la précédente, elle ne sera pas exécutée si la condition n'est pas satisfaite à l'entrée de la boucle :

```
A=11
WHILE A<10
  INC A
  PRINT SQR(A)
WEND
```

Le bloc d'instruction n'est pas exécuté. L'énoncé d'initialisation WHILE détermine que le traitement est effectué dès qu'est satisfaite la condition A<10. En notre exemple, A étant supérieur à 10, le programme suit son cours à la première instruction succédant à WEND.

Au cas où le traitement de la boucle est opéré, la condition formulée dans la ligne d'initialisation est testée lors de chaque exécution de la boucle. L'insertion de plusieurs conditions liées logiquement entre elles est ici aussi possible.

A l'inverse des précédentes, la structure de boucle FOR...NEXT dont l'utilité est indéniable, ne suspend aucunement l'exécution des instructions qui y sont internes, à de pareilles conditions. L'exécution de la boucle est opérée un nombre de fois déterminé a priori.

```
FOR A%=1 TO 225
  B=SQR(A%)
  PRINT "Racine de ";A%;" = ";B
NEXT A%
```

Une variable numérique quelconque, en notre exemple A%, est indiquée dans la ligne d'initialisation FOR. Elle est incrémentée d'une valeur de 1 lors de chaque exécution de la boucle jusqu'à ce que la valeur finale (255) soit atteinte. L'itération de la boucle est assurée par l'adjonction du nom de la variable à l'instruction de reprise NEXT. Cette boucle

connaît d'autres variantes qui sont présentées dans la description de ladite instruction quelques chapitres plus loin. Si vous utilisez une version supérieure à la 3.01, cette boucle FOR s'exécutera comme un WHILE, alors que pour les versions précédentes, elle fonctionnait comme un REPEAT. Pratiquement, cela signifie que si vous aviez un programme du genre :

```
FOR A%=debut% TO fin%
    B=SQR(A%)
    PRINT "Racine de ";A%;" = ";B
NEXT A%
```

Avec debut%=3 et fin%=2, la boucle s'exécutera une seule fois en V3.00/V3.01 et pas du tout en V3.02/V3.03. Les cas où cela risque de vous gêner sont rares mais il fallait que vous le sachiez.

La boucle DO LOOP (Faire ... Fin Faire) peut ne contenir aucune condition mais peut aussi en avoir sous la forme DO WHILE ... LOOP, DO ... LOOP UNTIL et même DO WHILE ... LOOP UNTIL ! Cependant, on leur préfèrera WHILE WEND et REPEAT UNTIL pour des raisons de clarté. Quant au DO WHILE LOOP UNTIL, il est plus une preuve de la souplesse de ce Basic que d'une réelle utilité.

```
  DO
  INC A%
  B=SQR(A%)
  PRINT "Racine de ";A;" = ";B
LOOP
```

L'abandon d'une boucle DO LOOP sans condition est possible uniquement par l'emploi de la combinaison de touches d'interruption <Control/Shift/Alternate>, ou suite à l'insertion préalable d'une condition spécifique d'interruption. Cette condition d'interruption se nomme EXIT IF ; pour de plus amples informations, veuillez consulter la description de ladite commande.

Il est possible d'imbriquer des boucles les unes dans les autres.

```
WHILE I%<10
  INC I%
  FOR J=1 TO 10
    REPEAT
      INC K
      PRINT "I% = ";I%;"J = ";J;"K = ";K
    UNTIL K>I%*10
  NEXT J
WEND
```

Lors d'imbrications de cette sorte, il convient d'être attentif à l'ordre d'insertion des instructions de fin de boucle (NEXT / UNTIL / LOOP / WEND). Il est inverse de celui observé lors de l'insertion des instructions d'initialisation : (FOR/REPEAT/DO/WHILE).

```
Erroné: REPEAT                Juste: REPEAT
          WHILE A<10                    WHILE A<10
          UNTIL A=10                    WEND
          WEND                          UNTIL A=10

        FOR I=1 TO 10               FOR I=1 TO 10
          FOR J=1 TO 10               FOR J=1 TO 10
          NEXT I                      NEXT J
          NEXT J                      NEXT I
```

Suite à l'inobservance de l'ordre logique, l'interpréteur vous en avertit lors du lancement du programme à l'aide d'un message d'erreur. Mais on vous a dit que le GFA Basic possédait un éditeur orienté langage oui ou non ? Ben oui mais ce n'est pas une erreur de syntaxe qu'il reconnaît lorsqu'on inverse deux fins de boucles ... Allons donc ! Et la fonction Test du menu alors, c'est pour la décoration ? Entrez par exemple le premier exemple faux et placez le curseur sur la ligne du REPEAT. En cliquant sur Test (ou bien en appuyant sur F10), l'interpréteur va contrôler la validité de ces imbrications et vous affichera le message d'erreur : Do sans Loop. Un message du style "Ca ne tourne pas rond" aurait été plus imagé mais moins précis ...

Ceux qui connaissent d'autres interpréteurs BASIC et qui ont l'habitude de construire des boucles à l'aide de l'instruction GOTO devraient se familiariser le plus vite possible à l'emploi des présentes structures de boucles. L'avantage est que les instructions GOTO ne sont pas reconnues en tant qu'éléments de structure dans le GFA BASIC. Autrement dit, elles ne sont pas reconnaissables d'emblée lors de la lecture d'un listing de programme, alors que les boucles GFA sont affectées distinctement d'un retrait de ligne, d'une indentation.

En BASIC standard :

```
10 A=A+1:PRINT A;:If A<20 Then Goto 10
```

En GFA Basic :

```
Label:                  ! En ce cas, on utiliserait évidemment
INC A                   ! une boucle FOR...NEXT.
PRINT A;                ! Cet exemple ne sert qu'à
```

```
IF A<20                 ! illustrer les différences
  GOTO Label            ! de structures de boucle.
ENDIF
```

Ou mieux encore :

```
WHILE A<20        ou    DO                ou      REPEAT
  INC A                   INC A                       INC A
  PRINT A;                PRINT A;                    PRINT A;
WEND                      EXIT IF A=>20               UNTIL
                        LOOP
```

Notez à ce sujet que Frank Ostrowski n'avait pas inséré l'instruction GOTO dans les premières versions du GFABasic. Cela avait créé un problème dans le sens où beaucoup d'utilisateurs s'étaient plaints qu'ils ne pouvaient pas transcrire leurs programmes en BASIC standard facilement. L'auteur a transigé en implémentant le GOTO et a inséré un compteur de lignes pour ceux qui voulaient des numéros de lignes dans les programmes GFABasic.

4.15 Les opérations de comparaison

Lors des descriptions précédentes des structures de boucles, divers opérateurs de comparaison ont déjà été utilisés. Ils sont représentés par:

```
le signe d'égalité                                  =
le signe d'égalité relative, égalité sur 28 bits    ==
le signe inférieur                                  <
le signe supérieur                                  >
le signe de non-égalité                             <> ou ><
le signe inférieur ou égal                          <= ou =<
le signe supérieur ou égal                          >= ou =>
```

Ces opérateurs sont utilisables à des fins de comparaison de deux valeurs ou de deux expressions de texte.

```
PRINT "ABC">"BCD"
```

ou

```
A$="un"
B$="deux"
PRINT A$<>B$
```

ou

```
PRINT 123=234
```

ou

```
A%=1736
B=6511423.241
PRINT A%<B
```

Ces exemples apparaissent à première vue quelque peu étranges. Néanmoins leur signification n'est pas du tout douteuse, dès que l'on sait que le résultat d'une telle comparaison s'interprète en valeur de vérité : O ou -1 (FALSE ou TRUE). Les illustrations 1 et 3 de ce fait amènent un résultat nul, la chaîne ABC n'étant pas supérieure à celle BCD et la valeur 234 n'étant pas égale à 123. Les exemples 2 et 4 délivrent à l'inverse un -1, "vrai" puisque A$ et B$ sont effectivement inégaux (<>) et "A%" est inférieur à "B".

L'utilisation de variables booléennes est ici parfaitement imaginable. L'affectation de valeurs de vérité à des variables est souvent intéressante, notamment en ce qui concerne la lisibilité du programme. Votre programme fera 10 octets de plus et tournera 1/1000 de seconde moins vite mais vous pourrez parfois éviter une semaine complète de débogage (surtout quand on sait que le temps de développement est souvent primordial dans la réalisation d'un programme) et si vous laissez de côté le programme pendant longtemps, vous vous y retrouverez plus facilement. Par exemple, comparez :

```
IF (x% > 160) and (x% < 180)
   INC z%
ENDIF
avec
taillemoyenne! = (taillecm% > 160) AND (taillecm% < 180)
IF taillemoyenne!
   INC nbmoyens%
ENDIF
```

L'avantage de la deuxième méthode vous permet de savoir ce que fait le programme. De plus, vous faites l'économie de commentaires qui auraient été indispensables avec la première manière.

Vous pourriez vous étonner des comparaisons d'expressions de texte que nous avons vu un peu plus haut. D'une certaine manière, ces dernières vont de soi. Lors de la comparaison de deux expressions de texte, l'interpréteur compare les caractères des deux chaînes selon leur rang respectif et regarde laquelle contient une lettre dont le code ASCII est

supérieur. Si les premières lettres de chaque chaîne étaient identiques, il passera aux suivantes et ainsi de suite. Si les deux chaînes sont de longueurs inégales et que leurs lettres sont identiques, la chaîne supérieure sera la plus longue ("AGAGA" sera supérieur à "AGA" mais "GAGA" sera supérieur à "AGAGA").

Toutefois, il convient d'observer que lors de la comparaison de chaînes de caractères, les minuscules sont supérieures aux majuscules car de code ASCII plus élevé : les codes ASCII de A à Z s'échelonnent de 65 à 90, ceux de a à z de 97 à 122. Une comparaison conséquente selon l'ordre alphabétique nécessite le passage en majuscules des chaînes comparées, ou leur passage en minuscules (Cf. UPPER$).

```
PRINT "ABC"<"ABCD"="abc">"abcd"
```

Cet exemple fictif n'est guère compréhensible d'emblée. D'abord les deux premières expressions sont vérifiées. ABC étant inférieur à ABCD, il en résulte la valeur de vérité -1. Comme la fausseté de la seconde comparaison : abc>abcd est évidente, le chiffre 0 en résulte. L'égalité de ces deux valeurs de vérité est ensuite vérifiée. Comme leur inégalité est patente, la valeur 0 en résulte.

Lorsque les expressions contiennent d'autres signes que les caractères alphabétiques, par exemple des chiffres, des signes spéciaux, des espaces, leur comparaison s'effectue selon le même schéma (Cf. MAX, MIN). L'étude d'une table de codes ASCII valant mieux qu'un long discours, reportez-vous à la fin de ce livre pour plus d'informations.

4.16 Les règles de priorité

Les règles d'ordre des diverses opérations arithmétiques propres à l'informatique sont analogues à celles observées en mathématiques.

X=SQR(12^3+(36-22^1.2)-(-4/3))*SIN(13)

Les formules de ce genre se résolvent d'après l'ordre suivant :

1. Fonctions : SQR, TAN, ATN, etc., tout comme DEFFN.

2. Parenthèses : ()

3. Elévation à la puissance : ^

4. Moins Unaire (Négation) : -

5. Multiplication / Division : *, /

6. Modulo / Division entière : MOD, DIV ou \

7. Addition / Soustraction : +,-

Quant aux opérateurs logiques et ceux de comparaison, dont l'insertion dans ces formes opératoires est possible, elles respectent l'ordre suivant:

8. Opérateurs de comparaison : =, ==, <>, >, <, =>, <=

9. Opérateurs logiques : AND, OR, NOT, XOR, IMP, EQV

```
PRINT (12^3+(36-22^1.2)-(-4/3))*3>14^3
```

ou

```
PRINT (12^3+(36-22^1.2)-(-4/3))*3 AND 14^3
```

Lors de l'emploi d'opérateurs seuls de même priorité au sein d'une unique expression, les opérations s'effectuent de gauche à droite. Les opérations prioritaires doivent être mises entre parenthèses. A l'intérieur des parenthèses, les opérations sont traitées selon leur ordre habituel.

4.17 Exercices de manipulation

Ce chapitre vous a présenté certaines instructions BASIC. Nous supposons que vous connaissez maintenant les instructions élémentaires.

N'étant pas nés de la dernière pluie, nous savons parfaitement que la plupart d'entre vous ne nous ont pas attendu pour taper leur premier programme GFA Basic. Mais nous n'allons pas en tenir compte. Après avoir surmonté les difficiles étapes que sont l'insertion de la disquette contenant le GFABasic, la sélection de l'icône et le lancement de

l'interpréteur, vous voici en face du splendide écran plein de fonctions dans ses menus. Comme toute forme de vie Amigoïde qui se respecte, vous avez passé un certain temps à regarder à quoi servait tout ça et même à descendre l'écran GFABasic pour vérifier que le Workbench était toujours là. C'est ici que la tradition nous indique que le premier programme doit être l'affichage (par n'importe quel moyen) de la chaîne "Hello, World !" (le point d'exclamation est facultatif mais de bon goût, car il exprime une satisfaction certaine). Entrons donc dans l'éditeur :

```
p "Hello, World !"
```

en ponctuant la saisie de la pression de la touche <Return>.

La manifestation visible d'une des caractéristiques propres au GFA BASIC s'ensuit instantanément. La plupart des instructions GFA possèdent une abréviation qui sera reconnue par l'interpréteur (quand on vous dit qu'il est orienté langage cet éditeur !). L'interpréteur vous montre qu'il a compris en affichant :

```
PRINT "Hello World"
```

Cette particularité économise bien des peines, en particulier lors de saisies d'une longueur certaine, puisque la simple insertion de "gr" supplée celle de GRAPHMODE.

Vérifiez-le à l'aide de la simple saisie de "gr 1" conclue à l'aide d'une pression de la touche <Return>. Instantanément, GRAPHMODE 1 apparaît à l'écran. Puis à l'aide des touches de direction, positionnez-vous dans la ligne de GRAPHMODE 1, puis pressez simultanément les touches <Control>+<Y>, aussitôt la seconde ligne disparaît.

Lors des saisies de texte, seule l'insertion du guillemet ouvrant est requise, l'interpréteur reconnaissant de lui-même la fin du texte.

```
PRINT "Texte
```

ou

```
A$="Texte
```

L'éditeur GFA BASIC est un programme d'une puissance peu commune, dont les multiples possibilités ne se dévoilent que suite à une utilisation soutenue et avertie.

Sa qualité intrinsèque réside dans d'innombrables petits détails, qui assurent un confort d'utilisation indéniable. Certes cela ne va pas sans contrepartie, puisque lors de vos propres développements de programmes, il vous faudra veiller à toutes ses petites différences, mais toute qualité n'est-elle pas à ce prix ? Nous avons d'ailleurs eu des discussions à ce sujet avec certains de ses détracteurs où nous sommes tombés d'accord sur le fait que l'absence de Couper/Copier/Coller était le seul défaut majeur. GFA réfléchit d'ailleurs au problème ...

Après cette brève parenthèse, il importe maintenant de lancer notre petit programme. Plusieurs possibilités nous sont offertes à cette fin. Elles procèdent soit par :

1. Le pointage de l'option RUN disposée dans la barre de menus de l'éditeur suivi de la pression du bouton gauche de la souris,

2. les pressions simultanées des touches <Shift> et <F10>,

3. la commutation du mode direct suivie de l'insertion de la commande "ru", l'abréviation de "RUN" validée à l'aide de la pression de la touche <Return>,

4. sélection de l'entrée Run dans le menu déroulant,

5. pression des touches Amiga R,

6. sauver le programme avec une icône, quitter l'interpréteur et lancer le programme du Workbench en cliquant sur l'icône du programme,

7. sauver le programme sans icône et lancer le programme du CLI avec Run GFABASIC:GFABASIC <nom du programme>.

Nous voyons donc là 7 grandes catégories d'utilisateurs du GFABasic sans compter les variations que certains originaux pourraient apporter.

Chacune de ces procédures amène le renouvellement de l'écran, puis l'édition de votre salutation amicale en haut à gauche de l'écran, suivie de celle d'une dite boîte d'alerte contenant le message "Fin du programme". Dès que la touche <Return> est pressée ou suite au cliquement de l'option "Return", l'interpréteur assure le retour dans la

fenêtre de l'éditeur. En ce faisant, la première pierre de votre future carrière de programmeur a été posée. Tout le reste n'étant plus que de la dentelle (et un peu d'exercice, avouons-le).

Pour le moment, il est utile de se familiariser progressivement avec les multiples possibilités présentes. En guise d'essai, cliquez avec la souris l'option "Direct" de la barre de menus de l'éditeur. Il apparaît dans le bas de la fenêtre le prompt "OK >". Cette invite est celle du mode d'exécution directe des instructions ou mode commande. En mode direct, il vous est impossible d'écrire des programmes, seules des lignes de commande isolées (contenant au maximum 255 signes) peuvent être lancées. Suite à la validation de la saisie de la commande par la pression de la touche <Return>, la commande ainsi envoyée est immédiatement exécutée. Insérez, s'il vous plaît :

```
OK >p "Hello World
```

Dans la fenêtre d'édition apparaît :

```
Hello World
```

Vous observez que l'instruction a été exécutée aussitôt après une pression sur la touche <Return>. Le mode direct vous permet d'éprouver en toute tranquillité l'efficacité des instructions disponibles. Certes quelques instructions ne peuvent pas être testées en mode direct, par exemple celles dont la syntaxe nécessite plusieurs lignes. Néanmoins, le mode direct est particulièrement indiqué pour l'exercice des ordres disponibles. A titre de nouvel essai, veuillez insérer :

```
OK >a 1,"Alerte Rouge|Télétransportation activée",1,"HyperEspace|Désactivation",a
```

Puis frappez la touche <Return>. Rien de plus simple! Dès que vos exercices en mode direct vous semblent suffisants, il vous est possible de commuter l'éditeur à l'aide de la touche <Esc> ou de la combinaison de touches d'interruption <Control/Shift/Alternate>.

Ce retour dans l'éditeur nous permettra d'écrire notre premier programme digne de ce nom. La chose essentielle qu'assure un programme est la gestion des entrées/sorties. Certes d'autres instructions importent tout autant que celles des entrées et sorties (en langue anglaise Input/Output ou l'abréviation I/O) mais l'absence de ces dernières interdit tout développement. Nous débuterons de ce fait avec elles.

```
CLS
INPUT "Quel est votre nom : ";Nom$
INPUT "Quel est votre âge : ";AGE%
PRINT "Pressez, s'il vous plaît, la touche <Return>"
REPEAT   UNTIL INKEY$=CHR$(13)
PRINT "Vous vous appelez ";NAME$;"."
PRINT "Vous êtes âgé de ";AGE%;" ans."
```

Lors de l'exécution de la boucle, vos nom et âge sont demandés. D'abord l'écran est effacé par la commande CLS et puis le curseur est positionné dans le coin gauche du haut de l'écran. Les insertions du nom et de l'âge, validées chacune par la pression de <Return>, sont sauvegardées dans des variables : le nom : une chaîne de texte dans une variable de chaîne, et l'âge: une valeur numérique dans une variable de type entière. Ensuite la pression de la touche <Return> est une nouvelle fois requise par l'instruction PRINT. La condition introduite par UNTIL détermine la vérification de la pression ou non d'une touche définie. La pression de la touche <Return> dont le code ASCII est 13, amène l'abandon de la boucle et l'édition à l'écran de vos entrées munies de commentaires. En cas de pression d'une touche différente, rien ne se passe.

Les deux PRINT montrent clairement que les éditions que cet ordre assure peuvent varier. Le délimiteur ";" permet de rester sur la même ligne. L'ordre d'insertion des expressions importe peu. Une ligne de commande PRINT peut s'énoncer ainsi :

```
A$="Texte"
PRINT LEN(A$)''''A$;"en GFA";CHR$(66);"ASIC", CRSCOL'INKEY$;
```

Une telle ligne connaît des insertions de type divers. A la ligne précédente, la variable A$ est initialisée avec le texte "Texte". LEN(A$) renvoie la longueur de la chaîne contenue dans la variable A$, autrement dit la longueur de "Texte". Avec A$, le contenu de la variable est retourné. Puis suit l'expression " en GFA ". Ensuite est placé un code ASCII isolé (code ASCII 66 = B). Puis est insérée une nouvelle expression BASIC suivie de l'indication de la position actuelle du curseur. Enfin est communiqué le caractère frappé au clavier. Lors de l'exécution de la ligne d'instruction, les fonctions intégrées à cette dernière sont traitées et dès que leur traitement est abouti, leur résultat est édité. De ce fait l'instruction INKEY$ n'est seulement traitée que suite à l'édition du résultat des fonctions qui la précédent. Toutes les fonctions (opérations assurant la délivrance de résultats) sont insérables dans une ligne d'instruction PRINT.

En outre le formatage des données éditées peut être déterminé a priori à l'aide de symboles propres. L'insertion d'une virgule entre les expressions instruit lors de l'édition le retrait de l'expression consécutive de cinq tabulations (Cf. PRINT). A l'inverse, celle d'une apostrophe n'amène que l'insertion d'un espace à la position indiquée. La signification d'un point-virgule en tant que délimiteur a déjà été présentée. Dans notre exemple, ce dernier a été placé en fin de texte afin que la prochaine édition PRINT (ou OUT ou WRITE) ou INPUT (ou INP, INPUT$) soit directement accolée à l'édition précédente.

A l'aide de :

```
PRINT AT(10,10);
INPUT "Insertion: ",A$
```

il vous est possible de déterminer la position de l'insertion à effectuer. Celle-ci réalisée, le curseur saute généralement au début de la ligne suivante. Le dernier exemple expose deux nouvelles variantes d'édition. Dans la première, l'énoncé AT(X,Y) instruit le placement du curseur à un lieu déterminé de l'écran (Cf. PRINT). Dans la seconde, l'insertion de la virgule après le commentaire assure lors de l'exécution, que la saisie débute directement derrière le commentaire, en l'absence de toute insertion d'espace ou de point d'interrogation. Veuillez comparer les lignes de commande suivantes :

```
INPUT "Insertion : ";A$
INPUT "Insertion : ",A$
```

Parmi les descriptions d'instructions sont disposées bien d'autres instructions dont les innombrables applications possibles sont susceptibles de répondre à tous les souhaits imaginables.

Essayons un peu de graphisme. Aimeriez-vous dessiner un cercle ou un carré, ou plutôt une ligne à main levée? Tout cela est d'une simplicité enfantine ! Tous les langages BASIC avancés offrent, cela va de soi, d'innombrables instructions graphiques. Le GFABASIC en offre deux fois plus. Néanmoins, cela n'est guère aussi évident qu'il n'y paraît aujourd'hui quand on pense qu'il fallait, il y a quelques années seulement, écrire tous les algorithmes à la main et quand on sait que rares sont les gens qui à l'improviste sont à même de détailler de la construction d'un cercle ou d'une ellipse.

Cela est bien plus simple aujourd'hui. La liste des instructions fournie en appendice consigne beaucoup d'instructions graphiques, dont les applications sont d'une évidence et d'une simplicité indéniables. Notre écran est divisé en Hires (High resolution ou en français Haute résolution), en lignes horizontales de 640 pixels ou points et en verticales de 256 pixels. Le point de coordonnées 0/0 est positionné usuellement (Cf. OPENW) dans le coin gauche extrême du haut de l'écran. Grâce à la détermination du point d'origine, chaque point de l'écran est définissable. Il suffit uniquement de décider en quelles coordonnées ou en quelle position de l'écran, vous souhaitez voir afficher un objet graphique et puis transmettre l'instruction contenant les coordonnées correspondantes. Ces choses faites, l'objet désiré apparaît instantanément à l'écran. La ligne suivante effectue le traçage d'une ellipse dont le centre est à peu près le milieu de l'écran.

```
ELLIPSE 310,120,300,110
```

Les deux premiers paramètres déterminent la position, le troisième le rayon horizontal et le quatrième le rayon vertical. Ces indications de coordonnées réfèrent, comme nous le disions, à la résolution Hires. Toutefois, il existe plusieurs modes graphiques différents pour l'Amiga. Le second est dit Lowres (Low Resolution ou en langue française : basse résolution), de 320 points horizontaux et de 256 points verticaux (les 512 points correspondent au mode entrelacé). A l'aide de ces informations et des descriptions des instructions, vous ne devriez guère rencontrer de problèmes lors du développement de vos propres applications graphiques.

L'utilisation des flags est illustrée ici. L'énoncé flag! représente une variable booléenne, dont la tâche est de noter dans la seconde boucle REPEAT si la pression du bouton gauche de la souris a été opérée. Si tel est le cas, un rectangle est dessiné par deux fois à la même position en mode XOR. Cela a pour effet que le contour du rectangle n'apparaît brièvement qu'à chaque exécution de la boucle. Dès que le bouton droit de la souris est relâché, le traçage du rectangle est renouvelé en mode REPLACE (Cf. GRAPHMODE).Veillez lors de la pression simultanée des boutons de la souris, pendant l'exécution du programme, à presser d'abord le bouton droit et puis le bouton gauche car sinon en résulte le dessin d'un point en mode ligne, avant que le procès ne débute.

Veuillez noter que la boucle FOR...NEXT est initialisée à l'aide d'une variable réelle. Avec une variable entière, le programme tracerait une ligne droite et répèterait l'opération indéfiniment, puisque la variable numérique serait toujours mise à zéro.

Notre introduction au langage de programmation GFA BASIC et l'initiation aux principes et aux fonctions élémentaires de la programmation nous ont menées fort loin. Il ne nous fut guère facile de trouver les mots justes pour rendre ces thèmes abordables et compréhensibles aux débutants. Du moins nous nous y sommes appliqués et nous espérons que ces diverses connaissances vous seront d'une aide certaine pour vos prochaines applications.

De multiples modèles de programmation sont disséminés tout au long de ce livre selon une progression étudiée. Ils vous permettront d'approfondir vos récentes connaissances. Un dernier conseil avant de conclure : délaissez lors de la période d'apprentissage tout programme étranger. Veillez à ce que la complexité des choses entreprises soit progressive, de manière à ce que vous ayez des repères solides où vous raccrocher.

Chapitre 5

Les instructions d'Entrées/Sorties

5.1 L'entrée de données

FORM INPUT (F) — Saisit une chaîne formatée

FORM INPUT Nombre, VAR$

FORM INPUT est une combinaison des instructions INPUT et INPUT$. Elle permet de définir la longueur maximale du texte à insérer. La longueur maximale de la chaîne est de 255 caractères. Dès que l'utilisateur dépasse la longueur du texte définie au préalable, le curseur ne bouge plus de sa dernière position. L'insertion n'est close qu'avec la pression de la touche <Return>. Les modifications du texte obéissent aux mêmes procédures que celles décrites sous INPUT. Il en est de même pour l'insertion de signes spéciaux. Toutefois l'adjonction d'une liste de variables ou d'une constante de texte prédéterminée est ici impossible. Le curseur est positionné automatiquement sur la première position de la ligne d'insertion.

Exemple :

```
FORM INPUT 32, Textevar$
```

Du reste, le positionnement de la ligne d'insertion obéit aux mêmes règles que celles observées avec INPUT et LINE INPUT, en amenant le curseur à la place où le premier caractère saisi doit apparaître, à l'aide de :

```
PRINT AT(XP,YP);
```

Grâce au point-virgule concluant la ligne d'instruction PRINT, le retour chariot (CR = Carriage Return) et le saut de ligne (LF = Line Feed) sont oblitérés. Le prochain signe à éditer est placé juste derrière le dernier signe édité.

FORM INPUT [F AS] Autorise une saisie complémentaire

FORM INPUT Nombre AS Var$

Les règles précédemment observées valent pour cette dernière, à la seule différence qu'elle édite le contenu d'une variable de chaîne et autorise l'édition ultérieure de la chaîne contenue (ou d'une partie de celle-ci). "Nombre" représente le nombre de caractères à éditer à partir du début de la chaîne. Suite à la saisie complémentaire validée par <Return>, la nouvelle chaîne obtenue est mémorisée dans la variable, remplaçant de ce fait le contenu précédent de la variable. "Nombre" étant inférieur à la longueur de la variable spécifiée, il convient de libérer de l'espace pour la saisie à l'aide de la touche <Delete>. Lors de la notification d'une variable vide, cette instruction remplit la même fonction que FORM INPUT.

Exemple (en liaison avec MID$()= et MID$):

```
a$="Constante"                      ! une chaîne quelconque
b$=MID$(a$,8,3)                     ! sépare trois signes
PRINT AT(10,10);"=> ";a$;" => ";    ! détermine le lieu de
                                    ! saisie
FORM INPUT 7 AS b$                  ! 8 signes pour la saisie
MID$(a$,8,LEN(b$))=b$               ! entre les données
PRINT AT(10,10);"=> ";a$;SPC(15)    ! édite la chaîne
```

INKEY$ Acquiert, détecte le caractère frappé

Caractères$=INKEY$ =>(affectation de caractère) [IF/WHILE/UNTIL]LEN(INKEY$)=>(condition:INKEY$ >'') [IF/WHILE/UNTIL]INKEY$='Z' =>(condition: pression de 'Z')

INKEY$ acquiert le caractère qui vient d'être frappé au clavier. Toutes les touches disponibles sur le clavier mais aussi de tous celles spécifiques, par exemple F1 à F10, les touches de direction, peuvent être récupérées. De plus la détection de presque toutes les combinaisons de <Control>, <Alternate> ou <Shift> avec d'autres touches est ainsi enregistrable. Les touches de fonction en combinaison avec la touche Shift connaissent de ce fait une double assignation. La fonction commute en un autre mode interne dès qu'une combinaison de touches correspondant à une séquence CSI est appuyée. En général, une chaîne

d'un caractère unique, contenant le code ASCII correspondant au caractère frappé, est renvoyée. Cette dernière y est introduite par l'énoncé CHR$(155) suivi d'un à trois autres signes. Aucune touche n'étant pressée lors d'un test INKEY$, une chaîne vide ("") est retournée.

Exemple :

```
PRINT "Testez une touche ou une combinaison de touches"
PRINT " (<Esc> pour arrêter le test)"
DO                              ! initialise boucle principale
  REPEAT                        ! boucle des entrées
    key$=INKEY$                 ! test
  UNTIL key$>""                 ! jusqu'à qu'une touche soit
                                ! frappée
  PRINT AT(10,10);SPACE$(30)    ! ligne effacée
  PRINT AT(10,10);              ! curseur placé au début
  IF LEN(key$)=1                ! un octet?
    PRINT "ASCII: ";
    PRINT ASC(key$)
  ELSE                          ! plusieurs octets ...
    PRINT "Séquence CSI : ";
    FOR i%=1 TO LEN(key$)
      PRINT ASC(MID$(key$,i%));" ";
    NEXT i%
  ENDIF
  EXIT IF key$=CHR$(27)         ! interruption par <Esc>
LOOP                            ! fin de boucle principale
```

Toutefois, un problème apparaît. Le tampon de mémoire du clavier a la fâcheuse habitude de conserver les touches venant d'être frappées, c'est-à- dire toutes. Cela a pour conséquence lors du contrôle de la frappe d'un unique caractère, l'édition de tous les caractères contenus dans le tampon de mémoire du clavier, éventuellement mémorisés suite à la pression continue d'une touche. On peut remédier à cela par l'insertion après le test INKEY$, d'une boucle vidant le tampon du clavier.

```
REPEAT
UNTIL INKEY$=""
```

La détection de la frappe d'une touche à l'aide d'INKEY$ (à titre d'illustration : un menu avec diverses options dont la sélection est assurée à l'aide de touches) s'effectue dans les autres BASIC selon le modèle suivant :

```
100 A$=INKEY$: IF A$="" Then Goto 100
```

Cette construction est remplacée en GFA Basic

```
REPEAT
UNTIL LEN(INKEY$)
```

ou

```
REPEAT
UNTIL INKEY$<>""
```

Ces dernières remplissent les mêmes tâches. Elles attendent la frappe d'une touche quelconque du clavier. Le test de la frappe d'une touche déterminée s'ordonne dans d'autres langages BASIC selon la possibilité fort usitée :

```
100 A$=INKEY$:IF A$<>"b" THEN GOTO 100
```

La même chose s'énonce en GFA Basic:

```
REPEAT
UNTIL INKEY$="b"
```

Cette boucle contrôle la frappe d'une touche déterminée, dans notre cas, il s'agit de la touche b. La frappe d'une autre touche amène la réitération de la boucle. Autrement dit, le programme reste dans la boucle d'attente jusqu'à ce que la touche: "b" soit pressée.

INPUT {INP} — Saisit des données

INPUT ["Texte";] Var1 [,VAR2,...]
INPUT #Canal, Var1 [,VAR2,...]

Cette instruction est l'une des plus utilisées pour la saisie de valeurs ou de chaînes dans un programme. L'adjonction d'un texte devant l'endroit où va débuter la saisie est possible.

Après la virgule ou le point-virgule, est spécifiée la variable devant réceptionner la donnée saisie. Le point-virgule a pour effet de placer un espace et un point d'interrogation devant le curseur. La virgule positionne le curseur juste derrière l'instruction ou la constante de texte.

La saisie de plusieurs valeurs ou chaînes à l'aide de l'instruction INPUT, requiert l'adjonction, dans la ligne de commande, d'une liste de variables séparées l'une de l'autre par une virgule. La saisie ainsi ordonnée n'est achevée qu'après l'entrée de toutes les variables spécifiées. Tout type de variables peut y être spécifié. La programmation de la saisie simultanée de variables de texte et de valeurs avec une seule instruction INPUT est donc possible. Si lors de l'exécution de cette dernière, la donnée saisie ne correspond pas au type de la variable propre à la réceptionner (à titre d'illustration : l'entrée de texte dans une variable numérique), l'ordinateur émet un beep lumineux (un léger scintillement affecte l'écran) et vous oblige à recommencer la saisie.

Chaque donnée isolée lors de saisies successives avec une seule ligne INPUT peut être close par la touche <Return> ou être séparée de la suivante par un point virgule. Toutefois, la séparation effectuée à l'aide de la touche <Return> positionne le curseur sur le début de la ligne suivante.

L'emploi de la virgule lors de la saisie des données en l'absence de la commutation de la prochaine variable réceptrice, résulte de la mise entre guillemets du contenu saisi à affecter à la variable. Si vous voulez saisir une chaîne contenant une virgule, vous devez la placer entre guillemets.

Exemple :

```
PRINT AT(10,10)
INPUT "Nombre, Chaîne, Nombre : ",A%,B$,C%
PRINT A%,B$,C%
```

Le nombre maximal de caractères insérables dans une chaîne d'insertion est de 255. Lors de la saisie d'une chaîne, les corrections des données insérées s'opèrent à l'aide des touches suivantes :

```
<Backspace>          =supprime le signe à gauche du curseur,
<Delete>             =supprime le signe sous le curseur,
<Flèche à gauche>    =déplace le curseur d'un espace vers la
                      gauche,
<Flèche à droite>    =déplace le curseur d'un espace vers la
                     =droite, du moins s'il n'est pas positionné
                      en fin de ligne,
<Flèche en bas>      =déplace le curseur en début de ligne,
<Flèche en haut>     =déplace le curseur en fin de ligne.
```

L'utilisation de signes spéciaux lors de la saisie des données, s'effectue à l'aide des deux possibilités suivantes :

1. <Alternate> et une autre touche,

2. <Amiga> et une autre touche.

INPUT permet une saisie à partir de canaux d'entrée autres que le clavier. Cela peut être un fichier sur disquette ou le port série. Pour cela, il faut indiquer le numéro de canal utilisé précédé d'un #. (Cf. OPEN).

Exemple :

```
OPEN "I",#1,"FICHIER.DAT"
INPUT #1,A$,B$,C$
```

ou

```
OPEN "I",#2,"CON:1/1/200/200/Saisie"
INPUT #2,A$,B$,C$
```

Ces exemples requièrent la stricte correspondance entre les types des données de fichier lues et ceux des variables spécifiées.

INPUT$() — Saisie de chaînes de caractères

A$=INPUT$(Nombre)
A$=INPUT$(Nombre, #Canal)

Cette instruction permet la saisie de chaînes de caractères en mode caché. A la différence des autres procédures de saisie de données, la variable réceptrice ne suit pas l'instruction. L'énoncé A$=INPUT$(10) effectue l'affectation de la chaîne dans la variable A$. PRINT INPUT$(10) affiche immédiatement la chaîne saisie. Cet autre IF INPUT$(4)="xxxx" combine la saisie avec un test.

Le nombre entre parenthèses, adjoint à la fonction détermine la nombre de caractères pouvant être saisis. Dès que celui-ci est atteint, l'exécution du programme se poursuit sans attendre la pression de la touche <Return>. Le texte entré est invisible lors de la saisie. Les corrections de texte s'y effectuent de la même manière qu'avec INPUT.

Une forme syntaxique de spécification du canal lui est aussi acquise : (A$=INPUT$(10,#1)). L'utilité particulière de cette dernière est d'assurer la lecture du Nombre de données spécifié d'un fichier de disquette. L'affectation d'un fichier entier dans une chaîne (il est entendu que sa taille n'est pas plus élevée que la longueur maximale d'une chaîne qui est de 32767 octets), peut s'effectuer, à titre d'illustration, à l'aide du programme suivant (Cf. BGET# et LINE INPUT#) :

```
OPEN "I",#1,"FICHIER.DAT"
A$=INPUT$(MIN(32767,LOF(#1)),#1)
CLOSE #1
PRINT A$
```

Il est inutile de connaître d'avance la taille du fichier puisqu'elle vous sera communiquée par LOF(). On évite le dépassement de la longueur maximale de la chaîne résulte de la limitation du nombre de données à lire opérée par MIN.

LINE INPUT (LI) — Saisie d'une chaîne de caractères

LINE INPUT ["Texte";] Var$[VAR1$,...]
LINE INPUT #Canal, Var$[VAR1$,...]

Cette instruction est apparentée à INPUT. Elle se différencie de la précédente dans le fait qu'elle répond exclusivement à l'entrée de texte et que lorsqu'une liste de variables est spécifiée chacune d'elles doit être validée par la touche <Return>. Alors que dans INPUT, le séparateur qu'est la virgule initialise la prochaine variable, la virgule insérée dans la chaîne saisie est ici interprétée tel un élément du texte. De même qu'avec INPUT, il y est possible de placer une constante de texte qui figurera devant l'endroit de la saisie.

Exemple :

```
LINE INPUT "Veuillez entrer le texte : ",Aa$,Bb$,Cc$
```

L'utilisation d'une virgule ou d'un point-virgule entre le texte et la première variable permet de placer un espace suivi d'un point d'interrogation devant le curseur. Les possibilités d'édition sont identiques à celles d'INPUT.

La seconde syntaxe permet la lecture de données à partir du fichier ou de l'interface spécifié dans #Canal. Ceci explique l'introduction de l'intitulé de l'instruction par le mot LINE. Si aucun retour chariot suivi d'un saut de ligne (CR/LF) n'est disposé au sein du texte lu ou si plus de 254 caractères sont compris entre deux CR/LF, le texte est automatiquement coupé au 254ème signe et affecté tel quel à la variable correspondante. Des entrées de texte de plus de 256 caractères (254+CR+LF=256) sont impossibles.

Si le texte contient des codes ASCII, inférieurs à 32, autres que ceux des codes de fin de ligne (CR/LF), des erreurs sont possibles lors de l'affectation.

Note: Cette instruction n'est utile qu'avec des textes ne contenant que des caractères de code ASCII supérieurs à 31 et dont la séparation des lignes a été effectuée à l'aide de CR+LF.

5.2 Sortie des données

PRINT [? ou P] — **Edite les données**

PRINT [AT(Colonne,ligne)][,']"Texte"[[;,']Var[;,']Expr...;]
PRINT [#Canal,[;]]"Texte"[[;,']Var[;,']Expr...;]

La fonction PRINT permet la sortie de textes ou de variables à l'écran ou vers un fichier.

```
A%=1
A$="- Démo"
PRINT AT(1,12);
PRINT CHR$(27);"p";" PRINT"'A$,A%,"Touche: ";
```

```
PRINT "!";CHR$(27);"q";At(1,13);
PRINT STRING$(40,LEFT$(A$))
```

Les composantes de cette série d'instructions PRINT, revêtent les significations suivantes :

```
AT(XP,YP)
```

Cette option détermine la position du curseur à partir de laquelle s'effectue l'édition des données. La fenêtre d'édition se décompose en 77 colonnes et 28 lignes (XP=1-77, YP=1-28), pour mémoire : le point d'origine de l'écran est la position curseur HOME.

La même chose s'obtient à l'aide de LOCATE. L'énoncé : AT(XP,YP) est insérable, comme le montre notre exemple, à l'intérieur d'une ligne d'édition. Une même ligne de commande PRINT permet ainsi la définition de diverses positions d'écran.

; (le point-virgule)

Il sert à lier diverses expressions entre elles. L'expression suivant ce signe est directement concaténée à la précédente. S'il est placé en fin de ligne, la première expression de la prochaine instruction PRINT se rattache à celle éditée en dernier puisque, dans ce cas, le retour chariot et le saut de ligne sont simplement ignorés.

, (virgule)

Ce signe instruit lors de l'édition le retrait de l'expression qui le suit, de cinq arrêts de tabulation à partir de celle éditée en dernier. Les positions X de ces arrêts sont 1,17,33,49,65.

' (apostrophe)

L'apostrophe représente un espace. Lors de l'édition, un espace apparaît chaque fois qu'une position déterminée par l'apostrophe est atteinte.

L'intégration de fonctions de chaîne et numériques à nos exemples démontre leur souplesse d'utilisation dans des lignes de commandes PRINT. Consultez à cette fin, les descriptions des fonctions usitées, sous CHR$(), STRING(), LEFT$(), etc.

Les constantes de type texte (par exemple PRINT "Texte") doivent être introduites à l'aide d'un guillemet ouvrant, sinon elle est interprétée telle une variable sans contenu. Si rien ne suit la constante, l'interpréteur place de lui-même le deuxième guillemet.

Une instruction PRINT isolée en l'absence de l'adjonction d'autres expressions, commande l'insertion d'une ligne vide (CR/LF). PRINT# est une variante de PRINT. Le signe # est suivi du nombre spécifiant le numéro du canal à utiliser. Il est ainsi possible d'écrire des données dans un fichier ouvert, par exemple : PRINT#1;"BASIC".

PRINT USING {P USING} — Edite un texte formaté

PRINT USING "format", Expr [Var,...][;]
PRINT USING FORMAT$, Expr [Var,...][;]
PRINT #Canal, USING "format", Expr [Var,...][;]

Cette instruction permet de déterminer une édition formatée des valeurs ou des chaînes de caractères. Elle est suivie en premier lieu d'une chaîne ou d'une variable de chaîne, où est notifié le format désiré. Après une virgule, sont adjointes les expressions ou les variables devant correspondre au format choisi. L'adjonction d'expressions multiples demande leur séparation par une virgule. Les signes de formatages disponibles sont les suivants :

\# représente une position de caractère numérique

```
PRINT USING "######",Int(31421/3)
```

Sortie : `10473`

. représente une position de point décimal

```
PRINT USING "######.####",31421/4
```

Sortie : `7855.2500`

\+ instruit l'édition du signe positif

```
PRINT USING "+######.####",31421/4
```

Sortie : `7855.2500`

- instruit l'édition du signe négatif

```
PRINT USING "-######.####",31421/-4
```

Sortie : - 7855.2500

* ordonne le remplissage des espaces non occupés par la valeur éditée, sinon identique au cardinal: #

```
PRINT USING "*######.####",31421/1.4
```

Sortie : **22443.5714

Lors de son emploi derrière le point décimal, il en résulte l'édition d'autant d'astérisques que celles spécifiées, la valeur communiquée étant arrondie au rang désiré.

```
PRINT USING "*######.#****",31421/1.4
```

Sortie : **22443.6****

$ renvoie un dollar

```
PRINT USING "$######.####",31421/1.4
```

Sortie : $22443.57

, insère une virgule pour la séparation des milliers

```
PRINT USING "##,###,###.####",3142*2781.71
```

Sortie : 8,740,132.820

^^^^instruit l'édition sous forme exponentielle. Les # placés en premier représentent les rangs des chiffres significatifs et ^ les rangs de l'exposant (E+xxxx). Les positions superflues de chiffres significatifs sont remplies de 0 et l'exposant est ajusté.

```
PRINT USING "#.#######^^^^",13711*64
```

Sortie : 8.77504000E+05

! Commande l'édition du premier caractère d'une chaîne :

```
PRINT USING "!sing !n !FA","Uhu","igitt","Gaga"
```

Sortie : Using in GFA

& permet l'édition de la chaîne complète

```
PRINT USING "&house","Mul"
```

Sortie : `Mulhouse`

\ .. instruit l'édition du nombre de caractères délimité par \ ..\ (antislashs:\ inclus)

```
PRINT USING "\..\sbourg","Strass"
```

Sortie : `Strasbourg`

_ amène l'interprétation du signe qui le suit, non pas comme un caractère de formatage mais comme un caractère ASCII.

```
PRINT USING "Channel _###_\ &",44,"XYZ"
```

Sortie : `Channel #44 \ XYZ`

Les caractères de formatage et la séquence de chaînes, de valeurs ou d'expressions sont insérables dans un ordre quelconque, du moins tant que les types de paramètres correspondent aux formats spécifiés. Si cette correspondance n'est pas observée, l'interpréteur rencontrant par exemple des formats illogiques, un point d'interrogation apparaît à la position litigieuse. Le format prévu étant dépassé lors de l'édition d'un nombre, un signe de pourcentage précède la valeur éditée.

En outre le choix entre le point ou la virgule à titre de séparateur décimal (Cf. MODE) vous est laissé. L'insertion d'un point virgule (PRINT USING "....",Expr,Liste...;) inhibe le retour à la ligne CR/LF.

La sélection d'un canal de données (PRINT #1 USING...) redirige les données dans le canal spécifié.

WRITE — Ecrit les données

WRITE [#Canal,] ["Texte"[, Var, Expr ;...]]

Cette instruction de sortie répond spécifiquement à la sauvegarde de données dans des fichiers séquentiels. Toutefois, elle est utilisable à des fins d'édition de texte et de données sur l'écran. Elle est fort semblable à PRINT bien qu'elle connaisse une construction syntaxique différente. D'autre part, elle insère des virgules entre les expressions.

La signification propre de cette instruction ne se révèle que lors de la lecture simultanée de plusieurs valeurs ou chaînes contenues dans un fichier, entreprise à l'aide d'une instruction INPUT#. Pour cela, les données isolées doivent être séparées les unes des autres par des virgules. Puisque WRITE# écrit les virgules aux places correspondantes dans le fichier, les données sont distinguées les unes des autres lors de leur lecture par INPUT# et ainsi affectables aux variables spécifiées.

Exemple :

```
OPEN "O",#1,"Amis"                      ! ouvre le fichier pour l'écriture
RESTORE n_oms                           ! positionne le pointeur de fichier
FOR i%=1 TO 3                           ! 3 lignes
  READ nom$,mÉtier$,tÉl$                ! 3 données
  WRITE#1,nom$,mÉtier,tÉl$              ! écrit dans le fichier
NEXT i%                                 ! prochaine ligne
CLOSE #1                                ! clôt le fichier
n_oms:
DATA " Elisabeth "," Reine              "," Londres/122333    "
DATA " Kashogghi "," Milliardaire       "," Riad/1.000.000.000"
DATA " Yéti      "," Homme des neiges   "," Himalaya/inconnu  "
OPEN "I",#1,"Amis"                      ! ouvre le fichier pour lecture
PRINT "Mes admirateurs : ";CHR$(10)     ! Bla...
PRINT "Données non formatées"           !...bla
a$=INPUT$(LOF(#1),#1)                   ! lit le contenu complet du fichier
PRINT a$                                ! Sortie en l'absence de formatage
SEEK #1,0                               ! pointeur en début de fichier
PRINT "Editées à l'aide de WRITE";CHR$(10)
FOR i%=1 TO 3                           ! 3 lignes
  INPUT #1,n.om$,m.Étier$,t.Él$         ! 3 expressions
  WRITEn.om$,m.Étier$,t.Él$             ! Sortie à l'aide de
                                        ! WRITE
NEXT i%
SEEK #1,0                               ! pointeur en début de fichier
PRINT CHR$(10);"Editées à l'aide de PRINT";CHR$(10)
FOR i%=1 TO 3                           ! 3 lignes
  INPUT #1,n.om$,m.Étier$,t.Él$         ! 3 expressions
```

```
  PRINT n.om$,m.Étier$,t.Él$          ! Sortie à l'aide de PRINT
NEXT i%
CLOSE #1                              ! clôt le fichier
```

5.3 Opérations Ecran

HTAB (HT) — Position horizontale du curseur

HTAB xpos

Cette instruction sert essentiellement à assurer la compatibilité avec d'autres langages BASIC et complète CRSCOL.

LOCATE (LOCAT) — Positionne le curseur

LOCATE Colonne, Ligne

Positionne le curseur à la colonne et à la ligne spécifiées. Elle est analogue à PRINT AT(Colonne,Ligne). Consultez la description correspondante.

POS() — Etablit la colonne en cours

Var=POS(bidon)

POS délivre la position courante du curseur en terme de nombre de colonnes à partir du dernier retour chariot (CR). Lors de son emploi, l'adjonction d'un argument muet numérique mis entre parenthèses, est requise. Le choix de ce nombre importe peu puisque ce chiffre n'amène aucune autre influence sur la fonction.

Une ligne d'écran ne peut contenir plus de 80 signes. Néanmoins, la valeur délivrée par POS n'est pas nécessairement identique à celle de la colonne en cours, observée à l'écran. Lors de l'édition, par exemple d'une chaîne de 150 caractères, POS délivre une valeur de 150, bien que le curseur soit positionné dans la 70ème colonne.

La présence des codes CHR$(8) et CHR$(13) au sein de la dernière ligne éditée, influence la position délivrée par POS() :

```
Backspace          (BS=CHR$(8))   amoindrit POS() d'une unité
Carriage Return    (CR=CHR$(13)) annule POS()
```

SPC() — Sortie des espaces

PRINT SPC(Nombre)
PRINT [Expressions ; Valeurs ; etc.;] SPC(Nombre)[; etc.]

La fonction SPC n'est utilisable qu'en association avec PRINT. Le nombre mis entre parenthèse représente le nombre désiré d'espaces à éditer (SPC = SPaCe) à partir de la position en cours du curseur. Ce nombre peut revêtir des valeurs comprises entre 0 et 255.

Exemple :

```
PRINT "==>";SPC(20);"Fin"
      Sortie: ==>          Fin
```

La ligne d'instruction suivante est incorrecte :

```
A$="==>"+SPC(20)+"Fin"
```

Un message d'erreur en résulte. SPACE$(20) autorise de telles constructions.

TAB() — Détermine un arrêt de tabulation

TAB(Position)
PRINT [Expressions ; Valeurs ; etc. ;] TAB(Nombre)[; etc.]

Cette fonction oblige le curseur à se positionner sur l'arrêt de tabulation spécifié. Des valeurs comprises entre 0 et 255 sont utilisables. Un modulo est appliqué aux valeurs supérieures (par exemple 257 donnera 2 et 513 donnera 1). Si la valeur de TAB est inférieure à la position en cours du curseur, l'arrêt de tabulation est disposé dans la prochaine ligne. TAB tout comme SPC, n'est utile qu'associé à PRINT. Des constructions de chaînes avec TAB sont impossibles.

Exemple :

```
FOR J%=0 TO 11                    ! 11 fois
  RESTORE T_exte                  ! place le pointeur de données
  FOR I%=1 TO 2                   ! 4 données ...
    READ A$,A%                    ! ...à lire ...
    PRINT TAB(J%*6);A$''''''A%    ! ...à éditer
  NEXT I%                         ! prochaine donnée
NEXT J%                           ! prochaine position
T_exte;
DATA GFA-,1,BASIC,2
```

VTAB (VT) **Positionne le curseur à la ligne notifiée**

VTAB Ligne

VTAB et HTAB ont, en comparaison avec LOCATE et PRINT AT, l'avantage de permettre un positionnement du curseur à l'aide de l'indication isolée d'une colonne ou d'une ligne et non de celle des deux ensemble.

5.4 Les opérations de disquettes

La spécification d'un nom de chemin d'accès peut être de mise lors de manipulation de fichiers de disquettes. Le nom de chemin se compose en regard des conventions de l'AmigaDOS, des éléments suivants :

```
[Périphérique logique | Lecteur]:[Nom de répertoire/.../Nom de
répertoire/]Nom de fichier

Périphérique logique | Lecteur:
```

Tout en début du chemin s'énonce le nom de la disquette ou l'identificateur du lecteur dont l'accès est désiré, suivi d'un double point. Les périphériques logiques sont ceux qui ont été définis avec la commande du DOS Assign. Les identificateurs de lecteur se désignent comme suit :

```
DF0:, DF1:, DF2:, DF3: les quatre lecteurs maximum
DH0:, DH1:,...:        chaque partition de disque dur Amiga
JH0:, JH1:,...:        partition de disque dur PC
RAM:                   l'éventuel RAMDISK disponible
```

L'identification du lecteur peut être omise. Mais dans ce cas, la recherche s'effectue sur le lecteur courant.

Nom de répertoire :

En deuxième lieu, s'énonce le nom du répertoire contenant le fichier dont l'accès est désiré. Il vous est possible de créer des sous-répertoires, toutefois nous vous déconseillons une arborescence de plus de trois niveaux hiérarchiques afin que vous puissiez toujours être assurés d'une vue d'ensemble sur le contenu de votre disque dur. Les noms de répertoire se séparent les uns des autres par un trait incliné (/). Un nom de répertoire ne peut contenir plus de 30 caractères.

Nom de fichier :

La longueur maximale d'un nom de fichier est de 30 caractères. Il est séparé par une barre de fraction (trait incliné) du nom de son répertoire.

Exemples :

```
DF1:UTILITY/OUTPUTS/IMPRIM.DAT
```

Cet énoncé ordonne la recherche du fichier IMPRIM.BAT dans le sous-répertoire OUTPUTS du répertoire UTILITY de la disquette DF1.

```
PROGRAMMES:GFA_BASIC/GRAPHIQUE.LST
```

Cet énoncé ordonne l'accès au fichier GRAPHIQUE.LST du répertoire

```
GFA_BASIC de la disquette PROGRAMMES.
DH0: EXEMPLE
```

Cet énoncé dirige la recherche du fichier EXEMPLE sur la partition 0 du disque dur Amiga. En GFA Basic, la transmission du nom du chemin d'accès sous forme de variable ou d'expression de chaîne ou de combinaison de celles-ci, est possible dans la plupart des cas.

Par la suite, le concept de "#Canal" apparaîtra fréquemment. Sous ce terme est compris, lors des opérations relatives aux fichiers, l'identificateur (de 0 à 90) du canal du fichier auquel on désire accéder.

BLOAD [BL] charge un fichier en une zone de mémoire

BLOAD "Nom_de_fichier" [,Start]

Cette commande exécute le chargement d'un fichier quelconque complet disposé sur une mémoire de masse : une disquette ou un disque dur, ou en RAMDISK, en une adresse quelconque de la mémoire vive. Le nom de fichier ou son chemin d'accès est à porter dans "Nom_de_fichier". Le paramètre Start détermine l'adresse de début de la zone mémoire où les données seront mémorisées. Cette dernière indication étant omise, l'adresse cible utilisée est celle ayant servi de source lors du dernier appel de BSAVE. L'assignation de ces deux paramètres dans des variables est possible.

Au préalable de l'appel d'une commande BLOAD, il convient de s'assurer que le chargement des données n'amène pas leur écriture en des segments de mémoire contenant des informations essentielles. L'évitement de cela s'obtient dans la plupart des cas, par la simple création d'un tampon de mémoire d'une taille conséquente sous la forme d'une variable de chaîne.

BSAVE [BSAVE] Sauvegarde de zone de mémoire

BSAVE "Nom_de_fichier", Start, Nombre

Cette commande dont l'utilisation est d'une fréquence élevée, est d'un confort certain. Elle offre la possibilité de sauvegarder un nombre quelconque d'octets d'une zone ou bloc de mémoire spécifié (BSAVE = Block-SAVE) sur disquette. La notification du nom du fichier devant contenir ce bloc est requise en premier. Après une virgule, suivent l'adresse de l'octet du début de la zone, et, après une autre virgule, le nombre d'octets à sauvegarder.

CHAIN [CHAI] — Lance un programme (Autostart)

Cette instruction est analogue à LOAD. Toutefois le programme ainsi chargé, s'en trouve simultanément lancé. Puisque cette modalité de lancement de programme, comme toute autre d'ailleurs, amène la suppression de toutes les variables et matrices mémorisées, l'échange de ces dernières entre programmes "chaînés" est impossible. Néanmoins deux possibilités permettent de surseoir à cette impossibilité :

Il vous est possible, avant l'appel du programme chaîné, d'écrire toutes les informations que vous voulez échanger, dans un fichier séquentiel (Cf. PRINT#, WRITE#), puis de lire celles-ci au début du programme chaîné (Cf. INPUT#) et par la suite, d'effacer le fichier séquentiel correspondant. L'avantage de cette procédure consistedans le fait que la taille des données échangées n'est limitée que par l'espace libre disponible sur le disque.

Exemple :

Programme 1

```
A%=1025                      ! une variable quelconque
B=399.22                     ! une variable quelconque
C$="BASIC"                   ! une variable quelconque
OPEN "O",#1,"VARS.DAT        ! ouvre le fichier séquentiel
WRITE #1,A%,B,C$             ! écrit les données
CLOSE #1                     ! Clôt le fichier séquentiel
CHAIN "PROGRAMME2.GFA"       ! appelle le programme 2
```

Programme 2

```
OPEN "I",#1,"VARS.DAT        ! ouvre le dit fichier séquentiel
READ #1,A%,B,C$              ! lit les données
CLOSE #1                     ! clôt le fichier séquentiel
KILL "VARS.DAT"              ! détruit le fichier séquentiel
PRINT "Variables du Programme 1: ";A%,B,C$
```

L'appel de CHAIN conduit l'effacement du complet contenu de la mémoire utilisateur y compris le programme appelant. Si le programme appelé n'a pas été protégé avec PSAVE, le programme appelé apparaît dans l'éditeur à la fin de l'exécution.

CHDIR [CHR] — Changer le répertoire courant

CHDIR "Nom_de_répertoire"

Cet ordre change le répertoire courant. Le répertoire courant est celui sur lequel tous les accès disques opérés aboutissent dans la suite du programme, du moins tant que le chemin d'accès n'est pas modifié.

Le chemin d'accès menant au répertoire est adjoint à la suite de la commande, par exemple CHDIR "UTILITY". L'utilisation d'une variable est aussi possible, par exemple CHDIR Répertoire$.

Si la chaîne de caractères définissant le nouveau chemin d'accès au fichier contient un double point (par exemple "DF1:" ou "LIBS:"), il sera effectué ce que l'on appelle un changement de périphérique (physique dans l'exemple de "DF1:"et logique pour "LIBS:"). Si le ":" figure, seul le répertoire courant deviendra le répertoire principal (ou la racine) de l'unité en cours (unité physique, bien sûr !).

En outre, il vous est possible de sélectionner le répertoire immédiatement supérieur par la simple adjonction d'un trait incliné à gauche (/). Supposons que le sous-répertoire en cours "Chemise1" est contenu dans le répertoire "Dossier" contenant, par ailleurs, un sous-répertoire "Chemise2" dont la mise en cours est désirée. En ce cas de figure, la notation : CHDIR "/Chemise2" au lieu de : CHDIR "/Dossier/Chemise2" réalise ce désir.

DFREE() — Retourne l'espace libre de la disquette

Var=DFREE(0)

DFREE(0) délivre l'espace libre momentanée du lecteur en cours (sélectionné à l'aide de CHDIR). La valeur 0 est un argument muet et ne recèle aucune autre signification. Attention cependant au fait que le RAM-Disk renverra toujours 0 à l'appel de DFREE. On obtient la taille de la mémoire disponible avec la fonction AvailMem().

DIR — Edite un répertoire

DIR ["Nom de chemin"] [TO "fichier"]

Cette commande ordonne l'édition du répertoire ou du sous-répertoire dont le chemin d'accès est spécifié. Cette édition peut être dirigée dans un fichier ou sur l'imprimante. Le nom du chemin obéit aux règles décrites en début de chapitre.

Exemple :

```
DIR: "DF0:PROGRAMMES"
```

Edite le répertoire de PROGRAMMES.

```
DIR "DF1:" TO "PRT:"
```

Dirige l'édition du répertoire de la disquette DF1 sur l'imprimante.

DIR$() — Retourne le chemin d'accès en cours

Var$=DIR$(0)

Cette fonction retourne le chemin d'accès (déterminé à l'aide de CHDIR) momentanément en cours. Aucun répertoire n'étant en cours, une chaîne vide est délivrée.

EXIST() — Vérifie l'existence d'un fichier

Var=EXIST(Nom de fichier)

Cette fonction vérifie l'existence ou l'inexistence du fichier spécifié. Elle retourne soit la valeur 0 (faux), soit -1 (vrai). La description du nom de chemin obéit aux mêmes règles que celles décrites en début de chapitre.

Exemple :

```
IF EXIST("DF1:DONNEES.DAT")      ! si DONNEES.DAT existe sur
                                 ! DF1
OPEN "I",#1"DF1:DONNEES.DAT"     ! alors ouvre le fichier
(....suite du programme...)
ELSE                             ! sinon
  PRINT "DONNEES.DAT est inexistant"
ENDIF
```

FILES — Edition étendue de répertoire

FILES ['Nom de chemin'] [TO 'Fichier']

Si l'édition des seuls noms de fichiers est insuffisante, cette commande vous assure la livraison d'informations supplémentaires sur les fichiers répertoriés. Son exécution amène l'édition complémentaire de leur taille, date et heure de sauvegarde. Du reste, l'utilisation de cet ordre est strictement identique à celle de DIR (par exemple Files édite tous les fichiers et sous-répertoires (leur contenu mis à part) du répertoire en cours).

KILL { K } — Supprime des fichiers de disque

KILL 'Nom de fichier'

Cette commande supprime un fichier. La spécification de son nom ou de son chemin d'accès (Cf. "Chemin" en début de chapitre) est requise.

LIST { LIS } — Liste et sauvegarde les programmes (ASCII)

LIST ['Nom de fichier']

Cette commande liste sur l'écran le programme contenu en mémoire utilisateur ou le sauvegarde sur disquette sous forme de fichier ASCII. Son insertion isolée directe ou sous forme de ligne de programme, en l'omission d'un nom de fichier, amène le listage du programme tout

entier dans la fenêtre d'édition. L'édition de ce listing peut être interrompue avec la pression de la combinaison de touches d'interruption GFA BASIC: <Control/Shift/Alternate>. La sauvegarde de ce listing en un fichier ASCII requiert lors de l'insertion de la commande l'adjonction du nom du fichier, devant contenir le listing. Cet ordre est identique à la fonction de l'éditeur Save,A. Il vous est possible relier un programme sauvegardé de cette manière à l'aide de la fonction de l'éditeur : Merge à celui disposé en mémoire utilisateur. L'omission des guillemets encadrant le nom de fichier ne porte pas à conséquence puisque l'interpréteur y supplée de lui-même. Il en va de même lors de l'oubli de l'extension (.LST). Elle est accolée automatiquement par l'interpréteur, du moins tant qu'aucune n'est spécifiée. Lors du double emploi d'un même nom de fichier dans un même répertoire, le fichier déjà existant est automatiquement rebaptisé avec l'extension .BAK.

Les caractères et les codes de contrôle du texte contenu dans un fichier ASCII y sont écrits sous leurs propres codes ASCII et selon leur ordre d'apparition initiale. La plupart des traitements de texte offrent la possibilité de sauvegarder et d'éditer des textes en format ASCII. De tels fichiers de textes sont interchangeables entre programmes, systèmes et ordinateurs différents, du moins s'ils respectent l'American Standard Code for Information Interchange. Les fichiers dont le format est différent, comme par exemple les fichiers .GFA, sont chiffrés d'après un autre code. Le code spécifique du GFA Basic : le Token Code répond d'autres nécessités. Il accélère les vitesses de chargement, de sauvegarde et de traitement. Il autorise des effets spéciaux, le cryptage des programmes (Cf. PSAVE), etc. Les fichiers Token ne sont pas échangeables avec d'autres programmes (interpréteurs, traitements de texte, etc.). Leur compatibilité n'est aucunement assurée.

L'impression du listing résulte de la simple notification du port de l'imprimante PRT: au lieu de celle du nom de fichier. Cette option opère le même procès que celui observé par la commande LLIST où nulle option n'est prise en compte (cf la fonction de Llist).

LOAD — Charge un programme

LOAD "Nom_du_programme"

Cette commande autorise le chargement d'un programme GFA Basic, en mémoire utilisateur. En son principe, cet ordre procède de manière analogue à celle observée lors de l'activation de l'option Load de menu de l'éditeur ou lors de la pression de la touche <F1> dans la fenêtre de l'éditeur. La spécification du nom de chemin d'accès (selon la syntaxe usuelle décrite en début de chapitre) du programme à charger, est requise. Lors de l'omission de l'extension (.GFA), l'interpréteur raccole de lui-même, par exemple :

```
LOAD "UTILITY/TROIS_D"
```

Le programme en cours est arrêté et le chargement du programme "TROIS_D" est opéré. Il ne reste plus qu'à le lancer avec l'insertion de RUN en mode de commande direct, ou à l'aide d'une pression de <F10> ou avec un cliquement de l'option Run du menu de l'éditeur. Cette procédure n'est pas requise suite au chargement d'un programme sauvegardé avec PSAVE puisque son lancement est automatique.

MKDIR [MK] — Crée un répertoire

MKDIR "Nom de répertoire"

A l'aide de cet ordre, il est possible de créer un répertoire, en un lieu quelconque d'un disque ou d'une disquette. En cas de nécessité, le nom de chemin (descrit en début de chapitre) est spécifié dans la ligne de commande. Les exemples suivants illustrent certaines de ses applications possibles :

```
MKDIR "PRIVES"
```

Crée le répertoire nommé : PRIVES dans le répertoire en cours.

```
MKDIR "DF0:DOCUMENTS/PRIVES"
```

Crée le répertoire nommé : PRIVES dans le répertoire déjà existant: DOCUMENTS sur la disquette disposée dans le lecteur DF0:. L'existence d'un répertoire dont le nom de chemin est identique, provoque l'apparition d'un message d'erreur.

NAME [NA AS] — Renomme un fichier

NAME "Ancien_nom" AS "Appellation_nouvelle"

Cet ordre permet de changer le nom d'un fichier. Il requiert deux paramètres. Le premier est le nom du fichier dont la modification est désirée. Le second est inséré après l'énoncé AS et désigne l'appellation nouvelle qui remplacera l'ancienne. L'affectation des deux noms peut s'ordonner sous forme d'expression ou variable de chaîne ou de combinaison des deux. La notification du nom de chemin si elle est requise, obéit ici aux mêmes règles que celles décrites en début de chapitre. Veillez en ce cas, à indiquer le même identificateur de lecteur (par exemple DF0:), et le même chemin, pour les deux noms. Si le chemin de l'ancien nom correspond au chemin du répertoire en cours, sa spécification peut être omise dans l'appellation nouvelle.

PSAVE [PS] — Sauvegarde un programme protégé en lecture

PSAVE "Nom de programme"

Son exécution est identique à celle de SAVE. Toutefois cet ordre possède une propriété minime particulière dont l'effectivité est fort impressionnante. L'initiale P du nom de la commande représente l'adjectif protected (protégé). Les programmes GFA sauvegardés à l'aide de cet ordre sont :

- Protégés en lecture. Leur listage est impossible.
- Lancés immédiatement après leur chargement (Cf. LOAD, Autostart).

– Non modifiables. La tentative de la modification de leur listing dont l'édition est impossible, à l'aide de l'interpréteur, se solde tôt ou tard par un blocage total de la machine. Les programmes protégés à l'aide de PSAVE rendent indisponibles les pointeurs des noms de variables requis, et l'interpréteur lors de leur recherche se perd en un embrouillamini d'adresses. Notez bien que les programmes protégés sont faits pour fonctionner avec le GFABASRO et non sous éditeur.

RENAME [REN] — Renomme un fichier

RENAME "Ancien_nom" AS "Appellation_nouvelle"

Cet ordre est analogue à NAME (Cf. NAME).

RMDIR [RM] — Supprime un répertoire

RMDIR "Nom_de_répertoire"

La syntaxe de cette commande est identique à celle de CHDIR. Cependant, elle n'effectue pas de changement de répertoire, mais sa suppression pure et simple. Toutefois, si des fichiers sont encore contenus dans le répertoire, son exécution est impossible et provoque un message d'erreur. Dans ce cas, il convient d'abord d'effacer les fichiers qui y sont entreposés à l'aide de KILL.

SAVE [SA] — Sauvegarde d'un programme codé

SAVE "Nom du programme"

SAVE sauvegarde en format Token le programme disposé en mémoire utilisateur, sous le nom spécifié, sur une mémoire de masse (disquette ou disque dur) ou en RAMDISK.

5.5 Gestion des fichiers

BGET (BG) — Lit des portions de fichiers

BGET [#]Canal,Start,Nombre

Canal est l'identificateur d'un fichier ouvert à l'aide d'OPEN "I" ou "U". A partir de la position courante du pointeur de fichier, l'instruction lit le nombre spécifié d'octets du fichier ouvert et les place en la zone de mémoire débutant à l'adresse Start. Si le fichier n'est pas ensuite clos par CLOSE, le pointeur de fichier reste positionné sur l'octet suivant, celui qui est lu en dernier. Le prochain accès au fichier (BGET#, INP(#), INPUT#, PRINT#, OUT#, etc.) s'effectue alors à cet endroit.

Exemple :

```
OPEN "O",#1,"DF0:TEST.DAT"    ! ouvre un fichier output
PRINT #1;"GFA-BASIC";         ! y écrit une chaîne
CLOSE                         ! et le ferme
A$=SPACE$(5)                  ! crée un petit tampon
OPEN "I",#1,"DF0:TEST.DAT"    ! ouvre fichier pour lecture
SEEK #1,4                     ! saute 4 octets
BGET #1,VARTPRT(A$),5         ! lit 5 octets dans le tampon
CLOSE                         ! ferme le fichier
PRINT A$                      ! édite la chaîne
```

BPUT (BP) — Copie une zone de mémoire en un fichier

BPUT [#]Canal, Start, Nombre

Lit le nombre spécifié d'octets à partir de l'adresse Start et les copie à partir de la position du pointeur de fichier, dans le fichier (ouvert par OPEN "O" ou "U") notifié par son identificateur de canal. A l'inverse de BSAVE, cette instruction permet l'écriture des données sur celles déjà existantes ou leur insertion à la suite dans le fichier. Le fichier n'étant pas clos avec CLOSE à la fin de l'opération, le pointeur de fichier reste positionné sur l'octet suivant celui écrit en dernier. Le prochain accès au fichier (BGET#, INP(#), INPUT#, PRINT#, OUT#, etc.) se réfère alors à cette position.

CLOSE [CL] — Clôt un canal de données

CLOSE [#Canal]

Tous les canaux ouverts avec OPEN requièrent leur fermeture avec CLOSE #Canal afin que leur numéros soient utilisables pour d'autres fichiers. L'utilisation isolée de CLOSE s'il y a omission de toute spécification complémentaire, opère la fermeture simultanée de tous les canaux. L'ouverture renouvelée à l'aide d'OPEN d'un canal non encore clos avec CLOSE #Canal, provoque l'apparition d'un message d'erreur. Après l'exécution du programme, l'accès aux fichiers ouverts est possible à partir du mode de commandes direct, tant qu'aucune modification de programme n'est entreprise dans l'éditeur.

EOF () — Vérifie l'atteinte de la fin de fichier

Var=EOF(#Canal)

Cette fonction détermine l'atteinte ou non de la fin de fichier (EndOfFile). Il retourne la valeur -1 (TRUE) si le pointeur de fichier est sur le dernier octet du fichier spécifié à l'aide de son canal, en cas contraire la valeur 0 (FALSE).

```
OPEN "I",#1,"Nom"    ! ouvre un fichier afin de le lire
WHILE EOF(#1)=False  ! tant que la fin du fichier n'est pas
                     ! atteinte
  PRINT INP(#1)      ! lit et édite un octet isolé
WEND                 ! réitère la boucle
```

FIELD [FI AS ou FI AT] — Attribue des énoncés à des champs

FIELD #Canal, Nombre AS Var1$[, Nombre AS Var2$]
FIELD #Canal, Nombre AT(Adr1)[, Nombre AT(Adr2)]

Cette instruction réserve, dans les enregistrements du fichier en accès direct, un nombre d'emplacements équivalent à celui des composantes Nombre AS Var$ notifiées. Les nombres d'octets ainsi spécifiés en Nombre sont affectés aux variables de chaîne Var1$, Var2$, etc. qui les

suivent, et celles-ci sont remplies simultanément d'espaces. Dans la seconde forme syntaxique, Nombre contient le nombre d'octets devant être lus à partir de l'adresse spécifiée entre parenthèses.

Exemple :

```
a%=673123
b%=VARPTR(a%)
c&=1000
d=61234.1231
FIELD #1,4 AT(b%),2 AT(*c&),8 AT(*d)
```

Les formes syntaxiques AT et AS sont miscibles en une même ligne de commande FIELD (par exemple : FIELD #1,20 AS a$,2 AT(*b&)). En outre, l'instruction FIELD n'est pas limitée à une seule ligne. Les réservations d'emplacements au sein d'un même fichier peuvent se poursuivre sur plusieurs lignes de programme. Ces dernières sont considérées comme appartenant à un même ensemble. Illustration sous 5.5.1 "les modalités fonctionnelles d'un fichier à accès direct".

GET # — Lit un enregistrement

GET #Canal [, Numéro_d'enregistrement]

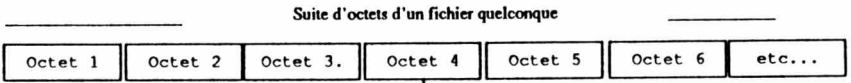
L'instruction lit l'enregistrement spécifié dans le fichier en accès direct (Cf. OPEN "R") notifié à l'aide de son numéro de canal. Lors de l'affectation d'enregistrements à un fichier à accès direct, l'attribution d'un numéro est requise pour chaque enregistrement. Cette indication permet la lecture de l'enregistrement correspondant par GET# et son placement dans la variable de chaîne spécifiée avec FIELD. Lors de l'omission du numéro d'enregistrement, l'enregistrement après le dernier GET ou celui placé par RECORD. Illustration sous 5.5.1 "les modalités fonctionnelles d'un fichier d'accès direct".

LOC () — Délivre la position du pointeur de fichier

Var=LOC(#Canal)

Cette abréviation de LOCATION retourne la position en cours du pointeur d'écriture et de lecture (en anglais File-pointer) du fichier spécifié à l'aide de son numéro de canal. L'évaluation de la position s'opère à partir du début de fichier en terme d'octets.

Exemple :




```
Hypothèse : Le pointeur actuel est placé sur l'octet 4
Pointeur%=Loc(#1)    ! La valeur 4 est retournée dans Pointeur%
```

LOF () — Retourne la longueur d'un fichier

Var=LOF(#Canal)

Cette fonction (l'abréviation de Length Of File) retourne la longueur en terme d'octets du fichier - spécifié par son numéro de canal mis entre parenthèses - disposé sur disquette ou disque dur.

Exemple :

```
OPEN "U",#1,"Nom_de_fichier   ! d'abord ouvrir le fichier
PRINT LOF(#1)                 ! Variante 1 : édition directe
A%=LOF(#1)                    ! Variante 2 : affectation à 1 variable
IF LOF(#1)>32767              ! Variante 3 : test conditionnel
  A$=INPUT$(32767,#1)
ELSE
  A$=INPUT$(LOF(#1),#1)       ! Variante 4 : liaison
ENDIF
et ainsi de suite...
```

OPEN { O } Ouvre un canal de données

OPEN "Mode", #Canal, "Nom_de_fichier" [,longueur d'enregistrement]

Cette commande ouvre un fichier et introduit de ce fait la gestion des fichiers. A première vue, sa syntaxe paraît quelque peu compliquée. Sa connaissance acquise, ses qualités seront fort appréciées. Les possibilités ainsi offertes de gestion des fichiers sont proprement inépuisables.

Mode :

"O" Output ouvre un fichier ou le crée si cela est nécessaire. Si le fichier spécifié existe déjà, sa longueur est annulée. Dans ce mode, seul l'accès à des fins d'écriture est possible.

"I" Input ouvre un fichier existant en lecture. Le pointeur est positionné sur le premier octet du fichier. La longueur des données n'est affectée en rien lors des accès en lecture. Seuls des accès de lecture sont possibles en ce mode.

"A" Append ouvre un fichier existant et positionne le pointeur en fin de fichier. Toutes les données transmises au fichier sont ajoutées en fin de fichier (append = accrocher, ajouter).

"U" Update ouvre un fichier à des fins simultanées d'accès en écriture et en lecture. La longueur des données initiale reste conservée. Lors l'envoi de données, l'écriture peut s'effectuer de ce fait au delà de la fin de fichier.

"R" Random ouvre un fichier en accès direct, disposé en mémoire vive. Seuls un nombre d'enregistrements d'une taille totale maximum de 65535 octets y sont insérables. Veuillez consulter le chapitre 5.5.1 "Les modalités fonctionnelles d'un fichier d'accès direct" pour de plus amples précisions.

#Canal :

Un maximum de 100 fichiers peuvent être ouverts simultanément. Lors de l'ouverture d'un fichier, un chiffre est affecté à chaque fichier à des fins d'identification. La notification de cet identificateur précédé du cardinal: #, permet l'accès au fichier ainsi identifié, à l'aide de instructions: PRINT#, INPUT#, BGET#, etc. Les numéros de canal doivent être compris entre 0 et 99.

Nom_de_fichier :

Désigne le nom du fichier à ouvrir sous le mode d'accès spécifié. Son chemin d'accès y est notifié selon les règles décrites en début de chapitre.

Exceptions :

Presque toutes les instructions d'entrées/sorties #Canal, sont utilisables pour accéder aux diverses interfaces de l'Amiga. Un fichier virtuel est ouvert bien qu'il s'agisse en réalité d'un port. Divers ports sont disponibles dans l'Amiga :

CON : Celui de la console.

PRT : Celui de l'imprimante. Il assure l'accès à l'imprimante sélectionnée dans Preferences.

PAR : Celui de liaison parallèle; éventuellement le port de l'imprimante. Au contraire de PRT, aucune conversion des codes de contrôle n'y est effectuée).

SER : Celui de la liaison série, ses paramètres étant ceux sélectionnés dans Preferences.

COM1: Le port série avec les paramètres définis à l'ouverture du fichier.

PIPE : Le gestionnaire de pipes (ou tubes).

SPEAK : La synthèse vocale.

RAW : Console étendue

NEWCON : Console des fenêtres Shell.

AUX : Comme SER: mais sans tampon.

L'énoncé complet du port, le double point compris est à indiquer à la place du nom de fichier. La spécification d'un mode propre d'OPEN peut être omise lors de l'accès à ces interfaces.

Longueur_d'enregistrement :

Il est possible d'affecter des enregistrements identifiés à l'aide de numéro, dans un fichier d'accès direct. Le paramètre Longueur_d'enregistrement répond précisément de cela. Sous ce terme est comprise une valeur dont la détermination, à des fins de gestion du fichier, est de votre entier ressort. Elle détermine le nombre d'octets alloués à chaque enregistrement du fichier. Lors de l'omission de cette indication dans une ligne d'instruction OPEN, l'interpréteur prévoit une longueur d'enregistrement de 128 octets. De plus amples informations à ce sujet sont consignées dans le chapitre 5.5.1 "Les modalités fonctionnelles d'un fichier d'accès direct".

Une ligne d'instruction OPEN peut être abrégée à l'extrême. Ainsi, par exemple l'énoncé : o I 1 NOM.LST est traduite par l'interpréteur en : OPEN "I",#1,"NOM.LST" ou celui : o o 1 NOM$ en OPEN "o",#1,NOM$. L'omission des guillemets, du cardinal ou de la virgule ne porte aucunement à conséquence tant que les composantes isolées sont séparées entre elles par un espace.

PUT # [PU] — Ecrit un enregistrement

PUT #Canal [,Numéro_d'enregistrement]

Cette instruction écrit, dans le fichier d'accès direct dont le canal est notifié, l'enregistrement désigné contenu dans une variable de chaîne et d'adresse spécifiées à l'aide de FIELD. Lors de l'omission du numéro d'enregistrement, l'enregistrement lu est le suivant (le dernier lu par GET) ou celui placé par RECORD. De plus amples informations à ce sujet sont consignées dans le chapitre 5.5.1 "Les modalités fonctionnelles d'un fichier d'accès direct".

RECALL [RECA] Lit des champs à partir d'un fichier

RECALL #Canal, Matrice$(), Nombre, Varlignes

En liaison avec STORE#, cette instruction est d'une utilité très appréciable notamment lors du développement d'un traitement de texte. Qui s'est déjà occupé d'une telle tâche, sait de quoi je parle.

RECALL lit le nombre de lignes de texte spécifié en Nombre, dans le fichier ouvert spécifié en #Canal et les place dans le tableau Matrice$. Lors de cette opération les codes de contrôle CHR$(10) (Carriage Return) et CHR$(13) sont interprétés en tant que codes de fin de lignes.

Les éléments de la matrice sont ordonnées selon le cas à partir soit de l'élément 0 (OPTION BASE 0), ou soit de l'élément 1 (OPTION BASE 1), et affectées en une ligne isolée. Si la matrice est trop courte - le nombre de ses éléments étant moindre que celui des lignes lues, la lecture s'arrête au dernier élément en l'absence de l'apparition d'un quelconque message. La lecture est abandonnée suite à l'atteinte de la fin de fichier sans apparition de message non plus. Varligne$ désigne une variable numérique (un nombre à virgule flottante ou entier - de 32 bits). Le nombre des octets lus y est retourné suite à l'exécution de l'instruction.

L'expression "de TO à" peut remplacer Nombre, les lignes sont alors placées de l'élément "de" jusqu'à celui "à".

Si fichier n'est pas fermé noamment avec CLOSE, le pointeur de lecture après une instruction RECALL, reste positionné sur le début de la prochaine ligne. Cela signifie que lors de l'appel suivant de RECALL seule la partie restante du fichier est lue. La lecture du début de fichier étant requise ou désirée, il convient de placer au préalable le pointeur sur le début de fichier avec SEEK #Canal,0.

RECORD (REC) Positionne le pointeur d'enregistrement

RECORD [#] Canal, Numéro_d'enregistrement

Positionne le pointeur d'enregistrement pour les accès des prochaines instructions GET# et PUT# dans un fichier d'accès direct. L'omission du paramètre Numéro_d'enregistrement dans une instruction GET# ou PUT#, induit la lecture ou l'écriture de l'enregistrement sélectionné par RECORD#.

RELSEEK (REL) Déplace le pointeur de fichier

RELSEEK #Canal,[-] Offset

RELSEEK (relative seek) déplace le pointeur du fichier dont le canal est noté, en regard de sa position en cours, vers la fin ou le début du fichier, et cela du nombre d'octets spécifié dans Offset. La direction vers le début de fichier requiert l'assignation d'une valeur négative dans Offset.

Exemple :

```
OPEN "U",#1,"Nom_de_fichier" ! ouvre le dit fichier
SEEK #1,INT(LOF(#1)/2        ! le pointeur pointe l'octet du
                             ! milieu du fichier
RELSEEK #1,3                 ! le pointeur est déplacé vers
                             ! lafin de fichier de 3 octets
RELSEEK #1,-6                ! le pointeur est déplacé vers
                             ! le début de fichier de 6
                             ! octets, il pointe maintenant
                             ! le 3ème octet précédent le
                             ! milieu de fichier déterminé
                             ! par SEEK
et ainsi de suite...
```

Veillez lors de l'emploi des instructions SEEK et RELSEEK à ne pas donner au pointeur des positions de fichier inexistantes. La notification du nombre 0 ou de nombres trop élevés dans Offset provoque l'apparition du message d'erreur "SEEK FALSE?".

SEEK [SEE] Positionne le pointeur de fichier

SEEK #Canal,[-] Offset

Cette instruction positionne le pointeur dans le fichier désigné à l'aide de son canal, sur l'octet spécifié dans Offset. L'indication d'une valeur positive réfère le positionnement du pointeur sur le début de fichier; celle d'une valeur négative sur la fin de fichier. Une possibilité d'application de cet ordre est exposée dans l'exemple précédent.

TOUCH [TOU] Actualise la date d'un fichier

TOUCH #Canal

Affecte la date système en cours (Cf. TIME$) dans les paramètres internes du fichier spécifié (ouvert par OPEN). L'édition de ces paramètres s'obtient à l'aide de FILES.

STORE [STOR] Ecrit des enregistrements dans un fichier

STORE #Canal, Matrice$()[,Nombre]

STORE sauvegarde les éléments contenus dans le tableau Matrice$ selon leur ordre initial, dans le fichier dont le canal est notifié. Lors de cette opération, un code de fin de ligne CR/LF (Carriage Return/Line Feed) est ajouté à chaque ligne écrite. Le paramètre optionnel : Nombre permet de déterminer le nombre maximal d'éléments à sauvegarder. L'omission de ce paramètre entraîne l'écriture de tous les éléments de la matrice dans le fichier.

Il vous est possible de vous servir de l'expression "de TO à" au lieu de spécifier Nombre, comme avec l'instruction RECALL, afin de ne sauvegarder qu'une partie du tableau.

5.5.1 Les fichiers d'accès direct

Les commandes de manipulations de fichiers d'accès direct nécessitent un approfondissement particulier.

Tous les autres modes d'accès à une disquette ou à un disque dur, sont séquentiels. Les données y sont écrites ou lues signe par signe en une série linéaire. Cet ordre n'est modifiable qu'avec SEEK ou RELSEEK.

Un fichier à accès direct est analogue à un tableau à deux dimensions. Plusieurs enregistrements peuvent y être rassemblés sous un même libellé, précisément ici le nom de fichier. En analogie à l'identification d'un élément de tableau à l'aide de son indice, chaque enregistrement d'un fichier d'accès direct est identifié à l'aide d'un numéro propre. La création d'un tel fichier requiert en premier lieu son ouverture. Son statut particulier résulte de la commutation du mode "R" lors de son ouverture :

```
OPEN "R",#1,Nom_de_fichier,longueur_de_l'enregistrement
```

Cette chose faite, l'ordre FIELD permet de déterminer le nombre d'emplacements à réserver dans un enregistrement et leur taille en octets. La longueur du tampon étant notifiée lors de l'ouverture(OPEN), il convient de veiller à ce que la taille totale des emplacements ou champs réservés ne dépasse pas la longueur de l'enregistrement. Néanmoins elle peut être inférieure. Ces déterminations de la longueur de l'enregistrement et de la distribution en son sein des champs ne devraient plus, à partir de là, être modifiées. Si jamais vous transgressiez cette règle, la transmission correcte des données aux champs alloués n'est plus assurée. Certes il vous est possible d'allouer de nouveaux champs à un nouvel enregistrement dans un fichier de même nom, mais les données précédemment contenues s'en trouveront supprimées.

D'autre part, la taille des enregistrements écrits à l'aide de PUT dans le fichier ne doit non plus être modifiée, car les données de l'enregistrement suivant rentrerait en collision avec celles de celui en cours. Le respect des longueurs déterminées des champs et des enregistrements s'obtient par l'utilisation des deux fonctions LSET et RSET. Elles insèrent les données transmises dans les variables de chaînes affectées aux champs sans modifier leur longueur.

L'extension et la modification d'un fichier à accès direct s'opère grâce à l'insertion ou à l'ajout de nouveaux enregistrements à l'aide de l'instruction PUT#. Les enregistrements écrits peuvent aussi être supprimés sans détour. Suite à la lecture d'un tel fichier avec l'aide de RECALL, il suffit d'y traiter les enregistrements à l'aide d'INSERT et de DELETE, et puis de le sauvegarder avec STORE.

L'extension d'un enregistrement (par l'ajout de données par exemple) n'est seulement possible que suite à la création d'un nouveau fichier avec une nouvelle structure FIELD, par le transfert des enregistrements - selon le principe des vases communicants - et de leur extension simultanée, dans le nouveau fichier.

Veillez à indiquer, à chaque ouverture d'un fichier à accès direct déjà existant, une longueur d'enregistrement supérieure ou strictement égale à celle qui lui est propre. Si la longueur de l'enregistrement est omise, la valeur par défaut est de 128 octets. Au cas où la dimension portée lors de la création du fichier est supérieure à celle-ci, l'essai de l'accès à un champ extérieur à cette longueur provoque l'apparition du message d'erreur "Fin de fichier atteinte".

Le graphique suivant expose ce principe :

1er enregistr. Longueur 45 octets			**2ème enregistr.** Identique			**3ème enregistr.** Identique		
Chp1 10 Oct	Chp2 20 Oct	Chp3 25 Oct	Chp1 10 Oct	Chp2 20 Oct	Chp3 25 Oct	Chp1 10 Oct	Chp2 20 Oct	Chp3 25 Oct

Il vous est possible de déterminer librement le nombre des champs, leur taille et la longueur des enregistrements. La seule limite existante est celle de la longueur maximale des enregistrements qui est de 65535 octets; la taille totale des données comprises dans les enregistrements spécifiés, doit respecter par voie de conséquence la même limite.

Le programme suivant illustre et approfondit le présent sujet. Nous espérons qu'il vous incitera à mener vos propres expérimentations. Ce programme se compose de quatre blocs d'instructions distincts. Le premier bloc crée un fichier à accès direct d'une longueur d'enregistrement de 90 octets. Ces 90 octets sont distribués en 3 champs d'une taille respective de 40, 30 et 20 octets à l'aide de l'instruction

FIELD. Simultanément y sont affectées trois variables de chaînes remplies par des espaces, pour la réception future des données des champs. Ensuite la boucle FOR...NEXT introduit la saisie des variables de 6 enregistrements. Suite à la saisie des variables, les contenus des variables INPUT sont alignés à gauche et insérés à l'aide de LSET dans les variables de champs réservées à cet effet auparavant. En dernier, les enregistrements sont écrits l'un après l'autre avec leur numéro propre (notifié ici par l'indice de boucle I%) à l'aide de PUT# dans le fichier.

Le deuxième bloc démontre l'extension et la modification d'un fichier à accès direct. Les textes de ce programme sont arbitraires, il vous est possible de les modifier selon vos propres exigences ou souhaits. Les deux derniers blocs montrent deux possibilités de sortie et d'entrée des données.

5.6 Instructions d'entrées/sorties sur les ports

INP — Lecture d'un octet sur le port spécifié

Var=INP(#Canal)

Cette fonction sert à lire un octet isolé dans le fichier notifié. Le numéro du canal de données correspondant est spécifié mis entre parenthèses.

OUT [OU] — Envoie un ou plusieurs octets sur un port

OUT #Canal,Octet1[,Octet2[Octet3,...]]

Cet ordre est le contraire d'INP(). Il envoie des valeurs d'octets dans un canal de données déterminé.

Exemple :

```
OUT #1,65
```

écrit la valeur ASCII du caractère A (code ASCII 65) à la position en cours du pointeur du fichier dont le canal de données est 1.

5.7 La bibliothèque DOS de l'Amiga

La bibliothèque DOS (en langue anglaise: DOS Library) représente comme toutes celles contenues dans l'Amiga, une collection de routines exploitables par le programmeur en tout autre langage informatique. Ces routines ont été implémentées pour simplifier les accès aux divers périphériques, plus particulièrement ceux aux mémoires de masse (disque dur, disquettes, etc.).

Heureusement, le GFA BASIC offre de nombreuses instructions d'entrées et de sorties bien plus avantageuses. A quoi bon alors utiliser cette bibliothèque DOS ? En certaines circonstances, si vous souhaitiez par exemple analyser le répertoire d'une disquette, la bibliothèque DOS pourrait se montrer d'une utilité irremplaçable puisque le GFA BASIC n'offre aucune commande comparable.

Toutefois la présence de certaines autres routines est proprement superflue en ce qui nous concerne, puisque les instructions correspondantes du GFA BASIC sont d'un confort d'usage bien meilleur. Néanmoins, notre propos n'est pas de vous déconseiller leur usage. Les brèves descriptions suivantes s'en tiennent à l'essentiel. Si jamais vous désiriez approfondir la connaissance des routines DOS de l'Amiga, leur exposition détaillée est présentée dans la Bible de l'Amiga Tomes I et II, aux Editions Micro Application.

Comment s'opèrent l'appel et l'intégration des routines DOS de l'Amiga? Ces procédures à l'inverse de ce qu'il en est avec l'Amiga BASIC sont d'une simplicité enfantine. Leur mise à disposition est analogue à celle des fonctions GFA BASIC. Leur appel ne requiert la moindre cérémonie d'ouverture.

L'éditeur vérifie d'ailleurs la justesse de leur syntaxe lors de leur insertion. Néanmoins il ne s'occupe pas des erreurs système dues à des données dont la sémantique est incorrecte. Ces dernières amènent soit un message d'erreur Amiga DOS ou soit dans le pire des cas, l'intervention de Guru surnommé la Méditation à laquelle seule l'initialisation de la machine peut remédier. Nous vous conseillons d'être fort prudent lors de l'emploi des routines DOS et de lire attentivement les 31 prochaines descriptions des routines disponibles.

Note: Utilisez la commande VOID (ou) pour l'appel de toutes les fonctions qui ne retournent pas la moindre valeur.

Les fonctions usuelles d'entrées et de sorties

Open — Ouverture d'un fichier

Filehandle = Open(Nom, Mode)

Ouvre le fichier dont le nom est spécifié dans le paramètre Nom.

Attention : La chaîne du nom doit être ponctuée par un octet nul ! Cette dite convention C à respecter par les chaînes, vaut pour pratiquement toutes les transmissions de données aux fonctions du système d'exploitation.

Mode peut revêtir la valeur 1005 afin d'ouvrir un fichier à des fins d'écriture et de lecture, ou celle 1006 en but de la création d'un fichier et de l'écriture de données en son sein (Attention: s'il existe déjà un fichier dont le nom est identique, sa suppression en résulte), ou celle 1004. Ce dernier mode est analogue au mode 1005, toutefois le fichier ne peut plus être ouvert par d'autres programmes jusqu'à sa fermeture, l'accès y est en somme exclusif.

Filehandle retourne le pointeur de la structure de gestion du fichier spécifié. Elle contient d'importantes informations sur le fichier, dont la signification ne concerne en règle ordinaire que la gestion interne de l'AmigaDOS. Si le fichier n'a pu être ouvert, Filehandle contient une valeur nulle.

La fonction du Filehandle au sein des fonctions suivantes est analogue à celle du numéro de fichier observée dans le GFA Basic. Le Filehandle sert à la distinction et à l'identification d'un fichier.

Close — Clôt un fichier

Statut = Close (Filehandle)

Clôt une fichier ouvert. Filehandle contient la position de la structure du gestionnaire de fichiers maintenue pointée depuis l'ouverture du fichier. Statut indique si le fichier a été correctement fermé (Code d'erreur supérieur à 200).

Read — Lit les données

Nombre = Read(Filehandle, Tampon, Longueur)

Read lit le nombre d'octets spécifié dans le fichier désigné par Filehandle et les entrepose en mémoire vive à partir de l'adresse Tampon. Nombre retourne le nombre d'octets effectivement lus. Le renvoi de la valeur -1 signifie le surgissement d'une erreur lors de la lecture.

Write — Ecrit les données

Nombre = Write(Filehandle, Tampon, Longueur)

Write écrit le nombre d'octets spécifié en Longueur dans le fichier notifié en Filehandle, à partir de l'adresse Tampon. Nombre retourne le nombre d'octets effectivement écrits. Le renvoi d'une valeur -1 signifie le surgissement d'une erreur lors de l'écriture.

Seek — Positionne le pointeur de fichier

Position = Seek(Filehandle, Intervalle, Mode)

Seek positionne, ou selon déplace, le pointeur interne du fichier précisé dans Filehandle. Ce pointeur pointe toujours la position du prochain octet à lire ou à écrire.

Mode détermine le sens du décompte. Si Mode est égal à -1, la valeur contenue dans Inervalle en réfère au début ; si mode égale +1 , à la fin du fichier et si mode égale 0, à la position en cours du pointeur. Retrait contient, bien entendu, la valeur du déplacement désiré en nombre d'octets.

Position retourne, suite à l'exécution de la fonction, la position en cours du pointeur. La conservation de la position momentanée du pointeur s'obtient par les indications de 0 dans Mode (=déplacement sur la position en cours) et de 0 octet dans Retrait.

Input — Retourne le canal d'entrée en cours

Position = Seek(Filehandle,0,0)

Filehandle = Input()

Délivre le Filehandle du canal d'entrée en cours.

Output — Retourne le canal de sortie en cours

Filehandle = Output()

Délivre le Filehandle du canal de sortie en cours

WaitForChar — Attend la réception d'un caractère

Statut = WaitForChar(Filehandle, Timeout)

Attend, le délai durant spécifié dans Timeout en termes de millisecondes, la réception d'un caractère du fichier désigné par Filehandle.

Si aucun caractère n'est réceptionné durant le temps déterminé, Statut contient la valeur 0, sinon -1.

Le signe réceptionné est lisible à l'aide de Read.

IsInteractive — Retourne le type de canal

Statut = IsInteractive (Filehandle)

IsInteractive retourne en Statut la valeur -1 si le fichier désigné par Filehandle est de type interactif (les entrées et les sorties de données y étant possibles), en cas contraire 0.

IoErr — Retourne les erreurs d'entrées et sorties

Erreur = IoErr()

En Erreur est retourné le numéro de la dernière erreur survenue lors du traitement de l'une des précédentes fonctions.

La plupart des fonctions assurent d'elles-mêmes la communication de l'erreur survenue, en retournant la valeur nulle (0). IoErr assure la connaissance précise de l'origine de l'erreur.

La gestion des disquettes et des fichiers

Lock — Détermine le statut d'accès

Verrou = Lock(Nom, Mode)

Lock cherche le fichier ou le sous-répertoire spécifié par Nom et y crée un verrou dont l'adresse est retournée dans la variable de même nom. Ce verrou, à vrai dire son adresse, est nécessité par la suite par les fonctions suivantes pour accèder au fichier ou répertoire ainsi verrouillé.

Le mode d'accès au fichier est déterminé dans le paramètre Mode. La notification de -2 en Mode autorise la lecture des données du fichier par plusieurs programmes ; l'indication de -1 spécifie que seul le programme en cours peut écrire dans le fichier.

DupLock — Copie du verrou

Verrou2 = Duplock(Verrou1)

Duplock copie le verrou dont l'adresse est spécifiée et délivre dans Verrou2 l'adresse de la copie. Seule la copie de verrous crées en mode -2, autorisant la lecture des données, est possible.

Unlock — Supprime un verrou

Void Unlock(Verrou)

Unlock supprime un verrou créé à l'aide de Lock ou de Duplock dans la mémoire utilisateur. A cette fin, l'adresse du verrou est portée dans le paramètre de même nom.

Ceux qui nous ont suivis jusqu'à ce point, s'apercevront bientôt que leurs efforts n'ont pas été vains. Bien que les précédentes fonctions n'intéressent effectivement que les spécialistes, les suivantes apportent des informations essentielles dont l'obtention à l'aide de l'interpréteur est impossible.

Info — Délivre des informations sur la disquette

Statut = Info(Verrou, Infoadresse)

Info crée dans la zone mémoire débutant à Infoadresse, un bloc d'informations concernant la disquette spécifiée dans Verrou. Verrou doit nécessairement pointer le nom d'une disquette, d'un fichier ou d'un sous-répertoire d'une disquette. L'identificateur de lecteur peut remplacer le nom de la disquette (voir l'exemple).

Attention : Une adresse divisible par quatre est requise pour Infoadresse. Pour la réservation de l'emplacement mémoire pour la table des informations, l'utilisation de la fonction MALLOC est de ce fait, conseillée sinon obligatoire. La zone mémoire réservée par MALLOC débute toujours et par principe à une adresse divisible par huit. Le bloc d'informations créé par Info a la structure suivante:

Octet	Signification
0 - 3	nombre des erreurs disquette
4 - 7	unité de disquette installée
8 - 11	statut de la disquette : 80 la disquette est protégée en écriture,82 l'écriture y est possible
12 - 15	nombre total des unités d'allocations
16 - 19	nombre des unités d'allocation utilisées
20 - 23	nombre des octets d'une unité d'allocation
24 - 27	le type de la disquette. -1 : disquette non-insérée,BAD:disquette illisible,DOS:disquette DOS
28 - 31	pointeur du nom de disquette
32	disquette active (<>0), inactive (0)

Pour accéder aux informations isolées, utilisez soit l'instruction PEEK et celles apparentées, soit l'une des instructions d'accès à la mémoire utilisateur exposées dans le chapitre 12 (CARD{}, etc.). La plus

intéressante est certainement celle qui notifie la protection ou non en écriture de la disquette insérée dans un des lecteurs. Certes l'AmigaDOS réagit avec un System Requester lors d'un accès en écriture sur une disquette protégée. Mais la communication du statut de la disquette en préalable à son accès en écriture permet une prévention personnelle. A cette fin, j'ai écrit la fonction suivante :

```
FUNCTION testdisquette(drive$)     ! drive$ contient
 ' l'identificateur du lecteur
  LOCAL drive%,lock%,adr%,statut% ! variables locales
  drive$=drive$+CHR$(0)     ! chaîne close par un octet nul
  drive%=VARPTR(drive$)     ! pointeur sur le début de la chaîne
  lock%=Lock(drive%,-2)     ! lock% contient l'adresse du verrou
  adr%=MALLOC(36,1)         ! 36 octets pour le bloc d'informations
  VOID Info(lock%,adr%)     ! appel de la fonction Info
  statut%=PEEK(adr%+11)     ! cherche le statut de la disquette
  VOID MFREE(adr%,36)       ! libère la zone mémoire réservée
  IF statut%=82 THEN        ! disquette non protégée?
    RETURN TRUE             ! oui, retourne -1
  ELSE                      ! la disquette est protégée
    RETURN FALSE            ! retourne 0
  ENDIF
ENDFUNC
```

Comment s'utilise Testdisquette ? Admettons que vous désirez vous assurer de la possibilité d'un accès en écriture sur la disquette insérée dans le lecteur DF0. Cela pourrait s'illustrer de la façon suivante :

```
IF @Testdisquette("DF0:")=FALSE THEN
  ALERT 0,"Veuillez déverrouiller la disquette ",1,"OK",d%
ENDIF
CreateDir                              ! créé un nouveau répertoire
Verrou = CreateDir (Nom)
```

Crée dans le répertoire en cours, le sous-répertoire dont le nom est porté dans Nom. Un verrou dont l'adresse est retournée dans Verrou, lui est affecté de manière analogue à celle de l'utilisation de la fonction Lock.

CurrentDir — Sélectionne un répertoire

Verrou_ancien = CurrentDir(Verrou)

Assure le changement de répertoire courant indiqué par Verrou. Verrou_ancien retourne l'adresse du répertoire précédent.

Parentdir — Retourne le répertoire supérieur

Nouveau_verrou = ParentDir(Verrou)

Délivre en Nouveau_verrou l'adresse du répertoire supérieur à celui dont l'adresse est spécifiée en Verrou.

Examine — Livre des informations sur les fichiers

Statut = Examine(Verrou, Infoadresse)

Examine crée dans la mémoire utilisateur à partir d'Infoadresse, un bloc d'informations sur le fichier ou le répertoire indiqué par Verrou.

Le bloc d'informations revêt la structure suivante :

```
Octet          Signification
-------------------------------------------------------------------

  0 -   3      numéro de disquette
  4 -   7      type d'entrée (>0: sous-répertoire, sinon fichier)
  8 - 115     nom d'entrée (seuls 30 octets sont utilisables)
116 - 119     type d'accès (Cf. SetProtection)
bit 0: fichier non effaçable
bit 1: fichier non exécutable
bit 2: fichier protégé en écriture
bit 3: fichier protégé en lecture
120 - 123     type d'entrée (Cf. ci-dessus)
124 - 127     longueur du fichier en octets
128 - 131     nombres des unités d'allocation utilisées
132 - 143     date de création
144 - 259     commentaire (seuls 80 octets sont utilisables)
```

Pour lire ces informations, procédez comme suit :

```
(dnom$ contient un nom de fichier ou de répertoire)
dnom$=dnom$+CHR$(0)            ! la chaîne doit être close par 0
dnom%=VARPTR(dnom$)            ! pointeur en début de chaîne
lock%=Lock(dnom%,-2)           ! affecte un verrou
Infoadr%=MALLOC(260,1)         ! réserve 260 octets pour le
                               ! bloc d'informations
Void Examine(lock%,Infoadr%)   ! appel de la fonction
```

Il vous est maintenant possible d'accéder aux informations isolées :

```
longueur%={infoadr%+124}        ! longueur du fichier
com$=CHAR{infoadr%+144}         ! commentaire
et ainsi de suite...
```

Veillez à ne pas oublier en conclusion de libérer la zone mémoire allouée par MALLOC avec :

```
VOID MFREE(infoadr%,260)
```

Examine communique aussi le nom de la disquette. Cela peut se révéler fort utile si vous désirez vous assurer que l'insertion de la disquette a été effectuée correctement dans le lecteur. Pour cela, il suffit d'indiquer simplement dans dnom$ l'identificateur du lecteur : par exemple DF0:. La ligne d'instruction suivante répond précisément à cela :

```
disk$=CHAR{infoadr%+8}
```

ExNext — Retourne la prochaine entrée de répertoire

Statut = ExNext(Verrou, Infoadresse)

ExNext communique la prochaine entrée du répertoire spécifié dans Verrou et dispose les informations concernant cette entrée dans le bloc d'informations dont l'adresse est précisée dans Infoadresse (Cf. Examine). ExNext permet ainsi la lecture terme à terme du répertoire d'une disquette en son entier. Après la dernière entrée, Statut retourne une valeur nulle.

DeleteFile — Supprime un fichier

Statut = DeleteFile(Nom)

Supprime un fichier ou un sous-répertoire (Nom contient l'adresse du nom du fichier).

Attention : Lors de la suppression d'un sous-répertoire, veillez à ce qu'il ne contienne plus aucun fichier. Sinon, il convient de les supprimer l'un après l'autre avec DeleteFile.

Rename — Renomme un fichier

Statut = Rename (Ancien_nom, Appellation_nouvelle)

Renomme le fichier ou le répertoire noté avec la nouvelle appellation.

SetProtection — Affecte une protection à un fichier

Statut = SetProtection (Nom, Mode)

Modifie le statut d'accès du fichier ou du répertoire nommé. En Mode, seuls les quatre bits de poids faible (8 bits en 1.3) revêtent une signification :

```
bit 0: fichier non effaçable
bit 1: fichier non exécutable
bit 2: fichier protégé en écriture
bit 3: fichier protégé en lecture
bit 4: fichier archivé
bit 5: fichier pure
bit 6: fichier script
bit 7: fichier caché
```

Les bits 4 à 7 se testent de manière inverse par rapport aux bist 0 à 3.

Exemple :

```
dnom$="Nom_de_fichier"+CHR$(0)     ! un nom quelconque
dnom=VARPTR(dnom$)                 ! pointe dessus
mode=2^0 OR 2^2                    ! fichier non protégé en
                                   ! lecture ou en écriture
Statut=SetProtection(dnom,mode)    !
```

Aucune erreur n'apparaissant lors du traitement, "Statut" contient la valeur TRUE (-1).

SetComment — Adjoint un commentaire au fichier

Statut=SetComment(Nom, Commentaire)

Adjoint un commentaire de 80 caractères maximum au fichier ou au répertoire nommé.

Exemple :

```
dnom$="Nom_de_fichier"+CHR$(0)      ! un quelconque nom
dnom=VARPTR(dnom$)                  ! pointe le pointeur dessus
com$="Commentaire"+CHR$(0)          ! un commentaire quelconque de 80 caractères
maximum
com=VARPTR(com$)                    ! pointe dessus
Statut=SetComment(dnom,com)         ! place le commentaire
```

En l'absence de problème, Statut contient la valeur TRUE (-1). La lecture du commentaire adjoint à un fichier, s'obtient par la commande CLI : List, List devant être appelé avec l'ordre EXEC sous GFA BASIC. A titre d'exemple, EXEC "list DF0:",0,0 liste tous les fichiers de la disquette insérée dans le lecteur df0:, leur éventuel commentaire compris. Cette édition s'effectue dans une fenêtre CLI, et non pas dans la fenêtre d'édition du GFA BASIC.

Organisation de processus

CreateProc — Crée un nouveau processus

Procès = CreateProc(Nom, Priorité, Segment, Pile)

Crée sous le nom indiqué par Nom, une nouvelle structure de procès.

DeviceProc — Retourne le processus d'Entrées/Sorties utilisé

Processus=DeviceProc(Nom)

Retourne en Processus le numéro d'identification du processus utilisant le canal d'Entrées/Sorties sélectionné dans Nom.

Delay — Suspend le processus en cours

VOID Delay(Temps)

Suspend le processus en cours, le temps durant indiqué en cinquantième de seconde en Temps.

Exit — Clôt le processus

VOID Exit(paramètre)

Clôt le processus en cours (ou selon, l'exécution du programme) et libère les zones mémoires mobilisées par celui-ci.

LoadSeg — Charge un fichier de programme

Segment = LoadSeg(Nom)

Charge le fichier de programme nommé, dans la mémoire utilisateur. En Segment, est retournée l'adresse du premier module du programme.

UnLoadSeg — Efface un fichier de programme chargé

VOID UnloadSeg(Segment)

Supprime un fichier de programme chargé avec LoadSeg et libère la place mémoire occupée par celui-ci. Segment contient l'adresse du premier module du programme (Cf. LoadSeg).

DateStamp — Communique la date et l'heure

VOID DateStamp(Adresse)

Retourne la date système en cours en format AmigaDOS, en trois mots longs à l'Adresse spécifiée.

Execute — Exécute une commande CLI

Statut = Execute(Commande, Input, Output)

Exécute une commande quelconque CLI. En Commande, est spécifié le pointeur du texte de la commande. Input et Output contiennent le filehandle du canal d'Entrées/Sorties désiré.

Note: L'appel direct des fonctions DOS GetPacket et QueuePacket n'est pas à ce jour assuré à partir du GFA BASIC.

5.8 Instructions d'impression

HARDCOPY [H] (Partie) — Imprime l'image écran

HARDCOPY
HARDCOPY Rastport, Colormap, Mode, Retrait_G, Retrait_D, Largeur, Hauteur, Colonnes, Lignes, Flags
HARDCOPY Flags

HARDCOPY assure l'impression du contenu actuel de l'écran. Les sélections d'impression portées dans Preferences sont celles effectives pendant son exécution. Comme les entrées de HARDCOPY ne sont pas définissables à partir de la syntaxe de la commande, l'ouverture d'au moins une fenêtre à l'écran, est impérativement requise.

Les seconde et troisième versions syntaxiques de la commande permettent d'instruire des copies d'écran selon des règles et des manières fort diverses. Leurs paramètres se détaillent comme suit :

Rastport : Adresse du Rastport devant effectuer la copie d'écran.

Colormap : Adresse de la Colormap, contenant les couleurs de l'écran.

Mode : Contient les modes de résolution, sélectionnés dans la structure du Viewport.

Retrait_G : Retrait sur le bord gauche du Rastport.

Retrait_G : Retrait sur le bord droit du Rastport.

Largeur : Largeur de l'image écran à imprimer

Hauteur : Hauteur de l'image écran à imprimer

Colonnes : Nombre de colonnes à imprimer

Lignes : Nombre de lignes à imprimer

Flags : Seuls les 12 derniers bits de poids faible importent. Chacun de ces bits a une significationpropre.

Octet	Signification (s'il est activé)
0	L'unité de la valeur de Colonnes est le 1/1000 de pouce.
1	L'unité de la valeur de Lignes est le 1/1000 de pouce.
2	La copie d'écran s'effectue en largeur maximale.
3	La copie d'écran s'effectue en hauteur maximale.
4	La valeur de Colonnes divise la largeur maximale.
5	La valeur de Lignes divise la hauteur maximale.
6	opérée la copie d'écran. 001 règle l'impression d'écran.
7	Les marges respectées sur la papier doivent correspondre à celles observées sur l'écran.
8-10	Ces bits servent à déterminer en quelle résolution est opérée la copie d'écran. 001 règle l'impression en faible résolution et 111 en haute résolution.
11	Supprime le saut de page en fin d'impression et permet l'impression en continu de plusieurs copies d'écran.

Note: Les descriptions de ce que sont un Rastport, un Viewport ou la Colormap, de la communication de leur adresse et des valeurs qui leur sont assignables, sont exposées dans le chapitre réservé aux applications graphiques.

Exemple :

```
OPENW 5                  ! ouvre une petite fenêtre BASIC centrée
GRAPHMODE 1              ! sélectionne le mode graphique 1
ELLIPSE ,150,160,50,40 ! dessine une ellipse
ELLIPSE ,150,160,80,25 ! dessine une seconde ellipse
HARDCOPY                 ! imprime l'écran du Workbench
CLOSEW 5                 ! ferme la fenêtre
```

Le programme réalise un petit dessin sur l'écran et copie le contenu de l'écran du Workbench sur le papier.

LLIST [LL] — Imprime le listing d'un programme

LLIST

Cette commande imprime le listing du programme en cours. Elle est analogue à l'option Llist de l'éditeur. Son exécution peut être interrompue par l'emploi de la fonction d'interruption du GFA Basic: <Control/Shift/Alternate>. L'interpréteur est alors de nouveau disponible pour tout autre traitement et seul est imprimé le contenu du buffer (le tampon de mémoire) de l'imprimante.

LPOS () — Communique la position de la tête d'impression

Var=LPOS(Dummy)

Cette fonction retourne la position virtuelle de la tête d'impression dans le tampon de ligne de l'imprimante, dont la valeur maximale est de 255 caractères. Puisque celle-ci n'est pas nécessairement identique à la position physique de la tête d'impression en cours (celle qui se manifeste concrètement), il peut être fort intéressant en certaines circonstances de connaître sa position actuelle dans le tampon de ligne. Après un

retour chariot (Carriage Return) le décompte de LPRINT débute de nouveau à zéro. Un argument muet : une quelconque valeur, mis entre parenthèses est adjoint à la commande.

Exemple :

```
LPRINT "Position virtuelle de la tête d'impression"
FOR I=1 TO 150
  LPRINT LPOS(0)'
NEXT I
A=LPOS(0)
PRINT A
```

LPRINT (LPR) — Imprime des données

LPRINT [,'] "Texte" [[;']Var1[;'] Expr...]

Le traitement opéré par LPRINT est fondamentalement identique à celui exécuté par PRINT. Leur syntaxe, mis à part le paramètre AT(Xposition,Yposition), est analogue. La seule différence est que LPRINT n'édite pas les caractères, les chaînes et les valeurs à l'écran, mais les dirige sur l'imprimante. Malheureusement cela ne va pas de soit, l'emploi de code de contrôles, de codes Escape est requis pour instruire les fonctions spécifiques d'impressions. Certes la plupart des imprimantes respectent l'émulation EPSON, malgré cela il arrive souvent que le résultat obtenu à l'impression démente celui escompté, par exemple que les types de caractères ou les retraits désirés soient absents sur la feuille imprimée.

En comparaison des autres ordinateurs, les modalités fonctionnelles de l'impression propres à l'Amiga sont bien plus simples, puisqu'il dispose d'une table de codes de contrôle standard. Ces codes de contrôle d'impression du moins s'ils sont transmis en utilisant le périphérique logique : PRT, ce qui est le cas avec LPRINT, sont convertis en considération de l'interface sélectionnée dans Preferences, dans les codes de contrôles spécifiques de l'imprimante connectée. L'avantage de cela est aisément compréhensible. Tant que lors de la programmation, vous ne vous servez que des codes de contrôle standard, peu importe quelle imprimante est connectée à votre Amiga. L'essentiel est que le mode de liaison de votre imprimante soit correctement choisi.

Mais vous devriez toujours veiller à ce que l'imprimante connectée soit toujours en mesure de traiter les codes de contrôle utilisés. Rien ne sert d'employer par exemple, le code de contrôle du jeu de caractères japonais si votre imprimante ne dispose du jeu correspondant. Veuillez vous assurer au préalable des possibilités propres de votre imprimante en consultant son manuel d'utilisation.

Quels sont ces codes de contrôle? La table suivante expose toutes les séquences de contrôles disponibles :

```
CHR$(27)+"c"                initialise l'imprimante
CHR$(27)+"#1"               débranche tous les modes particuliers
CHR$(27)+"D"                saut de ligne
CHR$(27)+"E"                saut de ligne + retour chariot
CHR$(27)+"[0m"              caractère normal
CHR$(27)+"[1m"              Ecriture en gras
CHR$(27)+"[22m"             Désactiver gras
CHR$(27)+"[3m"              Ecriture en italique
CHR$(27)+"[23m"             Désactiver italique
CHR$(27)+"[4m"              soulignement
CHR$(27)+"[24m"             suspension du soulignement
CHR$(27)+"["+x$+"m"         couleur de premier plan (x$ entre  "30" et "39"),
                            d'arrière plan (x$entre "40" et "49"
CHR$(27)+"[0w"              corps (taille) standard
CHR$(27)+"[2w"              sélection de la police Elite
CHR$(27)+"[1w"              suspension de la police Elite
CHR$(27)+"[4w"              caractère condensé
CHR$(27)+"[3w"              suspension du style condensé
CHR$(27)+"[6w"              caractère élargi
CHR$(27)+"[5w"              suspension du style élargi
CHR$(27)+"[2"+CHR$(34)+"z"  sélection de la police NLQ
CHR$(27)+"[1"+CHR$(34)+"z"  suspension de la police NLQ
CHR$(27)+"[4"+CHR$(34)+"z"  sélection du style double frappe
CHR$(27)+"[5"+CHR$(34)+"z"  suspension du style double frappe
CHR$(27)+"[6"+CHR$(34)+"z"  sélection du style ombré
CHR$(27)+"[5"+CHR$(34)+"z"  suspension du style ombré
CHR$(27)+"[2v"              sélection du style surligné
CHR$(27)+"[1v"              suspension du style surligné
CHR$(27)+"[4v"              sélection du style souligné
CHR$(27)+"[3v"              suspension du style souligné
CHR$(27)+"L"                mise en indice
CHR$(27)+"[2p"              activation de la police proportionnelle
CHR$(27)+"[1p"              suspension de la police proportionnelle
CHR$(27)+"[0p"              supprimer l'espace proportionnel
CHR$(27)+"["+x$+"E"         espace proportionnel = x$
CHR$(27)+"[5F"              alignement gauche
CHR$(27)+"[7F"              alignement droit
CHR$(27)+"[6F"              justification
CHR$(27)+"[0F"              supprimer la justification
CHR$(27)+"[3F"              ajuster la largeur des caractères
CHR$(27)+"[1F"              centrage
```

```
CHR$(27)+"[0z"              interligne d'1/8 de pouce
CHR$(27)+"[1z"              interligne d'1/6 de pouce
CHR$(27)+"["+x$+"t"         longueur de page x$
CHR$(27)+"["+x$+"q"         saut de page x$
CHR$(27)+"[0q"              supprimer saut de page
CHR$(27)+"[B"               jeu de caractères américain
CHR$(27)+"[R"               jeu de caractères français
CHR$(27)+"[K"               jeu de caractères allemand
CHR$(27)+"[A"               jeu de caractères anglais
CHR$(27)+"[E"               jeu de caractères danois (N1)
CHR$(27)+"[H"               jeu de caractères suédois
CHR$(27)+"[Y"               jeu de caractères italien
CHR$(27)+"[Z"               jeu de caractères espagnol
CHR$(27)+"[J"               jeu de caractères japonais
CHR$(27)+"[6"               jeu de caractères norvégien
CHR$(27)+"[C"               jeu de caractères danois (N2)
CHR$(27)+"#9"               marge de gauche
CHR$(27)+"#0"               marge de droite
CHR$(27)+"#8"               marge du haut
CHR$(27)+"#2"               marge du bas
CHR$(27)+"#3"               supprimer marge
CHR$(27)+"["+x$+";"+y$+"r"  titre courant du haut à x$ lignes titre courant du bas
                            à y$ lignes
CHR$(27)+"["+x$+";"+y$+"s"  marges de droite x$ et de gauche
                            y$
CHR$(27)+"H"                placer une tabulation horizontale
CHR$(27)+"J"                placer une tabulation verticale
CHR$(27)+"0g"               effacer une tabulation horizontale
CHR$(27)+"3g"               supprimer toutes les tabulations horizontales
CHR$(27)+"1g"               supprimer une tabulation verticale
CHR$(27)+"4g"               supprimer toutes les tabulations verticales
CHR$(27)+"#4"               supprimer toutes les tabulations
CHR$(27)+"#5"               placer les tabulations standards
```

Lors de l'insertion des codes de contrôle, veillez à les introduire selon leurs séquences de touches (par exemple x$="123") et non pas avec leur code ASCII comme s'il s'agissait de caractères !

Certes la mémorisation des séquences de contrôle par exemple celle CHR$(27)"[1m" de la mise en gras, n'est pas chose évidente. Et la consultation répétée de leur table risque d'être fort lassante. Rien ne sert de se plaindre plus longtemps, d'autant que la programmation de fonctions personnelles résolvant ce problème est, comme le montre le prochain exemple, chose fort aisée :

```
DEFFN gras$ = CHR$(27)+"[1m"
```

Si maintenant vous désirez imprimer en gras, il ne reste plus qu'à insérer @gras$. Cela est bien plus facile à retenir, n'est-ce pas? Nous vous conseillons de noter toutes les séquences de contrôle dont l'usage

vous est le plus fréquent, et de les affecter chacune à une fonction dont le nom est d'une mémorisation aisée. Ensuite il ne reste plus qu'à les sauvegarder toutes dans un même programme (pourquoi pas sous le nom de CODES_IMPRIMANTE.LST?) sur la disquette. Cette chose faite, l'accession de vos programmes sur l'imprimante ne requiert plus que les insertions en ceux-ci, du nom choisi et de CODES_IMPRIMANTES.LST à l'aide de la fonction Merge en fin.

Une telle table des séquences de contrôle standard est certes fort avantageuse, mais se montre insuffisante lorsque votre imprimante dispose de possibilités qui n'y sont pas recensées. De nombreuses imprimantes proposent le style Outline. Que faire en un pareil cas ? Rien ne sert à chercher dans la table, vous ne feriez que perdre votre temps.

L'Amiga dispose d'une autre interface: celle du périphérique logique : PAR:. L'utilité de ce dernier n'est guère différente de celle de PRT:. Son accession s'opère en mode de liaison parallèle (ou nommé mode Centronics). A l'inverse de PRT:, nulle conversion des séquences de contrôle n'y est effectuée, ce qui veut dire qu'elles sont transmises telles quelles à l'imprimante connectée, en considération de la sélection d'imprimante notifiée dans Preferences. Que faire en pareil cas ? D'abord est requise l'ouverture d'un fichier logique affecté à PAR:, à l'aide de :

```
OPEN "O",#1,"PAR:"
```

puis la transmission de la séquence désirée à l'imprimante avec :

```
PRINT #1,CHR$(27)...
```

Les prochaines données (le texte à imprimer) sont alors imprimées à l'aide de :

```
PRINT #1,....
```

Les mêmes règles de syntaxes que celles observées avec LPRINT sont à respecter. Enfin n'oubliez pas de clore le fichier:

```
CLOSE #1
```

Si l'exotisme de votre imprimante induit sa connexion sur le port série de votre Amiga, l'utilisation de l'interface série, du périphérique logique SER: est requise. La manière de procéder est identique à celle observée pour PAR:. En premier lieu, il convient d'ouvrir un fichier logique à

l'aide de : OPEN "O",#1,"SER:", puis de transmettre le code de contrôle et d'autres données avec: PRINT, et enfin clore le fichier avec: CLOSE #1. Mais n'oubliez jamais que la liste des codes citée plus haut ne fonctionne qu'avec PRT:. Vous devrez donc vous munir du manuel de votre imprimante et d'une bonne dose de patience ...

5.9 Production de sons

SOUND [SO] — Produit des sonorités musicales

SOUND Fréquence, Durée[, Volume] [, Canal]

L'instruction SOUND produit un son dont le choix de la fréquence peut osciller entre 20 Hertz et 15000 Hertz. Le paramètre Durée est une variable numérique de 16 bits, qui indique le nombre de répétition de la courbe sinusoïdale de l'onde de son (Cf. WAVE).

L'indication optionnelle du volume peut s'échelonner de 0 (volume éteint) à 255 (pleine puissance). L'échelle des valeurs de volume est linéaire ; la valeur 127 amène de ce fait une puissance de volume moyenne. Cette dernière est d'ailleurs celle par défaut.

Dans Canal, est déterminé lequel des quatre canaux sonores de l'Amiga (numérotés de 0 à 3) est à utiliser. La valeur du canal sonore par défaut est 0. Chaque canal est accessible isolément, autrement dit l'émission simultanée de jusqu'à quatre sonorités est possible.

Lors d'une émission stéréophonique, les canaux 0 et 3 représente la sortie sonore gauche et ceux 1 et 2 celle de droite.

A côté de la production de simples sonorités, la reproduction des notes ou de pièces musicales entières est de loin bien plus intéressante. Acette fin, il convient de connaître les fréquences de son affectées aux notes isolées.

Note		Fréquence (Hz)
c	do	261.6
c#	do#	277.2
d	ré	293.7
d#	ré#	311.2
e	mi	329.7
f	fa	349.3
f#	fa#	370.0
g	sol	392,0
g#	sol#	415.3
a	la	440.0
a#	la#	466.2
h	si	493.9

Cette table expose les fréquences de la seconde octave. Le calcul des fréquences des autres octaves est, comme cela est concevable, aisé puisqu'avec chaque octave, la fréquence de la note double. La note la dans la troisième octave a une fréquence de 2*440 = 880 Hertz, dans la quatrième octave, celle 2*440 = 880 Hertz, et ainsi de suite. La consultation de la table suivante est proposée à tous ceux qui sont rebutés par ces calculs. Elle comprend les notes les plus usitées et leurs fréquences correspondantes en Hertz :

Octave:		1	2	3	4	5
c	do	130.8	261.6	523.2	1046.4	2092.8
c#	do#	138.6	277.2	554.4	1108.8	2217.6
d	ré	146.9	293.7	587.4	1174.8	2349.6
d#	ré#	155.6	311.2	622.4	1244.8	2489.6
e	mi	164.9	329.7	659.4	1318.8	2637.6
f	fa	174.7	349.3	698.6	1397.2	2794.4
f#	fa#	185.0	370.0	740.0	1480.0	2960.0
g	sol	196.0	392,0	784.0	1568.0	3136.0
g#	sol#	207.7	415.3	830.6	1661.2	3322.4
a	la	220.0	440.0	880.0	1760.0	3520.0
a#	la#	233.1	466.2	932.4	1864.8	3729.6
h	si	247.0	493.9	987.8	1975.6	3951.2

WAVE [WA] — Précise la forme de l'onde

WAVE Canal, forme()

Cette instruction permet de déterminer pour chacun des quatre canaux sonores (de 0 à 3) de l'Amiga, une forme d'onde. Celle-ci précise le mode d'émission du son au travers du canal sélectionné. Les valeurs de la forme de l'onde correspondent à une matrice de 256 éléments variant entre -128 et +127. De ce fait, il convient d'utiliser de préférence des variables matricielles entières d'un octet dont le préfixe est "".

Lors d'une utilisation isolée de l'instruction SOUND, sans insertion préalable de celle de WAVE, une courbe sinusoïdale est affectée à l'onde émise dans le canal sélectionné. Cette forme est utilisable avec WAVE à l'aide du mot-clé SIN(): WAVE Canal, SIN().

SAY — Emet des phonèmes synthétisés

SAY Texte$ [Mode%()]

L'instruction SAY sert à traduire des codes de phonèmes précédemment obtenus à l'aide de celle TRANSLATE$. Les codes de phonèmes sont indiqués à cette fin dans la chaîne Texte$. La combinaison de ces deux instructions est d'ailleurs offerte : SAY TRANSLATE$("BONJOUR")

Les sept valeurs composant la matrice Mode%, permettent de modifier la "voix" de l'Amiga.

Mode%(0) Contient la fréquence, donc de ce fait la tonalité de la voix. Les valeurs propres de ce mode s'échelonnent de 0 (tonalité basse) à 320 (tonalité haute). La valeur par défaut est 110.

Mode%(1) Détermine l'inflexion de la "voix". Les choix expressive (0) et monocorde (1) sont possibles ; 0 est la valeur par défaut.

Mode%(2) Détermine la vitesse du débit de l'émission "vocale" en termes de mots par minute (de 40 à 400) ; 150 est la valeur par défaut.

Mode%(3) Spécifie le genre de la "voix": masculine (0) ou féminine (1); la genre par défaut est masculin (0).

Mode%(4) Contient la fréquence d'échantillonnage, qui influe essentiellement sur la hauteur de la tonalité. Les valeurs possibles sont comprises entre 5000 (tonalité basse) et 28000 (tonalité haute). La valeur présélectionnée est celle 22000.

Mode%(5) Définit le volume de la voix. Les valeurs admises vont de 0 (absence de volume) jusqu'à 64 (puissance maximale). La valeur par défaut est 64.

Mode%(6) Détermine lequel des quatre canaux (de 0 à 3) de l'Amiga est à utiliser. Toutes les combinaisons de canaux peuvent être employées :

```
0 Canal 0
          1 Canal 1
          2 Canal 2
          3 Canal 3
          4 Canaux 0 et 1
          5 Canaux 0 et 2
          6 Canaux 3 et 1
          7 Canaux 3 et 2
          8 Canaux 0 ou/et 3
          9 Canaux 1 ou/et 2
         10 n'importe quelle paire de canal libre (valeur
            par défaut)
         11 n'importe quel canal libre
```

La multitude des paramètres est d'un premier coup d'oeil impressionnante. Afin de faciliter leur manipulation, nous vous conseillons d'assigner une ligne d'instruction SAY complète dans une ligne DATA et d'entreprendre en cette dernière toutes les prochaines modifications de paramètres :

```
FOR L%=0 TO 9
   READ Mode%(L%)    ! boucle de lecture
NEXT L%
! ligne obtenue
DATA 110,0,150,0,22000,64,10
```

Pour féminiser la voix, il suffit de modifier le 0 en quatrième position en un 1.

Attention : SAY nécessite lors de son exécution, le narrator.device disponible dans le répertoire DEVS: de la disquette système. Lors de l'appel de SAY, l'insertion de la disquette système dans un des lecteurs connectés est requise, sinon une alerte générale est déclenchée jusqu'à obtention de la nourriture demandée.

TRANSLATE$() — Traduit les textes en phonèmes

Chaîne_de_texte=TRANSLATE$("Texte")

TRANSLATE traduit des expressions écrites de la langue usuelle en code de phonèmes "compréhensibles" par l'instruction SAY. Toutefois leur combinaison souffre d'un léger défaut, puisqu'elle n'est qu'en mesure de traiter des expressions anglaises, autrement dit seule une "vocalisation" anglaise (pour le moment ...) y est assurée. Les textes français sont parfaitement inaudibles.

Néanmoins il est possible d'écrire les phonèmes de langue française, selon les règles phonétiques admises dans le monde anglo-saxon, de remplacer par exemple le "i" en français par le "ee", etc... Sinon, seuls des essais d'approximation répétés vous permettront d'assurer une émission audible.

Chapitre 6

Les structures de programme

6.1 Les constructions de boucle

DO... LOOP (DO ... L)

Boucle perpétuelle DO [WHILE condition] [UNTIL condition] ...boucle à double condition LOOP [WHILE condition] [UNTIL condition]

DO
... bloc d'instructions à exécuter
LOOP
ou :
DO [WHILE condition] [UNTIL condition]
... Blocs d'instruction à exécuter si la condition de DO est vraie ou selon le cas, tant que la condition de LOOP est vraie.
LOOP [WHILE condition] [UNTIL condition]

En règle ordinaire, une boucle DO...LOOP n'est abandonnée que suite à la rencontre d'une instruction d'interruption (END,EDIT,STOP) ou après satisfaction de la condition portée dans une instruction EXIT IF, ou suite à une instruction GOTO appelant un label étranger à la boucle ou encore après l'emploi de la fonction d'interruption.

D'autre part, il est possible d'adjoindre une condition de branchement et une condition d'abandon à une construction DO LOOP. Tant l'instruction DO que celle LOOP peuvent être complétées par un test conditionnel de type WHILE ou UNTIL (Cf. WHILE...WEND, REPEAT...UNTIL). Lors de l'utilisation de la structure DO...LOOP en l'absence de l'adjonction d'instructions de boucle complémentaires, LOOP est remplaçable par ENDO (l'interpréteur en assure la traduction en LOOP). L'insertion d'une structure DO LOOP dans une autre et ce à quelque niveau d'imbrication que ce soit, est possible.

Veuillez consulter l'application qui en est faite dans l'exemple illustrant EXIT IF.

FOR...NEXT (F ... N) — Boucle à compteur interne

FOR Nombre=Début TO [DOWNTO] Fin [STEP Pas]
...Bloc d'instructions à exécuter
NEXT Nombre

Un compteur est intégré à la structure même de la boucle FOR...NEXT.

La première ligne de la boucle contient en Début la valeur d'initialisation de boucle et en Fin celle d'abandon de boucle. La valeur numérique Nombre est lors de l'initialisation de la boucle identique à celle de Début et suite à chaque exécution de la boucle, incrémentée ou selon le cas décrémentée d'une unité et cela jusqu'à l'atteinte de la valeur de Fin. A chaque décompte, toutes les lignes d'instructions comprises entre le FOR et le NEXT correspondant sont exécutées. Dès que la valeur d'abandon est atteinte, le programme se poursuit à la prochaine ligne d'instruction exécutable après NEXT.

Le paramètre STEP étant omis lors de l'insertion de l'instruction FOR...TO...NEXT, le pas d'incrémentation du compteur de boucle est affecté de la valeur par défaut : +1. Lors de l'emploi de la structure de boucle FOR...DOWNTO...NEXT, l'indication d'une valeur supérieure dans Début à celle de Fin est requise, puisque dans cette forme d'écriture, le pas du compteur est négatif : -1. L'option STEP ne lui est pas adjoignable. L'option STEP (uniquement avec les boucles TO) agit de manière à ce que la valeur ou l'expression qui lui est adjointe soit comprise en tant que pas de décompte. La notification de valeurs négatives y est aussi possible, mais de même manière qu'avec l'emploi de DOWNTO, le choix d'une valeur de Fin inférieure à celle de Début est logiquement requise.

Si lors de l'emploi de STEP, les valeurs échelonnées entre Début et Fin ne sont pas divisibles entièrement par la valeur du pas adjointe à STEP, le calcul de la valeur finale traitée, s'opère de la façon suivante :

```
ABS(Fin-Début)-((ABS(Fin-Début)) MOD ABS(Pas)
```

Si en une boucle positive, la valeur de Début est supérieure à celle de Fin ou si en une boucle négative, la valeur de Début est inférieure à celle de Fin, la boucle est malgré cela exécutée au moins une unique fois avec la valeur de Début (cette remarque n'est plus valable pour les versions supérieures à la 3.01).

En lieu et place de NEXT on peut écrire ENDFOR Var.

REPEAT ... UNTIL (REP ... U) — Boucle conditionnelle

REPEAT Bloc d'instructions à exécuter
UNTIL condition

La fonction de décision REPEAT...UNTIL est insérable dès qu'il ne s'agit pas de lier le nombre de réitérations de la boucle à l'atteinte d'une valeur finale et dès qu'une exécution de boucle au moins est requise a priori.

La condition d'abandon de boucle y est vérifiée en fin de boucle. Ce qui veut dire que la boucle est exécutée au moins une fois avant que la condition adjointe à UNTIL soit testée. L'insertion d'une instruction d'abandon conditionnel : EXIT IF est possible au sein de la boucle. Dès la satisfaction de la condition portée dans cette dernière, le programme poursuit son déroulement à la ligne d'instruction suivant le prochain UNTIL.

Cette construction répond particulièrement à l'insertion de combinaisons multiples d'abandons conditionnels au sein d'une même boucle. Il est ainsi possible de lier simultanément en tant que conditions d'abandon, les pressions d'une touche et d'un bouton de la souris déterminés, la validation à l'aide de la souris d'une option de menu définie, le contenu d'une variable ou l'atteinte de limites spécifiées a priori.

En admettant que l'abandon d'une boucle REPEAT...UNTIL doit résulter de la satisfaction des conditions suivantes :

```
la pression du bouton gauche de la souris et l'atteinte de la
valeur 100
```

ou

de la frappe de la touche <ESC>

ou

de l'écoulement de plus de 10 secondes

la construction de la boucle pourrait s'énoncer ainsi :

```
T%=TIMER                    ! note l'heure précise de début
REPEAT                      ! initialise la boucle
  INC A%                    ! compteur d'incrémentation +1
  K=MOUSEK                  ! détection de la pression du
                            ! bouton
  Key%=ASC(RIGHT$(INKEY$)) ! retourne la valeur de la touche
  '                           frappée
UNTIL (K=2 AND A%>99) OR Key%=27 OR (TIMER-T%)>2000
A l'aide des opérateurs booléens AND/OR/NOT/XOR/IMP/EQV, ces
conditions peuvent être combinées ensemble de toutes les
manières imaginables. ENDREPEAT est utilisable en lieu et place
d'UNTIL. L'interpréteur transforme automatiquement cette
expression en UNTIL.
```

WHILE ... WEND [W ... WE] — Boucle conditionnelle

WHILE condition
... bloc d'instructions à exécuter
WEND

La boucle WHILE ... WEND a la propriété de vérifier la condition de son exécution, immédiatement suite à son branchement.

Il est ainsi possible que suite au branchement de la boucle, le programme se poursuive immédiatement après le prochain WEND. La condition requise étant insatisfaite lors de l'atteinte de la ligne d'initialisation de la boucle, aucune exécution du bloc d'instructions contenu entre WHILE et WEND n'a lieu. Consultez à ce sujet l'exemple illustrant la description d'EOF().

ENDWHILE peut être employé en lieu et place de WEND. L'interpréteur opère de lui-même la permutation de ces énoncés.

6.2 Branchements conditionnels

EXIT IF — Abandon conditionnel de la boucle

EXIT IF condition

Cette instruction se montre d'une utilité essentielle dès qu'il s'agit d'insérer en une structure de boucle FOR...NEXT, DO...LOOP, REPEAT...UNTIL ou WHILE...WEND, une ou plusieurs conditions d'abandon et cela en n'importe quel(s) lieu(x) de celle-ci.

Une boucle complétée d'une telle condition de fin est abandonnée indépendamment de ses propres conditions d'exécution, dès que la condition introduite par EXIT IF est vraie. Le programme se poursuit ensuite à la première ligne d'instruction suivant la fin de boucle. Il importe de noter que lors d'une telle procédure, le programme saute directement de la ligne EXIT IF derrière celle de l'instruction de reprise de boucle. Le bloc de programme contenu entre EXIT et la reprise de boucle y est purement et simplement ignoré.

Les emplois de la boucle DO...LOOP et de la condition de sortie EXIT permettent de structurer différemment le programme illustrant REPEAT...UNTIL, comme voici :

```
T%=TIMER                ! mémorise l'heure précise de branchement
DO                      ! branche la boucle
  EXIT IF (TIMER-T%)>2000 ! 10 secondes se sont-elles écoulées?
  INC A%                ! compteur d'incrémentation +1
  K=MOUSEK              ! retourne la valeur du bouton pressé
  Key%=ASC(Right$(INKEY$))! retourne la valeur de la touche
frappée
  EXIT IF Key%=27       ! la touche <ESC> a-t-elle été frappée?
  EXIT IF K=2 AND A%>100  !si bouton droit et compteur > 100
```

La particularité de cet exemple est d'induire l'abandon immédiat de la boucle dès l'écoulement de plus de 10 secondes et cela avant la satisfaction de toute autre condition de sortie. Ce qui veut dire qu'en ce cas, les tests de détection de la frappe d'une touche et de la pression du bouton de la souris, et la valeur initiale A% du compteur sont conservés et pris en compte tels quels lors du prochain branchement de la boucle.

EXIT IF est insérable au sein des fonctions de décision IF...ENDIF et SELECT...ENDSELECT. Sa vérification amène la poursuite du programme à la suite de la structure de boucle où cette sortie conditionnelle de boucle est insérée.

IF (ELSE) ENDIF (I...(E...)EN) Fonction de décision conditionnelle

ELSE IF (E) Sous-fonction de décision conditionnelle

IF condition [THEN]
... bloc d'instructions à exécuter si la condition requise est satisfaite
[ELSE
... bloc d'instructions à exécuter si la condition requise est insatisfaite]
ENDIF
ou :
IF condition1 [THEN]
... bloc d'instructions à exécuter si la condition1 requise est satisfaite
[ELSE IF condition2
... bloc d'instructions à exécuter si la condition1 requise est insatisfaite et la condition2 requise est satisfaite
[ELSE IF condition3
... bloc d'instructions à exécuter si toutes les conditions précédentes sont insatisfaites et la condition3 requise est satisfaite
...
selon le cas, d'autres prises de décision conditionnelle ELSE IF
...
[ELSE
... bloc d'instructions à exécuter si toutes les précédentes conditions ont été insatisfaites
ENDIF

Il est possible en GFA BASIC de lier l'exécution d'un bloc d'instructions étendu : une division particularisée d'un programme, à la seule satisfaction d'une unique condition. Seul ENDIF (ou selon le cas ELSE) est à même de clore une séquence d'instructions dont l'exécution dépend de la satisfaction préalable d'une condition IF (SI en langue anglaise).

La condition requise étant vraie, le bloc d'instructions introduit par IF et clos par l'ENDIF (ou par ELSE) correspondant, est exécuté. Lors de l'emploi de l'option ELSE, les instructions contenues entre ELSE et l'ENDIF correspondant, sont traitées dès que la condition requise se montre fausse. Dès que le bloc d'instructions assujetti à un IF ou à un ELSE est exécuté, le programme se poursuit immédiatement à la ligne suivant l'ENDIF correspondant.

Les fonctions de décision IF peuvent être imbriquées en des niveaux de profondeur quelconques.

L'adjonction complémentaire de THEN après IF répond d'un souci de compatibilité avec d'autres dialectes BASIC. Son omission en GFA BASIC n'entraîne aucune conséquence.

La première instruction IF étant fausse, et l'une des conditions ELSE étant vraie, seul le bloc d'instructions introduit par celle qui est vraie, est traité. Suite à son exécution, le programme se poursuit à la ligne d'ordre suivant ENDIF. Si lors de l'emploi de l'option de clôture conditionnelle ELSE, aucune des conditions IF précédentes n'est satisfaite, le bloc d'instructions introduit par ELSE est exécuté. ELSE est l'alternative de tous les IF consignés.

Les fonctions de décision illustrant la suite de nos développements connaissent l'adjonction de plusieurs conditions simultanées, combinées entre elles à l'aide de l'opérateur logique AND. Le bloc d'instruction n'est ainsi à exécuter que si toutes les conditions qui y sont consignées sont vraies.

```
IF condition1=True AND condition2=True AND condition3=FALSE
...bloc d'instructions à exécuter
ENDIF
```

Dans cette illustration, le bloc d'instructions n'est traité que si toutes les conditions indiquées sont satisfaites. La seconde version se montre d'emblée bien plus bavarde que celle initiale. Néanmoins elle a l'avantage d'isoler les prises de décisions une à une et donc de particulariser les blocs d'instructions qu'elle pourrait introduire.

```
FILESELECT "Votre choix :","Charger","DF0:",f$
IF f$>""                                        ! condition1
 PRINT "Boite Ok ou Double Clic"                ! exécution1
 IF RIGHT$(f$,4)=".DAT"                         ! condition2
  PRINT "Extension correcte"                    ! exécution2
  IF LEN(f$)>4                                  ! condition3
  PRINT "Nom du fichier : ";LEFT$(f$,LEN(f$)-4)! exécution3
  PRINT "Extension : .DAT"                      ! fin de bloc
  ENDIF
 ENDIF
ENDIF
```

Il en irait de même lors de l'emploi de l'opérateur logique OR en lieu et place du AND. La spécification de plusieurs conditions combinées entre elles par l'opérateur OR, suspend l'exécution du bloc d'instruction ainsi introduit à la vérification de l'une des conditions spécifiées. Dans l'illustration suivante, les précédentes conditions sont exposées sous leur forme négative, afin de faciliter la comparaison des deux structures de boucle.

```
If condition1<>True OR condition2<>True OR condition3<>False
  exécution si les conditions sont testées fausses
ELSE
...bloc d'instruction à exécuter
ENDIF
```

Dans la pratique, cela pourrait s'énoncer ainsi :

```
FILESELECT "Votre choix :","Charger","DF0:",f$
IF f$=""                                        ! condition1
  PRINT "Sélection de la boîte d'abandon"       ! Résultat
ELSE IF RIGHT$(f$,4)<>".DAT"                    ! condition2
  PRINT "Extension incorrecte"                  ! Résultat
ELSE IF LEN(f$)=4                               ! condition3
  PRINT "Nom de fichier incorrect"              ! Résultat
ELSE                ! aucune des conditions n'étant validée
  PRINT "Sélection de fichier correcte !"       ! donc exécution
ENDIF
```

La différence essentielle entre la structure ordinaire IF et le ELSE IF tient dans l'instruction ENDIF qui clôt le IF ordinaire. Il est donc possible d'exécuter des blocs d'instructions indépendants de celui traité et liés aux conditions précédentes. La nécessité de tels raffinements est certes extrêmement rarissime, mais n'est pas à écarter absolument.

L'écriture de la procédure à l'aide d'une structure ELSE IF est impossible puisque les lignes PRINT 5 et PRINT 6 n'y sont pas insérables.

Ce qui est commun aux deux structures, est que suite à la vérification d'une condition, les tests de conditions suivants sont ignorés.

Vous vous demandez certainement comment s'effectue la prise de décision, autrement dit la vérification de la condition spécifiée. L'explication de la vérification d'une simple condition, comme par exemple IF A%=1, est aisée. Si la valeur 1 est affectée à la variable A%, la condition est vraie, sinon elle est fausse. Introduisez, s'il vous plaît en mode direct :

```
PRINT 1=1
```

L'interpréteur compare selon les règles de l'algèbre booléenne, les deux valeurs en présence. Ensuite il livre la valeur de vérité: -1 (True, soit Vrai). Veuillez consulter à ce sujet, la signification des opérateurs isolés et leurs règles de priorité, dans le chapitre 8.1.

Lors de combinaisons logiques complexes, de multiples difficultés peuvent surgir. Il convient d'ordonner précisément et logiquement les opérations booléennes à l'aide des parenthèses. Ces dernières se montrent d'une importance capitale, puisque elles entraînent des résultats divers selon leur emplacement.

```
PRINT (2^2=4)+((7-1)>3)          donne : -2
PRINT (2^2=4+((7-1)>3)           donne :  0
PRINT (2^2=4)+(7-1)>3            donne : -1
PRINT (2^2=4)+7-(1>3)            donne :  6
```

Rien de mystérieux à cela ! La position des parenthèses explique ces différents résultats. L'explication est fort simple. Dans la première ligne, l'énoncé (2^2=4) représente le résultat d'une condition de même que l'expression ((7-1)>3). Les deux expressions sont testées séparément sur leur propre valeur de vérité. La première expression étant vraie: 2 puissance 2 étant égal à 4, la valeur TRUE (-1) en résulte.

La deuxième expression est également vraie puisque 7 moins 1 (donc 6) est supérieur à 3, la valeur TRUE (-1) est retournée. Puis les deux valeurs de vérité sont additionnées. Nous avons donc bien comme résultat :(-1)+(-1)=2.

Dans la seconde ligne, les parenthèses de la première expression ont simplement été omises. Il en résulte une toute autre constellation. Dans ce cas, est d'abord calculé le résultat de 2 puissance 2 (4). Le signe d'égalité n'y est pas considéré comme appartenant à la première expression, mais comme opérateur de combinaison avec la deuxième expression. Cette dernière délivre elle aussi TRUE (-1). L'évaluation finale s'énonce 4=(-1). 4 n'étant pas -1, la ligne d'instruction édite et délivre la valeur de vérité: FALSE, donc 0.

Dans la troisième ligne, la première expression est une nouvelle fois (2^2=4). La modification est intervenue au sein de la seconde expression. Les parenthèses externes y sont absentes. 2^2=4 est une chose évidente ! Nous obtenons de nouveau TRUE (-1) pour la première expression. La deuxième expression est réduite à (7-1). Cela donne 6. Ensuite sont additionnées les deux expressions: 6+(-1), ce qui donne 5. La vérification de la supériorité du résultat de cette addition au chiffre 3, est entreprise. Comme 5 est effectivement supérieur à 3, le résultat final de cette ligne sera la valeur de vérité TRUE (-1).

La quatrième et dernière ligne se montre vue de loin identique aux précédentes. Ici seuls le 7 de la seconde expression a été mis hors parenthèses et (1>3) mis ensemble. La première expression nous est déjà connue. Elle délivre, comme en première et troisième lignes, la valeur -1 (TRUE). Puisque les parenthèses ont priorité sur tout autre opérateur, nous calculons ensuite l'expression (1>3). L'usage veut que 3 ne soit pas inférieur à 1. Nous obtenons ainsi la valeur 0 (FALSE). Quelques connaissances mathématiques minimes, nous permettent de factoriser les présentes expressions : (-1)+7+(0)=6.

Mais tout ceci serait trop simple, si les opérateurs logiques n'existaient pas !

```
PRINT 10*10 AND 14 OR 3<>2          donne:  -1
PRINT 10*10 AND (14 OR 3)<>2        donne: 100
PRINT 10*(10 AND 14 OR 3<>2)        donne: -10
```

Ces trois lignes dont l'identité apparaît certaine aux yeux du profane, délivrent néanmoins des résultats différents.

La solution de cette énigme :

Dans la première ligne aucune parenthèse n'a été posée. Il s'ensuit que la seule priorité à observer est celle que connaissent les opérateurs logiques entre eux. Ainsi, est calculé premièrement le résultat de l'expression 10*10. Cela donne 100. Puis ensuite a lieu la comparaison <>. Les comparaisons ont priorité sur les opérations de combinaisons logiques. D'abord est vérifiée l'inégalité de 2 et 3. Cette inégalité étant usuelle et vraie, nous en obtenons la valeur -1 (TRUE). En conclusion, sont traitées les opérations booléennes propres à cette ligne. Puisque les deux sont formulées en un niveau identique, il s'ensuit la question de la priorité de l'un sur l'autre. Est-ce AND qui a priorité sur OR, ou inversement ? En de pareils cas, la règle d'or suivante vaut : l'ordre de priorité s'ordonne de gauche à droite ! D'ailleurs nous supposons que vous avez étudié les opérateurs logiques, soit en lisant le chapitre 4 : Les bases élémentaires du BASIC, soit par vos propres moyens.

```
100 en Binaire :   01100100
AND   14 en Binaire :   00001110
                        --------

00000100 ==> en décimal : 4 >--
 ____________________________________________________________!
 !     4 en Binaire :        100
OR    -1 en Binaire : 1...111111
                      ----------

                      1...111111 ==> en décimal : -1
                                     ===============
```

Le résultat de la première ligne est donc -1 (TRUE).

Dans la seconde ligne, aucune autre modification que la mise entre parenthèses de (14 OR 3) constituant ainsi une expression, n'a été entreprise. Nous obtenons donc :

```
      14 en binaire :   00001110
OR    -1 en binaire :   00000011
                        --------
                            1111   ==> en décimal: 15
                                       ==============
```

Comme en première ligne, la valeur 100 résulte de 10*10. Puisqu'ici aussi la comparaison <> a priorité sur AND, nous déduisons l'énoncé vrai : donc la valeur -1; 15 étant effectivement inégal à 2. Enfin, nous combinons les deux résultats partiels avec l'opérateur logique AND :

```
      100 en binaire:   01100100
AND    -1 en binaire: 1...111111
                      ----------
                      01100100  ==> en décimal: 100
                                    ===============
```

Le résultat de la seconde ligne est 100.

La troisième ligne n'est guère différente. Comme nous le disions, l'expression mise entre parenthèses est calculée en premier lieu. Nous y trouvons avec la priorité la plus haute, le signe d'inégalité <>. Trois étant différent de deux, nous obtenons donc TRUE (-1). En second lieu, les combinaisons AND et OR sont placées vis à vis de l'autre sur un même niveau. Donc nous commençons par la gauche :

```
       10 en binaire :    01100100
AND   -14 en binaire :    00001110
                          ---------

                          00001010 ==> en décimal : 10 >-
  ------------------------------------------------------!
  !-> 10 en binaire :        1010
OR     -1 en binaire :  1...111111
                        ----------

                        1...111111 ==> en décimal : -1
                                       ===============
```

Pour résultat de l'expression mise entre parenthèses, nous obtenons de nouveau -1 (TRUE). Cette valeur : -1 est maintenant multipliée par 10 et il en résulte -10, n'est-ce pas ?

Nous espérons que les présents exercices vous seront d'une aide certaine lors de vos futures confrontations avec les combinaisons logiques. Du moins, l'initiation à la complexité de leurs traitements vous est maintenant acquise. La combinaison de conditions telle que la suivante ne risquera plus de vous étonner :

```
IF (10*(10 AND 14 OR 3<>2))=-10
  PRINT "EUREKA"
ENDIF
```

Afin d'exercer vos tout récents talents, voici un petit programme résolvant un casse-tête chinois.

Imaginez que vous ayez à résoudre le problème suivant :

```
XX*X     Un nombre à deux chiffres est à multiplier
         avec un deuxième nombre sur un chiffre.
  XX     Le résultat doit être un nombre à deux chiffres
+ XX     Il y est ajouté une valeur de deux rangs de
_____    chiffres et le résultat de l'opération doit de
= XX     nouveau être un nombre à deux chiffres.
```

Le piment de la chose est que tous les neufs chiffres échelonnés de 9 à 0 doivent être utilisés au moins une fois dans les 9 rangs de chiffres possibles. Chacun des chiffres ne doit donc apparaître qu'une seule fois dans les diverses opérations. Un seul résultat correct est possible.

Le prochain programme résoud ce problème de manière absolument non mathématique. Certes on peut envisager diverses solutions mathématiques bien plus élégantes. Mais ces dernières sont d'une complexité extrême et comme il serait peu sage de vous contraindre à approfondir vos connaissances mathématiques, nous nous sommes arrêtés à une solution "logique". Nous nous en excusons auprès des mathématiciens. Nous devons d'ailleurs vous avouer que nos divers essais de formalisation algébrique du présent problème se sont soldés par des échecs et nous avons préféré abandonner.

Une autre particularité de ce programme est de permettre des économies substantielles de temps de traitement par la modification ultérieure de ses lignes d'instruction. Veuillez essayer de le modifier mais de manière telle que son algorithme logique ne s'en trouve ni bouleversé, ni altéré.

A titre d'illustration, il est possible de déplacer la dernière condition IF

```
IF LEN(résultat$+B$)=9
```

trois lignes plus bas, juste derrière B$=Str$(B%).

En outre il est aussi possible de supprimer la transformation en chaîne I$=Str$(I%) et d'insérer directement l'expression Strs$(I%) en lieu et place d'I$, ou d'utiliser en lieu et place de variables entières de 4 octets : I%, J%, K%, A%, B% et C%, simplement des variables réelles. L'économie de temps réalisée dans les trois cas s'en trouvera accrue. Le choix des formulations est de votre libre ressort. Ces diverses

expérimentations d'accélération du traitement instruit par le programme, vous seront fort utiles pour les développements et optimisations de vos propres programmes.

```
t%=TIMER                        ! mémorise l'heure de début
FOR i%=12 TO 98                 ! Boucle énumérant tous les nombres
                                ' à 2 chiffres possibles
  IF (i% MOD 11)>0 AND (i% MOD 10)>0 ! Si I% est entièrement
    '                           ! divisible par 10 ou 11, il ne
    '                           ! peut s'agir uniquement d'un
                                ! nombre à
    '                           ! deux chiffres identiques (11,
    '                           ! 22,..) ou de chiffres de
                                ! dizaines (10, 20...) ici
    '                           !  inutilisables.
    i$=STR$(i%)                 ! transforme I% en chaîne
    i2$=STR$(i% MOD 10)         ! le chiffre du rang des
                                ! unités de I
    i3$=STR$(i% DIV 10)         ! le chiffre de rang décimal de I
    FOR j%=1 TO 9               ! boucle du 1er rang
      j$=STR$(j%)               ! transforme J% en une chaîne
  IF INSTR(i$,j$)=0             ! la valeur de J% est-elle contenue
                                ' dans celle de I%
      a%=i%*j%                  ! si non, la multiplication s'opère
  IF a%>11 AND a%<99            ! la valeur résultante est-elle à 2
                                ' rangs de chiffres et est-elle
  IF (a% MOD 11>0) AND (a% MOD 10>0) ! non divisible
                                ! entièrement par 11 ou 10?
      CLR c%                    ! efface la zone mémoire INSTR
      a$=STR$(a%)               ! transforme A% en une chaîne
      a2$=STR$(a% MOD 10)       ! chiffre des unités de A%
      a3$=STR$(a% DIV 10)       ! chiffre des dizaines de A%
      c%=INSTR(a$,i3$)          ! la dizaine de I% est-elle en A%?
      c%=MAX(c%,INSTR(a$,i2$))  ! chiffre des unités d'I% en
                                ! A%?
      c%=MAX(c%,INSTR(a$,j$))   ! J% est-il contenu dans A%?
  IF LEN(i$+j$+a$)=5 AND c%=0   ! Si 5 résulte des calculs
              '  menés et si tous les chiffres
              ' additionnés sont distincts l'un de
              ' l'autre
    FOR k%=12 TO 98             ! boucle pour la valeur à
              ' additionner
      k$=STR$(k%)               ! transforme K% en une chaîne
      k2$=STR$(k% MOD 10)       ! chiffre des unités de K%
      k3$=STR$(k% DIV 10)       ! chiffre des dizaines de K%
    résultat$=i$+j$+a$+k$       ! donne le résultat des opé
              ' rations déjà menées
  IF LEN(resultat$)=7       ! 7 chiffres ?
  IF (k% MOD 11>0) AND (k% MOD 10>0) ! K% est-il divisible
              ' entièrement par 10 ou 11?
      c%=INSTR(a$,k3$)          ! la dizaine de K% est-elle
```

```
                    ' en A%?
    c%=MAX(c%,INSTR(a$,k2$)) ! le chiffre des unités de
                    ' K% en A%?
    c%=MAX(c%,INSTR(i$,k3$)) ! la dizaine de K% en I%?
    c%=MAX(c%,INSTR(i$,k2$)) ! le chiffre des unités de
                    ' K% est-il en I%?
    c%=MAX(c%,INSTR(k$,j$))  !  J% est-il contenu en K%?
IF c%=0                     ! Tous les chiffres obtenus
                    ' sont-ils distincts l'un de l'autre?
    b%=a%+k%                ! le résultat de la multiplication
                    ' est additionné avec K%
    b$=STR$(b%)             ! transforme B% en une chaîne
IF b%>11 AND b%<99          ! le résultat est-il à 2 rangs?
IF (b% MOD 11>0) AND (b% MOD 10>0) ! est-il en-
                    ' tièrement divisible par 10 ou 11?
IF LEN(resultat$+b$)=9      ! le nombre de tous les
                    ' chiffres employés est-il 9?
    c%=INSTR(b$,i3$)        ! Vérifie si l'un
    c%=MAX(c%,INSTR(b$,i2$)) ! l'un des
    c%=MAX(c%,INSTR(b$,a3$)) ! chiffres utilisés
    c%=MAX(c%,INSTR(b$,a2$)) ! est
    c%=MAX(c%,INSTR(b$,k3$)) ! contenu
    c%=MAX(c%,INSTR(b$,k2$)) ! dans le résultat
    c%=MAX(c%,INSTR(b$,j$))
IF c%=0                         ! si non
              PRINT "Secondes: ";(TIMER-t%)/200 ! alors
              PRINT                             ! édite
              PRINT ''i%;" * ";j%               ! la
              PRINT "----------"                ! solution
              PRINT "   ";a%                    ! du
              PRINT "  +";k%                    ! problème
              PRINT "----------"                ! ...
              PRINT "  =";b%                    ! ...
                  END                           ! fin!
                ENDIF !
               ENDIF  ! pour les tests INSTR
              ENDIF   ! ne pouvant pas être introduits
             ENDIF    ! avec la fonction MAX()
            ENDIF     ! Les tests INSTR permettent de
           ENDIF      ! déterminer si les chiffres isolés
          ENDIF       ! ont été ouou non déjà utilisés
                      ! auparavant.
        NEXT k%       ! Puisque plusieurs conditions
                      ! INSTR se
      ENDIF           ! succèdent et sont testées sur
                      ! leur valeur
    ENDIF             ! nulle, la valeur C% du prochain
                      ! test peut
  ENDIF               ! être superposée. La conservation
 ENDIF                ! des rangs de chiffres déjà
                      ! trouvés,
```

```
    NEXT j%                 ! est assurée par leur sauvegarde
  ENDIF                     ! grâce à MAX() dans la ligne
NEXT i%                  z  ! suivante de test INSTR().
```

SELECT (CONT) CASE (TO) (DEFAULT) ENDSELECT

{ S } { CON } { DEFA } { ENDS }
Fonction de choix multiple

SELECT Expression
CASE Constante1 [TO Constante2 [,[...] TO [...]]]
... bloc d'instructions à exécuter au cas où Expression est identique à Constante1, ou lors de l'emploi optionnel de TO dès qu'Expression est vérifiée comprise dans l'intervalle délimité par Constante1 et Constante2
[CONT]
[CASE Constante1 [Constante2 [,Constante3 [,...]]]
... bloc d'instructions à exécuter au cas où Expression est vérifiée identique à Constante1, à Constante2 ou à Constante3 ou ...]
...ou selon, d'autres cas alternatifs CASE
[CONT]
[DEFAULT]
...bloc d'instructions à exécuter si toutes les vérifications précédentes se sont avérées fausses]
ENDSELECT

Cette fonction de prises de décisions circonstanciées offre la possibilité de lier une induction déterminée à une expression donnée (select=sélectionner/case=cas). Nombreux sont ceux qui connaissent le rôle de cette construction dans le langage C. Expression peut être une quelconque expression numérique ou alphabétique, ou le résultat d'une expression d'opération logique, arithmétique.... L'indication de variables ou de constantes y est aussi possible.

Lors de l'utilisation d'une expression alphanumérique: une constante ou une variable, seuls ses quatre premiers caractères sont pris en considération lors de l'opération de comparaison. Elle est transformée par l'interpréteur en une valeur de 4 octets.

Une valeur étant contenue dans Expression, seule l'indication d'une valeur numérique ou d'un texte de maximum 4 caractères (une constante ou une variable de chaîne) est possible en Constante puisque la comparaison est menée sur deux valeurs. Une chaîne de texte étant contenue dans Expression, les quatres premiers caractères de la chaîne sont comparés en termes de codes ASCII, avec les quatres premiers de Constante ou des Constantes données, du moins si elles ont été notifiées auparavant.

Exemples :

```
A%=ASC(INKEY$)*2^24+ASC(INKEY$)*2^16+ASC(INKEY$)*2^8+ASC(INKEY$)
SELECT A%
CASE "abcd"
  PRINT "abcd ont été saisis"
DEFAULT
  PRINT "N'importe quoi !"
ENDSELECT
```

En ce cas, la frappe de 4 touches est testée. La valeur ASCII de l'ensemble des touches est assemblée en une valeur entière de quatre octets. CASE opère de manière identique au cas où une chaîne de texte est contenue dans Expression. Une chaîne MKL$, MKI$ ou CHR$ doit être d'une longueur identique à Constante, pour que la comparaison puisse être entreprise.

```
Une chaîne de trois caractères :
   (Valeur ASCII du premier   signe)*(2^16)
+ (Valeur ASCII du second     signe)*(2^ 8)
+ (Valeur ASCII du troisième signe)

Une chaîne de deux caractères:
   (Valeur ASCII du premier   signe)*(2^8)
+ (Valeur ASCII du second     signe)
    et ainsi de suite...
```

Lors de la sélection de chaîne avec SELECT, seule l'indication d'une chaîne de quatre caractères dans Constante après Case est autorisée.

Dès qu'Expression correspond à Constante, la séquence d'instructions suivante est exécutée. Celle-ci ayant été traitée, le programme se poursuit dans la ligne d'instruction succédant à ENDSELECT. Les autres alternatives éventuelles CASE disposées dans la même structure sont ignorées. La construction SELECT...CASE se montre en

cela analogue au IF...ELSE...IF. Leur différence essentielle est que l'interpréteur se charge lui-même des comparaisons ordonnées dans une construction SELECT...CASE

Sur le plan de la lisibilité, nous voyons que l'instruction CASE est supérieure à son équivalent IF...ELSE. En ce qui concerne la vitesse, l'instruction CASE s'exécute plus rapidement. Les constructions de type IF...ELSE IF devraient donc être réservées aux cas que l'instruction SELECT...CASE ne permet pas de traiter.

```
T%=TIMER
FOR I%=1 TO 10000          ! 10000 réitérations de la boucle
  SELECT I%
  CASE 20000               ! premier  test
  CASE 20000               ! deuxième test
  ENDSELECT
NEXT I%
PRINT "Test CASE : ";(TIMER-T%)/200;"secondes"
T%=TIMER
FOR I%=1 TO 10000          ! 10000 réitérations de la boucle
  IF I%=20000              ! premier test
  ELSE IF I%=20000         ! deuxième test
  ENDIF
NEXT I%
PRINT "Test ELSE IF : ";(TIMER-T%)/200;"secondes"
```

Dans la première structure de comparaison, vous observez grâce aux sauts de lignes que dès qu'une condition CASE est satisfaite aucun des tests CASE suivants n'est mené. De ce fait, dans l'illustration suivante, seule la ligne PRINT "A" est traitée, bien que la condition alternative suivante : CASE "A" TO "Z" soit aussi satisfaite.

```
X$="A"
SELECT X$
CASE "A"
  PRINT "A"
CASE "A" TO "Z"
  PRINT "A à Z"
ENDSELECT
```

A l'aide de l'indication optionnelle de TO, il est possible de délimiter un intervalle déterminé, par exemple CASE 1 TO 10, CASE "a" TO "z" ou CASE "abc" TO "xyz" dans lequel Expression devra être compris pour que l'exécution du bloc d'instructions ainsi initialisée soit entreprise. En cas d'omission de l'indication de la valeur inférieure, précédant TO ou de la valeur supérieure, suivant TO, est affectée en tant que valeur par défaut, la plus petite ou la plus grande possible disponible dans

l'interpréteur. L'emploi de la virgule en tant que séparateur autorise le regroupement de plusieurs indications isolées en une unique ensemble, par exemple : CASE "a","h","j","m" ou CASE 1,33,7. Il est possible d'adjoindre des conditions multiples en une même ligne d'instruction CASE: par exemple : CASE TO "b","ABC" TO "XYZ",65,66,67,Æ".

Il est possible d'insérer dans une structure SELECT...ENDSELECT, une condition par défaut DEFAULT en prévision du cas où aucune des conditions introduites par CASE ne serait satisfaite. Le bloc de programme introduit par DEFAULT et délimité en fin par ENDSELECT est exécuté dès que toutes les précédentes conditions n'ont pas été remplies, cela est fort analogue à l'emploi que connaît ELSE avec les conditions IF.

Si suite à la fin d'un branchement de boucle, CONT précède directement l'instruction CASE suivante ; CONT agit de manière à ce que le bloc d'instructions initialisé par le CASE suivant soit exécuté sans que la vérification de la condition qui y est assignée ne soit entreprise. Suite au traitement de ce bloc de programme, le programme se poursuit dans la première ligne suivant l'instruction d'abandon ENDSELECT de la présente fonction de décision. CONT peut aussi être inséré directement devant DEFAULT, ce qui amène, du moins si le bloc d'instructions placé directement devant CONT est exécuté, l'exécution de la séquence alternative introduite par DEFAULT. Si CONT n'est pas directement positionné devant CASE ou DEFAULT, cette instruction est interprétée en tant qu' instruction de saut de programme et induit l'arrêt du traitement en cours.

Du reste rien ne sert d'insérer des lignes de programme entre SELECT et le premier CASE, puisque ces lignes ne sont pas enregistrées par le BASIC.

A la place de DEFAULT, OTHERWISE peut être écrit. Cette expression est modifiée en DEFAULT par l'interpréteur.

6.3 Déclaration de zone

! Déclare un commentaire au sein d'une ligne d'instructions

Ligne de commandes ! texte du commentaire

Ce signe d'introduction a une fonction identique à celle de la commande REM. Le texte ainsi introduit est déclaré en tant que commentaire du programme et ignoré en tant que tel par l'interpréteur. Cette forme de déclaration de commentaire vous est déjà familière puisque nos programmes en font part. En lignes DATA, cette délimitation est toutefois impossible, puisque les commentaires y sont compris en tant que texte DATA.L'instruction INLINE n'autorise pas non plus les commentaires.

```
INPUT A$        ! saisie d'une chaîne
PRINT A$        ! édite la chaîne
EDIT            ! fin de programme
```

S'il est employé en début de ligne, le point d'exclamation est modifié en l'abréviation de REM: l'apostrophe (').

DATA [D] Déclare la mémorisation des données

DATA [Données numériques [,['] données de texte['],...]]

Cette instruction permet la mémorisation d'une liste de valeurs séparées l'une de l'autre par une virgule, ou de texte. Elle anticipe leur lecture future entreprise à l'aide de l'instruction READ.

Comme noté ci-dessus pour RESTORE, DATA peut pointer un label déterminé. READ lit alors les données DATA selon l'ordre observé dans la liste consignée sous ce label. En l'absence de l'emploi préalable de RESTORE, le nombre de données DATA lues à partir du début de programme dépend de celui des ordres READ qui y sont disposés. Mais au cas où y seraient insérées plus d'instructions READ que d'ordres DATA disponibles jusqu'à la fin du programme, il en résulterait l'apparition du message d'erreur correspondant.

Une particularité du GFA BASIC est de ne pas requérir la mise entre guillemets des données de texte saisies, afin de permettre l'emploi de la virgule au sein d'un texte. En l'omission des guillemets, la virgule est évaluée en tant que séparateur des chaînes de données et l'interpréteur lit tous les signes apparaissant (les espaces aussi) dans l'intervalle délimité par les virgules.

Les instructions READ numériques requièrent la disponibilité préalable de données numériques. Elles peuvent toutefois être indiquées en format binaire, hexadécimal, octal. Si une instruction READ de texte lit un signe numérique, ce dernier est interprété simplement en tant que caractère de texte.

Il est impossible d'affecter des variables à une donnée DATA. En l'absence de l'insertion au sein d'un programme de l'instruction READ, n'importe quel texte peut être contenu dans les données DATA, ce texte étant délaissé par l'interpréteur.

Veuillez consulter à cette fin les exemples illustrant WRITE et TAB().

READ [REA] Affecte les données DATA à des variables

READ Var [,Var2,Var$,Var2$,...]

Les données introduites par l'instruction DATA sont affectées aux variables adjointes à READ. En l'absence de l'emploi d'un nom de label instruit par RESTORE, le nombre de lignes DATA lues d'après l'ordre de leur insertion dans le programme en cours, dépend du nombre d'instructions READ disponibles. Pour de plus amples informations, veuillez consulter l'instruction DATA.

REM [R ou ' ou !] — Insertion de commentaire

REM [Commentaire]

Cette instruction autorise l'insertion de commentaire au corps de votre programme et ce en n'importe quel lieu, afin d'en faciliter ou d'en améliorer la compréhension. Ces lignes de commentaires ne sont aucunement prises en compte par l'interpréteur. L'emploi de son abréviation: l'apostrophe " ' " est possible.

En outre un guillemet ouvrant (le signe introductif d'un commentaire au sein d'une ligne de programme) est aussi utilisable. Celui-ci s'il est inséré en début de ligne, est modifié en l'apostrophe, le signe introductif d'un commentaire REM.

Nous vous conseillons vivement de commenter avec soin vos propres productions. Lors de programmes de taille élevée, ces commentaires vous faciliteront fort sa lecture et la recherche des blocs de programme logiquement particularisés. Ces diverses annotations en rendront sa logique interne bien plus significative lors de sa communication à des personnes étrangères ou lors d'une relecture du programme quelques temps après sa création.

RESTORE [RES] — Positionne le pointeur de données

RESTORE [Nom_de_label]

L'emploi de RESTORE en l'omission de l'indication de tout nom de label, amène le positionnement du pointeur des données sur celles contenues dans la première instruction DATA située dans le programme. A l'aide de l'adjonction d'un nom de label, il vous est possible de pointer la première donnée DATA listée sous ce label.

C'est à partir de là que seront lues les prochaines données DATA, jusqu'à ce qu'une nouvelle instruction RESTORE soit exécutée ou la dernière donnée DATA listée soit lue. De plus amples informations sur ce sujet sont consignées sous DATA.

6.4 Déclaration de variables

DEFBIT — Déclaration de variables booléennes

DEFBIT Defstring$

Les instructions DEFXXX assurent la déclaration globale des types de variables. Il est possible d'employer diverses chaînes de définition afin d'affecter toutes les variables à leur type correspondant spécifié à l'aide d'un nom précis.

Cette déclaration est menée en règle générale en début de programme, car sinon la distinction entre les variables déclarées et celles librement définies en serait alourdies. Si jamais une nouvelle déclaration est opérée au sein d'un programme, la précédente est de ce fait désactivée. Suite à une déclaration, l'adjonction d'un suffixe (le suffixe signifiant le type d'une variable entière est %, celui d'une variable booléenne ! et celui d'une variable de chaîne alphanumérique $) aux noms des variables n'est plus requise .

Malgré cela, il est possible à tout instant de définir une variable isolément, même si vous présentez ce même nom de spécification comme une déclaration globale. A cette fin, l'adjonction du suffixe correspondant au nom de la variable est demandée. Cette variable est ainsi distinguée de toutes celles déclarées. Les variables définies isolément ont en règle générale priorité sur les déclarations globales.

DefString$ est une chaîne (une constante, une variable ou une expression) à l'aide de laquelle le nom de spécification est déterminé). Dans les suivants exemples de définitions les chaînes Defstring$ sont échangeables l'une avec l'autre.

Définitions :

DEFBIT "a"
Toutes les variables dont le nom contient en tant que premier caractère un "a", sont, du moins en l'absence d'une définition isolée, déclarées booléennes (Var!).

DEFBYT "b,c,g-l"
Toutes les variables dont le nom débute avec un b, c, g, h, i, j, k, ou un l, sont déclarées entières d'un octet (Var).

DEWRD
Toutes les variables dont le nom commence avec l'énoncé word sont déclarées entières de 2 octets (Var&).

DEFINT "i1,i2,I3"
Toutes les variables dont le nom comprend comme premières lettres i1, i2 ou i3 sont déclarées entières de 4 octets (Var%)

DEFFLT "a-c,x-z"
Toutes les variables dont le nom débute avec un a, b, c, x, y ou z, sont déclarées à virgule flottante de 8 octets (Var#).

DEFSTR "d-f"
Toutes les variables dont le nom contient en première position la lettre d,e ou f sont déclarées variables de chaîne (Var$).

Exemple :

```
Defint "i"              ! toutes les variables débutant avec la
                        ! lettre i valent à partir de
                        ! maintenant en tant que variable
                        ! entière de 4 octets
Print String$ = "XYZ"   ! n'importe quelle correspondance avec
                        !une variable chaîne commençant par i
Print i String $ : XYZ! sont imprimés, sortent jusqu'à la
                        ! place de la définition, la
                        ! spécificaton chaîne indiquée
                        ! directement reste valable.
I 4 Octets :            ! n'importe quelle correspondance avec
                        ! une variable commençant par i sans
                        ! postfix.
Print 1_4 octets        ! 1548892 est sorti imprimé. Les
                        ! chiffres après la virgule sont
                        ! intégrés. La définition 4 octets est
                        ! valable.
```

DEFBIT (DFB) Déclarer des variables d'entiers 1 octet

DEFBIT Defstring$

DEFFLT(DEFFL) Déclarer des variables de virgule flottante 8 octets

DEFFLT Defstring$

Voir explication de DEFBIT

On peut utiliser à la place de DEFFLT, DEFSNG ou DEFDBL. Les indications sont converties par l'éditeur lors du contrôle d la syntaxe en DEFFLT. Sinon voir DEFBIT.

DEFINT (DEFI) Déclarer des variables d'entiers 4 octets

DEFINT Defstring$

Voir explication de DEFBIT

DEFSTR (DEFS) Déclarer des variables de chaîne de caratères.

DEFSTR Defstring$

Voir explication de DEFBIT

DEFWRD (DEFW) Déclarer des variables d'entiers 2 octets

DEFWRD Defstring$

Voir explications de DEFBIT

6.5 Sous-programmes

DEFFN — Définit une fonction

DEFFN Nom_de_fonction [Liste_de_variables]=Fonction

DEFFN représente une confortable possibilité de définition concise d'une fonction numérique ou alphanumérique. Nom_de_fonction représente un quelconque nom dont l'appel par FN (ou @) amène la mise à disposition de la fonction recouverte. Le choix de ce nom ne connaît aucune limitation. Même l'utilisation de noms d'instructions BASIC ou de variables existantes est permis. Cela est possible car le nom de la fonction est inséparable de la fonction définie qu'il recouvre. Toutefois l'emploi de noms de variables matricielles est interdit puisqu'à la différence des variables, leur initiale peut être un chiffre.

L'adjonction d'une liste de noms de variables dont les contenus peuvent être traités à l'intérieur de la fonction y est optionnelle. Les paramètres isolés y sont insérés séparés l'un de l'autre à l'aide d'une virgule. Des variables de type différent : réelles, entières, de chaîne peuvent être placées dans la liste des paramètres. Néanmoins à l'intérieur de la fonction, ne sont permises que des opérations correspondant au type de la fonction utilisée. En d'autres termes, la fonction étant déclarée comme renvoyant une chaîne (par exemple: DEFFN Fonction$), seules des opérations de chaînes y sont exécutables. Naturellement seules des opérations numériques sont insérables dans des fonctions numériques.

Lors de l'insertion d'une liste de paramètres, il convient d'assurer à la suite de l'appel de la fonction, la transmission des valeurs, des chaînes, des variables ou expressions qui leur correspondent. Lors d'une transmission de valeurs, chaînes, etc... inadéquate à la liste (la correspondance des types n'étant pas observée ou le nombre des paramètres se montrant différents), une erreur en résulte et provoque l'apparition d'un message.

L'utilisation des variables de liste à l'intérieur de la fonction n'est pas nécessairement requise. Elles peuvent servir d'arguments muets (dummies). D'un autre côté des variables globales non contenues dans la liste, sont utilisables dans la fonction. Suite à l'appel de la fonction, les contenus actuels des variables utilisées sont repris et traités. S'il existe une variable globale dont le nom est identique à celui d'une

variable de la liste, son contenu n'est pas pris en compte, puisque les variables de la liste sont traitées à l'intérieur de la procédure en tant que variables locales. Elle a exclusivement pour valeur, celle qui lui est affectée par la liste des paramètres lors de l'appel de la fonction. Suite à la sortie de la fonction, le contenu de la variable globale est de nouveau disponible.

La longueur maximale d'une fonction est identique à celle d'une ligne d'insertion (256 caractères). Si et espace se montre insuffisant pour la définition complète d'une fonction, d'autres fonctions peuvent être appelées. Une fonction est ainsi utilisable à des seules fins d'appels de fonctions et d'enchaînement de ces dernières.

Les fonctions sont définissables en n'importe quel lieu du programme. Puisque le programme lors de son lancement initialise toutes les instructions DEFFN, les fonctions sont insérables en fin de programme même si leur appel à l'aide de FN ou @ les précède dans le texte du programme. En début de programme, l'interpréteur recherche toutes les instructions DEFFN disséminées dans le corps du programme. Leur contenu lui est, de ce fait, connu avant l'exécution de la première ligne de commande.

Observation : L'insertion de deux fonctions s'appelant réciproquement dans des boucles perpétuelles, inhibe la fonction d'interruption de boucle : <Control/Shift/Alternate>. Les appels récursifs entraînent le même effet.

FN [@] — Appel de fonction

FN Nom_de_fonction [(Liste_de_paramètres)]

FN et son abréviation : @, servent à appeler des fonctions dont la définition a été opérée par vos propres soins. Les paramètres sont adjoints mis en parenthèses et séparés l'un de l'autre à l'aide d'une virgule. Comme mentionné sous DEFFN, il convient de s'assurer que les présents paramètres spécifiés correspondent aux types des variables indiquées dans la liste de variables de DEFFN.

Le résultat d'une fonction comme celui de toute autre fonction BASIC se laisse éditer à l'aide des instructions d'édition (PRINT, WRITE, TEXT, OUT), affecter en une variable de type correspondant ou intégrer dans un test conditionnel ou une expression.

FUNCTION...RETURN...ENDFUNC[FR...RET..ENDF]F Fonction

FUNCTION Nom [(Var1, Var2%, Var3$,...[VAR Var,...])]
...partie de programme à exécuterRETURN Back
ENDFUNC

FUNCTION signale le début d'une fonction de plusieurs lignes définie par vos propres soins. L'adjonction d'une liste optionnelle de paramètres (comme avec DEFFN et PROCEDURE) qui réceptionneront par la suite les données transmises par FN (ou @), est possible. Mais il convient de veiller à ce que les types des variables correspondent aux paramètres transmis.

Il est aussi possible à l'aide de VAR de définir, à l'intérieur de cette liste, des variables aux quelles est directement transmissible une variable globale à l'aide de FN (Cf. VAR, PROCEDURE). En outre, la transmission de paramètres indirects de pointeur, comme avec PROCEDURE est possible (Cf. GOSUB).

Nom est un identificateur quelconque, désignant la fonction. Un nombre élevé de lignes de programme sont insérables dans FUNCTION. A l'inverse des fonctions définies par DEFFN, les fonctions définies par FUNCTION peuvent s'appeler elles-mêmes et manifester la récursivité la plus absolue.

En principe, FUNCTION est analogue à PROCEDURE. Néanmoins bien qu'ENDFUNC (comme RETURN pour PROCEDURE) clôt une structure FUNCTION, à l'inverse du RETURN propre à PROCEDURE elle n'assure pas la reprise de la procédure. Seule l'insertion de RETURN Back qu'il convient de ne pas confondre avec la fin de procédure : RETURN, assure le retour de la fonction FUNCTION.

Dès que le programme rencontre au sein d'une procédure FUNCTION, une instruction de retour : RETURN Back, la valeur, l'expression ou le contenu de variable Back est retourné au programme en tant que

résultat de fonction et l'appel de la fonction est ensuite renouvelé. De nombreuses instructions RETURN Back sont insérables dans une même FUNCTION (par exemple au sein de tests IF..ELSE IF...ENDIF. Comme les appels FN de fonctions composées d'une seule ligne, l'appel d'une fonction composée de plusieurs lignes est insérable dans des expressions ou des tests conditionnels et utilisable à des fins d'affectation de variables.

L'adjonction du suffixe : $ au nom de la fonction (par exemple: FUNCTION Nom$) assure la délivrance du résultat de la fonction sous forme de chaîne.

Exemple :

```
A$="1 ** UPPER/LOWER-Test_de_Chaîne ** 1"
PRINT "NORMAL: ";A$
PRINT "UPPER$: "Upper$(A$)
PRINT "LOWER$: ";@Lower$(A$)

'
FUNCTION Lower$(L.str$)
  'recherche les majuscules de la chaîne indiquée et les
  'transforme en minuscules.
  'L.str$ = chaîne à transformer
  LOCAL L.c%
  FOR L.c%=1 TO LEN(L.str$)
    SELECT MID$(L.str$,L.c%,1)
    CASE "A" TO "Z"
      MID$(L.str$,L.c%,1)=CHR$(ASC(MID$(L.str$,L.c%,1))+32)
      ' MID$(L.str$,L.c%,1)=CHR$(ASC(MID$(L.str$,L.c%,1))OR
        32)
      ENDSELECT
   NEXT L.c%
   RETURN L.str$
ENDFUNC
```

Les possibilités d'insertion de fonctions composées de plusieurs lignes, comme le suggère ce petit programme, sont multiples et ne connaissent d'autres limites que celles de votre propre fantaisie. Si l'implémentation de la fonction RINSTR n'avait pas été assurée par Franck Ostrowski, son développement serait fort aisé à l'aide de ce type d'application. Si l'échange du contenu d'A$ par celui de la chaîne Lower CASE est désiré après la fin de fonction, la simple modification suivante de l'en tête de la fonction répond à ce souhait:

```
FUNCTION Lower$(Var L.str$)
```

La ligne de commentaire: ' MID$... fait part d'une petite astuce. Les deux lignes MID$... sous "A" TO "Z", sont interchangeables. Il peut être utile de savoir que la conversion des majuscules et des minuscules s'ordonne sur le bit 5 (2^5=32) de leur code ASCII. Ce bit étant activé, il s'agit de minuscules (codes ASCII de 97 à 122), s'il est supprimé, de majuscules.

GOSUB [G ou @] — Branche une procédure

GOSUB Procédure [(Liste_de_paramètres)]

Un branchement GOSUB assure le saut immédiat de la ligne GOSUB à la procédure correspondante. Suite au traitement de la procédure, le retour à la ligne suivant l'appel GOSUB correspondant est assuré par l'instruction de fin de procédure : RETURN et le programme se poursuit à partir de cette ligne.

Le GFA Basic vous offre aussi la possibilité optionnelle de transmettre à la procédure désignée une liste de paramètres de longueur quelconque. Ces paramètres y seront affectés à une liste de variables, spécifiée et mise entre parenthèses dans l'en tête de la procédure (par exemple : PROCEDURE Test(Var,Var1$...)). Le nombre des paramètres GOSUB et celui des variables réceptrices de PROCEDURE ne sont limités que par la longueur maximale d'une ligne de programme qui est de 256 signes.

La seule limite à observer est la correspondance entre le nombre, les types et l'ordre des paramètres spécifiés dans la présente liste et ceux des variables réceptrices notifiées dans l'en tête de PROCEDURE.

Le pointeur représenté par une astérisque (*Var) autorise un adressage indirect des variables. Cela signifie que le pointeur spécifié dans la liste des paramètres transmet à la procédure l'adresse de sa variable, et non pas son contenu.

Les variables initialisées en tête de procédure valent à l'intérieur de celle-ci en tant que variables locales, autrement dit leur contenu n'est disponible qu'au sein de cette procédure. Les variables globales, extérieures à la procédure, pouvant avoir le même identificateur que

les locales sont différenciées automatiquement par l'interpréteur. Suite au retour dans le programme principal, les contenus des variables globales sont restaurés.

L'emploi des caractères alphanumériques (A à Z et 0 à 9), du souligné et du point est autorisé pour l'énonciation des noms de procédure. A la différence des noms de variables, l'utilisation d'un chiffre en première position est permise.

En outre, la transmission directe de variables à l'aide d'un paramètre de pointeur (*Var), est possible. Néanmoins il convient de s'assurer au préalable de la présence à la position correspondante de la liste des variables réceptrices de PROCEDURE, d'une variable VAR de même type. Une application de GOSUB est illustrée sous PROCEDURE.

GOTO [GOT] Branchement inconditionnel d'un label

GOTO Label

Elle est l'une des instructions les plus usitées des langages BASIC traditionnels. A l'inverse de ceux-ci, le GFA Basic sursoit pour ainsi dire à son application par l'offre de possibilités bien plus élégantes à l'aide de GOSUB et de FN.

Elle n'exécute rien de plus que le saut du programme à partir de la ligne GOTO jusqu'au label spécifié. Le choix du nom du label ne connaît aucune limitation. Comme les noms de procédures, un nom de label peut débuter avec un chiffre. Toutefois l'adjonction terminale d'un double point au nom du label est requise.

Du reste le même label est utilisable comme cible d'un saut RESTORE Label. Des sauts dans ou à partir des boucles FOR...NEXT ou de PROCEDURE ne sont pas autorisés. De tels essais de saut provoquent invariablement l'apparition d'un message d'erreur.

LOCAL [LOC] — Déclare les variables locales

LOCAL Variable_locale [, Liste_de_variables]

La déclaration de variables locales s'opère au sein d'une procédure ou d'une fonction. Les variables ainsi déclarées ne sont utilisables que dans la procédure où leur déclaration a lieu, ou dans les procédures appelées par la précédente. Elles tiennent lieu dans les sous-procédures de variables globales.

Lorsqu'une variable possède le même nom dans le programme principal ou dans la routine appelante que celui d'une variable locale de la procédure en cours (Cf. PROCEDURE), l'interpréteur les différencie de lui-même.

Lors d'une déclaration de plusieurs variables, leur séparation à l'aide de virgules est requise. Divers types de variables sont utilisables.

ON...GOSUB — Branchement conditionnel de procédure

ON Expr_évaluable GOSUB Procédure1 [,Procédure2, Procédure3,...]
ON Expr_évaluable Procédure1 [,Procédure2, Procédure3,...]

Cette instruction permet la sélection entre plusieurs procédures dont l'appel dépend de la valeur prise par l'expression numérique, la variable ou la constante spécifiée. La décision de branchement s'ordonne par pas de 1. Nul branchement ne résulte de la valeur 0. Les branchements se conforment au schéma suivant :

```
Valeur >= 1 < 2 amène l'appel de la Procédure1
Valeur >= 2 < 3 amène l'appel de la Procédure2
Valeur >= 3 < 4 amène l'appel de la Procédure3
et ainsi de suite...
```

Lorsque l'expression évaluable revêt une valeur supérieure au nombre des procédures notifiées ou égale à zéro, le programme se poursuit en l'absence de tout branchement à la ligne suivant celle ON GOSUB.

Toutes les formes syntaxiques ON GOSUB propres au GFA Basic n'autorisent uniquement que l'appel de procédures qui n'initialisent pas de liste de paramètres.

Exemple :

```
ON MENU(1)-20 GOSUB Top,Close,Full,Vide,Vide,Vide,Size,Move
PROCEDURE Top
  'mesures d'actualisation d'une fenêtre
RETURN
PROCEDURE Close
  'mesures de clôture d'une fenêtre
RETURN
PROCEDURE Full
  'mesures de l'agrandissement maximale d'une fenêtre
RETURN
PROCEDURE Size
  'mesures de modification de la taille d'une fenêtre
RETURN
PROCEDURE Move
  'mesures pour le déplacement d'une fenêtre
RETURN
PROCEDURE Vide
  'procédure factice sans contenu appelée par une évaluation
  'ne devant induire la moindre effectivité
RETURN
```

Il est largement préférable d'utiliser la structure SELECT...CASE, le ON...GOSUB n'ayant été placé pour des raisons de compatibilité avec les autres BASIC.

ON BREAK (CONT) (GOSUB) — Gestion Interruption

ON BREAK GOSUB Procédure
ON BREAK
ON BREAK CONT

La première forme syntaxique suite à son traitement dans le cours d'un programme, branche la procédure spécifiée dès que la fonction d'interruption <Control><Shift><Alternate> est utilisée.

La seconde amène l'activation de la fonction normale d'interruption, autrement les pressions simultanées de <Control><Shift><Alternate> amèneront l'arrêt du programme.

La troisième agit de manière à ce que la fonction d'interruption soit inhibée jusqu'à la fin du programme ou jusqu'au prochain ON BREAK ou ON BREAK GOSUB. Il convient de manier précautionneusement cette instruction car dans les boucles perpétuelles, seules les conditions d'interruptions spécifiques insérées à cet effet et la combinaison de touches Reset dont l'utilisation n'est jamais souhaitable, sont en mesure d'assurer l'arrêt du programme.

Exemple :

```
ON BREAK GOSUB Arret        ! Branchement de la procédure Break
PRINT AT(1,1);"1er  niveau"
PRINT AT(1,1);"2ème niveau : <Control><Shift><Alternate>+<ESC>"
DO                          ! boucle perpétuelle
  IF Ex%=1                  ! atteinte du 2ème niveau?
    IF INKEY$=CHR$(13)      ! pression de <Return>?
      CLS
      PRINT AT(1,1);3ème niveau"
      PRINT AT(1,2);"Arrêt par <Control><Shift><Alternate>
      ON BREAK              ! activation de la fonction d'arrêt
    ENDIF
  ENDIF
LOOP
PROCEDURE Arret             ! la routine Arret ne s'appelle
  '                           qu'avec les touches d'interruption
  IF INKEY$=Chr$(27)        ! pression préalable d'<ESC>?
    Ex%=1                   ! mise en place du flag du niveau
! 3
    PRINT AT(1,1);"2ème niveau"
    PRINT AT(1,2);"3ème niveau par <Return>"
    ON BREAK CONT           ! inhibe la fonction BREAK
  ENDIF
RETURN
```

PROCEDURE (PRO)...RETURN (RET) Titre de procédure

ou

SUB (SU)...ENSUB (ENDSU)

ou

PROCEDURE [PRO]...ENDPROC [ENDP]

PROCEDURE Nom [(Liste_de_variables)]
...bloc d'instructions à exécuter
RETURN

Cette instruction permet la définition de sous-programmes. Ces derniers sont clos à l'aide de l'instruction de retour RETURN. RETURN assure le retour du programme dans la ligne d'appel et sa poursuite à la prochaine ligne.

Il n'est guère rationnel de disséminer et de répéter certains passages de programme identiques à l'intérieur du programme principal. Les procédures sont d'une utilité sans pareille dès qu'il s'agit d'appeler en plusieurs lieux du programme un même sous-programme.

La déclaration d'une liste de variables locales (Cf. LOCAL) réceptionnant les données transmises à l'aide de GOSUB, est optionnelle. Les variables de cette liste peuvent être de divers types du moins tant qu'ils correspondent à ceux des paramètres transmis.

Des caractères (A à Z) ou chiffres (0 à 9), le souligné ou le point sont utilisables dans le nom de la procédure. A l'inverse des noms de variables, l'initiale y peut être un chiffre (un conseil : évitez !).

L'appel d'une procédure par elle-même ou de quelconque procédures par une procédure est possible.

La définition, à l'aide de VAR, de variables à l'intérieur de la liste de paramètres, permet la transmission d'une variable globale (Cf. VAR). En outre, SUB est utilisable à la place de PROCEDURE, et ENDPROC ou ENSUB sont équivalents à RETURN.

Les applications de ces instructions sont disséminées dans l'ouvrage. En outre de plus amples informations sont consignées sous FUNCTION, LOCAL et GOSUB.

VAR — Transmission directe de variables

PROCEDURE Nom([Var1,...],VAR NomVariable1 [,NomVariable2$,...)]
FUNCTION Nom([Var1,...],VAR NomVariable1 [,NomVariable2$,...)]

En guise d'alternative aux variables pointeur indirectes, il est possible de transmettre directement des variables en une procédure ou une fonction, en les précédant du mot-clé VAR dans la liste des variables réceptrices locales de l'en tête d'une instruction FUNCTION ou PROCEDURE. Les noms des variables introduits par VAR représentent à l'intérieur de la fonction ou de la procédure les variables transmises par GOSUB ou FN. Il convient de veiller à ce que les variables VAR soient positionnées en fin de la liste de l'en tête. Tous les noms de variables placés derrière VAR, sont interprétés tels ceux de variables VAR.

La variable VAR est effectivement identique à la variable globale qu'elle représente sous un autre nom, celui indiqué dans l'en tête de la procédure. Celui-ci n'est utilisé que durant l'exécution de la procédure ou de la fonction. Chaque modification de la variable réceptrice à l'intérieur de la procédure se répercute directement sur le variable globale transmise.

Les contenus des variables globales portant les mêmes noms que les variables réceptrices, restent conservés. Le nom interne à la procédure ou à la fonction a de ce fait, jusqu'au retour dans le programme principal, un caractère strictement local. Toutefois, la variable globale de même nom n'est aucunement disponible à l'intérieur de la routine. Après le retour dans le programme principal (ou au niveau supérieur), le contenu global est de nouveau restauré.

Il convient de veiller à ce que les variables correspondantes soient de même type. Le non respect de cette correspondance, tout comme d'ailleurs la transmission d'une constante ou d'une expression en tant que paramètre, provoque l'apparition du message d'erreur: "Type VAR erroné".

6.6 Appel de programmes assembleur, C ou PRG

C:() Appel de routine résidente compilée en C

Var=C:Adresse_Routine ([, Liste_de_paramètres])

Suite à C:, est spécifiée l'adresse d'une routine résidente. L'adjonction d'une liste de paramètres mise entre parenthèses est optionnelle. La transmission des paramètres s'effectue d'après les conventions usuelles du C sur la pile. Lors de la spécification de paramètres de 32 bits, il convient d'introduire le paramètre par l'abréviation: L:, le format du mot (16 bits) étant celui par défaut.

Exemple :

```
VOID C:(L:Para1%,Para2%,W:Para3%,L:Para4%)
```

crée la pile suivante :

```
run:
move.l      (sp),return    ; = adresse du saut de retour
move.l     4(sp),para1     ; = 1er  paramètre Para1% (Long)
move       8(sp),para2     ; = 2ème paramètre Para2% (WORD)
move      10(sp),para3     ; = 3ème paramètre Para3% (WORD)
...etc.
...programme machine
rts                        ; n'utiliser uniquement que rts!!
```

Lors de l'absence de paramètres, l'adjonction de parenthèses vides() est requise. Suite au retour,le contenu de D0 (comme en C) est lisible (PRINT C:VAR()), transmissible dans une variable (A%=C:VAR()) ou combinable dans un test conditionnel (IF C:Var()).

CALL [CAL] Appel de routine compilée (assembleur)

CALL Adresse_Routine ([, Liste_de_paramètres])

En premier lieu s'énonce l'adresse de la routine machine dont l'appel est désiré. La notification d'une liste de paramètres séparés l'un de l'autre à l'aide de virgules est optionnelle. Ceux-ci peuvent être de divers types bien qu'ils soient interprétés sous forme de valeurs de 32

bits. Le programme machine réceptionne sur la pile après l'adresse du saut de retour, une valeur de 16 bits représentant le nombre de paramètres, puis une adresse de 32 bits suivis par les paramètres en format 32 bits notifiés à partir du BASIC. Les paramètres de chaînes sont transmis par le BASIC avec leur adresse de début.

Exemple :

```
CALL Prog%(17, "Para", A%, B$, *Var%)
```

crée la pile suivante :

```
run:
move.l      (sp),return    ; = adresse du saut de retour
move       4(sp),nombre    ; = 5 paramètres
move.l     6(sp),adresse   ; = pointeur sur le bloc de
                               paramètres
move.l   adresse,a1        ; = pointeur après a1
move    (a1),para1         ; = paramètre 1 en para1
move    4(a1),para2        ; = paramètre 2 en para2
...etc.
...programme machine
rts                        ; Veuillez n'utiliser que rts!!
```

EXEC { EXE } — Transmet des commandes CLI

EXEC Commandes$,In,Out

Cette instruction autorise la transmission d'une commande à l'interface CLI (Command Line Interface) afin de lancement d'un quelconque programme exécutable. La commande CLI est spécifiée en Commandes$. In et Out représentent les numéros des canaux (aussi dénommés filehandle) d'entrées et de sorties désirés. En règle générale, on y indique -1. Cette indication provoque l'ouverture d'une fenêtre CLI. Suite au lancement du GFA Basic à partir de l'interface CLI, la fenêtre active est utilisée. Le GFA Basic ayant été appelé à partir du Workbench, une nouvelle fenêtre CLI est ouverte. La redirection des présentes entrées/sorties sur un autre périphérique ou un fichier requiert la création d'un filehandle correspondant dans l'interface : CLI ou en GFA Basic à l'aide de la fonction DOS : Open (consultez à cette fin le cinquième chapitre réservé à la bibliothèque DOS) et puis

l'indication de celui-ci en In et Out. Dans la majorité des cas, cela n'est guère requis, car les programmes appelés se chargent de la sélection des canaux d'entrées/sorties.

En théorie n'importe quelle commande CLI est exécutable à l'aide d'EXEC. La chose la plus intéressante dans la pratique est leur appel à partir de programmes d'application. Deux modalités d'exécution se présentent suite à leur appel (elles valent naturellement aussi pour l'appel d'une commande CLI simple comme "List"):

- le GFA Basic attend la fin de l'exécution du programme
- le programme s'exécute en parallèle à GFA Basic. Pendant ce temps, le GFA Basic est naturellement utilisable.

La première modalité s'obtient par la seule spécification en Commande$ du nom du programme (ou selon, celle de son nom de chemin) dont l'appel est désiré.

Exemple :

```
EXEC "DF0:Utilities/Calculator",-1,-1
```

Suite au lancement du programme Calculator (disponible sur la disquette Workbench), GFA Basic attend jusqu'à ce que vous ayez fini vos calculs, à vrai dire jusqu'à la fermeture de la fenêtre du "Calculator". Alors seulement la poursuite du travail sous GFA Basic est possible.

La seconde modalité d'exécution résulte de l'insertion liminaire de Run.

Exemple :

```
EXEC "Run DF0:Clock",-1,-1
```

Cette ligne amène le lancement du programme Clock, disponible sur la disquette Workbench et son exécution parallèle à celle du GFA Basic.

Le nombre des programmes exécutés simultanément n'est limité théoriquement que par l'espace de mémoire vive disponible. Toutefois observez que chaque programme complémentaire mobilise des temps de calcul propres. Un ralentissement des programmes isolés en résulte.

MONITOR [M] Appel de routines machine

MONITOR [(Paramètre)]

Paramètre contient une valeur optionnelle qui est livrée en D0 à la routine appelée.

La commande MONITOR produit usuellement une Illegal Instruction Exception. Ceci est une erreur système apparaissant dès qu'à l'intérieur d'une routine machine est essayée l'exécution d'une commande inconnue de l'Amiga. Cette erreur résulte le plus souvent suite au pointage du "néant" par le PC (ProgramCounter = Pointeur CPU pointant la prochaine adresse à prendre en compte) dû à une erreur de programme.

Une Illegal Instruction Exception est dans les circonstances usuelles impossible en GFA Basic. Divers programmes détectent cette erreur et s'en servent afin de brancher ensuite un débogueur, un moniteur machine ou de quelque chose d'analogue. MONITOR se montre particulièrement adapté à l'appel de tels programmes.

Mais attention, lors du traitement d'une Illegal Instruction Exception, l'ordinateur se sert d'un vecteur (adresse mémoire 16) pour sauter à l'adresse du programme MONITOR. En règle générale, ce vecteur pointe sur la routine du système d'exploitation produisant les Guru-Meditations. En préalable à votre première utilisation de MONITOR, il convient que vous laissiez dérouter ce vecteur (de préférence par le programme lui-même) sur l'adresse du programme MONITOR, autrement il en résulterait en blocage de la machine.

RCALL [RC] Appel de routine machine avec accès de registre

RCALL Adresse, Matrice%()

Appelle une routine assembleur résidente. L'adjonction d'un nom de matrice de type entière de 32 bits (avec au moins 16 éléments) où l'affectation avant l'appel de longs mots est possible, et optionnelle. Les

contenus des 15 premiers éléments de cette matrice (avec OPTION BASE 0) sont copiés dans le registre CPU avant l'exécution de la routine:

```
Matrice%(0) jusqu'à matrice%(7)  ---> d0 jusqu'à d7
Matrice%(8) jusqu'à matrice%(14) ---> a0 jusqu'à a6
```

Suite à la sortie de la routine (à l'aide de RTS), les contenus de registre sont retournés dans le même ordre dans la matrice. La variable: matrice%(15) avec OPTION BASE 0 sinon matrice%(16) (OPTION BASE 1) retourne l'actuel pointeur de la pile utilisateur (Sp = a7 seulement en retour!).

RESTORE [RES] — Positionne le pointeur DATA

RESTORE [Nom_de_label]

RESTORE en l'absence de la notification d'un nom de label, positionne le pointeur DATA sur la première ligne DATA du programme. RESTORE Nom_de_label induit le pointage de la première donnée DATA suivant le label spécifié.

Les données DATA sont lues à partir de ce lieu jusqu'à la prochaine insertion de la commande RESTORE ou la lecture de la dernière donnée DATA disponible. De plus amples précisions à ce sujet, sont exposées sous DATA.

Chapitre 7

Les opérations de texte

7.1 Les manipulations de chaînes

MID$()= Affectation de portion de chaîne

MID$(Cible$, Début[,Nombre])="Texte"

Insère le texte spécifié dans la chaîne Cible$ à la position déterminée en Début. Le paramètre optionnel spécifie le nombre de caractères maximal de Texte à insérer dans la chaîne cible. La longueur initiale de la chaîne n'est modifiée en rien par la présente opération. Le nombre maximal de caractères de Texte insérables dépend étroitement de la longueur délimitée par la position de début d'insertion et celle de la fin de la chaîne cible.

Son application est illustrée sous FORM INPUT et FUNCTION.

LSET (LS) Affectation d'une chaîne alignée à gauche

LSET Cible$="Texte"

Texte est une chaîne ou variable de chaîne dont le contenu est copié aligné à gauche dans la chaîne cible. Lors de cette opération, la longueur initiale de la chaîne cible reste inchangée. Si la chaîne cible est plus courte que celle à insérer, cette dernière est tronquée. Si la chaîne cible est plus longue que le texte copié, elle sera complétée à l'aide de blancs.

Son application est illustrée dans le chapitre 5.5.1 : Les modalités fonctionnelles d'un fichier à accès direct.

RSET (RS) Insertion d'une chaîne alignée à droite

RSET Cible$="Texte"

RSET est en son principe analogue à LSET. Leur seule différence consiste en ce que RSET insère le texte à partir de la fin de la chaîne cible. Ces deux instructions ont un rôle prépondérant lors de l'utilisation de fichiers à accès direct, car elles permettent la modification des chaînes en l'absence de toute altération de la longueur des lignes d'insertion.

7.2 Analyse de chaînes

INSTR() Recherche de chaîne dans une chaîne

Var=INSTR(Cible$, Recherche$ [,Début])
Var=INSTR([Début,]Cible$, Recherche$)

Retourne une valeur spécifiant la valeur de la position absolue de la chaîne cherchée au sein la chaîne cible. Au cas où la chaîne cherchée ne serait pas trouvée, une valeur nulle est retournée. Si les les deux chaînes sont vides, la fonction de recherche retourne 1. L'emploi du paramètre optionnel Début permet de débuter la recherche à une position déterminée. Il convient d'observer que la présente recherche différencie les majuscules des minuscules.

Exemple :

```
cible$="Bizarre, bizarre, j'ai dit bizarre. Vous avez dit
bizarre ?"
recherche$="iz"
astring(0,cible$,recherche$,ibk$)
FOR i%=1 TO LEN(ibk$) STEP 2
  PRINT CVI(MID$(ibk$,i%,2))
NEXT i%
PROCEDURE astring(c.pos%,c.str$,r.str$,VAR rt$)
  ' Routine INSTR étendue, qui retourne toutes les positions
  ' d'une chaîne au sein d'une chaîne cible et délivre la
  'liste de ces positions sous forme de données MKI$.
  ' C.pos% = position de la chaîne où la recherche doit
  '          débuter. 0 y est interprété 1 comme dans INSTR.
  ' C.str$ = chaîne cible
```

```
  ' R.str$ = chaîne cherchée
  ' rt$    = variable VAR , elle retourne soit 0 (la chaîne
  ' étant introuvable), soit la liste des positions
  ' trouvées selon leur ordre.
  LOCAL x$,pos%
  REPEAT                                  ! boucle de recherche
    pos%=INSTR(r.pos,c.str$,r.str$)       ! détermine la
                                          ! position
    IF pos%>0                             ! chaîne contenue?
      x$=x$+MKI$(pos%)                    ! retourne la position
      r.pos=SUCC(pos%+1)                  ! la position de
                                          ! début +1
    ENDIF
  UNTIL pos%=0                            ! plus de chaîne
  rt$=x$
RETURN
```

LEFT$() Retourne les caractères de gauche d'une chaîne

Var$=LEFT$(Cible$ [,Nombre])

Retourne le premier caractère de la chaîne cible lors de l'omission du paramètre optionnel Nombre. Ce paramètre permet de spécifier le nombre de caractères de la chaîne. Si le nombre spécifié est plus élevé que la longueur de la chaîne, la chaîne est transmise intégralement dans la variable réceptrice. Si la chaîne cible est vide (""), une chaîne vide est livrée.

Son application est illustrée sous PRINT et sous DATA.

LEN() Retourne la longueur de la chaîne

Var=LEN(Var$)

Retourne la longueur de la chaîne spécifiée. Son application est illustrée sous FORM INPUT, INKEY$(), LINE INPUT, INSTR et en d'autres lieux de l'ouvrage.

MID$() Retourne une quelconque portion de chaîne

Var$=MID$(Cible$, Début,[,Nombre]

Délivre une partie quelconque de la chaîne cible spécifiée. Début est l'indice ou commence l'extraction de la chaîne. Nombre détermine le nombre de caractères à extraire à partir de la position Début. Lors de son omission ou de la notification d'un nombre de caractères à lire supérieur à la longueur de la portion de chaîne restante, toutes les lettres disposées à la suite de la position indiquée sont retournées.

Son application est présentée entre autres sous FORM INPUT et RIGHT$().

PRED() Retourne le prochain code ASCII inférieur

Var$=PRED(Expression$)

Délivre le code ASCII immédiatement inférieur à celui du premier caractère de l'expression indiquée. Cette fonction est utilisable comme abréviation de CHR$(ASC(Expression$)-1).

Exemple :

```
For i%=65 TO 76
   PRINT "Code ASCII du caractère ";CHR$(i%)
   PRINT " moins 1 = ";ASC(PRED(CHR$(i%)));
   PRINT " = caractère ";PRED(CHR$(i%))
NEXT i%
```

RIGHT$() Retourne les derniers caractères d'une chaîne

Var$=RIGHT$(Cible$ [,Nombre])

L'omission du paramètre optionnel Nombre entraîne la délivrance du dernier caractère de la chaîne transmise. Lors de l'utilisation de l'option, le nombre de caractères de la fin de chaîne spécifiés, sont retournés.

Exemple :

```
x$=" Ceci est une illustration de la présentation d'une chaîne "
x$=x$+" de coupure de mots. Dans la chaîne réceptrice (ici :
A$)"
x$=x$+" est re-tournée une liste de paire de valeurs MKI$"
x$=x$+" contenant dans l'ordre les posi-tions et les lon-gueurs"
x$=x$+" des chaînes extraites."
cut(x$,", -:.)",20,*a$)              ! Appel
FOR i%=1 TO LEN(a$) STEP 4           ! passage de chaîne MKI$
  '                                  ! en pas de 4
  spos%=CVI(MID$(a$,i%,2))           ! lecture d'une position
                                     ! de portion
  slen%=CVI(MID$(a$,i%+2,2))         ! lecture d'une longueur
                                     ! de portion
  PRINT MID$(x$,spos%,slen%)         ! édite la portion de
                                     ! chaîne
NEXT i%
PROCEDURE cut(c.str$,c.sgn$,c.lar%,c.vec%)
  ' Divise la chaîne transmise en portions selon une longueur
  ' maximale spécifiée, où une liste de traits d'union est
  ' affectée, ces derniers ayant priorité sur la longueur. La
  ' chaîne retournée reste inchangée.
  ' C.str$ =la chaîne transmise dont la coupure est désirée
  ' C.sgn$ =une quelconque liste de traits d'union. Les
  '         portions de chaîne finissent avec le premier
  '         signe disposé derrière le trait d'union. C.sgn$
  '         étant vide (""), C.lar% est la valeur exclusive
  '         de coupure.
  ' C.lar% =nouvelle longueur maximale de ligne
  ' C.vec% =pointeur sur une variable de chaîne réceptrice
  '         contenant à la fin du traitement une liste de
  '         paires de valeurs qui spécifient dans l'ordre les
  '         positions et les longueurs des portions de
  '         chaînes isolées.
  LOCAL c.dum$,cd.vec$,c.pos%,c.a$,c.j% ! adapte la portion
  ' sur la longueur
  c.pos%=1                           ! tampon des positions +1
  DO
    c.a$=LEFT$(c.str$,MIN(c.lar%,LEN(c.str$)))
    IF LEN(c.sgn$)>0                 ! présence du trait
                                     ! d'union ?
      FOR c.j%=LEN(c.a$) DOWNTO 1    ! recherche caractère par
        '                            ! caractère
        EXIT IF INSTR(c.sgn$,MID$(c.a$,c.j%,1)) ! abandon si
        '                            ! le trait d'union est
                                     ! trouvé
      NEXT c.j%                      ! prochaine caractère
    ENDIF
    IF c.j%=0                        ! Pas de trait d'union ?
      c.j%=c.lar%                     ! la longueur a priorité
    ENDIF
```

```
    c.dum$=LEFT$(c.a$,MIN(c.lar%,c.j%)) ! portion coupée
    c.str$=RIGHT$(c.str$,LEN(c.str$)-MIN((c.lar%),LEN(c.dum$)))
    '                                  ! raccourcit la variable
    '                                  ! de sortie
    cd.vec$=cd.vec$+MKI$(c.pos%)+MKI$(LEN(c.dum$))
    '                                  ! constitue les chaînes
                                       ! MKI$
    ADD c.pos%,LEN(c.dum$)             ! nouvelle position de
                                       ! début
    EXIT IF LEN(c.str$)<c.lar%         ! abandon si reste >
                                       ! longueur maximale
  LOOP
  cd.vec$=cd.vec$+MKI$(c.pos%)+MKI$(LEN(c.str$)) ! reste MKI$
  *c.vec%=cd.vec$                      ! retourne la chaîne MKI$
RETURN
```

RINSTR() Recherche de chaîne à rebours dans une chaîne

Var=RINSTR(Cible$, Recherche$ [,Début])
Var=RINSTR([Début,]Cible$, Recherche$)

Retourne la valeur de la position de la chaîne cherchée dans la chaîne cible. A l'inverse d'INSTR, la recherche est menée en sens arrière à partir de la fin de la chaîne. Pour de plus amples précisions, veuillez consulter INSTR.

SUCC() Retourne le prochain code ASCII supérieur

Var=SUCC(Expression$)

Livre le prochain code ASCII immédiatement supérieur de celui du premier caractères de la chaîne notée. Cette fonction est utilisable comme abréviation de CHR$(ASC(Expression$)+1).

Exemple :

```
a$="A"
WHILE a$<>"Z"
   PRINT a$,ASC(a$)
   a$=SUCC(a$)
WEND
```

7.3 Le formatage de chaînes

SPACE$() — Constitue une chaîne d'espaces

Var$=SPACE$(Nombre)
Retourne une chaîne contenant Nombre espaces.

STRING$() — Répète la chaîne

Var$=STRING$(Nombre,"Texte")
Var$=STRING$(Nombre, ASCII)

Constitue une chaîne où le texte spécifié est répété Nombre fois. Texte peut contenir une variable ou une expression de type chaîne. La multiplication d'un caractère s'obtient à l'aide de la spécification de son code ASCII. La chaîne retournée ne doit pas dépasser 32767 caractères (la longueur maximale d'une variable de chaîne).

Son application est illustrée entre autre sous PRINT.

TRIM$() — Elimine les espaces

Var$=TRIM$(Cible)

Elimine tous les espaces disposés en début ou en fin de la chaîne cible. Une chaîne vide ("") est retournée si la chaîne transmise se trouve constituée uniquement d'espaces ou est vide.

Exemple :

```
FOR i%=1 TO 4
  READ A$
  PRINT "Avant : ->";A$"<-"
  PRINT "Après : ->";TRIM$(A$);"<-"
NEXT i%
DATA    String 1   ,    String 2,String    3,String 4    ,
```

```
UPPER$()                    convertit les minuscules en majuscules
Var$=UPPER$(Cible$)
```

Convertit toutes les minuscules contenues dans la chaîne cible spécifiée en lettres majuscules (de codes ASCII: 65 à 90). Tous les autres signes ne sont pas affectés par cette conversion. Les accents sont aussi convertis.

Chapitre 8

Les instructions arithmétiques

8.1 Les opérateurs

Les opérateurs arithmétiques :

^	le symbole d'élévation à la puissance (s'obtient avec accent circonflexe plus espace)
-	le symbole de négation
*	le symbole de multiplication
/	le symbole de division
DIV	le symbole de division entière
MOD	le symbole du reste de la division entière
+	le symbole d'addition
-	le symbole de soustraction

Les opérateurs de comparaison :

=	le signe d'égalité
>	le signe supérieur
<	le signe inférieur
<> ou ><	le signe d'inégalité
=>	le signe égal ou supérieur

=< le signe égal ou inférieur

== le signe presque équivalent (égalité sur 28 bits)

Les opérateurs logiques :

AND conjonction : le résultat d'AND est vrai si les deux arguments sont vrais.

OR disjonction : le résultat d'OR est vrai, si l'un des arguments est vrai.

XOR ou exclusif : le résultat de XOR est faux, si les deux arguments sont tous deux vrais ou tous deux faux.

NOT négation : échange les valeurs de vérité en leur inverse.

IMP implication : l'implication d'IMP est fausse dès qu'une affirmation implique une négation

EQV équivalence : l'inverse de XOR. Le résultat est faux si les deux arguments se différencient l'un de l'autre.

Priorités :

() les parenthèses (la priorité la plus élevée)

^ élévation à la puissance

- le signe négatif

* / la multiplication et la division

+ - l'addition et la soustraction

= > <==
<> <= => les opérateurs de comparaison

NOT AND
OR XOR les opérateurs logiques (la priorité la moins élevée)
IMP EQV

8.2 Les opérations mathématiques

ADD [AD] — Instruction d'addition

ADD Var, Valeur

Additionne Valeur au contenu de la variable Var et délivre le résultat dans Var.
Si Var est un entier, le nombre réel éventuellement spécifié en Valeur, est convertit en nombre entier.

DEC — Décrémentation

DEC Var

Décrémente la variable VAR d'une unité.

DIV [DI] — Instruction de division

DIV Var, Valeur

Divise le contenu de la variable numérique Var par Valeur et dépose ensuite le résultat en Var. Idem que Var pour les réels.

INC [IN] — Incrémentation

INC VAR

Augmente la valeur de la variable numérique d'une unité.

MUL [MU] — Instruction de multiplication

MUL Var, Valeur

Multiplie le contenu de la variable numérique Var par Valeur et délivre ensuite le résultat en Var. Veuillez prendre en compte l'observation exposée sous l'instruction ADD.

SUB [SU] — Instruction de soustraction

SUB Var, Valeur

Soustrait Valeur du contenu de la variable numérique Var et dépose ensuite le résultat en Var. L'observation que nous avons faite pour l'instruction ADD est une nouvelle fois valable.

ADD() — Fonction d'addition entière

Var=ADD(Valeur1, Valeur2)

Additionne Valeur1 à Valeur2 et délivre le résultat entier correspondant.

Note: Contrairement aux instructions ADD, SUB, MUL et DIV, et à l'instar de toutes les autres fonctions, il est possible d'assigner les présentes fonctions, dans des éditions, des affectations, des expressions, des tests de conditions, etc. En outre seules des valeurs entières peuvent être ainsi calculées. Seule la partie précédent la virgule est prise en compte lors de l'addition de deux nombres réels. Les nombres n'y sont pas arrondis.

Les valeurs peuvent être indiquées sous formes de constante, de variable, d'expression ou de fonction, l'opération ne les altère aucunement. Les nombres réels sont réduits avant l'exécution de l'opération à leur partie entière.

Exemple :

```
PRINT MOD(ADD(DIV(100,33),MUL(345,SUB(12,5.6))),38)
```

correspond à :

```
PRINT INT(100/33)+INT(345*INT12-INT(5.6)) MOD 38
```

DIV() Fonction de division entière

Var=DIV(Valeur1, Valeur2)

Divise Valeur1 par Valeur2 et délivre le résultat entier correspondant. Veuillez prendre en compte l'observation qui a été faite pour la fonction ADD().

MOD() Fonction modulo entier

Var=MOD(Valeur1, Valeur2)

Calcule le reste entier de la division entière de Valeur1 par Valeur2 et délivre le résultat entier correspondant.

MUL() Fonction de multiplication entière

Var=MUL(Valeur1, Valeur2)

Multiplie Valeur1 par Valeur2 et délivre le résultat entier correspondant.

SUB() — Fonction de soustraction entière

Var=SUB(Valeur1, Valeur2)

Soustrait Valeur2 de Valeur1 et délivre le résultat entier correspondant.

8.3 Fonctions numériques

ABS() — Fonction de mise en absolu

Var=ABS(Argument)

ABS retourne Argument sous forme de valeur absolue, non signée. Cette valeur est toujours égale ou supérieure à 0.

EVEN() — Teste la parité d'un nombre

Var=EVEN(Argument)

Renvoie -1, si l'argument spécifié est pair, sinon 0.

EXP() — Fonction exponentielle

Var=EXP(Argument)

Retourne e à la puissance Argument, ce qui équivaut à 2.718281828462^Argument. Argument doit être dans tous les cas supérieur à 0 sinon un message d'erreur apparaît. EXP est la fonction inverse de LOG.

FIX() Fonction de nombre entier

Var=FIX(Argument)

Retourne la partie entière du nombre réel Argument. Cette fonction n'arrondit ni à l'unité supérieure ni à celle inférieure, elle supprime tout simplement les rangs décimaux. Les nombres négatifs s'en trouvent grandis : FIX(-12,33) donne -12. FIX est identique à TRUNC.

FRAC() Délivre les rangs décimaux

Var=FRAC(Argument)

Retourne la fraction décimale du nombre réel spécifié, autrement dit les chiffres au delà de la virgule. Argument étant un nombre entier, la valeur nulle: 0 est renvoyée.

INT() Fonction de nombre entier

Var=INT(Argument)

Convertit le nombre Argument en un nombre entier. Si Argument est un nombre réel, il est retourné arrondi à l'unité inférieure. A l'inverse de FIX() et de TRUNC(), les valeurs des nombres négatifs s'en trouvent agrandies.
INT(-12.33) donne -13 INT(33.17) donne 33

LOG() Fonction logarithmique

Var=LOG[10](Argument)

LOG() retourne le logarithme naturel (népérien) en base e de l'expression numérique : Argument, mise entre parenthèses.

```
base e => 2.71281828 => Nombre d'Euler
```

Le résultat de 2.718281828462^LOG(Argument) est donné par Argument. LOG10 délivre le logarithme décimal, dit de Briggs, d'Argument en base 10.

Le résultat de 10^LOG10(Argument) donne Argument. Argument est nécessairement supérieur à 0, sinon il en résulte un message d'erreur. LOG est la fonction inverse d'EXP().

ODD() — Teste l'imparité d'un nombre

Var=ODD(Argument)

Renvoie -1, si Argument est impair, sinon une valeur nulle : 0 est délivrée.

PRED — Communique le prochain nombre entier inférieur

Var=PRED(Argument)

Délivre le prochain nombre entier inférieur à Argument. Lors de l'assignation de nombres réels dans Argument, leur conversion en nombre entier est assurée avant l'exécution de l'opération (Cf. INT).

ROUND() — Arrondit un nombre

Var=ROUND(Argument [,Rang])

Arrondit Argument. Suite à la notification d'un rang précis de la fraction décimale, le nombre est arrondi au rang précisé. Lorsque le rang spécifié est négatif, le nombre est arrondi sur le rang correspondant précédant la virgule.

SGN() Teste le signe d'une valeur

Var=SGN(Argument)

Teste le signe d'Argument.

```
1 si Argument > 0, -1 si Argument < 0, 0 si Argument = 0
```

SQR() Fonction d'extraction de racine

Var=SQR(Argument)

Calcule la racine carrée (en anglais SQuare Root) d'Argument. Une extraction de racine plus élevée nécessite le détour par l'exponentiation fractionnelle :

```
racine cubique  = Argument^(1/3)
racine quatre   = Argument^(1/4)
et ainsi de suite...
```

Une valeur supérieur ou égale à 0 doit être affectée à Argument sinon il en résulte un message d'erreur.

SUCC() Retourne le prochain nombre supérieur

Var=SUCC(Argument)

Délivre la prochain nombre qui suit Argument. Suite à la notification d'un nombre réel en Argument, sa conversion en nombre entier est assurée automatiquement (Cf. INT).

TRUNC() — Fonction de nombre entier

Var=TRUNC(Argument)

TRUNC est identique à FIX. Pour de plus amples précisions, veuillez consulter la description de la dite fonction.

8.4 Les fonctions trigonométriques

ACOS() — Fonction arc cosinus

Var=ACOS(Argument)

Calcule l'arc cosinus d'Argument (Cf. ATN() pour de plus amples précisions). Veuillez noter que cette fonction renvoie pour l'instant des valeurs incorrectes et utilisez -ACOS(Argument) pour obtenir le résultat escompté. Toutefois cela sera modifié prochainement ...

ASIN() — Fonction arc sinus

Var=ASIN(Argument)

Calcule l'arc sinus d'Argument (Cf. ATN() pour de plus amples précisions).

ATN() — Fonction arc tangente

Var=ATN(Argument)

L'Argument est une valeur de tangente dont l'arc correspondant est calculé en radians. Une valeur contenue entre -PI/2 et +PI/2 en résulte. Si l'on nécessite une indication de l'arc en terme de degré, le résultat ainsi obtenu doit être multiplié par 180/PI, ou transformé en degrés à l'aide de DEG().

COS() Fonction cosinus

Var=Cos(Argument)

Calcule le cosinus d'Argument. Argument est une valeur spécifiée en radians. Si la valeur de l'arc n'est disponible qu'en termes de degrés, il faut d'abord le multiplier par PI/180, ou le convertir en radians à l'aide de RAD().

Exemple :

```
FOR i=0 TO 90 STEP 15
  PRINT "sinus:     ";I;SIN(I*PI/180);" degrés"
  PRINT "cosinus:   ";I;COS(I*PI/180);" degrés"
  PRINT "tangente: ";I;TAN(I*PI/180);" degrés"
NEXT i
FOR i=-PI+0.0000001 TO PI STEP PI/90
  PRINT "sinus:     ";I;SIN(I);" radians"
  PRINT "cosinus:   ";I;SIN(I);" radians"
  PRINT "tangente:   ";I;SIN(I);" radians"
NEXT i
```

Une autre exemple exposant l'emploi des fonctions COS() et SIN() illustre la fonction PI.

COSQ() Fonction cosinus (interpolation)

Var=COSQ(Degrés)

Retourne le cosinus (interpolation) en 16ème de degrés de la valeur spécifiée en Degrés (environ dix fois plus rapide que COS()).

DEG() Fonction de conversion en degrés

Var=DEG(Radians)

Convertit en degrés la valeur spécifiée en radians. Son résultat correspond à celui d'Argument*180/PI.

PI **La valeur pi**

PI

Une valeur constante de 3.1415926536 est contenue à cette variable réservée.

Exemple :

```
graphique : CBOX

yt%=2                                   ! diviseur de la résolution Y
DEFFILL ,2,4                            ! trame gris
FOR i%=0 TO 360 STEP 12                 !
  @cbox(2,320,200/yt%,250,i%)           ! boucle PBOX
NEXT i%                                 !
CLS                                     ! effacement de l'écran
FOR i%=0 TO 720 STEP 12
  ADD j%,3              ! rayon +3      ! boucle BOX
  @cbox(1,320,200/yt%,10+j%,i%)
NEXT i%
PROCEDURE cbox(mod%,xp%,yp%,rd%,an%)
  ' produit un carré pouvant être représenté en un
    quelconque 'angle et avec de quelconques grandeurs.
  ' Mod% = mode de représentation (1 = BOX, 2 = PBOX)
  ' Xp%, Yp% = coordonnées du point médian (le point de
              rotation)
  ' Rd% = la valeur de l'arc déterminé
  ' An% = angle d'inclinaison
  '
  LOCAL j%,i%,yt%,dg
  yt%=2                             ! diviseur de la résolution Y
  ERASE px%()                       ! efface le champ POLY-X
  ERASE py%()                       ! efface le champ POLY-Y
  DIM px%(4),py%(4)                 ! dimensionne les champs POLY
  FOR i%=-an%+45 TO -an%+360+45 STEP 90 ! un pourtour
    dg=i%*PI/180                    ! convertit en radians
    px%(j%)=xp%+(SIN(dg)*rd%*SQR(2)/2+0.5)      ! calcule
    py%(j%)=yp%+(COS(dg)*rd%/yt%*SQR(2)/2+0.5) ! les coordonnées
    '
    INC j%                          ! compteur des angles +1
  NEXT i%                           ! prochain angle
  IF mod%=1                         ! teste le mode de
                                    ! représentation
    POLYLINE 5,px%(),py%()          ! alors POLYLINE
  ENDIF
  IF mod%=2                         ! représentation PBOX?
```

```
      POLYFILL 5,px%(),py%()     ! alors POLYFILL
    ENDIF
  RETURN
```

RAD() — Fonction de conversion en radians

Var=RAD(Degrés)

Convertit en radians la valeur spécifiée en degrés.

SIN() — Fonction sinus

Var=SIN(Argument)

Calcule le sinus d'Argument (Cf. COS() pour de plus amples précisions).

SINQ() — Fonction sinus (interpolation)

Var=SINQ(Degrés)

Retourne le sinus (interpolation) en seizièmes de degré de l'angle spécifié en degrés (environ dix plus rapide que SIN()).

TAN() — Fonction tangente

Var=TAN(Argument)

Calcule la tangente de l'argument notifié (Cf. COS() pour de plus amples précisions).

8.5 Les opérations de comparaison

MAX() Etablit la valeur ou la chaîne supérieure

Var=MAX(Expression1, Expression2 [Expression3,...])
Var$=MAX(Expression1$, Expression2$ [Expression3$,...])

Donne la valeur la plus élevée d'un liste de valeurs ou la chaîne la plus grande d'une liste de chaînes. N'importe quelle expression numérique ou alphanumérique, valeur, chaîne ou variable est acceptée. Mais toutes les expressions spécifiées doivent être de même type.

Lors de l'indication de chaînes, les caractères des expressions spécifiées sont comparés selon leur position respective au sein de ces dernières. Les comparaisons s'effectuent de position à position en partant de la première, jusqu'à ce que la valeur ASCII d'un des caractères comparés soit distincte de celle de l'autre ou la fin de l'une des chaînes soit atteinte. Dans le premier cas, la chaîne la plus grande est celle dont le dernier caractère comparé possède la valeur ASCII la plus élevée. Dans le second cas, la chaîne la plus grande est celle la plus longue.

MIN() Etablit la chaîne ou la valeur inférieure

Var=MIN(Expression1, Expression2 [Expression3,...])
Var$=MIN(Expression1$, Expression2$ [Expression3$,...])

Retourne la valeur la moins élevée d'une liste de valeurs ou la chaîne la plus petite d'une liste de chaîne (pour de plus amples explications, veuillez consulter ci-dessus MAX()).

8.6 Les opérations de bits

AND() Fonction conjonctive

Var=AND(Valeur1,Valeur2)

Combine logiquement les deux valeurs spécifiées selon le mode AND et livre un résultat entier (Cf. chapitre 4: Les éléments de base du BASIC).

Les fonctions booléennes peuvent suppléer aux opérations logiques correspondantes :

```
AND(x,y)        correspond à    (x AND y)
 OR(x,y)        correspond à    (x OR  y)
XOR(x,y)        correspond à    (x XOR y)
EQV(x,y)        correspond à    (x EQV y)
IMP(x,y)        correspond à    (x IMP y)
```

Exemple :

```
PRINT &11101101 AND &X1111''''''AND(%11101101,%1111)
PRINT &HED OR &HF''''OR($ED,$F)
PRINT 237 XOR 15''''XOR(237,15)
```

Les énoncés contenant de telles fonctions, sont d'une lecture bien aisées pour le programmeur et d'une interprétation bien plus facile et rapide (un gain de vitesse d'environ 10%) par le BASIC.

Lors la description de la boucle IF...ENDIF, diverses combinaisons logiques furent intégrées à une même condition de branchement de boucle. Une forme conditionnelle souvent usitée est, par exemple :

```
IF ((v1%=w1 OR v2%=w2) AND v3%(ex1%+w2)) OR v4%<w2
```

A l'aide des fonctions de combinaisons, cette condition peut se formuler ainsi :

```
IF OR(AND(OR(v1%=W1,v2%<>w1),v3%>(ex1%+w2)),v4%<W2)
```

Ces mises en parenthèses facilitent fort l'analyse de la ligne d'instruction, n'est-ce pas ? Le léger défaut de ce genre de formulation est de gêner tant soit peu la vision des chaînes comparées.

L'énoncé :

```
IF v1%>w1 OR v2%<>w1 OR v3%<>v1% OR v4%=w2
```

s'écrit à l'aide des présentes fonctions :

```
IF OR(OR(OR(v1%>w1,v2%<>w1),v3%<>v1%),v4%=w2)
```

Ce n'est en fait qu'affaire de goût, rien n'empêche d'ailleurs d'utiliser la première énonciation.

BCHG() — Modifie un bit isolé

Var=BCHG(Valeur, Bit)

Modifie le Bit spécifié de Valeur (selon la formule: Valeur XOR 2^ Bit). Si le bit était activé (1), il s'en trouve inactivé (0), et inversement.

Veuillez observer pour ce qui est des fonctions concernant les bits isolés, que le décompte des bits commence à droite à partir du bit 0. Un quelconque bit peut être désigné dans Bit, toutefois le traitement d'un bit supérieur à 31 (la largeur maximale est de 32 bits) est impossible. Les valeurs supérieures à 2^32-1 (&HFFFFFFFF est la valeur la plus élevée possible) ne sont non plus traitées.

BCLR() — Inactive un bit isolé

Var=BCLR(Valeur, Bit)

Désactive le Bit désigné de Valeur (selon la formule: Valeur AND NOT 2^Bit). Si le bit était auparavant activé, il s'en trouve désactivé. Un bit déactivé le reste (pour de plus amples précisions Cf. BCHG().

BSET() Active un bit isolé

Var=BSET(Valeur, Bit)

Active le Bit désigné de Valeur (selon la formule: Valeur OR 2^Bit). Si le bit était auparavant désactivé donc représenté à l'aide de 0, il s'en trouve activé donc valant 1. S'il était 1, il reste 1, donc placé (pour de plus amples précisions Cf. BCHG()).

BTST() Teste l'état d'un bit

Var=BTST(Valeur, Bit)

Teste l'état du Bit désigné dans Valeur (selon la formule: -SGN (Valeur AND 2^Bit). Si le bit est activé, un -1, donc TRUE est retourné, sinon un 0, donc FALSE (pour de plus amples précisions Cf. BCHG()).

BYTE() Retourne l'octet de poids faible d'une valeur

Var=Byte(Valeur)

Délivre en l'absence de signe les 8 bits de faible poids de Valeur.

CARD() Livre le mot de poids faible d'une valeur

Var=CARD(Valeur)

Délivre en l'absence de signe les 16 bits de poids faible de Valeur.

EQV() Fonction d'équivalence

Var=EQV(Valeur1,Valeur2)

Combine logiquement les deux valeurs spécifiées en mode d'équivalence et livre le résultat entier (pour de plus amples précisions Cf. AND()).

IMP() Fonction d'implication

Var=IMP(Valeur1,Valeur2)

Combine logiquement les deux valeurs assignées en mode d'implication, et livre le résultat entier (pour de plus amples précisions Cf. AND()).

OR() Fonction de disjonction

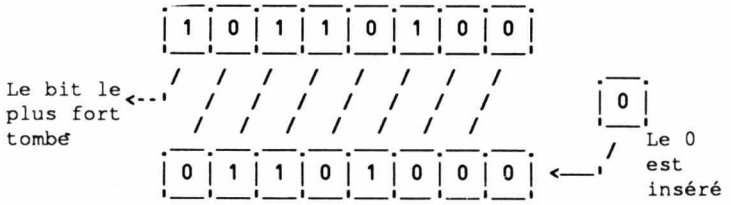
Var=IMP(Valeur1,Valeur2)

Combine logiquement les deux valeurs assignées en mode de disjonction et livre le résultat entier (pour de plus amples précisions Cf. AND()).

SHL() Décale logiquement les bits vers la gauche

Var=SHL(Valeur, Bits) Long-Shift-Left
Var=SHL&(Valeur, Bits) Word-Shift-Left
Var=SHL(Valeur, Bits) Byte-Shift-Left

Décalee le contenu de Valeur du nombre de Bits spécifié vers la gauche. Pour chaque bit à déplacer, Valeur est multipliée par 2. La fonction équivaut à la multiplication entière: Valeur*2^Bits. Le ou les bits libérés sont inactivés, annulés.

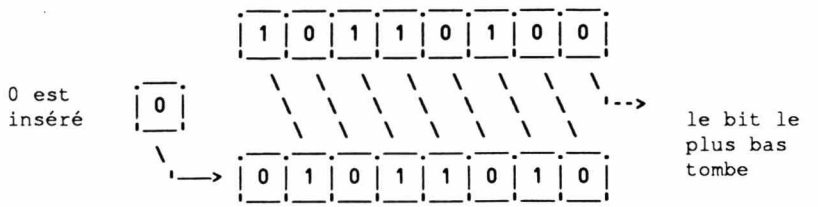


L'adjonction d'un & derrière SHL, amène le décalage des 16 premiers bits et celle de | celui des 8 premiers bits (LO-BYTE). Si la valeur assignée à SHL() est de plus de 8 bits, seuls les 8 bits de poids faible sont traités par l'opération. Il en va de même pour SHL&() (16 bits), bien que le bit 15 du résultat soit copié dans les bits 16-31 puisqu'il a valeur de signe.

SHR() Décale logiquement les bits vers la droite

Var=SHR(Valeur, Bits) Long-Shift-Right
Var=&SHR(Valeur, Bits) Word-Shift-Right
Var=SHR(Valeur, Bits) Byte-Shift-Right

Déplace le contenu de Valeur du nombre de bits notifié vers la droite. Pour chaque bit à déplacer, Valeur est divisée par 2. La fonction équivaut à la division entière : Valeur*2^Bits. Le ou les bits libérés sont inactivés, donc remplis de 0. Le signe des valeurs signées n'y est pas conservé.



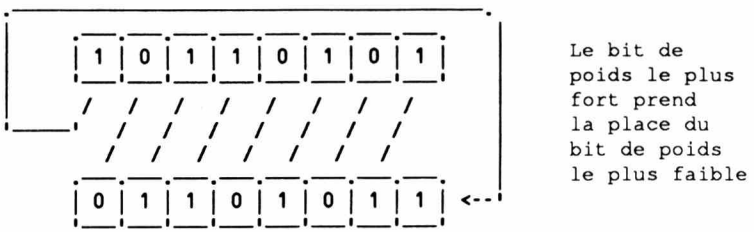
L'indication d'un & derrière SHR, amène le décalage des 16 bits de poids faible (LOW-WORD). Le | indique le décalage de l'octet le plus faible poids. La valeur assignée à SHR() étant de plus de 8 bits, seuls les 8

bits de poids faible sont délivrés par l'opération. Il en va de même pour SHR&() (16 bits), bien que le bit 15 du résultat soit copié dans les bits 16-31 puisqu'il a valeur de signe.

ROL() Rotation des bits vers la gauche

Var=ROL(Valeur, Bits) Long-Rotate-Left
Var=ROL&(Valeur, Bits) Word-Rotate-Left
Var=ROL(Valeur, Bits) Byte-Rotate-Left

Affecte une rotation au contenu de Valeur du nombre de Bits précisé vers la gauche. Chaque bit de droite se libérant est occupé par le bit décalé à gauche.

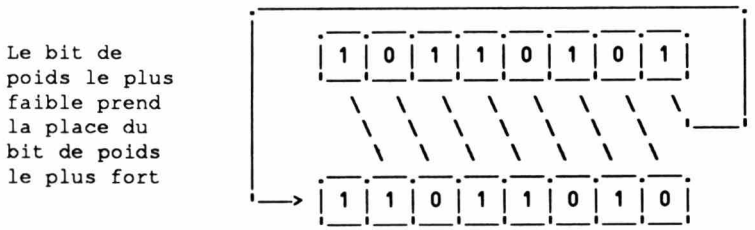


L'adjonction d'un & derrière ROL, amène la rotation du mot inférier et le | celle de l'octet le moins significatif. La valeur assignée à ROL() étant de plus de 8 bits, seuls les 8 bits les plus faibles sont livrés par l'opération. Il en va de même pour ROL&() (16 bits), bien que le bit 15 du résultat soit copié dans les bits 16-31 puisqu'il a valeur de signe.

ROR() Rotation des bits vers la droite

Var=ROR(Valeur, Bits) Long-Rotate-Right
Var=&ROR(Valeur, Bits) Word-Rotate-Right
Var=ROR(Valeur, Bits) Byte-Rotate-Right

Affecte une rotation au contenu de Valeur du nombre de bits notifié vers la droite. Chaque bit de gauche se libérant est occupé par le bit décalé à droite.



Lors de l'indication d'un & derrière ROR, seuls les 16 premiers bits (LO-WORD) sont affectés par la rotation ; l'affectation des 8 premiers bits étant marquée par un 1. Si la valeur assignée à ROR() est de plus de 8 bits, seuls les 8 bits de poids faible seront livrés par l'opération. Il en va de même pour ROR&() (16 bits), bien que le bit 15 du résultat soit copié dans les bits 16-31 puisqu'il concerne le signe.

SWAP Permute les mots de poids faible et fort

Var=SWAP(Valeur)

Fonction (Cf. instruction SWAP)

Permute le mot de poids faible - les bits 0 à 15 - de Valeur par celui de poids fort (HI(gh) Word) - les bits 16 à 31. Valeur y est traitée en tant que valeur entière de 32 bits. Lors de l'indication de nombre réels, les

chiffres disposés après la virgule sont avant l'exécution de l'opération supprimés (Cf. INT()). Des formats de valeurs inférieurs à 32 bits sont étendus en un mot long.

WORD() — Etend une valeur sur 32 bits

Var=WORD(Valeur)

Etend arithmétiquement Valeur en une valeur signée de 32 bits. Le fonction copie le bit 15 de Valeur dans les 16 premiers bits du résultat. Si le bit 15 (le seizième de gauche) de Valeur est activé, le résultat de WORD(Valeur) est négatif.

XOR() — Fonction de disjonction

Var=XOR(Valeur1,Valeur2)

Combine logiquement les deux valeurs spécifiées en mode XOR et livre le résultat entier (pour de plus amples précisions Cf. AND()).

8.7 Génération de nombres aléatoires

RAND() — Nombre aléatoire entier 16 bits

Var=RAND(n)

Retourne un nombre aléatoire entier 16 bits, donc sans signe, compris entre 0 (inclus) et n (exclus). Les valeurs d'n supérieures à 65535 sont recalculées à l'aide de n MOD 65535 sur l'intervalle considéré.

RANDOM — Nombre aléatoire 32 bits

Var=RANDOM(n)

Retourne un nombre aléatoire entier 32 bits compris entre 0 inclus et n exclus, n pouvant aussi être négatif.

RANDOMIZE — Initialise le générateur de nombres aléatoires

RANDOMIZE [(Start)]

Initialise le générateur des nombre aléatoires. Suite à l'emploi du paramètre optionnel Start, le générateur est initialisé avec cette valeur. L'utilisation répétée d'une même valeur Start ramène toujours la même série de nombres aléatoires. A l'initialisation RANDOMIZE est fixé aléatoirement. Vous pouvez retrouver une valeur aléatoire (après plusieurs RANDOMIZE) en exécutant RANDOMIZE TIMER.

RND() — Retourne un nombre réel (à 12 décimales) aléatoire

Var=RND[(Argument)]

Retourne Un nombre réel à 13 rangs de chiffres, compris entre 0 inclus et 1 exclus. Le paramètre Argument est optionnel et son omission ne porte aucunement à conséquence. Lors de son emploi, Argument vaut en tant qu'argument muet sans autre signification que celle-ci.

Exemple :

```
PRINT RND(0)*10+3 ! retourne un nombre réel aléatoire compris
                  ! entre 3 et 13
```

Chapitre 9

Les applications graphiques

9.1 Les définitions graphiques

BOUNDARY Commute la génération des formes graphiques

BOUNDARY Flag

Commute ou suspend la génération des formes propres aux objets graphiques dont l'initiale est la lettre P (PBOX, PCIRCLE, POLIFYLL, etc.). Flag étant égal à 1, la commutation est opérée et Flag étant égal à 0, suspendue.

Exemple :

La procédure suivante permet de générer des figures géométriques différenciées l'une de l'autre dès que vous modifiez tant soit peu, les paramètres de réglages et les indications de coordonnées. La matrice des coordonnées a été dimensionnée préventivement sur 60 points maximum, pour permettre des modifications futures de pas. Le traçage d'un triangle rectangle s'obtient ainsi par la simple modification du pas (STEP) de 120 en 90. Un hexagone régulier est obtenu suite au règlage du pas (STEP) sur 60 et aux modifications simultanées des indications de points de POLYLINE en 7 points et POLYFILL en 6 points.

```
DEFMARK 4                       ! sélection du symbole de
                                ! marquage
DEFFILL ,2,4                    ! remplissage gris
FOR i%=0 TO 360 STEP 20         ! un tour entier par pas de 20
  BOUNDARY 0                    ! suspend la génération des
                                ! formes
  @ctri(1,120,110,100,i%)
  @ctri(2,120,110,60,i%)
  BOUNDARY 1                    ! commute la génération des
                                ! formes
  @ctri(2,520,110,60,i%)
NEXT i%
```

```
PROCEDURE ctri(mod%,xp%,yp%,rd%,an%) ! génère en Hires un
  ' triangle représentable en une grandeur et un angle
  ' quelconques.
  ' Mod% = mode de représentation ( 1 = mode contour, 2 =
  ' mode remplissage).
 La communication des coordonnées de l'angle, en l'absence de
  ' tout traçage de la figure à l'écran lors de l'exécution
  ' résulte de mod%=0. Les coordonnées de l'angle peuvent
  ' être lues en retour à partir de Px%(0)/Py%(0) à
    Px%(2)/Py%(2).
  ' Xp%/Yp% = coordonnées du point médian
  ' Rd%      = valeur de l'arc en radians
  ' An%      = angle d'inclinaison
 LOCAL j%,i%,yt%                ! variables locales
  yt%=1                         ! diviseur Y de résolution
  ERASE px%()                   ! efface la matrice POLY-X
  ERASE py%()                   ! efface la matrice POLY-Y
  DIM px%(4),py%(4)             ! dimensionne les matrices
                                ! POLY
  FOR i%=-an% TO -an%+360 STEP 120 ! un tour complet
    px%(j%)=xp%+(SIN(i%*PI/180)*rd%+0.5)      ! calcule ...
    py%(j%)=yp%+(COS(i%*PI/180)*rd%/yt%+0.5) ! les
                                              ! coordonnées
    INC j%                      ! compteur d'angles +1
  NEXT i%                       ! prochain angle
  IF mod%=1                     ! le contour uniquement?
    POLYLINE 4,px%(),py%()      ! alors dessine les contours
  ENDIF
  IF mod%=2                     ! remplissage de la figure?
    POLYFILL 3,px%(),py%()      ! alors trame la figure
  ENDIF
RETURN
```

COLOR (C) — Définition de la couleur des lignes

COLOR Premier_plan, Arrière_plan, Contour

Détermine la palette des couleurs dans laquelle les fonctions graphiques traçant les points ou les lignes puisent leur couleur. En Premier_plan est indiqué le numéro de la couleur dont vous désirez revêtir les lignes tracées, la seconde couleur à indiquer est celle de l'Arrière_plan. En Contour est portée la couleur dont vous souhaitez voir coloriés les contours de vos figures graphiques dessinées à l'aide d'instructions débutant avec la lettre P.

L'étendue de la palette des couleurs dépend entièrement de la résolution de votre écran. L'étendue maximale est comprise entre 0 et 63 teintes possibles, mais avec des résolutions supérieures ou un nombre de BitPlanes moindre, elle se restreint :

D'ordinaire les règles suivantes prévalent :

```
1 BitPlane  offre    2 couleurs
2 BitPlanes offrent  4 couleurs
3 BitPlanes offrent  8 couleurs
4 BitPlanes offrent 16 couleurs
5 BitPlanes offrent 32 couleurs
6 BitPlanes offrent 64 couleurs
```

Toutefois les prochaines observations sont à prendre en compte :

- en haute résolution : Hires (640 x 256 points), seuls 4 BitPlanes, donc 16 couleurs, sont disponibles implicitement

- l'utilisation de plus de 4 BitPlanes requiert des astuces particulières.

Certes l'on pourrait se demander à quoi bon les couleurs comprises entre 32 et 63? Celles-ci sont requises pour deux modes graphiques particuliers de l'Amiga. Il s'agit des modes EXTRA_HALFBRITE et HAM, dont les descriptions respectives sont exposées sous OPENS.

Résolution		**BitPlanes**	**Couleurs**
Hires	(640/256)	4	0 ... 16
Hires entrelacé	(640/512)	4	0 ... 16
Lowres	(320/256)	5	0 ... 31
Lowres entrelacé	(320/512)	5	0 ... 31
HAM	(320/256)	6	0 ... 63 (4096 couleurs implicites)
HAM entrelacé	(320/512)	6	0 ... 63 (4096 couleurs implicites)
EHB	(320/256)	6	0 ... 63 (32 avec 2 niveaux d'intensité)
EHB entrelacé	(320/512)	6	0 ... 63 (32 avec 2 niveaux d'intensité)

A ces diverses résolutions, correspondent des couleurs propres présélectionnées lors de l'initialisation de la machine. Le contenu des registres des couleurs est déterminable à l'aide de l'instruction SETCOLOR. Une illustration de cette détermination est d'ailleurs exposée à la suite de la description de cette dernière.

DEFFILL { DEFF } Détermine la trame de remplissage

DEFFILL [Couleur],[Style],[Motif]
DEFFILL [Couleur],Motif$

Détermine la couleur, le style et le motif de la trame de remplissage pour les instructions graphiques P. Elle instruit la définition d'une trame. Le paramètre Couleur est précisé sous COLOR.

```
Style                                  motifs de trame
_____________________________________________________

0 = couleur d'arrière plan      ==>    inexistant
1 = couleur de premier plan     ==>    inexistant
2 = pointillé                   ==>     2 à 24
3 = strié                       ==>     1 à 12
4 = définitions personnelles    ==>     ceux del'Amiga ou
                                ==>     ceux du programmeur
```

Les paramètres dont la modification n'est pas désirée, peuvent être omis néanmoins les insertions de leur séparateur: la virgule, sont requises. La formulation DEFFILL Couleur,Motif$ permet de déterminer une trame personnelle. A cette fin, l'affection d'une chaîne de 16 mots (ou selon, de 32 ou de 64 mots) en Motif$ est nécessitée. Celle-ci contient en format MKI$, le motif binaire de chacune des 16 lignes du motif insérées selon l'ordre correspondant.

Exemple 1 :

```
F.Motif$=MKI$(&X1111111111111111)
F.Motif$=F.Motif$+MKI$(&X1000000000000000)
F.Motif$=F.Motif$+MKI$(&X1000000110000000)
F.Motif$=F.Motif$+MKI$(&X1000001001000000)
F.Motif$=F.Motif$+MKI$(&X1000010000100000)
F.Motif$=F.Motif$+MKI$(&X1000100000010000)
F.Motif$=F.Motif$+MKI$(&X1001000110001000)
F.Motif$=F.Motif$+MKI$(&X1010001111000100)
F.Motif$=F.Motif$+MKI$(&X1001000110001000)
```

```
F.Motif$=F.Motif$+MKI$(&X1000100000010000)
F.Motif$=F.Motif$+MKI$(&X1000010000100000)
F.Motif$=F.Motif$+MKI$(&X1000001001000000)
F.Motif$=F.Motif$+MKI$(&X1000000110000000)
F.Motif$=F.Motif$+MKI$(&X1011111111111110)
F.Motif$=F.Motif$+MKI$(&X1001111111111100)
F.Motif$=F.Motif$+MKI$(&X1000000000000000)
DEFFILL 1,F.Motif$
PBOX 100,100,200,200
```

L'affectation d'un bloc de 16 mots pour chaque BitPlane disponible, est possible en Motif$. Les mots ordonnés de 1 à 16 représentent alors le motif binaire du premier BitPlane, ceux ordonnés de 17 à 32 celui du second BitPlane, ceux échelonnés de 33 à 48 celui de troisième BitPlane, et ainsi de suite...

Exemple 2 :

```
GRAPHMODE 2
FOR i%=0 TO 7                  ! crée un
  DEFFILL i%,3,i%              ! un motif
  PCIRCLE 16,16,16             ! binaire
  DEFFILL i%+2,3,i%+2          ! quelconque
  PCIRCLE 48,16,16
NEXT i%
@defill3(7,7,df$)              ! appel (propre à la version
                               ! 3.0)
DEFFILL 1,df$                  ! édite le motif de trame
PBOX 15,64,192,192             ! nouvellement créé
GRAPHMODE 3                    ! mode XOR
@dfill3(42,7,df$)              ! appel (propre à la version
                               ! 3.0)
DEFFILL 1,df$                  ! édite le nouveau motif
PBOX 105,14,192,112            ! de trame (avec
                               ! enchevêtrement)
  'PROCEDURE dfill3(d.x%,d.y%,VAR d.ff$)  ! seulement version
3.0
  ' copie une zone quelconque de 16x16 points de l'écran
 1,df$                 ! édite le nouveau motif
PBOX 10
  ' variable de chaîne
  ' d.x% = le point d'origine sur l'abscisse X
  ' d.y% = le point d'origine sur l'ordonnée Y
  ' d.ff% = la variable de chaîne contenant le motif de trame
  ' suite à l'exécution de l'instruction
  LOCAL d.fr$,dc1%,dc2%,d.f$,xb%      ! variables locales
  GET d.x%,d.y%,dx%+15,d.y%+15,d.fr$ ! sauvegarde la zone écran
  d.f$=RIGHT$(d.fr$,32)               ! transmet le motif
  d.ff$=d.f$                          ! restitue du motif
RETURN
```

DEFLINE (DE) — Détermine la forme des traits

DEFLINE [Style]

Cette instruction permet d'affecter aux commandes traçant des lignes ou contour (BOX, LINE, CIRCLE, DRAW) des styles de trait déterminés.

```
Styles :
        0  trait en couleur d'arrière plan
        1  trait continu                    ____________________
        2  trait brisé 1                    ____ ____ ____ ____
        3  trait pointillé                   _ _ _ _ _ _ _ _ _ _
        4  trait brisé/pointillé            __ _ __ _ __ _ __ _ __
        5  trait brisé 2                    __  __  __  __  __
        6  trait brisé doublement          ___ __ _ ___ __ _
           pointillé

de    - 1 styles
à - 32767 de l'utilisateur
```

La définition personnelle d'un style de trait nécessite l'établissement d'un motif binaire de 15 bits, où chaque bit représente un point du trait. Ce nombre est à transmettre en tant que valeur signée négative. Ces modifications de style ne valent que pour une épaisseur de trait 1. Toutes les autres épaisseurs de traits sont dessinées en tant que trait complet.

Quant au choix des extrémités des traits, ils s'énumèrent comme suit :

```
0 = bout pointu
1 = terminaison en forme de flèche
2 = bout arrondi
```

Exemple :

```
FOR i%=1 TO 6
  DEFLINE i%
  LINE 200,60+i%*16,300,60+i%*16
  PRINT AT(40,8+i%*2);i%
NEXT i%
DEFLINE -18149      ! -18149 = -&X100011011100101
FOR i%=1 TO 90 STEP 4
  BOX 10+i%,10+i%,190-i%,190-i%
NEXT i%
```

DEFMOUSE[DEFMO] Sélectionne la forme du pointeur désirée

DEFMOUSE Forme
DEFMOUSE Souris$

Appel d'une forme de représentation du pointeur de la souris définie personnellement, ou présélectionnée par le système.

```
Les symboles disponibles sont :
0 = la flèche          1 = le double XX (le curseur de texte)
2 = le sablier         3 = le doigt pointant
4 = la main ouverte    5 = la croix fine
6 = la croix épaisse   7 = la croix avec contour
```

En lieu et place de Forme, il est possible d'adjoindre en Souris$, une variable de chaîne dont le contenu en format MKI$ spécifie la forme que devra revêtir le pointeur de la souris.

Mot 1 :
Abscisse du point d'origine

Mot 2 :
Ordonnée du point d'origine (Toutes les actions futures de la souris (SourisX et SourisY) référeront à ce point d'origine.)

Mot 3 :
Toujours MKI$(1)

Mot 4 :
Couleur du masque (arrière plan de l'image du pointeur) blanc = MKI$(0), noir = MKI$(1)

Mot 5 :
Couleur du pointeur de la souris

Mots 6 à 21:
Lignes de motif 16 bits du masque du pointeur

Mots 22 à 37:
Lignes de motif 16 bits de l'image du pointeur

La souris peut revêtir quatre couleurs différentes dans l'Amiga.

Exemple :

```
OPENW 0          ! ouverture d'une fenêtre d'édition
PAUSE 20         ! réserve du temps pour le cliquement
FOR i=0 TO 8
  DEFMOUSE i     ! affecte une nouvelle forme au pointeur
  PAUSE 40       ! réserve du temps pour l'apparition du pointeur
NEXTi%
```

GRAPHMODE [GR] — Sélectionne le mode graphique

GRAPHMODE Mode

Mode spécifie le mode opératoire auquel se conforment les représentations graphiques sur l'arrière plan existant.

0 = JAM1 (transparent)
Les élément graphiques (PBOX, LINE) utilisés y sont intégralement dessinés. Toutefois la sortie de texte se limite aux lignes nécessitées pour la représentation. L'espace autour du texte n'en s'en trouve pas affecté.

Tout nouveau point s'y représente sous son propre masque OR en tant que tel.

1 = JAM2 (replace)
La surface de l'élément graphique employé (PBOX, LINE, etc.) y est peinte. Tout objet recouvert par cette surface peinte, à vrai dire tramée, est superposé et remplacé. Lors de l'édition de texte, l'arrière plan sur lequel s'insère le texte est peint avec la couleur d'arrière plan.

```
Tout nouveau point se représente sous son propre masque AND en
tant que tel.
```

2 = COMPLEMENT (image négative)
Tout les points de l'image n'ayant pas été tracés auparavant, y sont tracés, et inversement. Seule l'image dessinée est affectée par le présent mode. Cette affectation est analogue à celle opérée en JAM1.

3 = INVERSEVID (vidéo inverse)
Ce mode d'affichage effectue l'édition, par exemple d'un texte, en vidéo inverse. Autrement dit là où devrait être exposé le texte, le fond graphique est effacé. Et là où en règle ordinaire apparaissait l'arrière plan, règne l'obscurité (l'Amiga y place les points).

Tous ces modes de représentation graphiques se laissent combiner ensemble. L'addition de deux identificateurs de mode amène la mise à disposition d'un nouveau mode graphique. Toutefois certaines limites existent ici aussi. C'est pour cette raison que 0 fut affecté à JAM1. Comme vous vous en doutez: l'addition d'une valeur de mode à 0 ne modifie en rien la valeur additionnée. Néanmoins la combinaison des modes JAM2 et COMPLEMENT est utilisable et induit de possibilités de création graphique fort intéressantes.

Le prochain programme de dessin commute successivement les 4 modes graphiques ci-décrits :

```
DEFFILL ,2,5
PBOX 10,10,200,90
BOX 8,8,202,124
TEXT 36,118,"G R A P H M O D E"
DEFFILL ,2,4
FOR i%=1 TO 4
  GRAPHMODE 2^(i%-2)
  PBOX i%*50-35,22,i%*50-20,160
  GRAPHMODE 0
  TEXT i%*50-18,158,INT(2^(i%-2))
NEXT i%
```

SETCOLOR [SET] — Définition de la palette des couleurs

SETCOLOR Registre, Rouge, Vert, Bleu
SETCOLOR Registre, Nuance

La couleur dont le registre, celui de l'écran en cours, est spécifié, peut être définie soit en tant que ton de vert, rouge et bleu (de 0 à 15) ou en soit tant que nuance des trois couleurs (de 0 à 4095).

```
Nuance = (proportion de rouge*256)+(proportion de
vert*16)+(proportion de bleu)
```

En certaines occasions, l'indication d'une nuance définie peut se montrer bien plus intéressante que l'emploi simultané des trois paramètres indépendants. Le GFA Basic distingue d'ailleurs le nombre précis de paramètres usités. Lors de la seule indication du paramètre Nuance, la composition de la nuance s'ordonne comme suit :

Suite aux indications par exemple de 3 pour le bleu, 2 pour le rouge et 5 pour le vert, s'effectue le calcul de la valeur totale de la nuance. La machine multiplie la proportion de rouge par 256 (2^8), celle de vert par 16 (2^4) et celle de bleu par 1 (2^0) et ensuite additionne les résultats précédents. En notre illustration, la valeur de la nuance à indiquer est de 595.

L'utilisation de chiffres hexadécimaux simplifie l'opération comme le montre la suite :

```
      SETCOLOR 1,&H253
ou
      A=&H253                    correspond à
      SETCOLOR 1,A               SETCOLOR  1,2,5,3
ou
      A$="&H"+"2"+"5"+"3"
      A=VAL(A$)
      SETCOLOR 1,A
```

4096 couleurs différentes (3 couleurs primaires et 16 tons (ou niveaux de gris) = 16*16*16 = 4096 couleurs) sont ainsi disponibles.

Les bits 0 à 7 de Nuance déterminent la proportion de rouge, les bits 8 à 15 la proportion de vert, et ceux 16 à 23 la proportion de bleu.

Lors de sorties standard dans la fenêtre d'édition du GFA Basic sur le Workbench , les 4 registres de couleurs connaissent une affection déterminée.

```
registre 0 = couleur de fond (bleu)
registre 1 = couleur du texte dans la fenêtre d'édition (blanc)
registre 2 = couleur du texte dans la fenêtre de l'éditeur
             (noir)
registre 3 = couleur du texte des messages d'erreur (rouge)
```

Malheureusement le cercle goethéen des couleurs ne nous est ici d'un moindre secours puisque le principe de soustraction des trois couleurs primaires rouge, bleu, jaune n'est pas de mise. L'observation de ce

principe de sélection de couleurs demanderait la soustraction du bleu du vert afin d'obtention du jaune. Cela mis à part, il nous permettra d'illustrer notre propos :

```
        rouge
violet      orange
        blanc
 bleu       jaune
        vert
```

Un peu de physique des couleurs :

Il existe trois couleurs primaires : rouge, bleu, jaune (en langage professionnel : magenta, cyan, jaune).

Le mélange de deux couleurs primaires à volume égal donne la couleur complémentaire (celle inverse) de la couleur primaire restante.

```
Le mélange des couleurs primaires rouge et bleu donne le violet
(la couleur complémentaire du jaune).

Le mélange des couleurs primaires rouge et jaune donne l'orange
(la couleur complémentaire du bleu).

Le mélange des couleurs primaires jaune et bleu donne le vert
(la couleur complémentaire du rouge).

Le mélange des couleurs primaires rouge, bleu et jaune donne le
blanc.
```

Pour mémoire, le noir est l'absence de couleurs.

Ce programme illustratif produit un jeu de couleurs que nous espérons suggestif. Les personnes intéressées y trouveront certainement motif à exercice ou à développements futurs. Les couleurs y sont pulsées avec une fréquence d'environ 58 cycles par minute (ce qui équivaut à la fréquence du battement du pouls lors de l'énervement). Cette fréquence est aisément modifiable dans la boucle d'attente placée dans le dernier bloc d'instructions.

9.2 Les instructions d'éléments graphiques

BOX, PBOX (BO, PB) — Trace, peint un rectangle

[P]BOX X_gauche,Y_haut, X_droit, Y_bas

Les axes X_gauche/Y_haut et X_droit/Y_bas représentent les diagonales du rectangle à dessiner en mode trait (BOX) ou à remplir avec l'actuel motif de trame de DEFFILL (PBOX).

Toutefois si le grain de la trame désirée est d'une taille trop élevée (par exemple en vue de tramer le fond d'une loupe), il convient si possible de remplacer l'instruction PBOX par PUT. La vitesse du remplissage opérée par cette dernière est bien plus rapide. Avec des trames de grains peu volumineux, la différence de vitesse de tramage n'est pas significative. Mais avec de gros grains, le temps de tramage s'en trouve doublé. La petite routine suivante en apporte la démonstration. Elle agrandit une quelconque zone de l'écran et l'affecte en une zone de mémoire afin de conservation pour un usage futur à titre d'échantillon de trame.

L'observation d'une petite condition préalable est requise. D'abord doit être opérée la détermination d'une zone quelconque de l'écran, zone où viendra s'afficher l'image grossie. En outre il vous est possible de spécifier la hauteur et la largeur du grain de trame. Il est déconseillé de recouper la zone d'édition de l'image initiale avec celle de son grossissement. La zone d'édition de l'image grossie est limitée dans la routine, à l'écran visible.

Suite à l'appel de la routine, l'image grossie est dessinée à l'écran.

Puis la routine attend la modification du contenu de la loupe opérée à l'aide de la pression du bouton gauche de la souris. La loupe est abandonnée dès que le bouton droit de la souris est cliqué. Ensuite le fond de l'écran recouvert par la loupe, est restauré.

La modification du contenu de la loupe est ordonnée par le cliquement sur le point de trame désiré. La couleur du point cliqué est modifiée automatiquement. Cette couleur lui est aussi affectée lors du déplacement du pointeur de la souris jusqu'au prochain relâchement de la touche de la souris. Chaque cliquement de la souris sur un point de

la loupe, amoindrit son indice de couleurs d'une unité. Lors d'un quadruple cliquement dans le Workbench sur le même point, ce dernier revêt tour à tour les quatres couleurs possibles. Il est conseillé de ce fait, lors d'un cliquement d'un point de la loupe, de maintenir le bouton appuyé le temps requis par la visualisation de la couleur délivrée. Il vous est possible, le temps durant de l'appui du bouton de la souris, de dessiner à l'intérieur de la loupe avec la couleur ainsi sélectionnée. Dès que la touche est relâchée, l'indice de la couleur du point de la loupe pointé par la souris, est amoindri d'une unité. Le Flag étant noté 1, la zone initiale de l'écran est en toute simultanéité affectée par les modifications survenues dans la loupe. Suite à la sortie de la procédure, le contenu actuel de la loupe est enregistré dans la matrice larr% (largeur de la trame,hauteur de la trame) et peut être de ce fait employé hors de la procédure.

Si la représentation de la loupe à l'écran, vous semble superflue, il vous suffit de noter 2 en Flag. En ce cas, seule la matrice larr% est remplie par la zone de l'écran indiquée. L'index de la matrice correspond à la position des points dans la trame. L'indexation débute avec O, par exemple : larr%(0,0) contient la valeur de la couleur du premier point en haut à gauche de la zone écran et larr%(12,6) la valeur de la couleur du point aux coordonnées de trame 12 points à partir de gauche et 7 points à partir du haut. Il est logiquement concevable qu'une telle matrice - dépendant de la taille d'une trame - requiert une quantité de mémoire élevée. Une trame de 50*50 points nécessite 10000 octets (50*50*4).

Des précisions complémentaires sont exposées dans les commentaires adjoints à la procédure.

```
' Préparatifs propres aux deux exemples d'application :'

FOR i%=0 TO 100              ! 100
  DEFFILL MAX(1,RANDOM(2^2)),2,RANDOM(22)
  x%=RANDOM(590)             ! peint
BOX avec...
  y%=RANDOM(200)             ! ...une couleur, une trame...

BOX x%,y%,x%+50,y%+50        ! ...et une position aléatoires
NEXT
'
' Exemple d'application 1 :
'
PRINT "Appelez la loupe à l'aide du bouton droit de la souris"
DO                            ! boucle perpétuelle
  GRAPHMODE 3                 ! XOR mode pour la boîte à images
```

```
  MOUSE xx%,yy%,k%          ! cherche le statut de la souris
  xx%=MAX(30,xx%)           ! limite la position Y de la
                            ! souris
  yy%=MAX(30,yy%)           ! limite la position X de la
                            ! souris
  BOX xx%-1,yy%-1,xx%+20,yy%+20 ! trace la boîte...
  BOX xx%-1,yy%-1,xx%+20,yy%+20 ! à images
  IF MOUSEK=2               ! le bouton droit est-il pressé ?
    GRAPHMODE 1             ! commute en mode REPLACE
    loupe(xx%-40*6,yy%-40*6,20,20,6,6,1) ! appel de la loupe
  ENDIF
LOOP
 UNTIL MOUSEK=3
CLOSE #1
END
'
' Exemple d'application 2 (effacer au préalable l'exemple 1):
'
PRINT "Affecte l'arrière plan de la souris (20*20) dans la
matrice larr%"
PRINT "Veuillez presser un quelconque bouton de la souris!"
REPEAT                          ! boucle d'attente...
UNTIL MOUSEK                    ! ...de la pression du bouton
6,MOUSEY-20*6,20,20,6,6,2) ! appel de la loupe
CLS                             ! effacement de l'écran
FOR i%=0 TO 20-1                ! 20 lignes
  FOR j%=0 TO 20-1              ! 20 colonnes
    COLOR larr%(i%,j%)          ! définit la couleur des grains

LOT 100+i%,100+j%          ! trace les grains de la trame
  NEXT j%
NEXT i%
'
PROCEDURE loupe(xp%,yp%,lr%,hr%,sx%,sy%,flg%)
  ' pour Hires/Midres/Lowres
  ' xp% = coordonnée X du cadre de la loupe
  ' yp% = coordonnée Y du cadre de la loupe
  ' lr% = largeur de la trame à agrandir
  ' hr% = hauteur de la trame à agrandir
  ' sx% = largeur du grain de trame grossi
  ' sy% = hauteur du grain de trame grossi
  ' flg% = flag
  '    0 = la loupe est activée et son contenu est
  ' modifiable, le dessin original restant
  ' Inchangé.
  ' 1 = la loupe est activée et son contenu est
  ' modifiable, les modifications entreprises dans la
  ' loupe se répercuteront dans le dessin original.
  ' 2 = seule la matrice est remplie avec les grains de
  ' couleurs de la trame originale. La loupe n'y est
  ' pas activée.
  ' La matrice de la trame est aussi complétée en mode 0 et
```

```
    ' 1 et peut être évaluée suite à la sortie de la routine
    '
    LOCAL i%,j%,lb$,mx%,my%,mk%,xp2%,yp2%,pt%
    LOCAL pnt%,mxx%,myy%,pxr%,py%
    sx%=MIN(sx%,16)                    ! largeur max. du point = 16
    xp%=MIN(639,MAX(xp%-10,0))          ! limite les...
    yp%=MIN(255,MAX(yp-10,0))   ! ...de coordonnées...
    xp2%=MIN(639,MAX(xp%+lr%*sx%+18,0)) ! ... de la surface...
    yp2%=MIN(255,MAX(yp%+hr%*sy%+18,0)) ! ... de la loupe
    GET xp%,yp%,xp2%,yp2%,lb$          ! sauvegarde l'arrière plan
    ERASE larr%()                      ! efface la matrice
    DIM bl$(2^2),larr%(lr%,hr%) ! réalise DIM
  PUT...
    IF flg%<>2                         ! ...et matrice de la loupe?
      FOR i%=0 TO 3                    ! toutes couleurs possibles
        DEFFILL i%,2,8                 ! établit la couleur de trame
        pxr%=xp%+4+MIN(lr%*sx%,18)  ! établit la hauteur et...
        pyr%=yp%+4+MIN(hr%*sy%,18)  ! ...la largeur de BOX

  BOX xp%+4,yp%+4,pxr%,pyr%            ! dessine et colore BOX
        GET xp%+5,yp%+5,xp%+5+15,yp%+5+15,bl$(i%) ! sauvegarde BOX
        MID$(bl$(i%),1,4)=MKI$(MAX(1,sx%))+MKI$(MAX(1,sy%))
        ' adapte l'en-tête de PUT à la taille désirée du grain
      NEXT i%
      DEFFILL 0,0,0                    ! DEFFILL blanc
      COLOR 1                          ! couleur pour le cadre de BOX

  BOX xp%,yp%,xp2%,yp2%                ! efface le fond de la loupe
      BOX xp%,yp%,xp2%,yp2%            ! trace le cadre de la loupe
      BOX xp%+4,yp%+4,xp2%-4,yp2%-4! trace le cadre de la loupe
    ENDIF
    MOUSE mx%,my%,mk%                  ! l'état en cours de la souris
    FOR i%=0 TO lr%-1                  ! indice X de la trame
      FOR j%=0 TO hr%-1                ! indice Y de la trame
        pnt%=
  POINT(mx%+i%,my%+j%)                 ! établit la couleur du grain
        IF pnt%                        ! le grain est-il placé?
          larr%(i%,j%)=pnt% ! note la couleur du grain dans la
          '                          matrice
          IF flg%<>2           ! la loupe doit-elle être activée
            PUT xp%+5+i%*sx%,yp%+5+j%*sy%,bl$(pnt%) ! trace
          ENDIF                                  !...le point de la
                                                 ! loupe
        ENDIF
      NEXT j%
    NEXT i%
    IF flg%<>2                     ! la loupe est-elle activée
      REPEAT                       ! boucle de la loupe
        MOUSE mxx%,myy%,mk%        ! cherche le statut de la
                                   ! souris
        REPEAT                     ! mise en attente...
        UNTIL MOUSEX<>mxx% OR MOUSEY<>myy% OR MOUSEK
```

```
                              ! ...jusqu'à action de la souris
      IF mxx%>xp%+5 AND mxx%<xp%+4+lr%*sx% ! autre X-souris?
        IF myy%>yp%+5 AND myy%<yp%+4+hr%*sy%! autre Y-souris?
          mxx%=INT(mxx%-(xp%+5))/sx% ! calcule les points...
          myy%=INT(myy%-(yp%+5))/sy% ! ...en considération..
          '                          ! ..des indices de trame
          IF mk%=1                ! pression du bouton gauche?
            IF flg%=1
              PLOT mx%+mxx%,my%+myy% ! donc placement du point
            ENDIF
            larr%(mxx%,myy%)=pt% ! actualise la matrice de
                                 ! trame
            PUT xp%+5+mxx%*sx%,yp%+5+myy%*sy%,bl$(pt%)
            '                    ! place le point de la loupe
          ELSE                   ! le bouton gauche est-il
            '                    ! relâché?
            pt%=
POINT(xp%+5+mxx%*sx%,yp%+5+myy%*sy%)-1
            '                    ! décrémente la couleur du
            '                    ! point
            IF pt%<0             ! indice de la couleur < 0?
              pt%=MAX(1,2^2-1)   ! indice sur maximum
            ENDIF
            COLOR pt%            ! place la couleur
            '                    ! originale du point
          ENDIF
        ENDIF
      ENDIF
    UNTIL mk%=2                  ! le bouton gauche est-il
                                 ! pressé?
    PAUSE 5                      ! petite pause pour le
                                 ! cliquement
  ENDIF
  ERASE bl$()               ! efface matrice
PUT-BOX
  PUT xp%,yp%,lb$           ! restaure le fond
RETURN
```

CIRCLE, PCIRCLE (CI,PC) — Trace, peint un cercle

[P]CIRCLE X_centre,Y_centre,Rayon

X_centre et Y_centre représentent les coordonnées du centre du cercle. Rayon est la moitié de son diamètre.

CIRCLE trace le cercle dont les coordonnées de son centre et le rayon sont notifiés. Le rayon ne peut revêtir de valeurs supérieures à 511, car le Blitter n'est pas encore à même de traiter des surfaces graphiques plus élevées.

ELLIPSE, PELLIPSE [ELL, PE] — Trace, peint une ellipse

[P]ELLIPSE X_centre, Y_centre, X_rayon, Y_rayon

"X_rayon" est le rayon de l'ellipse en axe X et "Y_rayon" est le rayon de l'ellipse en axe Y. X_centre et Y_centre déterminent le centre de l'ellipse.

FILL [FI] — Trame les surfaces avec des motifs

FILL X_position, Y_position [,Couleur]

X_position et Y_position déterminent le point de l'écran à partir duquel le remplissage s'opère jusqu'à la rencontre de sa limite, tracée par les points d'une autre couleur que la sienne.

Il convient de manier cette instruction avec une certaine prudence pour les deux raisons suivantes. En premier lieu, lors du tramage de l'écran, le moindre pixel, l'unité physique élémentaire de l'écran, disponible est utilisé par l'expansion de la trame. Cela amène parfois le tramage de zones d'écran adjacentes, la suppression implicite de leur contenu et le triste constat de leur perte, la restauration des contenus originaux des zones d'écran ainsi affectées, étant le plus souvent vouée à l'échec.

En deuxième lieu, bien que cette fonction de remplissage soit relativement rapide puisqu'elle bénéficie des ressources du Blitter traitant à la vitesse de l'éclair des images écran de 1024 x 1024 points en seulement une seconde, un certain ralentissement de l'affichage peut en résulter. Celui-ci est plus ou moins élevé selon le temps de calcul nécessité par la complexité du motif de la trame.

Il est possible en version 3.0 d'interrompre la procédure de remplissage à l'aide de la fonction d'interruption GFA Basic. Lors de cette opération, l'appui continu des touches d'interruption est requis, la séquence interne de remplissage en cours devant d'abord se terminer.

En version 3.0, celle disponible sur notre Amiga, il est possible d'indiquer ,dans le paramètre Couleur, une valeur de teinte. Cette couleur seule limite le remplissage. Tous les autres points sont peints avec la dite teinte. Si par exemple le point de l'écran de Xposition et de Yposition est disposé dans une ligne, le remplissage ne s'effectue que sur cette ligne.

En mode clipping (Cf. CLIP), le tramage ne s'opère que jusqu'aux frontières de la zone CLIP.

Exemple (seulement en basse résolution : lowres) :

```
OPENS 2,0,0,320,256,4,0
OPENW #2,0,0,320,256,0,15,2
DEFFILL 7,2,6
PBOX 10,10,310,190
DEFFILL 11,2,8
PBOX 90,40,180,90
FOR i%=1 TO 15
  DEFFILL i%,2,8
  PCIRCLE RANDOM(80)+85,RANDOM(50)+45,10
NEXT i%
DEFFILL 4,2,8
FILL 176,44,7
DELAY 5
CLOSEW #2
CLOSES 2
```

POLYFILL [POLYF] — Peint un polygone

POLYFILL Pt, Xp(),Yp() [OFFSET Xdiff,Ydiff]

Les mêmes règles valent pour cette instruction que celles observées avec POLYLINE, à la seule différence que la surface délimitée par le polygone tracé est peinte, tramée à l'aide du motif indiqué.

Une illustration de son application est exposée sous POLYLINE.

TEXT [T] — Edition de texte en mode graphique

TEXT Xt, Yt, "Texte"

Le texte peut y être indiqué sous sa forme usuelle, sous forme de variable de chaîne ou d'expression composée de texte: (A$+Strs$("1"). Xt et Yt représentent la position de l'écran où le texte aligné à gauche sera édité. Les sorties se limitent à l'écran.

Le point de référence du texte est toujours le premier caractère (même s'il s'agit d'un espace) positionné en bas à gauche de la boîte de texte, du moins en écriture ordinaire (horizontale et de gauche à droite). En règle usuelle, c'est ce caractère qui est positionné sur le point de coordonnées Xp et Yp.

Exemple :

```
DEFFILL ,2,1
PBOX 20,5,300,120
DEFTEXT 1,16,0,16
FOR i%=1 TO 4
  GRAPHMODE i%
  tx$="TEXTE (normal) GRAPHMODE "+STR$(i%)
  TEXT 50,15+i%*20,tx$
NEXT i%
```

9.3 Instructions graphiques traits et points

DRAW [DR] — Trace des points et les relie

DRAW TO Xposition,Yposition
DRAW X1,Y1 [TO X2,Y2[TO X3,Y3...]

La première forme syntaxique relie le point tracé de coordonnées Xposition et Yposition, avec le dernier point tracé par DRAW,

LOT ou LINE. La seconde forme dessine une ligne brisée de longueur quelconque passant par la chaîne de points indiquée. En l'omission de la chaîne optionnelle (donc avec DRAW X1,Y1), un unique point est tracé à la position indiquée comme lors de l'usage de PLOT.

Exemple :

Afin de vous éviter la frappe fastidieuse des coordonnées des points d'une chaîne, nous vous proposons le petit outil d'aide suivant. Dès qu'il est lancé, il vous est possible de tracer les points isolée à l'aide d'une simple pression du bouton gauche de la souris. Une matrice où sont affectées, suite à chaque pression du bouton de la souris, les coordonnées du point cliqué, est disposée dans le programme. Cette matrice est composée de deux index à mille éléments chacun, autrement dit il vous est possible de tracer mille points au maximum. Dès que cette somme est dépassée ou si vous pressez la touche droite de la souris, cette matrice est sauvegardées sous forme de lignes DATA sous le nom de DRAW.LSR sur le RAMDISK.

Suite à la clôture du premier programme, chargez à l'aide de Merge le fichier DATA en mémoire utilisateur et puis lancez le second programme. Vous observerez alors que votre oeuvre se dessinera d'elle-même sur l'écran, à vrai dire grâce aux lignes DATA et à l'instruction DRAW.

```
Programme 1 :
OPENW #0
DIM px(1000),py(1000)     ! établit la matrice des points
DO                        ! boucle d'insertion
  MOUSE x,y,k             ! cherche l'état en cours de la
                          ! souris
  IF k=1                  ! le bouton gauche est-il pressé?
    PAUSE 1               ! petite pause pour le cliquemement
    DRAW x,y              ! trace le point
    INC count             ! compteur de points +1
    px(count)=x           ! sauvegarde la coordonnée X
    py(count)=y           ! sauvegarde la coordonnée Y
 ENDIF
 EXIT IF k=2 OR count=1000 ! le traçage de 1000 points ou...
LOOP                      ! la pression de la touche droite?
OPEN "O",#1,"RAM:DRAW.LST" ! ouvre le fichier
PRINT #1;"D.rawkoos:";CHR$(13) ! écrit le label DATA
FOR j=1 TO 100            ! 100 lignes DATA
  PRINT #1,"D ";          ! écrit les données DATA
  FOR i=1 TO 10           ! 10 paires de coordonnées par ligne
    INC ci                ! compteur d'index +1
    PRINT #1;STR$(px(ci));",";STR$(py(ci)); ! écrit X et Y
    EXIT IF ci=>count     ! abandon si nombre < 1000
    IF i<10               ! fin de ligne non atteinte?
      PRINT #1;",";
    ENDIF                 ! alors insertion d'une virgule
  NEXT i                  ! la prochaine paire de coordonnées
  PRINT #1;CHR$(13)       ! insertion de CR (Carriage Return)
```

```
  EXIT IF ci=>count       ! abandon si nombre < 1000
NEXT j                    ! prochaine ligne DATA
PRINT #1;"D ";STR$(1111)  ! insère code de fin de DATA
PRINT #1;"D ";STR$(1111)  ! insère code de fin de DATA
CLOSE #1                  ! clôture du fichier
EDIT                      ! fin de programme

Programme 2:
DIM px(1000),py(1000)     ! établit la matrice des points
RESTORE d.rawkoos         ! positionne le pointeur de DATA
READ px(1),py(1)          ! lit la lère paire de
                          ! coordonnées...
LOT px(1),py(1)           ! ... et trace le point correspondant
FOR i=2 TO 1000           ! les coordonnées restantes
  READ px(i),py(i)        ! lit
  DRAW  TO px(i),py(i)    ! et dessine
  EXIT IF px(i)=1111      ! abandon dès que la fin est atteinte
NEXT i                    ! la prochaine paire
```

Additif

Le remplacement, dans le second programme, de la ligne :

```
DRAW TO px(i),py(i)
```

par la suivante :

```
BOX px(i),py(i),px(i)+10,py(i)+10
```

amène l'expression du dessin précédent sous une nouvelle forme.

DRAW $ [DR] — Simule une plume traçante

DRAW Def$[,Const[,'Def[,Var[,...]]]]

Simule une table traçante à l'écran. En Def$/"def" s'indiquent des commandes de traçage sous forme d'expression alphanumérique, de variable de chaîne ou de constante de texte. Les abréviations suivantes représentent les commandes de traçage :

fd n (forward) déplace la plume traçante n pixels en avant

bk n (backward) déplace la plume traçante n pixels en arrière

sx n mise en grandeur scalaire des valeurs fd et bk en direction X avec la valeur n

sy n mise en grandeur scalaire des valeurs fd et bk en direction Y avec la valeur n

sx 0 suspend la mise en grandeur scalaire de direction X

sy 0 suspend la mise en grandeur scalaire de direction Y

lt n (left turn) tourne la plume de n degrés vers la gauche

rt n (right turn) tourne la plume de n degrés vers la droite

tt n (turn to) positionne la plume traçante en direction absolue de n degrés (Cf. SETDRAW)

ma x,y (move déplace la plume (pu) sur la position absolue) absolue x/y (Cf. SETDRAW)

da x,y (draw déplace la plume (pd) sur la position absolute) absolue x/u et trace une ligne en ce faisant

mr x,y (move déplace la plume (pu) sur la position relative) x/y relative à la sienne présente.

dr x,y (draw déplace la plume (pd) sur la position relative) x/y relative à la sienne présente et trace une ligne en ce faisant

co n (color) établit la couleur des lignes (Cf. COLOR)

pu (pen up) lève la plume traçante

pd (pen down) pose la plume traçante

(n contient la valeur à indiquer, x et y les coordonnées à atteindre)
Une ligne d'instruction DRAW pourrait s'énoncer comme suit :

```
DRAW "ma",x%,y%,"tt",INT(10.52),"pd",A$,LEN(B$),"lt60 fd3.4"
```

Les indications d'éloignements, d'angles, de coordonnées, etc... peuvent être notifiées sous forme d'expressions, de constantes, de variables ou au sein de Def$/"Def". Le nombre d'instructions isolées séparées par des

virgules n'y est limité que par la longueur maximale d'une ligne d'instruction (pour mémoire: 255 caractères). L'omission de la virgule ne porte guère à conséquence.

Exemple :

```
GRAPHMODE 3                    ! mode XOR
FOR i%=0 TO 640 STE
 5       ! une fois de gauche à droite
  FOR j%=0 TO 1                ! deux fois l'un après l'autre
    SETDRAW 100+i%,170+COS(i%*
I/180)*50,i%+90 ! place la plume
    DRAW "pd rt90 fd20 rt90 fd30 lt45 fd14.1 lt45 fd40"
    DRAW "lt45 fd14.1 lt45 fd10 lt90 fd20 lt90 fd10 rt90"
    DRAW "fd20 rt90 fd30 rt90 fd60 rt90 fd40 rt45 fd28.3"
    DRAW "rt45 fd60 rt45 fd28.3 rt45 fd40"
    DRAW "pu fd30 pd fd40 rt90 fd20 rt90 fd30 lt45 fd14.1"
    DRAW "lt45 fd10 lt90 fd20 rt90 fd20 rt90 fd20 lt90 fd40"
    DRAW "rt90 fd20 rt90 fd80 rt45 fd28.3 rt45 fd40"
    DRAW "pu fd30 pd fd30 rt45 fd28.3 rt45 fd80 rt90 fd20"
    DRAW "rt90 fd40 lt90 fd30 lt90 fd40 rt90 fd20 rt90 fd80"
    DRAW "rt45 fd28.3 pu rt135 fd30 pd fd10 lt90 fd30 lt90"
    DRAW "fd10 lt45 fd14.1 lt45 fd10 lt45 fd14.1 pu"
  NEXT j%
NEXT i%
```

DRAW() — Livre les attributs de la plume traçante

Var=DRAW(Index)

Livre les attributs de la position en cours de la plume traçante.

Index :

```
0 = position X en cours
1 = position Y en cours
2 = angle de la direction en cours (en degrés)
3 = nombre scalaire de la direction X en cours
4 = nombre scalaire de la direction Y en cours
5 = pen-flag (-1 = pen down, 0 = pen up)
```

Les indices 0 à 4 livrent des valeurs en virgule flottante. La tortue (turtle) du langage LOGO, représentée d'ordinaire avec une flèche, vous est peut-être familière. A l'aide des deux procédures suivantes, il vous est possible de simuler cette tortue graphique. La procédure Showt

sauvegarde l'arrière plan sous la tortue et dessine une flèche sur sa position en cours. Celle Hidet restaure l'arrière plan original et supprime de ce fait la tortue.

```
SETDRAW 20,20,90                    ! place la tortue
PRINT "Manipulation de la tortue par la souris"
showt                               ! montre la tortue
DO
  dxmax=MAX(15,MIN(624,DRAW(0)))    ! les déplacements de la...
  dymax=MAX(15,MIN(240,DRAW(1)))    ! ... tortue y sont limités...
  SETDRAW dxmax,dymax,DRAW(2)       ! ...aux frontières de l'écran
  hidet                             ! cache la tortue
  DRAW "fd2"                        ! dessine 2 pixels
  showt                             ! montre la tortue
  m%=MOUSEK
  IF m%=1                           ! pression du bouton gauche?
    DRAW "lt2"                      ! alors 2 degrés à gauche
  ELSE IF m%=2                      ! pression du bouton droit?
    DRAW "rt2"                      ! alors 2 degrés à droite
  ENDIF
LOOP UNTIL m%=3
'
PROCEDURE showt                     ! procédure tortue
  LOCAL xtrtl,ytrtl,xtrtl2,yrtl2,wtrtl,ptrtl
  xtrtl=DRAW(0)                     ! quelle position X en
                                    ! cours?
  ytrtl=DRAW(1)                     ! quelle position Y en
                                    ! cours?
  xtrtl1=MAX(0,MIN(639,xtrtl-15))    !...délimite...
  ytrtl1=MAX(0,MIN(255,ytrtl-15))    !...domaine...
  xtrtl2=MAX(0,MIN(639,xtrtl1+30))   !...GET
  ytrtl2=MAX(0,MIN(255,ytrtl1+30))
  wtrtl=DRAW(2)                     ! l'angle en cours?
  ptrtl%=DRAW(5)                    ! statut de la plume en
                                    ! cours?
  GET xtrtl1,ytrtl1,xtrtl2,ytrtl2,trtlbckgrnd$ ! sauvegarde
  '                                            ! l'arrière
                                               ! plan
  DRAW "pd lt120 fd11 rt150 fd20 rt120 fd20 rt150 fd11"
  '                                 ! dessine la tortue
  SETDRAW xtrtl,ytrtl,wtrtl         ! place l'ancien statut de
  '                                 ! la tortue
  IF ptrtl%=0                       ! la plume était-elle levée?
    DRAW "pu"
  ENDIF
RETURN
'
PROCEDURE hidet                     ! effacement de la tortue
  PUT xtrtl1,ytrtl1,trtlbckgrnd$
RETURN
```

LINE (LI) — Trace une ligne

LINE X1,Y1,X2,Y2

Trace une ligne entre les points indiqués par leurs coordonnées selon le type de ligne en cours (défini à l'aide de DEFLINE).

Exemple :

```
FOR j%=1 TO 20
  LINE 10,j%*10,100,j%*10
  LINE j%*10,10,j%*10,100
NEXT j%
```

PLOT (PL) — Trace un point

PLOT X_position,Y_position

X_position et Y_position représentent les coordonnées du point à tracer.

POINT() — Etablit la valeur de couleur d'un point de l'écran

Var=POINT(X_position,Y_position)

Délivre la numéro du registre de la couleur dont le point aux coordonnées indiquées, est peint.

Exemple :

```
t%=TIMER
DEFFILL 2,2              ! DEFILL pointillés
PBOX 10,10,100,100       ! une petite boîte
FOR x%=10 TO 100         ! tous les points en direction X
  FOR y%=10 TO 100       ! tous les points en direction Y
    IF POINT(x%,y%)=0 ! le point n'est-il pas placé?
      PLOT 100+x%,y%  ! oui, alors place le point
    ENDIF
  NEXT y%                ! prochaine colonne
```

```
NEXT x%                 ! prochaine ligne
PRINT (TIMER-t%)/200" secondes"
```

Le résultat du programme est la mise en négatif de la boîte initialement peinte, chaque point bleu de la boîte originale étant remplacé par un point de couleur noire.

POLYLINE — Trace un polygone

POLYLINE Point, Xp(),Yp() [OFFSET Xdiff, Ydiff]

Les instructions POLY- (POLYFILL,POLYLINE) prennent en considération les attributs de remplissage, de marquage et de traçage précédemment définis avec DEFLINE, DEFILL ou DEFMARK.Dans les éléments de matrice de type entier Xp() et Yp(), il convient d'indiquer les paires de coordonnées désirées selon l'ordre du traçage effectif du polygone. La premier point indiqué du polygone est affecté à l'indice 0 (OPTION BASE 0) des éléments de matrice, ou selon à l'indice 1 (OPTION BASE 1), les contenus des deux premiers éléments de matrice étant utilisés en tant que valeurs des coordonnées du premier point.Le nombre des angles du polygone à tracer est à porter dans Point. Il est possible d'indiquer un nombre de 128 points maximum. La clôture du polygone sur lui-même est assurée par l'identité des dernier et premier points indiqués (Xp(n) = Xp(0), Yp(n) = Yp(0)).On évite ainsi la saisie fastidieuse des coordonnées d'une même figure dont le déplacement en un autre lieu de l'écran est désiré, est obtenu grâce à l'adjonction de l'instruction OFFSET. Cette dernière en complément des instructions POLY amène le déplacement de la figure dessinée, selon les directions X et Y déterminées positives ou négatives.

Exemple :

```
DIM x(10),y(10)              ! la matrice des points
GRAPHMODE 3                  ! mode XOR
DO                           ! boucle perpétuelle
 FOR i=0 TO 9                ! 10 points
x(i)=RANDOM(100)+56*i        ! coordonnée X aléatoire
y(i)=RANDOM(128)+40          ! coordonnée Y aléatoire
 NEXT i  x(i)=x(0            ! dernier X = premier X  y(i)=y(0)
! dernier Y = premier Y  bb=INT(RND*24)              ! motif de
trame aléatoire  cc=RANDOM(5)+1              ! type de ligne
aléatoire
```

```
 DEFFILL 1,2,bb              ! établit le motif
 POLYFILL 9,x(),y()          ! peint les surfaces
 FOR j%=1 TO 2               ! 2 fois
   FOR i%=3 TO 99 STEP 6     ! 16 offsets
     POLYLINE 11,x(),y() OFFSET i%,i% ! trace le polygone
NEXT i%                      ! prochain offset
 NEXT j%
 POLYFILL 9,x(),y()          ! efface les surfaces
LOOP
```

SETDRAW (SETD) — Positionne la plume traçante

SETDRAW Xpos, Ypos, Degrés

Place la plume traçante à la position absolue de coordonnées Xpos et Ypos selon la direction de traçage portée en Degrés (Cf. DRAW$).

```
          0 / 360
            !
            !
270---Xpos/Ypos---90
            !
            !
           180
```

9.4 Les opérations graphiques

CLIP (CLI) — Limite les sortie graphiques

CLIP Xpos, Ypos, Largeur, Hauteur,[OFFSET X,Y]
CLIP Xg,Yh TO Xd,Yb [OFFSET X,Y]
CLIP #Numéro_window"[OFFSET X,Y]
CLIP OFFSET X,Y
CLIP OFF

Délimite une surface rectangulaire sur l'écran où toutes les sorties graphiques (à part celles instruites à l'aide de PUT) seront dirigées par la suite. Il permet la détermination du point d'origine (de coordonnées 0 et 0), la position de référence des prochaines sorties graphiques, en

un lieu quelconque de l'écran. Toutes les dessins (tracés et peints par LINE, BOX, PBOX, DRAW, etc.) outrepassant les limites de cette surface rectangulaire, sont coupées sur ces limites.

1ère forme syntaxique :
Xpos et Ypos représentent la position du rectangle CLIP, Largeur et Hauteur ses données correspondantes.

2ème forme syntaxique :
Xg et Yh représentent la position de l'angle de gauche et du haut du rectangle CLIP; Xd et Yb l'angle du bas et de droite.

3ème forme syntaxique :
#Numéro_window contient le numéro de la fenêtre GFA Basic dont la surface et la position doivent valoir pour le rectangle CLIP.

4ème forme syntaxique :
X et Y déterminent les coordonnées d'origine (en référence au coin haut gauche de l'écran) des sorties graphiques futures (les sorties PUT mises à part). OFFSET X,Y peut aussi compléter les formes précédentes et assurer ainsi la détermination de l'origine de coordonnées 0 et 0.

5ème forme syntaxique :
CLIP OFF éteint le mode CLIPING en cours.

Exemple :

```
DEFFILL ,2,4                ! DEFFILL gris
BOX 2,2,218,118             ! peint les fonds
DEFFILL ,2,2                ! DEFFILL bleu pâle
CLIP 10,10,100,100          ! au point 10/10, une boîte clip
                            ! de 100 pixels de largeur et de
                            ! hauteur
PCIRCLE 60,60,60            ! peint des cercles
CLIP 110,10 TO 210,110 OFFSET 100,10
déclare la zone d'écran de coordonnées 110/10 de 210/110 en
tant que surface CLIP et positionne le point 0 en 100/10
CIRCLE 60,50,60             ! peint des cercles
CLIP OFF                    ! éteint le CLIPING
CLIP OFFSET 0,0             ! fixe de nouveau l'origine sur
                            ! 0/0
BOX 10,10,210,110           ! trace un rectangle autour des
                            ! cercles dessinés
```

GET — Sauvegarde une zone de l'écran

GET X_gauche, Y_haut, X_droit, Y_bas,Var$

Cette instruction (à ne pas confondre avec GET# propre aux fichiers) sauvegarde la zone de l'écran dont les coordonnées sont spécifiées, dans la matrice Var$ sous forme de motif de bits. Les coordonnées de la zone à transférer doivent se trouver à l'intérieur de l'écran sinon l'interpréteur n'exécute pas l'instruction.

Cette description abrupte ne laisse guère deviner les possibilités propres de l'instruction. Elle est l'une des plus essentielles de la programmation graphique.

Notre prochain programme propose une nouvelle démonstration, celle précisément de la création des menus pop up. Ces menus à la différences de ceux (drop down menus ou pull down menus) de l'interface Intuition, ne sont pas déroulants. Ils apparaissent à la position de l'écran où leur présence est requise, précisément sous le pointeur de la souris.

Il est possible de réserver une quelconque zone de l'écran à la représentation du menu ou de sensibiliser cette dernière à l'appel du menu. Suite au positionnement du pointeur à l'intérieur de cette zone sensibilisée, le menu se représente à l'instant même de son appel, sous le pointeur de la souris. Après la sélection d'une option du menu ou suite au déplacement du pointeur hors du menu en l'absence de la sélection d'une option, le menu disparaît. Quant aux conditions d'appel du menu, elles vous sont librement déterminables.

Lors de la création des menus, nous nous sommes efforcés d'assurer l'observance des définitions conventionnelles des menus déroulants propres au GFA Basic. Ainsi par exemple une ligne de texte de menu désactivée, introduite à l'aide d'un tiret, est représentée grisée à l'écran.

Mais une différence essentielle est qu'à partir de la liste des DATA, contenant les lignes de texte isolées, de quelconques options de menu peuvent être découpées et indiquées à l'aide d'un index d'initialisation librement choisi. En outre l'édition à l'écran d'une ligne de texte de menu est limitée en une largeur de 100 pixels en Hires/Midres ou de 50 pixels en Lowres. Le texte est formaté sur cette largeur et placé à sa position dans la menu par l'instruction TEXT.

```
start%=1                        ! représenter à partir de la 2ème
'                                 option uniquement
nombre%=20                      ! représenter en tout 6 options
DIM matrice$(start%+nombre%) ! crée la matrice pour autant
'         d'éléments d'options que requis par la démonstration
RESTORE m.datas                 ! positionne le pointeur DATA
FOR i%=1 TO start%+nombre% ! DATA texte du menu
  READ matrice$(i%)              ! lit
NEXT i%
m.datas:
DATA ...,Sauvegarde,Charge,Supprime,Copie,.....,QUITTE
DATA ...,Sauvegarde,Charge,Supprime,Copie,.....,QUITTE
DATA ...,Sauvegarde,Charge,Supprime,Copie,.....,QUITTE
BOX 50,50,300,200
DO
  REPEAT
    IF MOUSEX>50 AND MOUSEX<300 AND MOUSEY>50 AND MOUSEY<399
      @menu(start%,8,0,0,300,255,*matrice$(),*index%)
    ENDIF
    ' une autre forme possible d'appel
    ' IF MOUSEK=2
    '   @menu(start%,nombre%,0,0,639,255,*matrice$(),*index%)
    ' ENDIF
  UNTIL index%>0                ! ou MOUSEK
  IF matrice$(index%+start%)="Sauvegarde"
    matrice$(index%+start%)="-Sauvegarde"
    matrice$(index%+start%+1)="Charge"
    PRINT AT(1,1);"sélection : Sauvegarde   "
  ENDIF
  IF matrice$(index%+start%)="Charge"
    matrice$(index%+start%-1)="Sauvegarde"
    matrice$(index%+start%)="-Charge"
    PRINT AT(1,1);"sélection: Charge        "
  ENDIF
  IF matrice$(index%+start%)="Supprime"
    PRINT AT(1,1);"sélection: Supprime      "
  ENDIF
  IF matrice$(index%+start%)="Copie"
    matrice$(index%+start%)=CHR$(8)+" Copie"
    PRINT AT(1,1);"sélection: Copie         "
    GOTO label
  ENDIF
  IF matrice$(index%+start%)=CHR$(8)+" Copie"
    matrice$(index%+start%)="Copie"
    PRINT AT(1,1);"sélection: Copie     "
  ENDIF
  IF matrice$(index%+start%)="Quitte"
    PRINT AT(1,1);"sélection: Quitte       "
  ENDIF
  label:
  PRINT "Index du menu : ";index%
  EXIT IF matrice$(index%+start%)="Quitte"
```

```
  CLR index%
LOOP
'
PROCEDURE menu(pml%,mmx%,mxg%,myh%,mxd%,myb%,c.adr%,v.adr%)
  ' menu Pop-Up
  ' pml%  =  indice du texte de l'option du menu apparaissant en
  '          première position. Les textes doivent être insérées
  '          à partir de l'indice 0 dans la matrice.
  ' mmx%  =  nombre des options de menu représentées à partir de
  '         pml%. L'index du menu est toujours compris entre 1
  '         et mmx%.
  ' mxg%, myh%, mxd%, myb%
  '       =  coordonnées du cadre où le menu est représenté.
  '          Si le menu ne rentre pas dans le cadre,
  '          les coins du bas et de droite du menu et du cadre
  '          se superposent toujours.
  ' c.adr% = pointeur sur la matrice de texte
  ' v.adr% = pointeur sur une variable retournée.
  '          La variable retournée contient toujours après
  '          la sélection, l'indice de l'option de menu
  '          sélectionnée. Afin d'établir l'indice du texte
  '          correspondant dans la matrice, il faut additionner
  '          "pml%" à l'indice de l'option.
  LOCAL mmen$,msk%,m.key$,yi%,yi2%,mrs%,m.i%,lsr%
  LOCAL mxg2%,mxd2%,myh2%,myb2%
  DIM dum$(1)                  ! matrice de permutation locale
  SWAP *c.adr%,dum$()          ! permute les indices
  mxg2%=MIN(MAX(MOUSEX-68,mxg%),mxd%-136)
  mxd2%=mxg2%+136
  myh2%=MIN(MAX(MOUSEY-6,myh%),myb%-18+mmx%*18)
  myb2%=myh2%+(18+mmx%*18)
  mxg%=MIN(mxg%,mxg2%)
  myh%=MIN(myh%,myh2%)
  '
  ' Les six dernières lignes remplissent la difficile tâche de
  ' combiner ensemble la zone de représentation du menu et la
  ' position afin de représenter le menu si possible sous
  ' le pointeur de la souris mais non à l'extérieur du cadre
  ' précédemment délimité.
  '  GET MAX(0,mxg2%),MAX(0,myh2%),MIN(639,mxd2%),MIN(myb2%,231),mmen$
  ' sauvegarde l'arrière plan du menu
  DEFFILL 1,0,0                ! DEFFILL blanc
  GRAPHMODE 1                  ! mode "replace"
  PBOX mxg2%,myh2%,mxd2%,myb2%
  BOX mxg2%+1,myh2%+1,mxd2%-1,myb2%-1
  DEFFILL 1,2,4                ! DEFFILL gris
  PBOX mxg2%+6,myh2%+6,mxd2%-6,myb2%-6
  DEFFILL 1,0,0                ! DEFFILL blanc
  FOR m.i%=1 TO mmx%           ! toutes les lignes du menu
    GRAPHMODE 1                ! mode "replace"
    PBOX mxg2%+13,myh2%-6+m.i%*18,mxd2%-13,myh2%+6+m.i%*18
    GRAPHMODE 2                ! mode transparent
    IF LEFT$(dum$(m.i%+pml%))="-" ! 1er signe = - ?
```

```
    TEXT mxg2%+20,myh2%+3+m.i%*18,RIGHT$(dum$(m.i%+pml%),LEN(dum$(m.i%+pml%))-1)
    ELSE                          ! ligne active
      TEXT mxg2%+20,myh2%+3+m.i%*18-1,dum$(m.i%+pml%)
    ENDIF
  NEXT m.i%                       ! prochaine ligne
  DEFMOUSE 3                      ! DEFMOUSE doigt pointant
  DEFFILL 1,1,1                   ! DEFFILL noir
  GRAPHMODE 3                     ! mode XOR
  BOUNDARY 0                      ! version 3.0: supprime les cadres P-
  REPEAT                          ! boucle de sélection
    ON MENU                       ! attente d'événement
    MOUSE xko.x%,yko.y%,msk%      ! cherche le statut de la souris
    yi%=(INT(yko.y%-(myh2%-8))/18) ! calcule l'indice de la
    '                                 ligne
    IF xko.x%>mxg2%+12 AND yi%>0 AND xko.x%<mxd2%-13 AND yi%<=(mmx%)
      ' pointeur sur une option du Menu ?
      IF LEFT$(dum$(yi%+pml%))<>"-"  ! option sélectionnable?
        PBOX mxg2%+14,myh2%-5+yi%*18-1,mxd2%-14+1,myh2%+5+yi%*18
      ENDIF
      REPEAT                    ! attente d'une action de la souris
        yi2%=INT((MOUSEY-(myh2%-8))/18) !cherche nouvel indice
        msk%=MOUSEK             ! cherche le statut de la souris
        m.key$=INKEY$           ! acquiert le caractère entré
        ON MENU                 ! attente d'événement
      UNTIL MOUSEX<mxg2%+12 OR MOUSEX>mxd2%-13 OR yi%<>yi2% OR msk%>0 OR m.key$>""
      IF LEFT$(dum$(yi%+pml%))<>"-" ! option sélectionnable?
        PBOX mxg2%+14,myh2%-5+yi%*18-1,mxd2%-14+1,myh2%+5+yi%*18
      ELSE                       ! option inactive!
        CLR msk%                 ! effacе le statut des bouton de la souris
      ENDIF
    ELSE
      CLR yi2%
    ENDIF
  EXIT IF (MOUSEX<mxg% OR MOUSEX>mxd% OR MOUSEY<myh% OR MOUSEY>myb%) AND yi2%<=mmx%
    ' abandon de la boucle dès que le pointeur est positionné
    ' en dehors des limites du cadre de représentation
    m.key$=INKEY$             ! une frappe de caractère?
  UNTIL (msk%>0 OR m.key$>"") AND yi2%<=mmx%
  ' abandon de la boucle suite à l'utilisation du clavier ou de
  ' la pression d'un bouton de la souris
  DEFMOUSE 0                  ! DEFMOUSE flèche
  DEFFILL 1,0,0               ! DEFFILL blanc
  GRAPHMODE 1                 ! mode "replace"
  PUT MAX(0,mxg2%),MAX(0,myh2%),mmen$ ! restaure arrière plan
  BOUNDARY 1                  ! version 3.0: remet les cadres P-
  SWAP *c.adr%,dum$()         ! matrice de nouveau globale
  ERASE dum$()                ! effacе les variables locales
  *v.adr%=yi2%                ! retourne les indices choisis
  PAUSE 5                     ! petite pause pour le cliquement
RETURN
```

D'autres applications de l'instruction GET sont disséminées tout le long de l'ouvrage.

PUT (PU) — Spécifie une zone d'écran

PUT X_gauche, Y_haut, Var$ [,Mode, Masque]

PUT (à ne pas confondre avec l'instruction PUT# concernant les fichiers) transfère une image écran mémorisée à l'aide de GET, sur l'écran aux coordonnées X_gauche et Y_haut. La taille de l'image définie avec GET y reste inchangée. Les coordonnées du coin bas droit résultent de la hauteur et de la largeur de l'image mémorisée par GET. La position peut être choisie de telle façon à ce que l'image ne soit que partiellement visible ou d'ailleurs complètement invisible à l'écran.

Avec l'option Mode, il vous est possible de déterminer un mode graphique. Le mode graphique par défaut est le mode Replace. Il est déconseillé d'utiliser des valeurs négatives lors des sélections des modes car elles inhibent la fonction.

Mode :

DestInvert (&H30)
Surimposition de l'image transférée sur la zone d'écran recouverte.

SourceInvert (&H50)
Mise en inversion de l'image transférée.

ExclDestSource (&H60)
L'image transférée est combinée en mode XOR avec la zone d'écran recouverte.

OrDestSource (&H80)
La combinaison de l'image transférée et de la zone d'écran recouverte n'est effective que pour leurs bits identiques.

DestSource (&HC0)
L'image transférée recouvre la zone d'écran où elle est positionnée.

9.4.1 Organisation d'une chaîne PUT

Il serait peut être utile de consacrer quelques instants à cette question pour clarifier les présents développements. Les trois premiers mots (6 octets) d'une chaîne PUT contiennent selon l'ordre suivant :

Mot 1
La largeur de l'image moins 1 (=X_droit - X_gauche)

Mot 2
La hauteur de l'image moins 1 (=Y_bas - Y_haut)

Mot 3
Le nombre des BitPlanes

Seul un BitPlane assure la correspondance entre les bits d'une trame binaire résultant de l'exécution d'une instruction PUT et les points physiques de la zone écran mémorisée. En tous les autres cas, la combinaison des bits superposés représente le numéro du registre de couleur où ce point puise sa couleur.

Ces "header" (en têtes) introduisent les informations de bits de la zone écran mémorisée, ordonnées comme en mémoire.

Cette procédure d'une explication fort aisée, respecte le schéma suivant:

Nous présupposons que la zone d'écran à mémoriser a une largeur de 26 pixels et une hauteur de 15 pixels. La largeur de la zone d'écran est divisée par 16 (l'étendue d'un mot), et lors de l'obtention d'un reste suite à cette division, la valeur 1 est ajoutée au résultat de la division entière:

```
WORD 1 = INT(26/16)+ABS((26 MOD 16)>0)
```

La largeur de la trame de bits d' une chaîne PUT est toujours entièrement divisible par 16. Dans notre exemple les bits-pixels 17-26 de la première partie de la zone écran occupent les bits 0 à 9 du cinquième mot contenant la chaîne, les bits 10 à 15 restant étant occupés par "n'importe quoi". Ces cases inutiles contiennent donc ainsi des bits inutilisables.

```
<                  32 bits (= 2 mots)>
<largeur de la zone mémorisée = 27 pixel>          inutiles

* * * * * * * * * * * * * * * * * * * * * * * * * * 0 0 0 1 0 0   1
* * * * * * * * * * * * * * * * * * * * * * * * * * 0 1 0 0 0 0   2
```

```
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 0 1 0 0  3
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 0 0 1 0  4
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 1 0 0 0  5
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 1 1 0 0  6
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 0 0 0 0  7
* * * * * DONNEES DE L' IMAGE * * * 0 0 0 0 0 0  8
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 1 0 1 0 0  9
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 0 0 0 0 0 10
* * * * * * * * * * * * * * * * ** * * * * * * * * * 1 0 0 0 1 0 11
* * * * * * * * * * * * * * * * ** * * * * * * * * * 1 0 0 0 0 0 12
* * * * * * * * * * * * * * * * ** * * * * * * * * * 1 0 1 0 1 1 13
* * * * * * * * * * * * * * * * ** * * * * * * * * * 0 1 1 1 0 0 14
* * * * * * * * * * * * * * * * ** * * * * * * * * * 1 1 0 0 1 1 15
 <   le premier mot de la ligne    le dernier mot de la ligne
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Toutes les lignes de l'écran vont se placer selon ce modèle dans la chaîne cible. Les bits 0 - 15 du sixième mot et les bits 0 et 9 du septième mot contiennent les données de la deuxième ligne de pixels; les bits 0 - 15 du huitième mot et les bits 0- 9 du neuvième mot celles de la troisième ligne de pixels, etc.

Avec plusieurs BitPlanes, l'opération se complique fort. Il nous fallut bien plus d'une journée pour savoir comment nous y prendre pour clarifier une telle procédure. Après de nombreux essais d'exposition, nous nous sommes décidés à nous servir d'une application graphique, à titre d'illustration et de démonstration.

Le même schéma prévaut pour la couleur à la seule différence que les informations de couleur d'un point de l'écran sont signifiées à l'aide de 2 ou de plusieurs BitPlanes (4 à vrai dire) solidaires et superposés. Chaque registre de couleur d'un point quelconque résulte ainsi de la somme des bits qui lui sont affectés (un en chaque BitPlane). La couleur d'une image GET/PUT bien que les bits inutilisés du dernier mot soient à prendre en compte, est organisée de manière identique à celle de l'image de l'écran.

Dans le graphique, les bits ont été numérotés selon leur ordre. En tant qu'exemple, j'ai choisi une trame de 14*6. Imaginons la mémorisation à l'aide de GET, d'une zone d'écran de 14 pixels de largeur et d'une hauteur de 6 pixels. Les premiers 6 octets de l'en tête de la chaîne (header) ont déjà été détaillés précédemment. En notre exemple, ceux-ci sont suivis par les informations de bits propres aux lignes isolées de quatre mots consécutifs.

Lors de la superposition des quatre BitPlanes, le registre de couleur affectant un point de l'écran résulte d'une position de quatre bits superposés qui lui est correspondante. En l'illustration suivante, nous traitons du pixel 4/3 de la zone d'écran découpée, pour détailler du décompte de la couleur.

```
En BitPlane 1, ce bit est activé  : 2^0 =  1
En BitPlane 2, il est aussi activé: 2^1 =  2
En BitPlane 3, il est désactivé:      0 =  0
En BitPlane 4, il est aussi activé: 2^4 =  8

                                    ------
La somme des quatre bits donne:       11
                                    ======
```

Le pixel 4/3 de l'image tire donc sa couleur du registre de couleur 11. Afin d'approfondir la question, nous prenons les quatres bits 205, 221, 237 et 317 du côté droit.

```
En BitPlane 1, ce bit est désactivé:      0 = 0
En BitPlane 2, il est activé:           2^1 = 2
En BitPlane 3, il est aussi activé:     2^2 = 4
En BitPlane 4, il est désactivé:          0 = 0

                                        ------
Au pixel 14/4 est affecté le registre:       6
                                        ======
```

Vous observez dans le graphique, que les deux derniers bits de chaque ligne sont vides. Néanmoins chaque mot incomplet requiert un mot complet. Si en notre illustration, l'image était d'une largeur de 17 points, une occupation de mémoire du double en résulterait, bien que le second mot de chaque ligne ne contiendrait qu'un bit.

Grâce à la numérotation des mots, il vous est possible de savoir dans quel ordre les mots sont disposés en mémoire utilisateur.

Ces connaissances élémentaires vous assurent une compréhension relative des procédures graphiques interne de l'Amiga. Néanmoins les instructions sont opératoires bien même lorsqu'elles sont incomprises. Du reste les instructions complexes suppléent fort heureusement à nos incompréhensions et évitent l'acquisition d'un savoir bien plus complexe.

9.4.2 L'organisation de la mémoire graphique

SPRITE { SPR } — **Définit un lutin et le supprime**

SPRITE #Numéro, Def.var$
SPRITE #Numéro ["]
SPRITE #Numéro [,Xpos,Ypos]
SPRITE ON/OFF

A l'aide de Def.var$ est indiquée une variable de chaîne, dont le contenu en format MKI$ définit la forme du pointeur de la souris.

La forme du pointeur de la souris imagine ce que l'on appelle un lutin (en langue anglaise : sprite). Le pointeur de la souris à l'inverse des autres lutins est solidaire de la course de la petite bête sur le bureau. Avec l'instruction SPRITE, il vous est possible de définir plusieurs lutins, et de les positionner et de les bouger sur l'écran.

La création d'un tel lutin est analogue à l'affectation d'une nouvelle forme au pointeur de la souris. La forme du lutin consiste en une trame de 16*y pixels. Elle est affectée en une chaîne où les données du lutin sont converties en données BitMap.

La chaîne du lutin obéit au format suivant :
Quatre octets sont nécessités par chaque ligne graphique du lutin. Chacun de ces octets est déposé sous forme de caractère ASCII dans la chaîne ; les bits activés marquant les points et ceux désactivés les espaces libres. Puisqu'un lutin développe une largeur de 16 points, les deux premiers et derniers octets de chaque ligne sont assemblés en deux paires distinctes représentant les points disposés les uns après les autres.

L'attribution des couleurs du lutin est du ressort de ces deux paires. Ces deux largeurs de 16 bits d'une ligne de points sont superposées. Pour chaque point est alors recherché la paire de bits superposés qui lui est affectée. Les combinaisons suivantes en résultent :

Paire de bit	Couleur
00	transparent
01	premier registre de couleur
10	second registre de couleur
11	troisième registre de couleur

La combinaison des bits délivre une valeur de couleur. La première valeur de couleur, la combinaison 00, est toujours transparente, ce qui veut dire que l'arrière plan reste visible. Quant au trois autres affectations de couleurs, elle dépendent du numéro du lutin.

Le tableau suivant, vous permettra de sélectionner à l'aide de SETCOLOR les couleurs dont vous désirez peindre votre lutin.

Numéro de lutin	Registre de couleurs
0/1	17, 18, 19
2/3	21, 22, 23
4/5	25, 26, 27
6/7	29, 30, 31

La deuxième forme syntaxique permet de supprimer rapidement le dessin d'un lutin. La suppression du pointeur de la souris peut aussi être ainsi opérée à l'aide de l'indication de son numéro (0.Sprite).

La troisième forme autorise le positionnement du lutin sur l'écran. A cette fin, on indique les coordonnées X et Y en une trame de 320 x 256 points, qui vaut indépendamment de la résolution de l'écran.

La suppression ou la mise en cours d'un lutin s'ordonne grâce à la quatrième forme et l'emploi des mots clefs OFF ou ON, qui affectent aussi le pointeur de la souris.

```
PRINT                              ! Activer la fenêtre de sortie
sp1$=STRING$(64,-1)
SPRITE #1,sp1$
FOR i%=0 TO 2000 STEP 4            !
  x%=108+SIN(i%*PI/180)*(100)
  y%=108+COS(i%*PI/180)*(40)       ! représente
  VSYNC                            ! le lutin
  SPRITE #1,x%,y%                  !
NEXT i%
SPRITE OFF                         ! débranche le lutin
VSYNC { VS }                        synchronisation VBL
VSYNC
```

Attend le prochain retour de ligne. Le rayonnement électronique débute le rafraîchissement d'image dans le coin haut gauche de l'écran et puis dessine l'une après l'autre toutes les lignes de l'écran jusqu'à ce qu'il ait abouti dans le coin droit inférieur. Cette chose faite, il débute de nouveau le rafraîchissement d'image à partir du coin supérieur gauche de l'écran. Ce balayage d'écran est opéré environ 50 fois par seconde.

Lors de sorties graphiques à l'aide de PUT et de GET, il peut être fort sensé d'attendre le prochain balayage d'écran. La synchronisation verticale d'une sortie graphique avec le balayage d'écran, permet de limiter les scintillements d'interférence. Toutefois lors de sorties graphiques nécessitant plus de temps de calcul que les rafraîchissements d'écran ordinaire, ces interférences sont inévitables. Une application de cette instruction est exposée sous le description de SPRITE. L'effectivité de VSYNC est observable suite à la suppression de la ligne VSYNC dans l'illustration.

9.5 L'animation d'objets

L'Amiga est en mesure de gérer des représentations animées et mobiles autres que les lutins. Son coprocesseur le "Blitter" assure la gestion d'objets graphiques complémentaires: les BlitterObjects (Bobs). La vitesse de traitement offerte par le Blitter amène des rafraîchissements des surfaces graphiques extrêmement rapides et répond particulièrement à l'animation d'objets.

Les instructions offertes en GFA Basic respectent le standard AmigaBASIC, afin d'assurer la meilleure compatibilité possible.

OBJECT.SHAPE — Définit l'apparence de l'objet

OBJECT.SHAPE Numéro_d'objet, DefString$
OBJECT.SHAPE Numéro_d'objet, Ancien_Objet

Cette commande sert à définir l'apparence d'un objet. La première forme syntaxique définit une image à l'aide d'une chaîne de définition. Cette chaîne vous est livrée dans l'un des programmes disposés sur la

disquette GFA Basic. Il se nomme IFF_TO_BOB.GFA et permet la conversion des images IFF crées par exemple avec DeluxePaint, en Bobs.

La seconde forme syntaxique copie simplement la forme de l'ancien objet dans le nouveau. Il vous est ainsi possible de dupliquer aisément une image et de modifier par exemple les couleurs de la copie à l'aide d'OBJECT.PLANES.

OBJECT.CLOSE — Supprime un objet

OBJECT.CLOSE [Numéro_d'objet [,Numéro_d'objet [,Numéro_d'objet ...]]]

Supprime définitivement un objet édité dans la fenêtre. Cette chose faite, l'objet n'est plus appelable et sa nouvelle définition est requise.

La suppression d'images n'est pas limitée en nombre. Elle nécessite la séparation des numéros des objets à supprimer par la virgule. Si la suppression de tous les objets actifs est désirée, le simple appel de l'instruction en l'absence de tous paramètres suffit.

OBJECT.ON — Rend visible l'objet

OBJECT.ON [Numéro_d'objet [Numéro_d'objet [,Numéro_d'objet ...]]]

Rend visible un objet préalablement défini. A l'aide des autres instructions, l'animation de cet objet peut être entreprise.

Le nombre des paramètres adjoints n'est ici non plus limité. En l'absence de paramètres, tous les objets sont commutés.

OBJECT.OFF — Rend invisible l'objet

OBJECT.OFF [Numéro_d'objet [Numéro_d'objet [,Numéro_d'objet ...]]]

Supprime un objet dans la fenêtre, autrement dit sa représentation à l'écran est suspendue. L'objet en tant que tel est toujours disponible en GFA Basic, il vous est possible de l'activer à tout instant.

Les objets dont la représentation n'est pas désirée, sont à indiquer par leur numéro, séparé l'un de l'autre par une virgule. En l'absence de l'indication de paramètres, tous les objets sont inactivés.

OBJECT.CLIP — Détermine la zone d'insertion des objets

OBJECT.CLIP X_gauche, Y_haut, X_droit, Y_bas

Il n'y pas lieu de mobiliser la fenêtre entière pour l'édition des objets animés en GFA Basic. Il vous est possible de définir en cette dernière une zone rectangulaire limitant l'édition des Bobs. A cette fin, est utilisée OBJECT.CLIP. Les coordonnées indiquées réfèrent à des positions absolues à l'intérieur de la fenêtre.

OBJECT.START — Débute le déplacement d'objets

OBJECT.START [Numéro_d'objet [Numéro_d'objet [,Numéro_d'objet ...]]]

Initialise le déplacement des objets indiqués. En l'absence de l'indication de numéro d'objet, l'instruction affecte tous les objets actifs.

OBJECT.STOP — Arrête le déplacement d'objets

OBJECT.STOP [Numéro_d'objet [Numéro_d'objet [,Numéro_d'objet ...]]]

Stoppe le déplacement des objets indiqués. En l'absence de l'indication de numéros d'objets, l'instruction affecte tous les objets actifs.

OBJECT.X — Détermine la position X

OBJECT.X Numéro_d'objet, X_position

Détermine la position X de l'objet. Cette valeur réfère à le fenêtre où sera représenté l'objet.

OBJECT.Y — Détermine la position Y

OBJECT.Y Numéro_d'objet, Y_position

Détermine la position Y de l'objet. Cette valeur réfère à le fenêtre où sera représenté l'objet.

OBJECT.AX — Accélère le mouvement d'un objet

OBJECT.AX Numéro_d'objet, Pixels_par_seconde

Si vous désirez dans votre programme bouger un objet, il vous est possible non seulement de le faire selon une vitesse linéaire, mais d'imprimer une accélération à ce mouvement. La présente instruction répond précisément de cela. Derrière l'indication du numéro d'objet, s'opère celle de l'accélération, spécifiées en pixels par seconde.

OBJECT.AX ne détermine l'accélération de l'objet qu'en direction X!

OBJECT.AY — Accélère le mouvement d'un objet

OBJECT.AY Numéro_d'objet, Pixels_par_seconde

Accélère le mouvement en direction Y (Cf. OBJECT.AX).

OBJECT.VX — Détermine l'accélération d'un objet

OBJECT.VX Numéro_d'objet, Vitesse

Détermine la vitesse de l'objet en direction X. Celle-ci est indiquée en pixels par seconde.

OBJECT.VY — Détermine l'accélération d'un objet

OBJECT.VY Numéro_d'objet, Vitesse

Détermine la vitesse de l'objet en direction Y. Celle-ci est indiquée en pixels par seconde.

OBJECT.PLANES — Change les registres de couleurs d'un objet

OBJECT.PLANES Numéro_d'objet, [,Masques_des_bits [,Plans_des_valeurs]]

Cette commande fort proche de la machine, permet d'influencer la représentation des Bobs. Deux valeurs de 8 bits y sont utilisées afin de déterminer la partition de l'image en BitPlanes d'écran isolés.

Lors de l'indication des Masques_des_bits, est déterminé en quels BitPlanes, l'image du Bob est à écrire. L'activation d'un bit signifie que le masque de couleur de l'image du Bob correspondant à ce BitPlane doit être représenté. La suppression d'un bit de définition d'un BitPlane, amène la suppression du masque de couleur correspondant.

Plans_de_valeurs compense les BitPlanes précédemment inactivés. En plaçant ici les bits précédemment supprimés, vous ordonnez ainsi à l'Amiga de placer ces BitPlanes entièrement, autrement dit d'y représenter une couleur.

Il vous est ainsi possible avec la première valeur de supprimer des couleurs dans l'objet et puis avec la seconde d'y ajouter des couleurs. En outre, il vous est possible d'éditer un Bob avec une seule couleur, et de le colorier avec la commutation des BitPlanes non utilisés en Plans_de_valeurs. Grâce à cela, les Bobs se laissent facilement peindre en couleurs multiples sans que leur définition interne en soit modifiée.

Le prochaine illustration arithmétique vous facilitera la détermination des deux valeurs.

Le Bob à quatre BitPlanes dont seuls le premier et le troisième doivent être représentés :

```
BitPlane 1: 2^0 =  1
BitPlane 2: 2^1 =  2
BitPlane 3: 2^2 =  4
BitPlane 4: 2^3 =  8
```

Nous obtenons ainsi une valeur de 5 pour Masques_de_bits. En outre le quatrième BitPlane doit aussi être rempli de bits afin de représentation. Pour ce faire, la valeur 8 est portée dans Plans_de_Valeurs.

OBJECT.PRIORITY — Spécifie l'ordre des objets

OBJECT.PRIORITY Numéro_d'objet, Priorité

Lors d'insertions de Bobs multiples dans une fenêtre, le problème suivant se pose pour l'ordinateur : en quel ordre dessiner les objets graphiques puisqu'il l'ignore? Cela importe particulièrement lors de représentations perspectives aux nombreux chevauchements.

Cette instruction permet d'affecter à chaque représentation graphique d'un Bob un indice de priorité. Plus élevé est cet indice, moins grande est sa priorité de représentation, cette dernière chevauchera ainsi les autres objets déjà représentés donc affectés d'un indice de priorité moindre.

La valeur de cet indice peut être choisie entre 0 et 32367, choix qui devrait suffire à toutes vos applications.

OBJECT.HIT — Détermine la sélection d'une collision d'objets

OBJECT.HIT Numéro_d'objet [,Valeur1 [,Valeur2]]

Il est parfois nécessaire que l'enregistrement d'une collision de deux objets ne soit pas opéré. En réponse à cela, on pourrait d'une part ignorer certaines collisions à l'aide de tests conditionnnels. Mais cela ne serait guère opportun, puisque pour chaque nouvelle collision suivante, il faudrait écrire de nouveaux tests conditionnels.

L'instruction OBJECT.HIT répond précisément de cela, puisqu'elle permet de renouveler l'affectation de tous les objets. Sous affectation, nous comprenons la nécessité de l'indication ou non de la collision. Cela est réalisée grâce au recours à des masques qui indiquent quel est le numéro d'objet à signifier lors d'une collision.

GFA Basic signifie toutes les collisions. Il nous est possible d'éviter cela grâce à l'insertion des deux valeurs. La première, une valeur de 16 bits détermine avec chacun de ses bits, quels sont les objets avec lesquels elle peut rentrer en collision, cette dernière étant signifiée. Lors d'une collision, la Valeur2 de l'objet étranger est combinée avec la Valeur1 de notre objet. La collision est signifiée seulement lorsque l'une des valeurs est différente de zéro.

OBJECT.X() — Délivre la position X

X_Position=OBJECT.X(Numéro_d'objet)

Délivre la position X en cours de l'objet identifié par Numéro_d'objet.

OBJECT.Y() — Délivre la position Y

X_Position=OBJECT.Y(Numéro_d'objet)

Délivre la position Y en cours de l'objet identifié par Numéro_d'objet.

OBJECT.VX() — Délivre la vitesse en direction X

X_Vitesse=OBJECT.VX(Numéro_d'objet)

Délivre la vitesse momentanée en direction X, en pixels par seconde.

OBJECT.VY() — Délivre la vitesse en direction Y

Y_Vitesse=OBJECT.VY(Numéro_d'objet)

Délivre la vitesse momentanée en direction Y, en pixels par seconde.

OBJECT.AX() — Délivre l'accélération en direction X

X_Accélération=OBJECT.AX(Numéro_d'objet)

Délivre l'accélération actuelle en direction X, en pixels par seconde.

OBJECT.AY() — Délivre l'accélération en direction Y

Y_Accélération=OBJECT.AY(Numéro_d'objet)

Délivre l'accélération actuelle en direction Y, en pixels par seconde.

ON COLLISION GOSUB — Branchement après collision

ON COLLISION GOSUB Procédure

Branche la procédure notifiée suite à la collision de deux objets ou d'un objet avec le bord de la fenêtre.

COLLISION — Détermine les modalités la collision

Lieu=COLLISION(Numéro_d'objet)

Détermine l'objet avec lequel celui indiqué est entré en collision. Quatre cas exceptionnels signifiés avec les valeurs suivantes peuvent s'y présenter aussi :

```
-1  désignant le bord haut de la fenêtre
-2  désignant le bord gauche
-3  désignant le bord bas
-4  désignant le bord droit
```

En outre, il est possible d'indiquer la valeur 0 ou -1 en tant qu'argument. COLLISION(0) délivre le numéro de l'objet, qui a pris part en tant que dernier à la collision et COLLISION(-1) le numéro de la fenêtre où s'est opérée la collision.

Chapitre 10

Conversion des données

ASC() Caractères de texte en code ASCII

Var=ASC("Caractère")

Délivre le code ASCII du caractère: "Caractère". Avec une chaîne, seul le code du premier caractère est retourné. Si la chaîne indiquée est vide (""), la valeur zéro est livrée. ASC() représente la fonction inverse à CHR$().

Le tableau des codes ASCII est fourni en appendice au présent ouvrage. ASCII: American Standard Code for Information Interchange (en langue française: code américain standard pour l'échange d'informations).

BIN$() Numérique en binaire

Var$=BIN$(Expression)
Var$=BIN$(Expression [,Rang])

Convertit Expression en une chaîne binaire. Expression représente une quelconque constante, expression, variable ou fonction numériques.

A l'aide du paramètre optionnel Rang, il vous est possible de limiter la valeur convertie à un rang déterminé (de 1 à 32).

Exemple :

```
PRINT BIN$(1273530)
'
PRINT
FOR i%=1 TO 4                ! 4 valeurs binaires
```

```
  READ a%                       ! lit
  PRINT BIN$(a%),"="'a%         ! convertit et retourne
NEXT i%
DATA &X1001110,&X100110101,&X10001110,&X100111001
'
PRINT
FOR i%=1 TO 12                  ! 12 valeurs binaires
 READ a$                        ! lues en tant que chaînes
 b$=RIGHT$(STRING$(14,"0")+a$,14) ! formate la chaîne
                                  ! binaire
 '                                ! sur 14 signes
 PRINT b$,                      ! édite la chaîne binaire
 PRINT "="'VAL("&X"+a$)         ! édite la valeur  FOR j%=1 TO
LEN(b$)                         ! tous les signes de la chaîne
 IF VAL("&X"+b$) AND 2^j% !     le bit est-il placé?
     PLOT 200+j%,30+i%          ! alors place le point
   ENDIF
  NEXT j%
NEXT i%
DATA 00011100011100011100
DATA 00011100011100011100
DATA 00011100011100011100
DATA 11100011100011100111
DATA 11100011100011100111
DATA 11100011100011100111
DATA 00011100011100011100
DATA 00011100011100011100
DATA 00011100011100011100
DATA 11100011100011100111
DATA 11100011100011100111
DATA 11100011100011100111
```

10.1 Les systèmes numériques

Quatre systèmes numériques sont courants en informatique :

- le système décimal
- le système hexadécimal
- le système binaire
- le système octal

Le système décimal le plus usuel a pour base le nombre dix (déci: du latin decem, 10, par exemple décamètre = 10 mètres). Tous les nombres en système décimal réfèrent à cette base. Ainsi les unités résultent de

l'élévation à la puissance 0 du chiffre 10 (10^1 = 0), les décimales de l'élévation à la puissance 1 (10^1 = 10, les centièmes de l'élévation à la puissance 2 (10^2 = 100), et ainsi de suite...

Dans les autres systèmes, l'élévation à la puissance est strictement identique, à la seule différence que d'autres valeurs sont utilisées en tant que base. En système hexadécimal (Hexa + déci = Hexadéci soit 6 + 10 = 16), le nombre 16 représente la base, en système binaire (bi=2) le nombre 2 et en système octal (octo = 8) le nombre 8. Ainsi s'expliquent les moindres valeurs revêtues par les rangs des chiffres binaires et celles bien plus élevées revêtues par les rangs des chiffres hexadécimaux.

Les valeurs des rangs :

en système décimal:

< 5ème	4ème	3ème	2ème	1er	rang
10000	1000	100	10	1	valeur/format
10^4	10^3	10^2	10^1	10^0	puissance

en système hexadécimal :

< 5ème	4ème	3ème	2ème	1er	rang
65536	4096	256	16	1	valeur
$10000	$1000	$100	$10	$1	format
16^4	16^3	16^2	16^1	16^0	puissance

en système binaire :

< 5ème	4ème	3ème	2ème	1er	rang
16	8	4	2	1	valeur
%10000	%1000	%100	%10	%1	format
2^4	2^3	2^2	2^1	2^0	puissance

en système octal

< 5ème	4ème	3ème	2ème	1er	rang
4096	512	64	8	1	valeur
&10000	&1000	&100	&10	&1	format
8^4	8^3	8^2	8^1	8^0	puissance

Le système hexadécimal développe une particularité propre puisque le nombre 16 ne s'y laisse pas représenter en un chiffre arabe de rang unique. Une astuce répond de cette impossibilité. Les nombres de 0 à 9 s'y représentent comme en système décimal. Les nombres de 10 à 15 sont désignés avec des lettres (A=10, B=11, C=12, D=13, E=14, F=15).

Le chiffre décimal 1037 s'ordonne donc comme suit :

```
  (10^0) * 7 =    7
+ (10^1) * 3 =   30
+ (10^2) * 0 =    0   (le troisième rang n'est pas
                       occupé)
+ (10^3) * 1 = 1000

     somme:    1037
               ======
```

Le chiffre binaire %100011 donne d'après les valeurs de ses rangs (ordonnés de droite à gauche, comme dans le système décimal) :

```
  (2^0) * 1 =  1
+ (2^1) * 1 =  2
+ (2^2) * 0 =  0     (le troisième rang n'est pas
                      occupé)
+ (2^3) * 0 =  0     (le quatrième rang n'est pas
                      occupé)
+ (2^4) * 0 =  0     (le cinquième rang n'est pas
                      occupé)
+ (2^5) * 1 = 32

     somme:   35
              =====
```

La même chose pour le chiffre hexadécimal $F3B0 :

```
  (16^0) *  0 =     0   (le premier rang n'est pas occupé)
+ (16^1) * 11 =   176
+ (16^2) *  3 =  2816
+ (16^3) * 15 = 61440

     somme:    64432
               =======
```

Et pour le chiffre octal &10537 :

```
   (8^0) * 7 =    7
 + (8^1) * 3 =   24
 + (8^2) * 5 =  320
 + (8^3) * 0 =    0   (le quatrième rang n'est pas
                       occupé)
 + (8^4) * 1 = 4096

     somme:    4447
               ======
```

L'inversion de cette transformation s'ordonne comme suit, en notre illustration le chiffre décimal 1352 est converti en un chiffre binaire :

```
 1352 : 2 = 676  reste 0 >--------------------------
  676 : 2 = 338  reste 0 >-------------------------| |
  338 : 2 = 169  reste 0 >-----------------------| | |
  169 : 2 =  84  reste 1 >---------------------| | | |
   84 : 2 =  42  reste 0 >-------------------| | | | |
   42 : 2 =  21  reste 0 >-----------------| | | | | |
   21 : 2 =  10  reste 1 >---------------| | | | | | |
   10 : 2 =   5  reste 0 >-------------| | | | | | | |
    5 : 2 =   2  reste 1 >-----------| | | | | | | | |
    2 : 2 =   1  reste 0 >---------| | | | | | | | | |
    1 : 2 =   0  reste 1 >-------| | | | | | | | | | |
    0 : 2 =   0  reste 0 >-----| | | | | | | | | | | |
                             | | | | | | | | | | | | |
le chiffre binaire s'écrit :   0 1 0 1 0 1 0 0 1 0 0 0
```

Les nombres décimaux se laissent aussi convertir dans les deux autres systèmes de chiffres selon le même schéma. L'exemple suivant démontre la conversion du chiffre décimal 35117 en un chiffre hexadécimal :

```
35117:16 = 2194 => 2194*16 = 35104 => diff. = 13 (D)>--------
 2194:16 =  137 =>  137*16 =  2192 => diff. =  2 (2)>------| |
  137:16 =    8 =>    8*16 =   128 => diff. =  9 (9)>------ | |
    8:16 =    0 =>    0*16 =     0 => diff. =  8 (8)>--- | | |
                                                       | | | |
                          le chiffre hexadécimal s'écrit:   8 9 2 D
```

Avec les langages de haut niveau comme le GFA Basic, il n'est évidemment plus nécessaire de procéder manuellement à ces conversions. Néanmoins en certaines circonstances, il peut être avantageux de s'y connaître tant soit peu.

Le système hexadécimal a été développé pour une raison déterminée. Le contenu de quatres bits (tétrade ou nibble) est précisément représentable en un nombre hexadécimal. Un nombre binaire de quatre rangs de chiffres peut revêtir que la valeur maximale 15 (2^0+2^1+2^2+2^3). Cette valeur précise est représentable à l'aide d'un seul chiffre hexadécimal:F. Un octet (8 bits) s'y compose d'une double tétrade (2*4 = 8), un mot (16 bits) d'une quadruple tétrade (4*4 = 16 bits) et mot long (32 bits) d'un ensemble de 8 tétrades.

CFLOAT() Un nombre entier en un nombre à virgule flottante

Var=CFLOAT(Valeur)

Convertit Valeur en un nombre réel. CFLOAT représente la fonction inverse à CINT.

CHR$() Un code ASCII en caractère de texte

Var$=CHR$(Valeur)

Délivre le caractère correspondant à la Valeur spécifiée. Cette valeur étant supérieure à 255, le caractère correspondant à la valeur MOD 256 (ou Valeur AND 255) est retourné. CHR$() est la fonction inverse de ASC().

Les applications de cette fonctions sont exposées sous INKEY$(), LINE INPUT, PRINT, WRITE et ASC().

CINT() Un nombre à virgule flottante en un nombre entier

Var=CINT(Valeur)

Convertit Valeur (un nombre réel) en un nombre entier. A l'inverse d'INT, le nombre est arrondi auparavant. CINT représente la fonction inverse à CFLOAT.

CVI0, CVL0, CVS0, CVD0 Chaîne en un nombre formaté

Fonction : Résultat:
Var=CVI("2 caractères") : un nombre entier sur 16 bits
Var=CVL("4 caractères") : un nombre entier sur 32 bits
Var=CVS("4 caractères") : un nombre réel (format Amiga BASIC)
Var=CVD("8 caractères") : un nombre réel (format MBASIC ou GFA Basic)

Ces diverses fonctions assurent la conversion du nombre de caractères correspondant en un nombre au format spécifié ci-dessus. Elles sont inverses à MKI$, MKL$, MKS$, MKD$.

Leurs applications sont illustrées sous INSTR() et dans la procédure Cut sous RIGHT$().

HEX$0 Numérique en hexadécimal

Var$=HEX$(Expression)
Var$=HEX$(Expression [,Rang]

Convertit Expression en une chaîne hexadécimale. Expression représente une quelconque variable, constante, expression ou fonction numérique. Il vous est possible limiter la valeur convertie à un rang déterminé (de 1 à 8) en Rang.

En outre, le préfixe standard des nombres hexadécimaux (par exemple $FA5C16) y est reconnu et transformé automatiquement par l'interpréteur (par exemple &HFA5C16). Lors de l'emploi isolé du & (par exemple &FA5C16), l'expression est également transformée dans la valeur hexadécimale correspondante.

MKI$(), MKL$(), MKS$(), MKD$()Nombre formaté en chaîne

Fonction: Résultat :
Var$=MKI$(Valeur entière de 16 bits) chaîne de 2 caractères
Var$=MKL$(Valeur entière de 32 bits) chaîne de 4 caractères
Var$=MKS$(Valeur réelle Amiga BASIC) chaîne de 4 caractères
Var$=MKD$(Valeur réelle MBASIC/GFA Basic)chaîne de 8 caractères

La Valeur indiquée entre parenthèses selon la taille et le format specifiés est convertie en l'expression de chaîne correspondante.

La construction des variables peut différer à l'extrême d'un compilateur, d'un interpréteur ou d'un système à l'autre. Pour ne pas dilapider un temps précieux avec des opérations arithmétiques lors de la conversion des valeurs isolées d'autres langages, dans le format requis, il vous est possible de vous servir de ces fonctions pour simplifier ces échanges de valeurs.

Ces fonctions sont inverses à CVI(), CVL(), CVS(), CVD().

Leur application est illustrée entre autre dans la procédure Cut sous RIGHT$().

OCT$() Numérique en octal

Var$=OCT$(Expression)
Var$=OCT$(Expression [,Rang]

Convertit Expression en une chaîne octale. Expression représente une quelconque variable, constante, expression ou fonction numériques. Le préfixe &O (par exemple : A%=&O16501) est utilisable lors de l'indication de nombres entiers en format octal.

A l'aide du paramètre optionnel Rang, il est possible de limiter la valeur convertie à un rang déterminé (1 à 11).

De plus amples précisions sont fournies sous: Les systèmes numériques" dans le chapitre 10.1.

STR$() — Numérique en chaîne de texte

Var$=STR$(Valeur)
Var$=STR$(Valeur [,Rang [,Réel]])

Délivre une chaîne de texte de longueur correspondante au nombre de chiffres spécifié en Valeur en un format décimal. Valeur peut être indiquée en un quelconque système numérique. Les nombres hexadécimaux, binaires ou octaux spécifiés sont auparavant mis en format décimal.

La suite de chiffre ainsi délivrée n'est aucunement un nombre qui représente une valeur, mais exclusivement une chaîne contenant les chiffres isolés de la valeur en tant que caractères de texte. STR$ représente la fonction inverse à VAL.

A l'aide du paramètre optionnel Rang, il est possible de limiter la valeur convertie à un rang déterminé (avant ou après la virgule, ou le point décimal). Le paramètre optionnel : Réel spécifie à quel rang après la virgule la valeur doit être arrondie. Les rangs arrondis après la virgule sont pris en compte dans la chaîne de valeur retournée, même si Valeur ne contient aucun rang après la virgule.

Exemples :

```
PRINT STR$(572.6169,5,3)  donne   2.617
PRINT STR$(6169,9,5)      donne 169.000000
```

VAL() — Chaîne en numérique

Var=Val(Var$)

Convertit tous les signes du début d'une chaîne, qui répondent en propre de la représentation d'une valeur numérique, en un nombre réel de système décimal.

Var$ est une quelconque chaîne de texte, d'expression ou de variable dont le contenu du début est scruté afin d'y découvrir si les signes du texte contenus représentent une valeur d'un des quatre systèmes

numériques. Cette recherche est interrompue suite à l'atteinte de la fin de la chaîne ou dès que la fonction rencontre un caractère de texte non convertible.

Si le premier signe du texte est un caractère de texte non convertible ou si la chaîne est vide, une valeur nulle est retournée.

Exemples :

```
A$="> ";STR$(123456);" <"
PRINT A$
Donne:  > 123456 <
'
A$="1011010 <- binaire"
PRINT VAL("&X"+A$)          donne:          90     (=&X1011010)
'
A$="1141331 <- octal"
PRINT VAL("&O"+A$)          donne:      312025     (=&O01141331)
'
A$="AF451DE <- hexadécimal"
PRINT VAL("&H"+A$)          donne: 1837823902
'
PRINT VAL("&HEESignes")     donne:         238      (=&HEE)
'
A$="&X110123"
A%=VAL(A$)
PRINT A%                    donne:          13      (=&X1101)
'
PRINT VAL("2.37E+07")       donne:    23700000      (=2.37E+07)
```

VAL?() Retourne le nombre de signes de texte convertibles

Var=VAL?(Var$)

Communique le nombre de signes de texte convertibles en début de la chaîne Var$ indiquée (Cf. VAL()). Var$ représente une quelconque chaîne de texte ou de variable devant être scrutée afin d'y découvrir le nombre de signes convertibles. Dès que la fonction rencontre un signe inconvertible, la recherche est interrompue.

Exemples :

```
PRINT VAL?("237E7")     donne: 6 (format exponentiel)
'
A$="&X110123E"
PRINT VAL?(A$)          donne: 6 (l'identificateur inclus)
```

Chapitre 11

Manipulations de tableaux, pointeurs et mémoire

11.1 Manipulations de tableaux

ARRAYFILL [ARR] — Affecte des chiffres à un tableau

ARRAYFILL Tableau(),Valeur

Tableau désigne un quelconque matrice numérique et dimensionnée de type entière, réelle ou booléenne. Tous les éléments de ce tableau sont occupés par la valeur spécifiée en Valeur. Cette valeur doit correspondre au type de la matrice désignée (par exemple: d'entier à entier).

Exemple :

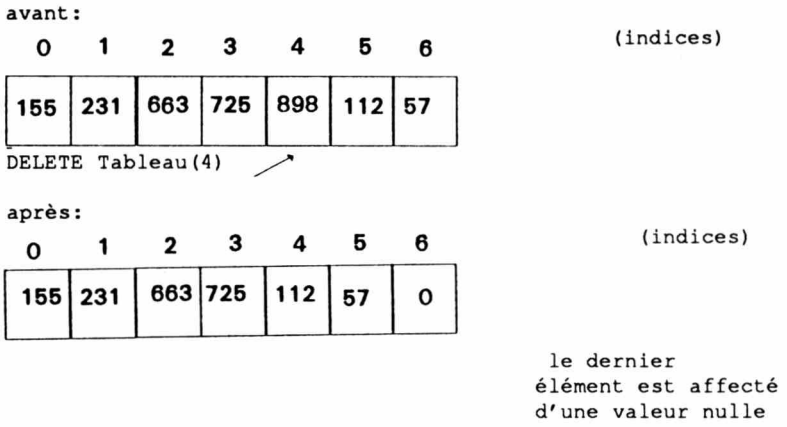
```
DIM Tableau%(20,32,16)
FOR A%=0 TO 20                !---- remplit
  FOR B%=0 TO 32                  | tous les éléments
    FOR C%=0 TO 16                | de la matrice
      Tableau%(A%,B%,C%)=237      | avec la valeur 237
    NEXT C%                       |
  NEXT B%                         |
NEXT A%                       !---
Méthode GFA:DIM Tableau%(20,32,16)
ARRAYFILL Tableau%(),237       !--- agit pareillement
```

DELETE (DEL) Supprime un élément isolé de la matrice

DELETE Tableau(Indice)
DELETE Tableau$(Indice)

Supprime l'élément isolé dont l'indice est spécifié, dans la matrice Tableau ou Tableau$. Chaque élément précédent est déplacé à l'intérieur de la matrice, d'une position vers le bas. Ensuite le dernier élément est affecté de la valeur nulle ou en des tableaux de chaînes, d'une chaîne vide. DELETE est la commande inverse à celle d'INSERT.

Exemple :



DIM Dimensionne un tableau

DIM Arr(Ind1 [,Ind2,...]) [,Arr1(Ind1 [,Ind2,...])...]

Spécifie la ou les dimensions de Arr1 (ou selon, Arr2, Arr3, etc.) et réserve en conséquence la quantité de mémoire requise. Arr représente une quelconque matrice numérique ou alphanumérique.

Ind détermine le nombre d'éléments affectés à une dimension. Avec des matrices ou tables pluridimensionnelles (par exemple DIM Tableau(5,20,7), le nombre des éléments est limité à 65535, avec des tables unidimensionnelles (par exemple DIM Tableau$(100), il n'est limité que par la taille de la mémoire utilisateur.

Attention : Avec des dimensions élevées, la position des adresses des variables restantes, particulièrement avec des variables de chaînes, peut se trouver déplacée. Si des routines machine sont disposées en ces variables de chaîne et l'une d'elle est appelée en l'absence d'une procédure VARPTR préalable, le Guru vous communiquera ses impressions. L'emploi d'INLINE (Cf.la dite instruction) pour la mise en mémoire d'une routine est de ce fait préférable et vivement conseillé.

11.1.1 Création d'une table à dimensions multiples

Représentez-vous, s'il vous plaît une armoire. Cette armoire est divisée en deux parties, et précisément en deux moitiés, celle de gauche et celle de droite. En chacune de ces moitiés, sont disposés des tiroirs. Admettons que chacune de ces moitiés dispose de deux tiroirs.

Ces tiroirs sont divisés à leur tour en compartiments isolés. Chaque tiroir contient trois compartiment. Notre armoire dispose ainsi de 12 compartiments (2*2*3).

Le dimensionnement correspondant s'énonce donc ainsi :

```
DIM Tableau (1,1,2)       =>   en OPTION BASE 0
DIM Tableau (2,2,3)       =>   en OPTION BASE 1
```

Vous vous étonnez certainement de la non affectation de (2,2,3) dans le premier énoncé. Cela répond de la raison que le décompte au sein d'une tableau débute toujours avec l'indice 0, du moins tant que l'élément 0 n'est pas éliminé avec OPTION BASE 1. Avec la première ligne est présupposée que le plus petit indice d'une dimension revêt la valeur 0.

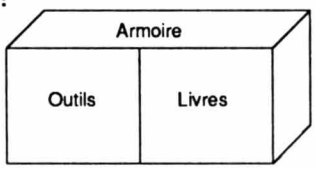
L'indice indiqué signifie donc: "dimensionne jusqu'à l'élément X", ce qui donne :

```
Elément 0 et 1 de la première dimension
Elément 0 et 1 de la deuxième dimension          DIM(1,1,2)
Elément 0, 1 et 2 de la troisième dimension
```

En ce qui suit, nous présupposons l'activation d'OPTION BASE 0. Dans cette armoire, diverses choses doivent être entreposées qui dépendent de la taille des compartiments. En notre illustration, ce sont des valeurs (valeurs de quantité pour être plus précis). Afin d'assurer une meilleure orientation, nous affectons d'abord aux deux moitiés de notre armoire des objets différenciés.

```
Sa moitié gauche est réservée aux outils     (1D-indice=0)
Sa moitié droite est réservée aux livres     (1D-indice=1)
```

Première dimension:



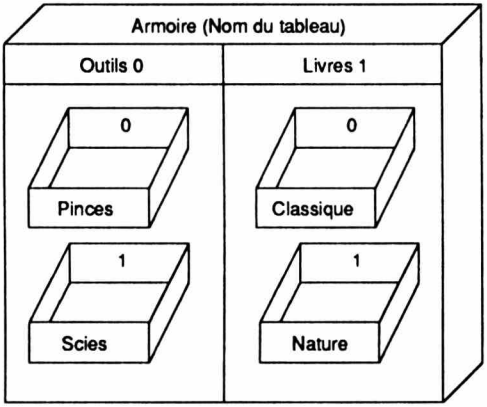
Nous réservons les tiroirs de la partie outils de notre armoire à l'entreposage des outils suivants :

```
Premier tiroir:  les pinces        (2D-indice=0)
Deuxième tiroir: les scies         (2D-indice=1)
```

Nous réservons les tiroirs de la partie livres de notre armoire aux deux catégories de littérature suivantes :

```
Premier tiroir: classique        (3D-indice=0)
Deuxième tiroir: nature           (3D-indice=1)
```

Deuxième dimension:



Maintenant nous allons nommer les éléments :

```
Pour la partie outils:              indice 0 de la 1ère dimension

   dans le tiroir: pinces          indice 0 de la 2ème dimension

        1er  compartiment: pinces plates          (3D-indice=0)
        2ème compartiment: tenailles              (3D-indice=1)
        3ème compartiment: pinces monseigneur     (3D-indice=2)

   dans le tiroir: scies            indice 1 de la 2ème dimension

       1er  compartiment: scies égoïne           (3D-indice=0)
        2ème compartiment: scies à chantourner    (3D-indice=1)
        3ème compartiment: scies à refendre       (3D-indice=2)

Pour la partie livres               indice 1 de la 1ère dimension
    dans le tiroir: classique       indice 0 de la 2ème dimension

       1er  compartiment: essais                  (3D-indice=0)
        2ème compartiment: nouvelles              (3D-indice=1)
        3ème compartiment: romans                 (3D-indice=2)

   dans le tiroir: nature          indice 1 de la 2ème dimension

        1er  compartiment: animaux                (3D-indice=0)
        2ème compartiment: végétaux               (3D-indice=1)
        3ème compartiment: minéraux               (3D-indice=2)
```

Les présents développements peuvent sembler outrés. Mais leur finalité est de montrer comment s'effectue l'accès aux dits compartiments à l'aide de leur indice.

Sous indice est compris d'ordinaire une marque distinctive ou un identificateur de taille quelconque qui représente une position déterminée en un index de table et identifiant l'élément correspondant placé en cette position, par exemple :

```
La catégorie : animaux
_____________________

    1 = domestiques
    2 = de bât
    3 = de proie
    4 = marins
    5 = à piquants
```

Au lieu d'utiliser le concept d'animaux de proie, l'on pourrait dire les animaux(3) et consulter le tableau à l'indice 3 afin d'en connaître la signification.

Transposons maintenant notre illustration en termes de place mémoire utilisateur. Ainsi grâce à l'indication des indices, il est possible à tout instant d'accéder aux informations, aux contenus ainsi indexés. Ceci est particulièrement important pour l'écriture de certaines informations à affecter à une catégorie ou à une famille déterminée.

La lecture du contenu du compartiment "végétaux" peut être opéré par son édition (PRINT Armoire(1,1,1), afin d'une quelconque affectation (A%=Armoire(1,1,1) ou d'une combinaison dans une expression (par exemple : IF 2*Armoire(1,1,1)+10). Il est aussi possible d'y écrire quoique ce soit avec par exemple: Armoire(1,1,1)=XYZ.

Pour faciliter l'identification des moitiés, tiroirs et compartiments isolés, il est possible d'affecter aux indices numériques des noms de variable.

```
outils=0
         pinces=0
                        pinces_plate=0
                        tenailles=1
                        pinces_monseigneur=2

         scies=0
                        scies_égoïne=0
                        scies_à_chantourner=1
                        scies_à_refendre=2
livres=1
         classique=0

                        essais=0
                        nouvelles=1
                        romans=2

         nature=1
                        animaux=0
                        végétaux=1
                        minéraux=2
```

Maintenant la recherche du nombre de livres disponibles sur le sujet "végétaux" apparaît bien plus simplifiée :

```
IF Armoire(livres,nature,végétaux)=0
  PRINT "Aucun livre sur les plantes n'est recensé."
ENDIF
```

ou

```
IF Armoire(outils,scies,scies_égoïnes)>10
  PRINT "Il est grand temps de se mettre à l'ouvrage!"
ENDIF
```

Nous espérons, que ce léger aperçu vous facilitera la compréhension des étroites imbrications observées dans les tables à dimensions multiples. Dans nos illustrations, nous nous sommes tenus à limiter les profondeurs des dimensions, pour que l'on puisse aisément saisir les emboîtements qui y sont disposés.

En conclusion, imaginez un terrain sur lequel seraient construits cents entrepôts. En chaque entrepôt sont disposés 60 rayonnages comprenant 50 étages. Chaque étage est divisé verticalement en 200 étagères et chacune de ces étagères en 40 casiers.

En dernier, en chaque casier sont disposés 20 boîtes divisées elles aussi en 10 compartiments.

```
DIM Entrepôts(100,60,50,200,40,20,10)
```

Le nombre d'éléments résultant de cela n'est guère traitable pour le moment sur un Amiga : 100*60*50*200*40*20*10 = 480 milliards (Bien que ...) mais cela offre un aperçu sur les possibilités presque illimitées d'insertions dans les matrices multidimensionnelles.

DIM?() — Délivre le nombre d'éléments d'une table

Var=DIM?(Tableau())

Tableau est un quelconque nom de matrice numérique ou alphanumérique. DIM? délivre le nombre des éléments contenus dans la table. Les matrices sans dimension retourne la valeur nulle (0).

Exemple :

```
DIM Tableau%(2,3,4)
IF DIM?(Tableau%())>0
  PRINT "Le tableau contient ";DIM?(Tableau());" éléments."
ELSE
  PRINT "Le tableau n'est pas dimensionné."
ENDIF
```

ERASE [ERA] — Supprime un ou des tableaux

ERASE Tableau()
ERASE Tableau1() [,Tableau2() [,...]]

Tableau représente la matrice dont la suppression est désirée. Les dimensions sont annulées et la place mémoire occupée à cet effet est libérée.

En outre il est aussi possible d'indiquer une liste de tableaux afin de les supprimer en une seule ligne de commande.

Les matrices dont on n'a plus besoin après leur utilisation, devraient être effacées immédiatement afin que la poursuite du programme puisse disposer de la mémoire utilisateur ainsi occupée.

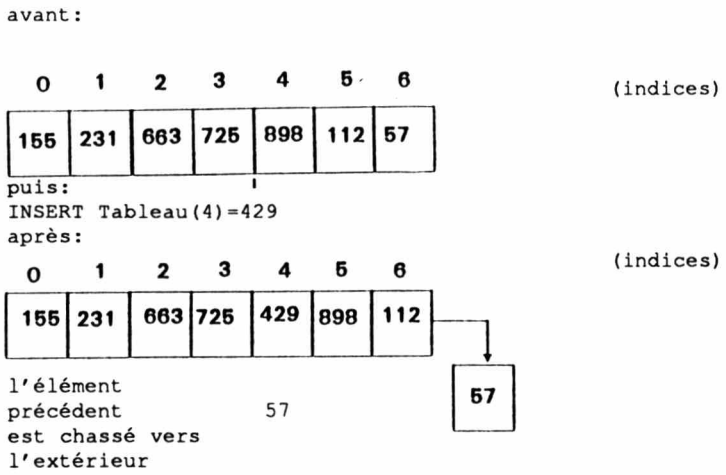
Une intéressante possibilité d'insertion consiste à insérer les tableaux en des procédures (Utilities). Comme il est entendu que le dimensionnement propre à une procédure ne peut être entrepris dans le programme principal, il convient de le réaliser au sein de l'utilitaire. A la suite de l'exécution de cette procédure, la table est effacée afin qu'au prochain appel elle puisse être nouvellement dimensionnée.

INSERT — Insère un élément isolé dans un tableau

INSERT Tableau(Indice)=Valeur
INSERT Tableau$(Indice)=Texte

Insère un élément isolé et indexé : une valeur ou un texte dans la matrice Tableau ou Tableau$. Chaque élément précédent est décalé d'une position vers le haut. Le dernier élément de la matrice est ainsi supprimé.

Exemple :



OPTION BASE [OPT BASE] Elément de base d'un tableau

OPTION BASE Base

Base détermine l'élément de base de toutes les matrice (0 ou 1). La base peut être modifiée plusieurs fois au sein d'un programme, afin que les éléments définis puissent être adaptés au nouvel index (par exemple: avec OPTION BASE 1 A$(0) devient A$(1), A$(1) devient A$(2), etc.).

Un tableau étant créé sous OPTION BASE 0 avec DIM Tableau(5), il en résulte que PRINT Tableau(6) provoque un message d'erreur et que l'élément Tableau(0) est accessible sans autre forme de procès. OPTION BASE 1 étant ensuite inséré, il en résulte que le dernier élément précédent Tableau(5) est accessible maintenant avec Tableau(6) et que l'appel du précédent élément Tableau(0) provoque l'apparition d'un message d'erreur.

La détermination de la base est effective pour toutes les tables dimensionnées en cours.

QSORT [QS] Tri rapide des éléments de tableau

QSORT Tableau([Signe]) [,Nombre[,Tableaux2%()]]
QSORT Tableau$([Signe]) [WITH Critère()] [,Nombre [,Tableaux%()]]

QSORT est l'abréviation de Quick SORT (tri rapide). Les éléments d'une matrice peuvent être triés selon leurs grandeurs numériques ou l'ordre alphabétique. Tableau() désigne une matrice numérique de quelconque type. Tableau$() est une matrice de chaînes.

Le signe moins ou celui plus peut être adjoint à la commande de tri, à l'intérieur de parenthèses (par exemple: QSORT Tableaux(+)). Il détermine le sens du tri. Si l'on désire trier les valeurs ou les chaînes en un ordre décroissant, donc des plus hautes valeurs ou hauts caractères aux moins élevés, en commençant par l'élément 0, il convient d'insérer le signe moins. Le signe plus ou l'absence de signe ordonnè un tri dans l'ordre croissant.

L'indication d'un nombre en Nombre est optionnelle. Elle détermine le nombre du dernier élément à trier (par exemple: 6 = 0 - 5 en OPTION BASE 0 ou 1 - 6 en OPTION BASE 1). Il est possible d'indiquer en option un tableau de type entier de 4 octets (Tableaux2%) dont les éléments sont triés indépendamment de leur contenu en parallèle avec la table triée en propre.

Avec des tableaux de chaîne, une matrice complémentaire quelconque (Critère() est une table de variables de type entier de 1, 2 ou 4 octet) d'au moins 256 éléments dont les contenus d'éléments présupposent l'ordre de tri, peut être spécifiée par WITH. Si les codes ASCII étaient contenus un à un dans les 256 éléments de Critère() selon leur ordre (de 0 à 255), Critère() pourrait être omis. Mais si à l'inverse, par exemple: les caractères a et A étaient échangés, A se tiendrait donc dans la liste trié sur a (en règle ordinaire, c'est l'inverse) pendant que les autres caractères seraient triés normalement. Il est ainsi possible de définir des critères de tri complètement arbitraires.

Exemple 1 :

```
DIM Tableau%(20)          ! Matrice DIM
PRINT "non trié"
FOR i%=0 TO 20            ! 20 fois
  Tableau%(i%)=RAND(100)! affecte des nombres aléatoires
  PRINT Tableau%(i%)      ! et les édite
NEXT i%                   ! prochain élément
QSORT Tableau%()          ! trie la matrice
PRINT AT(15,1);"Tri dans l'ordre croissant:"
FOR i%=0 TO 20            ! 20 éléments triés
  PRINT AT(15,i%+2);Tableau(i%) ! nouvelle édition
NEXT i%
```

Exemple 2 :

```
DIM Tableau%(20)          ! matrice DIM
PRINT "non trié"
FOR i%=0 TO 20            ! 20 fois
  Tableau%(i%)=RAND(100)! affecte des nombres aléatoires
  PRINT Tableau(i%)       ! et les édite
NEXT i%                   ! prochain élément
QSORT Tableau%(-),5       ! trie 5 éléments dans l'ordre
décroissant
PRINT AT(15,1);"Tri dans l'ordre décroissant:"
FOR i%=0 TO 20            ! 20 éléments triés
  PRINT AT(15,i%+2);Tableau(i%) ! nouvelle édition
  IF i%<5                 ! dans les limites du tri?
    PRINT AT(20,i%+2);"<-----triés"
  ELSE
```

```
    PRINT AT(4,i%+2);"----------->"
NEXT i%
```

Exemple 3 :

```
DIM Tableau%(20),Tableau2%(20)! matrices dont une parallèle DIM
PRINT "Indices:          non triés"
PRINT SPC(27);"|"
FOR i%=0 TO 20                  ! 20 fois
  Tableau%(i%)=RAND(100)        ! affecte des nombres aléatoires
  Tableau2%(i%)=i%              ! indexation de la table parallèle
  PRINT Tableau%(i%), Tableau2%(i%) ! édite
NEXT i%                         ! prochain élément
QSORT Tableau%(+),15,Tableau2%() ! 15 éléments + indices de
'                                 Tableau2 à trier
A$="|tri croissant:"            ! tri des indices de Tableau2:"
PRINT AT(28,1);A$
FOR i%=0 TO 20                  ! 20 éléments
  PRINT AT(28,i%+3);"| ";Tableau%(i%) ! édite Tableau
  PRINT AT(52)i%+3);Tableau2%(i%)     ! édite les indices
  IF i%<15                      ! dans la zone de tri?
    PRINT AT(34,i%+3);" ----triés ----"
  ELSE
    PRINT AT(34,i%+3);" - non triés - "
  ENDIF
NEXT i%
```

Exemple 4 :

L'utilisation dans les trois exemples précédents, en lieu et place des matrices de type entier de 4 octets (Tableau%), d'un quelconque autre type de matrice numérique ou alphanumérique aurait été tout autant possible. Le mode de tri et son résultat aurait été en leur principe de même nature, à la seule différence que le tri d'un tableau alphanumérique ne s'ordonne pas sur une suite numérique, mais sur une suite de caractères en considération de leur code ASCII correspondant. La prochaine illustration démontre le tri d'après un ordre de tri quelconque.

```
DIM tableau$(40),tableau2%(40),critEre|(256)
PRINT "non triés:        indices de la matrice:"
PRINT SPC(27);"|"
x$="äÄöÖüÜç"                ! les trémas et le cédille majuscule
x2$="AAOOUUC"               ! le critère de tri
FOR i%=0 TO 255             ! tous les codes ASCII
  critEre|(i%)=i%           ! Insère le critère de tri
  IF INSTR("abcdefghijklmnopqrstuvwxyz",CHR$(i%)) ! le plus
    '                       ! petit caractère?
    critEre|(i%)=ASC(CHR$(i%-32)) ! remplacer par les grands
```

```
                                    ! caractères
  ENDIF
  IF INSTR(x$,CHR$(i%))      ! trémas ou cédille majuscule?
    critEre|(i%)=ASC(MID$(x2$,INSTR(x$,CHR$(i%)),1))
    '                        ! remplacer par le critère de tri
  ENDIF
NEXT i%
FOR i%=0 TO 40               ! 40 fois
  FOR j%=0 TO 8              ! 8 caractères par chaîne
    x%=RAND(59)+65           ! codes ASCII aléatoires
    '          A-Z = 26 caractères
    '          a-z = 26 caractères
    '       äÄöÖüÜç =  7 caractères
    '                  ---------------
    '       total:  = 59 caractères
    '
    ' RAND(59) sélectionne donc une valeur comprise entre 0
    ' et 58.
    ' A cette valeur est ensuite ajoutée 65, ce qui implique
    ' que le code ASCII de "A" est ici 65.
    IF x%>115                ! X% supérieur à A-Z et a-z?
      x%=ASC(MID$(x$,x%-116,1)) ! alors tréma ou cédille
    ELSE IF x%>90            ! X% est-il compris entre a-z?
      ADD x%,6               ! X%+6 = au moins ASC ("a")
    ENDIF
    tableau$(i%)=tableau$(i%)+CHR$(x%) !enchaîne les caractères
  NEXT j%
  tableau2%(i%)=i%           ! indexation de la matrice parallèle
  PRINT tableau$(i%),tableau2%(i%) ! et édition
NEXT i%                      ! prochain élément
```

SSORT [SS] — Tri Shell des éléments d'un tableau

SSORT Tableau([Signe]) [,Nombre [,Tableau2%()]]
SSORT Tableau([Signe]) [With Critère()][,Nombre [,Tableau2%()]]

Les mêmes règles syntaxiques ont cours ici que celles observées avec QSORT. La différence essentielle entre ces deux algorithmes de tri est que le traitement opéré par QSORT est récursif et nécessite de ce fait bien plus de place de mémoire que celui opéré par SSORT. En raison de cela, SSORT se montre dans la plupart des cas bien plus lent.

La procédure ShellSort (développée par un anglais du nom de Shell) est en raison de ses spécificités plus apte à trier les tableaux déjà triés auparavant, ce qui en certaines circonstances amène lors d'un tri Qsort,

un ralentissement indéniable. En la plupart des cas, la sélection d'une commande de tri dépend dans la plupart des cas de vos goûts personnels, de la quantité de mémoire disponible libre ou des circonstances présentes.

11.2 Les opérations de mémoire

ABSOLUTE [AB] — Adressage de la variable

ABSOLUTE Var, Adresse

Cette fonction change l'adresse de la variable numérique (d'un quelconque type) en l'Adresse absolue de la mémoire utilisateur.

Adresse est strictement identique à celle retournée par VARPTR(Var) :

```
ABSOLUTE Var%,9952
PRINT 9952=V:Var%
donne: -1 (TRUE)
```

Un adressage opéré avec ABSOLUTE équivaut à celui opéré par :

```
POKE Adresse,Valeur_d'octet avec des variables d'un octet Var|
DPOKE Adresse,Valeur_du_mot avec des variables d'un mot Var&
LPOKE Adresse,Valeur_du_mot_long, avec des variables d'un mot
long Var%
```

L'interrogation d'une adresse de variable ABSOLUTE équivaut à celles entreprises avec :

```
Valeur_d'octet     =PEEK(Adresse) avec des variables d'un
                    octet Var|
Valeur_du_mot      =DPEEK(Adresse) avec des variables d'un mot
                    Var&
Valeur_du_mot_long=LPEEK(Adresse) avec des variable d'un mot
                    long Var%
```

Il convient de respecter les formats des nombres propres aux variables réelles ou booléennes. L'emploi d'ABSOLUTE avec des variables matricielles ou de chaîne est impossible. ABSOLUTE est utilisable à l'intérieur de PROCEDURE et de FONCTION avec des variables préalablement définies comme locales tout comme avec celles définies au niveau global. Néanmoins le retour dans le programme principal, amène la suspension de la présente "mise en absolu".

Il est possible d'employer un signe d'égalité à la place de la virgule entre Var et Adresse (ABSOLUTE Var%=Adresse).

BMOVE [B] — Copie de blocs de mémoire

BMOVE Source, Cible, Nombre

A partir de l'adresse Source, le nombre d'octets spécifiés sont lus et copiés dans la zone mémoire débutant à l'adresse Cible. Les zones de mémoire source et cible peuvent se recouper.

La spécification d'un nombre nul d'octets est interdite. Lorsqu'une expression est spécifiée pour le nombre d'octets à copier (par exemple: X_octets*Lignes), il peut s'ensuivre dans certaines circonstances, la transmission d'une valeur nulle qui provoque le blocage total de votre machine. Cet ennui peut être évité grâce à la fonction MAX(1,Nombre) qui assure la transmission d'un octet au moins.

BMOVE traite plus rapidement les adresses paires qu'impaires. BMOVE est une commande dont l'absence interdirait toute programmation proprement professionnelle. Les dialectes plus anciens du BASIC se différenciaient des langages plus proches de la machine, ou des langages machines en ce qu'ils n'offraient aucunement la possibilité d'un transfert rapide des blocs de mémoire. Les transferts de portions de mémoire y étaient nécessairement exécutés à l'aide de boucles FOR...NEXT et d'instructions PEEK/POKE, octet par octet. BMOVE supplée ce genre de procédure. Son domaine d'application est tellement vaste que nous préférons vous laisser décider du commencement. Sa mise en oeuvre est d'ailleurs des plus aisées.

BYTE{}, CARD{}, LONG{} Lit ou écrit un contenu de mémoire

Lecture du contenu BYTE{}=écriture d'un octet CARD{}=écriture d'un mot ou de deux octets LONG{}=écriture d'un mot long

Syntaxe (affectation en une adresse (comparable à D-L-POKE))
BYTE{Adresse}=Valeur (écrit un octet)
CARD{Adresse_paire}=Valeur (écrit deux octets ou un mot)
LONG{Adresse_paire}=Valeur (écrit quatre octets ou un mot long)
Syntaxe (lecture à partir d'une adresse (comparable à D-L-PEEK)
Var=BYTE{Adresse} (lit un octet)
Var=CARD{Adresse_paire} (lit deux octets ou un mot)
Var=LONG{Adresse_paire} (lit quatre octets ou un mot long)

A partir d'Adresse, un nombre d'octets (1, 2, 4) en considération du format correspondant sont lus ou écrits.

Lors de la lecture d'une valeur de 4 octets, la notation de LONG peut être omise.

Exemple :

```
{123456} lit 4 octets à partir de l'adresse 123456.
```

CHAR{} Lit ou écrit un texte en format C

Var$=CHAR{Adresse} (fonction de lecture d'un texte C)
CHAR{Adresse}=Expression$ (écriture d'un texte C)

Une chaîne en format C est lue ou écrite. La fonction d'écriture ponctue automatiquement la chaîne Expression$ d'un Octet nul et l'écrit à l'adresse notifiée. La fonction de lecture débute la lecture à partir d'Adresse et l'interrompt dès qu'elle rencontre la premier octet nul et retourné le texte lu.

DOUBLE{}, SINGLE{} { DO } = } { SI } = }

Lecture ou écriture en format IEEE réel double ou simple

Syntaxe (lecture à partir de l'adresse):
VarRéel=DOUBLE{Adresse_paire} (nombre réel IEEE de 8 octets)
VarRéel=SINGLE{Adresse_paire} (nombre réel IEEE de 4 octets)
Syntaxe (écriture à partir de l'adresse):
DOUBLE{Adresse_paire}=Valeur (nombre réel IEEE de 8 octets)
SINGLE{Adresse_paire}=Valeur (nombre réel IEEE de 4 octets)

Lors de l'écriture, la valeur indiquée de 8 ou 4 octets est interprétée dans le format IEEE et écrite à l'adresse indiquée. Lors de la lecture d'une valeur, le contenu des 8 ou 4 octets introduits par l'Adresse indiquée est interprété en tant que valeur en format IEEE et retourné en VarRéel.

FLOAT{} Lecture ou écriture de 8 octets en format réel

VarRéel=FLOAT{Adress_paire} (fonction de lecture)
FLOAT{Adresse_paire}=Valeur (fonction d'écriture)

Lors de l'écriture, la valeur spécifiée est interprétée en tant que valeur de 8 octets en format réel de la version 3.0 du GFA Basic et écrite sous cette forme dans les 8 octets suivant l'adresse spécifiée. Lors de la lecture, le contenu des 8 octets suivant l'adresse notifiée est interprété en tant que valeur réelle et retourné en VarRéel.

INT{}/WORD{} Ecriture/lecture d'1 nbre entier signé de 2 octets

VarEnt=INT{Adresse} (fonction de lecture)
VarEnt=WORD{Adresse} (fonction de lecture)
INT{Adresse}=Valeur (commande d'écriture)
WORD{Adresse}=Valeur (commande d'écriture)

INT{} et WORD{} sont les deux noms d'une même fonction ou d'une même commande. Lors de l'écriture, la valeur notifiée est interprété en tant que valeur signée en format entier de deux octets et écrite sous cette forme dans les 2 octets suivant l'adresse spécifiée. Lors de la lecture, le contenu des 2 octets suivant l'adresse notifiée est interprété en tant que valeur entière signée de 2 octets et retournée dans la variable réceptrice (de quelconque type).

Les valeurs signées de 2 octets sont nécessairement comprises entre -32768 (&X1000000000000000) et 32767 (&X0111111111111111). Le bit le plus élevé (bit 15) des deux octets suivant Adresse étant activé, la valeur retournée est négative. Le nombre entier positif résultant des octets inférieurs au quinzième octet (Valeur AND (2^15-1) est additionné à -32768 et délivre ainsi la valeur négative.

L'indication en VarRéel d'une variable d'un octet (par exemple : Var|) n'amène que la prise en compte de valeurs entières non signées, positives comprises entre 0 et 255.

PEEK, DPEEK, LPEEK Lecture de contenus de mémoire

Var=PEEK(Adresse) (lit un octet)
Var=DPEEK(Adresse_paire) (lit un mot - deux octets)
Var=LPEEK(Adresse_paire) (lit un mot long - quatre octets)

Un nombre d'octets (1, 2 ou 4) selon le format correspondant sont lus à partir d'Adresse. Avec DPEEK et LPEEK, seules des adresses paires y sont utilisables.

(D)(L)POKE, DPOKE, LPOKE — Modification de contenus

POKE Adresse,Octet (écriture d'un octet)
DPOKE Adresse_paire,Word (écriture d'un mot)
LPOKE Adresse_paire,Long (écriture d'un mot long)

Ecrit le valeur indiquée (Octet,Word,Long) selon le format correspondant dans les 1, 2, ou 4 octets suivants l'adresse notifiée. Avec DPOKE et LPOKE, seules des adresses paires sont utilisables.

Veillez lors de vos expérimentations des instructions POKE, à ne pas écrire malencontreusement dans des zones de mémoire essentielles, celles allouées au système et à l'interpréteur. Même s'il n'en résulte pas toujours immédiatement une interruption brutale, l'insertion de données fausses en ces zones de mémoires amènent des erreurs de traitement décelables au bout d'un certain temps seulement. Ces erreurs de traitement sont bien plus nocives en fin de compte qu'une interruption brutale, puisque les données risquent de s'en trouver endommagées du fait de leur simple chargement et de leur sauvegarde alors que l'on se croit en toute sécurité.

Nous vous conseillons d'agir avec la plus grande prudence lors des manipulations de la mémoire utilisateur. Si jamais une instruction POKE se perdait (autrement dit dès que son effectivité n'est pas manifeste), il convient de sauvegarder au plus vite toutes vos données (les programmes, etc.) sur disque ou disquette, et puis de procéder à une initialisation, à un Reset de la machine, afin de prévenir le pire.

11.3 La gestion de la mémoire utilisateur

INLINE — Allocation de mémoire interne au BASIC

INLINE Adresse%,Octets

Alloue à l'intérieur du listing du programme une zone mémoire d'une taille maximale de 32700 octets. La taille désirée est spécifiée à cette fin en Octets. Adresse% est une variable de type entier de 4 octets (non pas une variable matricielle), assurant un retour d'information. Dès que la programme rencontre une ligne d'instruction INLINE, l'adresse de

début de la zone mémoire INLINE en cours y est délivrée par le BASIC. Rien d'autre n'est entrepris. Que quelque chose soit en cette zone de mémoire ou à quelles fins cette adresse d'origine et cette zone de mémoire sont réservées, ne dépend que de ce que vous y avez entreposé lors de la création du programme.

La particularité de cette commande consiste précisément en ce que l'éditeur réagit déjà lors de l'insertion d'une ligne INLINE. Partout où est écrite une ligne INLINE et close par <Return> ou par une autre fonction de l'éditeur, est réservée une zone mémoire de la taille spécifiée. Lors de la sauvegarde du programme avec la fonction de l'éditeur Save ou des commandes SAVE ou PSAVE, les zones mémoires INLINE sont sauvegardées avec leur contenu complet en tant que parties prenantes du programme. Le prochain chargement du programme à l'aide de Load ou de LOAD, restitue les zones de mémoire INLINE.

La fonction Save,A ou la commande LIST assurent seulement la sauvegarde de la ligne d'instruction, pas celle de la zone mémoire INLINE. Lors d'un chargement d'un listing ASCII avec Merge, l'allocation de la zone mémoire a certes lieu, mais la restitution du contenu n'y est pas opérée.

L'insertion d'un commentaire ! au sein d'une ligne de programme INLINE est impossible, puisque la zone de mémoire est placée à la position du commentaire. Lors de l'allocation, la zone de mémoire est remplie d'octets nuls. La suppression d'une ligne INLINE existante amène la restitution de la zone de mémoire allouée au BASIC.

Mais à l'inverse suite à une modification ultérieure d'une ligne INLINE (la modification de la taille de la zone de mémoire allouée ou celle du nom de la variable), l'interpréteur vous interroge en premier lieu si l'ancienne zone de mémoire allouée doit être supprimée. En cas de réponse positive ("OK") à cette question, l'ancienne zone allouée est supprimée et une nouvelle zone correspondante aux nouvelles indications, est créée.

Lorsque le curseur est positionné sur une ligne INLINE déjà close donc déjà validée, on peut appuyer sur la touche <Help>. Certes cela est évidemment possible sans cela, mais cela n'a aucun intérêt. Veuillez donc appuyer sur la touche <Help>, le curseur étant positionné dans une ligne INLINE, de suite apparaît un barre de menu.

Un cliquement de souris sur Load introduit le chargement possible d'un quelconque fichier (par exemple : des codes machines, des fichiers d'image) dans la zone de mémoire allouée. Save instruit la sauvegarde de la zone de mémoire sur disquette. Dump autorise la sortie du contenu INLINE en valeurs hexadécimales de deux octets (avec des indications Offset) sur l'imprimante, et Clear supprime (en l'absence de tout message de contrôle) l'emplacement de mémoire INLINE.

Lors de la sauvegarde, l'extension .INL est adjointe au fichier INLINE, du moins en l'absence de l'indication d'une autre. Lors du chargement, l'extension .INL est présélectionnée de même. Toutefois un quelconque fichier peut être chargé. Le fichier à charger devrait être soit tout aussi grand que la zone mémoire INLINE ou supérieur à celle-ci. Les fichiers ne remplissant pas la zone de mémoire INLINE, sont refusés et provoquent l'apparition du message: "Fin de fichier atteinte". Les fichiers d'une longueur supérieure à celle de la zone INLINE sont coupés à la longueur d'INLINE.

Récapitulation :

1. Ecriture de la ligne INLINE (indications de la variable de retour et la taille). Le remplissage, la sauvegarde, l'impression et la suppression d'une zone de mémoire s'opèrent:
2. suite au positionnement du curseur dans une ligne INLINE, par la pression de la touche <Help>,
3. et ensuite par la sélection des options de menu correspondantes.

MALLOC() Allocation de zone mémoire système

Back=MALLOC(Nombre,Type)

MALLOC alloue un Nombre d'octets de la mémoire système. Cette zone de mémoire est protégée contre les accès des autres programmes (exécutés en parallèle). En Back est retournée suite à l'allocation, l'adresse de début de la zone allouée (en cas d'erreur, Back contient la valeur nulle). Cette adresse doit être impérativement notée, afin que l'emplacement réservé puisse être libéré plus tard avec MFREE.

Note: La valeur contenue dans Nombre est toujours arrondie sur le prochain multiple de 8.

La variable Type détermine le genre de mémoire à allouer.

Type=2
La zone de mémoire à réserver doit être disposée en des dites puces de mémoire. Ces puces de mémoires sont la zone de mémoire de 512 Kilo-octets le plus dasse de la mémoire. Seule cette zone est accessible par les circuits réservés à la gestion du graphisme et du son. La sauvegarde de données graphiques et sonores nécessite la réservation de ce type de mémoire.

Type=4
La zone de mémoire à réserver doit être disposée à l'extérieur des dites puces de mémoire en dite mémoire dynamique. Cela n'a de sens que si vous disposez de plus de 512 Kilo-octets de mémoire.

Type=1
La zone allouée suite à sa création ne doit plus être déplacée. La version actuelle du système d'exploitation n'autorise aucunement le déplacement ultérieur d'une zone de mémoire allouée. Mais des raisons de compatibilité conseillent néanmoins ce choix et particulièrement lors de l'utilisation d'une zone de mémoire réservée sur une très longue durée.

'Type=1'
Permet la combinaison OR avec un autre type de mémoire. 'Type= 1 OR 2' par exemple alloue des puces de mémoires non mobiles.

Type=65536
La zone de mémoire réservée est remplie 0. Ce type de mémoire est combinable à l'aide de la combinaison OR, à tout autre type.

Note: Les zones allouées de mémoire statique ne devant plus (Type=1) être déplacées bien que peu importe où elles se trouvent, il importe lors de réservation de mémoire, d'occuper d'abord la mémoire dynamique et puis ensuite si la place y est trop limitée de réserver de la mémoire statique.

Exemple :

```
Back=MALLOC(1000,4 OR 65536)
```

Alloue une zone mémoire de 1000 octets au dehors de la mémoire statique et remplit la zone réservée avec des 0.

```
Back=MALLOC(3200,2)
Réserve 32000 octets en mémoire statique
Back=MALLOC(150,2 OR 1 OR 65536)
```

Alloue une zone mémoire de 150 octets en mémoire statique et remplit la zone réservée avec des 0.

MFREE() — Libère les zones allouées MALLOC()

Back=MFREE(Adresse, Nombre)

Libère les zones de mémoires réservées à l'aide de MALLOC(). En Adresse, est spécifiée l'adresse du début de la zone de mémoire à libérer (qu'il convient de noter lors de l'appel de MALLOC !). L'indication de la taille de la zone à libérer est requise en Nombre. En Back est retournée la taille de mémoire libérée (donc Nombre).

Attention : Veillez, s'il vous plaît, à ce que les paramètres indiqués à la suite de MFREE correspondent aux indications initiales. L'insertion d'une adresse (en Adresse) ou d'une taille de zone de mémoire (en Nombre) incorrecte, amène une interruption système intempestive (Guru-Meditation).

RESERVE { RESE } — Détermine la mémoire utilisateur BASIC

RESERVE [Nombre]

Nombre indique la taille de la mémoire utilisateur BASIC (programme et variables compris) en octets. 64 Kilo-octets sont réservés par défaut. Cette valeur est aussi prise en compte lors de l'omission du paramètre suite à l'insertion de RESERVE.

Attention : RESERVE efface complètement la mémoire contenant les variables!.

11.4 Les manipulations de pointeur

* **Pointeur de variables**

Var=*Var
Var=*Chaîne()

L'emploi d'un nom de variable Var ou Chaîne de quelconque types (Cf. TYPE) selon cette procédure, amène non pas la délivrance du contenu de la variable, mais celle de son adresse. Un tableau ou une chaîne étant notifié, son descripteur correspondant (Cf. ARRPTR(X()) ou ARRPTR X$) est retourné. Une variable numérique étant spécifiée, l'adresse où son contenu est disposé est délivrée.

Exemple 1 :

```
a%=12                ! affecte 12 à A%
b%=*a%               ! sauvegarde le pointeur sur A% en B%
*b%=33               ! affecte indirectement 33 à A%
PRINT "Adresse de variable A% (par pointeur): ";*a%
PRINT "Adresse de variable A% (par VARPTR):   ";VARPTR(a%)
PRINT "Adresse de variable A% (en B%):        ";b%
PRINT "Contenu de variable A% (direct):       ";a%
PRINT "Contenu de variable A% (indirect):     ";LPEEK(b%)
```

Exemple 2 :

```
var$="GFA Basic"           ! crée une variable de chaîne
var%=5                     ! crée une variable de numérique
'                          ! de type entier de 4 octets
GOSUB proc(*var$,*var%)    ! appel-Proc de pointeurs de
                         ! variables
PRINT var$,var%          ! édite variable globales
PROCEDURE proc(para1%,para2%) ! en tête avec réception de
  '                           ! pointeurs
  LOCAL lvar$,lvar,lvar%,i%   ! prépare variables locales
  FOR i%=0 TO DPEEK(para1%+4)-1 ! longueur de chaîne
    '                           ! à partir du descripteur
    z%=PEEK(LPEEK(para1%)+i%)   ! lit les caractères (à l'aide
    '                           ! du descripteur avec PEEK)
    lvar$=lvar$+CHR$(z%)        ! établit les variables
                                ! locales
  NEXT i%                       ! prochain caractères
  IF TYPE(para2%)=0             ! Para2%=variable réelle
                                ! (Cf. TYPE())
    BMOVE para2%,VARPTR(lvar$),8 ! transmet 8 octets dans la
    '                             ! variable réelle locale
```

```
    *para1%=STRING$(3,lvar$)+" / 2^"+STR$(lvar)+" = "
    '                    ! constitue la chaîne et la retourne
    *para2%=2^lvar       ! calcule le résultat et le renvoie
  ENDIF
  IF TYPE(para2%)=2
    LPOKE VARPTR(lvar%),LPEEK(para2%) ! transmet 4 octets dans
    '                          ! dans la variable entière locale
    *para1%=STRING$(3,lvar$)+" / 2^"+STR$(lvar%)+" = "
    '                    ! constitue la chaîne et la retourne
    *para2%=2^lvar%      ! calcule le résultat et le renvoie
  ENDIF
RETURN
```

VAR se prête mieux au à la transmission de tableaux à des sous-programmes. Elle permet une transmission directe de variables matricielles. Ces dernières y sont simultanément variables d'émission et de réception. Une réception des contenus des variables comme celle opérée dans l'exemple 2 n'est pas nécessitée lors de son utilisation.

ARRPTR — Adresse du descripteur de tableau et de chaîne

Var=ARRPTR(Var$)
Var=ARRPTR(Tableau())

Retourne l'adresse de début du descripteur de chaîne ou de tableau (Cf. le chapitre : Organisation et types de variables).

VARPTR (V) — Retourne l'adresse de la variable

Var=VARPTR(Var)

Délivre l'adresse de variables numériques ou l'adresse du premier caractère des variables de chaîne. Var représente une quelconque variable (ou élément de tableau). L'abréviation V peut être utilisée en lieu et place de l'intitulé complet de la fonction, par exemple: PRINT V:A$.

Exemple :

```
A$="BASIC"          ! place une chaîne
ADR%=VARPTR(A$)     ! cherche l'adresse de la chaîne
```

```
FOR i%=0 TO 5      ! 5 caractères
  PRINT CHR$(PEEK(ADR%+i%); édite
NEXT i%
```

11.5 La bibliothèque Exec de l'Amiga

La bibliothèque Exec de l'Amiga contient d'innombrables fonctions de gestion du multitâche, des interruptions et de la mémoire.

L'utilisation efficace des fonctions de la bibliothèque Exec, requiert des connaissances sur les fonctions propres du système d'exploitation qu'est l'AmigaDOS, dont la présentation ferait exploser le cadre du présent ouvrage. Nous vous prions d'excuser la brièveté des descriptions des fonctions isolées.

Si jamais la présente exposition vous apparaissait par trop succincte, la consultation des livres suivants: "La Bible de Amiga", "La Bible Tome II" ou "Le livre de l'AmigaDOS", édités par Micro Application, vous permettra d'approfondir le sujet.

Et surtout veuillez observer que de nombreuses routines EXEC influent le traitement des informations internes à l'Amiga!

Lors de leur manipulation, les erreurs qui surgissent, n'y génèrent souvent même plus de Guru-Meditation.

Une dernière observation à tous ceux qui se hasarderont malgré ces avertissement à utiliser ces routines : l'appel des routines Exec à partir du GFA Basic est analogue à celui des fonctions propres du GFA Basic. De quelconques cérémonies d'ouverture comme celles habituelles à l'Amiga BASIC n'y sont pas de mise.

Les fonctions isolées disponible :

```
SetTaskPri     modifie la priorité d'une tâche
Ancienne_priorité = SetTaskpri(Tâche,Nouvelle_priorité)
```

Cette première est certainement la plus intéressante des fonctions disponibles. SetTaskPri modifie la priorité (-128 à +127) d'une tâche. En Ancienne_priorité est retournée la priorité antérieure.

A cette fin quelques explications :

Bien que l'Amiga donne l'impression qu'il exécute simultanément plusieurs programmes lors d'un traitement multitâche, il convient de noter qu'en réalité un seul programme est traité à un moment précis. La particularité propre d'un traitement multitâche consiste en la commutation successive des programmes isolés en des fractions de temps infinitésimales. Ces exécutions fractionnées des tâches donne l'impression à l'usager d'un traitement simultané des programmes. La commutation entre les diverses tâches est assurée par le système d'exploitation.

A chaque tâche est réservé un temps déterminé de calcul. Dans le plus simple des cas, le temps de calcul est divisé également entre les tâches isolées. Lors de l'exécution simultanée, par exemple: de trois programmes, à chacun de ces derniers est réservé un tiers du temps de calcul. Mais souvent, une telle division linéaire du temps de calcul ne se montre guère judicieuse. A titre d'illustration, nous présupposons le traitement simultané de deux programmes GFA Basic. L'un des programmes exécute des calculs mathématiques fort complexes et l'autre ne sert qu'à imprimer un graphique. En l'absence d'une imprimante laser, le second programme attend la plupart du temps que le tampon de l'imprimante soit vidé avant de pouvoir transmettre la suite des données. En un tel cas de figure, il serait fort logique de réserver la part du lion, en temps de calcul bien entendu, au programme mathématique.

La possibilité d'affecter une priorité définie à chaque tâche répond précisément de tels cas de figure. Les valeurs qu'elle peut revêtir s'échelonnent entre -127 et +128. Plus élevée est la priorité d'une tâche, plus élevé est le temps de calcul qui lui est réservé.

Que veut-ce dire concrètement ? Cela est d'une simplicité élémentaire. Restons-en à notre exemple. Pour l'instant, nous vous invitons à vous positionner dans votre programme mathématique et à y insérer en mode direct :

```
VOID SetTaskPri(FindTask(0),5)
```

Ensuite commutez le programme d'imprimante et insérez-y de même en mode direct :

```
VOID SetTaskPri(FindTask(0),-5)
```

A l'une des tâches est ainsi allouée la priorité 5, à l'autre : -5. Cette relative proximité des valeurs est amplement suffisante en ce qui est du partage des ressources afin de traitement simultané de deux tâches. Naturellement l'insertion de SetTaskPri à l'intérieur d'un programme (et non uniquement en mode direct) est chose possible. Les deux possibilités suivantes assurent le retour à la priorité standard :

- VOID SetTaskPri(FindTask(0),0),
- la sélection de l'option de menu déroulant "TaskPri 0" dans le menu de l'éditeur du GFA Basic.

Supervisor — Commute le mode Superviseur

VOID Supervisor()

Commute le micro-processeur 68000 de l'Amiga en mode Superviseur.

InitCode — Initialise les modules résidents

VOID InitCode(Valeur_de_départ,Numéro_de_version)

Initialise tous les modules résidents avec la valeur de départ et le numéro de version notifiés.

InitStruct — Initialise une zone de mémoire

VOID InitStruct(Init, Tampon, Taille)

Initialise la zone de mémoire dont la taille est spécifiée, débutant à l'adresse du tampon notifiée, avec les valeurs contenues dans la mémoire à partir d'Init.

MakeLibrary — Crée une bibliothèque de fonctions

Adresse = MakeLibrary(...)

Crée une bibliothèque de fonctions et retourne son adresse en Adresse.

FindResident — Recherche l'adresse d'une structure résidente

Adresse = FindResident(Nom)

Recherche l'adresse de la structure résidente spécifiée en Nom et la retourne suite à l'aboutissement de la recherche en Adresse.

InitResident — Initialise une structure résidante

VOID InitResident(Adresse, Liste_de_segments)

Initialise une structure résidente à l'Adresse spécifiée utilisant la liste de segments dont l'adresse est notifiée en Liste_de_segments.

Debug — Active la routine de débogage

VOID Debug()

Active la fonction ROM Wack du système d'exploitation. Suite à cela, les données à déboguer sont transmises sur la liaison série RS-232, à des fins de contrôle externe du programme par un autre calculateur.

Disable — Suspend les interruptions

VOID Disable()

Suspend diverses interruptions.

Enable — Active les interruptions suspendues

VOID Enable()

Active les interruptions suspendues à l'aide de la fonction Disable.

Forbid — Débranche la commutation des tâches

Void Forbid()

Désactive le système multitâche de l'Amiga.

Permit — Branche la commutation des tâches

VOID Permit()

La fonction inverse à Forbid. Réactive le multitâche.

Setsr — Modifie le registre d'état

Ancien = Setsr(Nouveau,Masque)

Modifie le registre d'état du micro-processeur 68000 de l'Amiga et retourne la précédente valeur du registre en Ancien.

SuperState — Commute le mode superviseur

Pile = SuperState()

Commute le micro-processeur 68000 en mode superviseur. Pile contient l'ancien pointeur de la pile. Il convient de noter et conserver cette dernière valeur car elle est requise lors de l'appel du mode utilisateur (UserState)!

UserState — Commute le mode utilisateur

VOID UserState(Pile)

Commute le micro-processeur 68000 en mode utilisateur. Pile contient la valeur délivrée par Superstate.

SetIntVector — Place le vecteur d'interruption du système

Ancienne_adresse = SetIntVector(Numéro, Adresse_nouvelle)

Place le vecteur d'interruption et retourne en Ancienne_adresse l'adresse de l'ancienne structure (l'adresse de la nouvelle structure se trouve dans Adresse_nouvelle).

AddIntServer — Insertion d'une routine d'interruption de serveur

VOID AddintServer(Numéro, Adresse)

Insère au sein des routines d'interruption de serveur existantes, une nouvelle.

Cause — Exécute une interruption logicielle

VOID Cause(Adresse)

Exécute une interruption logicielle dont l'adresse est à spécifier dans Adresse.

Allocate — Alloue une zone de mémoire

Adresse = Allocate(En_tête, Taille)

Alloue une zone mémoire à partir d'une liste propre de gestion de mémoire. En cas de réussite, Adresse contient l'adresse de début de la zone de mémoire allouée.

Deallocate — Libère une zone de mémoire

VOID Deallocate(En_tête, Adresse, Taille)

Libère une zone mémoire allouée par la fonction Allocate.

AllocMem — Réserve un emplacement de mémoire

Adresse = AllocMem(Taille, Type)

Cette fonction correspond à celle GFA Basic: MALLOC.

FreeMem — Libère un emplacement de mémoire

Taille = FreeMem(Adresse, Taille)

Cette fonction correspond à celle GFA Basic: MFREE.

AvailMem — Communique la taille de mémoire libre

Mémoire_libre = AvailMem(Type)

Retourne la taille de mémoire libre à l'instant présent. En Type est spécifié le type de mémoire (mémoire statique, mémoire dynamique, etc.).

AllocEntry — Initialise une structure MemList

Adresse = AllocEntry(Memlist)

Initialise une structure MemList et retourne en cas de succès son adresse en Adresse.

FreeEntry — Supprime des zones de mémoire

VOID FreeEntry(Memlist)

Supprime toutes les zones de mémoire, réservées à l'aide d'une fonction AllocEntry en une structure MemList.

Insert — Insère une structure nodale

VOID Insert(Liste, Node1, Node2)

Insère une structure nodale en une liste de structures nodales.

AddHead — Insère la structure nodale en tête

VOID AddHead(Liste, Node)

Insère une structure nodale en début d'une liste doublement chaînée.

AddTail — Insère la structure nodale en fin

VOID AddTail(Liste, Node)

Insère une structure nodale en fin d'une liste doublement chaînée.

Remove — Supprime une structure nodale

VOID Remove(Node)

Supprime une structure nodale dans une liste chaînée de structures nodales.

RemHead — Supprime une structure nodale en début

Adresse = RemHead(Liste)

Supprime le premier élément d'une liste chaînée de structures nodales. Liste contient l'adresse de la liste chaînée. En Adresse est retournée l'adresse de la structure nodale supprimée.

RemTail — Supprime une structure nodale en fin

Adresse = RemTail(Liste)

Supprime le dernier élément d'une liste chaînée de structures nodales. En Adresse est retournée l'adresse de la structure nodale supprimée.

Enqueue — Insère une structure nodale dans la liste système

VOID Enqueue(Liste, Node)

FindName — Recherche d'une structure nodale

Adresse = FindName(Début, Nom)

Recherche en une liste d'adresse Début, une structure nodale système de nom Nom et retourne selon le cas l'adresse de la structure nodale correspondante.

AddTask — Déclare une nouvelle tâche

VOID AddTask(TâcheCB, InitialPC, FinalPC)

Affecte une nouvelle tâche au système.

RemTask — Clôt une tâche

VOID RemTask(Tâche)

Termine la tâche indiquée et l'efface.

FindTask — Recherche une tâche

Adresse = FindTask(Nom)

Recherche dans la liste des tâches celle dont le nom est indiquée et retourne son adresse en Adresse.

SetSignal — Place des bits signal d'état

Ancien_signal = SetSignal(Nouveau_signal, Masque_signal)

Place les bits de signal de réception de tâche. Nouveau_signal contient le nouveau statut, l'ancien statut est retourné en Ancien_signal.

SetExcept — Sélectionne les bits de signal

Ancien_signal = SetExcept(Nouveau_signal, Masque)

Sélectionne l'un des 32 bits de signal d'une tâche. En Ancien_signal, le précédent bit signal est renvoyée.

Wait — Attend un signal

VOID Wait(Signal)

Suspend une tâche jusqu'à l'émission d'un signal par une autre tâche.

Signal — Emet un signal

VOID Signal(Tâche, Signal)

Envoie un signal à une autre tâche.

AllocSignal — Alloue un bit signal

Retour = AllocSignal(Numéro_de_signal)

Alloue l'un des bits de signal disponible.

FreeSignal — Libère un bit signal

VOID FreeSignal(Numéro_de_signal)

Libère un bit signal alloué à l'aide d'AllocSignal.

AllocTrap — Alloue un numéro de déroutement

Numéro_de_déroutement=
AllocTrap(Numéro_de_déroutement)

Réserve un numéro de déroutement au sein des déroutements disponibles dans la tâche exécutée.

FreeTrap — Libère un numéro de déroutement

VOID FreeTrap(Numéro_de_déroutement)

Libère un déroutement alloué à l'aide de la fonction AllocTrap.

AddPort — Insère un nouveau port de messages

VOID AddPort(MsgPort)

Insère dans la liste système des ports de messages, un nouveau port de messages et le rend effectif pour chaque tâche.

RemPort — Supprime un port de messages

VOID RemPort(MsgPort)

Supprime un port de messages dans la liste des ports de messages.

PutMsg — Adjoint une information

VOID PutMsg(MsgPort, Message)

Adjoint un message au port de messages spécifié.

GetMsg — Cherche une information

MsgPort = GetMsg(MsgPort)

Cherche la prochaine information du port de messages notifié.

ReplyMsg — Renvoie l'information réceptionnée

VOID ReplyMsg(Message)

Renvoie l'information réceptionnée au port de messages émetteur (afin de confirmation de la réception).

WaitPort — Attente d'événement

VOID WaitPort(MsgPort)

Attente d'événement dans le port de messages spécifié.

FindPort — Recherche d'un port de messages

Adresse = FindPort(Nom)

Recherche dans la liste des ports de messages système, le port dont le nom est spécifié et retourne son adresse en cas d'aboutissement.

AddLibrary — Insère une nouvelle bibliothèque

VOID AddLibrary(Adresse)

Insère une nouvelle bibliothèque dans le système et la rend effective pour toutes les tâches.

RemLibrary — Supprime une bibliothèque

Retour = RemLibrary(Adresse)

Supprime une bibliothèque dans la liste des bibliothèques du système.

CloseLibrary — Clôt une bibliothèque

VOID CloseLibrary(Adresse)

Clôt la bibliothèque dont l'adresse est portée dans Adresse.

SetFunction — Insère 1 nouvelle fonction dans 1 bibliothèque

Adresse = SetFunction(Library, Fonctoffset, Fonctadresse)

Insère en une bibliothèques de fonctions, une nouvelle fonction. En Adresse est retournée l'adresse de l'ancienne fonction qui était placée à la position indiquée dans la bibliothèque.

SumLibrary — Calcule la somme de contrôle

VOID SumLibrary(Bibliothèque)

Calcule la somme de contrôle de la bibliothèque indiquée.

AddDevice — Adjoint un périphérique

VOID AddDevice(Adresse)

Adjoint à la liste des périphériques système, un nouveau périphérique et le rend effectif pour toutes les tâches.

RemDevice — Supprime un périphérique

VOID RemDevice(Adresse)

Supprime un périphérique dans la liste des périphériques système.

OpenDevice — Ouvre un périphérique

Retour = OpenDevice(Nom, Numéro, Requête_I/O, Flags)

Ouvre le périphérique désigné par son nom et initialise la structure correspondante de requête des Entrées et Sorties.

CloseDevice — Clôt un périphérique

VOID CloseDevice(Requête_I/0)

Clôt un périphérique à partir du système.

DoIO — Exécute la commande

Retour = DoIO(Requête_I/O)

Exécute la commande spécifiée dans la structure de requête des entrées et sorties.

WaitIO — Attente de l'exécution de la commande

Retour = WaitIO(Requête_I/O)

Attend l'exécution de la commande notifiée dans la structure de requête des entrées et sorties.

AddResource — Ajoute une ressource

VOID AddResource(Adresse)

Insère une ressource dans la liste des ressources système et la rend disponibles pour toutes les tâches.

RemResource — Supprime une ressource

VOID RemResource(Adresse)

Supprime une ressource de la liste des ressources système.

OpenResource — Ouvre une ressource

Adresse = OpenResource(Nom,Version)

Ouvre une ressource déjà installée et renvoie en Adresse, l'adresse de la structure de ressource correspondante.

OpenLibrary — Ouvre une bibliothèque

Adresse = OpenLibrary(Nom,Version)

Ouvre une bibliothèque de fonctions et retourne en Adresse, l'adresse de la structure correspondante de la bibliothèque.

Chapitre 12

Instructions de contrôle de programme

12.1 Lancement et interruption de programme

CONT (CON) Continue le programme après un ordre STOP

CONT

Après l'interruption d'un programme par STOP, il est possible de poursuivre son exécution à la ligne suivant celle de la commande STOP, en insérant CONT en mode direct. Rien ne sert de poursuivre un programme, si après STOP est employé CLEAR ou si le listing du programme est ensuite modifié ou si de nouvelles variables sont à insérer.

Dans quelques cas, il peut être nécessaire d'insérer CONT en des procédures ON BREAK GOSUB ou ON ERROR GOSUB. Qu'un pareil cas se produise, dépend de certaines circonstances qui ne sont pas toujours prévisibles. Mais s'il arrivait que le programme stoppât net en une telle procédure, il pourrait se montrer utile que vous ayez déjà entendu parler de cette possiblité.

EDIT (ED) Interrompt le programme

EDIT

Son effectivité est identique à END (Cf. la description de cette dernière instruction. Toutefois EDIT assure le retour en l'absence de tout avertissement préalable dans l'éditeur (l'interpétreur) ou dans l'interface utilisateur : Workbench (avec les programmes compilés).

END — Interrompt le programme

END

Interrompt le programme en cours. Les contenus des variables et des fichiers ouverts restent tels quels dans l'interpréteur jusqu'à leur prochaine modification par le programme ou jusqu'au prochain CLEAR, et sont accessibles à partir du mode direct.

La poursuite ultérieure du programme avec CONT est impossible. L'interpréteur y déchiffrant une instruction de fin de programme, le retour dans l'éditeur est automatique. Les programmes compilés assurent le retour immédiat dans l'interface Workbench en l'absence de tout avertissement d'interruption ou sur le programme d'appel (Cf. EXEC).

QUIT { Q } — Fin de programme

QUIT
QUIT [x]

QUIT est identique à SYSTEM et agit de manière à assurer le retour en l'absence de tout message de contrôle dans l'interface Workbench ou (éventuellement avec des programmes compilés) ou sur celui où l'appel du programme a eu lieu (donc le CLI ou EXEC).

La possibilité de l'indication dans x d'une valeur de 16 bits qui est retournée dans le programme d'appel (sur D0) et y peut être évaluée (Cf. la fonction EXEC).

Conventions générales :

```
x=0   le programme a été quitté correctement - sans erreur
x=1   des erreurs BASIC sont apparus (ERR = 0 à 109)
x<0   des erreurs système (ERR = -1 à -67)
```

Suite à l'apparition d'une erreur système ou BASIC, il pourrait se montrer avantageux d'insérer QUIT ERR en une routine d'interception d'erreur (Cf. ON ERROR GOSUB) en tant que fin de programme. Ainsi le programme appelant "saurait" les raisons de l'arrêt du programme appelé.

RUN [RU] — Débute, lance un programme

RUN
RUN "Nom_de_programme"

Lance le programme en cours depuis le début. En ce faisant, les variables et l'écran sont complètement effacés. RUN peut être aussi utilisé en mode direct.

La version présente 3.0 autorise l'indication d'un nom de programme. La programme spécifié est automatiquement chargé et lancé (Cf. CHAIN à titre de comparaison).

STOP [ST] — Interrompt un programme

STOP

STOP permet l'interruption de l'exécution d'un programme en n'importe quel lieu. Une boîte d'alerte apparaît amenant le choix entre la poursuite du programme ou la commutation du mode direct.

Comme cette procédure n'amène ni l'effacement des variables, ni la clôture du programme, la continuation pas à pas du travail par l'insertion des lignes d'instructions une à une ou la poursuite du programme à l'aide de CONT est possible en mode direct. La commutation de l'éditeur est tout aussi possible. Tant qu'aucune modification du programme n'est entreprise dans l'éditeur, sa poursuite avec l'insertion de CONT en mode direct est possible.

Exemple :

```
DO                ! boucle perpétuelle
  INC Aa          ! une quelconque variable numérique
  IF Aa>100       ! variable supérieure à 100?
    CLR Aa        ! efface la variable
    STOP          ! stoppe le programme
  ENDIF
  PRINT Aa''      ! édite la valeur
LOOP
```

SYSTEM [SYS] — Fin de programme (sortie de l'interpréteur)

SYSTEM

SYSTEM [x]
Est identique à QUIT (Cf. la description correspondante).

12.2 Les fonctions d'effacement

CLEAR [CLE] — Efface matrices et variables

CLEAR

Toutes les variables numériques sont annulées. Toutes les variables de chaîne deviennent des chaînes vides. Les matrices sont effacées et leurs dimensions annulées. CLEAR ne doit pas être utisé en des procédures ou des boucles FOR...NEXT. Lors du lancement d'un programme, CLEAR est exécuté automatiquement.

CLR — Efface une variable isolée

CLR Var [,Var%, Var$,...]

Supprime les contenus des variables spécifiées. La notification de tableaux n'est pas permise.

Exemple :

```
CLR A$,B%,C,D!
```

correspond à :

```
A$=""    B%=0     C=0    D!=0
```

CLS — Efface l'écran

CLS [#Canal]

Efface la fenêtre d'édition ou celle ouverte dans l'interface: Intuition, et positionne le curseur sur Home (le coin haut gauche de l'écran). Suite à l'écriture de CLS dans un fichier par l'option #Canal, l'effacement de l'écran est opéré lors de la lecture dès que le pointeur rencontre l'ordre CLS. L'édition du CHR$(12) à l'aide de PRINT agit de manière identique et est aussi agissante lors de la lecture d'un fichier.

NEW — Efface la mémoire programme

NEW

Efface la mémoire utilisateur BASIC, le programme et ses variables compris. La mémoire est ensuite libre pour de nouvelles applications.

12.3 Opérations de datation

DATE$ — Délivre la date système

Var$=DATE$
DATE$="Chaîne_de_la_date"

DATE$ est une variable de chaîne réservée, contenant la date système en cours, sous forme de chaîne de texte au format: DD.MM.YYYY (D = Day (jour), M = Month (mois), Y = Year (année). Il est possible d'affecter une nouvelle date à DATE$ à l'aide Chaîne_de_la_date. Le format de cette chaîne est décrit sous SETTIME.

Cette chose vous est certainement connue, il vous est possible d'indiquer la date en cours dans le programme Preferences dans l'interface utilisateur : Workbench. Cette mise à jour est prise en compte par le système. En l'absence de modification de cet enregistrement (Cf. SETTIME), vous obtenez toujours la date sauvegardée sur la disquette Workbench, qui ne peut-être actualisée que dans la séquence Startup en cas de possesion d'une horloge à pile.

DELAY [DELA] — Fonction d'attente (1/1 seconde)

DELAY Secondes

Secondes détermine l'attente que doit observer le programme (Cf. PAUSE).

PAUSE [PA] — Fonction d'attente (1/50 seconde)

PAUSE Durée

Durée détermine le temps d'attente à observer par le programme en cinquantième de seconde. Seule la fonction d'interruption est exclusivement active durant ce temps.

D'autres activités (ON ERROR GOSUB, ON MENU xxxxx GOSUB, EVERY/AFTER GOSUB, etc.) sont mises en place pour la durée indiquée.

L'inconvénient d'un ordinateur 16 bits (et à plus forte raison d'un 32 bits) n'est pas d'être lent, mais d'être tellement rapide qu'il faut de temps en temps lui accorder quelques instants de repos afin de pouvoir contrôler certaines exécutions de programme. D'ailleurs cela est observable dans les illustrations qui émaillent le présent ouvrage; bien souvent s'y énonce une "petite pause pour le cliquetis".

Imaginez la conclusion d'une boucle à l'aide de l'interruption conditionnelle EXIT IF MOUSEK=1 et l'introduction d'un des blocs d'instructions suivants par une condition IF MOUSEK=1. Même si entre ces dernières, bien d'autres lignes sont disposées, l'ordinateur est tellement rapide que le cliquement de la souris pour la sortie de la boucle est évalué simultanément comme la satisfaction de la condition du branchement du bloc IF MOUSEK=1. Pour que l'usager en de pareils cas, ait le temps de relâcher le bouton de la souris, il convient d'insérer entre les deux conditions une petite pause (par exemple PAUSE 5), imperceptible lors du déroulement du programme. Et surtout rappelez vous que les pauses du style FOR...NEXT habituelles en BASIC sont interdites surl'Amiga. Mettez un 68030 à 50Mhz et vous comprendrez pourquoi ...

SETTIME (SETT) — Mise à jour de la date et de l'heure

SETTIME Heure$, Date$

En Heure$ et Date$ sont spécifiées les nouvelles heure et date système. La transmission des deux chaînes est requise.

Format européen :

```
Heure$    =    "hh:mm:ss" ou "hhmmss"
Date$     =    "jj.mm.aaaa" ou "jj.mm.aa"
```

Format U.S. (seulement en version 3.0, Cf. Mode):

```
Heure$    =    "hh:mm:ss" ou "hhmmss"      (comme ci-dessus)
Date$     =    "mm/dd/yyyy" ou "mm/dd/yy"
```

L'omission du siècle lors de l'indication de l'année (par exemple : 86 au lieu de 1986), du moins s'il s'agit d'une donnée comprise entre 1980 et 2079, ne porte pas à conséquence. Les secondes (le double point inclus) peuvent être omises lors de l'indication de l'heure. Les secondes sont mises à 0, par exemple :

```
SETTIME "15:37"                 (ne modifie que l'heure)
SETTIME "15:37:22","15.07.88"   (modifie l'heure et la date)
SETTIME "","15.07.1988"         (ne modifie que la date)
```

Le non respect du format d'insertion, entraîne l'ignorance des données entrées par l'ordinateur et les contenus précédents sont conservés tels quels. Les secondes des données horaires ne sont du reste prise en compte que sous forme paire (0, 2, 4, etc...), les données impaires sont arrondies au prochain chiffre paire suivant.

TIME$ — Délivre l'heure système

Var$=TIME$
TIME$="Chaîne_de_l'heure"

TIME$ est une variable de chaîne réservée, contenant l'heure système en cours sous forme de chaîne de texte en format hh:mm:ss. Il est possible d'affecter en TIME$ une nouvelle heure en Chaîne_de_l'heure. Le format de cette chaîne est décrit sous SETTIME. Les secondes de la donnée horaire sont arrondies à l'unité paire supérieure.

Dans l'éditeur GFA, la donnée horaire est affichée à droite en haut de l'écran. Il aurait été judicieux d'y adjoindre une indication d'heure de réveil, en particulier pour le progammeur assidu (que je suis) afin de ne pas oublier l'heure du coucher. Toutefois il faudrait suite à l'atteinte de l'heure de réveil, qu'une fonction d'interruption soit activée puisque jusqu'à présent j'ai superbement et de loin ignoré toutes les limites horaires.

TIMER — Délivre le temps écoulé

Var=TIMER

Variable réservée, contenant le temps écoulé depuis l'initialisation du système en deux centièmes de secondes.

L'Amiga dispose d'une horloge interne, incrémentée d'une unité toutes les deux centièmes de seconde. Le décompte du temps débute à l'instant même du lancement du système et est augmenté chaque seconde d'une valeur 200. Peu importe qu'une quelconque application ait été exécutée entre temps. Le décompte est réglé sur le cycle d'horloge propre au microprocesseur. Autrement dit il vous est possible de déterminer précisément à l'aide de ce compteur, le temps écoulé depuis l'initialisation du système.

En outre l'utilisation du TIMER répond particulièrement à l'insertion de conditions d'exécution temporelles dans les programmes et au chronométrage des temps des traitements (les tests benchmark).

Exemple 1 (édition du temps écoulé) :

```
Time=TIMER                ! délivrance de l'heure en cours
DO                        ! boucle perpétuelle
  X%=(TIMER-Time)/200     ! différence en secondes
  PRINT AT(10,10);RIGHT$(STRING$(3,"0")+Str$(X%),3);" secondes"
LOOP
```

Exemple 2 (mesure de temps ou De la mauvaise renommée de l'instruction GOTO ...) :

```
OPENW #0
time=TIMER
PRINT "20000 - boucle entière (WORD) à vide FOR...NEXT : ";
FOR i&=0 TO 20000
NEXT i&
PRINT (TIMER-time)/200;" secondes"
time=TIMER
PRINT "20000 - boucle entière (LONG) à vide FOR...NEXT : ";
FOR i%=0 TO 20000
NEXT i%
PRINT (TIMER-time)/200;" secondes"
time=TIMER
PRINT "20000 - boucle réelle à vide FOR...NEXT : ";
FOR i=0 TO 20000
NEXT i
```

```
PRINT (TIMER-time)/200;" secondes"
time=TIMER
PRINT "20000 - boucle entière à vide REPEAT...UNTIL : ";
CLR i&
REPEAT
  INC i&
UNTIL i&=20000
PRINT (TIMER-time)/200;" secondes"
time=TIMER
PRINT "20000 - boucle entière à vide DO...LOOP : ";
CLR i&
DO
  INC i&
  EXIT IF i&=20000
LOOP
PRINT (TIMER-time)/200;" secondes"
time=TIMER
PRINT "20000 - boucle entière à vide IF...GOTO : ";
CLR i&
boucle:
INC i&
IF i&<20000
  GOTO boucle
ENDIF
PRINT (TIMER-time)/200;" secondes"
PRINT "20000 - boucle entière à vide IF...GOTO (Type 2) : ";
CLR i&
boucle2:
IF i&=20000
  GOTO fin
ENDIF
INC i&
GOTO boucle2
fin:
PRINT (TIMER-time)/200;" secondes"
Exemple 3 (quasi multitâche)
PRINT "Pressez, s'il vous plaît, une quelconque touche"
GRAPHMODE 3              ! mode XOR pour Box
DO
  key$=INKEY$            ! interroge le clavier
  IF key$>""             ! si une touche est pressée
    PRINT key$;          ! alors édite le caractère frappé
  ENDIF
  IF TIMER MOD 100=0     ! toutes les demi-secondes
  FOR i%=10 TO 100 STEP 6  !----------
      BOX 110-i%,110-i%,120+i%,120+i%  | processus s'exécutant
    NEXT i%                            | en parallèle
    FOR i%=100 TO 10 STEP -6           |
      BOX 110-i%,110-i%,120+i%,120+i%  |
    NEXT i%                  !----------
  ENDIF
LOOP
```

12.4 La gestion des erreurs

ERR — Retourne le code de l'erreur

Var=ERR

ERR est une variable réservée, qui suite à l'apparition d'une erreur, contient son numéro d'identification (Veuillez consulter la liste des erreurs livrée en appendice).

ERR$ — Retourne le message d'erreur

Var$=ERR$(Code_erreur)

La fonction ERR$ délivre le texte du message d'erreur correpondant au code d'erreur spécifié.

ERROR [ER] — Simule une erreur

ERROR Numéro_d'erreur

Numéro_d'erreur représente le numéro d'identification de l'erreur à simuler.

Cette simulation provoque soit l'édition du message d'erreur correspondant et la fin du programme ou soit lors de l'exécution de la ligne d'instruction ON ERROR GOSUB, le branchement de la procédure notifiée.

FATAL — Délivre le type d'erreur

Var=FATAL

Variable réservée. Une différenciation entre "erreur normale" et "erreur système" y est portée. Lors du surgissement d'une erreur système (l'adresse de la dernière instruction BASIC exécutée n'étant plus connue), elle contient un -1. Dans tous les autres, une valeur nulle : 0 y est retournée.

0 Une erreur ordinaire BASIC. Les adresses des variables et des labels, comme la pile GOSUB sont intactes.

-1 Erreur système fatale. Le registre des adresses est détruit, ce qui en cas ordinaire a pour conséquence l'apparition d'une Guru-Meditation. Il vous est possible en GFA BASIC de rattraper ce type d'erreur puisque le système, lors des erreurs système ordinaires, sauvegarde les contenus des registres en une zone mémoire particulière dont les données peuvent être lues par la suite à fin de réinitialisation, du moins en l'absence d'un blocage complet du système.

ON ERROR (GOSUB) — Branchement après erreur

ON ERROR GOSUB Procédure
ON ERROR

La première forme syntaxique amène suite à l'apparition d'une erreur système ou BASIC, le branchement de la procédure spécifiée. Dans ce cas, nul message d'erreur n'apparaît, la gestion de l'erreur est laissée au programmeur. Il vous est possible à l'aide de numéros d'erreur: ERR (et ERR$) d'éditer des messages d'erreurs personnels ou d'insérer un branchement dans votre programme afin de traiter l'erreur ou de le laisser se poursuivre tout simplement (Cf. RESUME).

Après la sortie d'une routine d'erreur, l'interpréteur commute la détection des erreurs propre au mode ordinaire. S'il y a de nouveau nécessité d'activer ON ERROR GOSUB, la routine d'erreur doit de nouveau être spécifiée dans la routine de détection avant le saut RESUME.

La seconde forme syntaxique commute le mode erreur ordinaire. Les messages d'erreur habituels y apparaissent de nouveau (insérez dans les programmes compilés l'OPTION correspondantes) et amènent l'interruption du programme.

La version 3.0, autorise l'omission de l'énoncé GOSUB lors de l'insertion de la première forme syntaxique. Il est inséré automatiquement par l'éditeur. Exemple sous Fatal.

RESUME [RESU] Poursuit le programme après une erreur

RESUME =>après la routine d'erreur, poursuit le programme avec la reprise de la ligne litigieuse
RESUME NEXT =>après la routine d'erreur, poursuit le programme avec la ligne suivant celle litigieuse
RESUME Label =>après la routine d'erreur poursuit le programme dans ligne Label spécifiée

Détermine en quelle ligne le programme doit se poursuivre suite à l'apparition d'une erreur et à son traitement (Cf. ON ERROR GOSUB). RESUME n'est utilisable qu'au sein de procédures. Son application en tant que routine de gestion d'erreur est chose fort sensée. L'instruction de RESUME Label assure aussi la sortie d'une procédure.

Exemple :

```
Label:                  ! Label de saut
PRINT 111               ! édite quoique ce soit
@ Routine               ! appel de routine
PROCEDURE Routine
  RESUME Label          ! saut au Label
RETURN
```

Si lors de l'emploi de forme syntaxique RESUME Label, le Label spécifié est placé en dehors de la routine contenant RESUME Label, la pile des sauts GOSUB est effacée et toutes les variables globales sont restaurées.

Après des erreurs FATAL (Cf. FATAL), il convient d'utiliser exclusivement RESUME Label. RESUME NEXT et RESUME pourraient éventuellement en ce cas amener un blocage de la machine.

Un exemple d'application de cette instruction est proposé sous FATAL.

12.5 Informations

CRSCOL — Retourne la colonne du curseur en cours

Var=CRSCOL

Variable réservée, contenant la position de la colonne où se trouve actuellement le curseur (CRSCOL = CuRSCOLumn).

A l'inverse de la fonction POS() délivrant la position relative du curseur en considération de la chaîne où il se trouve, il est possible d'établir à l'aide de CRSCOL la position de la colonne du curseur sur l'écran (Hires/Lowres: 1 - 80/1-60). La combinaison de CRSLIN et de CRSCOL représente l'inverse de PRINT AT(C,L).

Un exemple de l'application de cette fonction est exposé sous CRSLIN.

CRSLIN — Délivre la ligne du curseur en cours

Var=CRSLIN

Variable réservée, contenant la position de la ligne où se trouve le curseur (CRSLIN = CuRSorLINe). CRSLIN avec la fonte standard est toujours compris et cela en toutes les résolutions, entre 0 et 25. Suite à la spécification de Font = 1 dans la procédure Sysfont (Cf. ASC()) en haute résolution (Hires), CRSLIN s'échelonne de 1 à 50.

La combinaiseon de CRSCOL et de CRSLIN représente l'inverse de PRINT AT(C,L).

Exemple 1 :

```
FOR i%=0 TO 500                 ! 500 fois
  PRINT "Test CRSCOL/CRSLIN "; ! édite le texte
  IF CRSCOL>50                  ! la position du curseur > 50?
     PRINT                      ! alors insertion d'une ligne
  ENDIF
  IF CRSLIN=25     ! le curseur est-il dans la ligne inférieure?
     CLS                        ! effacement de l'écran
  ENDIF
NEXT i%
```

Exemple 2 :

```
a$="Ecriture Inverse"
PRINT AT(37,1);                 ! positionne le curseur
DO                              ! boucle des lignes
  PRINT AT(37,CRSLIN+1);        ! déplace le curseur
  DO                            ! boucle des colonnes
    PRINT AT(CRSCOL-2,CRSLIN);! déplace le curseur de 2
    '                           ! colonnes vers la gauche
    PRINT MID$(a$,(36-CRSCOL),1); ! édite les caractères
    EXIT IF CRSCOL<22           ! sortie si le curseur est
                                ! devant
    '                           ! la 22ème colonne
  LOOP
  EXIT IF CRSLIN>23             ! sortie si le curseur est sous
  '                             ! la 23ème ligne
LOOP
```

FRE() — Retourne les octets disponibles en RAM

Var=FRE(Dummy)
Var=FRE()

Cette fonction avant de retourner la place mémoire libre, procéde d'abord à un nettoyage de la mémoire (garbage collection). Dummy est une valeur entière factice.

L'interpréteur veille constamment à l'espace de mémoire disponible. A cette fin, les zones de variables superflues (par exemple celles des variables de chaînes ou de tableaux précédemment effacés) sont

supprimées et celles voisines sont rassemblées. Cette tâche est remplie sporadiquement par le BASIC. Sa manifestation affleure quelques fois à l'écran en induisant l'impression du bégaiement du programme, un 10ème de seconde durant.

Pour réaliser un nettoyage de la mémoire (garbage collection), FRE() est utilisable même si la délivrance de la place mémoire disponible n'est d'aucun intérêt. Il est conseillé d'insérer FRE immédiatement avant une recherche d'adresse de variable (VARPTR() ou ARRPTR()) car les positions des adresses de variables (en particulier celles des chaînes) peuvent être entre temps déplacé par l'exécution d'une Garbage Collection interne.

En la version 3.0 Amiga, l'argument muet (Dummy) est optionnel. Lors de l'indication de parenthèses vides, aucun nettoyage de la mémoire n'est assuré, seule la délivrance de l'espace mémoire disponible est opérée.

TYPE() — Délivre le type de la variable

Var=TYPE(Pointeur)

Pointeur représente le pointeur (astérisque: *), par exemple: PRINT TYPE(*Var%). Cette fonction délivre le type de la variable ainsi pointée.

Type :

```
-1 = apparition d'une erreur
 0 = variable réelle (Var)
 1 = variable de chaîne (Var$)
 2 = variable entière de 4 octets (Var%)
 3 = variable booléenne (Var!)
 4 = variable matricielle réelle (Var())
 5 = variable matricielle de chaîne Var$()
 6 = variable matricielle entière de 4 octets
 7 = variable matricielle booléenne (Var!())
 8 = variable entière de 2 octets (Var&)
 9 = variable entière de 1 octet (Var)
12 = variable matricielle entière de 2 octets (Var&())
13 = variable matricielle entière de 1 octet (Var())
```

Exemple :

```
A=10.1
A$="ABC"
A%=10
A!=-1
PRINT TYPE(*A)'TYPE(*A$)'TYPE(*A%)'TYPE(*A!)
```

12.6 Le multitâche

AFTER x GOSUB [AF] — Appel de routine d'interruption

AFTER Tops [GOSUB] Procédure

Appelle après un nombre déterminé de tops (1 seconde = 200 tops), la procédure notifiée. La commande n'est exécutée d'habitude que suite à l'exécution complète d'une instruction BASIC. Lorsque par exemple, le déroulement du programme est interrompu par SOUND x,x,x,x,Durée, PAUSE, DELAY, INP(), INPUT, BLOAD, etc..., le branchement de la procédure spécifiée n'est entrepris que suite à la poursuite du programme.

Exemple du grand jeu d'Emile Franc :

```
AFTER 100 GOSUB abc        ! appel AFTER
t%=TIMER
INPUT a$                   ! l'exécution d'AFTER est interrompue
DO
LOOP
PROCEDURE abc
  PRINT "AFTER 100 GOSUB seulement après ";TIMER-t%
  PRINT "100 tops se sont écoulés"
  END
RETURN
```

Note: Il est jusqu'à présent impossible de traiter simultanément plusieurs instructions d'interruption AFTER. La dernière insérée (AFTER GOSUB ou EVERY GOSUB) suspend l'effectivité des précédentes.

Exemple :

```
AFTER 1000 GOSUB aa          ! premier appel
AFTER 400 GOSUB bb           ! deuxième appel
DO
```

```
LOOP
PROCEDURE aa                    ! procédure pour le 1er appel
  PRINT 11111
RETURN
PROCEDURE bb                    ! procédure pour le 2ème appel
  PRINT 22222
RETURN
```

Dans notre exemple, seule la procédure du second appel est menée au bout de 2 secondes, une unique fois. Le second appel invalide le premier appel. PRINT 1111 ne sera donc jamais exécuté.

Toutefois il est possible de faire d'une intruction AFTER x GOSUB, une seconde EVERY x GOSUB. A cette fin, il suffit simplement d'iniatiliser une nouvelle fois la procédure AFTER x GOSUB, avant le RETURN qui lui correspond. Néanmoins, il faut veiller à ne pas atteindre le même durée que celle notifiée dans EVERY, puisque l'initialisation revendique pour elle même une donnée temporelle. Toutefois la présence d'une telle boucle d'interruption complémentaire à EVERY est le plus souvent préférable à son absence.

Exemple :

```
AFTER 60 GOSUB PROC1            ! initialisation d'AFTER
EVERY 100 GOSUB PROC2           ! initialisation d'EVERY
DO                              ! boucle perpétuelle
LOOP
PROCEDURE PROC1                 ! procédure AFTER
  PRINT "Procédure AFTER"
  AFTER 60 GOSUB PROC1          ! nouvelle initialisation d'AFTER
RETURN
PROCEDURE proc2                 ! procédure EVERY
 PRINT "procédure EVERY"
RETURN
```

AFTER CONT [AF CONT] Libère une routine d'interruption

AFTER CONT

Réactive après AFTER STOP le contrôle de simple interruption (Cf. AFTER STOP)

AFTER STOP [AF STOP] Suspend la routine d'interruption

AFTER STOP

Interrompt le contrôle de simple interruption. AFTER STOP permet de différer l'exécution de la procédure AFTER x GOSUB actuelle jusqu'à la prochaine insertion GOSUB. En l'absence d'une insertion d'AFTER CONT, AFTER x GOSUB reste inactif jusqu'à la fin du programme, à moins qu'un nouveau Timer ait été installé par un nouvel AFTER x GOSUB.

A titre d'illustration, nous présupposons le branchement d'une procédure AFTER, dépendant de l'écoulement de trois secondes. L'exécution y est pourtant suspendue après deux secondes par AFTER STOP et poursuivie après deux nouvelles secondes par AFTER CONT, ainsi AFTER x GOSUB, n'est exécuté que suite à l'écoulement de 4 secondes, bien que le temps d'attente de 3 secondes ait été dépassé. D'autres appels AFTER CONT pour le même processus, n'ont plus d'effectivité car la routine n'est exécutée qu'une seule fois.

Exemple :

```
t%=TIMER
AFTER 600 GOSUB proc1          ! initialisation d'AFTER
PRINT "AFTER 600 GOSUB est active"
EVERY 400 GOSUB proc2          ! initialisation d'EVERY
DO                             ! boucle perpétuelle
LOOP
PROCEDURE proc1                ! procédure AFTER
  PRINT "Procédure AFTER a pris : ";TIMER-t%;" Tops !"
  EDIT
RETURN
PROCEDURE proc2
  a%=a% XOR 1
  IF a%=1
    PRINT "AFTER 600 GOSUB interrompue"
    AFTER STOP
  ELSE
    PRINT "AFTER 600 GOSUB réactivée"
    AFTER  CONT
  ENDIF
RETURN
```

EVERY x GOSUB [EV] Appel de routine d'interruption

EVERY Tops [GOSUB] Procédure

Appelle constamment suite à l'écoulement du nombre déterminé de tops (1 seconde = 200 tops), la procédure notifiée. La commande n'est exécutée d'habitude que suite à l'exécution complète d'une instruction BASIC. Lorsque par exemple, le déroulement du programme est interrompu par SOUND x,x,x,x,Durée, PAUSE, DELAY, INP(), INPUT, BLOAD, etc., la procédure spécifiée n'est branchée que suite à la poursuite du programme.

Malheureusement les fans GFA durent attendre bien longtemps la disponibilité de cette instruction au sein de leur langage favori. En outre veuillez vous reporter à l'annotation exposée sous AFTER GOSUB.

EVERY CONT [EV CONT] Libère une routine d'interruption

EVERY CONT

Réactive après EVERY STOP le contrôle d'interruption EVERY. EVERY GOSUB devient de nouveau effectif.

EVERY STOP [EV STOP] arrête une routine d'interruption

EVERY STOP

Interrompt une contrôle d'interruption EVERY. EVERY GOSUB n'est plus exécuté ; sa réactivation est opérée à l'aide d'EVERY CONT.

12.7 Le débogage

DUMP (DU) Retourne nom et contenu de variables

DUMP [Defstring$] [TO Fichier$]

L'emploi de DUMP en l'omission des paramètres, amène l'édition des contenus de toutes les variables et des dimensions de tous les tableaux. L'emploi de Defstring$ permet d'affiner l'édition :

"a" Les contenus de toutes les variables et les dimensions de tous les tableaux, dont le nom débute avec la lettre a, sont retournées.

":" Tous les noms de label et leur numéro de ligne sont édités.

":a" Tous les noms de label débutant par la lettre a et leur numéro de ligne sont renvoyés.

"@" Tous les noms de procédures et de fonctions, de même que leur numéro de ligne sont retournés.

"@a" Tous les noms de procédures et de fonctions débutant avec la lettre a, de même que leur numéro de ligne sont retournés.

Le code @ est adjoint en fin des noms de procédures retournés. Il est suivi, du moins si la procédure existe encore dans le programme en cours, du numéro de ligne à laquelle la procédure débute. Il en va de même pour les noms des fonctions encore disponibles. Ces derniers sont codés avec les lettres FN. Les noms de fonctions de chaînes sont distinguées à l'aide de l'adjonction complémentaire d'un $.

Les procédures, les fonctions et les labels qui ont été définis et de nouveau supprimés dans le programme, sont représentées sans leur numéro de lignes. A l'inverse, les variables n'étant plus existantes, qui sont toutefois retournées car encore présentes sur la liste de référence interne, ne sont pas reconnaissables du premier coup d'oeil. Le seul signe distinctif qui leur est propre est leur absence de contenu (0 ou ""), bien que des variables existantes puissent aussi se présenter ainsi.

Les noms superflus sur la liste de référence interne sont supprimés à l'aide de Save,A, de New et ensuite Merge (Cf. 24.5 L'éditeur).

La délivrance du contenu d'une variable de chaîne est limitée à 60 caractères maximum. Le texte retourné est placé entre guillemets. Le contenu de la variable étant supérieur à 60 caractères, un signe supérieur : > vient clôre le texte retourné. Des caractères de code ASCII inférieurs à 32 étant disposés dans la variable, ces caractères sont représentés à l'aide d'un point. En outre il est possible indépendamment de l'option Defstring$, de diriger la sortie des informations sur un fichier, grâce à l'indication complémentaire d'un nom de fichier précédé de TO. Ce nom de fichier est complété automatiquement de l'extension .DMP, du moins en l'absence de toute spécification d'extension. Lors de l'existence d'un fichier de nom identique, le précédent est dénonmmé avec l'extension .BAK.

TRACE$ Livre la ligne de commande en cours au mode trace

Var=TRACE$

La variable réservée TRACE$ assure la délivrance du texte de la ligne d'intruction en cours en préalable à son exécution. Ceci à l'intérieur d'une procédure initialisée par TRON Proc.

TROFF [TROF] Eteint le mode trace

TROFF

TROFF commute le mode d'exécution ordinaire et interrompt de ce fait le mode trace initialisé par TRON - ou TRON Proc en version 3.0. A l'intérieur d'une routine TRON Proc, TROFF est sans effectivité.

TRON Proc [TR] Dirige le mode trace en une procédure

TRON Nom_de_procédure

Elle représente une amélioration propre à la version 3.0 de l'instruction fort connue TRON. Elle dirige le mode trace sur la procédure dont le nom est spécifié. L'édition de la ligne d'ordre en cours n'y est pas automatique comme d'ordinaire avec TRON, mais de quelconque réactions peuvent y être entreprises suite à l'édition par TRACE$ de la prochaine ligne d'instruction du programme ou de celle des contenus des variables par DUMP.

La désactivation de TRON Proc est elle aussi assurée par TROFF.

TRON ne vous est d'aucun secours lors de l'écriture d'une routine de déboggage personnelle car tant les routines d'interruption AFTER x GOSUB et EVERY x GOSUB que la présente procédure de déboggage, sont exclues des éditions TRACE$.

12.8 Manipulations diverses

$ Déclaration de domaine de texte (compilateur 3.0)

$ Texte

Déclare de quelconques lignes de programme sous forme de tampon de texte afin d'une compilation future par le compilateur de la vertion 3.0. L'interpréteur manipule de telles lignes comme les lignes de commentaires REM. De plus précises informations, ne sont à attendre que suite à l'apparition du compilateur de la version 3.0 (un exemple : $ X fonction_externe ...).

DEFLIST [DEFLIS] — Détermine le format du listing

DEFLIST Format

Format :

0 Les noms de commandes sont écrit en majuscules, ceux des variables en minuscules (PRINT var$). En version 3.0, le suffixe de déclaration globale y est omis (suffixe Cf. instructions DEFxxx, DEFBYT, etc.).

1 L'initiale des nom d'instructions et de variables est écrite en majuscule (Print Var$). En version 3.0, le suffixe de déclaration globale y est omis.

DEFLIST n'est utilisable que dans le mode direct de l'interpréteur.

DEFNUM [DEFN] — Arrondit les valeurs éditées

DEFNUM Chiffre

En Chiffre est spécifié le nombre de chiffres significatifs (de 3 à 13) affecté aux valeurs à éditer avec PRINT. Cette détermination des chiffres significatifs n'est modifiée que par la prochaine instruction DEFNUM. La notification du nombre maximal (13) amène l'édition de la valeur usuelle.

Les nombres réels dont le nombre de chiffres avant la virgule est supérieur à celui des chiffres significatifs spécifiés, sont arrondis au dernier rang significatif et complétés par des 0. Les chiffres des nombres réeels situés après la virgule et après le dernier chiffre significatif sont arrondis sur le dernier rang significatif.

La fonction arrondit les chiffres conformément aux règles mathématiques usuelles (Int(A+0.5)). Les valeurs contenues par les variables restent inchangées lors l'utilisation de DEFNUM. Elle n'influe aucunement la précision des calculs internes, menés par l'ordinateur.

```
a=RANDOM(1000000)+RND
PRINT "Valeur originale: ";a
FOR i%=3 TO 11
  DEFNUM i%
  PRINT a,"DEFNUM ";i%
NEXT i%
```

FALSE — Constante de fausseté

Var=FALSE

Variable réservée, contenant la valeur nulle : 0.

LET [LE] — Affectation de données

LET Var=Valeur
LET Var$=Texte

Assigne une valeur ou un texte à la variable notifiée. Cette instruction est facultative puisque les variables ne connaissent plus de limitations (PI, ERR, FATAL, TIMER, etc... mises à part) et que leur syntaxe est significative. LET n'obéit plus qu'à un besoin de compatibilité avec les autres langages BASIC (par exemple l'AmigaBASIC) dont on peut charger les programmes en mémoire utilisateur GFA à l'aide de Merge.

MODE [MOD] — Sélection du format numérique ou horaire

MODE Mode

Le format numérique observé par défaut lors d'une édition PRINT USING en GFA BASIC est le format européen. Les séparations des milliers y sont signifiées à l'aide de la virgule et celles des décimales à l'aide du point. Aux Etats-Unis, les conventions inverses prévalent. Il en va de même pour la représentation de la date (Cf. SETTIME).

Mode est à considérer tel un vecteur de 2 bits, dont le bit 0 détermine la représentation de la date propre à DATE$, FILES, SETTIME et DATE$=. Bit 1 décide du mode de représentation numérique d'une édition PRINT USING ou celle assurée par STR$().

Mode	USING	DATE$
&X00	#,###.##	25.05.1988
&X01	#,###.##	25/05/1988
&X10	#.###,##	25.05.1988
&X11	#.###,##	25/05/1988

TRUE — Constante de vérité

Var=TRUE

TRUE est une variable réservée, contenant la valeur constante: -1. Diverses fonctions (par exemple: EXIST) retournent une valeur de vérité. Afin d'assurer une meilleure lisibilité, l'insertion de la constant TRUE peut suppléer celle de -1 lors de tests conditionnels, de spécifications de Flags binaires ou d'affectations (Voir par ailleurs FALSE).

Exemple 1 :

```
FILESELECT "FileBox","Charger","SYS:",F$ ! sélection de fichiers
IF EXIST (F$)=TRUE                       ! le fichier existe-t-il?
...suite du
...programme
```

Exemple 2 :

```
DO                        ! boucle perpétuelle
  IF MOUSEK=1             ! pression du bouton gauche de la souris?
    bitflag!=bitflag! XOR TRUE ! permutation on/off du flag à
    '                     ! chaque pression du bouton de la souris
    onoff%=ABS(bitflag!=TRUE) ! donne: on = 1/off = 1
    PRINT AT(1,1);RIGHT$("on ",3*onoff%)
    PRINT AT(1,1);RIGHT$("off",3*ABS(onoff%=FALSE))
    PAUSE 6               ! petite pause de cliquement
  ENDIF
  CIRCLE MOUSEX*onoff%,MOUSEY*onoff%,10*onoff%
LOOP
```

D'emblée, vous concluez certainement de l'inutilité des deux premières lignes d'instruction. L'insertion en ces lignes d'un test de vérité, m'évite celle d'un test conditionnel correspondant: IF. La variable onoff% contient nécessairement la valeur 0 ou celle 1, bitflag! -1 ou 0. Avec la construction RIGHT$, je n'autorise l'édition de la chaîne que suite à la multiplication de la valeur 3 (la longueur de la chaîne) par 1. Si onoff% contient une valeur nulle (0), la longueur de la chaîne RIGHT$ est nécessairement nulle et aucune chaîne n'est éditée.

En une procédure ordinaire, la construction apparaîtrait comme suit :

```
IF bitflag!=TRUE
  PRINT AT(1,1);"on "
ELSE
  PRINT AT(1,1);"off"
ENDIF
```

SWAP [SW] Echange de variables, tableaux et pointeurs

SWAP Var1, Var2
SWAP Elément(x), Elément(y)
SWAP Matrice1(), Matrice2()
SWAP *Pointeur, Matrice()

Il convient de ne pas confondre l'instruction SWAP avec la fonction SWAP() disponible elle aussi dans la version 3.0.

SWAP échange les contenus de deux variables de même type, de deux matrices ou élements de matrice (numériques ou alphanumériques), un pointeur d'un descripteur de matrice (Cf. *) avec le descripteur de la matrice notifiée (des transmissions de matrice en des procédures). Les dimensions des matrices sont échangées simultanément avec les contenus. Il n'y pas lieu avec SWAP *Pointeur,Matrice() de dimensionner en préalable la matrice spéficiée.

Exemple 1 :

```
A%=100
B%=1000
PRINT "A%, B% avant SWAP: ";A%'B%
```

```
SWAP A%,B%
PRINT "A%, B% après SWAP: ";A%'B%
```

Exemple 2 :

```
DIM A%(10)
PROCEDURE Routine(X%)
  SWAP *X%,Dummy%()
  .
  ... à partir d'ici la précédente matrice A%() est appelable
  ... avec le nom Dummy%(). La matrice A%() est vide.
  ...
  ... texte de la routine
  .
  SWAP *X%,Dummy%()
  .
  ... à partir d'ici la matrice Dummy%() est de nouveau celle
      A%() et la matrice Dummy%() est vide.
  .
RETURN
```

L'avantage de la transmission indirecte de matrices en des procédures est d'assurer le traitement et l'appel de la matrice échangée et transmise par SWAP dans la procédure, non pas sous son nom propre (globale), mais sous un nom local.

Cette construction se montre superflue dans la version 3.0 puisque la transmission directe des matrices et des variables s'y opère à l'aide de VAR.

Exemple :

```
DIM Matrice(1)
xyz(Matrice%())
PRINT Matrice%(1)
PROCEDURE Xyz(VAR MatriceDummy%())
  MatriceDummy%(1)=100
RETURN
```

VOID [V] — Assignation muette

VOID Fonction

VOID inhibe le retour des informations opérées par les fonctions et amène de ce fait des vitesses de traitement bien plus rapides (en particulier pour les programmes compilés). Elle assure l'appel d'une fonction sans que cette dernière ait à retourner une variable. En de nombreux cas, la délivrance de la variable de retour, lors de l'appel d'une fonction se montre inutile, cette information n'ayant guère d'intérêt. Pour épargner de la place mémoire (et du temps), ces fonctions sont appelables avec VOID.

Exemple :

```
au lieu de: A=INP(2)     ! attente de la frappe d'une touche
            VOID INP(2)
au lieu de: A=FRE(0)     ! nettoyage de la mémoire (garbage
                           collection)
            VOID FRE(0)
```

Chapitre 13

Instructions relationnelles (Programme/Utilisateur)

ALERT [A] Création d'une boîte d'alerte

ALERT Icône%, Boxtext$, Def_button%,Buttontext$, Backvar%

Cette fonction est outil spécifique de dialogue entre l'utilisateur et la machine. Elle permet même la gestion d'innombrables menus. Néanmoins seules trois options au maximum y sont définissables. Elle est de ce fait particulièrement propre à la communication de brèves interrogations ou de notes.

Icône%

Représente une valeur qui sert à sélectionner un style d'icône particulier. Malheureusement à ce jour, cette fonction n'est pas encore implémentée dans la version 3.03.

Boxtext$

Les textes propres à la boîte d'alerte (informations ou questions) sont notifiés dans cette chaîne. Le symbole pipe: | est le séparateur des lignes isolées. Cinq lignes de 40 caractères maximum y sont insérables. L'insertion directe ou la transmission du texte sous forme d'expression ou de variable de chaîne est possible.

Def_button%

Les numéros de boutons (1, 2, 3) dont l'activation est désirée, sont spécifiés dans ce paramètre. Ce ou ces boutons sont exposés distinctement dans la boîte d'alerte, entourés d'un cercle plus gras (0 = nul bouton default) et sélectionnables par la pression de la touche <EU><R> ou par le cliquement de la souris suite à leur pointage.

Buttontext$

L'inscription du texte du bouton se réalise en ce lieu. Le symbole pipe: | est le séparateur des lignes isolées du texte des boutons.

Backvar%

La présente variable numérique assure la délivrance du numéro du bouton dont la validation a été opérée.

Exemple :

```
boxtext$="Veuillez cliquer l'une des options"
boxtext$=boxtext$+"|Puis je désignerai la bonne"
buttontext$="XX 1 XX|XX 2 XX|XX 3 XX"
ALERT 0,boxtext$,0,buttontext$,backvar%
boxtext$="C'était:  |  | l'option "+STR$(backvar%)
buttontext$="OK  |OK  |OK  "
ALERT 0,boxtext$,2,buttontext$,backvar%
```

La forme de la boîte d'alerte connaît des limites strictes pour des raisons de sécurité. Les indications qui y sont portées lors de sa création, sont vérifiées entièrement par l'interpréteur. A titre d'illustration, si jamais lors de l'insertion, une ligne contenait plus de 40 caractères, cette ligne serait tronquée par l'interpréteur au quarantième signe.

Ceux qui connaissent la construction d'une boîte de requête (Request Box), comprendront les limitations de taille observées.

FILESELECT (FILE) Sélection de fichiers

FILESELECT "Titre","OKTexte","Chemin",Backvar$

Cette instruction autorise la création d'une fenêtre de sélection de fichiers et retourne le nom du fichier sélectionné (ou selon, son chemin d'accès) choisi en Backvar$.

L'information retournée par Backvar$ peut revêtir quatre formes distinctes:

1. Suite à la sélection d'un fichier par l'usager, la variable contient son nom et son chemin d'accès en complet.

2. Nulle sélection n'ayant été opérée dans la liste, deux possibilités se présentent :

   A La transmission par défaut d'un nom de fichier dans laligne d'insertion du sélecteur et la confirmation par OK ont été opérées, le nom de ce fichier ou selon son chemin d'accès est donc contenu dans Backvar$.

   B En l'absence de la sélection et de la transmission d'un nom: l'usager ayant supprimé le contenu de la ligne d'insertion et validé OK, une chaîne vide est retournée en Backvar$.

3 Suite au cliquement d'Abandon, Backvar$ est absolument vide.

Cette instruction est l'un des outils essentiels de la gestion des contenus de disquettes. Le nombre de fonctions diverses assurées par cet ordre dont la simplicité d'usage apparaît élémentaire, se laissent deviner aisément puisqu'il supplée les modes ordinaires de chargement, de sauvegarde, de suppression, et de modification des fichiers.

Cette instruction édite en une fenêtre les fichiers disponibles dans le répertoire notifié à l'aide de son chemin d'accès. Tous les noms distingués à l'aide d'une astérisque, y représentent des sous-répertoires. Le cliquement d'un nom de sous-répertoire amène l'édition de son propre contenu et donc la sélection possible des fichiers qui y sont répertoriés. La sortie d'un sous-répertoire résulte du cliquement de la petite case rectangulaire en haut à gauche sous la ligne de titre.

Puisque le GFA BASIC déclare toujours le lecteur à partir duquel le programme a été chargé, comme celui en cours, la modification de l'identificateur du lecteur notifié dans le chemin d'accès peut être requise. La sélection de l'un des périphériques proposés sous la fenêtre d'édition des répertoires résulte du simple cliquement de l'option désirée, disposée en cette ligne. L'indication du nom de chemin peut aussi être entreprise à travers le gadget String. Suite à l'insertion, la pression de la touche <EU><R> amène d'elle-même pour ainsi dire l'édition du répertoire.

A vrai dire, il s'ensuit l'exécution invisible d'une routine qui gère l'édition, vérifie la justesse du nom de fichier inséré, ou réalise une copie de sauvegarde.

MOUSE { MOU } — Délivre le statut complet de la souris

MOUSE Xpos, Ypos, Touches

En Xpos et Ypos, sont retournées les coordonnées de la position en cours du pointeur de la souris, en Touches l'état des boutons de la souris.

```
0 = nulle pression n'a été opérée
1 = pression du bouton gauche    (bit 0    -  2^0      = 1)
2 = pression du bouton droit     (bit 1    -  2^1      = 2)
3 = pression des deux boutons    (bit 0+1  -  2^0+2^1  = 3)
4 = pression du bouton médian    (bit 2    -  2^2      = 4)
```

En considération de ce vecteur de bit, seuls les 2 bits de poids le plus faible recèlent un intérêt certain. Le bit 0 représente le bouton gauche de la souris, le bit 1 celui droit.

Note: Dans la version 3.0, des valeurs négatives sont délivrées en Xpos et Ypos (par exemple avec CLIP OFFSET) dès que le pointeur sort de sa fenêtre (ou dépasse par exemple le point d'origine CLIP).

Exemple :

En liaison avec la souris, se présente toujours le problème de la délimitation de la zone écran pour des définitions et des distinctions futures. En ces raisons, j'ai développé une procédure dont l'adaptation à vos propres besoins et la modification selon vos désirs, est fort aisée.

Toutefois cette routine exécute GRAPHMODE 1, DEFLINE 0,0,0,0 et COLOR 1, de ce fait suite à son exécution, la restauration des réglages précédemment en cours, est requise. Il vous est possible de modifier ces réglages à l'intérieur de la routine, pour atteindre d'autres effets (par exemple: GRAPHMODE 3 ou COLOR 0).

Le fonctionnement de cette procédure est fondamentalement identique à celui de la fonction Workbench. Toutefois peu importe ici si le bouton droit ou celui de gauche ou les deux sont pressés. Tout comme la fonction Workbench, la procédure est appelée à l'aide de la pression des deux touches, car sinon elle est instantanément interrompue. Une différence essentielle est que cette procédure traite aussi des valeurs négatives, autrement dit les coordonnées de la position du pointeur de la souris peuvent être inférieures à celles du point d'origine. Dans ce cas, des valeurs négatives sont retournées dans les variables.

```
DEFFILL ,2,4                ! DEFFILL gris
PBOX 10,10,200,200          ! trame le fond
DO                          ! boucle perpétuelle
  IF MOUSEK                 ! les boutons ont-ils été pressés?
    MOUSE x%,y%,k%          ! cherche les coordonnées de départ
    rubberbox(x%,y%,-30,-30,*xx%,*yy%) ! Appel
    BOX x%,y%,x%+xx%,y%+yy% ! trace le rectangle
  ENDIF
LOOP
'
PROCEDURE rubberbox(xp%,yp%,xmin%,ymin%,xret%,yret%)
  ' Xp%   = coordonnée X de départ
  ' Yp%   = coordonnée Y de départ
  ' Xmin% = largeur minimale de rubberbox (aussi négative)
  ' Ymin% = hauteur minimale de rubberbox (aussi négative)
  ' Xret% = pointeur sur un entier de 4 octets contenant à la
  ' fin de l'exécution la dernière largeur de box (aussi
                                                    nég.)
  ' Yret% = pointeur sur un entier de 4 octets contenant à la
  ' fin de l'exécution la dernière hauteur de box (aussi
                                                    nég.)
  LOCAL sc1$,sc2$,sc3$,sc4$,mx1%,my1%,mx2%,my2%,mk%
  DEFLINE 0                 ! DEFLINE plein/fin
  COLOR 1                   ! COLOR noir
  GRAPHMODE 1               ! mode Replace
  REPEAT
    MOUSE mx1%,my1%,mk%     ! cherche nouvelles coordonnées
    mx2%=MAX(xp%+xmin%,mx1%) ! arrime coordonnée X sur
                             ! minimum
    my2%=MAX(yp%+ymin%,my1%) ! arrime coordonnée Y sur
                             ! minimum
    GET
MIN(xp%,mx2%),MIN(yp%,my2%),MAX(mx2%,xp%),MIN(yp%,my2%),sc1$
```

```
    GET
MAX(xp%,mx2%),MIN(yp%,my2%),MAX(mx2%,xp%),MAX(yp%,my2%),sc2$
    GET
MIN(xp%,mx2%),MIN(yp%,my2%),MAX(mx2%,xp%),MAX(yp%,my2%),sc3$
    GET
MIN(xp%,mx2%),MIN(yp%,my2%),MIN(mx2%,xp%),MAX(yp%,my2%),sc4$
    BOX xp%,yp%,mx2%,my2%
    REPEAT                          ! attente...
      ON MENU                       ! ... d'événement
    UNTIL MOUSEXmx1% OR MOUSEYmy1% OR MOUSEK=0
    ' ..jusqu'à ce que la souris soit bougée ou un bouton
      pressé
    PUT MIN(xp%,mx2%),MIN(yp%,my2%),sc4$ !
    PUT MIN(xp%,mx2%),MAX(yp%,my2%),sc3$ ! restaure le fond
    PUT MAX(xp%,mx2%),MIN(yp%,my2%),sc2$ ! de box
    PUT MIN(xp%,mx2%),MIN(yp%,my2%),sc1$ !
  UNTIL mk%=0             ! abandon si bouton = 0
  *xret%=mx2%-xp%         ! retourne la dernière largeur de
                          ! box
  *yret%=my2%-yp%         ! retourne la dernière hauteur de
                          ! box
RETURN
```

D'autres exemples sur l'interrogation du statut de la souris sont disséminés dans l'ouvrage.

MOUSEX, MOUSEY, MOUSEY — Etats isolés de la souris

Var=MOUSEX donne la position X
Var=MOUSEY donne la position Y
Var=MOUSEK donne l'état des boutons (Cf. MOUSE)

Fonctions d'interrogation isolée des états de la souris (en ce qui concerne la version 3.0, Cf. l'annotation portée sous MOUSE).

STICK (STI) — Détermine le mode d'interrogation du port souris

STICK(Mode) - Instruction -

Détermine le mode d'interrogation du port de la souris. Cette fonction n'a été implémentée dans le GFA BASIC de l'Amiga que pour des raisons de comptabilité et n'a en tant que tel aucune effectivité.

Mode :

```
0 mode souris
1 mode joystick
```

Les instructions MOUSE et STICK() commutent automatiquement le mode correspondant, de ce fait dans la plupart des cas STICK est superflu.

Un exemple de son application est exposé sous STICK().

STICK() Interrogation du port souris dans le mode joystick

Var=STICK(Port_souris

Délivre l'état actuel du port souris spécifié (0 ou 1). L'interrogation de STICK(1) peut être entreprise en l'absence de la commutation préalable du mode joystick opérée à l'aide de l'instruction STICK. A l'inverse STICK(0) ne livre de valeurs utilisables que sous le mode joystick.

Les valeurs suivantes sont communiquées :

```
5     1     9
  \   !   /
   \  !  /
 4-- 0 --8
   /  !  \
  /   !   \
6     1     10
```

Considéré en tant que vecteur de bits, seuls les quatre derniers bits de poids faible de STICK() ont quelque intérêt :

```
bit 0: joystick en haut
bit 1: joystick en bas
bit 2: joystick à gauche
bit 3: joystick à droite
```

Exemple (ce programme présuppose la connexion d'un joystick sur le port 1) :

```
CIRCLE 7,7,6        !
CIRCLE 7,7,4        ! création
LINE 1,7,13,7       ! d'un pseudo-lutin
```

```
LINE 7,1,7,13       !
GET 0,0,15,15,a$
x%=160              ! coordonnée X d'origine
y%=100              ! coordonnée Y d'origine
STICK 1             ! commute le mode joystick
DO
  PRINT AT(1,1);"  ";AT(1,2);"  "
  IF STICK(1) AND 1  ! le joystick est-il agi vers le haut?
    y%=MAX(0,y%-1)   ! amoindrit la coordonnée Y
  ENDIF
  IF STICK(1) AND 2  ! le joystick est-il agi vers le bas?
    y%=MIN(255,y%+1)   ! élève la coordonnée Y
  ENDIF
  IF STICK(1) AND 4  ! le joystick est-il agi vers la gauche?
    x%=MAX(0,x%-1)   ! amoindrit la coordonnée X
  ENDIF
  IF STICK(1) AND 8  ! le joystick est-il agi vers la droite?
    x%=MIN(639,x%+1)   ! élève la coordonnée X
  ENDIF
  VSYNC
  IF STRIG(1)        ! pression du bouton fire du joystick?
    PRINT AT(1,1);"XWX, Bang ! The Future of the RADIO !"
    COLOR 0
    FOR i%=0 TO 6          ! effet
      CIRCLE x%+7,y%+7,i% ! graphique
    NEXT i%                !
    PRINT AT(1,1);"                                        "
  ENDIF
  PUT x%,y%,a$             ! place le pseudo-lutin
LOOP
```

STRIG() — Interrogation du bouton fire du joystick

Var=STRIG(Port_souris)

Délivre l'état actuel du bouton fire sur le port notifié (pression = 1, absence de pression = 0).
Un exemple de l'application de cette fonction est exposé sous STICK().

Chapitre 14

Programmation du fenêtrage et des écrans

L'Amiga est en mesure de gérer un nombre de fenêtres qui dépend de la mémoire disponible. Ses capacités se montrent en ce domaine inverses à ce qu'elles sont sur l'Atari ST, qui vit le premier l'envol victorieux du langage de programmation GFA BASIC. Certes de nombreuses instructions ont été transférées telles quelles. Néanmoins elles ont été complétées de quelques paramètres spécifiques. En outre l'Amiga peut gérer ces nombreuses fenêtres en divers écrans dont les propriétés graphiques sont distinctes. Le GFA BASIC possède des instructions répondant de cette particularité de l'Amiga.

La distinction des instructions de fenêtre (en anglais window) et de celles d'écran (en anglais screen), qui possèdent pour la plupart le même radical (Cf. OPEN, CLOSE, MOVE...), est opérée par l'adjonction terminale d'un W pour celles de fenêtre et d'un S pour celles d'écran.

14.1 Les instructions de fenêtrage GFA BASIC

CLEARW [CLW] — Supprime le contenu d'une fenêtre

CLEARW [#]Numéro

La fenêtre dont le numéro (de 0 à 15) est ainsi spécifié, est supprimée. La commande CLS peut aussi être utilisée. Toutefois la présente instruction n'efface que le contenu de la fenêtre, non pas la totalité de l'écran.

L'effacement du contenu de la fenêtre est toujours requis suite au déplacement ou à l'amoindrissement de la taille de cette dernière. Vous vous rendrez compte qu'avec le déplacement ou l'agrandissement de la fenêtre, les contenus de fenêtre se superposent. La gestion des contenus de fenêtre est de votre entière responsabilité, vous êtes seul à savoir ce

qu'il faut faire de leur contenu en certaines circonstances. Toutefois la gestion du fenêtrage est assurée à partir de d'Intuition mais cette dernière requiert bien plus d'espace mémoire. Veuillez consultez à cette fin la description d'OPENW.

Les éditions instruites à l'aide de PRINT ou toutes autres éditions de texte peuvent être aisément restaurées à l'aide de boucles, qui assurent le renouvellement de l'écriture des contenus dans leur fenêtre correspondante. A l'inverse le rafraîchissement des contenus graphiques est chose bien plus complexe. En ce cas, l'utilisation d'Intuition est vivement conseillée.

CLOSEW [CLW] — Fermeture d'une fenêtre

CLOSEW Numéro
CLOSEW [#]Numéro

La fenêtre dont le numéro est ainsi spécifié, est close. Des valeurs comprises entre 0 et 15 sont permises.

FULLW [FUW] — Agrandissement maximal de la fenêtre

FULLW Numéro
FULLW [#]Numéro

Numéro détermine le numéro de la fenêtre (0 - 15), dont l'agrandissement s'effectuera à partir de la ligne écran la plus haute jusqu'à la plus basse. La largeur de la fenêtre sera aussi portée sur celle maximale. Cette instruction qui n'est complètement opérationnelle que depuis la version 3.03 permet de gérer l'Overscan.

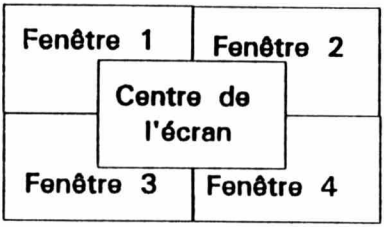
OPENW [OW] Ouverture d'une fenêtre

OPENW Numéro
OPENW Numéro, Xpos, Ypos, largeur, hauteur, IDCMP, Flags [,Numéro_d'écran[,Bitmap]]

OPENW Numéro ouvre, active la fenêtre dont le numéro (O - 15) est ainsi notifié. Les fenêtres créées y sont disposées à l'origine en l'absence de tout chevauchement sur l'écran du Workbench.

La fenêtre 0 occupe l'écran en son entier à partir de sa barre de titre.

```
Les fenêtres 1 à 4
```



Les fenêtres 5 à 15 sont de la même taille que celle 1 à 4 et disposées au centre de l'écran.

La seconde forme syntaxique, autorise la définition à l'aide d'une seule instruction de la fenêtre complète. Elle permet d'éviter la manipulation directe d'Intuition.

Les coordonnées de position déterminent le point d'origine de la fenêtre sur l'écran, autrement dit son coin haut gauche. A celles-ci sont adjointes les hauteur et largeur de la fenêtre en pixels graphiques de la résolution actuelle de l'écran.

Les deux paramètres suivants qui détaillent des propriétés de la fenêtre, sont essentiels car ils sont nécessités impérativement par l'interface Intuition. La première valeur IDCMP, sert à assurer la communication entre la fenêtre et l'usager. Elle détermine quelles sont les informations de la fenêtre à transmettre au programme afin par exemple d'avaliser une information de modification de la taille de la fenêtre. Le paramètre suivant est étroitement dépendant du précédent

puisqu'il définit les caractéristiques propres de la fenêtre. Les gadgets système peuvent être utilisés à cette fin. Certaines spécifications de la gestion de la mémoire graphique réservée à cette fenêtre peuvent y être portées aussi.

Les tables suivantes vous informent des valeurs possibles propres aux deux paramètres et de leur effectivité. Puisque ces valeurs sont originaires de la programmation du système d'exploitation, elles revêtent la forme de flags (drapeaux) de bits. Autrement dit, chaque propriété particulière est représentée à l'aide d'un flag de bits correspondant, ce qui en système décimal amène des valeurs peu habituelles. Ces tables se présentent donc sous la forme de trois colonnes de valeurs. La première colonne donne la valeur en système décimal, la seconde en hexadécimal et la troisième en binaire. Le choix d'un système numérique ne dépend plus de ce fait, que de votre propre initiative.

Les données IDCMP

décimales	hexadécimales	binaires	nom de flag
1	&H00000001	&X0000000000000001	SIZEVERIFY
2	&H00000002	&X0000000000000010	NEWSIZE
4	&H00000004	&X0000000000000100	REFRESHWINDOW
8	&H00000008	&X0000000000001000	MOUSEBUTTONS
16	&H00000010	&X0000000000010000	MOUSEMOVE
32	&H00000020	&X0000000000100000	GADGETDOWN
64	&H00000040	&X0000000001000000	GADGETUP
128	&H00000080	&X0000000010000000	REQSET
256	&H00000100	&X0000000100000000	MENUPICK
512	&H00000200	&X0000001000000000	CLOSEWINDOW
1024	&H00000400	&X0000010000000000	RAWKEY
4096	&H00001000	&X0001000000000000	REQCLEAR
16384	&H00004000	&X0010000000000000	NEWPREFS
32768	&H00008000	&X0100000000000000	DISKINSERTED
65536	&H00100000	&X1000000000000000	DISKREMOVED
262144	&H00400000	&X0000000000000000	ACTIVEWINDOW
524288	&H00800000	&X0000000000000000	INACTIVEWINDOW
1048576	&H00100000	&X0000000000000000	DELTAMOVE
2097152	&H00200000	&X0000000000000000	VANILLAKEY
4194304	&H00400000	&X0000000000000000	INTUITICKS

Les données de flags

décimales	hexadécimales	binaires	nom de flag
0	&H00000000	&X0000000000000000	SMARTREFRESH
1	&H00000001	&X0000000000000001	NEWSIZING
2	&H00000002	&X0000000000000010	WINDOWDRAG

4	&H00000004	&X0000000000000100	WINDOWDEPTH
8	&H00000008	&X0000000000001000	WINDOWCLOSE
16	&H00000010	&X0000000000010000	SIZEBRIGHT
32	&H00000020	&X0000000000100000	SIZEBOTTOM
64	&H00000040	&X0000000001000000	SIMPLEREFRESH
128	&H00000080	&X0000000010000000	SUPERBITMAP
256	&H00000100	&X0000000100000000	BACKDROP
512	&H00000200	&X0000000100000000	REPORTMOUSE
1024	&H00000400	&X0000000100000000	GIMMEZEROZERO
2048	&H00000800	&X0000001000000000	BORDERLESS
4096	&H00004000	&X0000010000000000	ACTIVATE
65536	&H00008000	&X0100000000000000	RMBTRAP
131072	&H00200000	&X1000000000000000	NOCAREFRESH

Exemple :

L'ouverture d'une fenêtre où tous les gadgets système sont disponibles et effectifs, requiert l'insertion de la ligne d'instruction suivante :

OPENW 0,0,0,640,0,4111

L'utilisation des flags IDCMP et les modalités de la transmission des informations sont détaillées sous les descriptions des instructions de menus.

TITLEW (TI) — Titrage des fenêtres

TITLEW Numéro, 'Titre de la fenêtre'['Titre de l'écran']
TITLEW [#]Numéro, 'Titre de la fenêtre'['Titre de l'écran']

La fenêtre dont le numéro est spécifié est titrée par l'énoncé notifié. Il vous est ainsi possible de modifier l'intitulé du titre par défaut: "GFA-BASIC".

En plus cette instruction permet de modifier le titre de l'écran. Il suffit simplement d'y adjoindre un autre texte, qui suite à l'exécution de la ligne d'ordre apparaît dans la barre de titre de l'écran où s'inscrit usuellement la mention: "Workbench Screen" destinée à l'usager. Cette ligne est utilisable à des fins personnelles.

FRONTW [FR] Mise en premier plan d'une fenêtre

FRONTW Numéro

A l'aide de cette instruction, la fenêtre éventuellement exposée en arrière plan est mise en premier plan. Cela importe fort dès qu'il y est question d'entrée ou de sortie de données. Cela vaut aussi pour les fenêtres d'informations devant être contrôlées par l'usager; il est préférable qu'elles soient en premier plan.

Cette instruction est particulièrement importante, lors de l'absence d'un gadget Depth Arrangement dans la fenêtre permettant à l'usager de commuter entre l'arrière et le premier plan.

BACKW [BA] Mise en arrière plan d'une fenêtre

BACKW Numéro

Cet ordre assure la mise en plan de la fenêtre indiquée et "libère" la surface prise pour l'édition de nouvelles fenêtres.

Cette instruction est particulièrement importante, lors de l'absence d'un gadget Depth Arrangement dans la fenêtre permettant à l'usager de commuter entre l'arrière et le premier plan.

MOVEW [MOV] Déplace une fenêtre

MOVEW Numéro, Xpos, Ypos

La fenêtre identifiée par son numéro est déplacée sur les nouvelles coordonnées de l'écran. Elle permet d'assurer la gestion programmée du "dragage" propre aux opérations utilisateur.

Toutefois il convient de veiller à ce que les nouvelles coordonnées correspondent logiquement à la taille de la fenêtre; en cas de débordement de désagréables surprises surgiront inévitablement.

SIZE [SIZ] — Détermine la taille de la fenêtre

SIZEW Numéro, Largeur, Hauteur

La taille d'une fenêtre est déterminable par l'utilisateur grâce au sizing gadget. Mais en certains cas, il serait préférable que vous réserviez cette détermination à votre propre programme afin de prévenir toute manipulation malencontreuse de la part de l'utilisateur.

L'instruction SIZEW n'autorise seulement l'indication de nouvelles hauteur et largeur pour la fenêtre précisément identifiée à l'aide de son numéro.

Il convient d'observer que l'indication d'une largeur amenant le débordement de l'écran par la fenêtre, amène inévitablement un blocage de la machine. Assurez-vous d'avance que la taille indiquée corresponde aux données physiques de l'écran!

LIMIT [LIM] — Limite la taille des fenêtres

LIMITW Numéro, Xmin, Ymin, X max, Y max

Même suite à la mise à disposition du sizing gadget à l'utilisateur afin qu'il puisse modifier de lui-même la taille de la fenêtre, celle-ci ne doit aucunement déborder les limites de l'écran. Une variation de la position X comprise entre 100 et 400 points écran est amplement suffisante.

L'instruction LIMITW assure la limitation de la fenêtre notifiée en considération des valeurs minimales et maximales spécifiées. L'interface Intuition, la couche logicielle (un ensemble de routines) assurant la gestion des écrans, des fenêtres et d'autres éléments, veille alors à ce qu'aucune autre valeur ne soit choisie.

SETWPEN — Détermine le traçage des caractères

SETWPEN Plume_ligne, Plume_surface

Les fenêtres du Workbench sont uniformes. Les lignes sont tracées en couleur blanche et le fond est représenté en couleur bleue. Certes cela n'est pas intangible puisque la sélection d'autres couleurs est offerte.

SETWPEN détermine à l'aide de ses paramètres de nouvelles valeurs standard. Ce qui veut dire qu'il convient de procéder à la modification des couleurs avant l'ouverture de la fenêtre dont on désire les couleurs différentes. Malheureusement il n'est plus possible de modifier ultérieurement les couleurs.

WINDOW() — Recherche de données de fenêtre

WINDOW(Numéro)
WINDOW(Adresse)

En GFA BASIC, les fenêtres sont gérées à l'aide de leur numéro. Il est bien plus simple d'appeler les fonctions à l'aide d'un numéro au lieu de noter la position de mémoire où les données de la fenêtre sont mémorisées. Toutefois on ne peut écarter la nécessité d'avoir à connaître son adresse pour par exemple l'appeler à partir d'une fonction interne au système d'exploitation qui ne traite que des adresses.

WINDOW(Numéro) permet l'affectation en une variable de l'adresse mémoire des données de la fenêtre spécifiée.

L'inversion de cette fonction instruit la délivrance du numéro de fenêtre délivré par le GFA BASIC lors de sa création.

14.2 Les instructions de gestion d'écrans GFA BASIC

OPENS — Ouverture d'un écran

OPENS Numéro [Xpos, Ypos, Largeur, Hauteur, Bitplanes, Mode]

Cette commande ouvre un nouvel écran. L'énumération des écrans s'ordonne de 1 à 15 (comme pour les fenêtres). L'écran par défaut est celui de numéro 0, qui n'est évidemment autre que l'écran du Workbench, précisément celui où s'éditent toutes les fenêtres en l'absence de la création d'autres écrans.

L'ouverture d'un écran nécessite la spécification de ses propres paramètres. L'écran ainsi crée se montre analogue à celui du Workbench et connaît les mêmes manipulations.

En ce faisant, les positions X et Y de l'écran à créer sont déterminées. Toutefois seule la valeur Y intéresse l'interface Intuition, car à ce jour, le processeur graphique n'est pas en mesure de gérer un déplacement d'X. La position X a été implémentée pour des raisons de compatibilité. De ce fait, son annulation autrement dit sa mise à zéro est requise afin de vous éviter de désagréables surprises.

Les largeur et hauteur de l'écran importent fort aussi. Elle déterminent la résolution de l'écran à créer. Un écran peut revêtir une résolution de 1008 x 1024 pixels du moins si la mémoire graphique requise est effectivement disponible (un BitPlane n'occupe pas moins de 131072 octets, ce qui correspond à 128 kilo-octets. Puis le nombre de ces points d'écran à représenter dans la zone visible de l'écran, dépend du mode graphique. Cette zone se limite d'ailleurs à 640 x 256 points, auxquels il convient d'ajouter ceux qui se perdent aux bords de l'écran bien que cela ne tienne aucunement à l'Amiga, plutôt au moniteur.

Und table représentant les diverses résolutions exposée en conclusion de la présente description. Le nombre des BitPlanes est d'ailleurs tout aussi essentiel que la résolution. Nous notions déjà auparavant, que chaque BitPlane mobilise une quantité déterminée de mémoire. Cette mémoire assure la gestion des points activés ou non de l'écran. Chaque

point y est affecté à un bit. L'activation d'un bit amène la visualisation du point correspondant, son inactivation celle de la couleur d'arrière plan de point considéré.

La représentation d'un nombre de couleurs plus élevé que les deux précédentes, nécessite la combinaison de plusieurs BitPlanes et la définition d'un code permettant le calcul de la couleur effective d'affichage. Ce nombre de couleurs dépend évidemment du nombre de BitPlanes :

```
couleurs = 2^BitPlanes
```

La table suivante prévaut en ce qui est de notre présent objet :

BitPlanes	Couleurs	
0	1	(la couleur de fond uniquement)
1	2	
2	4	
3	8	
4	16	
5	32	

L'Amiga n'est pas en mesure de traiter plus de six BitPlanes pour le moment. Bien que l'on y puisse représenter 4096 couleurs différentes, seuls six BitPlanes y sont toujours traités. Toutefois le codage des couleurs y obéit à des sélections différentes.

En conclusion, les suivantes table des modes de l'écran et des valeurs de résolution vous renseigneront sur le nombre de points affichés en regard du mode choisi.

Les modes écran :

modes	décimal	hexadécimal
GENLOCK_VIDEO	2	$0002
INTERLACE	4	$0004
EXTRA_HALFBRITE	128	$0080
HOLD_AND_MODIFY	2048	$0800
SPRITES	16384	$4000
HIRES	32768	$8000

Les différentes résolutions :

description	hexadécimal	résolution maximale	BitPlanes
LORES	$0000	320 x 256	5
HIRES	$8000	640 x 256	4
INTERLACE	$0004	320 x 512	5
HIRES INTERLACE	$8004	640 x 512	4
EXTRA_HALFBRITE	$0080	320 x 256	6
EXTRA_HALFBRITE LACE	$0084	320 x 512	6
HOLD_AND_MODIFY	$0800	320 x 256	6
HOLD_AND_MODIFY LACE	$0804	320 x 512	6

CLOSES — Fermeture d'une fenêtre

CLOSES Numéro

Cette commande simple amène la fermeture de l'écran ouvert, toutes les fenêtres qui y sont disposées sont closes dans le même mouvement ce qui évite d'avoir à les fermer.

FRONTS — Affiche l'écran en premier plan

FRONTS Numéro

FRONTS affiche en premier plan l'écran spécifié à l'aide de son numéro. Il instruit la même fonction que celle résultant de l'activation du gadget correspondant: Depth Arrangement à l'aide de la souris, à la seule différence que son insertion a lieu au sein d'un programme.

BACKS — Amène l'écran en arrière plan

BACKS Numéro

BACKS place l'écran dont le numéro est spécifié, en arrière plan et affiche ainsi le second écran. Il instruit la même fonction que celle résultant de l'activation du gadget correspondant: Depth Arrangement à l'aide de la souris, à la seule différence que son insertion a lieu au sein d'un programme.

MOVES — Déplace l'écran

MOVES Numéro, Xpos, Ypos

Déplace l'écran sur les coordonnées absolues Xpos et Ypos. Cet ordre permet la visualisation partielle d'un écran placé en arrière plan et ainsi la représentation simultanée de résolutions différentes sur l'écran, chose qui répond particulièrement à des sorties mêlées de texte et de graphique.

TITLES — Titrage d'un écran

TITLES Numéro, "Titre"

Comme toute fenêtre, il vous est loisible de modifier le titre d'un écran l'aide d'un quelconque texte.

Cette instruction permet la définition d'un texte qui s'affichera dans la ligne de tête de l'écran dont le numéro est spécifié. Un tel titre est nommé titre par défaut, car il n'est affiché que tant qu'une fenêtre n'est pas activée dans l'écran. Car chaque fenêtre possède un titre complémentaire au sien propre, qui suite à son activation s'édite dans la ligne de tête de l'écran. C'est seulement lors de son absence que l'écran utilise le titre par défaut.

SETSPEN — Sélection des couleurs de l'écran

SETSPEN Crayon_des_lignes, Crayon_des_surface

Comme pour les fenêtres, deux crayons de couleurs sont utilisés pour dessiner les éléments de l'écran: la barre de titre, les gadgets... Ces derniers ont pour valeur par défaut 0 et 1. Mais leur sélection en regard de la résolution, doit être entreprise avant l'ouverture de l'écran, sinon elles ne seront pas effectives car proprement ignorées.

SCREEN() — Cherche des données d'écran

SCREEN(Numéro)
SCREEN(Adresse)

La programmation interne des éléments de l'interface Intuition requiert la connaissance de l'adresse à partir de laquelle les données de l'écran sont mémorisées. Cette dernière est établie par la première fonction qui retourne la position de mémoire correspondante.

La seconde fonction délivre le numéro de l'écran qui quant à lui est requis par les instructions GFA BASIC.

Chapitre 15

Programmation de menus avec le GFA Basic

MENU Menu — Détermine les attributs des options de menu

MENU Option_de_menu, Attribut

Cette instruction permet d'affecter un mode de représentation inactif (ghosted) ou actif à une option de menu. En outre elle vous permet le placement devant l'intitulé de l'option du menu, un signe de cochage (Checkmark). Ce symbole signifie l'activation de la fonction recouverte par l'option suite à sa sélection.

A cette fin, l'indice de l'option du menu dont l'activation ou la non activation est désirée, est notifié en Option_de_menu. L'adjonction de l'attribut désiré s'ordonne suite à l'insertion préalable d'une virgule. Les attributs se décomposent comme suit :

```
  64    enlève le signe de cochage
 254    place le signe de cochage
  16    inactive l'option de menu
  64    active l' option de menu
```

Lors de l'utilisation de MENU Texte_de_menu, il convient en cas d'attribution du cochage, de réserver deux espaces dans l'intitulé d'option correspondant.

MENU Texte — Modifie l'intitulé d'une option de menu

MENU Option_de_menu, Texte

Suite à la suppression, à l'extension ou à la modification d'une fonction recouverte à l'aide d'une option de menu, cette instruction vous permettra de modifier aisément et en conséquence l'intitulé de l'option considérée, précédemment définie à l'aide d'une matrice.

En Option_de_menu, est notifiée l'indice de l'option de menu désirée. Le texte de remplacement y est notifiée sous forme de chaîne en Texte. Veillez observer que la longueur du texte à transmettre est limitée par celle du précédent, puisque le GFA Basic utilise le réceptacle du texte précédent lors de l'écriture du nouvel intitulé d'option. Aucune zone de mémoire complémentaire n'est disponible pour l'écriture d'un intitulé plus long.

MENU KEY — Affecte une touche à une option de menu

MENU KEY Option_de_menu, Code_ASCII

La sélection et l'activation d'une option de menu à l'aide de la frappe d'une touche définie sont usuelles. Une combinaison de la touche <Amiga> droite et d'une autre touche est assignable à une option de menu afin d'en assurer sa sélection ultérieure.

La notification du code ASCII de la touche à frapper suite à celle de l'option du menu répond précisément de cela. Cette assignation se matérialise ensuite dans le menu à hauteur de l'intitulé de l'option spécifiée, à l'aide du symbole de la touche Amiga suivi du caractère choisi.

Toutefois la réservation préalable de quatre ou plus espaces au sein de l'intitulé de l'option concernée est requise afin que la combinaison de touche représentée n'empiète pas sur le texte de l'option.

MENU Texte_de_menu$() Création de menu déroulant

MENU Array$()

Cette instruction communique à l'interpréteur les positions de mémoire contenant les titres et options du menu. Lors de son emploi, la notification du nom de la matrice unidimensionnelle contenant les textes du menu, est requise. La correspondance entre le nombre des éléments de cette matrice et celui des options de menu à affecter est à respecter scrupuleusement. L'observance des règles suivantes (qui reposent sur la norme ST) est plus ou moins requise lors de la création du menu de tête (le menu Desktop des accessoires) :

1ère chaîne = titre du menu (éventuellement le nom du programme)

2ème chaîne = un quelconque sous-titre (éventuellement une brève description du programme). Cette option peut recouvrir une quelconque application mais puisqu'elle est l'unique option utilisable de ce premier menu, elle sert usuellement à la communication d'une information (par exemple : le Copyright).

3ème chaîne = une rangée de signes moins (ou de traits d'union). Le nombre de tirets détermine la largeur du menu. Puisqu'en ce menu, sont usuellement disposées les accessoires de bureau, vous devriez référer cette largeur à l'intitulé d'accessoire dont la longueur est la plus élevée. Ce trait partage le menu en divers blocs de fonction.

4ème chaîne = à partir de maintenant suivent les options isolées du premier menu, recouvrant les accessoires, échelonnées jusqu'à la dernière option du menu.

9ème chaîne = chaîne vide. Cette chaîne nulle représente la conclusion d'un menu isolé.

Les autres menus dont la création est désirée sont adjoints à la suite de ce premier. Leur construction nécessite l'adjonction des options isolées à la suite de l'intitulé du menu (la chaîne de texte toujours visible dans la barre des menus). L'insertion d'une chaîne nulle ("") doit conclure celle de chaque menu isolé.

Dès que toutes les entrées de menu ont eu lieu, il convient d'y adjoindre en fin une chaîne vide en tant que code de fin d'entrée. L'assignation de signe de cochage à chaque option de menu (Cf. MENU Option_de_menu,Attribut), requiert l'insertion dans l'intitulé de chaque option de menu deux espaces vides.

En outre il vous est possible d'inactiver une option et de la représenter en mode fantôme . Pour ce faire, il suffit de placer un signe moins (ou un tiret: -) en début de la chaîne de texte. Lors de l'enregistrement de la matrice du menu, la commande inactive l'option ainsi signée.

Exemple :

```
DIM array$(40)
REPEAT
  READ array$(i%)
  PRINT array$(i%)
  INC i%
UNTIL array$(i%-1)="XXX"
array$(i%-1)=""
DATA Projet,Option de menu
DATA ---------------
DATA   Charger,  Sauver,  Nouveau,  Couper,  Copier,  Coller,
DATA MENU1,  Option1,  Option2,  Option3,
DATA MENU2,  Option1,  Option2,  Option3,
DATA MENU3,  Option1,  Option2,  Option3,
DATA MENU4,  Option1,  Option2,  Option3,
DATA XXX
MENU array$()
REPEAT
UNTIL MOUSEK=3
```

MENU KILL — Efface la barre des menus

MENU KILL

MENU KILL désactive les menus déroulants disponibles. A vrai dire, cette instruction efface la barre des menus dans la fenêtre et interdit ainsi la sélection de la moindre option. Toutefois cette instruction n'affecte en rien le tableau contenant les données constitutives des menus. L'activation de la barre des menus est possible à tout moment à l'aide de MENU Texte_de_menu.

Chapitre 16

Contrôle d'événements BASIC

MENU Création d'un tampon de contrôle d'événements

Var=MENU(Index)

La présente fonction recouvre une matrice (un tampon de contrôle d'événements) de nombres entiers, où sont adressées en permanence les données des événements se manifestant sur Intuition.

Index recense les indices des éléments de matrice disponibles. Cette fonction permet l'interrogation des éléments indexés de 0 à 10 (MENU(0) à MENU(10)), contenant les données nécessaires à la gestion de événements. La liste suivante détaille l'utilité propre de chacun de ces éléments indexés et les types d'informations qu'ils délivrent.

Les contenus de MENU(Index) :

MENU(0) Menu
Numéro du menu sélectionné dans la barre des menus, interne à la matrice des chaînes de menus définie lors de la création de l'ensemble des menus.

MENU(1) Class
Contient les drapeaux (flags) IDCMP, déterminant ce type d'information. Les trois éléments suivant sont à évaluer en regard du type du drapeau.

MENU(2) Code
Cette valeur est étroitement dépendante du précédent élément (Class). Elle conserve par exemple suite à la sélection d'une option de menu, le numéro correspondant.

MENU(3) Qualifier
Cette variable qui est d'une importance primordiale pour l'interrogation des événements survenus sur le clavier, note les Qualifier: les touches spécifiques, comme Shift, Ctrl, etc.

MENU(4) IAdress
La valeur déposée en ce lieu est l'adresse de mémoire de l'objet d'Intuition : un gadget ou une fenêtre, venant d'être manipulé.

MENU(5) MouseX
Peu importe la nature de l'information réceptionnée, la position relative X du pointeur de la souris est mémorisée en ce lieu. L'évaluation de la position de la souris ne requiert que l'interrogation de cet élément de tableau.

MENU(6) MouseY
Corrélativement à MENU(5), est disposée ici la position Y du pointeur de la souris. Elle est calculée relative au point d'origine (le coin haut gauche) de la fenêtre. Le point d'origine de la fenêtre étant différent de celui de l'écran, les coordonnées de la souris peuvent être négatives.

MENU(7) Seconds
Intuition sauvegarde non seulement les positions de la souris à l'instant précis du surgissement de l'événement, mais en toute simultanéité l'heure système précise afin de connaissance de l'instant précis de ce surgissement.

MENU(9) Micros
A l'inverse de MENU(7), le décompte du temps y est opérée forme de micro-secondes non pas en secondes.

MENU(10) WindowAdress
En ce lieu, se trouve toujours le pointeur de la fenêtre où l'événement s'est manifesté.

L'interrogation de la manifestation d'un événement dans un menu :

Une telle interrogation s'ordonne selon la procédure suivante, avec l'aide complémentaires d'autres fonctions GFA Basic, pour de plus amples précisions sur ces dernières, veuillez consulter leur descriptions disposées dans le présent ouvrage.

Dans le programme principal est défini, suite à la création des menus, un sous-programme d'évaluation des événements. Ce sous-programme s'initialise comme suit :

```
ON MENU GOSUB Evaluation
```

La boucle du programme principal peut d'ailleurs s'énoncer de telle manière :

```
REPEAT
  SLEEP
UNTIL Loop=FALSE
```

Cette boucle assure l'attente d' une information jusqu'à ce que la variable globale (ici Loop) soit spécifiée fausse (FALSE) à partir d'une quelconque sous-routine. La détection et la gestion d'un événement de menu sont possibles en n'importe quel lieu d'un programme.

La sous-routine peut d'elle-même, suite à l'établissement du numéro de l'option de menu sélectionnée à l'aide de la fonction MENU(), brancher la routine correspondante afin de l'exécution de la tâche ou de la procédure désirée.

```
PROCEDURE Evaluation
  Menu%=MENU(0) ! retourne le numéro de l'option sélectionnée
  MENU KILL     ! interdit toute autre sélection consécutive
  IF Menu%=1 THEN GOSUB Sauvegarde
  IF Menu%=2 THEN GOSUB Chargement
```

La fonction MENU() assurent bien sûr d'autres et très diverses détections d'événement. Tous les drapeaux IDCMP de la fenêtre sont utilisables à cette fin. Ces drapeaux sont spécifiés lors de la définition d'une nouvelle fenêtre. En ce faisant, vous communiquez à l'interface Intuition les détections d'événements à assurer. Les nombres retournés par MENU(1) correspondent exactement à ceux dont la notification est requise par la spécification des drapeaux lors de la définition de la fenêtre.

A côté des simples détections opérées à l'aide de MENU(0) qui retourne une information indexée, il existe naturellement des détections propres à l'interface Intuition assurant le retour d'informations bien plus détaillées. Toutefois de telles procédures requièrent des spécifications bien plus précises.

Informations sur les menus :

MENU(1) = 256 Une option (à vrai dire un menu) a été sélectionnée dans la barre des menus.

MENU(2) retourne le numéro exact de l'option ayant été sélectionnée sous forme de code binaire. Cette fonction vous délivre très précisément les numéros du menu, de son sous-menu, de l'option de cette dernière venant d'être sélectionnés dans la fenêtre. Le déchiffrement de ce code binaire s'ordonne comme suit : Numéro du menu = MENU(2) AND 31, Numéro du sous-menu = SHR(MENU(2), 5) AND 63, Numéro de l'option = SHR(MENU(2), 11) AND 31.

Informations sur les fenêtres :

MENU(1) = 1 Le gadget Size de la fenêtre a été activé.

MENU(10) contient suite à cela l'adresse (Intuition) de la fenêtre.

MENU(1) = 2 La taille de la fenêtre a été modifiée.

MENU(10) consigne suite à cela l'adresse (Intuition) de la fenêtre (WINDOW(Adresse) vous permet d'obtenir le numéro GFA Basic sous lequel la fenêtre est gérée).

MENU(1) = 4 Ordonne le rafraîchissement du contenu graphique de la fenêtre (les textes valent aussi comme graphiques), de précédentes superpositions gênant sa netteté. L'adresse de la fenêtre est consignée en MENU(10).

MENU(1) = 512 Le gadget Close de la fenêtre a été activé. L'adresse de la fenêtre est consignée en MENU(10).

MENU(1) = 8192 La fenêtre n'est pas en mesure d'activer la barre des menus cliquée à l'aide du bouton droit de la souris. L'adresse de la fenêtre est consignée en MENU(10).

MENU(1)=262144 Une fenêtre a été activée par le cliquement de la souris. L'adresse de la fenêtre est consignée en MENU(10) et est convertible en numéro de fenêtre GFA à l'aide de WINDOW(Adresse).

Les informations sur la souris :

MENU(1)=8 Un bouton de la souris a été pressé (lors de l'inactivation du contrôle des événements menu, seule la détection de la pression du bouton droit ; la manipulation du bouton gauche y équivaut à celle du droit.

MENU(2) consigne l'état des boutons de la souris et assure ainsi la détection précise du bouton venant d'être pressé ou relâchée. La valeur 104 y représente la pression du bouton gauche, celle 232 celle du droit. Une unité (1) y est additionné pour signifier le relâchement du bouton.

MENU(1)=16 La souris a été bougée.En MENU(5) et MENU(6) sont contenues les cordonnées relatives au point d'origine de la fenêtre.

MENU(1)=1048576 La souris a été bougée. En MENU(5) et MENU(6) sont contenues les modifications des coordonnées relatives à la dernière position au sein de la fenêtre.

Les informations sur les gadgets :

MENU(1)=32 Un gadget a été activé.

En MENU(4) retourne l'adresse mémoire du début de la structure du gadget. MENU(10) retourne l'adresse de fenêtre où le gadget est placé.

MENU(1)=64 Un gadget a été relâché. L'adresse de la structure du gadget est retournée de même que précédemment par MENU(4) et celle de la structure de la fenêtre en MENU(10).

Les informations sur la boîte de requête :

MENU(1)=128 A l'intérieur d'une fenêtre, une boîte de requête a été ouverte.

MENU(4) contient l'adresse de la structure de la boîte de requête et MENU(10) celle de la fenêtre concernée.

MENU(1)=2048 La représentation de la boîte de requête est impossible à l'intérieur de la fenêtre.

MENU(10) renvoie l'adresse de la fenêtre.

MENU(1)=4096 La dernière boîte de requête ouverte vient d'être fermée (la fenêtre est de nouveau disponible pour à des fins de dialogue, d'entrées et de sorties).

MENU(10) contient l'adresse de la fenêtre.

Informations sur le clavier :

MENU(1)=1024 Une touche a été pressée.

MENU(2) retourne le code ASCII de la touche frappée. N'oubliez pas d'y adjoindre leQualifier déposé dans MENU(3), afin d'obtenir la séquence de touches complète venant d'être utilisée.

MENU(1)=2097152 Une touche a été pressée.

MENU(2) est entrepose le code ASCII de la touche pressée. La conversion en termes de codes ASCII est automatique.

D'autres informations :

MENU(1)=16384 Les données du système ont été modifiées par un quelconque autre programme (les données de Preferences sont renouvelées).

MENU(1)=32768 Une nouvelle disquette a été insérée dans un lecteur quelconque.

MENU(1)=65536 Une disquette a été ôtée d'un quelconque lecteur.

MENU(1)=131072 Le Wokbench envoie cette information suite à des modifications le concernant en propre.

MENU(1)=4194303 Suite à le demande d'informations horaires, chaque IntuiTick provoque un message, ceci tous les dixièmes de secondes !

ON MENU — Branchement à des fins de pour détermination d'événement

ON MENU
ON MENU [Temps]

Cette instruction attend la délivrance d'une information et suite à son obtention branche le bloc d'instructions préalablement défini. Elle doit être exécutée au sein d'une boucle, car seule une détection permanente assure une recension complète des événements. En son absence, le branchement des procédures ON MENU...GOSUB est impossible.

L'insertion du paramètre Temps est optionnelle. Cette valeur spécifie le délai maximal d'attente en millième de secondes observé par l'instruction ON MENU (Cf. EVNT_MULTI) avant que le contrôle ne soit de nouveau rendu au BASIC. Des horaires suffisantes permettent l'enregistrement du relâchement du bouton de la souris par l'Input Device : le périphérique des entrées.

N'oubliez surtout pas d'insérer cette instruction à l'intérieur des boucles d'attente internes (REPEAT...UNTIL MOUSEK) car le programme se trouvant dans une boucle d'attente, nul événement ne peut être traité.

SLEEP — Attente d'une information

SLEEP

L'instruction SLEEP est une simplification de celle ON MENU. Son délai d'attente des informations retournées sur le canal des informations propre à l'interface Intuition, est d'une milliseconde.

ON MENU GOSUB — Détermine une procédure (événement de menu)

ON MENU GOSUB Nom_de_procédure

Elle branche la procédure spécifiée à l'aide son nom suite au cliquement d'une option d'un menu déroulant. La gestion de l'option sélectionnée s'opère au sein de cette procédure à l'aide de la fonction MENU(0).

Cette procédure étant inexistante, la détection des événements de menu est suspendue.

ON MENU BUTTON GOSUB — Appel d'une procédure (événement souris)

ON MENU BUTTON GOSUB Nom_de_procédure

Appelle la procédure spécifiée à l'aide de son nom, suite à une ou plusieurs pressions d'un ou plusieurs boutons de la souris ou de leur non pression.

A l'intérieur de la routine, il convient d'évaluer très précisément à l'aide des fonctions MENU() la ou les boutons pressés et le nombre de pressions opérées.

ON MENU KEY GOSUB — Appel d'une procédure (événement clavier)

ON MENU KEY GOSUB Nom_de_procédure

Opère la détection d'un événement à partir du clavier et appelle suite à la frappe d'une touche la procédure spécifiée, où la délivrance du code de la touche frappée s'obtient à l'aide de MENU(14).

L'inexistence du nom de la procédure amène l'interruption de la détection des événements survenus sur le clavier.

Chapitre 17

Programmation avancée et nouvelles commandes

Ce chapitre est basé sur les toutes dernières évolutions du GFA Basic, tant du point de vue de la programmation que du langage lui-même. En effet, si les détracteurs du GFA Basic ont affirmé que ce langage n'était pas très adapté aux performances de l'Amiga, ceux qui se sont investis dans son utilisation ont éprouvé le contraire et ont même émis des suggestions allant vers une amélioration sensible de ce langage.

Voici donc "en vrac" dans ce chapitre, les nouveautés et les dernières astuces de programmation en GFA Basic. Nous allons aussi vous présenter la programmation du Copper et de l'interface série, et évoquer la librairie IFF qui acompagnera sans doute le GF ABasic 3.04.

Des nouvelles commandes ont été implémentées depuis la rédaction de ce livre. Ces commandes ne sont pas traitées dans le chapitre les concernant, vous noterez d'ailleurs qu'elles sont documentées dans le fichier LISEZMOI.GFA de chaque version. Voici donc les nouveautés de la version 3.04 (par rapport à la version 3.01) :

17.1 Les nouvelles entrées de menu

Un menu supplémentaire a été implémenté contenant des opérations de Couper/Copier/Coller. Cela répond à la demande de nombreux utilisateurs désireux de faire de telles opérations entre deux éditeurs GFA Basic tournant simultanément. Le clipboard n'est pas encore supporté au moment ou nous écrivons, mais cela ne saurait tarder. Pour le moment, voici les nouvelles options de ce menu :

Cut
Permet de copier un bloc vers le fichier RAM:GFA.$$$ et détruit le bloc. Le raccourci <Amiga> U est utilisable.

Copy
Copie un bloc vers le fichier RAM:GFA.$$$, on peut aussi le sélectionner avec <Amiga> O.

Paste
Effectue l'opération Merge du fichier RAM:GFA.$$$. Cette opération est sélectionnable avec <Amiga> P.

Blk Start
Marque le début d'un bloc, son raccourci clavier est <Amiga> B.

Blk End
Marque la fin d'un bloc, le raccourci est <Amiga K>

Encore une fois, notez bien que ces opérations risquent d'être remplacées par des opérations sur le Clipboard, elles améliorent toutefois la convivialité de l'éditeur.

17.2 Le port série.

Les commandes de gestion du port série ont été considérablement enrichies. La syntaxe de la commande OPEN reconnait maintenant COM1: comme en Amiga BASIC, et une nouvelle variable nommée _IOExtSer est reconnue par l'interpréteur qui pointe vers le tampon d'Entrées/Sorties du port série. Vous avez donc maintenant un accès étendu au port série :

```
OPEN mode$,#canal,"COM1:[Vitesse[,Parité[,Longueur[,StopBits]]]]
```

La Vitesse est celle du transfert des données sur le Port Série. Par exemple avec un Minitel standard vous spécifierez 1200. Notez que si le hardware de l'Amiga accepte des vitesses de 1,000,000 bauds, elles ne sont pas reconnues par les puces. La vitesse du MIDI (31250 bauds) devrait vous suffire pour ce qui est de la limite supérieure.

Lors d'un transfert sur 7 bits ou moins, un bit supplémentaire peut être utilisé pour le contrôle de parité. C'est à dire que si la parité est paire, le huitième bit sera placé de manière à ce qu'il y ait un nombre pair de bits à 1 parmi les 8. On spécifie une parité paire avec la lettre "E" (pour

Even). La parité impaire correspond à un nombre impair de bits à 1, sa notation est "O" (pour Odd). Dans le cas d'un transfert sans contrôle de parité, spécifiez "N".

La Longueur permet de spécifier le nombre de bits à envoyer à chaque fois (de 5 à 8).

StopBits indique le nombre de bits envoyés à la fin de chaque paquet. Vous pouvez indiquer 1 ou 2.

Pour ceux d'entre vous qui possèdent un Minitel et le cable de raccord à leur Amiga, ils peuvent utiliser cette syntaxe pour l'utiliser comme un terminal. Le mode n'est d'aucune importance puisque le port série est un canal interactif. Voici un exemple :

```
OPEN "O",#1,"COM1:1200,N,7,1"  ! Ouverture du port série
PRINT #1,CHR$(12)              ! Caractère d'effacement
PRINT #1,"Hello, World !";     ! On s'en serait douté ...
PRINT #1,CHR$(27);CHR$(72);    ! Clignotement
PRINT #1,"Hello, World !";     ! Encore une fois
PRINT #1,CHR$(27);CHR$(73);    ! Arrêt du clignotement
CLOSE #1                       ! Fermeture du port série
```

_IOExtSer est un pointeur sur le bloc d'entrées-sorties du serial.device:

```
_IOExtSer +  0 : MsgNode
                  0 : Succ        Prochaine Entrée
                  4 : Pred        Entrée Précédente
                  8 : Type        Type de l'Entrée
                  9 : Pri         Priorité
                 10 : Name        Nom
                 14 : ReplyPort   Adresse du ReplyPort
                 18 : MNLength    Taille du Node

_IOExtSer + 20 : IOExt
                 20 : IO_DEVICE   Adresse du Device Node
                 24 : IO_UNIT     Numéro interne de l'unité
                 28 : IO_COMMAND  Commande
                 30 : IO_FLAGS    Drapeaux
                 31 : IO_ERROR    Statut de l'erreur

_IOExtSer + 32 : IOStdExt
                 32 : IO_ACTUAL   Nombre d'octets transférés
                 36 : IO_LENGTH   Nombre d'octets en attente
                 40 : IO_DATA     Adresse du tampon de données
                 44 : IO_OFFSET   Offset

_IOExtSer + 48 : Extension pour le serial.device
                 48 : CTLCHAR     Caractère de contrôle
```

```
52 : RBUFLEN     Longueur du tampon de saisie
56 : WBUFLEN     Longueur du tampon de sortie
60 : BAUD        Vitesse en bauds
64 : BRKTIME     Temps du break en microsecondes
68 : TERMARRAY   Caractères d'interruption
76 : READLEN     Longueur du paquet en lecture
77 : WRITELEN    Longueur du paquet en écriture
78 : STOPBITS    Nombre de stopbits
79 : SERFLAGS    Drapeau du port série
80 : STATUS      Mot d'état
```

Vous obtiendrez plus d'informations au sujet du serial.device et sa programmation en consultant La Bible de l'Amiga (Tomes I et II), parue aux Editions Micro Application.

_IOExtSer peut être utilisée soit en lecture, soit en écriture pour une gestion personnalisée du port-série (par exemple modification de la vitesse de transmission sans refermer le canal).

Les fonctions qui gèrent les fichiers sont disponibles avec COM1: : (LOF, LOC, INPUT$, etc.).

17.3 Nouvelles fonctions graphiques

SGET tableau$() [SG] — Copie l'écran dans un tableau

Cette commande recopie l'écran actif vers un tableau de chaînes de caractères. Chaque élément du tableau représente un plan couleur (BitPlane). Vous pouvez ainsi effectuer des opérations logiques sur l'image avant de la recopier vers un autre écran. Voici un exemple simple :

```
DIM a$(1)                                  ! Déclaration du tableau
SGET a$()                                  ! Capture de l'écran
OPENS 1,0,0,640,256,2,32768                ! Ouverture écran 1
SPUT a$()                                  ! Recopie d'écran
SETCOLOR 3,&HFF                            ! Changer couleur
a$(1)=STRING$(LEN(a$(1)),CHR$(255))        ! Modifier Plan 1
PAUSE 50
SPUT a$()                                  ! Nouvelle Recopie
PAUSE 50
CLOSES 1                                   ! Fermer écran
```

SPUT tableau$() {SPU} Copie un tableau dans un écran

Cette commande est l'inverse de SGET(). Vous devez faire attention à ne pas déborder sur l'écran destination auquel cas vous risquez d'avoir pour le moins un résultat étrange et au pire un blocage du système. Un exemple amusant de ce qui peut vous arriver est l'ouverture de l'écran en basse résolution :

```
OPENS 1,0,0,640,256,2,0
```

Ceci va vous produire un effet de loupe (sans plantage heureusement). Par contre vous perdrez la moitié droite de l'écran.

SCROLL{SC}dx,dy,x0,y0,x1,y1 Powerfull Amiga Scrollings !

Je ne vous ferais pas l'affront de vous expliquer ce qu'est le scrolling sur Amiga. La vitesse de réaffichage dépend uniquement du nombre de plans et de la taille du rectangle à déplacer.

Si toutefois vous n'auriez pas la moindre idée de la puissance de cette instruction, commencez par modifier l'exemple ci dessus avec l'écran en basse résolution en insérant en dessous de l'instruction SPUT a$() l'instruction :

```
SCROLL -320,0,0,0,640,256
```

qui vous fera récupérer la moitié droite de l'écran que vous aviez perdue. Voici un programme complet :

```
COLOR 2,3
PBOX 10,5,510,200
TEXT 20,20,"Le Scroll, ça colle !"
FOR i=0 TO 150
  SCROLL 1,1,15+i,10+i,200+i,30+i
NEXT i
FOR i=0 TO 150
  SCROLL 1,-1,165+i,160-i,350+i,180-i
NEXT i
```

Vous aurez certainement l'occasion de remarquer que tout dépassement des limites de l'écran sera sanctionné par un feu de joie (Le gourou seulement si vous avez de la chance !) où parfois le processeur sonore commencera une belle mélodie et de jolies couleurs apparaitront aléatoirement à l'écran. Les américains ont appelé cela un "Burn and Flames system crash".

17.4 Environnement du GFA Basic 3.04

A l'heure où nous écrivons ces lignes, un véritable environnement autour du GFABasic pointe le bout de son nez. Notamment, l'instruction FILESELECT garde ses spécifités mais a bénéficié des critiques formulées par les journalistes, les utilisateurs et Commodore International Limited. Tout ceci dans le but de la rendre plus conviviale, plus esthétique et plus proche des standards de programmation du système.

Un point qui avait été grandement critiqué était l'absence d'instructions permettant de gérer le format IFF. Il existe un moyen simple de pallier à cette difficulté en utilisant une iff.library. Plusieurs équipes de programmeurs travaillent actuellement sur ce sujet dans le but d'aboutir à une librairie standard pour le système. Une iff.library a été placée dans le domaine public, certes plus limitée que d'autres projets en cours mais offrant l'énorme avantage d'être disponible immédiatement. Il serait possible que cette bibliothèque de fonctions (uniquement pour le graphisme) soit distribuée avec les prochaines versions du GFABasic. Cette bibliothèque étant déjà disponible en France dans le domaine publique, vous pouvez déjà vous la procurer par vos propres moyens et l'utiliser mais au cas où une autre bibliothèque serait choisie par Commodore pour l'implémentation dans une version future du Workbench (1.4 ou plus), des problèmes de compatibilité ne sont pas à exclure.

17.5 Finissons en beauté !

Le GFABasic étant un langage vraiment nouveau tous les trucs et astuces nécessaires à sa programmation n'ont pas encore été découverts! Voici toutefois la démonstration de sa puissance et de son efficacité à contrôler directement le Hardware de l'Amiga. La chose de

ardue devient tout simplement intuitive ! Tout cela grâce à l'instruction ABSOLUTE qui permet de faire des merveilles de programmation sur ce plan. Le gros du travail consiste maintenant à déclarer chaque registre du système dont on a besoin avec une instruction ABSOLUTE, puis de les utiliser comme de vulgaires variables. C'est d'une lisibilité inégalée par les autres langages, surtout quand on sait que toutes les fonctions de haut niveau restent à portée de la main au cas où l'on en aurait besoin. Voici donc en exclusivité mondiale le premier programme de manipulation du Copper en GFA-Basic. Nous aurions pu utiliser des POKES mais bizarrement vous n'en trouverez aucun dans ce source. De plus, nous pensons sincèrement que vous comprendrez plus vite ce qu'il est sensé faire que si vous aviez eu le même en assembleur :

```
' Programmation du Hardware en GFABasic
'
OPENW #0
PRINT AT(15,10);"Programmation directe du Copper - Test du CIA-A"
dmabase%=&HDFF000
' Registres spécialisés
ABSOLUTE dmacon&,dmabase%+&H96 ! Utilisé pour le Copper
' Registres Copper
ABSOLUTE cop1lc%,dmabase%+&H80 ! Première liste Copper
ABSOLUTE copjmp1&,dmabase%+&H88
'
ABSOLUTE ciaapra|,&HBFE001     ! Utilisé pour le bouton gauche de la souris
'
startlist%=38                  ! Adresse de la liste système
'
cladr%=MALLOC(40000,2)         ! Allocation Mémoire
'
coul&=0                        ! Initialisation des variables
rcoul&=&H180
adr&=&H1FF
adr2&=-2
i%=0
'
REPEAT
  WORD{ cladr%+ind%}=rcoul&        ! WORD offre plus de sécurité que DPOKE
  WORD{ cladr%+ind%+2}=coul&       ! dans les débordements de variables.
  WORD{ cladr%+ind%+4}=CARD(adr&)
  WORD{ cladr%+ind%+6}=adr2&
  INC i%
  ADD ind%,8
  coul&=OR(i%,&HF)
  ADD adr&,&H10
  PRINT AT(20,15);"La liste comporte : ";i%*2;" instructions"
UNTIL adr&=WORD(&HFFEF)           ! Arrêt à la ligne 255
WORD{ cladr%+ind%}=-1             ! Fin de la liste du Copper
WORD{ cladr%+ind%+2}=-2
dmacon&=&H3A0                     ! Ecriture dans DMACON (Arrêt DMA)
```

```
cop1lc%=cladr%                          ! Adresse de la liste dans COP1LC
copjmp1&=0                              ! Activation (registre strobe)
'
dmacon&=WORD(&H8280)                    ! Activation DMA-Copper
'
REPEAT                                  ! Attente Bouton gauche de la souris
UNTIL NOT BTST(ciaapra|,6)              ! Test du CIAAPRA (MOUSEK=1 aurait marché !)
cop1lc%=LONG{startlist%+_GfxBase}       ! Adresse de la liste système dans COP1LC
copjmp1&=0
dmacon&=WORD(&H83E0)                    ! Retour à l'état initial
MFREE(cladr%,40000)                    ! Libérer la Mémoire occupée
END
```

Voilà, c'était notre petit cadeau pour la fin de ce livre. A vous maintenant de manipuler vos propres listes du Copper en n'oubliant pas de spécifier le bon type de variable pour les registres du hardware car sinon celui-ci ne prendrait même pas la peine d'en informer le Gourou. La règle générale veut que ces registres soient sur deux octets (WORD) mais méfiez-vous ...

Annexes

Annexe A : Table des codes ASCII

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

128		160		192	À	224	à
129	⟨F1⟩	161	¡	193	Á	225	á
130	⟨F2⟩	162	¢	194	Â	226	â
131	⟨F3⟩	163	£	195	Ã	227	ã
132	⟨F4⟩	164	¤	196	Ä	228	ä
133	⟨F5⟩	165	¥	197	Å	229	å
134	⟨F6⟩	166	¦	198	Æ	230	æ
135	⟨F7⟩	167	§	199	Ç	231	ç
136	⟨F8⟩	168	¨	200	È	232	è
137	⟨F9⟩	169	©	201	É	233	é
138	⟨F10⟩	170	ª	202	Ê	234	ê
139	⟨HELP⟩	171	«	203	Ë	235	ë
140		172	¬	204	Ì	236	ì
141		173	-	205	Í	237	í
142		174	®	206	Î	238	î
143		175	¯	207	Ï	239	ï
144		176	°	208	Ð	240	ð
145		177	±	209	Ñ	241	ñ
146		178	²	210	Ò	242	ò
147		179	³	211	Ó	243	ó
148		180	´	212	Ô	244	ô
149		181	µ	213	Õ	245	õ
150		182	¶	214	Ö	246	ö
151		183	·	215	×	247	÷
152		184	¸	216	Ø	248	ø
153		185	¹	217	Ù	249	ù
154		186	º	218	Ú	250	ú
155		187	»	219	Û	251	û
156		188	¼	220	Ü	252	ü
157		189	½	221	Ý	253	ý
158		190	¾	222	Þ	254	þ
159		191	¿	223	ß	255	ÿ

Annexe B :

Message d'erreurs du GFA Basic

0	Division par zéro
1	Débordement
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 ! -32768 .. 32767
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 ! non autorisée
10	Chaîne trop longue ! max. 32767 caractères
11	Le programme n'est pas ! en GFA BASIC 3.03
12	Programme trop long ! Mémoire pleine ! NEW
13	Le programme n'est pas ! en GFA BASIC 3.03 ! EOF - NEW
14	Tableau dimensionné deux fois
15	Tableau non dimensionné
16	Index de tableau 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 ! maximum 128
28	Le tableau ne peut avoir qu'une dimension
29	Nombre points plus grand que le champ
30	Merge - Ce n'est pas un fichier ASCII
31	Merge - Ligne trop longue - Arrêt
32	==> Syntaxe incorrecte ! Arrêt du programme

33 Marque non définie
34 Trop peu de données
35 Donnée non numérique
36 Erreur inconnue 036
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 Procedure
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. 31)
50 Pas de fichier "R"
51 Erreur inconnue 051
52 Champ plus grand que l'enregistrement
53 Erreur inconnue 053
54 Mauvaise longueur ! d'enregistrement GET/PUT
55 Mauvais numéro ! de phrase GET/PUT
56 Erreur inconnue 056
57 Erreur inconnue 057
58 Erreur inconnue 058
59 Erreur inconnue 059
60 Longueur de chaîne ! de Sprite erronée
61 RESERVE erreur
62 Erreur dans Menu
63 Erreur dans Reserve
64 Erreur dans pointeur
65 Champ < 256
66 Pas de tableau VAR
67 Erreur ASIN/ACOS
68 Erreur de type : VAR
69 ENDFUNC sans RETURN
70 Erreur inconnue 070
71 Index/Paramètre trop grand
72 IPeut pas ouvrir fenêtre/Écran
73 Erreur inconnue 073

74	Pas de fenêtre
75	Erreur Library
76	Erreur inconnue 076
77	Erreur inconnue 077
78	Erreur inconnue 078
79	Erreur inconnue 079
80	Erreur inconnue 080
81	Erreur inconnue 081
82	Erreur inconnue 082
83	Erreur inconnue 083
84	Erreur Objets
85	Erreur Imprimante
86	Erreur inconnue 086
87	Erreur inconnue 087
88	La mémoire ne répond pas
89	Erreur Sprite
90	Erreur dans Local
91	Erreur dans For
92	Resume (next) impossible ! Erreur fatale, For ou Local
93	Erreur pile
94	Erreur inconnue 094
95	Erreur inconnue 095
96	Erreur inconnue 096
97	Erreur inconnue 097
98	Erreur inconnue 098
99	Erreur inconnue 099
100	GFA-BASIC Version 3.03 F ! C Copyright 1986-1988 ! GFA Systemtechnik GmbH
101	Erreur inconnue 101
102	Erreur inconnue 102
103	Mémoire insuffisante
104	Table des tasks pleine
105	Erreur inconnue 105
106	Erreur inconnue 106
107	Erreur inconnue 107
108	Erreur inconnue 108
109	Erreur inconnue 109
110	Erreur inconnue 110
111	Erreur inconnue 111
112	Erreur inconnue 112
113	Erreur inconnue 113
114	Erreur inconnue 114
115	Erreur inconnue 115

116 Erreur inconnue 116
117 Erreur inconnue 117
118 Erreur inconnue 118
119 Erreur inconnue 119
120 Ligne des arguments invalide ou trop longue

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94)
Dépôt légal - Juin 1989
N° d'impression: 5123

Dans la même collection...

Référence	Titre		Prix T.T.C.
...			
...			
ML617A	Applications sous SUPERBASE disquette incluse		349.00
ML197	Bien débuter avec l'AMIGA		149.00
ML553	Bien débuter en C sur AMIGA		149.00
GL123	Guide SOS AMIGADOS/AMIGABASIC		149.00
ML634	Le livre du lecteur de disquette	disquette incluse	299.00
ML512	La Bible de l'AMIGA		299.00
ML612	La Bible de l'AMIGA	disquette incluse	399.00
ML505	Le Livre du Graphisme		249.00
ML605	Le Livre du Graphisme	disquette incluse	349.00
ML504	Le Grand livre de l'AMIGABASIC		249.00
ML604	Le Grand livre de l'AMIGABASIC	disquette incluse	349.00
ML513	Le Livre de l'AMIGADOS		199.00
ML198	Le Livre du Langage Machine		199.00
ML532	Le Livre de la Musique		199.00
ML188	Trucs et Astuces AMIGA		199.00
ML288	Trucs et Astuces AMIGA	disquette incluse	299.00
...			
...			

ACHETEZ LA DISQUETTE

Ce livre vous passionne, mais vous n'avez pas le temps de taper les programmes.

Commandez-nous la disquette des programmes du livre :

LE LIVRE DU GFA BASIC AMIGA

BON DE COMMANDE

A retourner à Micro Application - 58, rue du Faubourg Poissonnière - 75010 PARIS

NOM : .....

PRENOM : .....

ADRESSE : .....

CODE POSTAL : ..... VILLE : .....

Je désire recevoir la disquette du livre : **LE LIVRE DU GFA BASIC AMIGA**

Je joins à ce coupon un ☐ chèque de : 120 FF ☐ CCP de : 120 FF ou ☐ N° de carte bleue .....

DATE : SIGNATURE :

AMIGA

LE LIVRE DU GFA BASIC 3.0

BLEEK/HECHT/LITZKENDORF

Ça y est, vous voilà possesseur du meilleur Basic disponible sur Amiga. Il est grand temps de passer aux choses sérieuses! Pour vous aider à réaliser vos programmes, LE LIVRE DU GFA BASIC 3.0 détaille chaque instruction de ce langage, progressivement, suivant son domaine d'application : les variables, les tableaux, les entrées/sorties, le graphisme et le son...
Afin d'optimiser votre programmation, de choisir l'outil le mieux adapté à vos besoins, nous vous présentons dans cet ouvrage de nombreux exemples d'application en GFA, des conseils précieux et des astuces pour tirer parti des vastes possibilités de ce langage. Réalisez aisément des pop-up menus, créez des motifs de remplissage, animez des dessins en Logo, gérez un fichier à accès direct, maîtrisez parfaitement la gestion des disques ou de la mémoire utilisateur...
LE LIVRE DU GFA BASIC 3.0 : votre meilleur allié pour la programmation en langage structuré.

Principaux sujets traités :

- Introduction : organisation de la mémoire, variables, boucles, flags, matrices et vecteurs...
- Les instructions d'entrées/sorties : gestion de l'affichage, opérations disque, gestion des fichiers, instructions d'impression...
- Textes et nombres : manipulation de chaînes, instructions arithmétiques, opérateurs logiques et binaires, nombres aléatoires...
- Le graphisme : instructions graphiques, chaînes PUT, animation d'objets, dessin en points ou trait...
- Les systèmes numériques, tableaux, tables multi-dimensionnelles...
- La mémoire utilisateur, les manipulations de pointeurs, la gestion des erreurs...
- La bibliothèque EXEC de l'Amiga, la programmation des fenêtres et menus, du Copper...

9 782868 992079

REF. : ML 562 / PRIX : 149 F
ISBN : 2-86899-207-2 / ISSN : 0980-1928

EDITIONS MICRO APPLICATION
58, RUE DU FAUBOURG POISSONNIERE
75010 PARIS TEL (1) 47 70 32 44