ATARI-Software

OMIKRON. BASIC

Series ATARI ST

ATARI®

...la technology de pointe a bas prix.

INTRODUCTION

Vous venez d'acquérir le Basic OMIKRON. Bravo! Pour que vous puissiez bien vous en servir, nous avons conçu ce manuel d'apprentissage et de référence. Nous pensons avoir mis tout ce qui vous sera nécessaire pour débuter, comme pour développer en OMIKRON Basic. Néanmoins, il serait préférable que vous ayez une certaine connaissance du Basic ou d'un autre langage (comme PASCAL ou C).

MISE EN ROUTE

Le basic OMIKRON existe sous deux formes, en cartouche ou sur disquette.

– EN CARTOUCHE –

Sur le côté gauche de votre ATARI ST, vous trouverez le port cartouche. Vous devez insérer la cartouche OMIKRON dans le connecteur.

Comment procéder ?

- Eteignez l'unité centrale et tous les périphériques.
- Regardez bien la cartouche, sur un côté se trouve les composants, de l'autre vous pouvez voir une surface peinte en blanc. **ATTENTION !** Le côté sur lequel sont enfichés les composants doit toujours se trouver vers le sol.
- Allumez le système.
  - **Premier cas**: Votre ATARI démarre, le message '[04] OMIKRON.cartridge selftest ... ok' s'affiche en haut de l'écran. C'est parfait.
  - **Deuxième cas**: Votre ATARI ne démarre pas ou bien le message affiché se termine par 'bad'. Dans ce cas, contactez votre revendeur pour vérifier votre machine et la cartouche.

- Installez un lecteur 'O'. Cliquez sur une icône disque, puis rendez–vous dans le menu Options du bureau. Cliquez l'option 'Installer une unité disque...'. Donnez le nom 'O' au lecteur, puis 'OMIKRON' comme libellé. Cliquez 'Installer'. Ouvrez le lecteur 'O'. Démarrez le seul programme qui se trouve dans ce lecteur. Vous êtes maintenant dans le basic OMIKRON!
- **IMPORTANT**: Si vous ne voulez pas réinstaller à chaque fois le disque 'O', nous vous conseillons de faire ce qui suit:
  - Installez le disque 'O'
  - Sauvegardez le Bureau
  - A chaque démarrage de la machine, utilisez la disquette sur laquelle vous avez sauvé le bureau. L'ordinateur installera de lui–même le disque 'O'. (Sur un disque dur comportant un Boot automatique, vous n'avez pas besoin d'utiliser une disquette particulière, en effet GEM sauvegarde le fichier de configuration du bureau sur le disque C).

– SUR DISQUE –

- Ouvrez le lecteur contenant le programme 'OM–BASIC.PRG' et démarrez–le. Vous êtes maintenant dans le basic OMIKRON!
- **ATTENTION**: Si vous installez l'OMIKRON basic dans un dossier AUTO, celui–ci fonctionnera mais sans la partie AES du GEM. En effet, tout programme situé dans un dossier auto démarre avant que l'AES soit initialisé. Vous ne pourrez donc pas travailler avec la souris, les fenêtres, les menus, etc.

A PROPOS DE CE MANUEL...

Nous avons séparé le manuel en deux volumes. Pour quelles raisons ?

En premier lieu, nous ne pensons pas qu'un livre de 450 pages soit très pratique à manipuler. Secondo, l'existence de deux volumes permet de bien différencier les pôles d'intérêt:

- Un volume de références
- Un volume pour les explications détaillées.

D'autre part, les instructions ont été classées par catégories (par exemple, les instructions ayant un rapport avec les fichiers sont regroupées par ordre alphabétique dans le chapitre 'FICHIERS').

REGLES TYPOGRAPHIQUES:

Tout au long des pages de ce manuel, vous trouverez deux polices de caractères utilisées de la façon suivante:

- Cette police de caractère pour les explications.
- **Cette autre police pour la syntaxe et les exemples.**

- Les commandes et les fonctions sont toujours indiquées en lettres capitales. Exemple: RETURN, PRINT
- Les variables n'ont que la première lettre des mots en capitale. Exemple : Prix, Manuel_De_L_Omikron

Note: Ces deux dernières règles sont appliquées dans le basic lui–même. Quand vous tapez 'print', l'éditeur affiche après validation 'PRINT'.

LES TOUCHES

Les touches du clavier sur lesquelles vous devez appuyer sont indiquées entre crochets [].

Par exemple, la touche ESC, située en haut à gauche, est indiquée par [ESC].

Le clavier de l'ATARI dispose parfois de touches en double exemplaire. C'est le cas de [Shift], [*], [/], [+], [–], [.],[(], [)], [Enter] et des chiffres. Dans le cas général, vous pouvez appuyer sur la touche qui vous plaît, l'OMIKRON n'en tient pas compte. Il existe pourtant des particularités, nous les aborderons avec l'ordre INKEY$.

Les abréviations retenues pour les touches du clavier sont:

- [Control] = [CTRL]
- [ALTERNATE] = [ALT]
- [Shift] = [Sf]

L'éditeur exige parfois des combinaisons de touches. Les combinaisons de touches sont indiquées par:

- [Shift]–[CTRL], où le signe '–' signifie: Appuyez sur l'une des touches [Shift], maintenez cette touche enfoncée, puis appuyez sur [CTRL].
- [F1][F2], où l'absence de caractère de liaison indique qu'il faut successivement taper [F1] puis [F2].

SYNTAXE

A chaque instruction correspond une ou plusieurs syntaxes. Les syntaxes sont imprimées de la manière suivante:

- Les <> indiquent une entrée obligatoire.
- Les [] indiquent une entrée optionnelle.
- Les ... indiquent qu'on peut répeter l'entrée précédente.

COMPATIBILITE AVEC M–BASIC.

L'OMIKRON est l'un des meilleurs basics existant, et ce pour plusieurs raisons. L'une de ces raison est sa compatibilité quasi–parfaite avec le MICROSOFT–BASIC, ce qui vous permet de reprendre directement la plupart des programmes écrits en Basic 'normal'. Il existe pourtant quelques différences qu'il est bon d'avoir en tête lors du portage d'un programme. Dans ce qui suit, nous vous indiquons les fonctions concernées et les moyens de corriger ces différences.

LOG

En M–BASIC, LOG(X) calcul le logarithme népérien de X. En OMIKRON, vous devez taper LN(X).

MKS$, MKD$

Transforme une expression numérique en une chaîne alphanumérique. Le format de représentation des nombres utilisé par l'OMIKRON est sensiblement différent, car plus précis. Cette précision demande plus de mémoire. Aussi, les deux fonctions retourneront des chaînes de caractères plus longues. Le problème n'est vraiment gênant qu'avec FIELD et dans ce cas, nous vous conseillons d'augmenter un peu la taille des champs.

DATA

En MBASIC, 'DATA Maison' est équivalent à 'DATA "Maison" '. Pour l'OMIKRON, il s'agit d'une variable numérique. Vous obtiendrez l'erreur 'CONFUSION DE TYPE' si vous ne rajoutez pas les guillemets.

RANDOMIZE

N'existe pas en OMIKRON. Effacez cette commande.

RND(–X)

Ne donne pas la valeur initiale du générateur aléatoire, mais une valeur comprise entre 0 et –1.

WIDTH

N'existe pas en OMIKRON.

ERASE

Cette instruction efface un tableau. En OMIKRON, vous redimensionnez avec DIM.

CHAIN...ALL

N'existe pas. Il faut taper toutes les variables à transmettre.

PRINT..USING

Quelques différences. Cf PRINT

CLOSE

En OMIKRON, CLOSE efface les buffers d'entrées-sorties.

Le GW-Basic est la version du basic la plus utilisée dans le monde. Elle fonctionne sur les compatibles PC. Comme le GW est très proche du M-Basic, il y a peu de complications supplémentaires.

Mais attention: les PC sont tout de même différents des ATARI ST (euphémisme!).

Par exemple: les PC disposent de la sur-brillance, du clignotement et d'un jeu de caractères semi-graphiques. En monochrome, l'ATARI ST ne peut pas simuler la sur-brillance. Donc, l'OMIKRON non plus. Pour le jeu de caractères, il est possible de faire un appel aux fonctions VDI de chargement de police. Pour le clignotement, il n'y a pas de solution générale.

Les éditeurs de l'OMIKRON

Les éditeurs ? Eh oui ! L'OMIKRON dispose de deux éditeurs !

Le premier éditeur est un éditeur pleine page qui ressemble assez à celui du Commodore 64. Il est simple et efficace pour les petits programmes, pour calculer rapidement, pour mettre au point les programmes. Sa vocation n'est toutefois pas d'écrire un programme de 1000 lignes, bien que cela soit tout à fait possible.

Le deuxième éditeur est de la dernière génération. Il possède des menus déroulants et utilise au maximum la souris. Extrêmement simple à utiliser, il convient parfaitement aux débutants. Sa puissance incroyable et sa vitesse époustouflante vont le faire adorer des professionnels.

Bien que l'OMIKRON dispose de deux éditeurs, il ne contient qu'une mémoire, ce qui signifie qu'on peut passer d'un éditeur à l'autre sans rien perdre du programme que l'on vient de taper.

Nous vous proposons d'aborder ces deux éditeurs sous le nom d'éditeur standard pour le premier et de super–éditeur pour le second.

L'éditeur standard.

Dès le chargement du basic OMIKRON, vous vous trouvez dans cet éditeur.

Comme nous l'avons déjà dit plus haut, cet éditeur est plutôt conçu pour la mise au point des programmes. Sa vitesse n'a pas été le point déterminant lors de son écriture. Venons en au fonctionnement.

Positionnement du curseur

Le curseur se déplace à l'aide des quatre flèches du clavier. [CTRL]–[Clr/Home] efface l'écran. [Clr/Home] positionne le curseur en haut de l'écran.

Entrée d'une ligne de programme.

Il faut bien distinguer le mode direct du mode programme: Le mode direct (pas de numéro de ligne) oblige l'OMIKRON à interpréter immédiatement la ligne après l'appui sur [Return]. C'est un mode qui permet de tester une instruction ou d'effectuer un calcul.

Exemple : **PRINT 990/1.186**

Il est possible de repositionner le curseur sur la ligne de commande, éventuellement de la modifier et de réexécuter celle–ci simplement en appuyant sur [Return]. Il est à noter que le curseur dans ce cas ne doit pas être obligatoirement en fin de ligne.

Le mode programme (un numéro pour chaque ligne) permet de bâtir un programme. On rentre les lignes en séquence, puis on exécute le programme par la commande 'RUN'. Exemple:

```
10 INPUT "Entrez A: ";A
20 PRINT "Le carré de A est: "; A^2
RUN
```

Listage d'un programme.

Pour ce faire, tapez 'LIST'. Le programme s'affiche à l'écran dans l'ordre croissant des numéros de lignes.

Si une ligne de texte dépasse en nombre de caractères la capacité de l'écran, le caractère '|' apparaît au début de la ligne suivante suivi du reste du texte.

Pour interrompre le déroulement, tapez [CTRL]–[C]. Pour suspendre le déroulement, appuyez sur [CTRL]–[S]. Pour reprendre le déroulement suspendus par [CTRL]–[S], appuyez sur [CTRL]–[Q]. Après un [CTRL]–[C], on peut poursuivre le déroulement des lignes du programme à l'aide des flèches du clavier.

Modification du texte sur l'écran.

Il existe deux modes pour modifier un texte:

- Le mode insertion (celui–ci est activé par défaut).
- Le mode recouvrement.

Vous pouvez passer de l'un à l'autre par la combinaison [CTRL]–[Insert].

Le mode recouvrement

Lorsque vous écrivez sur une ligne de texte, les caractères tapés viennent remplacer les anciens.

[Delete]: Le texte à partir du curseur suit celui–ci d'un cran vers la gauche et efface de ce fait le caractère qui se situait à la gauche du curseur.

[Backspace]: Déplace le curseur vers la gauche en remplaçant les caractères par des espaces.

[Insert]: Décale le texte à partir du curseur d'un cran vers la droite et place un espace.

Le mode insertion

Lorsque vous tapez un texte, celui–ci vient s'insérer dans le texte déjà existant.

[Delete]: Efface le caractère qui se trouve sous le curseur et déplace le texte à droite de celui–ci d'un cran vers la gauche.

[Backspace]: Même effet que [Delete] en mode recouvrement.

[Insert]: Son effet reste le même.

Sauvegarde d'un programme sous l'éditeur standard.

L'OMIKRON peut sauver un programme de deux manières: en code FIT ou en code ASCII.

SAVE "NOM"

Sauvegarde en code FIT. Le code FIT (Fast Interpreting Technic) est le résultat du fonctionnement semi–compilé de l'OMIKRON. Il s'agit d'un code directement exécutable par l'OMIKRON, directement compilable avec le compilateur. Quand vous tapez la commande SAVE "NOM", le code FIT est employé. Tapez "SAVE" tout court oblige l'OMIKRON à sauvegarder sous le dernier nom employé.

SAVE "NOM",A

Sauvegarde en code ASCII. Cette possibilité permet à l'OMIKRON de pouvoir communiquer avec d'autres dialectes basic. L'ASCII est universellement répandu sur les ordinateurs. Il est à noter que l'OMIKRON sera obligé de pré–compiler chaque ligne à la relecture, ce qui demande nettement plus de temps que la lecture du code FIT.

Chargement d'un programme sous l'éditeur standard.

Le chargement s'effectue là aussi en code FIT ou en ASCII.

LOAD "NOM"

Le chargement s'effectue depuis le chemin par défaut. Une instruction "NEW" est effectuée par l'OMIKRON puis le programme est chargé. Le nom du programme chargé devient le nom par défaut.

Exemples:

LOAD "MAG" : REM charge MAG.BAS depuis le chemin courant.
LOAD "C:\OMIKRON\DIVERS\MAGNETO.BAS": REM charge le programme MAGNETO.BAS depuis le chemin indiqué.

"LOAD" reconnaît automatiquement les programmes en ASCII. Mais attention : les programmes en ASCII doivent impérativement comporter des numéros de lignes, faute de quoi le chargement sera stoppé.

MERGE "NOM"

MERGE ajoute au programme déjà en mémoire le programme en ASCII indiqué. Exemples:

```
MERGE "TEO.BAS"
MERGE "D:\MASTER.BAS"
```

Si votre programme de base utilise des numéros de ligne, il faut impérativement que les numéros de ligne du bloc mergé soient différents des numéros de ligne du programme de base.

Exemple, vous avez le programme suivant en mémoire :

```
10 PRINT "On y vas"
20 Stat(50)
30 FOR T=1 TO 50: PRINT A!(T):NEXT T
```

Vous voulez charger le programme en ASCII suivant, qui vous sert à chaque fois que vous avez des statistiques à calculer :

```
1    REM ******** STATISTIQUE
2    DIM A!(100)
1000 DEF PROC Stat(N)
1010 LOCAL T
1020 FOR T=1 TO N:A!(T)=RND(50000): NEXT T
1030 RETURN
```

Après le MERGE, vous obtiendrez le programme suivant :

```
1   REM ******** STATISTIQUE
2   DIM A!(100)
10  PRINT "On y vas"
20  Stat(50)
30  FOR T=1 TO 50 : PRINT A!(T) : NEXT T
1000 DEF PROC Stat(N)
1010 LOCAL T
1020 FOR T=1 TO N:A!(T)=RND(50000):NEXT T
1030 RETURN
```

Les touches de fonctions

Il est possible de reprogrammer celles–ci de la façon suivante:

Exemple :

```
Key 5 = "RUN" [Return]
```

A partir de maintenant, si vous appuyez sur la touche de fonction F5, le mot 'RUN' sera automatiquement tapé. Si vous voulez que l'instruction 'RUN' soit automatiquement exécutée, il vous faut modifier la ligne de la façon suivante :

```
Key 5 = "RUN"+CHR$(13) [Return]
```

Quitter L'OMIKRON.

Pour quitter l'OMIKRON, il suffit de taper 'SYSTEM' ou [ALT]–[S], puis [RETURN] et enfin de répondre O (OUI), J (JA) ou Y (YES). L'OMIKRON est trilingue!

Les instructions VT–52.

Reportez–vous aux annexes et au manuel de référence pour plus de détails.

LE SUPER-EDITEUR

Passons maintenant au plus intéressant: Super-éditeur. Il a été écrit par Arthur SÖDLER pour répondre aux critiques qui avaient été formulées à propos de l'éditeur de la version 2xx. On lui reprochait d'un peu trop ressembler aux vieux éditeurs des huit bits, de ne pas utiliser la souris, d'être trop lent et de ne pas comprendre les menus. Eh bien! tout cela est arrangé !

- Utilisation de la souris pour définir les blocs, d'une manière très souple et rapide.
- Amélioration de la vitesse, jusqu'à trois fois plus rapide (en particulier pour l'affichage du programme à l'écran).
- Utilisation des menus pour l'appel des commandes. Ce qui ne l'empêche pas de comprendre les touches de fonctions.

Comment entrer dans le super-éditeur ?

Tapez 'EDIT' ou [HELP] depuis l'éditeur normal. Vous y êtes!

Les menus du Super-éditeur regroupent les fonctions par centre d'intérêts communs.

Nous allons explorer chaque menu du haut vers le bas. Nous avons conservé l'anglais dans les menus, car de nombreuses options sont aussi des instructions du basic, ce qui interdit leur modification. Mais nous traduirons chaque option dans ce manuel, n'ayez crainte.

Pour se servir des menus, il suffit de placer la souris sur le titre et le contenu du menu s'affiche. Choisissez l'option que vous désirez, puis cliquez sur le bouton gauche.

Vous pouvez cliquer sur le titre du menu. Dans ce cas, l'option la plus proche en dessous du titre sera sélectionnée.

Voici comment se présentent sur la première ligne les titres des menus.

TITRE: FILE (FICHIER)



SAVE *.* (SAUVER)

Sauvegarde du programme qui se trouve en mémoire. Cette option affiche le sélecteur d'objet, dès lors l'utilisation de ce sélecteur est standard mis à part les détails suivants :

- Le nom par défaut est celui qui se trouve en haut à droite dans le menu de l'éditeur.
- Si à la place du nom vous tapez "A:" ou "C:", vous obtiendrez le répertoire du lecteur A ou C. De plus, l'OMIKRON réaffichera le nom par défaut. Il vous suffira donc de cliquer "CONFIRMER".
- D'autre part, il est possible de conserver le chemin actuel comme chemin par défaut. Il suffit pour cela de confirmer sans qu'il y ait de nom dans la partie sélection.

Exemple: Après avoir choisi l'option 'SAVE', vous effacez le nom par défaut (avec plusieurs [Backspace] ou avec [ESC]) et tapez 'C:', puis [Enter]. Le sélecteur d'objet affiche le répertoire du lecteur C et le nom du fichier par défaut est replacé. Effacez ce nom, puis cliquez 'Confirmer' ou tapez [Enter]. Le nouveau chemin par défaut sera maintenant 'C:'. Vérifiez-le en redemandant la sauvegarde. L'instruction SAVE *.* est aussi accessible au clavier par [F8].

LOAD *.* (CHARGER)

Charge un programme qui se trouve sur disque dans la mémoire et efface celui qui pourrait déjà s'y trouver. Là aussi le sélecteur d'objet est utilisé et vous y retrouverez les mêmes astuces.

Pour obtenir LOAD *.* au clavier, utilisez [Sf]-[F8].

SAVE BLOCK *.* (SAUVER en ASCII)

Sauve un bloc en ASCII. Bien entendu, un bloc doit absolument être défini pour utiliser SAVE BLOCK. La sauvegarde est plus lente qu'une sauvegarde normale, car l'OMIKRON est obligé de décompiler le programme. Comme toutes les fonctions des menus faisant appel au sélecteur de d'objet, SAVE BLOCK utilise le changement de lecteur et la définition de chemin d'accès automatique.

Autre moyen d'accéder à cette commande : [F7][F8].

LOAD BLOCK *.* (CHARGER en ASCII)

Charge un bloc en mémoire. Contrairement à LOAD, LOAD BLOCK n'efface pas le programme existant. LOAD BLOCK intercale le bloc dans le programme exactement à l'endroit où se trouve le curseur. Le chargement est nettement plus lent, car l'OMIKRON doit précompilé chaque ligne avant de pouvoir charger la suivante.

Equivalent clavier : [F7][Sf]–[F8]

DIRECTORY (Répertoire)

Afficher à l'écran le répertoire du lecteur pris par défaut. Après le choix de DIRECTORY, il vous est demandé si le chemin par défaut vous convient (il s'affiche '*.*'). Confirmez par [Enter] ou modifiez le chemin ou le type de fichier puis tapez [Enter]. Les fichiers répondant aux critères du chemin d'accès sont affichés à l'écran sur deux colonnes. N'importe quelle touche vous permet de quitter cet écran ou de passer à la page suivante du répertoire.

NEW (NOUVEAU PROGRAMME)

Efface le programme. Confirmez par [RETURN]. Attention, une fois que vous avez autorisé l'OM à effacer le programme, il n'y a plus aucun moyen de le retrouver, sauf si vous l'avez sauvé auparavant.

QUIT (^C) (QUITTER)

Quitter le super–éditeur. Retour à l'éditeur standard. Pour quitter complètement l'OMIKRON, il faut ensuite tapez 'SYSTEM' [Return] puis confirmer par [O].

Vous pouvez obtenir cette commande par [CTRL]–[C].

TITRE: FIND (CHERCHER)



Le super–éditeur ne tient pas compte des majuscules/minuscules lors de la recherche. Par contre, les majuscules des chaînes de remplacement sont prises en compte.

FIND NEXT (CHERCHER SUIVANT)

Cherche la prochaine occurrence de la chaîne recherchée. Quand super–éditeur ne trouve plus de chaîne équivalente, il émet un 'ping' et laisse le curseur en place.

FIND (CHERCHER)

Cherche une chaîne de caractères. Après avoir choisi cette option, super–éditeur affiche sur l'avant–dernière ligne de l'écran le message 'SEARCH FOR:'. Sur la dernière ligne, vous devez indiquer la chaîne à rechercher. Un peu comme pour le sélecteur de fichiers, l'OMIKRON conserve l'ancienne chaîne recherchée. De cette manière, vous pouvez la corriger ou la conserver telle quelle. Pour supprimer l'ancienne chaîne, tapez [ESC]. Après avoir choisi la chaîne, tapez [Enter]. Le super–éditeur commence la recherche à partir de la position du curseur. Si vous voulez entreprendre la recherche depuis le début du programme, placez–vous sur la première ligne en tapant [Clr/Home]. Si il trouve la chaîne demandée, il s'arrête sur la première lettre de celle–ci. Bien sûr, l'écran est déplacé si besoin est. Dans le cas où la chaîne n'existe pas, super–éditeur ne bouge pas le curseur et émet un 'ping'.

Equivalent clavier: [F2][F2].

LIST (LISTER)

Lister toutes les occurences d'une chaîne. Super–éditeur affiche sur un écran spécial les lignes contenant la chaîne recherchée.

Chaque occurrence apparaît en inverse vidéo. Pour interrompre le listage, tapez [ALT] ou [Undo]. Pour voir les lignes suivantes, tapez n'importe quelle touche. Si vous trouvez la ligne qui vous intéresse, cliquez-la et super-éditeur se place automatiquement dessus à l'intérieur du programme.

Equivalent clavier : [F2][F3]

REPLACE ALL (TOUT REMPLACER)

Remplacer toutes les occurrences par une chaîne donnée. Après avoir choisi 'REPLACE ALL', super-éditeur affiche 'SEARCH FOR:' sur l'avant-dernière ligne et vous demande d'entrer la chaîne recherchée sur la dernière ligne. Tapez [Enter], puis super-éditeur affiche sur l'avant-dernière ligne 'REPLACE ALL BY:'. Effacez ou modifiez la chaîne, puis tapez [Enter]. Super-éditeur va modifier toutes les chaînes correspondantes en conséquence.

Equivalent clavier : [F3][F3]

QUERY REPLACE (REMPLACER A LA DEMANDE)

Remplacer à la demande une chaîne par une autre. Fonctionne exactement comme 'REPLACE ALL' en ce qui concerne l'entrée des chaînes. Lors d'un remplacement, super-éditeur vous demande de confirmer le remplacement par [Y] (ou [J] ou [O]), de passer à l'occurrence suivante par [N] ou d'abandonner la recherche par [ESC] ou [UNDO].

Equivalent clavier : [F3][F2]

FIND TOKEN (CHERCHER MOTS-CLES)

Chercher les mots-clés. Les mots-clés sont les variables, les procédures, les fonctions utilisateurs et les instructions de l'OMIKRON. La recherche des mots clés est largement plus rapide que pour les chaînes normales, car l'OMIKRON tient à jour une liste. Pour le reste, FIND TOKEN fonctionne comme 'FIND'.

Equivalent clavier : [F2] [Shift]-[F2]

LIST TOKEN (LISTER MOTS-CLES)

Lister les mots-clés sur un écran spécial. 'LIST TOKEN' s'emploie comme 'LIST'.

FIND DEF (CHERCHER DEFINITION)

'FIND DEF' cherche la définition d'une procédure ou d'une fonction. Fonctionne comme un 'LIST' standard.

Equivalent clavier : [F1]

RENAME TOKEN (RENOMMER MOTS-CLES)

Renomme les mots-clés. C'est une opération ultra-rapide qui échange le nom d'un mot-clé avec son nouveau nom. En fait, 'RENAME TOKEN' ne touche que quelques octets, ce qui explique sa vitesse.

LIST TO PRINTER (LISTER SUR IMPRIMANTE)

Envoie sur l'imprimante la liste demandée lors d'une opération avec l'une des fonctions de ce menu. Une fois cette option choisie, le super-éditeur vous demande combien de caractères votre imprimante accepte-t-elle par ligne. Le nombre le plus courant est 80, c'est pourquoi le super-éditeur l'adopte par défaut. Tapez [Enter] et l'impression commence. Si l'imprimante n'est pas prête, super-éditeur attend une vingtaine de secondes, puis annule l'impression. Par pitié, ne tentez pas de RESET!!! Normalement, l'OMIKRON continuera d'essayer d'imprimer, mais si vous le faites trop souvent, le système reboote!

FIND ERROR (CHERCHER ERREUR)

Chercher la première erreur de syntaxe en partant du début du programme. Super-éditeur positionne le curseur juste derrière l'erreur. Par ce procédé, il vous désigne l'emplacement de l'erreur dans la ligne. Si il n'y aucune erreur dans le programme, super-éditeur émet un 'ping'.

TITRE: BLOCK (BLOC)



INSERT (INSERER)

Insère le bloc actif à l'emplacement du curseur. Un bloc doit évidemment avoir été défini au préalable. Le bloc de départ n'est ni déplacé, ni annulé.

Equivalent clavier : [F7][F9]

MOVE (DEPLACER)

Déplacer le bloc actif à l'emplacement du curseur. Cette fonction déplace le bloc, ce qui signifie qu'il n'y a pas de copie réalisée.

KILL (SUPPRIMER)

Supprimer le bloc actif. A la suite du choix de cette option, vous devrez confirmer la suppression par [Enter].

Equivalent clavier : [F7][SHIFT]–[F9]

MARK BLOCK START (MARQUER LE DEBUT DU BLOC)

Repérer le début du bloc (curseur). Equivalent clavier : [F7][F7]

MARK BLOCK END (MARQUER LA FIN DU BLOC)

Repérer la fin du bloc. Equivalent clavier : [F7][SHIFT]–[F7]

LOAD BLOCK *.* (CHARGER LE BLOC)

Charger un bloc en mémoire. 'LOAD BLOCK' insère le bloc dans le programme exactement à l'endroit où se trouve le curseur.

Equivalent clavier : [F7][SHIFT]–[F8]

SAVE BLOCK *.* (SAUVER LE BLOC)

Sauver en ASCII un bloc. Bien entendu, le bloc doit absolument être défini pour utiliser SAVE BLOCK. Equivalent clavier : [F7][F8]

PRINT BLOCK (IMPRIMER LE BLOC)

Imprime le bloc actif. Une fois cette option choisie, le super–éditeur vous demande combien de caractères votre imprimante accepte–t–elle par ligne. Le nombre le plus courant est 80, c'est pourquoi le super–éditeur l'adopte par défaut. Tapez [Enter], et l'impression commence. Si l'imprimante n'est pas prête, super–éditeur attend une vingtaine de secondes, puis annule l'impression. Par pitié, ne tentez pas de RESET!!! Normalement, l'OMIKRON continuera d'essayer d'imprimer, mais si vous le faites trop souvent, le système reboote!

HIDE (DESACTIVER LE BLOC)

Désactive le bloc actif.

TITRE: MODE (MODE)



INSERT (INSERTION)

Passage en mode insertion/recouvrement. Le mode par défaut de l'OMIKRON est le mode insertion. Ainsi, quand un caractère est entré, les caractères de la lignes situés à droite sont décalés. En mode recouvrement, par contre, les caractères situés à droite sont effacés par le dernier caractère entré. Equivalent clavier : [CTRL]–[INSERT]

SWITCH SCREEN (PASSER D'UNE FENETRE A L'AUTRE)

Cf SPLIT SCREEN. Passage d'une fenêtre à l'autre. On peut aussi passer d'une fenêtre à l'autre avec la souris en cliquant sur le bouton gauche. Equivalent clavier : [SHIFT]–[F2]

SPLIT SCREEN (PARTAGER L'ECRAN EN DEUX FENETRES)

Partage l'écran en deux fenêtres. Par défaut, les deux fenêtres ont la même taille. Vous pouvez déplacer la barre de séparation

horizontale avec la souris et choisir ainsi la taille des deux fenêtres. Equivalent clavier : [SHIFT]–[F1]

```
 FILE  FIND  BLOCK  MODE  GO  RUN     Y:  249  X:  0  SIZE:  64612  ESSAI_3D.BAS
'
'*****************************************************************
'*****************************************************************
'*******                                                 *******
'*******   3-D                                           *******
'*******   ©Élie Vertz 1/6/88                            *******
'*******                                                 *******
    MODE =1
    FILL STYLE =2,4
    PBOX 0,0,LarecrX,HtecrX
    '
    Objet$= LEFT$(Objet$,8)
    FILL STYLE =0,8
    PRBOX 70,5,550,350
    PRBOX 5,5,60,100
    HtexX=15
    Texte(4,0,1,10,HtexX,Objet$)
    '
  ENDIF
  '
  TEXT STYLE =0
  TEXT HEIGHT =4
  TEXT 10,HtexX+8,"D="
```

CHANGE SIZE (CHANGER TAILLE DES CARACTERES)

Change la taille des caractères affichés à l'écran. Il y a deux tailles possibles, quelle que soit la résolution. Equivalent clavier : [SHIFT]–[F3]

```
'*****************************************************************
'*****************************************************************
'*******                                                 *******
'*******   3-D                                           *******
'*******   ©Elie Vertz 1/6/88                            *******
'*******   Version 4.20                                  OMIKRON
'*******                                                 *******
'*****************************************************************
'*****************************************************************
'
CLEAR 32256*3+10000
'
ON ERROR GOTO Plantage
ON HELP GOSUB Sortir_Du_Programme
'
Init_Donnees
Init_Graphique
Init_Gem
Creation_Objet
Vecteurs_Normaux
Init_Axes
Init_Decor
Sonnette
'
IF Resol%=2 THEN
 Table( MIN(Nbpts,90))
 Vues
 Pas_A_Pas
 'Quadri
ENDIF
'
IF Resol%=0 THEN
 REPEAT
  Trav%=2: SCREEN 1: LPOKE $44E,Ecran2!:Transfert:Ppal
  Trav%=1: SCREEN 2: LPOKE $44E,Ecran1!:Transfert:Ppal
 UNTIL Faux
ENDIF
'
Sortir_Du_Programme
'
'*
```

LINE NUMBERS (ACTIVER NUMEROS DE LIGNES)

Active/désactive les numéros de lignes. Attention: certains programmes utilisent des GOTO/GOSUB un numéro de ligne. Dans ce cas, il ne faut pas enlever les numéros de ligne car l'OMIKRON renumérote automatiquement si vous sortez du super–éditeur. Equivalent clavier : [CTRL]–[CLR/HOME]

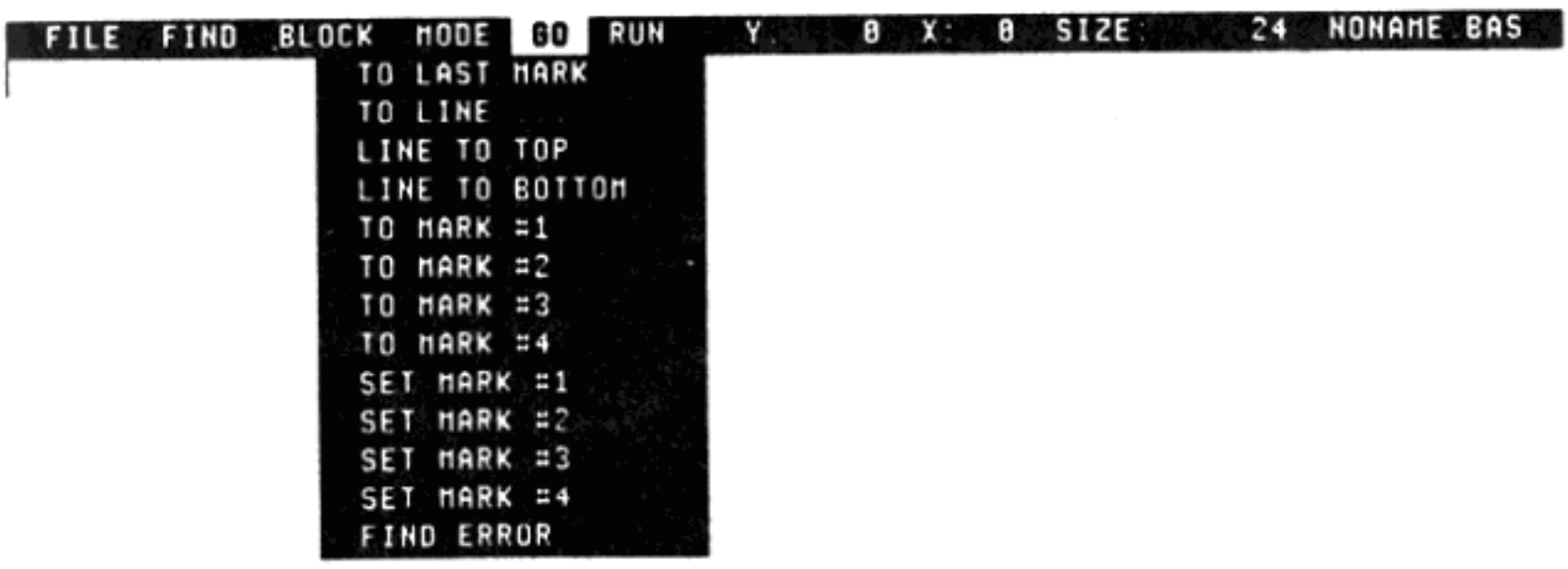
SHOW ERROR (AFFICHER LES ERREURS)

Active/désactive le signalement des erreurs.

SAVE SETTING (SAUVEGARDER LES PARAMETRES)

Sauvegarde les paramètres. L'OMIKRON sauvegarde sur le disque C: quand il existe ou sur le disque A: dans le cas contraire. Les paramètres concernés sont: – La taille des caractères ; – Le multifenêtrage ; – Le mode insertion/recouvrement ; – Le mode numéro de ligne ; – La recherche des erreurs ; – Les définitions de touches de l'éditeur standard ([F1], [F2] ...) – Les définitions de la souris et des touches [F4] et [F5] dans super–éditeur.

TITRE: GO (ALLER)



Ces options donnent à super–éditeur la puissance d'un traitement de texte.

TO LAST MARK (ALLER SUR DERNIERE MARQUE)

Positionne le curseur du super–éditeur sur la dernière marque définie.

TO LINE (ALLER A LA LIGNE...)

Positionne le curseur du super–éditeur à la ligne. De plus, il vous est possible de chercher la définition d'une procédure en tapant simplement "p" suivi de son nom. Super–éditeur se positionne alors sur la ligne contenant la définition. Equivalent clavier : [F1]

LINE TO TOP (AFFICHER LA LIGNE EN HAUT)

La ligne où se trouve le curseur va être affichée en haut de l'écran. Ceci permet de voir ce qui se situe en dessous sans pour autant perdre la ligne en cours. Equivalent clavier : [F1]–[cur up]

LINE TO BOTTOM (AFFICHER LA LIGNE EN BAS)

La ligne où se trouve le curseur va être affichée en bas de l'écran. Ceci permet de voir ce qui se situe au dessus sans perdre la ligne pour autant. Equivalent clavier : [F1]–[cur dwn]

TO MARK #1 (ALLER MARQUE 1)

Va à la marque numéro 1. On ne peut se rendre qu'à une marque déjà existante.

TO MARK #2 (ALLER MARQUE 2)

Va à la marque numéro 2. On ne peut se rendre qu'à une marque déjà existante.

TO MARK #3 (ALLER MARQUE 3)

Va à la marque numéro 3. On ne peut se rendre qu'à une marque déjà existante.

TO MARK #4 (ALLER MARQUE 4)

Va à la marque numéro 4. On ne peut se rendre qu'à une marque déjà existante.

SET MARK #1 (POSITIONNER MARQUE 1)

Positionne la marque numéro 1. SET MARK #1 remplace l'ancienne marque 1 par la nouvelle marque 1.

SET MARK #2 (POSITIONNER MARQUE 2)

Positionne la marque numéro 2. SET MARK #2 remplace l'ancienne marque 2 par la nouvelle marque 2.

SET MARK #3 (POSITIONNER MARQUE 3)

Positionne la marque numéro 3. SET MARK #3 remplace l'ancienne marque 3 par la nouvelle marque 3.

SET MARK #4 (POSITIONNER MARQUE 4)

Positionne la marque numéro 4. SET MARK #4 remplace l'ancienne marque 4 par la nouvelle marque 4.

FIND ERROR (CHERCHE ERREUR)

Cherche la première erreur en partant du début du programme. Le super–éditeur positionne le curseur juste derrière l'erreur. Par ce procédé, il vous désigne l'emplacement de l'erreur dans la ligne. Si aucune erreur n'est trouvée dans le programme, le super–éditeur émet un 'ping' du plus bel effet.

TITRE: RUN (RUN)



Ces options apportent énormément de confort et de simplicité à l'usage.

RUN (^R) (RUN)

Démarre le programme actuellement en mémoire. RUN remet à zéro les valeurs des variables. Il vous simplifie la vie, dans la mesure où vous ne devez pas sortir de super–éditeur puis tapez 'RUN', mais qu'il vous suffit de taper [CTRL]–[R]. Equivalent clavier : [CTRL]–[R]

SAVE & RUN (SAUVER & RUN)

Sauve le programme, puis le démarre. Tout comme 'RUN', cette fonction simplifie grandement les sauvegardes lors de la mise au point. Qu'y a t'il de plus énervant qu'un programme qui reset le système sans qu'on ait fait de sauvegarde? Et bien, cela ne vous

arrivera plus avec 'SAVE & RUN'.

TRON & RUN (TRON & RUN)

Met en route le mode TRACE et démarre le programme.

COMPILE (COMPILER)

Compile le programme actuellement en mémoire. C'est l'un des points forts de cette nouvelle version. Grâce à elle, vous compilez directement vos programmes depuis l'interpréteur, sans avoir besoin de quitter l'OMIKRON et de le recharger ensuite. En fait, si on ajoute la possibilité d'exécuter n'importe quel programme depuis l'interpréteur, le basic OMIKRON devient un environnement intégré de développement.

SAVE & COMPILE (SAUVER & COMPILER)

Sauve le programme, puis le compile. La sauvegarde s'effectue en code FIT. Cette option permet de ne cliquer qu'une seule fois au lieu de deux, ce qui est tout à fait pratique.

RUN *.BAS (RUN *.BAS)

Exécute un programme se trouvant sur disque. **Attention!** cela équivaut à charger un nouveau programme, donc à effacer le programme actuellement en mémoire. Si le programme actuellement en mémoire a une certaine valeur pour vous et que vous ne l'avez pas encore sauvé, faites-le avant de lancer le nouveau programme par un 'RUN *.BAS'.

EXEC *.PRG (EXECUTER *.PRG)

Exécute un programme se trouvant sur le disque. Dès que le programme est terminé, on retourne dans le super-éditeur. Grâce à 'EXEC *.PRG', l'OMIKRON autorise l'accès à la grande majorité des programmes tournant sur ATARI ST. Vous pouvez sans problème exécuter un assembleur, un traitement de texte ou un tableur, puis revenir à votre OMIKRON. Une restriction: il faut que le programme que vous appelez se conforme aux normes édictées par ATARI en ce qui concerne GEM. Ce n'est pas toujours le cas!

ACCESSORY (ACCESSOIRES)

Permet d'afficher un mini-menu, dans lequel se trouvent les accessoires. De là, tous les accessoires s'exécutent sans difficulté.



AUTRES POSSIBILITES

L'OMIKRON utilise aussi la souris. Dans quelle mesure? C'est ce que nous allons voir maintenant.

La définition d'un bloc à la souris! Voici le principal attrait d'une souris dans un éditeur. Super-éditeur permet de définir un bloc en cliquant le bouton gauche de la souris et en le maintenant enfoncé jusqu'à la fin du bloc. Super-éditeur indique le bloc en inverse vidéo.

Il est possible de déterminer un bloc de moins d'une ligne.

```
'!!!
```

LE RESET

IMPORTANT:

Le bouton RESET, situé derrière votre ATARI, permet normalement de booter la machine, c'est à dire de la remettre à zéro. Le basic OMIKRON redirige le RESET. Au lieu de démarrer la machine, le RESET retourne dans le super-éditeur. Ainsi, en cas d'erreur grave, vous ne perdez pas votre programme, puisque vous êtes en mesure de stopper et de sauver le programme. Il s'agit d'un point extrêmement important pour la mise au point de vos programmes et d'une fonctionnalité unique, qu'aucun autre langage ne supporte.

LES ERREURS

L'OMIKRON. Basic est un véritable interpréteur. Ce qui signifie qu'il est possible d'exécuter et de sauver un programme même si des erreurs de syntaxe y sont trouvées. Il s'agit là encore d'une fonctionnalité très importante pour la mise au point, car cela signifie que vous pouvez sortir d'une ligne comportant une erreur et que vous pouvez quitter le super-éditeur n'importe quand en y laissant des erreurs.

De plus, cette possibilité facilite le portage de programmes en basic venant d'autres machines, puisque vous n'êtes pas obligé de corriger toutes les erreurs avant de pouvoir essayer le programme transféré. Simplement, l'exécution sera interrompue dès que la ligne comportant l'erreur sera rencontrée.

Il existe enfin plusieurs fonctions uniquement accessibles depuis le clavier.

Positionnement du curseur :

[Shift]–[CTRL]–[droit]	Le curseur se positionne en fin de ligne.
[Shift]–[CTRL]–[gauche]	Le curseur se positionne en début de ligne.
[CTRL]–[droit]	Le curseur se déplace vers la droite de 3 caractères. Passage sur la ligne suivante en cas de dépassement de la fin de la ligne.
[CTRL]–[gauche]	Le curseur se déplace vers la gauche de 3 caractères. Passage sur la ligne précédente en cas de dépassement du début de la ligne.
[CTRL]–[haut] ou [Shift]–[F10]	Le super–éditeur affiche la page précédente.
[CTRL]–[bas] ou [F10]	Le super–éditeur affiche la page suivante.
[TAB]	Passage à la tabulation suivante.
[Shift]–[CTRL]–[haut]	Début du programme
[Shift]–[CTRL]–[bas]	Fin du programme.
[Clr/Home]	Fin/début du programme.

Fonctions spéciales :

[Undo]	Annulation de toutes les opérations effectuées depuis le passage sur une autre ligne (par [Enter] ou par l'une des flèches).
[CTRL]–[Delete]	Suppression des caractères de la ligne situées à droite du curseur. Ces caractères peuvent être rappelés en cas de fausses manipulation par [Undo], mais uniquement si vous n'avez pas confirmé la ligne.

[F6] Multiplication d'un caractère. Pour répéter un même caractère 50 fois, il existe une fonctions très pratique. Tapez le caractère désiré, puis [F6]. Super-éditeur vous demande le nombre de caractères désirés, entrez-le puis tapez [Enter]. Les caractères s'afficheront à partir da la position du curseur.

[Shift]-[F7] Redéfinition des touches et du bouton droit de la souris. Cette fonction rendra service à ceux qui rédigent des programmes répétitifs. On peut définir cinq touches, à savoir [F4], [F5], [Shift]-[F4], [Shift]-[F5] et le bouton droit de la souris.

Comment procéder: Tapez:

[Shift]-[F7] [la touche à redéfinir] ...

La définition ...

[La touche à redéfinir]. Les possibilités de redéfiniton sont innombrables, mais il faut s'en tenir aux touches (on ne peut pas se servir de la souris pour appeler les options des menus, ni définir des blocs...).

Les raccourcis clavier :

L'OMIKRON comprend les raccourcis clavier du type [ALT]-[X] pour MOUSEX. En employant ces raccourcis, vous gagnerez beaucoup de temps sur l'écriture des programmes. Voici la liste:

[ALT]-[A]	ASC(	[ALT]-[N]	NEXT
[ALT]-[B]	BLOAD	[ALT]-[O]	OPEN
[ALT]-[C]	CONT	[ALT]-[P]	PRINT
[ALT]-[D]	DATA	[ALT]-[R]	RETURN
[ALT]-[E]	ELSE	[ALT]-[S]	SYSTEM
[ALT]-[F]	FOR	[ALT]-[T]	THEN
[ALT]-[G]	GOTO	[ALT]-[U]	USING
[ALT]-[H]	HCOPY	[ALT]-[V]	VARPTR
[ALT]-[I]	INPUT	[ALT]-[W]	WHILE
[ALT]-[K]	KEY	[ALT]-[X]	MOUSEX
[ALT]-[L]	PRINT	[ALT]-[Y]	MOUSEY
[ALT]-[M]	MID$(		

Nous allons terminer par un récapitulatif des commandes au clavier. Chaque commande a été expliquée en détail, c'est pourquoi vous ne trouverez qu'un bref résumé.

'EDIT' ou [Help]		Entrer dans le super–éditeur.
[F1] <Ligne>	GO	Aller à la <ligne>.
[F1] <+ n >	GO <+n>	Aller n lignes plus loin.
[F1] <– n >	GO <–n>	Aller n lignes moins loin.
[F1] [haut]	GO [haut]	Ligne actuelle en haut de l'écran.
[F1] [bas]	GO [bas]	Ligne actuelle en bas de l'écran.
[Shift]–[F1]	SPLIT SCREEN	Multi–fenêtrage.
[F2] [F2]	FIND	Chercher une chaîne.
[Shift]–[F2]	SWITCH SCREEN	Passage sur l'autre fenêtre.
[F3] [F2]	QUERY REPLACE	Remplacer une chaîne à la demande.
[F2] [F3]	LIST	Lister occurences d'une chaîne.
[F3] [F3]	REPLACE ALL	Remplacer toutes les chaînes.
[F2] [Shift]–[F3]	FIND TOKEN	Chercher un mot–clé.
[F3] [Shift]–[F3]	LIST TOKEN	Lister occurences d'un mot–clé.
[Shift]–[F3]	CHANGE SIZE	Changer taille des caractères.
[F4]		Touche de fonction redéfinissable.
[Shift]–[F4]		Touche de fonction redéfinissable.
[F5]		Touche de fonction redéfinissable.
[Shift]–[F5]		Touche de fonction redéfinissable.
[F6]	REPEAT	Répétition du dernier caractère.
[Shift]–[F6] [F9]		Coupure d'une ligne.
[Sf]–[F6] [Sf]–[F9]		Collage de deux lignes.
[Shift]–[F7]		Définition des touches de fonctions.
[F7] [F7]	MARK BLOCK START	Marquer début de bloc.
[F7] [Shift]–[F7]	MARK BLOC END	Marquer fin de bloc.
[F7] [F8]	SAVE BLOCK *.*	Sauver bloc (ASCII).
[F7] [Shift]–[F8]	LOAD BLOCK *.*	Charger bloc (ASCII).
[F7] [F9]	INSERT	Insérer un bloc.
[F7] [Shift]–[F9]	KILL	Supprimer le bloc.

[F8]	SAVE *.*	Sauver le programme.
[Shift]–[F8]	LOAD *.*	Charger un programme.
[F9]		Insérer une ligne.
[Shift]–[F9]		Supprimer la ligne courante.
[F10]		Page suivante.
[Shift]–[F10]		Page précédente.
[Clr/Home]		Aller début/fin de programme.
[CTRL]– [Clr/Home]	LINE NUMBERS	Avec ou sans numéros de lignes.
[CTRL]–[C]	QUIT EDIT	Quitter l'éditeur.
[CTRL]–[R]	RUN	RUN.
[ALT]–<...>		Raccourcis clavier.
[Undo]		Annuler la dernière commande.
[Undo]		Annuler/remettre la dernière saisie de ligne.
[Tab]		Aller de tabulation en tabulation.
[CTRL]–[Insert]	INSERT	Mode insertion/recouvrement.
[CTRL]–[gauche] ou [droite]		Aller de 3 caractères en 3.
[CTRL]–[Sf]– [gauche]		Début de ligne.
[CTRL]–[Sf]– [droite]		Fin de ligne.
[CTRL]–[Delete]		Effacer la fin de la ligne.
[CTRL]–[up]		Page précédente.
[CTRL]–[bas]		Page suivante.

ORGANISATION DES VARIABLES

Au tout début, les ordinateurs ont permis le développement de la bombe atomique américaine. Ensuite, les scientifiques s'en servirent pour calculer les trajectoires des fusées Apollo (la lune!!!).

Le calcul, voilà un domaine où l'OMIKRON excelle. Sa précision et sa rapidité, son respect des standards au niveau des formats numériques (normes IEEE) font de lui l'allié naturel des mathématiciens et physiciens.

Pour répondre à tous les besoins, l'OMIKRON dispose de plusieurs types de variables: Flag (Booléen), Entier et virgule flottante (réel).

Voici une présentation rapide des types de variables de l'OMIKRON:

Type de variable	Suffixe	Domaine d'utilisation
Flag (Booléen)	%F	0 ou –1
Octet	%B	0 à 255
Entier court (16 bits)	%W ou %	–32768 à 32767
Entier long (32 bits)	%L ou rien	–2147483658 à 2147483657
Réel simple (6 octets)	!	$\pm$5,11E$^{+/-4931}$ (9 chiffres significatifs)
Réel double (10 octets)	#	$\pm$5,11E$^{+/-4931}$ (19 chiffres significatifs)

ATTENTION :

Les variables Flag et Octet ne sont utilisables que sous la forme de tableaux.

Utilisation des variables avec les suffixes :

```
Prim%F(N)=0 : REM l'élément N du tableau Prim(booléens) = 0.
A!=3.5 : REM la variable A (réel simple) recevra la valeur 3,5.
A#=1.3 : REM A (réel double) recevra la valeur 1,3 (réel simple).
```

Malheureusement, les suffixes rendent la lecture des programmes plus compliquée. Aussi, l'OMIKRON permet de définir un ou plusieurs groupes de variables comme appartenant à un type déterminé.

Voici les commandes à utiliser:

DEFINT	Entier court
DEFINTL	Entier long
DEFSNG	Réel simple
DEFDBL	Réel double
DEFSTR	Chaîne de caractères

Exemples:

- DEFDBL "A": Définit toutes les variables dont l'initiale est A comme des réels doubles.
- DEFINT "I,J,Q": Toutes les variables commençant par I, J, Q sont des entiers courts.
- DEFSNG "A–Z": Définit toutes les variables comme des réels simples.
- DEFSTR "G, K–O": Toutes les variables ayant pour initiales G, K, L, M, N et O sont des chaînes de caractères.

Bien entendu, DEFXXX n'affecte que les variables qui n'ont aucun suffixe:

Pas de #, ! , %, %L, %W, %B, %F ou de $.

Une variable A! reste réel simple, même après DEFDBL "A", alors que la variable Apple sera considérée comme un réel double.

IMPORTANT:

La commande DEFXXX doit toujours être entrée au tout début du programme, avant que vous n'ayez tapé quoi que ce soit. Il ne suffit pas de taper votre programme, puis ensuite d'ajouter un DEFXXX en ligne 1. Cela ne marche pas. Toutes les variables sans suffixes resteront des entiers longs (qui est le type de variable par défaut). Mais ce n'est pas un désastre si vous avez oublié de mettre votre DEFXXX au début.

Dans ce cas de figure :

- Terminez l'écriture de votre programme
- Insérez DEFXXX sur la première ligne
- Sauvegardez le programme avec SAVE "Nom",A
- Rechargez–le avec LOAD BLOCK.

Le DEFXXX fonctionnera alors à la perfection.

Une autre méthode:

- Sauvez le programme en ASCII (SAVE,A)
- Effacez–le de la mémoire (NEW)
- Entrez sur la première ligne : DEFSNG "A–Z"
- Rechargez le programme du disque avec MERGE

Lors du portage de programmes en provenance d'autres dialectes basic, n'oubliez jamais de mettre un 'DEFSNG "A–Z"' sur la première ligne. En effet, la grande majorité des basics considèrent les réels simples comme le type de variable par défaut. Or, les programmes en basic font largement appel aux nombres réels, notamment avec les fonctions mathématiques. Nous emploierons la notation suivante, dans ce paragraphe uniquement, afin d'améliorer la lisibilité:

A=SIN(X) doit se comprendre: A!=SIN(X!)

L'OMIKRON est capable de calculer en simple ou en double précision. Toutes les fonctions ont la possibilité de travailler en double précision, au détriment du temps d'exécution. Le fait de disposer de deux précisions constitue une certaine incompatibilité, dans la mesure où les interpréteurs basic ne comprennent en général qu'un seul type de variable réelle.

Mais au fait, quelles sont les différences entre simple et double précision ?

Les deux formats acceptent les mêmes limites en ce qui concerne le nombre maximum et le nombre minimum, à savoir –5,11E–4931 comme minimum et +5.11E+4931 comme maximum. Ces limites sont très largement suffisantes pour n'importe quel usage. Pour donner un repère, les scientifiques estiment qu'il y a moins de 10E100 atomes dans tout l'univers.

La simple précision dispose de 9 chiffres significatifs, la double précision de 19 chiffres significatifs. Mais il s'agit d'un minimum et, bien souvent, l'OMIKRON dispose de 10 et 20 chiffres significatifs.

Les deux formats employés par l'OMIKRON n'ont pas été choisi au hasard: ils correspondent aux normes IEEE, ce qui assure la compatibilité avec les co–processeurs arithmétiques (le 68881 de Motorola, le 80287 d'INTEL...) et avec des langages comme TURBO PASCAL (au niveau du format des nombres, s'entend).

Si les réels (ou virgules flottantes) sont si pratiques, il existe tout de même certains inconvénients à leur utilisation (rien n'est parfait). La méthode de calcul des nombres réels implique des erreurs d'arrondis, quelque soit la précision utilisée. Les erreurs d'arrondis sont inhérentes à la méthode et il faut bien s'en accomoder.

Expliquons pourquoi les erreurs d'arrondis surviennent: Le nombre de chiffres après la virgule est obligatoirement fini. Mais les nombres exprimés ne sont pas, eux, toujours finis.

Prenons 1/3 : 1/3=0.33333333333333333333 etc...

Pour stocker 1/3 en mémoire, vous pouvez employer autant de décimales que vous voulez, il est impossible d'obtenir exactement un tiers. L'OMIKRON effectue l'approximation la plus correcte possible, mais il s'agit d'une approximation.

De plus, les nombres sont stockés en mémoire sous forme binaire, ce qui signifie que certains nombres qui ne nous posent aucun problème présentent de grandes difficultés pour l'OMIKRON. Essayez ceci sur votre ordinateur:

```
? 10000 - 10000.1
```

Vous serez surpris du résultat. Entre 10000 et 0.1 l'écart est bien plus important en binaire qu'en décimal. Représentez–vous 1/10 en binaire: cela ne tombe pas juste, il faut aller chercher 1/16+1/32+1/64...

Afin de limiter au minimum les erreurs d'arrondis, il vaut mieux éviter de soustraire deux grands nombres très proches.

ORGANISATION DE LA MEMOIRE

Nous allons aborder ici une des parties les plus complexes d'un interpréteur Basic, à savoir la gestion de la mémoire. Nous nous limiterons à l'étude d'une fonction, FRE, mais vous trouverez d'autres renseignements dans le manuel de programmation.

FRE

Syntaxe : A=FRE(0)
Syntaxe : A=FRE(<Chaine$>)
Syntaxe : A=FRE("Lecteur")

FRE(0) permet de connaître la place disponible en mémoire. FRE retourne toujours un nombre entier et la taille est indiquée en octets.

La première syntaxe correspond au FRE standard, que l'on trouve sur tous les basics. Exemple:

```
? FRE(0)
```

La seconde syntaxe donne le nombre d'octets disponibles sur la chaîne spécifiée. Exemple:

```
? FRE(A$)
```

Enfin, FRE("Lecteur") retourne le nombre d'octets disponibles sur le lecteur spécifié. Exemple:

```
PRINT FRE("A:")
```

La seconde syntaxe effectue un GARBAGE COLLECTION avant d'afficher le nombre d'octets libres.

Qu'est-ce qu'un GARBAGE COLLECTION ?

Encore une fois, il faut bien comprendre que la gestion de la mémoire est l'une des fonctions les plus complexes d'un interpréteur basic. De la réussite de cette gestion dépend la qualité du basic.

Voici pourquoi la gestion de la mémoire est si compliquée:

```
A$="GOOD"
```

L'OMIKRON place la chaîne "GOOD" sur l'adresse la plus basse possible pouvant la contenir. Il doit donc disposer d'une plage de mémoire libre d'un seul tenant. De cette façon, il prend le moins de mémoire possible. Voici ce qu'on trouve en mémoire:

"GOOD0..." (le zéro indique un octet nul, dont la valeur est 0).

```
B$="MORNING"
```

Voici ce qu'on trouve en mémoire :

"GOOD0MORNING0..."

Si nous entrons:

```
A$=A$+B$
```

Que doit faire le basic ? Il y a deux solutions:

"MORNING0GOODMORNING0..."
|____B$____|__ A$=(exA$+B$)__|

"GOOD0MORNING0GOODMORNING0..."
|_exA$_|____B$____|__A$=(exA$+B$)___|

Bien sûr, la première méthode utilise au mieux la mémoire. Mais il faut recopier entièrement en mémoire la chaîne B$. Dans la réalité, il faut déplacer plusieurs, voire des centaines de chaînes. C'est rédhibitoire au niveau du temps d'exécution!!!

Pour contourner le problème, on est obligé d'utiliser la deuxième méthode. Cette fois, tout va bien, l'interpréteur est extrêmement rapide. Par contre, il reste au début de la mémoire une partie de la chaîne A$, qui est maintenant inutile.

Et pratiquement toutes les opérations portant sur des chaînes de caractères produisent ce genre de 'déchets'. A moins de disposer d'une mémoire infinie, il faut bien un jour la nettoyer pour récupérer la mémoire ainsi encombrée.

Quand le basic OMIKRON se rend compte qu'il n'a plus de place pour travailler, il effectue de lui-même un GARBAGE COLLECTION. Le GARBAGE COLLECTION peut parfois restituer plusieurs kilo-octets de mémoire.

Tapez l'exemple suivant :

```
B$="azert"
FOR T=1 TO 1000
    FOR U=1 TO 100
      A$=A$+B$
    NEXT U
    PRINT "Avant GARBAGE: "; FRE(0),"Après GARBAGE: "; FRE("")
NEXT T
```

Comme vous pouvez le voir, le GARBAGE peut être bien utile! En OMIKRON, un GARBAGE prend entre <1/200 ème de seconde et 5/200 ème de seconde. Pour apprécier à sa juste valeur cette performance, il faut savoir qu'un GARBAGE peut durer plusieurs secondes sur un micro-ordinateur comme le commodore 64.

LES CONVERSIONS DE TYPE

Les quatre fonctions qui viennent sont assez peu rencontrées dans un programme. Mais le basic OMIKRON les utilise automatiquement dans de nombreux cas (par exemple: A=SQR(100), où A est un entier long et où le résultat de SQR est un réel. Le basic fait de lui–même la conversion).

CDBL

Syntaxe:A#=CDBL(<Expression_Numerique>)

CDBL signifie 'Convert to DouBLe'. CDBL convertit une expression numérique quelconque en un réel double précision.

Exemple :

```
Total# = CDBL(a!)
```

CSNG

Syntaxe : A! = CSNG(<Expression_Numerique>)

CSNG signifie 'Convert to SiNGle'. CDBL convertit une expression numérique quelconque en un réel simple précision.

Exemple :

```
Montant! = CDBL(Total# )
```

CINT

Syntaxe : A% = CINT(<Expression_Numerique>)

CINT signifie 'Convert to INTeger'. CINT convertit une expression numérique quelconque en un entier 16 bits. Bien entendu, en cas de dépassement de capacité, un message d'erreur interrompt l'exécution du programme.

Exemple :

```
Copies% = CINT(Total# )
```

CINTL

Syntaxe : A = CINTL(<Expression_Numerique>)

CINTL signifie 'Convert to INTeger Long'. CINTL convertit une expression numérique quelconque en un entier 32 bits. En cas de dépassement de capacité, un message d'erreur interrompt l'exécution du programme.

Exemple :

```
Nombre_Maxi = CINT(Total# )
```

Les fonctions qui suivent sont liées à la gestion du disque. Elles permettent de coder les nombres sur une chaîne de taille fixe. La relecture des chaînes s'effectue par les fonctions inverses, qui sont aussi expliquées ici.

MKD$

Syntaxe : A$ = MKD$(<Expression_Numerique>)

MKD$ transforme une expression numérique en une chaîne de caractères. Concrètement, comment fonctionne MKD$?

Tout d'abord, il faut savoir que MKD signifie MaKe Double, passer en double. MKD$ sauve tous les nombres sous la forme d'une chaîne de caractères de 10 octets de long, soit le format des réels double précision. Pour les nombres qui ne sont pas à ce format, MKD$ opère d'abord une conversion en réel double. A quoi sert MKD$?

On l'utilise conjoitement avec les fichiers. MKD$ permet d'obtenir toujours la même taille lors de la sauvegarde sur fichier. Exemple sans MKD$:

```
A# = 0.14785 : B# = 25 : C# = 222222222
WRITE #1, A#, B#, C#
```

Dans cet exemple, aucun des trois nombres ne tiendra la même place sur le fichier, ce qui peut poser des problèmes à la relecture. Voyons maintenant un programme identique avec MKD$

```
A# = 0.14785 : B# = 25 : C# = 222222222
WRITE #1, MKD$(A# ),MKD$(B# ), MKD$(C# )
```

Cette fois, tous les nombres occupent la même place sur le fichier, soit dix octets chacun.

Pour relire les données, il suffit d'utiliser la fonction inverse de MKD$, CVD.

CVD

Syntaxe : CVD(<Chaine$>)

CVD convertit une chaîne de caractères en un réel double précision. La longueur de la chaîne à convertir doit être de 10 caractères.

Exemple :

```
A# = CVD(MKD$ (999.111))
```

MKS$

Syntaxe : A$ = MKS$(<Expression_Numerique>)

MKS signifie MaKe Single, passer en simple précision. Convertit l'expression numérique en une chaîne de 6 caractères. Exemple :

```
A$ = MKS$ (9.263)
```

CVS

Syntaxe: CVS(<Chaine$)

CVS convertit une chaîne de caractères en un réel simple précision. La longueur de la chaîne à convertir doit être de 6 caractères.

Exemple :

```
A! = CVS(MKS$ (-969.781))
```

MKIL$

Syntaxe : A$ = MKIL$(<Expression_Numerique>)

MKIL signifie MaKe Interger Long, passer en entier long (32 bits). Convertit l'expression numérique en une chaîne de 4 caractères.

Exemple :

```
A$ = MKIL$(3652639)
```

CVIL

Syntaxe: CVIL(<Chaine$)

CVIL convertit une chaîne de caractères en un entier long (32 bits). La longueur de la chaîne à convertir doit être de 4 caractères.

Exemple :

```
A = CVIL(MKIL$(-969526))
```

MKI$

Syntaxe : A$=MKI$(<Expression_Numerique>)

MKI signifie MaKe Interger, passer en entier (16 bits). Convertit l'expression numérique en une chaîne de 2 caractères.

Exemple:

```
A$ = MKI$(3639)
```

CVI

Syntaxe : CVI (<Chaine$)

CVI convertit une chaîne de caractères en un entier court (16 bits). La longueur de la chaîne à convertir doit être de 2 caractères.

Exemple :

```
A% = CVI(MKI$(-966))
```

OPERATEURS DE BASE

Voici la liste des fonctions arithmétiques de base:

+	Addition
–	Soustraction
/	Division
*	Multiplication
^	Puissance
\	Division entière (avec des résultats entiers)
MOD	Modulo (reste de la division entière)

L'avantage de la division entière réside dans sa vitesse par rapport à la division normale. Lorsque vous divisez des entiers, pensez à '\'.

MOD

Syntaxe: A= B MOD C

Retourne le reste de la division entière de B par C.

Exemples :

```
? 5 MOD 3
2
OK
? 100 MOD 30
10
OK
```

OPERATEURS DE RELATION D'ORDRE

L'OMIKRON dispose des opérateurs suivants:

>	Plus grand que
<	Moins grand que
=	Egal à
>=	Supérieur ou égal
< =	Inférieur ou égal
<>	Différent de

Il est tout à fait possible d'entrer les opérateurs dans n'importe quel ordre :

<= ou =<

L'OMIKRON dispose de deux autres fonctions d'ordre:

MIN

Syntaxe: A=MIN(B,C)

MIN retourne la plus petite des deux expressions numériques.
Exemple :

```
B=10000 : C=5000
? MIN(B,C)
5000
OK
```

MAX

Syntaxe : A=MAX(B,C)

MAX retourne la plus grande des deux expressions numériques.
Exemple :

```
B=10000 : C=5000 : PRINT MAX(B,C)
10000
OK
```

OPERATEURS LOGIQUES

L'OMIKRON peut effectuer de nombreuses opérations logiques (booléennes).

LIMITES: Les opérateurs logiques travaillent sur des nombres entiers uniquement. Ce qui signifie que les réels sont convertis en entiers avant que l'opérateur logique n'intervienne. Si la conversion ne se déroule pas correctement, l'OMIKRON affiche une erreur. En ce qui concerne les problèmes possibles avec les opérateurs logiques, consultez le manuel de programmation.

NOT

Syntaxe : A=NOT B

NOT inverse tous les bits du nombre B.

Exemple :

NOT(0000 0100) = 1111 1011

	NOT	
Valeur Départ	0	1
Résultat	1	0

AND

Syntaxe: A= B AND C

AND entre B et C.

Exemple :

(1100) AND (1010) = 1000

	AND			
B	1	1	0	0
C	1	0	1	0
A	1	0	0	0

OR

Syntaxe : A= B OR C

OR entre B et C.

	OR			
B	1	1	0	0
C	1	0	1	0
A	1	1	1	0

XOR

Syntaxe : A= B XOR B

XOR entre B et C.

	XOR			
B	1	1	0	0
C	1	0	1	0
A	0	1	1	0

EQV

Syntaxe : A= B EQV C

EQV entre B et C. (B EQV C) = NOT(B XOR C)

	EQV			
B	1	1	0	0
C	1	0	1	0
A	1	0	0	1

IMP

Syntaxe : A = B IMP C

IMP entre B et C.

	IMP			
B	1	1	0	0
C	1	0	1	0
A	1	0	1	1

NAND

Syntaxe : A = B NAND C

NAND entre B et C. (B NAND C) = NOT(B AND C)

	NAND			
B	1	1	0	0
C	1	0	1	0
A	0	1	1	1

NOR

Syntaxe : A = B NOR C

NOR entre B et C. (B NOR C) = NOT(B OR C)

	NOR			
B	1	1	0	0
C	1	0	1	0
A	0	0	0	1

SHR

Syntaxe : A = B SHR N

L'ensemble des bits composants le nombre B est décalé vers la droite de N fois. Un décalage à droite correspond à une division par deux (non signée).

SHL

Syntaxe : A = B SHL N

L'ensemble des bits composants le nombre B est décalé vers la gauche de N fois. Un décalage à gauche correspond à une multiplication par 2 (non signée). En règle générale, vous pouvez remplacer SHL, SHR par * et \.

Exemple :

```
? 100 SHL 2
400
OK
```

BIT

Syntaxe : A= BIT (B,C)

BIT teste un bit du nombre C. Le paramètre B représente le numéro du bit à tester. Par exemple, B=0 teste le bit de poids le plus faible du nombre C.

Exemple :

```
10 C=8
20 A=BIT(0 , C)
30 PRINT A
RUN
 0
OK
```

A contiendra 0, car le bit 0 de C est à zéro (8=1000 en binaire).

FONCTIONS MATHEMATIQUES

INT

Syntaxe : A= INT(B)

INT supprime les chiffres après la virgule d'un nombre réel. Elle retourne donc la partie entière du nombre.

Exemples :

INT(2/3) : REM donne 0
INT(1) : REM donne 1
INT(7.8) : REM donne 7
INT(-3.7) : REM donne -4

NOTE : Les nombres négatifs sont traités de la façon suivante: INT arrondi au nombre inférieur, ce qui signifie que :

INT(-5.1) : REM donne -6 Prenez-y garde!

Limites : INT n'est utile qu'avec les nombres réels. En effet, INT d'un nombre entier est égal à ce nombre entier.

FIX

Syntaxe: A= FIX(B)

FIX supprime les chiffres après la virgule d'un nombre réel. Elle diffère de INT par le fait que les nombres négatifs sont arrondis au nombre supérieur. FIX correspond au TRUNC de PASCAL.

Exemple :

FIX(3.5) : REM donne 3
FIX(-2.5) : REM donne -2
FIX(0.1) : REM donne 0

Limites: FIX n'est utile qu'avec les nombres réels. En effet, FIX d'un nombre entier est égal à ce nombre entier.

FRAC

Syntaxe : A= FRAC(B)

FRAC retourne la partie décimale d'un nombre réel.

Exemple :

```
FRAC(3.56) : REM donne 0.56
FRAC(1) : REM donne 0
```

En ce qui concerne les nombres négatifs, FRAC(X) = X–FIX(X), donc :

```
FRAC(-3.7) : REM donne -0.7
```

Limites : FRAC n'est utile que pour les nombres réels, car FRAC d'un nombre entier retourne toujours la valeur 0.

ABS

Syntaxe: A= ABS(B)

ABS retourne la valeur absolue d'un nombre. Les nombres positifs ne sont pas modifiés, les nombres négatifs changent de signe et deviennent positifs.

Exemple :

```
ABS(47.55) : REM donne 47.55
ABS(-333.1) : REM donne 333.1
ABS(0) : REM donne 0
```

SGN

Syntaxe : A= SGN(B)

Détermine le signe d'un nombre. Lorsque le signe est positif SGN renvoie 1, si le nombre est négatif SGN renvoie –1. Par définition, SGN(0) est nul.

Exemple :

```
SGN(47.55) : REM donne 1
SGN(-333.1) : REM donne -1
SGN(0) : REM donne 0
```

RND

Syntaxe: ***A= RND(B)***

Génère un nombre semi–aléatoire.

Exemple:

```
PRINT RND(1)
```

Retourne une valeur comprise entre 0 et 0,99999999. Il s'agit du RND standard, qu'on retrouve dans tous les dialectes basic.

```
RND(X)
```

Avec un nombre entier différent de 1, RND retourne un nombre entier compris entre 0 et X–1.

```
RDN(6)+1
```

Retourne un nombre compris entre 1 et 6.

```
RND(0)
```

Retourne le dernier nombre aléatoire généré.

Limites : RND retourne toujours des valeurs réelles. De plus, le paramètre de RND est toujours convertit en entier.

DEG

Syntaxe: ***DEG***

Par défaut, les ordinateurs calculent les fonctions trigonométriques (SIN, COS...) en radians. Mais si vous effectuez vos calculs en degrés, vous allez devoir effectuer des conversions sans arrêt. Aussi, DEG passe en degrés toutes les opérations trigonométriques.

NOTE : les instructions CLEAR et RUN repassent au mode par défaut, le mode RADIAN.

RAD

Syntaxe : RAD

Après l'exécution de RAD, l'OMIKRON effectue tous les calculs trigonométriques en radians. Le mode radians est le mode par défaut de l'OMIKRON, à l'instar de la plupart des langages.

Exemple :

```
10 RAD
20 FOR I!=0 TO PI*2 STEP PI/18          : REM Ne pas oublier les !
30 DRAW 100+SIN(I!)*50, 100+COS(I!)*50  : REM qui indiquent
40 NEXT                                 : REM un nombre réel.
```

NOTE : le mode radian est forcé après CLEAR et RUN.

SIN

Syntaxe: A=SIN(B)

SIN calcule le sinus d'un angle. L'angle est normalement donné en radian, donc SIN(PI/2)=0. Le passage en degrés se fait par DEG. Dans ce cas, SIN(180)=0.

Exemple :

```
10 RAD
20 PRINT SIN(PI/4)
```

NOTE: L'OMIKRON calcule toujours les fonctions trigonométriques avec le maximum de précision, c'est à dire en double précision (19 chiffres significatifs). Il convertit ensuite au format numérique nécessaire (entier, réel simple ...). Cela assure la meilleure précision possible dans tous les cas.

COS, TAN, COT

Syntaxe : A= COS(B)
Syntaxe : A= TAN(B)
Syntaxe : A= COT(B)

Ces fonctions permettent de calculer le cosinus, la tangente et la cotangente de l'angle B.

SEC, COSEC

*Syntaxe : **A= SEC(X)***
*Syntaxe : **A= COSEC(X)***

SEC donne la sécante et COSEC la cosécante de l'angle X.

ATN ou ARCTAN

*Syntaxe : **A= ATN(B) ou A = ARCTAN(B)***

ATN retourne l'arc tangente de l'angle B.

ARCSIN, ARCCOS, ARCCOT

*Syntaxe : **A= ARCSIN(B)***
*Syntaxe : **A= ARCCOS(B)***
*Syntaxe : **A= ARCCOT(B)***

ARCSIN est la fonction inverse de sinus, ARCCOS est la fonction inverse de cosinus et ARCCOT la fonction inverse de cotangente.

Exemple :

```
A!=SIN(45) : B!=ARCSIN(A!)
PRINT B!
45
OK
```

Limites : L'argument de ARCSIN... doit être parfaitement cohérent, c'est à dire qu'il doit correspondre à un résultat possible de SINUS...

SINH, COSH, TANH, COTH

*Syntaxe : **A=SINH(B)***
*Syntaxe : **A=COSH(B)***
*Syntaxe : **A= TANH(B)***
*Syntaxe : **A= COTH(B)***

Ce sont les quatre fonctions hyperboliques disponibles sur OMIKRON.

SECH, COSECH

Syntaxe : A=SECH(X)
Syntaxe : A = COSECH(X)

SECH correspond à sécante hyperbolique et COSECH à cosécante hyperbolique. Ces fonctions sont très peu utilisées.

ARSINH, ARTANH, ARCOTH

Syntaxe : A=ARSINH(B)
Syntaxe : A=ARTANH(B)
Syntaxe : A=ARCOTH(B)

Ces fonctions correspondent aux arcs sinus, tangente et cotangente hyperboliques.

LN

Syntaxe : A= LN(B)

LN calcule le logarithme népérien (de base e, e=2,718281828). Dans la plupart des autres basics, cette fonction est appelée LOG().

Exemple :

```
? LN(2.718281828)
```

Limites : B doit obligatoirement être positif et supérieur à zéro.

LOG

Syntaxe : A = LOG(B,C)

LOG calcule le logarithme de base B du nombre C.

Exemple :

```
PRINT LOG(15,52)
1.4590733
OK
```

Limites : B comme C doivent être positifs et différents de 0.

EXP

Syntaxe : **A=EXP(B)**

EXP calcule l'exponentielle de base e de B.

Exemples :

```
? EXP(1500)
? EXP(1)
```

SQR

Syntaxe : A=SQR(B)

SQR retourne la racine carrée du nombre B.

Exemple :

```
? SQR(65536)
```

Limites : B doit être strictement supérieur à zéro.

FACT

Syntaxe : A=FACT(B)

FACT donne la factorielle de B, qui doit être un nombre entier positif. La factorielle peut s'exprimer sous la forme :

- FACT(X) = 1 * 2 * 3 * ... * X

Exemple :

```
? FACT(1500)/FACT(1499)
```

NOTE : La fonction FACT n'existe que dans peu de Basics, aussi, quand vous passez de l'OMIKRON vers un autre dialecte, assurez-vous de l'existence de FACT.

LES BASES NUMERIQUES

L'OMIKRON comprend trois bases numériques en plus du décimal. Ces trois bases sont :

- Le binaire (base 2)
- L'octal (base 8)
- L'hexadécimal (base 16)

Le Binaire

Il s'agit de la base dans laquelle travaillent les ordinateurs. La base deux ne comprend que 2 chiffres : 0 et 1. En OMIKRON, le préfixe du binaire est %.

L'Octal

Il s'agit de la base qui est encore utilisé sur certains ordinateurs. Il n'a pratiquement que des désavantages (format variant entre 1 et 3 chiffres pour la représentation d'un octet, entre autre). Le préfixe de l'octal est &.

L'Hexadécimal

C'est la base la plus répandue sur les ordinateurs. La base 16 comporte, en plus des chiffres décimaux 0 à 9, les lettres A à F correspondant à 10...15 en décimal. Le préfixe de l'hexadécimal est $.

Tableau récapitulatif des préfixes :

% ou &b	binaire
& ou &o	octal
&d décimal	(par défaut)
$ ou &h	hexadécimal

Il est entendu qu'un nombre ne change pas, quelle que soit sa représentation. Par exemple :

- CHR$($40)=CHR$(64).

Pour passer d'une base à l'autre, l'OMIKRON utilise trois fonctions :

BIN$

*Syntaxe : **A$=BIN$(<Expression_Numerique>)***

BIN$ convertit une expression numérique quelconque en une chaîne de caractères binaire.

Exemple :

```
? BIN$(8)
1000
OK
```

HEX$

*Syntaxe : **A$=HEX$(<Expression_Numerique>)***

HEX$ convertit une expression numérique quelconque en une chaîne de caractères hexadécimale.

Exemple :

```
? HEX$(65535);"    "; HEX$(&177)
FFFF    7F
OK
```

OCT$

*Syntaxe : **A$ = OCT$(<Expression_Numerique>)***

OCT$ convertit une expression numérique quelconque en une chaîne de caractères octale.

Exemple :

```
? OCT$($FF)
377
OK
```

LIMITES : Comme les opérations logiques, les conversions de bases n'acceptent une expression numérique que dans la mesure où elle respecte les limites imposées aux entiers longs.

VAL

Syntaxe : VAL(<Chaine$>)

VAL permet de passer en décimal une chaîne en octal,en binaire ou en hexadécimal.

Exemples :

```
PRINT VAL("$FF00")
65280
OK

PRINT VAL("&765")
501
OK

PRINT VAL("%110010")
50
OK
```

L'OMIKRON dispose aussi d'une fonction étonnante : vous pouvez convertir un nombre hexadécimal, octal ou binaire même s'il n'est pas entier!

Exemples :

- &1.4=1.5 en décimal
- $1.4=1.25 en décimal
- %1.1=1.5 en décimal

LES CHAINES DE CARACTERES

En utilisant les fonctions suivantes, vous êtes à même de manipuler les chaînes de caractères.

Addition de chaînes

Il est possible d'additionner deux chaînes ensembles. Pour additionner les chaînes, on utilise le signe '+'.

Exemple :

```
C$='"AAA" : D$='"DDD"'
B$=C$+D$
? B$
AAADDD
OK
```

Limites: la chaîne de caractères résultant de cette opération ne peut dépasser la limite de 32765 caractères.

Multiplication de chaînes:

L'OMIKRON dispose de la multiplication de chaîne, une fonction que les autres basics offrent sous la forme (diminuée) de STRING$.

*Syntaxe : <Chaine> * <Expression_Numerique>*

Exemples :

```
? "V" * 35
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
OK

A$='"OMIKRON" : PRINT A$*5
OMIKRONOMIKRONOMIKRONOMIKRONOMIKRON
OK
```

NOTE : Le caractère de multiplication se trouve obligatoirement APRES la chaîne à multiplier.

LEFT$

Syntaxe : A$=LEFT$(<Chaine>,<Expression_Numerique>)

LEFT$ retourne la chaîne de caractères correspondant aux n premiers caractères (deuxième paramètre) de la chaîne de départ (premier paramètre).

Exemples :

```
A$= "ATARI"" : PRINT LEFT$(A$,3) : REM donne ATA
PRINT LEFT$("DISQUE", 4) : REM donne DISQ
```

ERREURS COURANTES :

```
PRINT LEFT$("Vitesse",1000)
```

Impossible : la chaîne de départ contient moins de 1000 caractères.

```
PRINT LEFT$(A,7)
```

Impossible : Confusion de type, A est une variable numérique.

```
PRINT LEFT$(A$,B$)
```

Impossible: B$ n'est pas une variable numérique.

RIGHT$

Syntaxe : A$=RIGHT$(<Chaine>,<Expression_Numerique>)

RIGHT$ fonctionne comme LEFT$, mais commence à partir de la droite de la chaîne et non pas de la gauche.

Exemples :

```
A$="ATARI" : PRINT RIGHT$(A$,3) : REM donne  ARI
PRINT RIGHT$ ("DISQUE",4 ) : REM donne  SQUE
```

ATTENTION : RIGHT$ ne retourne pas la chaîne, il la découpe. Vous ne retrouverez donc pas les caractères en sens inverse.

MID$

Syntaxe:A$=MID$(<Chaine>,<Exp_Numerique>,<Exp_Numerique>)
Syntaxe:A$=MID$(<Chaine>,<Expression_Numerique>)

MID$ retourne les n caractères (troisième paramètre) à partir du a ième caractère (deuxième paramètre) de la chaîne de départ (premier paramètre). Les caractères sont comptés de la gauche vers la droite, dans le sens de la lecture. Si le troisième paramètre est omis, tous les caractères à partir du a ième seront retournés.

Exemples :

```
PRINT MID$("123456789",3,4)
3456
OK

PRINT MID$("ZERTFINDECHAINE",5)
FINDECHAINE
OK
```

Voici un exemple d'utilisation :

```
10 INPUT A$
20 FOR T=1 TO LEN(A$)
30    PRINT "Le ";T;"ième caractère de la chaine est: ";
40    PRINT MID$(A$,T,1)
50 NEXT T
```

LEN

Syntaxe : A=LEN(<Chaine>)

LEN retourne la taille d'une chaîne.

Exemples :

```
PRINT LEN("1234")
4
OK

A$="" : PRINT LEN(A$)
0
OK
```

VAL

Syntaxe : A=VAL(<Chaine>)

On utilise souvent dans les programmes des chaînes de caractères contenant des nombres, comme par exemple "$FFEF" ou "153". Le problème se pose quand on doit utiliser ces chaînes 'numériques' dans des calculs. La solution existe: il faut se servir de VAL.

VAL transforme une chaîne de caractères représentant un nombre en une valeur numérique. Voici quelques exemples:

```
PRINT VAL("$FFFF")
65535
OK

PRINT VAL("1234")
1234
OK
```

Bien entendu, VAL ne fonctionne que sur les chaînes valides. Une chaîne valide est constituée de chiffres, éventuellement des caractères hexadécimaux et des codes de contrôle ($, & et %). Dans le cas où la chaîne n'est pas transformable, VAL retourne la valeur 0. Regardons ensemble ce qui est possible et ce qui ne l'est pas :

VAL("KDZD")=0	Caractères sans correspondance.
VAL("$FF")=255	Précédées de $, la chaîne est hexadécimale et les caractères hexa sont autorisés.
VAL($FF)	Erreur de syntaxe $FF n'est pas une chaîne.
VAL("−132.25 123")=−132.25	En effet, dès qu'un caractère non traduisible est employé, VAL arrête la conversion.
VAL("1.25E6")=1250000	Il est tout à fait possible d'employer la notation scientifique.

STR$

Syntaxe : A$=STR$(<Expression_Numerique>)

STR$ est la fonction inverse de VAL. Elle convertit une expression numérique quelconque en une chaîne alphanumérique. Voici quelques exemples:

```
10 A$= STR$(PI) : A$= "*"+A$+"*" : PRINT A$
20 PRINT "*"+STR$(-5)+"*"
RUN
* 3.1415926653897932*
*-5*
OK
PRINT STR$($FFFF) : REM donne 65535
```

ATTENTION :

- Afin d'aligner les nombres, les nombres positifs sont précédés d'un espace.
- Le format employé par STR$ pour transformer les nombre en chaîne est subordonné à USING. Vous obtiendrez donc des résultats différents selon le format choisi avec USING.

ASC

Syntaxe: A=ASC(<Chaine>)

ASC retourne le code ASCII d'un caractère alphanumérique. Le code ASCII permet d'indiquer à l'ordinateur un caractère parmi 256 (généralement). Le code ASCII de 'A' est 65 ($41), le code ASCII de '0' est 48 ($30)...

Exemples :

```
PRINT ASC("R") : REM donne 82
A$="AZERT" : PRINT ASC(A$) : REM donne 65
```

LIMITES :

- Si la chaîne indiquée dépasse 1 caractère de long, ASC ne retourne que le code du premier caractère.
- Si la chaîne ne contient aucun caractère, ASC déclenche une erreur 'Appel illégal de fonction'.

CHR$

Syntaxe : A$=CHR$(<Expression_Numerique>)

CHR$ exécute l'opération inverse de ASC. Elle retourne un des 256 caractères ASCII (numérotés de 0 à 255) que l'ATARI connaît.

Exemple :

```
FOR T=32 TO 128: PRINT CHR$(T);: NEXT T
```

LIMITES : Il est interdit d'utiliser des expressions numériques dont la valeur serait inférieure à 0 ou supérieure à 255.

INSTR

Syntaxe : A=INSTR(<Chaine_Depart>,<Chaine_Recherchee>)
Syntaxe:A=INSTR(<Exp_Num>,<Ch_Dep>,<Ch_Recherchee>)

INSTR permet de rechercher une chaîne (deuxième ou troisième paramètre) dans une chaîne de départ (premier ou deuxième paramètre), en partant éventuellement du n ième caractère de la chaîne de départ (premier paramètre).

Si la chaîne n'est pas trouvée, INSTR retourne 0 ; sinon, INSTR retourne la position dans la chaîne de départ de la première occurence de la chaîne recherchée.

Exemple :

```
A$="123456" : PRINT INSTR(A$,"23") : REM donne 2
```

SPC

Syntaxe : A$=SPC(<Expression_Numerique>)

La fonction SPC construit une chaîne constituée uniquement d'espaces. Le nombre d'espaces est fixé par le paramètre. En OMIKRON, SPC fonctionne aussi avec les chaînes (la plupart des autres basics n'acceptent SPC qu'avec PRINT). Pour conserver la compatibilité avec ces basics, nous avons conservé l'instruction SPACE$.

Exemple :

```
A$=SPC(5)+"OMIKRON"
```

SPACE$

*Syntaxe : **A$=SPACE$(<Expression_Numerique>)***

Absolument identique à SPC.

TAB

*Syntaxe : **PRINT TAB(<Expression_Numerique>);...***

TAB ne s'utilise qu'avec PRINT. Positionne le curseur sur la colonne n avant de continuer l'affichage.

Exemple :

```
PRINT TAB(20); "OMIKRON "
REM Le texte commencera sur la vingtième colonne.
```

Limites : Il n'est pas possible de dépasser 79 comme valeur pour l'argument, ce qui correspond à la dernière colonne de l'écran.

STRING$

*Syntaxe : **A$=STRING$(<Expression_Numerique>, <Caractere>)***

STRING$ fonctionne comme la multiplication de chaîne, mais n'opère que sur un caractère à la fois.

Exemple :

```
A$=STRING$(5,"*")
```

MIRROR$

*Syntaxe : **A$=MIRROR$(<Chaine>)***

MIRROR$ est une fonction exclusive que vous ne trouverez qu'en OMIKRON. Elle inverse une chaîne.

Exemple :

```
? MIRROR$("OMIKRON") : REM donne NORKIMO
```

NOTE : On utilise généralement cette fonction pour retrouver le nom d'un fichier dans un chemin d'accès. Reporter–vous au manuel de programmation pour plus de détails.

UPPER$

Syntaxe: A$=UPPER$(<Chaine>)

UPPER$ transforme en majuscules tous les caractères d'une chaîne. Les caractères spéciaux, les chiffres et les majuscules ne sont pas modifiés. En pratique, on utilise principalement UPPER$ pour faciliter les tests après la saisie, un exemple :

```
INPUT "Votre nom: ";Nom$
IF NOM$="TRAMIEL" OR NOM$="Tramiel" OR ...
```

est remplacé avantageusement par :

```
INPUT "Votre nom: ";Nom$
IF UPPER$ (NOM$)="TRAMIEL" ...
```

LOWER$

Syntaxe : A$=LOWER$(<Chaine>)

LOWER$ transforme en minuscules tous les caractères d'une chaîne. Les caractères spéciaux, les chiffres et les minuscules ne sont pas modifiés.

MODE

Syntaxe : MODE <Chaine>

MODE permet de basculer sur le jeu de caractère national. Il existe plusieurs modes possibles: français, allemand, anglais et américain (mode par défaut). MODE a une action sur INPUT USING, UPPER$, LOWER$ et DATE$ (les caractères accentués sont passés en majuscules automatiquement). Pour DATE$, il existe trois formats possibles:

JJ/MM/AA	Format anglais(GB) et français(F)
MM/JJ/AA	Format américain(USA)
JJ.MM.AA	Format allemand(D)

Exemple :

```
MODE "F"  : PRINT DATE$ : MODE "D" : PRINT DATE$
MODE "GB" : PRINT DATE$ : MODE "USA" : PRINT DATE$
```

LES VARIABLES SYSTEME

L'OMIKRON tient à votre disposition plusieurs variables qui vous informent du déroulement du programme ou des événements extérieurs. En général, on ne peut pas assigner de valeur à ces variables.

MOUSEBUT

MOUSEBUT indique l'état des boutons de la souris. Elle peut prendre quatre valeurs différentes :

0	Pas de bouton
1	Bouton gauche
2	Bouton droit
3	Boutons gauche et droit

MOUSEX

MOUSEX contient la position horizontale de la souris. Cette variable varie en fonction des déplacements de la souris.

MOUSEY

MOUSEY contient la position verticale de la souris. Cette variable varie en fonction des déplacements de la souris.

Exemple :

```
MOUSEON
REPEAT
   PRINT @(MOUSEY\16, MOUSEX \ 8 ); MOUSEX; " "; MOUSEY
UNTIL MOUSEBUT = 2
REM Changez le 16 et le 8 en fonction de la définition de l'écran.
```

Vous trouverez dans le manuel de programmation une petite routine qui permet de positionner la souris n'importe où sur l'écran, en passant par MOUSEX et MOUSEY.

TIMER

TIMER contient le temps écoulé depuis la mise en route de la machine en 200 ème de seconde.

Exemple :

```
10 T=TIMER
20 FOR Y=1 TO 10000
30 NEXT Y
40 PRINT (TIMER-T) /200
RUN
0.245
OK
```

TIME$

TIME$ contient l'heure système. L'ordre de réception des données est le suivant: HH:MM:SS. Contrairement aux autres fonctions, il est possible d'assigner une valeur à TIME$. Bien entendu, il faut respecter le format indiqué ci–dessus.

Exemple :

```
TIME$="11 : 25 : 32" : WAIT 10  : ? TIME$
11:25:42
OK
```

DATE$

DATE$ contient la date système. L'ordre de réception des données varie en fonction des paramètres de l'instruction MODE. Le mode USA est choisi par défaut et dans ce cas, la date est reçue sous la forme :

- MMJJAA

En mode F ou D, DATE$ est de la forme suivante :

- JJMMAA

Comme pour TIMER$, il est possible d'assigner une valeur à DATE$. Bien entendu, il faut respecter le format indiqué ci–dessus.

PI

PI équivaut environ à 22/7. PI oblige l'OMIKRON à effectuer les calculs en double précision. On peut passer en simple précision en utilisant une variable simple précision à la réception.

Exemple :

- Pi!=PI

CSRLIN

CSRLIN retourne le numéro de ligne où se trouve le curseur.

POS

POS retourne le numéro de la colonne dans laquelle se trouve le curseur. La paramètre qui suit POS n'a aucune utilité.

Exemple :

```
MOUSEOFF
REPEAT
   IF MOUSEX <>X OR MOUSEY <>Y THEN
     PRINT @(( MOUSEY \16),( MOUSEX \8)); ""; : P =  POS(0)
     PRINT CHR$(27); "j ";@(0,0); P; " ";  CHR$(27); "k";
   ENDIF
   X = MOUSEX : Y = MOUSEY
UNTIL  MOUSEBUT = 2
```

LPOS

LPOS fournie le numéro de la colonne sur laquelle se trouve la tête de l'imprimante. En réalité, LPOS retourne le numéro de la colonne calculée par l'ordinateur. Donc, si votre imprimante dispose d'un tampon, vous obtiendrez le numéro de la dernière colonne envoyé à l'imprimante et non pas le numéro de la colonne où se trouve réellement la tête de l'imprimante.

Exemple :

```
? LPOS(0)
```

LIMITES : LPOS ne fonctionne pas en mode graphique. LPOS ne retourne pas le bon numéro de colonne si des codes ESC ont été envoyés. LPOS ne fonctionne que si l'imprimante est branchée (!!!)

ERR, ERL, ERR$

Ces trois variables contiennent des éléments succeptibles de faciliter le débugage en cas d'erreur.

ERR	Numéro de l'erreur (cf Liste des erreurs).
ERL	Numéro de ligne où s'est produite l'erreur.
ERR$	Libellé de l'erreur.

Exemple :

```
ON ERROR GOTO Er_Corect
   PRINT A/(1-1)
   END
-Er_Corect
   PRINT ERR, ERL, ERR$
```

VERSION

Syntaxe : PRINT VERSION

VERSION est une variable qui contient le numéro de la version du basic OMIKRON utilisée. Si la version du basic OMIKRON que vous utilisez est la 3.01, vous obtiendrez 301 comme valeur dans VERSION.

Exemple :

```
PRINT VERSION
301
OK
```

Limites : VERSION ne fonctionne pas dans un programme compilé.

LES INSTRUCTIONS DE SORTIE

PRINT

Syntaxe : PRINT [<Chaîne>] ou [<Exp_Num>] [<Car_Controle>]

PRINT affiche à l'écran la ou les chaînes ou la (les) expressions numériques, suivit éventuellement de caractères de contrôle.

Exemples :

```
PRINT : REM Saute à la ligne suivante
PRINT "Omikron" : REM Affiche le mot Omikron à l'écran
PRINT Omikron : REM Affiche le contenu de la variable Omikron
PRINT "2*5" : REM Affiche "2*5" sur l'écran
PRINT 2*5 : REM Affiche 10 sur l'écran
```

PRINT dispose de deux caractères de contrôle :

- Le point–virgule qui oblige le PRINT à rester sur la ligne actuelle sans passer aucun espace.

Exemple :

```
PRINT "Car'" ; "l'ATARI" ;
```

Affiche sur l'écran "Carl'ATARI". Les deux mots sont rattachés. Pour l'affichage des nombres, le ';' laisse un espace derrière chaque nombre.

- La Virgule, qui permet au curseur de sauter à la prochaine tabulation sur la ligne. Il y a une tabulation tous les 8 caractères. Le prochain texte sera affiché à cet endroit.

Voici un exemple qui vous permettra de pratiquer un peu :

```
FOR An = 80 TO 87
    READ Turnover , Directeur$
    PRINT An, Turnover, Directeur$
NEXT An
DATA 50000, "ALIX", 500000, "ALIX", 750000, "ALIX"
DATA 48500, "SUPERMAN", 75000, "LEONARD"
DATA 215000, "ALAIN", 75000, "ALAIN", 25000, "ALAIN"
```

Vous pouvez utiliser plusieurs ";" et "," dans un même PRINT. Exemple :

'PRINT X; : PRINT Y;' est équivalent à : 'PRINT X ; Y;'

Autre possibilité :

```
PRINT "La valeur de X est " X
```

Pas de séparateur! L'OMIKRON ajoute tout seul les séparateurs. Il insère un point virgule automatiquement. Tapez ce qui suit :

```
10 PRINT "Salut" Adresse$; Nom$
```

Cette ligne sera transformée comme suit :

```
10 PRINT "Salut" ; Adresse$ ; Nom$
```

Il existe un raccourci clavier pour PRINT. Au lieu d'entrer les cinq lettres qui composent PRINT, vous pouvez taper [?], qui est l'abréviation standard du PRINT dans la plupart des basics.

'? 6*7' agit exactement comme : 'PRINT 6*7'

L'OMIKRON transforme automatiquement les "?" en "PRINT". Tapez ce qui suit pour vous en convaincre :

```
10 ? SIN(X) : REM Lister cette ligne après l'avoir écrite.
```

TAB

Syntaxe : PRINT... TAB(N)

Positionne le curseur sur la colonne N avant de continuer l'affichage.

Exemple :

```
10 PRINT TAB(20); "OMIKRON."
20 REM Le texte commencera sur la vingtième colonne.
```

Limites : Il n'est pas possible de dépasser 79 comme valeur pour l'argument, ce qui correspond à la dernière colonne de l'écran.

SPC

*Syntaxe : **PRINT...SPC(N)***

Affiche n espaces avant de continuer.

Exemple :

```
PRINT "OMIKRON." ; SPC(8) ; "SOFTWARE"
REM 8 espaces entre les mots OMIKRON. et SOFTWARE.
```

Limites : Il n'est pas possible de dépasser 32765 comme valeur pour SPC, ce qui constitue le maximum pour une chaîne de caractères.

PRINT USING

*Syntaxe : **PRINT USING <Masque> ,<Chaine>***

PRINT USING formate les affichages numériques. On utilise principalement PRINT USING pour afficher les tableaux. PRINT USING permet par exemple d'aligner les nombres sur la virgule, ce qui rend la lecture moins fatigante. L'exemple le plus simple consiste à indiquer le nombre de caractères avant et après la virgule. Pour ce faire, on utilise le masque d'édition :

```
A !=1.5 : PRINT USING "####.##" ,A!
 1.50
OK
```

Le masque d'édition accepte les commandes suivantes:

	Exemples	
#	"####"	Réserve l'emplacement d'un chiffre ou du signe du nombre. En effet, il ne faut pas oublier de réserver un emplacement pour le signe, même si celui-ci n'est pas visible (+). Ainsi l'exemple permet d'afficher un nombre qui aurait au maximum trois chiffres sur sa partie entière.
.	"###.##"	Le point placé de cette manière sépare l'emplacement réservé pour les parties entière et décimale du nombre.

.	".#####"	Le point placé au début de la chaîne indiquera la séparation des milliers. Dans ce cas, seule la virgule peut servir à positionner les décimales. Exemple : ".#####,#",1252.14 donnera 1.252,1
,	"###,##"	Même principe que le point dans le contexte correspondant.
,	",###.#"	La virgule va servir de séparateur tout les trois chiffres et dans ce cas c'est le point qui peut servir de point décimal.
–	"-###"	Dans tous les cas, le signe moins est affiché si la valeur est négative. Placer un signe moins ne sert qu'à indiquer sa position. Exemple : "###.####-",-1.258 donnera 1.2580-
+	"###+"	Force l'affichage du signe plus si le nombre est positif et indique l'emplacement du signe, qu'il soit positif ou négatif.
*	"*$####"	Le caractère qui suit l'étoile est utilisé comme caractère de remplissage. En règle générale, on choisi le 0 pour remplir les emplacements vides. Le caractère "_" n'est pas autorisé comme caractère de remplissage. Exemple : "**####.#",57.9 donnera ****57.9
_	"##_*##"	L'underscore force l'affichage du caractère qui le suit. On l'utilise seulement pour les caractères de commande car on peut placer n'importe quel autre caractère n'importe où dans la chaîne. Exemple : "##_+,## °C",20.78 donnera 20+,78 °C
^	"##^^^^"	Affichage en notation scientifique. Chaque "^" réserve un caractère pour cette notation. Exemple : "##^^^^^",1000000 donnera 1E +6

Un exemple récapitulatif :

PRINT USINC ",*******.** YEN " , Total !**

Affiche la valeur de la variable Total, avec deux chiffres après la virgule et une virgule pour séparer les milliers. Les premiers caractères sont remplacés par des '*' (à retenir pour les chèques).

Si Total!=1234.67 alors le résultat sera : *****1,234.67 YEN

Important:

- A moins que le masque ne contienne une indication contraire, le signe est toujours placé devant le nombre affiché. Pour demander un signe derrière le nombre, on indique un "+" ou "–" derrière le masque.
- On doit toujours prévoir un caractère pour le signe. "###.##" permettra l'affichage de 99 mais pas de 100, car il n'y a que 2 caractères de libres pour l'affichage.
- On peut mettre dans le masque l'affichage complet de ce qu'on désire à l'écran: "*0## Mille ### Francs, ## centimes –"
- Les erreurs de syntaxe dans une ligne contenant un USING proviennent souvent d'une erreur dans le masque.

USING

Syntaxe : USING <Masque$>

USING utilisé sans PRINT permet de formater l'affichage des commandes à venir. Le USING reste actif tant qu'un autre USING n'est pas exécuté. Pour annuler tout formatage de chaîne, il suffit de taper USING sans autre argument.

LPRINT

Syntaxe : LPRINT [<Chaine$>][,<Chaine$>]...

LPRINT fonctionne exactement comme PRINT, excepté le fait qu'il adresse l'imprimante au lieu de l'écran. Il est possible d'abréger LPRINT par L?.

PRINT#

Syntaxe : PRINT# <Canal>,<Sortie>

<Sortie> est une suite de chaînes alphanumériques, de variables numériques ou de constantes. PRINT# écrit les données sur le fichier de sortie spécifié. Le fichier doit avoir été ouvert auparavant. Il est possible d'abréger par ?#.

Pour plus de détails, reportez-vous au chapitre concernant les fichiers.

PRINT @

Syntaxe : PRINT @(<Ligne>,<Colonne>); <Sortie>

<Sortie> est une suite de chaînes alphanumériques, de variables numériques ou de constantes.

- PRINT@ doit se lire PRINT AT, (AFFICHER EN).

Il permet d'afficher à l'endroit déterminé par les deux paramètres ligne et colonne. La première position en haut à gauche de l'écran est accessible par :

```
PRINT@(0,0);"J'y suis, j'y reste"
```

@ n'est pas spécifiquement lié à PRINT et vous pouvez l'utiliser seul, dans une chaîne de caractères.

Exemple :

```
100  RAD
110  FOR X=0 TO 78
120  Y!=( SIN(X* PI /10)+ SIN(X* PI /8))/2
130  Sinu$ = Sinu$ +@(Y!*12+12,X) + "*"
140  NEXT X
150  PRINT Sinu$
160  A$ = INPUT$(1)
```

Limites: LPRINT @... ne fonctionne pas, car les imprimantes ne comprennent pas toutes les mêmes commandes de mise en page.

WRITE

*Syntaxe : **WRITE** <texte>*

Fonctionne comme PRINT. Les seules différences : WRITE entoure de guillemets ce qu'il affiche. Exemple :

```
WRITE "OMIKRON"
"OMIKRON"
OK

PRINT "OMIKRON"
OMIKRON
OK
```

De plus, WRITE affiche les virgules :

```
A=12 : B=15 : WRITE A,B,"Coucou"
12 ,15 ,"Coucou"
```

WRITE*

*Syntaxe : **WRITE*** *<Canal>,<Texte>*

Sortie séquentielle des données sur le fichier dont le canal est spécifié. Comme pour PRINT#, le fichier doit avoir été ouvert auparavant.

WRITE# est plus pratique que PRINT# car les guillemets et les virgules ajoutées par WRITE sont automatiquement reconnues par INPUT#. Il est donc possible de lire plusieurs champs avec un seul INPUT#, alors qu'il en faut un par champs avec PRINT#.

Exemple :

```
10 OPEN "O" ,1 ,"A:\NOM.FIC"
20 FOR T=1 TO 3
30    READ Nom$
40    WRITE*1,Nom$
50 NEXT T
60 CLOSE : END
70 DATA "Alix" , "Ghislain" , "Alain"
```

CMD

Syntaxe : CMD <Canal>

Nous abordons ici une commande particulièrement puissante. CMD permet de rediriger le vecteur des instructions de sortie.

Exemples :

```
OPEN "O",1,"TEST" : CMD 1
```

Déviation de toutes les instructions de sortie sur un fichier disque qui se nomme "TEST".

```
OPEN "P",4 : OPEN "O",8,"TEST2"
CMD 4 : ... : CMD 8
```

Déviation des instructions de sortie vers l'imprimante, puis redéviation des instructions de sortie sur le fichier TEST2.

Une utilisation possible de CMD consiste à tester l'imprimante et si l'imprimante n'est pas disponible, de rediriger toutes les instructions de sortie sur l'écran. Cette astuce est utilisée dans le programme VITESSE.BAS de la disquette originale.

LES ENTREES

L'OMIKRON dispose de cinq commandes permettant l'entrée de données. Il s'agit des fonctions standards du basic, mais il nous a semblé intéressant d'introduire une nouvelle fonction (INPUT USING) et de nouveaux modes de fonctionnement pour les commandes standards.

En effet, la saisie des données constitue l'une des fonctions principales des programmes. La plupart du temps, le programmeur invente sa petite bibliothèque, qu'il utilise et transforme au fur et à mesure du développement de son programme. Au bout d'un certain temps, il ne se souvient plus vraiment du fonctionnement de sa bibliothèque.

En fait, on s'aperçoit rapidement que l'agrément de la conception d'un logiciel repose aussi sur l'utilisation de fonctions standards, bien connues et qui ne changent pas toutes les semaines.

Voici les cinq commandes du basic OMIKRON :

INPUT	Ordre standard du basic.
LINE INPUT	Entrée d'une ligne complète.
INPUT USING	Voici la nouvelle commande de L'OMIKRON.
INKEY$	Interrogation du clavier, sans attente.
INPUT$	Entrée d'un nombre donné de caractères.

INPUT

Syntaxe: INPUT [@(Y,X);] ["texte";] <Variable> [,<Variable>...]

Le programme s'arrête, le texte apparaît sur l'écran, et l'utilisateur doit taper quelque chose au clavier. Pour terminer la saisie, il doit appuyer sur [Enter]. Les données entrées sont alors stockées dans la (ou les) variable(s). Puis le programme reprend son cours.

Exemple :

```
INPUT "Entrez un nombre s'il vous plaît : ";A
PRINT "Le carré de ";A;"est ";A^2
```

A l'exécution, 'Entrez un nombre s'il vous plaît: ?' apparaît sur l'écran et l'OMIKRON attend que vous entriez un nombre au clavier. Entrez 5, tapez [Enter] pour terminer la saisie. L'OMIKRON passe à la

ligne suivante est affiche 'Le carré de 5 est 25'. Il faut noter que INPUT place automatiquement un point d'interrogation à la fin de la ligne.

Si vous supprimer le texte dans INPUT, exemple :

```
INPUT A : REM Le basic OMIKRON n'affichera plus que '?'.
```

L'OMIKRON n'est pas (comme la majorité des basics) limité à une chaîne de caractères en ce qui concerne le texte. Le texte peut tout aussi bien être constitué d'opérations sur des chaînes. Illustration :

```
10  FOR I=1 TO 10
20     INPUT STR$(I)+"Oui : ";Entre$(I)
30  NEXT
```

Bien sûr, il est impossible de saisir des lettres par l'intermédiaire d'une variable numérique !

Et si vous voulez entrer du texte au lieu de chiffres? Aucun problème : il suffit de choisir une variable alphanumérique.

```
INPUT "Nom";NOM$
INPUT "Prénom";PRENOM$
PRINT "VOUS VOUS APPELEZ"; PRENOM$;" "; NOM$
```

Le nombre maximum de caractères saisis est de 255 caractères et autant de signes spéciaux, sans compter les virgules. Les virgules n'appartiennent pas à la chaîne entrée, car elles servent de séparateurs inter-variables. Quand l'entrée doit absolument contenir des virgules, utilisez plutôt les instructions LINE INPUT ou INPUT USING.

Un seul INPUT permet de renseigner plusieurs variables :

```
10  INPUT "3 NOMBRES ";A,B,C : PRINT  " Total = ";A+B+C
```

L'utilisateur devra saisir 3 variables séparées par des virgules :

```
RUN
3 nombres  ? 5,7,9 [Return]
Total = 21
OK
```

Le premier nombre est enregistré dans A, le second dans B, le troisième dans C.

C'est une façon intelligente de gagner de la place, en particulier pour entrer des coordonnées :

```
10 FOR I=1 TO 10
20    PRINT "Point n° ";I
30    INPUT "Entrez X et Y: ";X(I),Y(I)
40 NEXT I
```

Pour rendre le programme facile d'accès, vous pouvez séparer les INPUT: à chaque saisie correspond alors un texte d'information.

LINE INPUT

Syntaxe : LINE INPUT [@(Y,X):] ["texte";] <Variable> [,<Variable>...]

Cet ordre correspond pratiquement à l'ordre INPUT.

Limites: La saisie de plusieurs variables dans un seul LINE INPUT est interdite. Il faut un LINE INPUT pour chaque variable. De par sa définition, LINE INPUT refuse les variables numériques. Par contre, il autorise les virgules.

INPUT USING

Syntaxe : INPUT [@(Y,X):] ["texte";] <Chaine> USING [<Chaine_Com>] [,<Retour>] [,<Longueur>] [,<Car_Remplissage>] [,Position_Curseur>]

INPUT USING est l'une des instructions les plus complexes de l'OMIKRON basic. Nous ne vous recommandons pas de l'aborder avant d'avoir parfaitement compris comment se servir des autres INPUT.

INPUT USING réjouiera les amoureux du COBOL! C'est qu'il existe des instructions particulièrement puissantes en COBOL!

Par exemple :

- INPUT USING permet d'interdir ou d'autoriser la saisie d'un nombre donné de lettres, de chiffres et de caractères spéciaux.
- Il peut transformer un caractère en un autre.
- etc.

CHAINE DE COMMANDES.

0	Tous les chiffres sont autorisés.
a	Toutes les lettres de l'alphabet sont autorisées.
%	Les signes spéciaux sont autorisés.
^	Autorise la combinaison ([CTRL][Caractère]).
+	Le caractère qui suit cette commande est autorisé.
–	Le caractère qui suit cette commande est interdit.

Exemple :

On doit saisir un nom. Toutes les lettres doivent être admises, ainsi que le trait d'union (noms composés, "Scalbert–Dupont"). La chaîne de commandes sera composé de cette façon :

```
Commande$ = "a+-" : REM lettres (a) et trait d'union (+-) admis.
```

TRANSFORMATION DES CARACTERES

C x1 x2	Convertir le caractère x1 en x2.
U	Passage majuscule, toutes les lettres sont passées en majuscules.
L	Passage minuscule, toutes les lettres sont passées en minuscules.

Exemple :

Dans un programme de facturation, il est nécessaire de saisir des nombres décimaux. En France, le séparateur est la virgule, mais l'OMIKRON, en bon anglophile qu'il est, utilise les points comme séparateurs. INPUT USING peut saisir des virgules et afficher des points décimaux. Voici ce qu'il faut faire:

```
Commande$ = "0 C,. +. +-"
```

INPUT USING acceptera les chiffres de 0 à 9, convertira le ',' en '.' et acceptera les signes '.' et '–'. Nous vous incitons à utiliser au maximum les espaces dans les chaînes de commandes. En effet, ils rendent les INPUT USING bien plus simples à lire et ne sont pas pris

en compte par l'OMIKRON. D'autre part, les commandes peuvent être entrées en majuscules comme en minuscules.

TERMINER LA SAISIE

On peut toujours quitter la saisie par [Return] ou [Enter]. Mais INPUT USING étant paramétrable à ce niveau, on peut prévoir d'autres possibilités de sortie. On peut rendre un programme plus maniable en sortant avec une autre touche que [Enter] :

X z — Sortir de la saisie par le code ASCII indiqué. L'entrée est interrompue dès que l'on appuie sur la touche dont le code ASCII est z.

S z — Sortir de la saisie par le scancode indiqué. Le scancode d'une touche est lié à son emplacement sur le clavier. Les claviers disposent tous des mêmes scancodes quelque soit la nationalité de la machine (cf manuel de programmation). La saisie est terminée dès que l'on appuie sur la touche dont le scancode est z. La sortie par le scancode rend possible la différentiation entre les touches du pavé numérique et celles du clavier principal; cela permet aussi de sortir avec des touches ayant un code ASCII nul (comme [fleche haut], dont le scancode est 72)

< — Sortie par la gauche du champ de saisie avec la touche [fleche gauche]

> — Sortie par la droite du champ de saisie avec la touche [fleche droite]

Pour entrer une date, vous demandez à l'utilisateur de remplir trois INPUT USING. Le premier est réservé au jour, le second au mois, le troisième à l'année. Pour simplifier le passage d'un INPUT à l'autre, utilisez '<' et '>' dans la chaîne de commande.

Exemple :

```
10 -J : INPUT @(10,10);"JJ/MM/AA ";J$ USING "0"+">",Ret,2
20 PRINT @(10,50); "/";
30 -M : INPUT @(10,51); M$ USING "0"+"<>",Ret,2
40 IF Ret = -2 THEN GOTO J
50 PRINT @(10,53); "/" ;
60 INPUT @(10,54); A$ USING "0"+"<",Ret,2
70 IF Ret = -2 THEN GOTO M
```

Vous récupérez le code de la touche ayant permis de sortir de la saisie dans Ret. Les valeurs standards sont :

0	Pour [Return] ou [Enter]
–1	Pour '>'
–2	Pour '<'

Pour les autres touches, Ret retourne 4 octets codés de la même façon qu'avec INKEY$.

Reprenons notre programme de facturation. Il nous faut maintenant saisir le code client, le nom, le prénom, l'adresse, le code postal, la ville et le numéro de téléphone du client.

Code Client :

Le code client tient sur 8 caractères. Il est composé de lettres majuscules et de chiffres. Les caractères spéciaux sont interdits.

```
Cod_Cli$="a 0 U "
```

Nom :

Le nom est composé de 30 caractères au maximum. On autorise les lettres majuscules, '–' et l'espace.

```
Nom$ = " U a +- +  "
```

Prénom :

Le prénom contient au maximum 25 caractères. Il peut être composé de lettres et de '–'.

```
Pre$=" a +- "
```

Adresse :

Au maximum 30 caractères. Pratiquement tous les caractères sont permis: lettres, chiffres, espace et trait d'union.

```
Adr$ = " a  0 +- + "
```

Adresse2 :

Il faut deux lignes d'adresse dans certains cas. Cette ligne sera remplie optionnellement par l'utilisateur. 30 caractères de long, mêmes spécifications que Adr$.

```
Adr2$ = "a 0 +- + "
```

Code postal :

Uniquement les chiffres. La longueur est de 5 caractères.

```
Cod_Pos$ = " 0 "
```

Ville :

Lettres, chiffres, espace, trait d'union et slash composent la panoplie de caractères destinée à Vil$. Tout en majuscules. Longueur de 20 caractères.

```
Vil$ = " a 0 +/ + +- u "
```

Téléphone :

Uniquement des chiffres...plus quelques caractères spéciaux destinés à améliorer la lisibilité ('.', '/' et '–'). On décide que les '–' seront remplacés par des '/' par soucis d'homogénéité.

```
Tel$ = " 0 C-/ +/ +. "
```

Remarquez bien cette ligne de commande: elle contient une conversion de '–' en '/'. Mais il faut impérativement déclarer '/' comme étant un caractère valide en saisie (+/).

Comme on veut simplifier au maximum la saisie de l'adresse, on décide qu'il est possible de passer d'un champs à l'autre par les flèches haut et bas. Pour ce faire, nous allons 'accrocher' les chaînes de commandes les unes aux autres. On définit donc une seule fois les touches qui serviront au passage.

```
"s" + CHR$($48) + "s" + CHR$($50)
```

s pour 'exit by scancode'. On retrouvera les touches employées lors de la sortie du champs de saisie dans la variable de retour. Comme expliqué précédemment, elle contiendra 0 pour [Return] ou [Enter], -1 pour une sortie par [Fleche droite], -2 pour [Fleche gauche]. Pour les autres touches, la variable de retour contiendra 4 octets correspondants exactement à INKEY$.

Nous utiliserons le (_) comme caractère de remplissage. Le caractère de remplissage par défaut est justement '_', aussi nous n'ajouterons rien. Il n'est pas nécessaire d'indiquer la position de départ du curseur dans la chaîne à saisir.

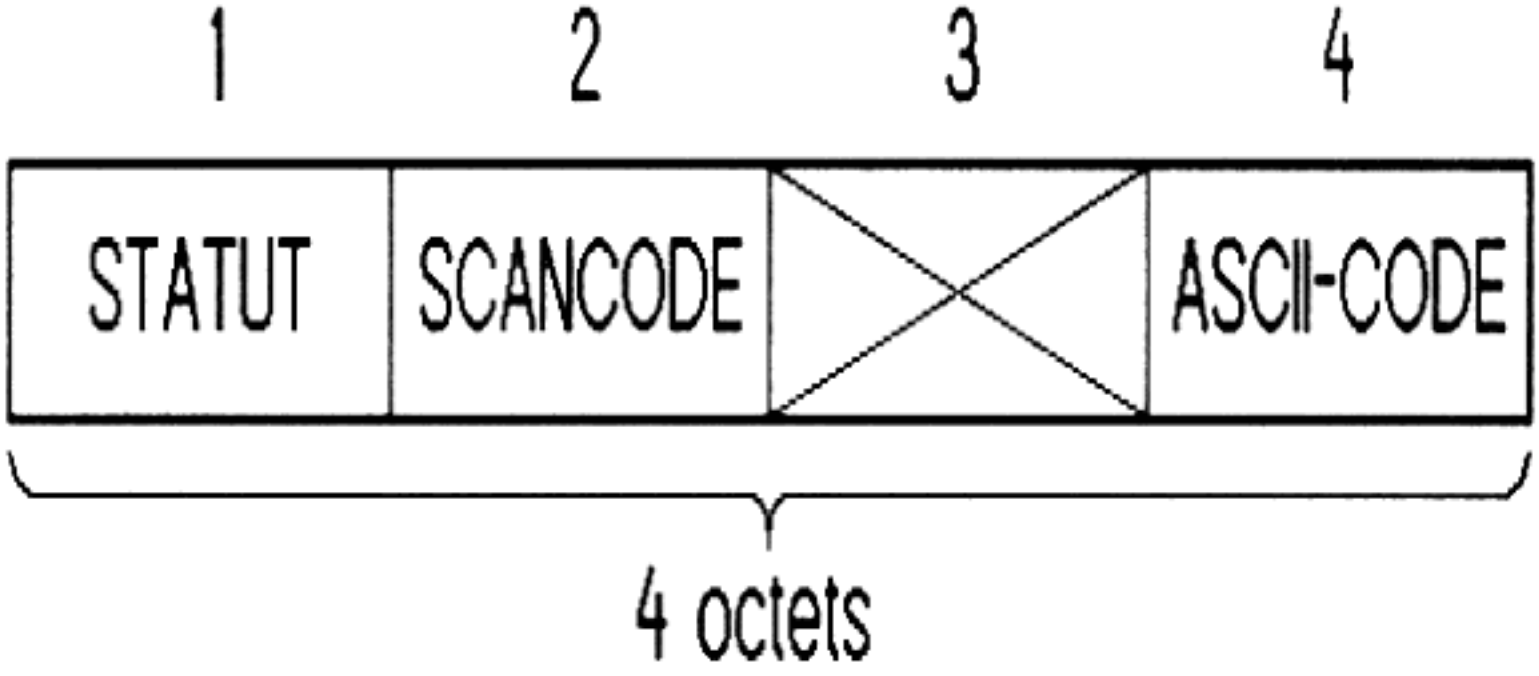
Voici le programme :

```
DEF PROC Saisie
    CLS
    PRINT @(3,5); "Code_Cli.: ";
    PRINT @(4,5); "Nom: ";
    PRINT @(5,5); "Prénom: ";
    PRINT @(6,5);"Adresse: ";
    PRINT @(7,5);"Adresse 2: ";
    PRINT @(8,5); "CP/Ville: ";
    PRINT @(10,5); "Téléphone: ";
    Exit$ = "s" + CHR$($48) + "s" + CHR$($50)
-Code : INPUT @(3,15); Cod$ USING "a 0 u "+Exit$,Ret,8
    IF (Ret AND $FF0000)=$480000 THEN GOTO Code
-Nom : INPUT @(4,15);Nom$ USING "a +- + u"+Exit$,Ret,30
    IF (Ret AND $FF0000)=$480000 THEN GOTO Code
-Pre : INPUT @(5,15);Pre$ USING "a 0 +- + "+Exit$,Ret,25
    IF (Ret AND $FF0000)=$480000 THEN GOTO Nom
-Adr : INPUT @(6,15);Adr$ USING "a +- + "+Exit$,Ret,30
    IF (Ret AND $FF0000)=$480000 THEN GOTO Pre
-Adr2 : INPUT @(7,15);Adr2$ USING "a +- + "+Exit$,Ret,30
    IF (Ret AND $FF0000)=$480000 THEN GOTO Adr
-Cp : INPUT @(8,15);Cp$ USING "0"+Exit$,Ret,5
    IF (Ret AND $FF0000)=$480000 THEN GOTO Adr2
    Cp$= LEFT$(Cp$+" 00000 ",5): PRINT @(8,15);Cp$;
-Vil : INPUT @(8,21);Vil$ USING "a0+ +/"+Exit$,Ret,25
    IF (Ret AND $FF0000)=$480000 THEN GOTO Cp
-Tel : INPUT @(10,15);Tel$ USING "0c-/+/+ "+Exit$,Ret,15
    IF (Ret AND $FF0000)=$480000 THEN GOTO Vil
RETURN
```

INKEY$

*Syntaxe : **A$=INKEY$***

INKEY$ est une fonction qui retourne une chaîne de caractères contenant la dernière touche tapée. INKEY$ n'interrompt pas le programme. En fait, l'OMIKRON ne retourne pas le code ASCII de la touche (ce que font les autres dialectes basic) mais une chaîne contenant quatre octets :



Le premier octet contient le statut des touches Shift

Shift droit	bit 0
Shift gauche	bit 1
Touche [CTRL]	bit 2
Touche [ALT]	bit 3
Statut [Caps Lock]	bit 4

Le second octet contient le scancode de la touche. Le troisième octet n'est pas utilisé. Le quatrième octet contient le code ASCII de la touche

Exemple :

```
10 A$ = ""                          : REM  Simulation du
20 WHILE  LEN(A$)=0                 : REM  INKEY$ d'un
30    A$=INKEY$                     : REM  Basic standard.
40 WEND                             : REM
50 PRINT RIGHT$ (A$,1)              : REM
```

INPUT$

Syntaxe : A$=INPUT$(<Nbr_Car>)
Syntaxe : A$ =INPUT$(<Nbr_Car>,<Num_Fichier>)

INPUT$ attend que le nombre de caractères spécifié ait été reçu du clavier avant de passer la main à l'instruction suivante. Les caractères sont stockés dans la variable alphanumérique réceptrice. Il n'y a pas de contrôle sur le type de caractère, contrairement à INPUT USING.

INPUT$ permet non seulement de scruter le clavier mais aussi de lire sur un fichier un nombre donné d'octets (un caractère correspond à un octet). Bien entendu, le fichier doit être ouvert en lecture! On peut ouvrir des fichiers sur le disque, sur les interfaces série, parallèle, MIDI et sur le clavier.

Exemple :

```
10 OPEN "V",1
20 A$=INPUT$(1,2) : REM Lire deux octets sur le canal 1
30 REM le canal 1 correspond ici au port série (OPEN "V").
```

LES COMMANDES

LIST

Syntaxe : LIST [-][<Exp_Numerique>][-<Exp_Numerique>]

LIST permet de visualiser le programme à l'écran. LIST n'est utilisable qu'avec l'éditeur standard, mais on peut aussi l'utiliser dans un programme.

Quelques exemples :

LIST	Listage de tout le programme sur l'écran.
LIST 100	Listage de la ligne 100 du programme.
LIST 100–	Listage de la ligne 100 à la dernière ligne.
LIST –100	Listage de la première à la 100 ème ligne.
LIST 100–200	Listage de la ligne 100 à la ligne 200.

On peut substituer le '–' par une ','.

Exemple :

```
LIST 100,200 : REM équivaut à LIST 100-200
```

Il est possible d'effectuer des calculs sur les numéros de ligne.

Exemple :

```
LIST (ERL),(ERL+100)
```

Bien entendu, on peut lister à partir d'une étiquette.

Exemple :

```
LIST Etiquette-1000
```

Le listage est extrêmement souple et permet de visualiser les lignes suivantes/précédentes grâce aux flèches.

On peut interrompre le listage par [CTRL]–[C], le suspendre par [CTRL]–[S] et le reprendre par [CTRL]–[Q].

LLIST

Syntaxe : LLIST [-][<Exp_Numerique>][-<Exp_Numerique>]

Imprime sur papier le listage demandé. LLIST fonctionne comme LIST (même paramètres...). Bien entendu, il est impossible d'utiliser les flèches pour réimprimer les lignes suivantes/précédentes.

Nous allons maintenant utiliser CMD avec LLIST. CMD, vous le savez, permet de rediriger toutes les sorties sur un fichier. On peut donc ouvrir des fichiers sur le lecteur de disquette, mais aussi sur le port série, parallèle, MIDI...Vous pouvez donc transmettre vos programmes directement par le minitel à un de vos amis. Vous trouverez un programme complet dans le manuel de programmation.

```
OPEN "V",1 : CMD 1
LLIST 100-200 : CLOSE 1
```

RUN

Syntaxe : RUN [<Expression_Numerique>]

Si une instruction est essentielle, il s'agit bien de RUN! RUN permet de démarrer un programme. Si vous êtes dans le super–éditeur, vous pouvez aussi taper [CTRL]–[R]. L'exécution d'un RUN efface toutes les variables. De plus, il vérifie si aucune erreur n'empêche le programme de se dérouler. Il vérifie les boucles, les IF...ENDIF...etc. En cas d'erreur, l'OMIKRON affiche le numéro, le type d'erreur et la ligne en cause.

RUN	Démarrage standard
RUN 1000	Démarrage à partir de la ligne 1000
RUN Etiqu	Démarrage à partir de l'étiquette Etiqu.

Si vous désirez conserver le contenu des variables intact, il suffit d'employer GOTO au lieu de RUN. Mais attention, un programme débuté par GOTO ne peut être repris par CONT après un [CTRL]–[C]. Il est possible de démarrer un programme à partir du disque:

```
RUN <Chemin_Acces>
```

Le programme spécifié est recherché sur le chemin, il est chargé puis exécuté. Le programme qui se trouvait en mémoire avant le RUN est perdu.

STOP

Syntaxe : STOP

Arrête le programme en cours. 'BREAK in <Numero_Ligne>' s'affiche à l'écran.

```
1000 PRINT "STOP arrive ..."
1010 STOP
RUN
BREAK in 1010
OK
```

Après un STOP, les fichiers restent ouverts, les variables conservent leurs valeurs. On peut effectuer des vérifications sur le contenu des variables, puis reprendre l'exécution du programme.

LIMITES : Il est impossible de continuer un programme ayant été modifié.

ATTENTION : Les fichiers ne sont pas fermés, et vous devez les fermer (par un CLOSE) avant de quitter l'OMIKRON.

CONT

Syntaxe : CONT

CONT permet la reprise d'un programme interrompu par [CTRL]–[C] ou STOP.

Exemple :

```
CONT <Etiquette>
```

CONT TO

Syntaxe : CONT TO <Etiquette> [TO <Etiquette>]

CONT TO est une évolution de CONT. CONT TO met en place des points d'arrêts (BREAK POINTS) dans un programme. CONT TO reprend l'exécution du programme à partir de la ligne interrompue (par [CTRL]–[C] ou STOP) et s'arrête à la première étiquette ou bien reprend l'exécution du programme à partir de la première étiquette et s'arrête à la seconde étiquette.

Exemples :

CONT TO 1000

Reprend le programme à partir de l'interruption, arrêt dès que la ligne 1000 est atteinte.

CONT TO 50 TO 100

Reprend le programme à partir de la ligne 50 et le stoppe en ligne 100.

END

Syntaxe : END

END termine le programme définitivement. END ferme tous les fichiers. Il est impossible d'utiliser CONT après END.

NEW

Syntaxe : NEW [<Chaine>]

Efface le programme en mémoire. Si un nom est donné, NEW attribue ce nom au nouveau programme. Après avoir effacé le programme, NEW effectue automatiquement un CLEAR.

Exemples :

```
NEW
NEW " SYSTEME.BAS "
NEW " SYSTEME "
```

LES BRANCHEMENTS

GOTO

Syntaxe : GOTO <Etiquette>

GOTO fait sauter l'exécution du programme à un endroit précis. <Etiquette> peut être un numéro de ligne ou une étiquette.

Exemples :

```
GOTO 500 : REM Saut en ligne 500.
GOTO Etat : REM Sauter à l'étiquette 'Etat'.
```

Pour sauter à une étiquette, celle-ci doit être définie.

```
-Jamais_Je_Ne_M_Arretes
GOTO Jamais_Je_Ne_M_Arretes
```

Pour définir une étiquette, il suffit de faire précéder le nom par un '-'. Si vous utilisez des GOTOs, nous vous conseillons fortement l'adoption des étiquettes. Vous n'aurez plus à vous préoccuper des numéros de ligne et les étiquettes sont quand même plus parlantes qu'un nombre.

```
GOTO 500+X
```

Il est possible d'effectuer des calculs sur les numéros de ligne.

```
GOTO (X)
```

Il est possible de sauter à un numéro de ligne par l'intermédiaire d'une variable numérique. Pour différencier la variable d'une étiquette, vous devez ajouter des parenthèses.

```
GOTO (A$)
```

Il est enfin possible de sauter à une étiquette par l'entremise d'une chaîne de caractères. Vous pouvez imaginer toute sorte d'astuces à partir de cela.

GOSUB...RETURN

Syntaxe: GOSUB <Etiquette>
Syntaxe: RETURN

GOSUB/RETURN sont les deux mots–clés qui, en basic standard, permettent l'exécution d'un sous–programme. L'OMIKRON comprend ces deux instructions et il est totalement compatible avec les autres basics sur ce point. Pour ceux qui ne connaissent pas GOSUB, apprenez que cette instruction saute exécuter un sous–programme et que RETURN retourne sur l'instruction suivant le GOSUB ayant servi à appeler ce sous–programme. GOSUB/RETURN permet de ne pas trop encombrer la partie principale du programme en extrayant les morceaux les plus répétitifs.

Exemple :

```
INPUT "Entrez A et B, je devine C";A$,B$
GOSUB Hypo
PRINT C
REM
REM suite du programme
REM
A=DIST1:B=DIST2: GOSUB HYPO' nouvel appel au sous-
programme Hypo
PRINT C
END
'
'
-Hypo
C=SQR(A^2+B^2)
RETURN
```

Les sous–programmes permettent de simplifier les programmes et de les raccourcir. Mais l'OMIKRON comprend un autre style de sous–programme: les procédures. Les procédures sont bien plus faciles à manipuler. Elles fonctionnent avec des variables locales et on peut les représenter comme des boîtes noires (on entre un ou des paramètres, il ressort un ou des résultats), ce qui simplifie grandement la mise au point d'un programme. Donc, regardez vite le chapitre consacré aux procédures et oubliez les GOSUB.

ON...GOTO

Syntaxe: ON <Expression_Numerique> GOTO <Etique1>,<Etique2>...

ON...GOTO remplace une succession de tests et de sauts. Dans le cas où l'expression numérique vaut 0, on passe à l'instruction suivant le ON...GOTO. Dans le cas où l'expression vaut 1, le programme saute à la première étiquette et ainsi de suite. Si la valeur de l'expression dépasse le nombre d'étiquettes, on passe à l'instruction suivant le ON...GOTO.

Exemple :

```
REPEAT
PRINT CHR$(27);"q"
PRINT @(5,25);"1°) Explications"
PRINT @(6,25);"2°) Un joueur"
PRINT @(7,25);"3°) Deux joueur"
PRINT @(8,25);"4°) High scores"
PRINT CHR$(27);"p"
INPUT @(10,25);"Entrez votre choix: ";Ch$
Ch= VAL(Ch$)
ON Ch GOTO Explic,Un_J,Deux_J,Hi_Sc
UNTIL 0
```

ON...GOSUB

Syntaxe : ON <Exp_Num> GOSUB <Sous_Prg1>, <Sous_Prg2>...

ON...GOSUB fonctionne comme ON...GOTO, mais saute à des sous-programmes. On passe donc à l'instruction située après le ON...GOSUB dès que le RETURN correspondant est rencontré.

PROCEDURES ET FONCTIONS

Le basic OMIKRON accepte les définitions de procédures et de fonctions.

LES PROCEDURES

Une procédure est l'équivalent d'un TOKEN (ou mot-clé) du basic. Une fois entrée, elle fait partie de la liste des mot-clés. Ce qui signifie, entre autre, que la procédure est accessible depuis l'éditeur standard sans qu'on ait besoin de taper 'RUN'. Tapez le programme qui suit dans le super-éditeur et tapez Hurle [Return] sous l'éditeur standard.

```
DEF PROC Hurle
    VOLUME 1,15: VOLUME 2,15: VOLUME 3,15
    REPEAT
      TUNE 1, RND(4000),2, RND(4000),3, RND(4000)
      NOISE 2, RND(32): NOISE 1, RND(31): NOISE 3, RND(32)
    UNTIL Fin_De_L_Alerte
RETURN
```

LE PASSAGE DE PARAMETRES

```
DEF  PROC Centrer( Texte$)
    PRINT  TAB (40-LEN(Texte$)/2);Texte$
RETURN
```

La procédure CENTRER place un texte au milieu de l'écran. Tapez le texte à centrer derrière le nom de la procédure (entre parenthèses ou non). Il sera alors automatiquement transféré dans la variable Texte$ de la procédure. Essayez :

```
Centrer ("ne panique pas !")
Centrer "Pas de parenthèse et ça marche !"
```

A lieu de passer des chaînes, vous pouvez aussi passer des variables :

```
A$="passez un bonne journée" : Centrer(A$)
```

Bien entendu, les variables passées doivent appartenir au même type que les variables attendues par la procédure. En cas d'erreur, comme par exemple Centrer(a), une erreur "CONFUSION DE TYPE" est affichée.

L'OMIKRON accepte aussi les variables locales à une procédure. Il faut aborder ici la différence entre variables locales et variables globales. Une variable globale est reconnue partout dans le programme : on peut la tester et la modifier dans le programme et dans les procédures.

Une variable locale appartient à une procédure : quand on modifie une variable locale, une variable globale portant le même nom n'est pas concernée.

```
A = 42
PRINT "Dans le corps du programme: ";A
Double(A)
PRINT "Toujours dans le corps du programme: ";A
END
DEF PROC Double (A)
    A=A*2
PRINT "On multiplie par deux le paramètre: ";A
RETURN
```

Bien que A soit modifié dans la procédure "Double", le contenu de cette variable reste inchangé après le retour de la procédure. En effet, un paramètre est, par défaut, considéré comme une variable locale à la procédure.

Dans le cas où vous utilisez dans la procédure des variables qui ne sont pas des paramètres passés lors de son appel, elles ne sont normalement pas locales mais globales. Prenons le même exemple pour le montrer :

```
A = 42
PRINT "Globale, corps du programme: ";A
Additionner (3)
PRINT "Globale, corps du programme: ";A
END
DEF PROC Additionner(Chiffre)
    A=Chiffre+A
    PRINT "Globale, procédure: ";A
RETURN
```

Mais vous pouvez définir des variables locales dans une procédure. Nous reprenons l'exemple précédent en le modifiant légèrement :

```
DEF PROC Additionner(Chiffre)
LOCAL A
```

Maintenant, la variable locale A de la procédure n'a plus rien à voir avec la variable A globale.

Bien entendu, l'OMIKRON est capable de passer plusieurs paramètres :

```
DEF PROC Bloquer(Texte$, Nombre_Espaces)
    LOCAL I
    FOR I=1 TO LEN(Texte$)
      PRINT MID$(Texte$, I,1)
      PRINT " "*Nombre_Espaces
    NEXT I
RETURN
```

Si vous tapez en mode direct 'Bloquer("Don't Panic!",1)', vous obtiendrez :

```
D o n ' t  P a n i c !
```

ou 'Bloquer(Don't Panic!",3)'

```
D  o  n  '  t     P  a  n  i  c  !
```

Nous venons d'aborder le passage de paramètres vers la procédure, ce qui est utile. Mais les procédures peuvent aussi retourner des valeurs. Il s'agit alors du retour de paramètres.

Exemple de retour de paramètres :

```
DEF PROC Plus_Court (R Cpt,R Nom):REM retour indiqué par R
    LOCAL I ,Gt : REM Deux variables sont locales
    FOR  I = MAX(Cpt,Nom) TO 1 STEP -1
      IF FRAC(Cpt/I) = 0 AND FRAC(Nom/I) = 0 THEN Gt = I : EXIT
    NEXT
    Cpt = Cpt/Gt : Nom = Nom/Gt
RETURN
```

Maintenant essayez :

```
X=36 : Y=42
Plus_Court(X,Y) : REM Pas de R lors de l'appel de la procédure
PRINT X, Y
```

Vous voyez que 36/42 devient 6/7.

Bien entendu, comme la plupart des langages modernes, l'OMIKRON est récursif et les procédures ainsi que les fonctions sont récursives.

La récursivité n'est pas toujours évidente à appliquer à un problème, mais nous allons vous donner l'exemple le plus fameux: FACTORIELLE. Comme chacun sait, la factorielle d'un nombre N équivaut à :

- FAC(N)=1*2*3*...*N

Nous allons voir un exemple de récursivité, non pas avec une procédure mais avec une fonction (en effet, dans ce cas, l'utilisation d'une fonction s'avère plus appropriée).

Voici la fonction Facto, qui retourne la factorielle d'un nombre :

```
DEF FN Facto(X)
   IF X=0 THEN RETURN 1
RETURN FN Facto(X-1)*X
```

La dernière ligne contient la récursivité (appel de la fonction par elle-même).

Comment cela fonctionne-t-il? Si par exemple, vous tapez PRINT FN Facto(3), le déroulement ressemblera à ce qui suit :

```
3>0 ALORS FN Facto(3-1)
   2>0 ALORS FN Facto(2-1)
      1>0 ALORS FN Facto(1-1)
            0 = 0 , la valeur 1 est retournée
      Le résultat est retourné en étant multiplié par 1
   le résultat est retourné en étant multiplié par 2
le résultat est retourné en étant multiplié par 3. Fini !
```

Ce qui nous amène directement à l'étude des fonctions.

LES FONCTIONS

Comme pour les procédures, l'OMIKRON permet les définitions de fonctions par l'utilisateur. Les fonctions autorisent elles aussi le passage de paramètres. Par contre, les fonctions ne sont capables de retourner qu'une seule valeur. En OMIKRON, il existe deux types de fonctions :

- Les fonctions qui tiennent sur une ligne ;
- Les fonctions qui tiennent sur plusieurs lignes.

FONCTIONS SUR UNE LIGNE

```
DEF FN Ttc(Montant)=Montant*1.186
```

Cette ligne décrit une fonction sur une ligne, qui porte le nom de Ttc. Tapez maintenant :

```
PRINT FN Ttc(100)
```

Lors de l'appel d'une fonction, FN précède toujours le nom de la fonction.

Le résultat de la fonction apparaît aussitôt : 118.6. Le nombre 100, indiqué entre parenthèses, est le paramètre passé à Ttc. Ce paramètre est multiplié par 1.186 et le résultat de cette multiplication est restitué en sortie dans la variable Montant qui est une variable locale. Ce genre de fonction est surtout utile pour les calculs :

```
Hors_Taxe=155:Port=5
PRINT "Total: ";FN Ttc(Hors_Taxe)+FN Ttc(Port)
```

Si votre fonction doit retourner une chaîne de caractères, vous devez placer le signe "$" à la fin du nom de la fonction. Exemple :

```
DEF FN Esc$(X) = CHR$(27)+CHR$(X)
```

Si vous disposez d'une imprimante EPSON ou compatible, le passage en mode gras s'effectue par :

```
LPRINT FN Esc$(69);
```

Naturellement, les chaînes de caractères peuvent aussi constituer de magnifiques paramètres. Exemple :

```
DEF FN Esc$(X$) = CHR$(27)+X$
```

Dans ce cas, le passage en gras sur votre EPSON se fera par :

```
LPRINT FN Esc$("E");
```

Mais si on désire utiliser plusieurs paramètres à la fois? Là encore, aucun problème. Voici quelques exemples :

```
DEF FN Midasc(Texte$,Place) = ASC(MID$(Texte$,Place,1))
DEF FN Hasard = -RND(-6)
```

Il est tout à fait possible d'appeler des fonctions depuis une fonction :

```
DEF FN Styl$(Styl) = FN Esc$(CHR$(Styl))
DEF FN Esc$(X$) = CHR$(27)+X$
```

Même si la fonction à laquelle on fait appel n'est définie que plus loin dans le programme (voir l'exemple ci–dessus).

FONCTIONS SUR PLUSIEURS LIGNES.

On peut décider d'employer des fonctions sur plusieurs lignes. Reprenons notre exemple précédent et transformons–le en une fonction sur plusieurs lignes :

```
DEF FN Ttc(Montant) : REM pas de signe égal
RETURN Montant * 1.186
```

On quitte une fonction comme on quitte une procédure: avec RETURN. La valeur de retour doit être indiquée après le RETURN.

Comparez avec la définition sur une ligne :

```
DEF FN Ttc(Montant)=Montant * 1.186
```

Une fonction à plusieurs lignes semble plus compliquée, mais elle offre de multiples possibilités. Exemple: Une société de vente par correspondance facture le port à ses clients de la façon suivante :

- Facture < 100 francs – forfait d'expédition
- Facture >= 100 francs – pas de forfait d'expédition

Cette règle ne peut s'écrire sous la forme d'une fonction sur une ligne, car on a besoin d'un IF...THEN...ELSE. Par contre, les fonctions sur plusieurs lignes acceptent sans difficulté les conditions :

```
DEF FN Port(Montant_Net)
IF Montant<100 THEN RETURN(5)
RETURN( 0)
```

Comme pour les procédures, les fonctions sur plusieurs lignes acceptent plusieurs variables locales (en dehors, bien entendu, des paramètres.)

```
DEF FN Total!(Montant_Net!)
LOCAL Montant_Brut!=Montant_Net!*1.186,Frais_Expe!
IF Montant_Brut!<100 THEN Frais_Expe!=5 ELSE Frais_Expe!=0
RETURN (Montant_Brut!+Frais_Expe!)
```

La variable Montant_Brut! est fixée comme étant locale et dans le même temps, on lui alloue une valeur. Il s'agit là d'une particularité du basic OMIKRON. Comme vous le voyez, on peut fixer le type de la fonction par le suffixe.

Les fonctions sur plusieurs lignes peuvent aussi agir sur des chaînes de caractères :

```
DEF FN Insert$(Texte$,Insert$,Place)
Texte$= LEFT$(Texte$,Place-1)+Insert$+ MID$(Texte$,Place)
RETURN Texte$
```

Limites : Les fonctions, qu'elles soient sur une ou plusieurs lignes, ne peuvent, contrairement aux procédures, contenir qu'un seul paramètre de retour.

EXIT n'est utilisable que sous la forme EXIT sans paramètre. La forme EXIT TO n'est pas interdite par l'interpréteur, mais le compilateur la rejette...

CONDITIONS ET BOUCLES

CONDITIONS

IF...THEN...ELSE...ENDIF

Ce groupe de commandes permet l'exécution conditionnelle d'instructions.

SI <Condition> ALORS -INSTRUCTIONS- SINON -INSTRUCTIONS- FINSI

IF <Condition> THEN -INSTRUCTIONS- ELSE -INSTRUCTIONS- ENDIF

Exemple :

```
-Lab : PRINT "J'ai pensé à un nombre. Tu dois le deviner."
INPUT "Tu trouves  "; Nb
IF Nb = 33 THEN PRINT "OK..." : END
IF Nb < 33 THEN PRINT "Trop petit ..." ELSE PRINT  "Trop grand !"
GOTO Lab
```

On peut exécuter plusieurs instructions après un seul THEN ou un seul ELSE.

Exemple :

```
IF Poids<100 THEN Port! = 1.3 : Emballage! = .75 : GOTO Calcul
```

Les trois instructions situées après le THEN sont exécutées si la condition (Poids<100) est vraie.

On convient qu'il est plus facile de lire un programme structuré. En fait, structurer un programme revient à mieux organiser l'espace occupé, à le mettre en page en quelque sorte. Quand on réfléchit à la forme d'un programme, on aperçoit mieux les erreurs de fond.

Une des méthodes simples consiste à séparer sur plusieurs lignes la condition et les instructions à exécuter en fonction du résultat de la condition. Il faut alors employer ENDIF à la fin du IF.

Exemple :

```
10 IF  UPPER$(Choix$) = "CALCUL"
20  ↑   THEN GOSUB En_Tete
30  |     ↑    GOSUB Calcul
40  |     |    GOSUB Expression
50  |     |    IF Som_Finale! > 10000 THEN PRINT "Merci. "
60  |     ↓    GOSUB Commission
70  ↓   ELSE IF UPPER$(Choix$) = "FIN" THEN GOTO Bilan
80 ENDIF
```

On peut aussi intégrer le THEN dans la ligne contenant le IF :

```
10 ┌→IF UPPER$(Choix$) = "CALCUL"  THEN
20 |       GOSUB En_Tete
30 |       GOSUB Calcul
40 |       GOSUB Expression
50 |       IF Som_Finale! > 10000 THEN PRINT "Merci. "
60 |       GOSUB Commission
70 ├→ELSE
80 |       IF UPPER$(Choix$) = "FIN" THEN GOTO Bilan
90 └→ENDIF
```

BOUCLES

Les boucles sont des parties de programme qui sont parcourues quand une ou des conditions sont remplies. Une boucle comporte obligatoirement un début et une fin. Nous allons aborder toutes les commandes permettant de construire une boucle existant en OMIKRON. Elles sont au nombre de trois :

- REPEAT...UNTIL
- FOR...NEXT
- WHILE...WEND

Même si le basic OMIKRON dispose de trois types de boucles différents, il faut bien savoir qu'on peut remplacer une boucle par une autre. Simplement, une certaine boucle sera plus facile à manipuler dans un certain contexte.

REPEAT...UNTIL

Syntaxe : REPEAT -INSTRUCTION- UNTIL <Conditions>

En français : REPETER...JUSQU'A

REPEAT...UNTIL teste la condition après avoir exécuté toutes les instructions qui se trouvent entre REPEAT et UNTIL. De ce fait, REPEAT...UNTIL est toujours exécutée au moins une fois.

Exemple :

```
PRINT "SVP, appuyez sur la barre espace"
REPEAT
    A$=INPUT$(1) : REM Attente d'une frappe au clavier
UNTIL A$ = "  "
PRINT  "Merci beaucoup "
END
```

La boucle REPEAT...UNTIL sera répétée jusqu'à ce que l'on tape sur la barre espace.

Un autre exemple pour que ça rentre :

```
CLS
PRINT "Cliquez le bouton gauche de la souris et on commence"
PRINT "Cliquez le bouton droit et on arrete..."
MOUSEON
REPEAT
    REPEAT UNTIL MOUSEBUT
    IF MOUSEBUT AND 2 THEN EXIT : REM touche droite = fin
    X = MOUSEX : Y = MOUSEY : REM point de départ
    REPEAT
        X1 = MOUSEX : Y1 =  MOUSEY
        LINE COLOR = 0
        MOUSEOFF  : DRAW X,Y TO X1,Y1 : MOUSEON
        REM efface la derniere ligne
        REPEAT
            Drapeau = (MOUSEBUT = 0) : REM souris cliquée?
        UNTIL X1<> MOUSEX OR Y1<> MOUSEY OR Drapeau
        REM Les 3 lignes précédentes testent un mouvement
        IF NOT Drapeau THEN
            LINE COLOR =1
            MOUSEOFF  : DRAW X,Y TO X1,Y1 : MOUSEON
            REM tracer la nouvelle ligne
        ENDIF
    UNTIL Drapeau
UNTIL Pour_Toujours
REM Pour_Toujours reste à 0, condition toujours fausse.
```

Ce programme trace sur l'écran des lignes partant d'un point fixe. En cliquant sur le bouton gauche de la souris, vous déterminez un point de départ. Tant que vous restez appuyé, vous tracez des droites. Appuyez sur le bouton de droite et la commande EXIT est exécutée pour quitter le programme.

WHILE...WEND

Syntaxe : WHILE <Conditions> -INSTRUCTIONS- WEND

Contrairement à REPEAT...UNTIL, WHILE...WEND teste la condition d'entrée avant d'exécuter les instructions. Une boucle WHILE...WEND ne sera donc parcourue que si la condition d'entrée est vraie.

```
INPUT "Quel fichier voulez vous consulter ?";Nom$
OPEN "I",1,Nom$
WHILE NOT EOF(1)
    A$ = INPUT$(1,1)
    PRINT A$;
WEND
```

Si le fichier demandé ne contient plus de données, le programme s'interrompra et affichera une erreur. WHILE...WEND permet d'exécuter le INPUT$ que si il reste des données à lire sur le fichier. Pour vérifier l'existence de données sur le fichier, il suffit de tester la 'NON fin' du fichier avec EOF.

FOR...NEXT

Syntaxe : FOR <Var_Num> = <Exp_Num> TO <Exp_Num>
[STEP<Exp_Num>] : -INSTRUCTIONS- NEXT <Var_Num>

Une boucle célèbre. Alors que WHILE et REPEAT sont utilisées quand le nombre d'itérations (c'est à dire le nombre de passages à l'intérieur de la boucle) n'est pas à priori connu, FOR...NEXT s'emploie quand on connait à l'avance le nombre d'itérations nécessaires. FOR...NEXT est plus rapide que les autres boucles.

Exemple :

```
PRINT "Les carrés des nombres de 1 à 10 :"
FOR I = 1 TO 10
    PRINT " Le carré de ";I;" est ";I^2
NEXT
```

On assigne la valeur 1 au compteur I au début de la boucle (FOR I=1). A chaque parcours de la boucle, sa valeur est incrémentée (augmentée de 1). Quand la valeur finale est dépassée (FOR I=1 TO 10 donc I=11), on sort de la boucle.

De plus, avec FOR...NEXT, vous pouvez indiquer un pas grâce à STEP. Le compteur ne sera plus incrémenté (+1), mais on lui ajoutera la valeur de STEP.

Exemple :

```
FOR Alpha= -90 TO 90 STEP 5 : PRINT SIN(Alpha) : NEXT Alpha
```

Ici, Alpha sera toujours augmenté de 5.

On peut aussi compter à rebours. Exemple :

```
FOR Sec = 10 TO 1 STEP -1
    PRINT "Encore ";Sec;" secondes jusqu'au départ ..."
    T=TIMER : REPEAT UNTIL TIMER-T > 200 : REM attendre 1 s.
NEXT : PRINT "Baoum "
```

Il est possible de sortir d'une boucle FOR...NEXT avant que le compteur n'atteigne la valeur limite. On utilise alors EXIT

EXIT

Syntaxe : EXIT [TO <Etiquette>]

EXIT permet de quitter une boucle, même si votre condition de sortie n'est pas remplie. On se sert d'EXIT quand une condition doit être testée à l'intérieur d'une boucle.

Exemple :

```
Num_Max = 100 : DIM Art(Num_Max)
PRINT "Entrez les articles. Fin avec '/'. "
FOR Art_Num=1 TO Num_Max
PRINT "Numéro de l'article: ";Art_Num
INPUT "Quantité à livrer: ";L$ : Qt_A_Livrer = VAL(L$)
IF L$ = "/" THEN EXIT
Art(Art_Num) = Art(Art_Num) + Qt_A_Livrer : NEXT
```

EXIT ferme toutes les boucles en cours et saute à la fin de celles-ci. Mais comment faire si le programme doit continuer ailleurs? Dans ce cas bien particulier, on utilise EXIT TO, une combinaison entre EXIT et GOTO. Les boucles sont alors terminées et le programme repart à l'endroit indiqué par EXIT TO.

Exemple :

```
INPUT "Entrez le nom d'un animal ";Mot$
FOR I=1 TO 4 : READ All$, Fr$
    IF UPPER$(Fr$)= UPPER$(Mot$) THEN EXIT TO Trouver
NEXT
PRINT "Mon immense base de connaissances ne me permets"
PRINT "malheureusement pas de traduire." : RUN
-Trouver : PRINT Mot$;" est un ";All$;" en Allemand." : RUN
DATA"Hund","chien","Katze","chat","Pferd","cheval","Kuh","vache"
```

TABLEAUX ET TRIS

CLEAR

Syntaxe : CLEAR
Syntaxe : CLEAR <GEMDOS> [, <Pile>]

CLEAR est une instruction qui :

- Efface le contenu des variables.
- Débranche les ON ERROR...GOTO.
- Débranche les fonctions 'multitâches' (ON TIMER...GOSUB).
- Ferme tous les fichiers ouverts.
- Vérifie la régularité des boucles.

En OMIKRON, CLEAR a une autre fonction, celle de réserver de la mémoire. On réserve de la mémoire pour GEMDOS et pour la pile processeur.

GEMDOS : cette mémoire est utilisée en OMIKRON par MEMORY. De plus, les fichier ressources prennent sur cet emplacement la mémoire dont ils ont besoin.

PILE PROCESSEUR : c'est la mémoire dont se sert l'OMIKRON pour les tris et pour tous les appels de procédures, de sous–programmes, ...

Il est possible de vérifier que la place réservée pour GEMDOS grâce à CLEAR est effectivement utilisable par MEMORY. Il suffit de taper :

```
CLEAR 10000
PRINT MEMORY( -1)
10000
OK
```

Comme vous pouvez le voir, l'emplacement disponible pour les MEMORY correspond à la taille demandée avec CLEAR.

DIM

Syntaxe : DIM <Tableau>(<Valeur>[,<Valeur]>..) [,<Tableau>...]

Dimensionnement d'un ou de plusieurs tableaux.

DIM Echiquier(7,7). Echiquier est un tableau à deux dimensions. Echiquier contient 8*8 cases et l'index peut varier entre 0 et 7. En effet, l'index d'un tableau est toujours 0 pour le premier élément.

Si on utilise un tableau non dimensionné, l'OMIKRON exécute automatiquement un DIM de dimension 10. Par exemple, ARTICLE(3,1,I)=12 force le dimensionnement du tableau ARTICLE à (10,10,10).

Il n'y a pas un nombre limité de dimensions (si ce n'est la longueur d'une ligne, soit 255 caractères).

En ce qui concerne le nombre d'éléments, voici les contraintes :

- Tableaux à une dimension : illimité
- Tableaux à plusieurs dimensions : 65535 éléments pour la première dimension et 65535 pour l'ensemble des autres dimensions.

DIM TABLEAU(1,999,99) est interdit, car la seconde et la troisième dimension multipliées ensemble donnent 100000, ce qui dépasse 65535. Par contre, la ligne suivante sera acceptée : DIM TABLEAU(999,99,1).

Mémoire nécessaire pour un tableau :

Tableau ENTIERS :

Booléens	(%F)	1 octet pour 8 éléments
Octets	(%B)	1 octet par élément (8 bits)
Mots	(%)	2 octets par élément (16 bits)
Longs	(%L)	4 octets par élément (32 bits)

Tableau REELS :

simple	(!)	6 octets par élément
double	(#)	10 octets par élément

Tableau de chaînes :

Chaîne	($)	6 octets par élément * longueur + 10.

N.B. : Contrairement à la majorité des basics, l'OMIKRON permet le redimensionnement des tableaux en cours de programme, sans perte du contenu. Nous vous conseillons de faire particulièrement attention lors du redimensionnement de tableaux à plusieurs dimensions. En effet, il est très facile d'inverser ou de se tromper dans les indexs.

```
DIM A(10,10)
DIM A(10,20) : REM Pas de modification
```

Mais :

```
DIM A(20,10) : REM Les indexs sont échangés, le contenu est
inversé.
```

Vous pouvez augmenter le nombre d'éléments, le diminuer, rajouter des dimensions : tout est possible sans limitation.

SWAP

Syntaxe : SWAP <Variable1>, <Variable2>

SWAP échange les valeurs de deux variables.

```
A=3 : B=5
SWAP A , B
```

Après le SWAP, A contient 5 et B contient 3. Utilisez SWAP chaque fois que vous le pouvez, vous gagnerez beaucoup de place et surtout du temps à l'exécution, en particulier pour les chaînes.

ATTENTION : SWAP ne peut fonctionner que si les variables à échanger appartiennent exactement au même type, c'est–à–dire :

```
SWAP X%,Y% : REM  OK
SWAP A$(I),A$(I+1) : REM  OK
SWAP A!,B% : REM impossible, réel et entier.
SWAP A,B% : REM impossible entier long et entier.
SWAP A$(1),B$ : REM Impossible, tableau de chaînes et chaîne.
```

(En fait, il est possible d'échanger des variables numériques avec les valeurs d'un tableau de variables numériques. Mais les chaînes ont une organisation bien particulière qui empêche l'utilisation de l'algorithme général).

MID$=

Syntaxe : MID$(<Chaine>,<Exp> [,<Exp>]) = Chaine_Remplacement

MID$ dispose d'une utilisation spécifique en OMIKRON. En effet, elle permet de remplacer des caractères à l'intérieur d'une chaîne. Sans cet usage détourné, cette opération nécessite la recopie de la chaîne, des échanges de pointeurs etc... ce qui demande du temps et de la mémoire. MID$ conserve la même adresse en mémoire pour la chaîne.

```
A$ = "Allo UnIKRON" : MID$(A$,6,2) = "OM"
```

A$ contient alors "Allo OMIKRON".

Exemple :

```
REM Remplacer tous les espaces par des soulignés ("_")
    FOR I=1 TO LEN(Texte$)
    IF MID$(Texte$,I,1) = " " THEN MID$ (Texte$, I,1) = "_"
NEXT I
```

Essayez une autre méthode, vous vous apercevrez rapidement de l'importance du gain de temps.

SORT

Syntaxe : SORT [ASC] <Tableau>
Syntaxe : SORT [ASC] <Tableau1> TO <Tableau2>

Le tri est un des premiers problèmes auquel se soient attaqué les informaticiens. Et depuis 40 ans, ils ont conçu tous les algorithmes imaginables. L'OMIKRON utilise deux méthodes de tri très différentes.

La première méthode est très rapide mais demande beaucoup de place en mémoire. Elle n'est pas recommandée pour les tris qui portent sur 10000 éléments ou plus. Il s'agit du QUICK SORT. Le QUICK SORT utilise un algorithme récursif, qui demande 12 octets par récursivité. Dans le cas général, on estime qu'il faut environ 30 récursivités pour trier un tableau de 1000 éléments. Mais dans le pire des cas, il est possible que chaque élément nécessite une récursivité. Comme son nom l'indique, le QUICK SORT est très rapide, puisque sa vitesse ne diminue que d'environ 1,5 fois le nombre d'éléments.

Quand l'OMIKRON s'aperçoit qu'il ne dispose pas de suffisamment de place sur la pile, il change de méthode de tri et emploie un grand classique : le TRI PAR BULLE. Le tri par bulle a l'énorme avantage de ne rien consommer en mémoire. En vertu de quoi, il s'agit de l'une des pires méthodes de tri en temps d'exécution. Le temps d'exécution du tri par bulle augmente selon le carré du nombre d'éléments.

L'OMIKRON peut trier tous les types de tableaux mais il n'est pas possible de trier des tableaux possédant plus d'une dimension.

Exemple :

```
DIM A(100)
FOR I = 0 TO 100
    A(I) = RND(10000)
NEXT I
SORT A(100 )
FOR I = 0 TO 100
    PRINT A(I) ,
NEXT I
```

La valeur indiquée après SORT ne doit pas dépasser le nombre maximum d'éléments du tableau. Si on indique une valeur n inférieure, seuls les n premiers éléments du tableau seront triés (de l'élément 0 à l'élément n–1).

L'exemple suivant montre comment trier plusieurs tableaux selon le classement d'un seul tableau.

```
DIM A(100), B(100), Index(100)
FOR I = 0 TO 100
    A(I) = RND(10000) : Index(I)=I
NEXT  I
FOR I = 0 TO 100
B(I) = A(I) : REM Copier le tableau B() dans A()
NEXT  I
SORT B(0) TO Index(0) :      REM Index() est un tableau d'indexs
                             REM pour le tableau A() non classé
FOR I = 0 TO 100
    PRINT A(Index(I)),
NEXT  I
```

IMPORTANT : le tri s'effectue sans tenir compte des majuscules/ minuscules pour éviter les erreurs de classement du type :

```
Zilog
souris
```

Il est cependant possible de trier par le code ASCII en indiquant ASC :

```
SORT ASC < Tableau_Chaines >
SORT ASC < Tableau_Chaines > TO < tableau2 >
```

READ, DATA, RESTORE

Un programme est conçu, quelque soit son usage, pour traiter des données. Il existe trois méthodes pour stocker les données avant et après traitement.

1°) Entrer les données à chaque exécution du programme par l'intermédiaire du clavier. Cette méthode est inexploitable, sauf pour les tests et la mise au point du programme.

2°) Charger et sauvegarder sur disque. C'est une méthode compliquée, qui introduit de nombreuses erreurs et qui se révéle mal adaptée pour des petites quantités de données. De plus, la nécessité de charger un fichier sur disque est pénalisante au niveau du temps d'exécution.

3°) On peut aussi se servir de DATA, READ et RESTORE. Ce sont ces instructions que nous allons aborder ici.

READ

Syntaxe : READ <Variable> [,<Variable>]...

READ lit un ou plusieurs éléments stockés dans une ligne débutant par l'instruction DATA. Les éléments (les données) sont lus par READ les uns après les autres, en séquence.

Après que vous ayez tapé 'RUN', l'OMIKRON lit le premier élément de la première ligne de DATA. Il se déplace ensuite sur l'élément suivant de la ligne ou, si cette ligne est complètement lue, sur la ligne de DATA suivante.

A chaque 'RUN', le pointeur employé par READ est replacé automatiquement sur le premier élément de la première ligne de DATA. Il est possible de déplacer le pointeur où on le désire grâce à l'instruction RESTORE.

Exemple :

```
READ X,Y : DRAW X,Y : REM point de départ
FOR I=1 TO 8
READ X,Y
DRAW TO X,Y:REM trace une ligne du point courant au point X,Y
NEXT
DATA 50,150,100,150,100,100,75,50,50,100
DATA 100,100,50,150,50,100,100,150
```

Dans ce programme, les coordonnées apparaissent à la fin sous la forme d'une liste. Ces coordonnées, une fois lues par READ et affichés grâce à l'instruction DRAW TO, forment une figure connue pour son attrait sur les jeunes enfants.

DATA

Syntaxe : DATA <Expression> [,<Expression>]...

Les éléments contenus dans une ligne commençant par un DATA peuvent être :

- Des expressions numériques.
- Des expressions alphanumériques.
- Des variables numériques.
- Des variables alphanumériques.
- Des tableaux.

Il faut bien distinguer ces différents types.

```
DATA 125,139+15,SQR(132),VARPTR(A)  'Expressions numériques
DATA "jeté","O"*15,CHR$(27)+"f",MID$(A$,2,1)
REM Expressions alphanumériques
DATA No,Radical%,PI,Nombre_D_Or# ' Variables numériques
DATA Omikron$, Langage$ ' Variable alphanumériques
DATA Abscisse%(O), Eglise$(5) ' Tableaux
```

Il s'agit là d'une différence parmi les plus importante avec le Microsoft basic, qui n'accepte en tant que DATA que des constantes. Vous pouvez, en OMIKRON, utiliser des tableaux et des variables.

RESTORE

Syntaxe : RESTORE [<Etiquette>]

L'ordre RESTORE sans paramètre place le pointeur sur la première ligne de DATA. On peut aussi déplacer le pointeur sur une ligne déterminée de DATA en indiquant une étiquette ou un numéro de ligne.

Exemple :

```
RESTORE 1000 : REM Place le pointeur sur le premier élément de
la ligne 1000
RESTORE Resultat : REM place le pointeur sur la ligne où se
trouve l'étiquette Resultat.
```

Comme ON...GOTO, ON...GOSUB, on dispose aussi de ON...RESTORE.

ON ... RESTORE

Syntaxe : ON <Expression_Numerique> RESTORE <Etiq1><,Etiq2>...

ON ... RESTORE fonctionne comme ON ... GOTO mais replace le pointeur des datas en fonction des étiquettes correspondantes.

Exemple :

```
INPUT "Mois : "; Mois
INPUT "1=Français, 2=Anglais : "; Langue
ON Langue RESTORE Mois_En_Francais , Mois_En_Anglais
FOR I = 1 TO Mois
    READ Mois$
NEXT  I
PRINT Mois$
-Mois_En_Francais
DATA "Janvier", "Février", "Mars", "Avril", "Mai", "Juin", "Juillet"
DATA "Aout'", "Septembre", "Octobre", "Novembre", "Décembre"
-Mois_En_Anglais
DATA "January", "February", "March", "April", "May", "June", "July"
DATA "August", "September", "October", "November", "December"
```

GESTION DU DISQUE

Ce chapitre concerne les instructions qui permettent de sauver, charger, renommer, ... en bref de gérer le disque en OMIKRON.

Tout d'abord, un rappel de quatre fonctions que nous avons déjà vu avec l'éditeur standard :

SAVE	Sauvegarde d'un programme sur disque.
SAVE,A	Sauvegarde d'un programme sur disque en ASCII.
LOAD	Charger un programme depuis le disque.
MERGE	Ajouter au programme en mémoire un programme sous forme ASCII.

Passons au reste.

RUN

*Syntaxe : **RUN "NOM"***

Charge le programme indiqué depuis le disque et l'exécute automatiquement. RUN "NOM" correspond aux deux instructions :

```
LOAD "NOM "
RUN
```

CHAIN

*Syntaxe : **CHAIN [MERGE] "Nom_Fichier" [,<Deb_Ligne>]***

L'OMIKRON autorise le chaînage de programmes avec CHAIN. Vous pouvez créer de grandes applications, chacune des tâches constituant un programme. Un programme principal charge les programmes avec CHAIN, ce qui permet le passage d'informations d'un programme à l'autre. CHAIN demande une certaine rigueur pour fonctionner correctement. En particulier, vous devez utiliser les mêmes noms de variables dans les programmes, du moins si vous désirez passer des paramètres.

CHAIN fonctionne comme RUN, c'est à dire que l'ancien programme est effacé de la mémoire. Mais, contrairement à RUN, il est possible de conserver les valeurs de certaines variables. Les variables dont on désire conserver les valeurs doivent être mentionnées après l'instruction COMMON. Il est possible d'indiquer un numéro de ligne pour obliger le programme appelé à démarrer en une ligne différente de la première ligne. CHAIN MERGE n'efface pas le programme en mémoire. Il se contente de charger à la suite un programme en ASCII. Ce programme en ASCII est ensuite exécuté.

COMMON

Syntaxe: COMMON <Liste_ Variable>,...

COMMON permet de passer des paramètres d'un programme à un autre lors de l'utilisation de CHAIN. COMMON, pour être actif, doit se trouver sur la première ligne exécutée (il s'agit donc de la ligne indiquée par CHAIN). A la différence du COMMON Microsoft Basic, COMMON en OMIKRON doit contenir dans les deux programmes les mêmes noms de variables dans le même ordre.

Exemple :

```
REM PROGRAMME 1
COMMON A
A = A+1 : PRINT A
CHAIN " PROG2 .PRG "

REM PROGRAMME 2
COMMON A
A = A+1 : PRINT A
CHAIN " PROG1.PRG "
```

Nouveau: Version 3.0

EXEC

Syntaxe : EXEC <Nom_ Programme>[,<Chaine_ Commande>]

EXEC permet le passage de paramètres au format GEM. Vous pouvez désormais exécuter n'importe quel programme fonctionnant sur ATARI avec EXEC en lui passant des paramètres (comme par exemple le nom d'un fichier à charger).

Exemple :

```
EXEC "C:\ 2ND_WORD.PRG", "C:\REMERCIM .DOC"
```

Charge et exécute le programme 2ND_WORD, qui charge automatiquement le fichier REMERCIM.DOC. Quand le programme appelé par EXEC se termine, la main est rendue à l'instruction suivant 'EXEC'.

Limites: Certains programmes ne respectent pas les formats établis par DIGITAL RESEARCH en ce qui concerne le retour et le passage de paramètres. Comme EXEC respecte strictement GEM, il est possible qu'un programme ne retourne pas au programme appelant mais au bureau (c'est le cas de 'REDACTEUR').

FILES

Syntaxe: FILES <Chemin_Acces>

FILES affiche à l'écran le contenu du disque dont le chemin est spécifié. Il est possible d'effectuer une sélection au niveau du nom de fichier.

Exemples :

```
FILES "*.* "
```

Affiche le contenu du chemin actif. Aucune sélection, tous les fichiers apparaissent. Remarquez le joker '*' qui remplace tous les caractères du nom (soit 8 caractères) et tous les caractères du suffixe (soit 3 caractères). Il est indispensable d'indiquer le '.' pour séparer le nom du suffixe. Si aucun '.' n'apparaît, seuls les fichiers sans aucun suffixe seront affichés.

```
FILES "A:\*.BAS "
```

Tous les fichiers contenu sur la disquette du lecteur A et dont le suffixe est BAS seront affichés.

```
FILES "C:\AUTO \*.TOS "
```

Affiche tous les fichiers dont le suffixe est TOS qui se trouvent dans le dossier AUTO sur le disque C.

Remarques :

- Le '\' se trouve à gauche de la touche [Return]. Il suffit de taper [ALT]–[\] pour l'obtenir.
- Vous n'êtes absolument pas obligé de taper un espace entre FILES et le chemin d'accès. Cette règle est vraie pour la majorité des instructions de l'OMIKRON.

NAME...AS

Syntaxe : NAME <Ancien_Nom> AS <Nouveau_Nom>

NAME...AS renomme un fichier.

Exemples :

```
NAME "PERDU.DOC " AS  "GAGNE.DOC "
NAME "A:\ IMPRIM\TRANS.PRG " AS "A:\IMPRIM\TRANS. TOS "
```

LIMITES :

- Le nouveau nom est repris tel quel par l'OMIKRON. C'est à dire que le suffixe n'est pas rajouté et que vous devez l'indiquer vous–même dans le nouveau nom.
- Le chemin d'accès doit être entré pour les deux noms.

KILL

Syntaxe : KILL <Chemin_Acces>

KILL supprime un ou des fichiers. Exemples :

```
KILL "TRYING.BAS "
```

Suppression du fichier TRYING.BAS qui se trouve sur le chemin d'accès actif.

```
KILL "A:\*.* "
```

Suppression de la totalité des fichiers de la disquette.

```
KILL "C:\*.ACC "
```

Suppression de la totalité des accessoires du disque C.

COPY

Syntaxe : COPY < Chemin_Acces > TO < Chemin_Acces >

Copie de fichiers.

Exemples :

```
COPY "A:\A_LIRE.DOC " TO "C:\A_LIRE.DOC "
```

Copie le fichier A_LIRE qui se trouve sur la disquette du lecteur A sur le disque C.

```
COPY "A:\*.BAS " TO  "E:\*.BAS "
```

Copie tous les fichiers portant l'extension BAS et situés sur le disque A sur le disque E.

BACKUP

Syntaxe : BACKUP < Chemin_Acces >

BACKUP sauvegarde le fichier en créant une copie de sauvegarde de la version précédente dont le suffixe est BAK et dont le nom est le même que celui du fichier sauvé.

Exemples :

```
BACKUP "CASH .BAS "
```

L'ancien fichier CASH.BAS est renommé CASH.BAK et le nouveau fichier se nomme CASH.BAS

```
BACKUP "*.DAT "
```

Tous les fichiers portant l'extension DAT sont sauvegardés ; les anciens DAT sont renommés BAK.

MKDIR

Syntaxe : MKDIR <Nom_Dossier>

Création d'un dossier sur le disque. Exemple:

```
MKDIR "C:\ OMIKRON "
```

Un dossier OMIKRON est créé sur le disque C.

RMDIR

Syntaxe : RMDIR <Nom_Dossier>

Efface un dossier du disque, à la condition expresse que le dossier soit vide. Exemple :

```
RMDIR "C:\ OMIKRON "
```

CHDIR

Syntaxe : CHDIR <Chemin_Acces>

Changement du chemin d'accès actif. Le chemin d'accès actif est le chemin d'accès auquel toutes les instructions opérant sur les fichiers se rapportent par défaut. Exemple :

```
CHDIR "C:\ "
```

Le chemin d'accès actif est C:\.

LES FICHIERS

L'OMIKRON dispose d'un grand nombre d'instructions permettant la gestion des fichiers. Il existe deux types de fichiers :

Les fichiers séquentiels :

Un fichier séquentiel conserve les données à la queue leu-leu.

Avantages : Simple à comprendre et à programmer, longueur des champs variable.

Inconvénients : Il est impossible de lire directement un enregistrement donné. On est obligé de lire tout le fichier.

Conclusion : Les fichiers séquentiels sont plutôt adaptés à des petites quantités de données et ils sont faciles à utiliser.

Les fichiers relatifs (ou directs) :

Un fichier relatif réserve pour chaque champs une longueur fixe. Pour accéder à un enregistrement, il suffit de donner son numéro et la lecture est immédiatement effectuée.

Avantages : Accès immédiat à un enregistrement, d'où une bien meilleure vitesse.

Inconvénients : Nécessite plus de mémoire sur le disque, car pour chaque champs la taille du plus grand champs est réservée. Gestion un peu plus compliquée.

Conclusion : Les fichiers relatifs sont conçus pour les fichiers de grande taille ou pour des applications dans lesquelles on doit retrouver rapidement un enregistrement.

FICHIERS SEQUENTIELS

Avant toute chose, il faut ouvrir le fichier.

OPEN

Syntaxe : OPEN <Type>, <Canal> [,<Nom_Fichier>] [,<Long_Maxi>]

Il existe de nombreux types de fichiers. Nous allons rapidement le passser en revue dans le tableau suivant :

"O"	OUTPUT	Fichier séquentiel en création
"I"	INPUT	Fichier séquentiel en lecture
"A"	APPEND	Fichier séquentiel en ajout
"R"	RELATIVE	Fichier relatif
"F"	FILES	Contenu du directory
"U"		Fichier relatif à taille variable
"P"	PRINTER	Vers l'imprimante
"C"	CONSOLE	Vers l'écran
"K"	KEYBOARD	Vers le clavier
"V"	SERIE	Vers la RS-232
"M"	MIDI	Vers la MIDI

- Les fichiers séquentiels utilisent les types I, O et A.
- Il est possible d'utiliser les numéros de canaux de 1 à 16.
- On peut ouvrir un fichier sur n'importe quel disque, il suffit de l'indiquer avant le nom.

Exemple :

```
OPEN "O", 1,"C:\AUTO\NUL.NUL "
```

La longueur maximale n'a de signification qu'avec les fichiers relatifs. Dans ce cas, la longueur indique la taille totale de tous les champs dans un enregistrement.

Voici un programme d'exemple :

```
OPEN "o", 1,"NOM.DAT "
REPEAT
    INPUT Nom$
    PRINT #1, Nom$
UNTIL Nom$ = "" : CLOSE 1
```

Ce qui nous amène à PRINT# :

PRINT#

Syntaxe : PRINT #<Numero_Canal>...

PRINT# fonctionne pratiquement comme PRINT. Il faut indiquer le canal de sortie prévu. Le canal doit correspondre à un numéro de fichier ouvert.

Dans un PRINT#, il est interdit d'utiliser le caractère ",", qui correspond au séparateur de champs. De plus, nous ne conseillons vraiment pas les caractères spéciaux du style ESC et compagnie. Souvent, le fichier sera rendu illisible par ce genre de chose. Si vous avez besoin de caractères spéciaux, utilisez WRITE#.

WRITE#

Syntaxe : WRITE #<Numero_Canal>...

Par rapport à PRINT#, WRITE# comporte certaines modifications. Avec WRITE#, les chaînes sont sauvées en étant entourées de "".

Exemple :

```
A$ = "AAA": WRITE #1, A$
"AAA"
OK
```

Le séparateur de données "," est sauvegardé avec les données.

Exemple :

```
A$ = "ABC" : B$ = "123" : WRITE #1,A$,B$
"ABC", "123"
OK
```

Il est nettement plus pratique d'utiliser WRITE#, car lors de la relecture du fichier, un seul INPUT# reconnaît automatiquement plusieurs champs séparés par ','. Avec PRINT#, le séparateur est la fin de ligne (CR, code ASCII 13), ce qui implique l'utilisation de plusieurs INPUT# pour relire le fichier.

CLOSE

Syntaxe : CLOSE [<Numero_Canal>]...

CLOSE ferme les fichiers spécifiés. Si aucune indication n'est donnée quant au numéro de canal, CLOSE ferme tous les fichiers ouverts.

Il est indispensable de fermer les fichiers en fin de programme. CLOSE ferme proprement les fichiers, après avoir sauvegardé les données qui se trouvaient dans le buffer.

INPUT*

Syntaxe : INPUT # <Numero_Canal>, <Variable>...

INPUT# fonctionne comme le INPUT standard. Il lit une chaîne de caractères sur un fichier. Les séparateurs sont la fin de ligne (code ASCII 13) et la virgule. Il est tout à fait possible de lire plusieurs variables avec un seul INPUT : il faut indiquer une suite de variables réceptrices.

LINE INPUT*

Syntaxe : LINE INPUT #<Numero_Canal>, <Variable>

Cet ordre correspond pratiquement à l'ordre INPUT#.

Limites :

- La saisie de plusieurs variables dans un seul LINE INPUT est interdite. Il faut un LINE INPUT pour chaque variable.
- De par sa définition, LINE INPUT refuse les variables numériques. Par contre, LINE INPUT autorise les virgules.

EOF

Syntaxe : EOF(<Numero_Canal>)

EOF signifie End Of File, fin de fichier. EOF permet de savoir si la fin du fichier spécifié est atteinte. Le numéro de canal est le numéro de fichier logique d'un fichier ouvert précédemment. Il faut bien séparer les résultats selon le type de fichiers.

Les fichiers séquentiels retournent la valeur vraie (–1) quand la fin de fichier est atteinte. Voici un programme de démonstration :

```
OPEN "O", 1,"TEST"
WHILE NOT EOF(1)
    PRINT INPUT$(1, 1)
WEND
```

Avec les fichiers relatifs, EOF retourne –1 uniquement après avoir relu le dernier enregistrement.

Quand vous utilisez le mode "F" (défini dans OPEN), vous pouvez vérifier si le directory contient quelque chose en tapant ce qui suit :

```
OPEN "F",4 ,...
IF EOF(4) THEN PRINT "Aucun fichier sur le disque."
```

Bien entendu, vous n'êtes pas obligé d'utiliser le canal 4.

LOF

Syntaxe : A = LOF(<Numero_Canal>)

LOF pour Lenght Of File, longueur du fichier. LOF retourne la taille du fichier spécifié. Dans le cas d'un fichier séquentiel ouvert en écriture, LOF retourne seulement la longueur des données déjà inscrites sur disque, sans prendre en compte les données encore dans le buffer. Pour les fichiers en accès direct, LOF retourne le nombre d'enregistrements du fichier.

INPUT$

Syntaxe : INPUT$ (<Nombre_caracteres>, <Numero_Canal>)

INPUT$ lit le nombre de caractères spécifié sur le canal dont le numéro a été entré. Voici un petit programme qui permet de lister le contenu d'un fichier.

```
DEF PROC Dump_Fich( Nom$)
    OPEN "I ",1, Nom$
    WHILE NOT EOF(1)
      PRINT INPUT$ (1,1)
    WEND
    RETURN
```

Il suffit d'appeler la procédure en indiquant le nom du fichier à lire.

FICHIERS RELATIFS (OU DIRECTS)

Les fichiers relatifs ont une longueur d'enregistrement fixée à la création du fichier. Chaque champs dispose de la taille du plus grand champs. Par exemple, tous les champs concernant le nom d'une personne auront 35 caractères de long, même si la plupart des noms comportent moins de 35 caractères.

On accède aux fichiers relatifs grâce à un index. Bien entendu, il n'y a qu'un enregistrement par index et qu'un index par enregistrement (comment s'y retrouver autrement ?).

La taille et l'organisation d'un fichier relatif est fixée par FIELD.

FIELD

Syntaxe : FIELD <Numero_Canal>, <Longueur> [AS <Variable>], ...

FIELD détermine la longueur attribuée à chaque variable pour un fichier relatif donné. La limite par enregistrement est d'un peu plus de 32000 caractères, ce qui est plus que suffisant. On ne doit pas utiliser les variables tampons spécifiées dans FIELD pour autre chose que les lectures/écritures sur un fichiers relatif. En effet, vous n'avez pas affaire à des variables standards, même si elles en ont l'aspect. Ces variables sont directement rattachées au tampon servant aux transferts vers et depuis le disque et leur longueur est fixe. Si cela n'apporte pas de modification dans le sens variable tampon vers variable normale, il n'en va pas de même pour l'inverse. Il est alors indispensable d'utiliser MID$, LSET et RSET.

LSET

Syntaxe : LSET <Variable_Tampon> = <Chaine>

La chaîne est transférée dans la variable tampon. Si la chaîne est trop grande, les caractères qui dépassent à droite sont supprimés. Si la chaîne est trop courte, des espaces remplacent les caractères manquants à droite.

Exemples :

```
FIELD 1, 14 AS A$
LSET A$ = "123456789ABCDEF"
```

A$ contiendra "123456789ABCDE", le caractère situé à l'extrême droite est supprimé (A$ fait 14 caractères de long)

```
LSET A$ = "123456"
```

Correspond à :

```
A$ = "123456" + " "*8
```

RSET

Syntaxe : RSET <Variable_Tampon>=<Chaine>

Cette fois, la chaîne est formatée à droite. Si la chaîne est trop grande, RSET supprime les caractères situés à gauche, si la chaîne est trop courte, RSET ajoute des blancs à gauche.

Exemples :

```
FIELD 1, 14 AS A$
RSET A$ = "123456789ABCDEF"
```

A$ contiendra "23456789ABCDEF", le caractère situé à l'extrême gauche est supprimé (A$ fait 14 caractères de long)

```
RSET A$ = "123456"
```

Correspond à :

```
A$ = " "*8 +"123456"
```

Ecrivons un mini-programme pour gérer nos adresses :

```
CLS
OPEN "R", 1, "ADRESSE.DAT",115
FIELD 1,30 AS Nom$, 25 AS Prenom$, 30 AS Adr$, 5 AS Cod_Pos$,
25 AS Ville$
REM boucle principale
REPEAT
    CLS : INPUT "<L>ire ou <S>aisir ?";Rep1$ USING "U+L+S",,1
    IF Rep1$ = "S" THEN Saisie : Ecriture ELSE Lecture
    INPUT "On continue (Oui ou Non) ?";Rep$ USING "U+O+N",,1
UNTIL Rep$ = "N"
CLS : PRINT "Terminé pour cette fois..."
```

```
CLOSE : END
DEF PROC Saisie
    CLS :INPUT "NOM: ";A$ : LSET Nom$ = A$
    INPUT "PRENOM: ";A$ : LSET Prenom$ = A$
    INPUT "ADRESSE: ";A$ : LSET Adr$ = A$
    INPUT "CODE POSTAL: ";A$ : LSET Cod_Pos$ = A$
    INPUT "VILLE: ";A$ : LSET Ville$ = A$
RETURN
DEF PROC Ecriture
    INPUT "Quel numéro d'enregistrement pour cette
    adresse?";Num_Enr : PUT 1, Num_Enr
RETURN
DEF PROC Lecture
    CLS :INPUT "Quel enregistrement dois-je lire ?";Num_Enr
    GET 1,Num_Enr
    PRINT Nom$, Prenom$ :PRINT Adr$, Cod_Pos$,Ville$
RETURN
```

Ce programme fonctionne très bien, à la condition qu'aucune virgule ne soit entrée par l'utilisateur. Vous pouvez modifier le programme en utilisant des INPUT USING et interdire ainsi l'entrée des virgules.

Plusieurs instructions nouvelles sont intégrées à ce programme et nous allons les détailler dans ce qui suit.

PUT

Syntaxe : PUT <Numero_Canal>, <Numero_Enregistrement>

PUT écrit dans le fichier à l'enregistrement spécifié les données contenues dans les variables tampons.

GET

Syntaxe : GET <Numero_Canal>, <Numero_Enregistrement>

GET lit sur le fichier l'enregistrement spécifié. Les données sont inscrites dans les variables tampons.

Nous venons d'aborder les chaînes de caractères, qui, ont l'a vu, ne posent pas de problèmes majeurs. Mais les nombres sont plus délicats à manipuler. En effet, les nombres ne font jamais la même

taille. C'est pourquoi il faut utiliser les fonctions MKxx$ pour convertir les nombres en des chaînes de caractères de longueur fixe. A la relecture, il faudra employer les fonctions CVxx pour retrouver ces nombres. Voici un exemple simple de ce procédé :

```
REM Ouverture du fichier, taille de 4 caractères (Entier long)
OPEN "r",1,"Nombres ",4 : FIELD 1,4 AS Buffer$
REM Ecriture de dix nombres...
FOR I = 1 TO 10 : LSET Buffer$ = MKIL$(I^2) : PUT 1,I : NEXT I
CLOSE 1
OPEN "r",1,"nombres ",4 : FIELD 1,4 AS Buffer$
REM Lecture des dix nombres...
FOR I = 1 TO 10 : GET 1,I : PRINT CVIL(Buffer$) : NEXT I
CLOSE 1
```

SEEK

Syntaxe : SEEK <Numero_Canal>, <Deplace>[, <Mode>]

SEEK permet de positionner le pointeur d'enregistrement où on le désire. Celui-ci correspond au numéro d'enregistrement d'un fichier (qu'il soit en lecture ou en écriture). Le premier paramètre indique le numéro de canal, le second le déplacement en enregistrements. SEEK accepte 3 modes : Avec 0 pour mode, le déplacement s'effectue depuis le départ du fichier. Il s'agit du mode par défaut.

La valeur 1 signifie que le déplacement s'effectue en prenant comme base l'enregistrement actuel (mode relatif).

Enfin, 2 donne un déplacement depuis la fin du fichier.

Limite : SEEK ne fonctionne qu'avec les fichiers relatifs.

LOC

Syntaxe : LOC(<Numero_Canal>)

LOC n'est utilisable qu'avec les fichiers en accès direct. LOC retourne le numéro du dernier enregistrement lu ou écrit.

Exemple :

```
? LOC (1)
```

LE TRAITEMENT DES ERREURS

ON ERROR...GOTO

Syntaxe : ON ERROR GOTO <Etiquette>

Quand un utilisateur novice se retrouve devant un message aussi explicite que 'accès hors limite', il lui arrive de paniquer. Aussi, l'OMIKRON permet l'interception des erreurs. Dès qu'une erreur est rencontrée, l'OMIKRON exécute une routine de traitement spéciale qui reprend le travail quand c'est possible ou qui termine le programme proprement (fermeture des fichiers, de la station de travail virtuelle...)

Voici un programme qui intercepte un seul type d'erreur. Les autres erreurs renvoient à un message et obligent à quitter le programme.

```
ON ERROR GOTO Traite_Err
PRINT CHR$(27); "f": CLS
-Disque
    OPEN "i",1, "Super.dat"
    WHILE NOT EOF(1)
      PRINT INPUT$(1, 1);
    WEND
-Fin : CLOSE : END
-Erreur : ERROR 16
-Traite_Err
    IF ERR = 53
  THEN MOUSEON
  FORM_ALERT (1,"[2][ATTENTION!!!!|Le fichier Super.DAT|est
  absent du disque.|Changez de disque.][Compris|Autre]",A)
  IF A = 2 THEN MOUSEOFF : RESUME Erreur ELSE MOUSEOFF :
  RESUME Disque
ELSE
  MOUSEON
  FORM_ALERT (1,"[2][ATTENTION!!!|Une erreur vient de se
  produire.|Téléphonez au 26 02 60 44|pour nous prévenir.]
  [Suite] ")
  FORM_ALERT (1,"[2][Indiquez nous ce qui suit:|"+ ERR$ +"|en
  ligne "+ STR$( ERL )+"][Compris]")
  MOUSEOFF
  RESUME Fin : ENDIF
```

Pour annuler le ON ERROR...GOTO, il suffit de faire un CLEAR (ou un RUN ou un NEW), de changer le programme (effacer une ligne par exemple) ou bien de taper ON ERROR GOTO 0.

L'OMIKRON contient plusieurs variables réservées au traitement des erreurs.

ERR ERR est une variable système contenant le numéro de la dernière erreur.

ERL ERL est aussi une variable système, contenant le numéro de la ligne où s'est produite la dernière erreur.

**ERR$** ERR$ est encore une variable système, contenant le message d'erreur (en français!).

De plus, il existe une instruction permettant de redémarrer en cas d'erreur bénigne :

RESUME

Syntaxe : RESUME [<Etiquette>]
Syntaxe : RESUME NEXT

RESUME sort de la routine de traitement d'erreur, de manière à poursuivre le programme interrompu. RESUME correspond un peu au RETURN pour GOSUB.

RESUME sans paramètre tente une nouvelle fois l'exécution de l'instruction ayant entrainé l'erreur.

RESUME NEXT saute à l'instruction qui suit celle où s'est produite l'erreur.

RESUME <Etiquette> saute à la ligne indiquée.

Voici quelques indications importantes concernant les erreurs en OMIKRON :

- Quand une erreur survient, le programme est stoppé.
- Un message d'erreur est affiché si ON ERROR n'est pas actif.
- Le message d'erreur indique à quel type d'erreur on a affaire.

- Si on interrompt une routine d'erreur pendant son exécution (par STOP ou [CTRL]–[C]), le message d'erreur est affiché. De plus, l'information BREAK IN... apparaît sous le message d'erreur.
- Les informations BREAK IN... apparaissent à chaque interruption, mais le message d'erreur n'est affiché qu'une seule fois.
- RESUME ne remet pas la pile à zéro : les boucles, les appels de procédures, ... sont intacts. Pour effacer la pile, utilisez EXIT.

Pour tester vos routines d'erreur, il existe une instruction spécifique :

ERROR

Syntaxe : ERROR <Numero_Erreur>

ERROR force l'OMIKRON à simuler une erreur correspondant au numéro indiqué. Grâce à ERROR, le test des routines de traitement des erreurs est grandement facilité.

MULTITACHE CONDITIONNEL

Qu'est ce qu'un traitement multitâche ?

Une tâche correspond grossièrement à l'exécution d'un programme. Les systèmes UNIX, VMS etc... autorisent l'exécution concurrente de plusieurs programmes. En fait, les programmes ne sont pas exécutés réellement au même moment. Le système passe d'un programme à l'autre très rapidement, ce qui donne à l'utilisateur l'impression que tous les programmes fonctionnent simultanément. Ne nous trompons pas : l'OMIKRON ne permet pas la création de programmes multitâches pour la bonne et simple raison que le système de l'ATARI ST (TOS) n'est pas multitâche. Par contre, l'OMIKRON offre plusieurs possibilités intéressantes qui ressemblent à du multitâche. Le principe d'application de ces commandes multitâches est toujours le même :

1°) Une condition déterminée (la souris est-elle cliquée?) est vérifiée en permanence par l'OMIKRON. La vérification est très rapide et ne ralentit pratiquement pas le programme principal.

2°) Dès que la condition est remplie, l'OMIKRON saute à un sous-programme. Ce sous-programme réalise ce qu'il a à faire, puis redonne la main au programme principal.

Voici maintenant les commandes de ce multitâche conditionnel :

ON MOUSEBUT GOSUB

Syntaxe : ON MOUSEBUT GOSUB <Etiquette>

Saute au sous-programme dès qu'une des touches de la souris est cliquée.

Exemple :

```
PRINT CHR$(27);"f";: CLS : MOUSEON
ON MOUSEBUT GOSUB Souris
REPEAT UNTIL 0 : REM boucle principale
-Souris
    MOUSEOFF : DRAW TO MOUSEX , MOUSEY : MOUSEON
RETURN
```

ON KEY GOSUB

Syntaxe : ON KEY GOSUB <Etiquette>

Saute au sous-programme dès qu'une touche est enfoncée. Les touches ne sont pas effacées du tampon clavier.

Exemple :

```
PRINT CHR$(27); "f"; : CLS
ON KEY GOSUB Clavier
REPEAT UNTIL 0
'
-Clavier
    PRINT RIGHT$( INKEY$ ,1)
RETURN
```

ON HELP GOSUB

Syntaxe : ON HELP GOSUB <Etiquette>

Saute au sous-programme dès que la touche [Help] est enfoncée. La touche [Help] est effacée du tampon clavier.

Exemple :

```
PRINT CHR$(27); "f"; : FILL STYLE = 2,4
PBOX 0, 0, 640, 400
A = MEMORY(32000)
ON HELP GOSUB Aide
'
REPEAT UNTIL 0
'
-Aide
    BITBLT 0, 0, 640, 400 TO A
    FILL STYLE = 0,1
    OUTLINE OFF
    PBOX 10,10 TO 630,389
    OUTLINE ON
BOX 12,12 TO 628,387: BOX 13,13 TO 627,386
PRINT @(10,10);"Ici se trouve l'écran d'aide..."
REPEAT UNTIL LEN( INKEY$ )<>0
BITBLT A TO 0, 0, 640, 400
RETURN
```

ON TIMER GOSUB

Syntaxe : ON TIMER <Expression_Numerique> GOSUB <Etiquette>

Le sous-programme est exécuté toutes les N secondes. N est exprimé en secondes. La durée minimum est de 1/200 ème de seconde.

Exemple :

```
CLS : PRINT CHR$(27);"f"; : MODE = 1
ON TIMER  2 GOSUB Horloge
'
'
CLIP 0, 0 TO 640, 400
REPEAT
   DRAW RND(640), RND(400) TO RND(640), RND(400)
UNTIL  0
'
'
-Horloge
   T = T+1 : IF T = 15 THEN T= 0 : CLS
PRINT @(T,71); CHR$(27);"p"; TIME$ ;  CHR$(27);"q";
RETURN
```

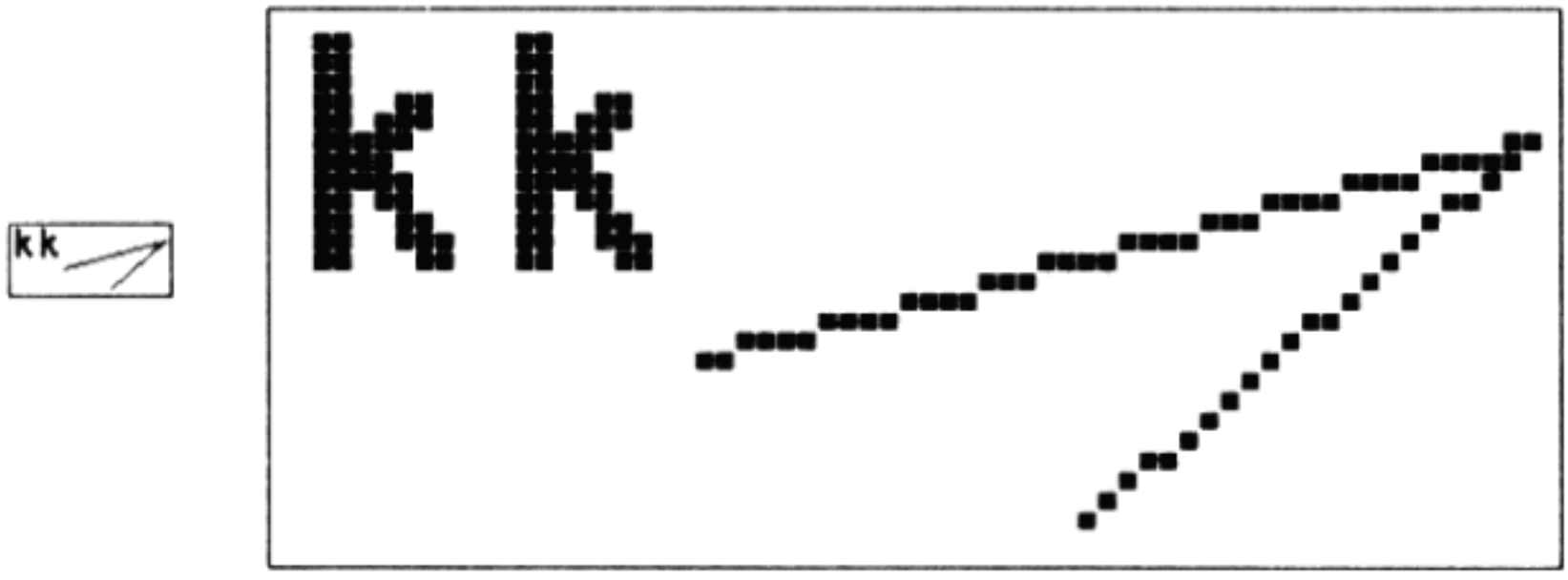
Ce programme affiche l'heure système toutes les deux secondes (l'horloge système est mise à jour toutes les deux secondes). Après l'exécution du sous-programme, le programme reprend son cours.

Pour supprimer les sauts, il suffit de taper :

```
ON TIMER 1 GOSUB 0
```

LE GRAPHISME

L'ATARI ST travaille en BIT-MAP. Ce qui signifie que tout ce qui concerne l'affichage à l'écran se résume à allumer une succession de points sur l'écran. Prenons l'exemple de l'affichage d'un caractère et d'une droite :



L'intérêt du BIT-MAP est d'être totalement paramétrable. On peut afficher ce que l'on veut où on veut. Les styles de caractères sont redéfinissables et les caractères eux-même sont modifiables (pour vous en persuader, regardez la police de caractères employée dans le super-éditeur : ce n'est pas la 'font' standard, mais notre police!)

Et toutes les fonctions graphiques agissent de la sorte.

Jusqu'ici, deux inconvénients majeurs interdisaient une grande diffusion du BIT-MAP :

- Les écrans BIT-MAP sont très gourmands en mémoire.
- L'ordinateur doit disposer d'une bonne réserve de puissance.

L'ATARI ST résout brillamment ces deux problèmes en proposant jusqu'à 4 Mo de mémoire vive et en offrant un microprocesseur 68000.

Les fonctions graphiques de base (ligne, cercle, ellipse, point...) de l'ATARI ST sont rassemblées dans le VDI. Vous pouvez donc utiliser ces fonctions en ouvrant une station de travail virtuelle, puis en appelant les fonctions VDI. Mais c'est très lourd à manoeuvrer, les erreurs de frappe sont fréquentes en raison de la syntaxe bizarre du VDI et la mise au point est lente. Aussi, l'OMIKRON permet d'utiliser les fonctions les plus courantes directement.

L'ATARI ST dispose de 3 résolutions graphiques :

- Haute résolution : 640*400 points noir/blanc (monochrome)
- Moyenne résolution : 640*200 points en 4 couleurs
- Basse résolution : 320*200 points en 16 couleurs

Le point 0,0 est toujours situé dans le coin supérieur gauche de l'écran et le point maximum se trouve dans le coin inférieur droit.

IMPORTANT : Avant d'utiliser une fonction graphique (DRAW, CIRCLE...), il faut toujours éteindre la souris. Essayez l'exemple suivant pour comprendre pourquoi :

```
REPEAT
CLS
PRINT CHR$(27); "f";"[CTRL]-[C] pour sortir."
INPUT "Routine correcte? <O>ui/<N>on ";Oui$ USING "U+O+N",,1
IF Oui$="O" THEN Correcte ELSE Beurk
UNTIL Je_Bats_Artur_Aux_Echecs
'
'
DEF PROC Beurk
    PRINT @(1,0); "Etrange disparition des points... "
    MOUSEON
    WHILE INKEY$ = ""
      IF MOUSEBUT = 1 THEN DRAW MOUSEX , MOUSEY
    WEND
    MOUSEOFF
RETURN
'
'
DEF PROC Correcte
    PRINT @(1,0);"Tous les points répondent présents. "
    MOUSEON
    WHILE INKEY$ = ""
      IF MOUSEBUT = 1 THEN
      MOUSEOFF : DRAW MOUSEX , MOUSEY :  MOUSEON
      ENDIF
    WEND
    MOUSEOFF
RETURN
```

SCREEN

Syntaxe : SCREEN <Ecran>[,Adresse][,Couleurs]

Le basic OMIKRON gère trois écrans : 0, 1 et 2. L'écran par défaut est 0. L'écran 0 est un peu particulier. En effet, il accepte certaines fonctions VT–52 que les autres écrans refusent. Il permet l'utilisation de HCOPY TEXT et travaille en mode ligne. Ce qui signifie qu'en cas de dépassement de la dernière colonne par une instruction d'affichage comme PRINT, la première colonne de la ligne suivante contient un '|'. Les deux lignes sont reliées et traitées comme une ligne unique. L'OMIKRON autorise le passage d'un des trois écrans à l'autre par l'intruction SCREEN. L'exemple ci–dessous montre comment procéder. Exemple :

```
10 Ecran1 = MEMORY(32000)
20 WHILE  MOUSEBUT = 0
30    SCREEN 1,Ecran1
40    PRINT @(12,10); "écran 1"; : Circle 200, 160, 50 : WAIT .2
50    SCREEN 0
60    PRINT @(20,5); "écran 0"; : BOX 50, 50 to 150, 200 : WAIT .2
70 WEND
```

Il faut absolument réserver de la mémoire pour les écrans 1 et 2. Pour cela, on utilise MEMORY. La taille d'un écran étant 32000 octets, il faut taper 'Variable= MEMORY(32000)' (cf ligne 10).

Si l'ATARI ST est en basse résolution, on peut le passer en moyenne résolution avec :

```
SCREEN N, A, 4
```

Le quatre signifie 4 couleurs disponibles. Il s'agit donc de la moyenne résolution. Le bureau ne tient pas compte de cette modification. Il est par contre impossible de passer de la moyenne à la basse résolution par SCREEN, de même que de passer de la haute résolution en moyenne ou en basse. Le problème vient de VDI qui ne tient aucun compte de ce que peut lui dire l'OMIKRON. Exemple :

```
SCREEN 1,MEMORY (32256),4 : REM 4 couleurs, 640*200 pixels.
```

Si vous n'indiquez aucune couleur, la résolution actuelle est prise par défaut.

Nouveau : Version 3.0

NDC

Syntaxe : NDC [<X>,<Y>,<W>,<H>]

GEM peut manipuler des coordonnées virtuelles. L'écran n'est donc plus adressé directement mais par l'intermédiaire d'un convertisseur. NDC fixe les paramètres du convertisseur, sachant que les coordonnées virtuelles d'un écran sont toujours 0,0 à 32767,32767 et que le point 0,0 est situé en bas à gauche (alors qu'en mode normal, le point 0,0 est situé en haut à gauche). Généralement, on détermine les paramètres de NDC en fonction de la résolution dans laquelle on se trouve (640,400 en haute; 640,200 en moyenne; 320,200 en basse).

Exemple :

```
10 NDC 0,0,640,400 : DRAW 0,0 to 32767,32767
```

DRAW

Syntaxe : DRAW <X>, <Y>
Syntaxe : DRAW TO <X>,<Y>
Syntaxe : DRAW <X>,<Y> TO <X2>,<Y2> [TO <X3>,<Y3>] [TO etc...]

DRAW <X>, <Y> affiche un point aux coordonnées X, Y. DRAW TO <X>, <Y> trace une ligne reliant le dernier point affiché au point de coordonnées <X>, <Y>.

La troisième syntaxe chaîne les droites les unes aux autres. DRAW utilise toujours le dernier mode choisi (MODE=) et la teinte déterminée par LINE COLOR. LINE STYLE permet de choisir un style de ligne, plein ou en pointillés et LINE WIDTH leur épaisseur..

Exemple :

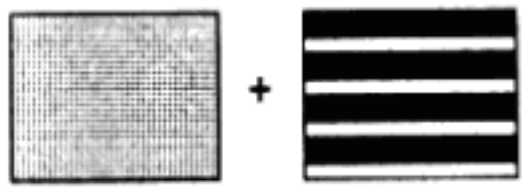
```
REPEAT
    CLS : PRINT CHR$(27); "f"; : X=1
    WHILE X<640 : REM 640 en haute résolution, 320 en basse
    X=X+9
    P = RND(200) : P2 = 400 -P
    DRAW X, P TO X, P2
    LINE STYLE = RND(6)
WEND
PRINT @(0,0); CHR$(7); "Une touche pour recommencer."
WHILE INKEY$ = "" : WEND
UNTIL  Sortie_Des_Cdroms_Par_Atari
```

MODE =

Syntaxe : MODE= <Mode_Dessin>

Il existe quatre modes possibles qui vont de 1 à 4. On se décide pour un mode ou un autre en fonction des effets recherchés. Le mode par défaut est le mode REMPLACE.

MODE =	Fonction
1	Remplace, c'est le mode standard.
2	Transparent.
3	XOR, OU exclusif.
4	Inverse transparent.



PALETTE

Syntaxe : PALETTE [Teinte],[Teinte],...

PALETTE permet de choisir la composition de chaque teinte. Il y a deux teintes en monochrome, 4 en moyenne résolution et 16 teintes en basse. Pour composer une teinte, on sélectionne un mélange des trois couleurs de base (le Rouge, Vert et Bleu –RVB). Chaque couleur de base dispose de 8 valeurs possibles.

Le premier chiffre indique la quantité de rouge, le second la part de vert et le troisième la part de bleu. Chaque couleur de base varie entre 0 et 7. La palette couvre donc bien les 512 teintes possibles (8 rouges * 8 verts * 8 bleus).

Il est largement plus simple de travailler en hexadécimal avec palette, puisqu'on voit immédiatement la quantité de chaque couleur de base introduite. L'hexadécimal est indiqué par '$'.

Exemple :

```
PALETTE $000, $700, $070, $777
REM Palette complète en moyenne résolution
```

Signification :

Teinte 0 = Noir (aucune couleur de base)
Teinte 1 = Rouge (Rouge à fond, vert et bleu à zéro)
Teinte 2 = Vert (Vert à fond, Rouge et bleu à zéro)
Teinte 3 = Blanc (Toutes les couleurs de base à fond)

ATTENTION! TRES IMPORTANT!

PALETTE fixe les teintes Hardware et non pas les teintes VDI. Ce qui signifie que quand vous entrez :

```
PALETTE $777,$000,$700,$007
```

Vous ne fixez pas les valeurs des teintes 0,1,2 et 3, mais les couleurs 0,15,1 et 2 de VDI. Voici un programme qui vous affiche le tableau de correspondance entre les couleurs VDI et les couleurs Hard. Cf le manuel de programmation pour plus d'informations

```
CLS : PRINT CHR$(27); "f";
CLIP 0, 0, 320, 200
PRINT @(13,0); "Première ligne : couleurs VDI."
PRINT @(14,0); "Deuxième ligne : Couleurs Hardware."
TEXT  HEIGHT = 4
FOR T = 0 TO 15
    FILL  COLOR = T
    PBOX  T*10, 0,10,100
    TEXT T*20, 128,  STR$(T)
NEXT T
FOR T=0 TO 15
    TEXT T*20, 136,  STR$( POINT(T*10+1,10))
NEXT T
PRINT @(18,0); "Les couleurs VDI servent aux ";
```

```
PRINT "instructions graphiques."
PRINT @(0,0); "Equivalence couleurs VDI et HARDWARE"
PRINT @(20,0); "Attention : Basse résolution seulement!";
```

LINE COLOR =

Syntaxe : LINE COLOR= <Teinte>

Le numéro de la teinte est un nombre compris entre 0 et 15. En monochrome, il y a seulement les teintes 0 et 1 (blanc/noir), en moyenne résolution, seules les teintes de 0 à 3 sont disponibles. Le numéros des teintes n'a rien à voir avec le numéro des teintes de la palette (cf le manuel de programmation). L'OMIKRON utilise le même classement des teintes pour LINE COLOR et les autres fonctions de ce style.

LINE STYLE =

Syntaxe : LINE STYLE =<Numero_Style>

A la place des lignes droites, vous pouvez ainsi obtenir des lignes discontinues ou pointillées par le changement du style de ligne :

Style	Dessin de la ligne	
1	1111111111111111	(Droit)
2	1111111111110000	(Tiret long, peu d'espaces)
3	1110000011100000	(Pointillés)
4	1111111000111000	(Tiret–point)
5	1111111100000000	(Court discontinu)
6	1111000110011000	(Tiret–point–point)
7		Définissable par l'utilisateur(Cf GEMLIB)

(0 indique l'absence de point, 1 la présence d'un point)

Exemple :

Cf DRAW

FILL STYLE =

Syntaxe : FILL STYLE=<Style_Remplissage>,<Index>

Fixe le type de remplissage et le motif de remplissage. Le style de remplissage est fixé comme suit :

0	Ne pas remplir (blanc).
1	Remplir entièrement (noir).
2	Remplir (pointillés).
3	Remplir (hachurés).
4	Remplir avec un motif défini par l'utilisateur (cf GEMLIB).

On peut aussi choisir la teinte avec FILL COLOR=. Le deuxième paramètre est significatif uniquement avec les styles de remplissages 2 et 3. Il indique le motif de remplissage. Consultez les dessins représentants les motifs de remplissage en annexe.

FILL COLOR =

Syntaxe : FILE COLOR=<Numero_Teinte>

Fixe les teintes avec lesquelles on doit colorier les surfaces à remplir. Pour les teintes, consultez LINE COLOR=.

TEXT STYLE =

Syntaxe : TEXT STYLE=<Effet>

TEXT STYLE détermine certains effets pour les caractères affichés avec TEXT. Chaque effet est codé sur un bit dans un octet réservé :

Bit	Valeur	Effet
0	1	Gras
1	2	Grisé
2	4	Italique
3	8	Soulignement
4	16	Evidé
5	32	Nuancé (impossible avec le GEM actuel)

Si vous voulez obtenir une écriture grasse et soulignée, additionnez les valeurs des bits 1 et 4 :

```
TEXT STYLE =  1+8 : TEXT 10,10 "OMIKRON ".
```

Exemple :

```
REPEAT
    CLS : PRINT CHR$(27); "f";
    INPUT "STYLE TEXTE =  "; Styl
    TEXT STYLE = Styl
    INPUT  "TAILLE TEXTE = ";Taille
    TEXT HEIGHT = Taille
    INPUT "COULEUR TEXTE = "; Couleur
    TEXT COLOR = Couleur
    TEXT 10,100, "APPUYEZ UNE TOUCHE POUR CONTINUER "
    WHILE INKEY$ = "" : WEND
UNTIL  0
```

TEXT HEIGHT =

Syntaxe : TEXT HEIGHT = <Hauteur_En_Pixels>

Fixe la hauteur des caractères affichés avec TEXT.

Exemple :

Cf TEXT STYLE

TEXT COLOR =

Syntaxe : TEXT COLOR = <Numero_Teinte>

Fixe la teinte des caractères affichés par TEXT. Pour les teintes, consultez LINE COLOR=.

Exemple :

Cf TEXT STYLE

Nouveau : Version 3.0 **OUTLINE**

Syntaxe : OUTLINE {ON|OFF}

Encore une instruction GEM. Elle autorise ou non l'entourage des cercles, ellipses, rectangles ...

Exemple :

```
PRINT CHR$(27); "f"; : CLS : OUTLINE  OFF : FILL STYLE = 3,5
PBOX 100,100 TO 200,200 : OUTLINE  ON : PBOX 300,100 TO 450,200
```

Nous venons de voir les fonctions de base. Passons maintenant à des fonctions plus évoluées :

Nouveau : Version 3.0 **POLYGON**

Syntaxe : POLYGON <Tableau>

POLYGON est une instruction en provenance directe de GEM. Elle permet d'afficher à l'écran un polygone dont les coordonnées se trouvent dans un tableau à deux dimensions. Le tableau est obligatoirement un tableau d'entiers 16 bits.

La première dimension du tableau est toujours 0 et l'abscisse commence toujours sur le premier élément du tableau (2ème dimension, index 0).

Les coordonnées doivent se trouver dans le tableau de la manière suivante :

A%(0,0)	Abscisse du premier point
A%(0,1)	Ordonnée de premier point
A%(0,2)	Abscisse du second point
A%(0,3)	Ordonnée du second point
...	
A%(0,n)	Abscisse du dernier point. L'abscisse du dernier point est égale à l'abscisse du premier.
A%(0,n+1)	Ordonnée du dernier point. elle est égale à l'ordonnée du premier.

Limites : Vous ne devez pas dépasser 130 éléments.

Nouveau : Version 3.0 **PPOLYGON**

Syntaxe : PPOLYGON <Tableau>

PPOLYGON fonctionne exactement comme POLYGON. En outre, elle remplit le polygone avec le style et la couleur de remplissage actifs. Exemple :

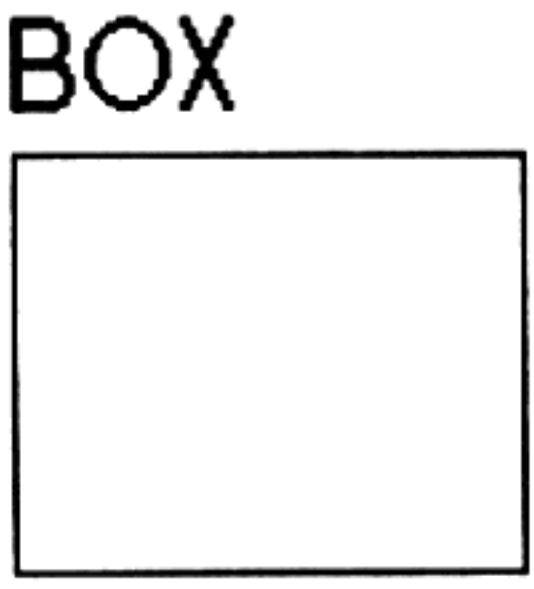
```
DIM A%(0,130) : REM Dimensionnement du tableau
PRINT CHR$(27); "f" : CLS : REM Effacement du curseur
CLS : REM Effacement de l'écran
MOUSEON : REM On allume la souris
FOR T = 0 TO 126 STEP 2
    REM STEP 2 car on fait deux points à la fois
    WHILE MOUSEBUT = 0 : REM Attente touche de la souris
    WEND
    A%(0,T) = MOUSEX : REM coordonnées dans le tableau
    A%(0,T+1) = MOUSEY
    MOUSEOFF  : DRAW MOUSEX , MOUSEY : MOUSEON
    WHILE MOUSEBUT <> 0 : WEND
    REM Efface la souris, affiche le point, réaffiche la souris
    REM On attend qu'aucune touche de la souris ne soit active.
NEXT T
A%(0,128) = A%(0,0) : A%(0,129) = A%(0,1)
REM Dernier point = premier point
MOUSEOFF : REM Toujours éteindre la souris.
'
'
FOR Fil=1 TO 22 : REM Et si on essayait tous les remplissages?
    FILL STYLE = 2 , Fil
    PPOLYGON A%(0,130) : REM Dimension maxi
NEXT  Fil
```

BOX

Syntaxe : BOX <X>,<Y> TO <X2>,<Y2>
Syntaxe : BOX <X>,<Y>,<Largeur>,<Hauteur>

Dessine un rectangle. La première syntaxe s'utilise comme la fonction équivalente du VDI. Il faut donner les coordonnées du coin supérieur gauche et celles du coin inférieur droit.

Avec la seconde syntaxe, vous devez indiquer le coin supérieur gauche ainsi que la largeur et la hauteur du rectangle. BOX imite cette fois l'AES. Le rectangle n'est pas rempli, mais vous pouvez choisir la teinte des lignes constitutives grâce à LINE COLOR= et le style des lignes avec LINE STYLE.



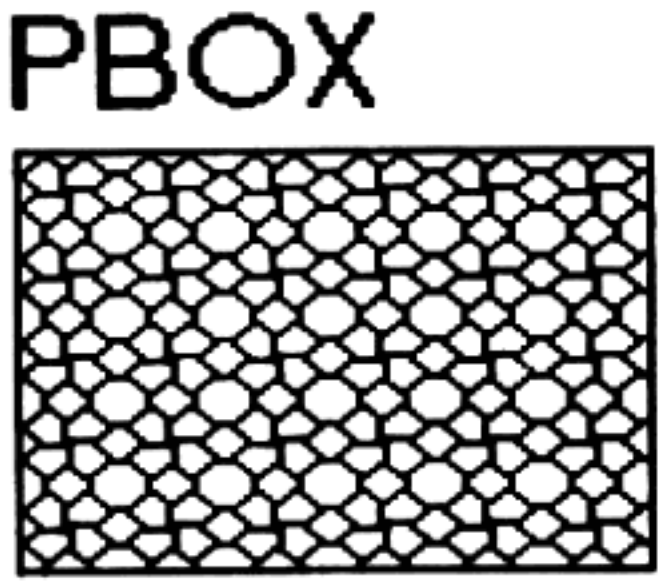
PBOX

Syntaxe : PBOX <X>,<Y> TO <X2>,<Y2>
Syntaxe : PBOX <X>,<Y>,<Largeur>,<Hauteur>

Dessine un rectangle rempli. La première syntaxe s'utilise comme la fonction équivalente du VDI. Il faut donner les coordonnées du coin supérieur gauche et celles du coin inférieur droit.

Avec la seconde syntaxe, vous devez indiquer le coin supérieur gauche ainsi que la largeur et la hauteur du rectangle. PBOX imite cette fois l'AES.

On détermine le style remplissage avec FILLSTYLE=, la teinte de remplissage avec FILL COLOR=, celle des lignes constitutives grâce à LINE COLOR= et le style des lignes avec LINE STYLE=.



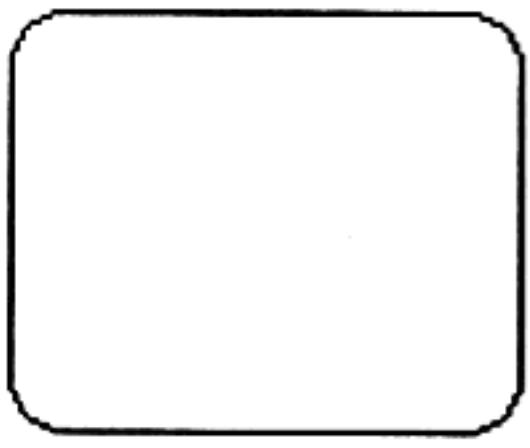
RBOX

Syntaxe : RBOX <X>,<Y> TO <X2>,<Y2>
Syntaxe : RBOX <X>,<Y>,<Largeur>,<Hauteur>

Dessine un rectangle aux coins arrondis. La première syntaxe s'utilise comme la fonction équivalente du VDI. Il faut donner les coordonnées du coin supérieur gauche et celles du coin inférieur droit.

Avec la seconde syntaxe, vous devez indiquer le coin supérieur gauche ainsi que la largeur et la hauteur du rectangle. RBOX imite cette fois l'AES. Le rectangle n'est pas rempli, mais vous pouvez choisir la teinte des lignes constitutives grâce à LINE COLOR= et le style des lignes avec LINE STYLE=.

RBOX



PRBOX

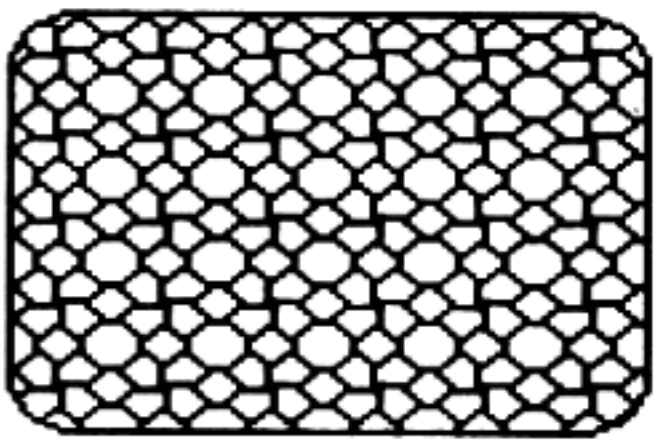
Syntaxe : PRBOX <X>,<Y> TO <X2>,<Y2>
Syntaxe : PRBOX <X>,<Y>,<Largeur>,<Hauteur>

Dessine un rectangle rempli aux coins arrondis. La première syntaxe s'utilise comme la fonction équivalente du VDI. Il faut donner les coordonnées du coin supérieur gauche et celles du coin inférieur droit.

Avec la seconde syntaxe, vous devez indiquer le coin supérieur gauche ainsi que la largeur et la hauteur du rectangle. PRBOX imite cette fois l'AES.

On détermine le style de remplissage avec FILLSTYLE=, la teinte de remplissage avec FILL COLOR=, celle des lignes constitutives grâce à LINE COLOR= et le style des lignes avec LINE STYLE=.

PRBOX



CIRCLE

Syntaxe : CIRCLE <X>,<Y>,<Rayon> [,[Angle_Depart] [Angle_Arrivee]]

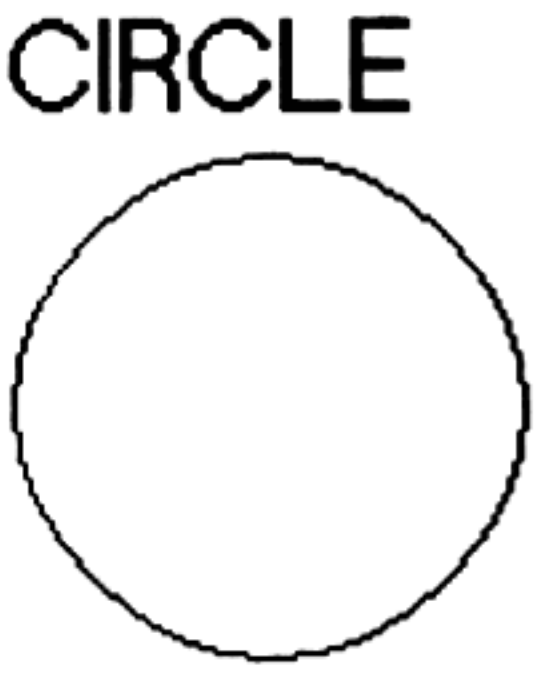
Dessine un cercle de centre X, Y.

Les angles de départ et d'arrivée ont une précision maximum de 1/10 ème de degré. Un angle de 90° se rentre sous la forme 900.

Le cercle n'est pas rempli, mais vous pouvez choisir la teinte du pourtour du cercle grâce à LINE COLOR= et son style avec LINE STYLE=.

Exemple :

```
CLIP 0,0,320,200
FOR T = 1 TO 50
CIRCLE RND(320),RND(200),RND(80)
NEXT T
```



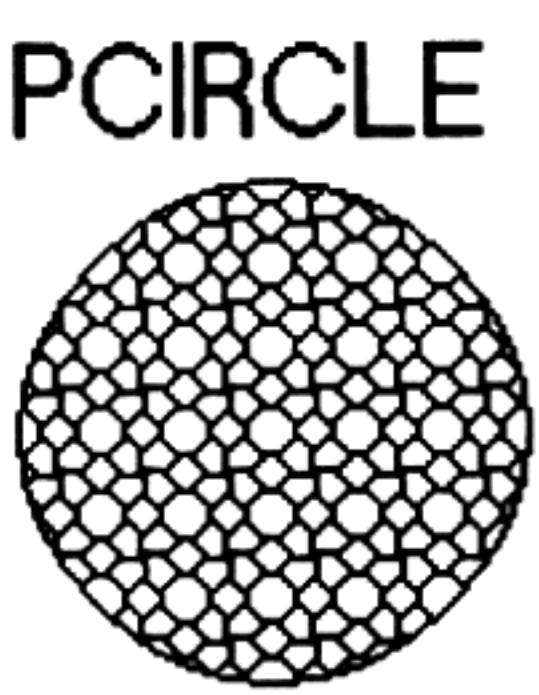
PCIRCLE

Syntaxe:PCIRCLE<X>,<Y>,<Rayon>[,[Angle_Depart][,Angle_Arrivee]]

Dessine un cercle rempli de centre X, Y.

Les angles de départ et d'arrivée ont une précision maximum de 1/10ème de degré. Un angle de 90° se rentre sous la forme 900. Le cercle est rempli. La teinte de remplissage est choisie par FILL COLOR= et le style par FILL STYLE=.

Vous pouvez choisir la teinte du pourtour du cercle grâce à LINE COLOR= et son style avec LINE STYLE=.



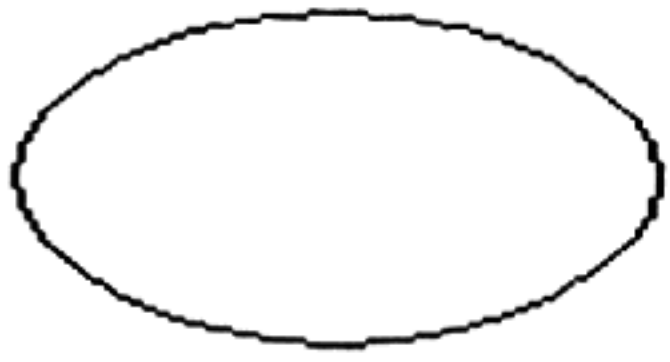
ELLIPSE

Syntaxe:ELLIPSE<X>,<Y>,<Ray_X>,<Ray_Y>[,[Ang_Dep][,Ang_Arr]]

Dessine une ellipse dont le centre se situe en X,Y et dont les deux rayons sont Ray_X et Ray_Y.

Les angles de départ et d'arrivée ont une précision maximum de 1/10 ème de degré. Un angle de 90° se rentre sous la forme 900. L'ellipse n'est pas remplie mais vous pouvez choisir la teinte de l'ellipse grâce à LINE COLOR= et son style avec LINE STYLE=.

ELLIPSE



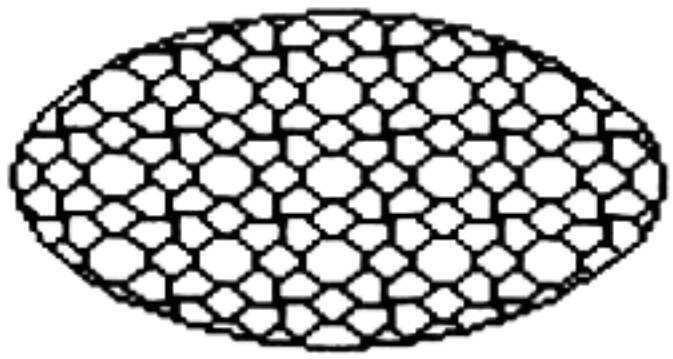
PELLIPSE

Syntaxe:PELLIPSE<X>,<Y>,<Ray_X>,<Ray_Y>[,[AnglDep],Ang_Arri]]

Dessine une ellipse remplie, dont le centre se situe en X, Y et dont les deux rayons sont Ray_X et Ray_Y.

Bien entendu, le style et la couleur du pourtour et du remplissage sont définissable comme précédemment.

PELLIPSE



FILL

Syntaxe : FILL <X>, <Y>,<Teinte_Limite>

Remplit une surface avec la teinte indiquée par FILL COLOR= et avec le motif choisi par FILL STYLE=. Attention, car le remplissage ne s'arrêtera qu'après avoir rempli entièrement la surface. Il n'y a aucun moyen de le stopper en route (à part éteindre le ST). La surface à remplir est limitée par les points ayant la couleur <Teinte_Limite>. Si <Teinte_Limite>=-1, la surface à remplir est limitée par tous les points ayant une autre teinte que la teinte de remplissage (cf le manuel de programmation).

TEXT

Syntaxe:TEXT<X>,<Y>,<"texte">,
[<Longueur>],[<Espacement_Mots>],<Espacement_Caracteres>]

Affiche le texte indiqué en X,Y.

Par exemple :

```
TEXT 100, 180, "OMIKRON pour toujours"
```

Ecrit le texte "OMIKRON pour toujours" au point X=100 et Y=180 (le coin supérieur gauche du texte est 100,180). Où cela se situe-t-il? Cela dépend de la résolution choisie :

- Monochrome (640*400 points), à gauche, environ à mi-hauteur de l'écran.
- Basse résolution (320*200 points), quasiment en bas de l'écran et environ au tiers de la largeur.

Les paramètres <Espacement_Mots> et <Espacement_Caracteres> sont des interrupteurs d'allumage (1) ou d'extinction (0).

Avec l'indication de la longueur, ils ont la signification suivante : le texte sera formaté à la longueur indiquée, soit par l'insertion d'espaces entre les mots, soit par l'insertion d'espaces entre les lettres ; et selon ce que vous indiquez, la combinaison des deux est possible.

Par exemple :

```
TEXT 100, 180, "Sam T. was here", 200, 1, 0
```

Le texte "Sam T. was here" sera situé au point X=100 et Y=180. Sa longueur sera étendue jusqu'à 200 points et cela grâce à l'insertion d'espaces entre les mots.

La teinte du texte est choisie avec TEXT COLOR=, la taille avec TEXT HEIGHT=, son orientation avec TEXT ORIENTATION= et les effets spéciaux avec TEXT STYLE=.

Voyons maintenant une fonction indispensable (en particulier quand vous travaillez avec les fenêtres).

CLIP

Syntaxe : CLIP <X>,<Y> TO <X2>,<Y2>
Syntaxe : CLIP <X>,<Y>, <Largeur>, <Hauteur>
Syntaxe : CLIP

CLIP fixe les limites de la fenêtre d'affichage pour les instructions graphiques. Après un CLIP, toutes les instructions graphiques ne tiendront plus compte de la taille réelle de l'écran mais du 'CLIPPING RECTANGLE'.

Essayez de qui suit en haute ou en moyenne résolution :

```
CIRCLE 600, 100, 150
REM l'écran est replié sur lui-même, comme une sphère
WHILE MOUSEBUT = 0 : WEND
CLS
CLIP 0, 0, 640, 400 : REM ou 200 en moyenne résolution
CIRCLE 600, 100, 150
```

Comme vous le voyez, sans CLIP le cercle déborde sur la gauche. Il faut donc dire à GEM que l'écran commence en 0,0 et finit en 640,400.

Dès lors, les fonctions graphiques ne s'afficheront que sur le rectangle 'clippé'.

LIMITES : En fait, dès qu'on utilise des fonctions graphiques, il faut employer CLIP. Sans CLIP, les fonctions graphiques n'ont pas de limites et elles n'hésitent pas à taper n'importe où dans la mémoire et font souvent des dégats (cf l'exemple suivant). Essayez cette ligne sans CLIP :

```
CIRCLE 200, 100, 3000
```

Votre ATARI plantera car CIRCLE va atteindre des espaces mémoires réservés qui appartiennent à l'OMIKRON ou au système.

Les CLIP standards à exécuter avant les instructions graphiques sont les suivants :

```
CLIP 0, 0, 640, 400 :   REM Haute résolution
CLIP 0, 0, 640, 200 :   REM Moyenne résolution
CLIP 0, 0, 320, 200 :   REM Basse résolution
```

Mais CLIP fonctionne avec d'autres tailles, tapez :

```
FILL STYLE = 1
PCIRCLE 125, 150, 35
```

Effacez l'écran ([CTRL]–[Clr/Home]), et tapez :

```
BOX 99, 99 TO 151, 151
CLIP 100, 100 TO 150, 150
PCIRCLE 125, 125, 35
```

Au lieu d'un cercle, on voit un rectangle (ou un carré, selon la résolution), aux coins duquel on distingue quatres arcs.

Pour annuler un CLIP, il n'y a qu'une seule façon :

```
CLIP [Return]
```

POINT

Syntaxe : A=POINT (X,Y)

POINT permet de tester la couleur du point ayant pour coordonnées X, Y. En monochrome, la teinte est soit 1, soit 0 (existence ou non du point). La valeur résultante est le numéro de la teinte du point dans les autres définitions.

ATTENTION: POINT retourne le numéro HARDWARE de la teinte. Cf PALETTE et le manuel de programmation.

BITBLT

Syntaxe : BITBLT <M> TO X,Y,L,H [,<MODE>]
Syntaxe : BITBLT <X>,<Y>,<L>,<H> TO <X2>,<Y2>,<L>,<H2>[,<MODE>]
Syntaxe : BITBLT <X>,<Y>,<L>,<H> TO <M>

Effectue un "BIT–BLock–Transfer" (transfert d'un bloc par bits). Dans le principe, BITBLT n'est rien d'autre que la copie d'une partie de l'écran en mémoire et vice–versa.

Récapitulons les possibilités :

- De l'écran sur l'écran.
- De l'écran à la mémoire.
- De la mémoire à l'écran.

Commençons par le plus simple :

Le transfert de l'écran vers l'écran. Les coordonnées X,Y indiquent le point supérieur gauche d'un rectangle de largeur L et de hauteur H. Ce rectangle est recopié sur le rectangle récepteur qui commence en X2, Y2, d'une largeur L2 et d'une hauteur H2.

Si les deux blocs n'ont pas exactement la même taille, l'OMIKRON prend toujours la plus petite largeur et la plus petite hauteur.

Exemple :

```
BITBLT 100, 100, 20, 20 TO 200, 200, 10, 10
```

Ici, BITBLT copie un rectangle de 10 sur 10 en 200,200.

BITBLT fonctionne avec CLIP. Les blocs sont affichés sur le rectangle 'clippé'. Il est par contre INTERDIT de prendre comme source un bloc qui n'est pas complètement dans le rectangle 'clippé'. Le bloc résultant contiendrait tout et n'importe quoi. Il est conseillé comme pour toutes les instructions graphiques d'utiliser un CLIP. Dans le cas de BITBLT, l'absence de CLIP signifie que BITBLT n'a aucune limite. Gare à la casse!!!

Copier une partie de l'écran en mémoire.

Avant toute utilisation de la mémoire pour sauvegarder un bloc, il est indispensable d'effectuer un ou plusieurs MEMORY. Utiliser un BITBLT mémoire sans MEMORY revient à détruire le système.

La syntaxe est un peu plus simple qu'avec le BITBLT précédent :

```
Place_Libre = MEMORY (500)
BITBLT 100, 100, 30, 20 TO Place_Libre
```

Le rectangle 100,100,30,20 sera copié dans la mémoire à l'adresse Place-libre réservée par MEMORY. Souvenez-vous que 30 et 20 indiquent la largeur et la hauteur et que les coordonnées du rectangle sont en fait 100,100,130,120.

Bien entendu, BITBLT permet aussi de recopier sur l'écran les parties de l'écran sauvegardées précédemment en mémoire. Toujours avec le même exemple :

```
BITBLT Place_Libre TO 150, 150, 30, 20
```

IMPORTANT : Quand vous sauvez des parties d'écran en mémoire, vous devez absolument noter la hauteur, la largeur et éventuellement la résolution.

Mais comment sont stockés les blocs en mémoire ?

Avec notre MEMORY, nous avons réservé 500 octets. Imaginons que l'adresse attribuée par l'OMIKRON pour cette zone de mémoire se situe en $7000. Après le BITBLT, nous trouvons plusieurs informations qui se situent devant le bloc proprement dit.

$7000	Nombre de plans * 2 (nombre de plans = racine carrée du nombre de couleurs), soit 2 pour monochrome, 4 en moyenne, 8 en basse.
$7002	Largeur en pixels.
$7004	Hauteur en pixels.
$7006	Début des données du bloc.

Si vous désirez transférer vous-même des blocs vers la mémoire, il vous faut impérativement tenir compte de ces trois mots. Si vous vous mélangez les pinceaux, votre BITBLT aura un drôle d'air! Bien entendu, BITBLT s'occupe tout seul des ces trois mots, alors un bon conseil : laissez-le faire!

LE MODE BITBLT

Vous ne pouvez indiquer le mode que si BITBLT déplace un bloc vers l'écran et non pas vers la mémoire. Mode indique à BITBLT l'opération logique à effectuer lors de la copie sur l'écran du bloc source.

Il existe 16 modes différents. Dans le tableau qui suit, S signifie Source, un point du bloc de départ, et B signifie But, le point du bloc d'arrivée ayant les mêmes coordonnées.

MODE	Résultat
0	= 0 (Pas de point)
1	S ET B
2	S ET (NON B)
3	S (Remplacement, mode par défaut)
4	(NON S) ET B
5	B
6	S XOR B
7	S OR B
8	NON (S OR B)
9	NON (S XOR B)
10	NON B
11	S OR (NON B)
12	NON S
13	(NON S) OR B
14	NON (S OR B)
15	= 1 (tous les points)

TEXT ROTATION =

*Syntaxe : **TEXT ROTATION** = <Expression_Numerique>*

TEXT ROTATION permet de fixer la direction du texte affiché. Il est possible d'entrer 0, 90, 180 et 270 °, sous la forme 900, 1800 et 2700. Exemple : TEXT ROTATION = 270: TEXT 100,100,"Tu me fais tourner la

LINE WIDTH =

*Syntaxe : **LINE WIDTH** = <Expression_Numerique>*

permet de choisir la largeur des droites affichées par DRAW. Les paramètres, pour être valables, doivent être des nombres impairs (1, 3, 6,7...).

SPRITES

Vous connaissez tous la superbe abeille qui s'affiche à l'emplacement du curseur de la souris quand votre ST travaille. Et bien, cette abeille est un SPRITE, un lutin en français. Les sprites permettent aux programmeurs de gérer 'simplement' une animation graphique. On n'a plus à calculer les déplacements des blocs avec des BITBLT. Il suffit d'indiquer le numéro du sprite, les coordonnées et le numéro du dessin.

Les SPRITES créés en basic OMIKRON sont rapides et disposent de plusieurs tailles. Afin de programmer les sprites, l'OMIKRON comprend deux instructions :

DEF SPRITE

Syntaxe : DEF SPRITE <Numero>,<Type>,[Adresse],[Sauvegarde]

Définit un SPRITE sans l'afficher à l'écran. Le premier paramètre donne le numéro du sprite (il y en a 8, qui vont de 0 à 7).

Le type de sprite se calcule selon la formule :

- Type = Résolution* 2 + Taille

La résolution est égale à :

0 Basse résolution
1 Moyenne résolution
2 Haute résolution

et la taille est indiquée par 1 pour petit et 2 pour grand.

Formats en pixels et types des SPRITES en fonction de la résolution et de la taille choisie :

Résolution	Petit	Type	Grand	Type
Basse	8 * 8	1	16 * 16	2
Moyenne	16 * 8	3	32 * 16	4
Haute	16 * 16	5	32 * 32	6

Exemple : vous voulez définir un sprite ayant une taille de 32*32 en haute résolution. Le calcul du type est le suivant :

- 2*2 (MODE 640*400)+ 2 (la taille) =6.

Adresse pointe sur un endroit libre de la mémoire. L'OMIKRON utilisera cet emplacement pour sauvegarder le fond lors des déplacements du sprite.

Pour <Sauvegarde>, consultez le manuel de programmation.

SPRITE

Syntaxe : SPRITE <numero>,[<X>],[<Y>],[<Dessin>] [<Couleur_Masque>],[<Couleur_Dessin>]

Affiche le sprite demandé sur l'écran en X, Y.

Si SPRITE est appelé pour la première fois, il faut indiquer l'adresse du dessin du sprite.

Dans le dessin du sprite, les informations s'articulent en deux parties :

- Le masque du sprite.
- Le dessin du sprite.

A chaque ligne du sprite correspond 1, 2 ou 4 octets de masque et 1, 2 ou 4 octets de dessin en mémoire. Donc, chaque ligne composant le sprite est composée d'une partie masque et d'une partie dessin.

Quelle que soit la définition, chaque pixel d'un sprite est codé sur deux bits (1 bit pour le masque et un bit pour le dessin). C'est pourquoi le choix des couleurs pour les sprites est restreint puisqu'il n'y a que deux possibilités.

La couleur d'affichage du sprite est donc déterminée par le bit du masque et le bit du dessin correspondant :

Bit du masque	Bit du dessin	Couleur résultante
0	0	Fond
1	0	Masque du sprite
0	1	INTERDIT
1	1	Dessin du sprite

LES FONCTIONS GEM DIRECTES

Ces fonctions GEM sont particulières car elles sont directement implantées dans le basic OMIKRON, ce qui signifie que vous pouvez les obtenir à tout moment sans charger la bibliothèque GEM.

FORM_ALERT

Syntaxe : FORM_ALERT(<Defaut>, <texte>, <Choix>)
Syntaxe : FORM_ALERT(<Defaut>, <texte>, < Choix>, <Retour>)

FORM_ALERT appelle une boîte d'alerte GEM. Les boîtes d'alerte servent à informer l'utilisateur d'un événement important. L'utilisateur peut généralement choisir entre 2 ou 3 possibilités :

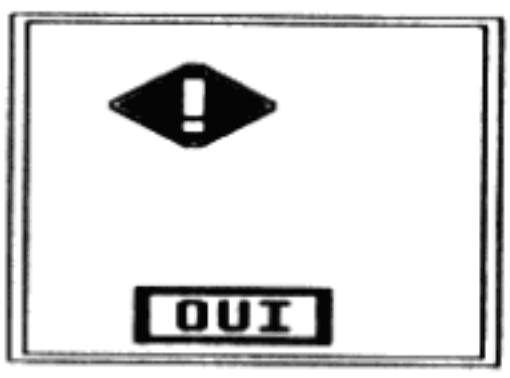
<Defaut> désigne le numéro du choix pouvant être directement validé par [Return]. <Defaut> est compris entre 1 et 3. Une valeur nulle signifie qu'aucun choix n'est validable par [Return].

<texte> est une chaîne. Sa construction est la suivante :

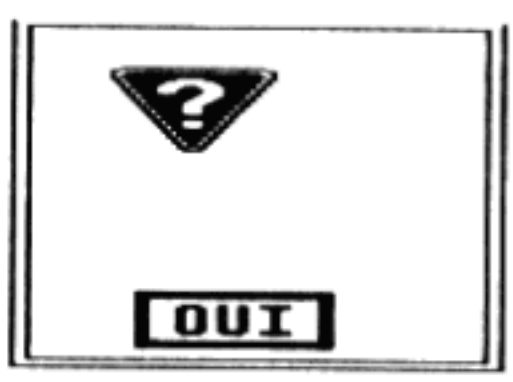
"[1][Ligne1/Ligne2/Ligne3]"

[1] est le numéro du symbole affiché en haut à gauche. La liste de ces symboles est la suivante :

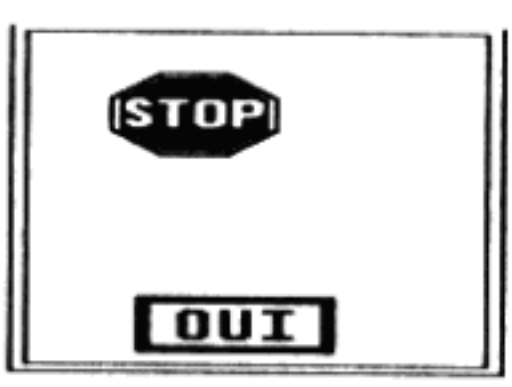
0	Aucun symbole
1	Symbole "Attention" (!)
2	Symbole d'interrogation (?)
3	Symbole STOP



Une action dangereuse va se produire.



Choix entre plusieurs alternatives.



Un problème vient d'arriver.

Les crochets appartiennent à la chaîne. Vous devez impérativement les taper sous peine d'obtenir de magnifiques erreurs.

[Ligne1/Ligne2/Ligne3] constitue le texte à afficher dans la boîte d'alerte. Le signe "|" ([Shift]-[#] situé à droite de [Return] sur le clavier) oblige GEM à passer à la ligne suivante.

[Choix1/Choix2/Choix3] sont les choix possibles dans la boîte d'alerte. Le signe "|" constitue le séparateur entre les choix. *<Retour>* est une variable dans laquelle le système retourne le numéro du choix sélectionné.

Exemples :

```
FORM_ALERT (1,"[2][La ville de REIMS|est-elle en Champagne?]
[OUI | NON]")
```

Affiche à l'écran une boîte d'alerte avec un point d'interrogation (symbole numéro 2) qui contient deux lignes de texte.

Première ligne	"La ville de REIMS"
Deuxième ligne	"est-elle en Champagne ?"

Juste en dessous, GEM affiche deux choix :

"OUI" et "NON".

```
ON ERROR GOTO Erreur
.
. Ici se trouve votre programme
.
-Erreur
Texte$ = "Une erreur vient de se produire"
Texte$ = Texte$+"|Erreur numéro"+STR$( ERR)+", ligne "
Texte$ = Texte$+ STR$(ERL) + "|" + LEFT$( ERR$,28) + ")"
MOUSEON
FORM_ALERT(2, "[2] [ "+Texte$ +"][STOP|CONTINUE]",Choix)
MOUSEOFF
IF Choix = 1 THEN Erreur = ERR : RESUME NEXT ELSE RESUME
```

LIMITES : Les cadres d'alerte NE SONT PAS CONCUS pour écrire des romans ! Si vous dépassez 40 caractères sur une ligne ou 30 caractères sur 2 ou 3 lignes, GEM **PLANTE** sans possibilité de reprise. Il est alors obligatoire d'éteindre l'ATARI. Donc, vous devez absolument tester la longueur des chaînes que vous allez afficher dans les cadres d'alerte.

FILESELECT

Syntaxe : FILESELECT(<Chemin_Acces>, <Nom_Fichier>, <Drapeau>)

FILESELECT appelle le sélecteur de fichier grâce auquel on peut choisir facilement un fichier.

<Chemin_Acces> est une chaîne contenant le chemin d'accès au fichier. La chaîne doit obligatoirement contenir un chemin d'accès correct (par exemple "A :*.*").

<Nom_Fichier> désigne une variable contenant le nom du fichier.

<Drapeau> est une variable par laquelle le système communique le choix de l'utilisateur : 0 pour ANNULER et 1 pour CONFIRMER.

Exemples :

```
Chemin$ = "A :\*.* "
FILESELECT (Chemin$, Nom_Fichier$, Okay)
IF Okay
THEN PRINT "Vous avez choisi ";Nom_Fichier$
ELSE PRINT  "Annulation !"
ENDIF
GEMDOS(Drive,$19) : REM Quel est le lecteur par défaut ?
Chem$ = " "* 64
Adr = LPEEK (SEGPTR+28)+ LPEEK (VARPTR (Chem$))
GEMDOS (,47,L Adr,0) : REM Chemin d accès par défaut
Chem$ = LEFT$(Chem$, INSTR(Chem$ + CHR$(0), CHR$(0))
Chem$ = CHR$(65 + Drive) +  ":" + Chem$ + "\*.BAS "
REM
MOUSEON : FILESELECT(Chem$, Nom, Okay) : MOUSEOFF
REM
Nom_Fich$ = LEFT$(Chem$, LEN(Chem$)-INSTR (MIRROR$
(Chem$ + "\"))+ Nom$
IF Okay AND Nom_Fich$<>" " THEN
    OPEN "I ",1,Nom_Fich$
ELSE
    END
ENDIF
```

Consultez le manuel de programmation pour plus de détails sur GEM.

MOUSEON

Syntaxe : MOUSEON

MOUSEON affiche la souris.

ATTENTION : si vous effacez cinq fois la souris, il faudra la réafficher cinq fois avant qu'elle ne réapparaîsse. C'est un avantage pour tous les sous-programmes qui peuvent ainsi remettre la souris dans son état initial en sortant.

Exemple :

```
CLS
Cercle (100, 100, 95, 5) : Zz_Up (190, 10, 3)
.
.
DEF PROC Cercle(X, Y, R, F)
    MOUSEOFF
    CIRCLE X, Y, R : Zz_Up (X,Y,F) : CIRCLE X, Y, R+1
    MOUSEON
RETURN
'
DEF PROC Zz_Up(X,Y,S)
MOUSEOF : FILLSTYLE = 2,5 : FILL X, Y : MOUSEON
RETURN
```

MOUSEOFF

Syntaxe : MOUSE OFF

Efface la souris.

IMPORTANT : Avant d'afficher du graphisme ou d'effacer l'écran, il faut d'abord éteindre la souris puis la réafficher une fois ces opérations terminées.

LA CHASSE AUX BUGGS

KEY

Syntaxe : KEY <Touche_Fonction> = "texte"

ATTENTION : KEY ne fonctionne que dans les programmes ou dans l'éditeur standard (cf LIMITES).

Il arrive qu'une chaîne de caractères revienne souvent dans un programme. On peut gagner beaucoup de temps en attribuant la chaîne à une touche de fonction. KEY permet de définir les touches [F1] à [F10] (KEY 1 à KEY 10) et [Shift]–[F1] à [Shift]–[F10] (KEY 11 à KEY 20), soit 20 touches différentes. La taille maximum pour la définition d'une touche de fonction est de 32 caractères.

Exemple :

Vous voulez que [F1] contienne le texte "OMIKRON", tapez :

```
KEY 1 = "OMIKRON"
```

Une pression sur [F1] affiche la chaîne "OMIKRON".

Les séquences VT52 (Effacement écran, curseur éteint...) sont autorisées. Exemple :

```
KEY 10 = "[CTRL]-[A][Cur_Droit][CTRL]-[A][Cur_Droit][Enter] "
```

Avance le curseur de deux lignes à chaque pression sur [F10]. Après avoir tapé cet exemple, quelque chose ne vous semble–t–il pas bizarre? Sur l'écran n'apparaît aucun [Cur_Droit]. A la place, deux caractères sont affichés à chaque pression de la touche [Cur_Droit], un signe 'ESC' et un 'B'. La raison : 'Escape–B' est le code VT–52 pour [Cur_Droit].

Certains caractères ne peuvent être entrés directement. Prenons le cas de [Enter]. Dès que vous appuyez sur [Enter], le basic OMIKRON estime que la définition de la touche est terminée et il passe à la ligne suivante. Voici une solution à ce problème :

```
KEY 10 = "RUN" + CHR$(13)
```

L'expression 'CHR$(13)' correspond au caractère [Enter] (cf Tableau des caractères ASCII, le manuel de programmation). Bien sûr, le fait de taper CHR$(13) ne vous dispense pas de taper [Enter] à la fin de la définition (au contraire !!!).

Une autre difficulté peut provenir des guillemets. Comment introduire la chaîne "FILES "A:*.BAS"+CHR$(13)" ? Il suffit de remplacer les guillemets appartenant à la chaîne par des CHR$(34) :

```
KEY 5 = "FILES " + CHR$(34) + "A:\*.BAS " + CHR$(34) + CHR$(13)
```

Une pression sur [F5] et les programmes en basic de la disquette présente dans le lecteur A s'affichent.

LIMITES : Les touches de fonctions ne sont pas utilisables dans le super-éditeur qui les réserve pour l'appel des fonctions étendues. L'utilisation principale des touches de fonctions est de supprimer les saisies répétitives.

KEY LIST

Syntaxe : KEY LIST

Affiche à l'écran toutes les définitions des touches de fonctions.

TRON

Syntaxe : TRON

Met en marche le mode TRACE. Avant l'exécution de chaque instruction, TRACE affiche le numéro de ligne ainsi que son libellé.

Exemple :

```
10 FOR I = 1 TO 5
20 LET A(I) = I^2
30 NEXT I : REM Tapez TRON et RUN

[10 FOR] [20 LET] [30 NEXT] [20 LET] [30 NEXT] [20 LET] [30 NEXT]
[20 LET] [30 NEXT] [20 LET] [30 NEXT] [30 END]
```

LIMITES : Cette instruction n'est pas forcément intéressante telle quelle et on préférera l'utiliser conjointement avec ON TRON...GOSUB. ON TRON...GOSUB permet de sauter après chaque instruction à un sous-programme de votre crû spécifiquement adapté à la mise au point de votre programme.

TROFF

Syntaxe : TROFF

Annule le mode TRACE.

IMPORTANT : Quand les ordres TRON et TROFF se trouvent dans un programme (mais pas en mode direct), les ordres TRON et TRACE sont empilés. Ce qui signifie que si TROFF est exécuté trois fois, il faut absolument exécuter trois TRON pour que TRON soit actif. Si TRON est tapé en mode direct, toute la pile est effacée. De même, en mode direct, TROFF arrête à coup sûr TRON.

ON TRON GOSUB

Syntaxe : ON TRON GOSUB <Sous_Programme>

ON TRON GOSUB se place au début d'un programme. L'instruction de saut GOSUB n'est validée que si le mode TRON est actif. Il faut bien différencier ON TRON GOSUB de TRON. En effet, TRON affiche à l'écran l'instruction ainsi que le numéro de ligne correspondant avant son exécution. Par contre, ON TRON GOSUB se contente de sauter à un sous-programme après chaque instruction. Le sous-programme appelé par ON TRON GOSUB est destiné à débugger le programme en affichant à l'écran, sur imprimante ou sur disque des informations concernant le déroulement du programme.

Comme exemple, nous vous proposons une (belle) petite routine de débuggage :

```
XBIOS(,15,7,0,174,-1,-1,-1) : REM Réglage de la RS-232
REM pour le Minitel standard. Cf Le manuel de programmation.
ON TRON GOSUB Bug .
'
' Ici se trouve votre programme .
'
-Bug
    OPEN "V", 16 : REM Ouverture du port série
    CMD 16 : REM LPRINT, LDUMP ... sortent sur le canal n° 16
    LPRINT ERL ,ERR$ :CLOSE 16
RETURN
```

Pour utiliser cette routine, il vous faut :

- Un minitel
- Une prise entre votre minitel et votre ATARI

Toutes les lignes ainsi que leur libellé apparaîtront sur l'écran du minitel, sans déranger l'affichage de votre ATARI. Pratique, non?

CONSEILS :

- Faites en sorte que votre routine prenne le moins de temps possible pour ne pas trop ralentir votre programme.
- Vous trouverez le numéro de ligne qui vient d'être exécuté dans ERL, le nom de la commande qui vient d'être exécutée dans ERR$.

LIMITES : Aucune routine de traitement d'erreur ne peut être utilisée si ON TRON...GOSUB est actif. En effet, ON ERROR...GOTO GOSUB est inopérant (ON TRON...GOSUB utilise les même variables que les fonctions de traitement des erreurs).

DUMP

Syntaxe : DUMP [Alpha1][,Alpha2]

Affiche toutes les variables suivies de leur valeur. Pour les tableaux, seule la dimension est indiquée.

Exemple :

```
CLEAR
A = 3 :B$ = "OMIKRON" : Q ! =3.5 : DIM Prim%F( 10000,10)
DUMP
```

La liste suivante est affichée :

```
A = 3
B$ = "OMIKRON "
Q! = 3.5
DIM PRIM%F (10000,10)
```

L'OMIKRON 3.0 permet aussi le DUMP sélectif. Au lieu de sortir toutes les variables à chaque fois, on peut parfaitement demander une restriction du DUMP. Reprenons le même exemple :

```
DUMP A-B
```

La liste sera cette fois :

```
A = 3
B$ = "OMIKRON "
```

LDUMP

Syntaxe : LDUMP [Alpha1][,Alpha2]

Imprime la liste des variables sur papier. Si vous voulez sauvegarder la liste des variables sur disquette, voici ce qu'il faut faire:

```
OPEN "O",1, "Nom_Fichier"
CMD 1
REM dirige toutes les sorties sur le fichier dont le numéro est 1.
LDUMP
CLOSE 1
```

Regardez bien CMD, c'est une commande extrêmement puissante et facile à mettre en oeuvre.

RENUM

Syntaxe : RENUM [<Nouvelle_Ligne>][,<Ancienne_Ligne>][,<Pas>]

Les programmes en basic souffrent d'un mal congénital. Après plusieurs modifications, ajouts et suppressions, les lignes sont classées n'importe comment. L'insertion entre deux lignes peut devenir impossible et on se perd dans les numéros...

Une solution consiste à renuméroter le programme ou une partie du programme. Grâce à RENUM, tous les GOTO, GOSUB etc... utilisant des numéros de ligne sont convenablement redirigés.

Exemple :

```
RENUM 1000, 100, 10
```

Renumérote le programme de façon à ce que la ligne 100 devienne la ligne 1000 et que les lignes suivantes soient numérotées de 10 en 10.

RENUM avec seulement un paramètre numérote tout le programme de 10 en 10. Le paramètre indique alors le numéro de la première ligne du programme après renumérotation.

RENUM sans paramètre exécute une numérotation de tout le programme de 10 en 10 avec la ligne 100 comme nouvelle première ligne.

Programme ayant besoin d'un bon petit coup de RENUM :

```
1   CLS
2   PRINT "Veni, vedi, ...euh!?! "
9   GOTO 958
11  PRINT "C'est bien ça! " : GOTO 10501
15  PRINT "Non, tu ne sais pas ta leçon." : GOTO 10501
958   INPUT "Termine cette phrase très connue en tapant le mot
qui manque : ";A$
993   A$ = UPPER$(A$ )
997   IF A$ = "VICI " THEN GOTO 11 ELSE GOTO 15
1050 INPUT "Encore une fois? "; Oui$
1060 Oui$ = UPPER$(Oui$ )
1070 IF Oui$ = "OUI" THEN GOTO 1 ELSE END

RENUM 10, 1, 10

10  CLS
20  PRINT "Veni, vedi, ...euh!?! "
30  GOTO 60
40  PRINT "C'est bien ça!" : GOTO 90
50  PRINT "Non, tu ne sais pas ta leçon." : GOTO 90
60  INPUT "Termine cette phrase très connue en tapant le mot
qui manque : ";A$
70  A$ = UPPER$(A$ )
80  IF A$ = "VICI" THEN GOTO 40 ELSE GOTO 50
90  INPUT "Encore une fois? "; Oui$
100   Oui$ = UPPER$(Oui$ )
110   IF Oui$ = "OUI" THEN GOTO 10 ELSE END
```

C'est tout de même plus clair!

Mais la meilleure solution, et de loin, consiste à ne pas utiliser de numéros de ligne du tout!

LES MATRICES

Le basic OMIKRON dispose de plusieurs commandes destinées à faciliter la vie des utilisateurs pour tout ce qui touche aux matrices. Les matrices sont indispensables aux scientifiques, particulièrement aux mathématiciens et aux physiciens. Cette documentation n'a pas pour but de vous expliquer comment fonctionnent les matrices : si vous en avez besoin, nous supposons que vous savez ce que c'est. Nous nous contenterons donc d'un bref rappel sur l'utilisation des matrices puis nous aborderons les instructions qui leurs sont associées.

Rappel :

On emploie les matrices pour résoudre des systèmes d'équations du type :

- $3X + 4Y = 11$
- $6X - Y = 4$

Dans ce système, on s'en sort facilement en multipliant la première équation par 2 puis en lui soustrayant la 2ème afin d'annuler X. On obtient alors 6X+8Y−(6X−Y)=18.

D'où 9Y=18, c'est à dire Y=2, X=1.

Toutefois, il existe une autre méthode qui permet d'être généralisée à la résolution de systèmes de N équations à N inconnues. C'est ici qu'interviennent les matrices. En effet, le système précédent peut se mettre sous la forme :

3	4	11
6	−1	4

On reconnaît dans ces deux lignes les coefficients du système précédent.

Cette méthode donne :

$$X = \frac{-(4*4-11*-1)}{(3*-1-4*6)} = \frac{-27}{-27} = 1 \qquad Y = \frac{(3*4-11*6)}{(3*-1-4*6)} = \frac{-54}{-27} = 2$$

Nous avons (4*4–11*–1) qui représente le déterminant de la matrice :

$$\begin{vmatrix} 3 & 4 & 11 \\ 6 & -1 & 4 \end{vmatrix}$$

(3*–1–4*6) celui de la matrice :

$$\begin{vmatrix} 3 & 4 & 11 \\ 6 & -1 & 4 \end{vmatrix}$$ (il doit être non nul)

et enfin (3*4–11*6) celui de la matrice :

$$\begin{vmatrix} 3 & (\ 4) & 11 \\ 6 & (-1) & 4 \end{vmatrix}$$

Cette méthode présente 2 avantages:

- – Tout d'abord, c'est une méthode donc un algorithme potentiel d'où une très grande facilité de programmation (surtout quand les fonctions de calcul sur les matrices sont implémentées).
- – Ensuite, cette méthode est généralisable à N inconnues. Le gain devient évident lorsque l'on commence à se frotter aux systèmes de 3 équations à 3 inconnues.

Les matrices et l'OMIKRON :

Pour l'OMIKRON, les matrices ne sont que des tableaux bidimensionnels de nombres réels pour lesquels certaines opérations sont permises.

Il est donc impossible de réaliser une opération matricielle sur un tableau d'entiers. Les matrices sont obligatoirement des tableaux de réels simples ou doubles.

On représente les matrices de la manière suivante :

```
MAT A! (Ligne,Colonne)
```

D'autre part, certaines opérations comme le calcul du déterminant d'une matrice ou de sa matrice inverse ne sont possibles que lorsqu'il s'agit d'une matrice carrée (matrice où le nombre de lignes est égal au nombre de colonnes).

On peut toutefois se servir de ces fonctions sur des matrices non carrées à condition d'effectuer une restriction pour rendre la matrice carrée lors de l'appel de la fonction. Exemple :

```
DIM A!(8,5)
...
X = DET(A!(4,4)) :      REM Seule la partie du tableau s'étendant
                        REM de A!(0,0) à A!(4,4) sera prise en compte
                        REM par la fonction matricielle.
Y = DET(A!(5,5))
```

MAT CLEAR

*Syntaxe : **MAT CLEAR** <Tableau(<Dimension>[, <Dimension>]...)>*

L'OMIKRON possède une fonction matricielle particulièrement simple à comprendre : MAT CLEAR. Cette fonction efface les éléments d'une matrice. De plus, MAT CLEAR n'est pas réservée aux matrices et peut être appliquée à tous les tableaux, même à ceux de dimension supérieure à 2. Entre parenthèses, on doit indiquer les dimensions maximales du tableau. Pour les tableaux à une dimension, il est possible d'effacer uniquement une partie du tableau.

Exemple :

```
DIM A(4,4,3), B(100 )
....
MAT CLEAR A(4,4,3) : REM efface le tableau
MAT CLEAR B(50) : REM efface le tableau jusqu'à l'élément B(50)
MAT CLEAR A(4,2,2)
REM Erreur! Dans un tableau à plusieurs dimensions, le tableau
ne peut être effacé qu'en une seule fois!
```

MAT A!(N,N) =1

Syntaxe : MAT A!(<Ligne>, <Colonne>)=1

Cet ordre permet de générer une matrice unité (matrice carrée dont tous les éléments de la diagonale principale sont égaux à 1).

Exemple :

```
CLS
MAT A!(3,3) = 1
FOR I = 0 TO 3
    FOR J = 0 TO 3
      PRINT @(1+2 * I,1+8 * J); A!(I , J)
   NEXT  J
NEXT  I
```

Limites : Il est impossible d'obtenir une matrice unité à partir d'une matrice qui n'est pas carrée.

A ne pas faire :

```
MAT A!(4,2) = 1
? Appel de fonction illégal
```

MAT * X

*Syntaxe : MAT A!(<N>,<M>) * <Expression numerique>*

Cette fonction permet de multiplier une matrice par un nombre. Le résultat est retourné dans la matrice. Exemple :

Si A!(1,2)= $\begin{vmatrix} 1 & 2 & 3 \\ 3 & 0 & 4 \end{vmatrix}$ alors MAT A!(1,2)*5 provoque

$\begin{vmatrix} 5*1 & 5*2 & 5*3 \\ 5*3 & 5*0 & 5*4 \end{vmatrix}$ ce qui nous donne A!(1,2)= $\begin{vmatrix} 5 & 10 & 15 \\ 15 & 0 & 20 \end{vmatrix}$

MAT A! = B! * C!

*Syntaxe : MAT A!(<N>,<M>) =B!(<N>,<Q>) * C!(<Q>,<M>)*

Cette fonction permet de multiplier une matrice par une autre. A quoi cela correspond t–il ?

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} * \begin{vmatrix} 1 & 2 & 5 \\ 2 & 0 & 6 \end{vmatrix} = \begin{vmatrix} 1*1 + 2*2 & 1*2 + 2*0 & 1*5+2*6 \\ 3*1 + 4*2 & 3*2 + 4*0 & 3*5+4*6 \end{vmatrix}$$

Ce qui donne en définitive :

$$\begin{vmatrix} 5 & 2 & 17 \\ 11 & 6 & 39 \end{vmatrix}$$

Un autre exemple :

$$\begin{vmatrix} 1 & 7 \\ 4 & 2 \end{vmatrix} * \begin{vmatrix} 3 \\ 2 \end{vmatrix} = \begin{vmatrix} 1*3 + 7*2 \\ 4*3 + 2*2 \end{vmatrix} = \begin{vmatrix} 17 \\ 16 \end{vmatrix}$$

Comme on le voit, le nombre de lignes de la deuxième matrice doit être égal au nombre de colonnes de la première matrice. De plus, le nombre de lignes de la matrice résultante correspond à celui de la 1ère matrice et le nombre de colonnes à celui de la 2ème.

Attention : La multiplication de matrices n'est pas commutative :

matrice 1 * matrice 2 est différent de matrice 2 * matrice 1!

MAT INV

Syntaxe : MAT A!(<N>,<N>=INV B!(<N>,<N>)

MAT INV permet d'obtenir l'inverse d'une matrice à condition que celle–ci soit effectivement inversible.

MAT INV est une fonction particulièrement gourmande en temps. Pour inverser une matrice en réels simple précision de 100*100, il faut compter environ 12 minutes. C'est très rapide si on se représente le travail monstrueux nécessaire à l'inversion d'une telle matrice, mais c'est tout de même lent par rapport à des méthodes qui exploitent les particularités de certains types de matrices. En effet, l'OMIKRON est tout à fait incapable de comprendre à quel type de matrice il a affaire et il est obligé d'appliquer un algorithme général, ce qui n'est pas très performant.

Exemple :

```
INPUT "COMBIEN D'INCONNUES DANS VOTRE SYSTEME "; N#
N# = N# -1
DIM A! (N#,N#) : DIM C! (N#,0) : DIM X! (N#,N#)
Entree(N#)
MAT X! (N#,N#) = INV A! (N#,N#)
MAT Y! (N#,0) =X! (N#,N#) * C! (N#,0)
Sortie(N#)
END
'
DEF PROC Entree (A#)
    LOCAL  I%L , K#
    FOR I%L = 0 TO A# : FOR K# = 0 TO A#
      PRINT  "ELEMENT"; I%L; " "; K# : INPUT A! (I%L,K#)
    NEXT  K# : NEXT  I%L
    FOR  I%L = 0 TO A#
      PRINT "RESULTAT DE LA LIGNE "; I%L : INPUT C! (I%L,0)
    NEXT  I%L
RETURN
'
DEF PROC Sortie (A#)
    FOR  I%L = 0 TO A#
      PRINT Y! (I%L,0)
    NEXT
RETURN
```

Comme on peut le voir dans ce programme de résolution de systèmes d'équations, la saisie des données représente une bien plus grande partie du programme que la résolution en elle-même.

LIMITES : Cette fonction ne peut opérer que sur des matrices carrées et inversibles.

DET

Syntaxe : ***X= DET(A!(<N>,<N>))***

Cette fonction permet de calculer le déterminant d'une matrice.

Là encore, c'est une opération très gourmande en temps d'exécution. En effet, il faut un nombre considérable de calculs pour obtenir le déterminant d'une matrice d'ordre élevé.

Exemple:

```
CLS
A!(0,0) = 1 : A!(0,1) = 2
A!(1,0) = 3 : A!(1,1) = 4
B!(0,0) = 1 : B!(0,1) = 2 : B!(0,2) = 5
B!(1,0) = 2 : B!(1,1) = 0 : B!(1,2) = 6
X# = DET(A!(1,1))
Y# = DET(B!(1,1))
FOR I% = 0 TO 1
    FOR J% = 0 TO 1
      PRINT @(1+2*I%,1+8*J%);  A!(I%,J%)
    NEXT  J%
NEXT I%
PRINT @(5,2); "DET = "; X#
FOR I% = 0 TO 1
    FOR J% = 0 TO 2
      PRINT @(1+2 * I%,21+8 * J%); B !(I%,J%)
    NEXT  J%
NEXT I%
PRINT @(5,22); "Matrice restreinte :"
FOR I% = 0 TO 1
    FOR J% = 0 TO 1
      PRINT @(7+2 * I%,21+8 * J%); B!(I%,J%)
    NEXT  J%
NEXT I%
PRINT @(11,22); "DET =" ;Y#
```

LIMITES : Cette fonction n'opère bien entendu que sur des matrices carrées.

ATTENTION :

Les fonctions MAT INV et MAT DET réclament beaucoup de place en mémoire pour fonctionner. Si vous obtenez une erreur 'Plus de mémoire' dans une ligne contenant l'une de ces deux fonctions, alors vous ne pouvez pas les utiliser sur des matrices de cette taille. Par contre, le reste du programme peut quand même fonctionner.

LE SON

L'ATARI ST ne dispose pas de ce qui se fait de mieux en matière de co-processeur sonore. Le ST compense grâce à sa puissance et à sa mémoire. Voici une brève description technique du YAMAHA 2149:

- 3 canaux de sons indépendants (ce sont les voies)
- 1 générateur de bruit
- 1 générateur de courbe d'enveloppe pour les trois voies.

Le basic OMIKRON comprend trois instructions qui permettent de tirer parti du YM-2149.

TUNE

Syntaxe : TUNE <Voie>, <Hauteur> [,<Voie>,<Hauteur>...]

TUNE attribue une fréquence à une ou plusieurs voies. <Hauteur> ne représente pas directement la fréquence. En fait, <Hauteur> est proportionnel à la période. Donc, plus <Hauteur> est petit, plus la fréquence est élevée (son aigu) et plus <Hauteur> est grand, plus la fréquence est basse (son grave). La division de la hauteur par 2 fait passer à l'octave supérieure. Le son le plus grave est produit directement par une hauteur de 4095. Quand <Hauteur> est très faible, le son émis est tellement aigu qu'il est à peine perceptible. Pour une fréquence de 0, le basic OMIKRON cesse l'émission sonore sur la voie correspondante.

VOLUME

Syntaxe : VOLUME <Voie>,<Volume>
Syntaxe : VOLUME <Voie>,<Enveloppe>,<Periode_Oscillation>

La première syntaxe permet de fixer l'amplitude du son sur la voie correspondante. Cette amplitude est directement proportionnelle au paramètre <Volume>. Ainsi, pour <Volume>=15 on obtient l'amplitude maximale.

La seconde syntaxe permet quant à elle d'obtenir des effets sur le son à l'aide du générateur d'enveloppe. La période d'oscillation indique la durée de base entre deux crêtes de l'enveloppe. Pour obtenir une durée approximative d'une seconde, vous devez indiquer 8000 comme période d'oscillation.

ATTENTION : il n'y a qu'un seul générateur d'enveloppe pour les trois voix. Par conséquent, l'enveloppe active correspond à la dernière commande VOLUME (deuxième syntaxe) exécutée.

Les différentes formes d'enveloppes disponibles sont indiquées en annexe.

NOISE

Syntaxe : NOISE <Masque>, <Frequence>

NOISE permet de contrôler le générateur de bruit. Quand vous indiquez une fréquence pour le générateur de bruit, celle-ci n'est pas fixe. En effet, un bruit se différencie d'un son par ses variations de fréquence.

En fait, <Frequence> choisit l'une des 32 plages de fréquences disponibles, notées de 0 à 31. 0 indique une sonorité riche en aigus et 31 une sonorité riche en graves autour de la fréquence de base déterminée par TUNE.

<Masque> est un octet qui indique les voies concernées par NOISE.

Voix	Bit concerné	Valeur
1	bit 0	1
2	bit 1	2
3	bit 2	4

Donc, pour fixer NOISE sur les voies 1, 2 et 3, il faut donner la valeur 7 à <Masque>. Exemple :

```
PRINT "Oh! Le bel avion! "
TUNE 1,1,2,2000,3,4000
VOLUME 1,15 : VOLUME 2,15 : VOLUME 3,15
USING "### "
WHILE  MOUSEBUT = 0
    NOISE 7, MOUSEY \12.5 : PRINT @(10,10); MOUSEY \12.5
WEND
```

ACCES DIRECT EN MEMOIRE

Le BASIC OMIKRON dispose d'une batterie d'ordres à l'aide desquels on peut exploiter le 68000 et ses acolytes qui se trouvent sur la carte électronique du ST. Ce chapitre explique comment utiliser ces ordres. Si vous ne connaissez pas l'assembleur, il n'est pas certain que ce qui suit vous apparaîtra comme tout à fait clair.

POKE

Syntaxe : POKE <Adresse>,<Valeur>

De POKE, fureter, pousser. Ecrit le contenu <Valeur> à l'adresse <Adresse>. <Valeur> est un octet dont la valeur est comprise entre 0 et 255.

Exemple :

```
POKE 400000,255
REM La valeur 255 est écrite à l'adresse 400000.
```

WPOKE

Syntaxe : WPOKE <Adresse>, <Valeur>

De POKE, fureter, pousser. WPOKE signifie WORD POKE. Ecrit le contenu <Valeur> à l'adresse <Adresse>. <Valeur> est un entier 16 bits dont la valeur est comprise entre –32768 et 32768.

Exemple :

```
WPOKE $56000,$2A00
REM La valeur $2A00 est écrite à l'adresse $56000-$56001.
```

LIMITES : Il faut bien faire attention aux adresses qui doivent être paires pour être adressables avec WPOKE et LPOKE.

LPOKE

Syntaxe : LPOKE <Adresse>,<Valeur>

De POKE, fureter, pousser. LPOKE signifie LONG POKE. Ecrit le contenu <Valeur> à l'adresse <Adresse>. <Valeur> est un entier 32 bits dont la valeur est comprise entre –2147483658 et +2147483657.

Exemple :

```
LPOKE $56000, $12FEDCBA
REM Ecrit en $56000-$56003 la valeur $12FEDCBA.
```

LIMITES : Il faut bien faire attention aux adresses qui doivent être paires pour être adressables avec WPOKE et LPOKE.

PEEK

Syntaxe : PEEK(<Adresse>)

De PEEK, jeter un coup d'oeil. PEEK lit un octet à l'adresse indiquée.

Exemple :

```
PRINT PEEK(400000)
REM Affiche le contenu de ce qui se trouve à l'adresse 400000.
```

WPEEK

Syntaxe : WPEEK(<Adresse>)

De PEEK, jeter un coup d'oeil. WPEEK lit un mot (16 bits) à l'adresse indiquée.

Exemple :

```
? WPEEK($100)
REM Affiche le contenu de $100 et $101.
```

LIMITES : Il est, tout comme avec LPEEK, interdit d'utiliser des adresses impaires.

LPEEK

Syntaxe :(<Adresse>)

De PEEK, jeter un coup d'oeil. LPEEK lit un mot long (32 bits) à l'adresse indiquée.

Exemple :

```
A = WPEEK($44E)
```

A contient le contenu de $44E à $452. Il s'agit, en l'occurence de l'adresse physique de l'écran.

LIMITES : Il est, tout comme avec WPEEK, interdit d'utiliser des adresses impaires.

LE LANGAGE MACHINE.

Le BASIC OMIKRON a beau être rapide, les programmes écrits en langage machine sont exécutés directement par le processeur 68000 et n'ont pas besoin d'être d'abord traduits par un interpréteur ou un compilateur. Si vous voulez apprendre sérieusement la programmation en langage machine, nous vous recommandons de travailler avec un livre sur la programmation du 68000. Vous aurez aussi besoin d'un bon assembleur. Nous l'avons!

Il s'appelle OMIKRON ASSEMBLEUR (Nous contacter pour plus d'informations concernant ce produit).

Commençons par la commande de base :

CALL

Syntaxe : CALL <Variable> [(Parametre, Parametre...)]

Appelle un programme en langage machine dont l'adresse est spécifiée par la variable. Le programme machine doit forcément se terminer par l'instruction RTS. Si votre programme machine est buggé, vous obtiendrez une erreur d'adressage ou n'importe quelle erreur. Les paramètres sont optionnels. Si vous indiquez des paramètres, ils seront placés sur la pile au format C, le premier paramètre en haut, le dernier en bas. Le programme reçoit le premier paramètre avec MOVE.W 4(A7),DO, le deuxième avec 6(A7),etc...

Exemple :

```
Allo = MEMORY(20) : REM réserve 20 octets en mémoire
FOR I = Allo TO Allo+18  STEP 2
    READ A : WPOKE I,A
NEXT  I
CALL Allo(3,L LPEEK($44E))
REM Adresse de l'écran à l'adresse $44E
END
'
DATA . ..
DATA $4E75 : REM il s'agit de RTS
```

L'adresse doit toujours être passée par l'intermédiaire d'une variable. Il est interdit d'indiquer une adresse directement. En effet, il sera possible, dans un avenir plus ou moins proche, de relier ('LINKER') des programmes en OMIKRON et en C, PASCAL...

– ATTENTION – ATTENTION – ATTENTION – ATTENTION –

L'OMIKRON fonctionne toujours en mode •superviseur. Les programmes en langage machine appelés depuis l'OMIKRON par un CALL sont aussi en mode superviseur. Donc, par pitié, pas de passage en mode superviseur dans le programme en langage machine (plantage assuré). Le programme en langage machine peut modifier tous les registres, excepté la pile système et la pile utilisateur. De plus, vous trouverez l'adresse de SEGPTR dans A0.

DEF USR

Syntaxe : DEF USR = <Valeur>

Fixe l'adresse à laquelle un programme machine est appelé avec USR.

USR

Syntaxe : USR (<Valeur>)

La fonction USR appelle un programme en langage machine dont l'adresse a été fixée par DEF USR. Il est possible d'envoyer un paramètre entre parenthèses qui sera passé par le registre D0 (mot long). La valeur qui se trouve dans D0 après la fin du programme machine sera considérée comme la valeur de retour.

Les ordres qui vont maintenant vous être présentés ne servent pas uniquement à relier l'OMIKRON à des programmes en langage machine. Ils serviront aussi à celui qui veut aller plus loin avec le basic OMIKRON.

Mais prudence :

Les instructions qui sont passées en revue ici ne sont pas destinées aux débutants. Avant d'aborder ce chapitre, assurez–vous de posséder parfaitement le basic et apprenez, ne serait–ce que schématiquement, comment fonctionne le 68000 et le système de l'ATARI ST (TOS). Nous n'irons pas très loin dans les bidouilles dans cet ouvrage, mais vous pouvez toujours jetter un coup d'oeil dans le manuel de programmation.

SEGPTR

Syntaxe : SEGPTR

SEGPTR contient l'adresse de départ du tableau SEGment-PoinTeR. Le SEGMENT POINTER est un tableau qui contient les pointeurs nécessaires au fonctionnement du BASIC OMIKRON. Nous avons récapitulé ci-dessous les pointeurs qu'on trouve dans le SEGMENT POINTER. On calcule l'adresse d'un élément du SEGMENT POINTER par la formule suivante :

- **Adresse=LPEEK(SEGPTR+Décalage)**

Décalage	Zone mémoire
0	Tableau des numéros de ligne 1Mot=numéro de ligne, 1 Mot Long=pointeur relatif de la ligne
4	Début du programme BASIC
8	Tableau du pointeur des variables
12	Tableau des variables
16	Réservé
20	Tableaux
24	Tampon fichier
28	Chaînes
32	Libre
36	Pointeur de programme (pointant sur l'ordre exécuté)
40	Garbage-Top
44	Garbage-Bottom
48	Garbage-High
52	Valeur maximale de la pile (en dessous de 1K pour l'unité centrale)
56	Bas de la pile
60	La plus grande adresse de la mémoire (mise en place avec CLEAR)

VARPTR

Syntaxe :VARPTR(<Variable>)

La variable se trouve à l'adresse qui est retournée par la fonction VARPTR. Pour les entiers, le nombre est fixé à 2 ou 4 octets. Pour les réels courts viennent d'abord 15 bits pour l'exposant, suivi de 32 bits de mantisse. Pour les réels doubles, il y a 64 bits de mantisse.

Les chaînes sont construites comme suit : 1 pointeur 32 bits (relatif au Segment des chaînes LPEEK(VARPTR + 28), voir SEGPTR) sur la chaîne de caractères, 16 bits pour la longueur. La chaîne de caractères se termine par "nul" et s'arrête à l'adresse indiquée par les 4 octets qui se trouvent devant la chaîne.

MEMORY

Syntaxe : A= MEMORY(-1)
Syntaxe : A= MEMORY(<Expression_Numerique>)
Syntaxe : A= MEMORY(<Chaine>)

MEMORY réserve de la place en mémoire, par exemple pour stocker un programme machine ou pour des écrans graphiques. MEMORY(−1) retourne le nombre d'octets libres sur la mémoire réservée pour GEMDOS. MEMORY(<Expression_Numerique>) réserve le nombre demandé d'octets dans la mémoire laissée libre par le basic. Dans le cas où il n'y a plus suffisamment de mémoire, l'erreur "Pas assez de mémoire" est affichée. MEMORY(<Chaine>) fait quasiment la même chose. Elle réserve de la mémoire et place la chaîne spécifiée dans la zone mémoire qu'elle vient de réserver.

Exemple :

```
Adresse = MEMORY("allo") : PRINT Adresse
```

Les blocs de mémoire fixés par MEMORY sont libérés par CLEAR, RUN ou par une modification du programme.

Nouveau! Version 3.0 — **MEMORY_BLOCK**

Syntaxe : MEMORY_BLOCK <Numero_Bloc>, <Longueur>, <Variable>

Il s'agit d'une des plus importantes innovations de la version 3. MEMORY_BLOCK réserve un bloc de mémoire qui sera conservé lors de la sauvegarde du programme. Vous pourrez donc inclure une image, un programme en langage machine ou des données **directement** dans votre source basic. Quand vous chargerez le programme en basic, vous chargerez aussi le ou les blocs en mémoire. <Longueur> indique la taille du bloc.

<Variable> est la variable qui donne l'adresse où se trouve le bloc et permet d'y accéder.

LIMITES : On ne peut pas charger plus de 100 blocs dans un programme (soit les blocs numérotés de 0 à 99).

Nouveau! Version 3.0 — **MEMORY_MOVE**

Syntaxe : MEMORY_MOVE <Depart>, <Longueur> TO <Recept>

Déplace un bloc mémoire qui se situe à l'adresse <Depart> et dont la taille est <Longueur> à l'adresse <Recept>.

Exemple :

```
A = MEMORY(500) : B = MEMORY(500)
MEMORY_MOVE A,500 TO B
```

LIMITES : MEMORY_MOVE ne fonctionne qu'avec des adresses paires. En effet, il déplace des mots (16 bits). Si vous voulez effectuer des déplacements octet par octet, utilisez MEMORY_MOVEB.

Nouveau! Version 3.0 — **MEMORY_MOVEB**

Syntaxe : MEMORY_MOVEB <Depart>,<Longueur> TO <Recept>

Déplace un bloc mémoire qui se situe à l'adresse <Depart> et dont la taille est <Longueur> à l'adresse <Recept>. Les adresses impaires sont autorisées, puisque MOVEB déplace le bloc octet par octet.

Exemple :

```
A=MEMORY(501):B=MEMORY(501)
MEMORY_MOVEB A,501 TO B
```

Nouveau! version 3.0 **INLINE**

Syntaxe : INLINE <Chaine>

INLINE permet d'introduire du langage machine directement dans les programmes basic. Cette instruction n'est pas conseillée aux débutants! INLINE comprend le langage machine sous la forme d'une chaîne de caractères (en hexadécimal). Quand INLINE est rencontrée, le basic exécute le programme en langage machine indiqué par INLINE.

Exemple :

```
INLINE "A0094E71"
```

LIMITES : INLINE ne peut contenir plus de 124 octets de code. La chaîne ne peut donc pas dépasser 248 caractères (il s'agit en fait de la taille maximum d'une ligne).

LES ACCES AU SYSTEME

IMPORTANT : Lors de l'utilisation des fonctions système, l'OMIKRON doit convertir les nombres en entiers sur 32 ou 16 bits. Cette conversion peut provoquer des erreurs d'arrondis et des dépassements de capacité. Aussi, nous vous conseillons fortement de n'utiliser que des variables entières lors de l'appel de ces fonctions.

BIOS

Syntaxe : BIOS([Variable_Retour], <Numero_Fonction> [,Parametre]...)

BIOS appelle les fonctions BIOS (Basic Input/Output System) de l'ATARI ST. Pour utiliser le BIOS, lisez attentivement le manuel de programmation de l'OMIKRON.

Exemple :

```
Buffer = MEMORY(512)
Nombre = 1
Numero = 0
Lecteur = 0
BIOS (Erreur,4,0,L Buffer,Nombre,Numero,Drive)
IF Erreur>0 THEN ERROR 57+256*(-Erreur)
FOR I = Buffer TO Buffer+511
    PRINT  HEX$( PEEK(I));
NEXT  I
```

La fonction BIOS numéro 4 lit le boot sector de la disquette du lecteur A. En cas d'erreur, la variable de retour reçoit une valeur différente de 0. Les codes correspondent à des erreurs différentes selon les fonctions appelées. Tout ceci est détaillé dans le manuel de programmation de l'OMIKRON Basic.

LIMITES : L'appel incorrect de fonction BIOS peut 'planter' le système ; aussi, sauvegardez votre programme avant d'exécuter un programme contenant des appels BIOS, XBIOS, GEMDOS, AES et VDI.

XBIOS

Syntaxe : XBIOS([Variable_Retour], <Numero_Fonction> [,Parametre]...)

XBIOS appelle les fonctions XBIOS (eXtended Basic Input/Output System) de l'ATARI ST. Consultez le manuel de programmation de l'OMIKRON pour en savoir plus sur la gestion du XBIOS en OMIKRON.

Exemple :

```
REPEAT
FOR I = 1 TO 2
    XBIOS(,37) 'Pas de variable de retour mais la ',' est obligatoire.
    PALETTE NOT (WPEEK($FF8240))
NEXT I
UNTIL LEN( INKEY$ ) OR MOUSEBUT
```

La fonction 37 de XBIOS attend la synchronisation verticale de l'écran.

GEMDOS

Syntaxe:GEMDOS([Variable_Retour],<Numero_Fonction>[,Parametre]...)

GEMDOS appelle les fonctions GEMDOS (Graphics Environment Manager-Disk Operating System) de l'ATARI ST. Comme pour les autres fonctions système, reportez-vous au manuel de programmation de l'OMIKRON BASIC.

Exemple :

```
CLEAR : Place = FRE(0)+ MEMORY(-1)
CLEAR Place-4096 : REM libère le maximum de mémoire
Environ = MEMORY ("PATH= "+ CHR$(0)+"A:\" + CHR$(0) +
CHR$(255) )
Command = MEMORY( "")
Nom = MEMORY ("A\LOGO.PRG")
GEMDOS( ,$4B,0,L Nom,L Command,L Environ)
CLEAR 65536
```

Le programme charge LOGO.PRG qui se trouve dans le lecteur A puis l'efface de la mémoire.

VDI

Syntaxe : VDI(Code_Op, Contrl_Array, Intin_Array, Ptsin_Array, Intout_Array, Ptsout_Array)

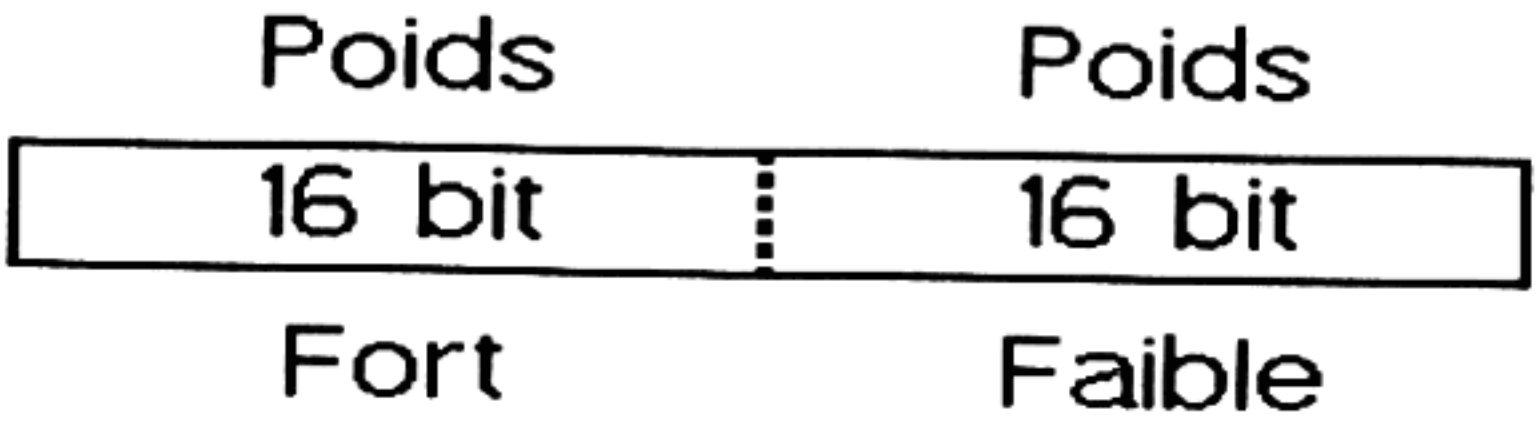
VDI appelle les fonctions GEM–VDI (Graphics Environment Manager–Virtual Device Interface) de l'ATARI ST. Pour plus d'explications et pour les exemples d'utilisation, consultez le manuel de programmation de l'OMIKRON Basic.

AES

Syntaxe : AES(Code_Op, Global_Array, Intin_Array, Addrin_Array, Intout_Array, Addrout_Array)

AES appelle les fonctions GEM–AES (Graphics Environment Manager–Application Environment Services). Pour plus d'explications et pour les exemples d'utilisation, regardez le manuel de programmation de l'OMIKRON Basic.

N'oubliez pas qu'avec GEM, les paramètres doivent être impérativement et sauf avis contraire passés par MOTS (16 bits). Dans le cas de variables de 32 bits (le format par défaut des nombres en OMIKRON), l'OMIKRON peut se charger de la conversion pour vous. Il vous suffit alors d'indiquer 'L Variable'. Il est possible que vous rencontriez ou que vous ayez vous même écrit des programmes contenant HIGH et LOW. Ces instructions étaient indispensables quand l'OMIKRON ne comprenait pas 'L'. Dans ce cas, il fallait décomposer les mots longs (entier 32 bits, soit deux mots de 16 bits) en HIGH(Variable), mot le plus significatif, et LOW(Variable), mot le moins significatif.



HIGH

Syntaxe : A=HIGH(B)

Retourne le poids fort de B et le place dans A.

Exemple :

```
B = $FF001234
PRINT HEX$ (HIGH(B))
FF00
OK
```

LOW

Syntaxe : A=LOW(B)

Retourne le poids faible de B et le place dans A. Cette fonction ne correspond pas à A=B AND $FFFF!

Exemple :

```
B = $67891055
PRINT HEX$ (LOW(B))
1055
OK
```

Nouveau! version 3.0

L

Syntaxe : L <Variable>

L retourne l'octet de poids fort et l'octet de poids faible d'une variable. L respecte le format Motorola, à savoir poids fort avant poids faible.

BLOAD

Syntaxe : BLOAD <Nom_Fichier>[,<Adresse>]

BLOAD charge le fichier à l'adresse indiquée ou bien sur l'écran si aucune adresse n'est spécifiée. Avec BLOAD, on peut donc reprendre des images qui ont été sauvegardées avec BSAVE ou qui ont été créées par DOODLE ou DRAW!. Vous trouverez dans le manuel de programmation une routine permettant de charger n'importe quelle image (STAD, DEGAS ELITE, GEM...).

BSAVE

Syntaxe : BSAVE <Nom_Fichier>[,<Adresse>, <Longueur>]

BSAVE sauvegarde soit le nombre d'octets <longueur> à partir de l'adresse indiquée, soit, quand ce paramètre manque, l'écran complet (32000 octets).

IPL

Syntaxe : IPL <Expression_Numerique>

IPL = Interrupt Priority Level (Niveau de priorité des interruptions). IPL permet de fixer le niveau d'interruption du MC 68000. Le niveau d'interruption s'étend de 0 (le plus faible) à 7 (le plus fort). En standard, l'OMIKRON est en mode 3. Voici une liste restreinte des fonctions système selon le niveau d'interruption :

IPL 1	Horizontal Blank Interrupt
IPL 3	Standard
IPL 4	Vertical Blank Interrupt Si vous utilisez IPL 4, l'écran n'est plus remis à jour, la souris ne fonctionne plus... en échange, la vitesse augmente d'environ 2%.
IPL 6	Plus de clavier, plus de RS–232, plus de WAIT et plus de ON TIMER...GOSUB. Mais c'est plus rapide!!!
IPL 7	Niveau d'interruption maximum.

WVBL

Syntaxe : WVBL

Wait for Vertical Blank Interrupt. WVBL attend le retour de trame (balayage en fin d'écran) qui a lieu tout les 1/50 à 1/70 ème de seconde, selon la résolution choisie. WVBL permet en théorie d'obtenir des images parfaitement stables, sans aucun tremblement ni clignotement.

LES BIBLIOTHEQUES

Les bibliothèques existent sous deux formes en basic OMIKRON.

La première forme est connu de tous. Il s'agit simplement de charger un programme en ASCII à l'aide de MERGE. Ce type de bibliothèque existe toujours en OMIKRON.

La deuxième forme est plus sophistiquée et nous a demandé beaucoup d'efforts. La bibliothèque est codée sous un format particulier, proche du code F.I.T. Ceci permet un chargement rapide de la bibliothèque, un listing très réduit et la confidentialité du code de la bibliothèque. L'appel des procédures appartenant à une bibliothèque codée est le même qu'avec n'importe quelle procédure. Une bibliothèque fournie sur la disquette utilise ce codage. Il s'agit de GEM.LIB. Nous allons vous expliquer comment charger GEM.LIB en utilisant la fonction LIBRARY.

LIBRARY

Syntaxe : LIBRARY <Proc>,<Chemin_Acces>

LIBRARY charge une bibliothèque codée portant le nom <Proc> et dont le chemin d'accès est indiqué par <Chemin_Acces>. Lors de la première exécution du programme, l'OMIKRON chargera la biliothèque depuis le disque, puis l'initialisera. A compter de ce moment, la bibliothèque est active et l'OMIKRON n'a plus besoin de la recharger à chaque 'RUN'. Par contre, un 'NEW' ou un 'LOAD' efface la ou les bibliothèques présentes en mémoire.

Exemple:

Au début du programme, tapez la ligne suivante:

```
10 LIBRARY GEM, "A:\GEM\GEM.LIB "
```

Charge la bibliothèque GEM qui se trouve dans le dossier GEM sur le lecteur A: et dont le nom est GEM.LIB.

En initialisant la bibliothèque, l'OMIKRON se positionne sur la dernière ligne du programme et ajoute la ligne suivante :

LIBRARY CODE

LIBRARY CODE n'est pas à proprement parler un 'token', un mot–clé. Il s'agit plutôt d'un repère destiné à vous, utilisateur. Vous ne devez en aucun cas modifier ou détruire la ligne contenant LIBRARY CODE, ce qui effacerait la bibliothèque. En effet, il faut savoir que la bibliothèque se trouve dans la ligne LIBRARY CODE!!! On ne la voit pas car une autre instruction est utilisée : il s'agit de **UNLIST**.

UNLIST interdit la visualisation du reste de la ligne. Comme LIBRARY CODE, ce n'est pas un 'token'. L'OMIKRON l'ajoute de lui–même, mais vous n'êtes pas autorisé à l'entrer.

INSTRUCTIONS DIVERSES

HCOPY

Syntaxe : HCOPY

HCOPY = Hard Copy. HCOPY imprime le contenu de l'écran sur l'imprimante connectée au ST. HCOPY correspond en fait à la fonction système [ALT]–[Help]. Si le résultat n'est pas correct sur votre imprimante, il s'agit sans doute d'une mauvaise configuration. Dans ce cas :

- Chargez l'accessoire CONTROL.ACC qui se trouve sur la disquette système fournie avec l'ATARI (pour charger un accessoire, il faut remettre en route l'ordinateur).
- Placez–vous sur l'accessoire 'Panneau de contrôle' et cliquez sur 'Imprimante'. Réglez ensuite les paramètres d'impression.

Nous vous proposons une autre méthode dans le manuel de programmation qui consiste à fabriquer votre propre 'driver' d'imprimante.

HCOPY TEXT

Syntaxe : HCOPY TEXT

HCOPY TEXT imprime le contenu de l'écran 0 sur l'imprimante. Contrairement à HCOPY, HCOPY TEXT n'imprime pas de graphisme mais envoie simplement le code ASCII des caractères présents sur l'écran 0. L'impression est donc beaucoup plus rapide.

MODE LPRINT

Syntaxe : MODE LPRINT <F>

MODE LPRINT remplit la même fonction que MODE "F", MODE "U"...

Avec MODE LPRINT "F" les caractères ê, à, ç, é, è, ù, ô, î, ë qui apparaissent à l'écran sont transposés de manière à correspondre aux codes des imprimantes EPSON et compatibles EPSON. La transposition n'existe qu'avec LPRINT. Les autres fonctions qui pourraient accéder à l'imprimante ne sont pas modifiées.

CLS

Syntaxe : CLS

CLS efface l'écran. On peut aussi taper :

```
PRINT CHR$(27); "E"
```

WAIT

Syntaxe : WAIT <Secondes>

WAIT est à utiliser quand une temporisation est nécessaire. On est amené à temporiser un programme quand la vitesse d'exécution est trop rapide. En général, on rencontre ce problème avec les périphériques (la prise série, les prises MIDI...), qui sont plus lents à transmettre ou à recevoir que le programme. WAIT fonctionne au 1/200 ème de seconde, ce qui est très largement suffisant pour les applications courantes, compte tenu du système de l'ATARI ST. Si votre but est de réaliser des applications en temps réel, vous ne pourrez de toute façon sans doute pas utiliser le TOS qui est trop lent. Il vous faudra passer à OS-9, par exemple.

Nouveau! Version 3.0

JOYSTICK

Syntaxe : A = JOYSTICK(X)

JOYSTICK retourne l'état du port joystick dans la variable A. Les données sont retournées sur un octet, dans l'ordre suivant :

Bit 7	Bouton de tir (1=enclenché)
Bit 0-3	Directions de la manette.

La variable X peut contenir la valeur 0 ou 1, ce qui indique à l'instruction le port choisi pour lire le joystick (manette de jeu).

On voit que, au niveau logiciel du moins, le joystick peut être déplacé dans les quatres directions à la fois!!!

Nouveau! Version 3.0 — **COMPILER**

Syntaxe : IF COMPILER THEN

COMPILER teste si le programme est compilé ou non. Si le programme est compilé, la variable COMPILER contient –1. Dans un programme interprété, COMPILER est à 0.

Exemple :

```
IF COMPILER THEN PRINT "Programme compilé"
ELSE PRINT  "Programme non  compilé"
ENDIF
```

REM

Syntaxe : REM <texte>

Pour éviter qu'un programme ne ressemble à des hiéroglyphes égyptiens, il est nécessaire de le commenter. Pour commenter un programme, le basic dispose d'une instruction spécifique, l'instruction REM. REM permet l'introduction de remarques (ou commentaires) dans un programme. Il est possible de remplacer REM par ' (apostrophe). L'avantage de ' est qu'on peut l'insérer n'importe où dans la ligne sans la faire précéder du : obligatoire avec REM.

Quand le basic tombe sur un REM, il passe directement à la ligne suivante sans regarder ce qui suit le REM.

Exemple :

```
Quant = 150 : REM On peut en mettre plus si nécessaire
IF Masse2! < 1E-7 GOTO 500
REM C'est une ligne sans instruction
REM On peut y mettre toutes sortes de choses
```

Si possible, il faut expliquer ce que fait la ligne et non pas décrire la ligne d'instructions

A oublier :

```
LINE STYLE = 3 : REM Fixe le style des lignes
```

Commentaire judicieux :

```
LINE STYLE = 3 : REM Lignes en pointillées
```

LE STANDARD VT-52

Le VT-52 permet de commander un terminal grâce à des combinaisons de caractères. Toutes les fonctions VT-52 ont été implémentées dans le système d'exploitation de l'ATARI : TOS. L'OMIKRON comprend ces fonctions VT-52. Ce qui signifie qu'il vous suffit de taper 'PRINT CHR$(27);''E''' pour effacer l'écran.

Voici une liste des fonctions VT-52.

CHR$(7)	Bip sonore.
CHR$(8)	Curseur une colonne vers la gauche.
CHR$(9)	Curseur huit colonnes à droite (équivaut à TAB).
CHR$(10)	Curseur vers le bas (aller à la ligne).
CHR$(11)	Curseur vers le bas (aller à la ligne).
CHR$(12)	Curseur vers le bas (aller à la ligne).
CHR$(13)	Curseur vers la marge gauche (Retour charriot).

Les séquences 'échappement':

Certaines fonctions VT-52 demandent à ce que le caractère ESCAPE (27) précède la fonction proprement dite.

IMPORTANT: Vous devez respecter les minuscules/majuscules.

Pour entrer une commande VT-52, tapez 'PRINT CHR$(27);''X''' où X représente l'un des caractères suivants :

ESC A	Curseur une ligne au dessus.
ESC B	Curseur une ligne en dessous.
ESC C	Curseur une colonne à droite.
ESC D	Curseur une colonne à gauche.
ESC E	Efface l'écran (= CLS).
ESC H	Curseur dans le coin supérieur gauche.
ESC I	Curseur une ligne au dessous (si nécessaire, décale l'écran vers le bas d'une ligne).
ESC J	Efface l'écran à partir du curseur.
ESC K	Efface la ligne à partir du curseur.
ESC L	Insère une ligne.
ESC M	Efface une ligne.

ESC Y l,c	Positionne le curseur. l = Numéro de ligne + 32. c = Numéro de colonne + 32.
ESC b col	Fixe la couleur de l'écriture : col= couleur.
ESC c col	Fixe la couleur de l'arrière-plan : col = couleur.
ESC d	Efface l'écran jusqu'au curseur.
ESC e	Allume le curseur.
ESC f	Eteint le curseur.
ESC j	Sauvegarde la position du curseur.
ESC k	Reprend la position sauvegardée du curseur.
ESC l	Efface la ligne.
ESC o	Efface la ligne jusqu'au curseur.
ESC p	Passe en inverse vidéo.
ESC q	Annule l'inverse vidéo.
ESC v	Met en route le suivi automatique (le curseur continue en colonne 0 de la ligne suivante après la colonne 79).
ESC w	Stoppe le suivi automatique.

ESC v et ESC w ne sont pas possibles sur l'écran 0. Par contre, les caractères spéciaux suivants ne fonctionnent qu'avec l'écran 0.

CHR$(1)	Mode ASCII.
CHR$(2)	Insère un caractère.
CHR$(3)	Efface un caractère.
CHR$(4)	Place le curseur en haut à gauche de la fenêtre.
CHR$(5)	Place le curseur en bas à droite de la fenêtre.
CHR$(8)	Déplace le curseur vers la gauche et efface un caractère.

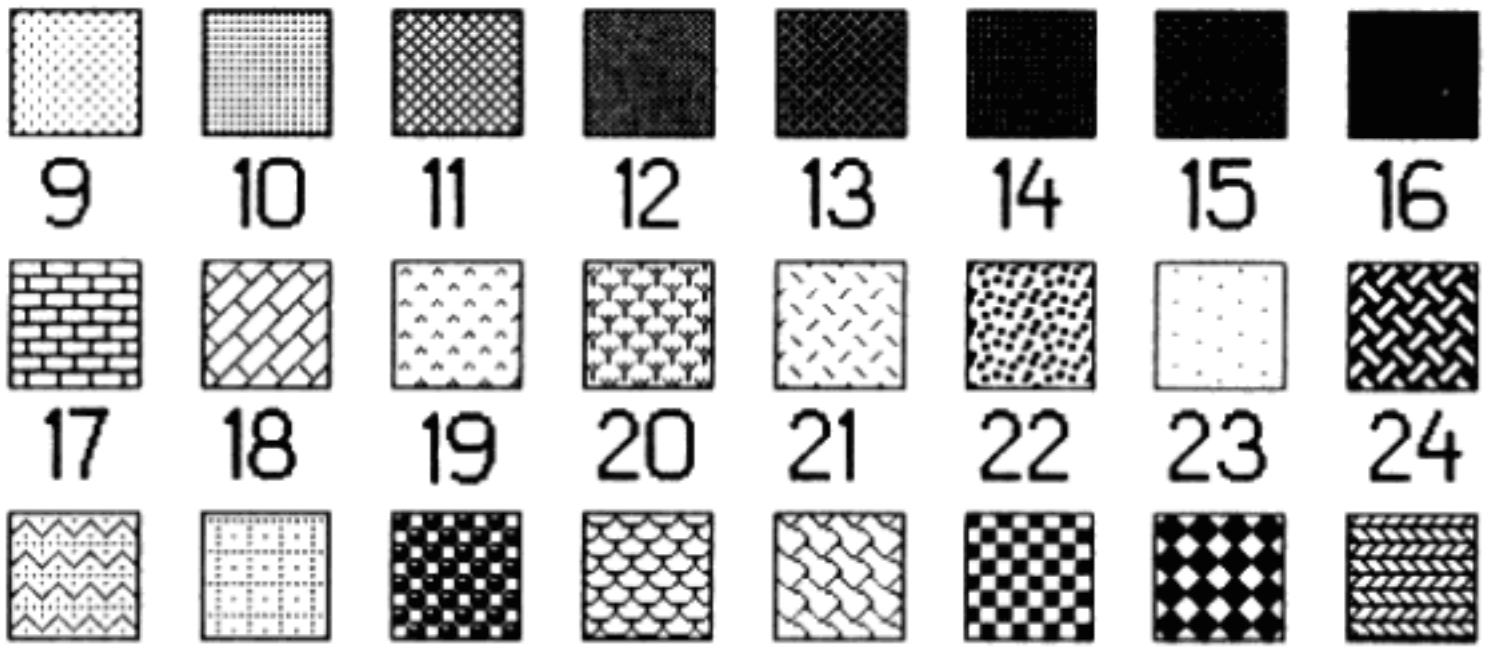
Accents: Après la frappe d'un accent, le curseur reste immobile. Le caractère suivant est combiné avec l'accent quand c'est possible.

EQUIVALENCE ENTRE COULEURS VDI ET HARDWARE

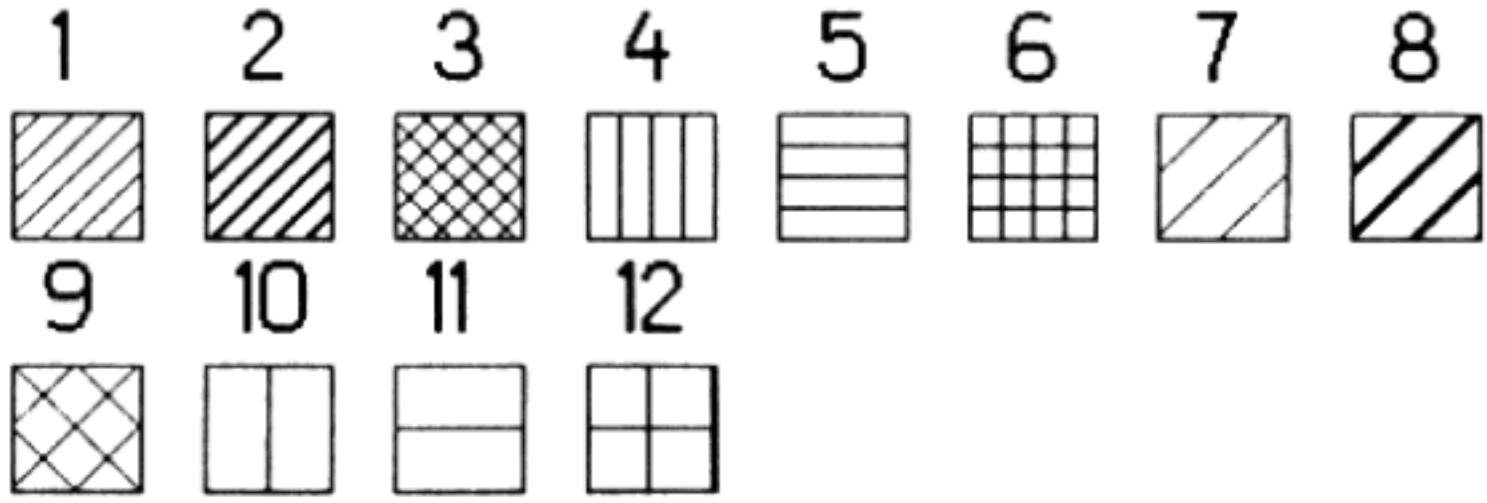
SYSTEME	HARD		
Couleurs VDI	Basse	Moyenne	Haute
0	0	0	0
1	15	3	1
2	1	1	
3	2	2	
4	4		
5	6		
6	3		
7	5		
8	7		
9	8		
10	9		
11	10		
12	12		
13	14		
14	11		
15	13		

STYLES DE REMPLISSAGE

2.



3.



LES MESSAGES D'ERREURS TOS

Numéro d'erreur	Signification
1	Erreur générale
2	Lecteur de disquette pas prêt
3	Ordre inconnu
4	Erreur de checksum
5	Exécution impossible
6	Trace introuvable
7	Boot–sector illisible
8	Secteur introuvable
9	Plus de papier
10	Erreur d'écriture
11	Erreur de lecture
12	Erreur générale
13	Disque plein
14	Le disque doit être changé
15	Ordre inconnu
16	Secteurs mal écrits ("Verify Error")
17	Pas de disquette dans le lecteur
32	Numéro de fonction invalide
33	Programme/fichier introuvable
34	Chemin d'accès introuvable
35	Trop de fichiers ouverts, impossible d'en ouvrir d'autres
36	Utilisation impossible
37	Numéro de fichier invalide
39	Plus assez de mémoire
40	Aucun bloc de mémoire à cette adresse
46	Nom de lecteur invalide
49	Il n'existe plus d'autres fichiers

TYPES DE FICHIERS

Le type de fichier est spécifié par le paramètre qui suit l'ordre OPEN.

Le type de fichier peut être :

"I" "Input" Fichier séquentiel sur disque ; uniquement en lecture. Exemple :

```
OPEN "I", 1, "A:\DESKTOP.INF "
```

"O" "Output" Fichier séquentiel sur disque ; uniquement en écriture. Exemple :

```
OPEN "O", 1, "ADRESSE.DAT "
```

"A" "Append" Comme "Output", mais conserve les données déjà existantes. Exemple :

```
OPEN "A", 1, "ADRESSE.DAT "
```

"R" "Random" Accès Direct sur disque. Exemple :

```
OPEN "R", 1, "STAMM.DAT ",142 : FIELD , ...
```

"F" "Files" Fichier directory. Exemple :

```
OPEN "F", 1, "A:\*.* ",63.
```

Le nombre 63 est un masque qui indique le type des fichiers sélectionnés. A chaque bit du masque correspond un type de fichier. Voici la liste des types de fichiers :

Bit 0	1	Fichiers protégés en écriture
Bit 1	2	Fichiers partagés
Bit 2	4	Fichiers système
Bit 3	8	Nom de la disquette
Bit 4	16	Sous–répertoire
Bit 5	32	(non utilisé par le GEM DOS).

Exemple : Regardez le programme FILES sur la disquette originale.

"P" "Printer" Imprime sur papier un fichier séquentiel. Exemple :

```
OPEN "P",1
```

OPEN "V" pour "V24" Ouvre un fichier RS232. Exemple :

```
OPEN "V",1
```

OPEN "M" pour "MIDI" Ouvre un fichier MIDI. Exemple :

```
OPEN "M",1
```

OPEN "K" pour "Keyboard" Permet l'envoi de commandes sur le processeur clavier. Exemple :

```
OPEN "K",1 : PRINT#1, CHR$(10); CHR$(8); CHR$(16);
CLOSE 1
```

ATTENTION : Après OPEN "K",1, toutes les commandes que le processeur clavier envoie (c'est-à-dire tous les mouvements de la souris, chaque pression sur les touches), sont lus par INKEY$ ou INPUT$. CLOSE 1 ainsi que CLEAR, END, etc annulent cet état. L'entrée par le clavier ne fonctionne donc plus quand vous lancez OPEN"K" en mode direct ou quand vous oubliez CLOSE 1. Un seul remède : RESET.

LISTE DES MESSAGES D'ERREURS.

Liste des erreurs classées par ordre numérique.

Les explications concernant l'erreur sont fournies en dessous du message. Dans tout les cas, nous vous conseillons de lire la partie expliquant la correction des erreurs dans le manuel de programmation. Les erreurs les plus courantes comme les plus délicates y sont passées en revue.

1 Structure trop longue

Une structure contient plus de 64 Ko de programme (une structure peut être une boucle FOR...NEXT, WHILE...WEND, REPEAT...UNTIL).

2 Erreur de syntaxe

Sans commentaire.

3 RETURN sans GOSUB

L'OMIKRON a rencontré l'ordre RETURN sans trouver sa contrepartie (DEF PROC, DEF FN ou GOSUB). Exemple :

```
100  GOSUB Entre : PRINT "Trois fois ";A;" égal ";3*A
300  -Entre : INPUT "Donnez-moi un nombre, patron";A
500  RETURN
```

4 Plus de DATA

Un ordre READ ne trouve plus de DATA.

5 Appel illégal de fonction

Une fonction est utilisée d'une manière qui n'est pas correcte. Essayez une autre fonction ou vérifiez les paramètres.

6 Dépassement

L'OMIKRON dispose d'un domaine de calcul qui s'étend de -5,11E-4931 à +5,11E+4931. Si le message 6 s'affiche, c'est que vous avez dépassé ces limites. Si le programme gère vos économies, permettez-nous de vous féliciter!

7 Plus de mémoire

Deux possibilités:

- Plus de place en mémoire. Programme trop grand ou tableaux trop largement dimensionnés.
- Plus de place sur la pile processeur (STACK). La pile est utilisée par :
  - Les boucles ouvertes.
  - Les appels de sous-programmes.
  - Les appels de procédures/fonctions
  - Certains ordres (SORT...)

8 Commande inconnue

La ligne ou l'étiquette désignée n'existe pas.

- GOTO 100. Il n'y a pas de ligne 100
- LISTAGE (NOM$, 10). LISTAGE n'existe pas ou les paramètres sont incorrects.

9 Accès hors limites

Vous tentez d'accéder à un élément qui se situe en dehors des limites d'un tableau.

```
10 DIM A(5)
20 I = 20
30 PRINT A(I)
40 END
50 REM A(20) est trop grand par rapport à DIM A(5)
```

10 Définition multiple

Une procédure est définie plusieurs fois.

```
100  DEF PROC EAZYGEM : RETURN
110  DEF PROC EAZYGEM : PRINT "ICI EAZYGEM" : RETURN
```

11 Division par zéro

L'OMIKRON vient de tomber sur une division par zéro, ce qui le met dans tous ses états.

12 Accès direct illégal

Ordre non exécutable directement.

```
PRINT "DIRECT "
DEF PROC DIRECT : RETURN
REM DEF PROC doit toujours être appelé.
```

13 Confusion de type

Le type de variable de convient pas.

Exemples :

```
A$ = 2
PRINT CHR$(A$)
```

14 RETURN sans fonction

Correspond à l'erreur 'RETURN sans GOSUB' pour les fonctions à plusieurs lignes.

15 Chaîne trop longue

Une chaîne alphanumérique ne peut dépasser la taille (respectable) de 32000 caractères.

16 Formule trop complexe

La formule exprimée dépasse la capacité de l'OMIKRON.

17 Impossible de continuer

Si vous avez modifié le programme après un [CTRL]–[C] ou bien si une erreur est venue interrompre le bon déroulement du programme, il n'est plus possible d'exécuter l'ordre CONT.

18 Fonction utilisateur non définie

La fonction appelée n'existe pas.

19 Pas de RESUME

Sans commentaire.

20 RESUME sans erreur

L'ordre RESUME est exécuté sans qu'une erreur ait eu lieu.

21 Utilisez EXIT

Certains basics font des saletés plein partout.

```
100  FOR T = 1 TO 10
110  INPUT "La bourse ou la vie ? <O>ui ou <N>on ";Rep$
120  IF UPPER$(REP$) = "O" THEN T = 11 : NEXT T
130  NEXT T
```

Franchement, c'est laid. Et c'est interdit en OMIKRON.

22 Opérande manquant

Il vous manque un opérande dans la ligne.

```
100  DATA 1, 2, : REM Il manque un opérande après la virgule
```

23 Ligne pleine

Une ligne contient au maximum 255 caractères. La vôtre en contient plus.

24 REPEAT sans UNTIL

Une boucle REPEAT ne trouve pas de UNTIL.

25 UNTIL sans REPEAT

Il manque un REPEAT à un UNTIL.

26 FOR sans NEXT

Une boucle FOR ne trouve pas de NEXT.

27 NEXT sans FOR

Il manque un FOR à un NEXT.

28 IF sans THEN ou ENDIF

Deux possibilités se présentent :

- Il manque un THEN à un IF.
- Il manque un ENDIF à un IF à plusieurs lignes.

```
90    -Deb
100   IF Prix > 990
110   THEN PRINT "Pas donné"
120   ELSE PRINT "C'est bon marché "
130   GOTO Deb
```

29 WHILE sans WEND

Une boucle WHILE ne trouve pas de WEND

30 WEND sans WHILE

Il manque un WHILE à un WEND

31 THEN, ELSE ou ENDIF sans IF ou THEN

Il manque un IF ou un THEN à un THEN, ELSE ou ENDIF.

33 RESET

Vous avez appuyé sur le bouton RESET. Mais rien de grave si le système n'est pas déjà planté, car l'OMIKRON conserve votre programme en mémoire.

34 Erreur Bus

Un programme en langage machine vient d'effectuer une très belle erreur bus.

35 Erreur d'adressage

Un programme en langage machine vous fait une crise d'erreur d'adressage.

36 Opcode inconnu

Un programme en langage machine contient une instruction que le 68000 ne connaît pas.

37 Division par zéro

Un programme en langage machine essaye de diviser par zéro un nombre (DIVU ou DIVS). Il n'y arrivera pas!

38 Erreur CHK

Une exception CHK vient d'être exécutée dans un programme en langage machine.

39 Erreur TRAPV

Un programme en langage machine vient de trouver une exception "Trap on overflow".

40 Violation de privilège

Un programme en langage machine utilise une instruction en mode UTILISATEUR. L'OMIKRON fonctionne toujours en mode SUPERVISEUR.

Il faut donc toujours enlever les passages en mode SUPERVISEUR puisque vous y êtes déjà.

41 Bit trace allumé

Un programme en langage machine est en mode TRACE–BIT.

42 Exception XXX

Un programme en langage machine vient d'exécuter une exception numéro XXX.

45 EXIT sans structure

EXIT a été exécuté, mais aucune boucle n'est ouverte, aucun sous–programme n'est en route, aucune routine d'erreur n'est en cours. Alors, EXIT d'où ?

46 Utilisez EXIT TO dans les fonctions

Une fonction à plusieurs lignes ne doit pas utiliser un simple EXIT. Prenez un EXIT TO.

47 Matrice irrégulière

La matrice à inverser par MAT INV n'est pas régulière et n'a donc pas d'inverse.

50 Dépassement FIELD

Le FIELD en question dépasse la limite: la cpacité maximale de son tampon est dépassée.

52 Mauvais numéro de fichier

Les numéros de fichiers sont compris entre 1 et 16. En dessous ou au–delà, c'est impossible.

53 Fichier introuvable

Le fichier demandé n'existe pas sur le chemin indiqué ou sur le chemin par défaut.

54 Mauvais mode d'accès

Le mode d'accès du fichier spécifié lors de l'ouverture ne convient pas pour l'opération indiquée.

55 Fichier déjà ouvert

Tentative d'ouverture d'un fichier déjà ouvert (on ne peut pas!)

56 Fichier fermé

Un fichier doit absolument être ouvert avant qu'on puisse faire quoi que ce soit dessus (Si, si!!).

57 Erreur TOS #XX

Le système annonce qu'une erreur système s'est produite.

61 Disque plein

Il n'y a plus la place nécessaire sur le disque pour la bonne exécution de l'instruction.

62 INPUT après fin

Tentative de lecture d'un fichier séquentiel après la fin de celui-ci.

63 Mauvais numéro d'enregistrement

Dans un fichier en accès relatif, on a tenté de se servir d'un enregistrement qui dépasse les capacités de GEMDOS.

64 Mauvais nom de fichier

Des caractères interdits ont été utilisés dans un nom de fichier.

66 Pas de numéros de lignes

Il manque des numéros de lignes à un programme en ASCII chargé par LOAD. Il faut utiliser 'LOAD BLOCK' ([Shift]-[F8]).

67 Trop de fichiers

GEMDOS n'accepte qu'un certain nombre de fichiers sur une disquette. Au-delà, il faut reprendre une autre disquette ou effacer des fichiers.

LISTE DES MESSAGES D'ERREURS.

Liste des erreurs classées par ordre alphabétique.

Les explications concernant l'erreur sont fournies en dessous du message. Dans tout les cas, nous vous conseillons de lire la partie expliquant la correction des erreurs dans le manuel de programmation. Les erreurs les plus courantes comme les plus délicates y sont passées en revue.

12 Accès direct illégal

Ordre non exécutable directement.

```
PRINT "DIRECT "
DEF PROC DIRECT : RETURN
REM DEF PROC doit toujours être appelé.
```

9 Accès hors limites

Vous tentez d'accéder à un élément qui se situe en dehors des limites d'un tableau.

```
10 DIM A(5)
20 I = 20
30 PRINT A(I)
```

5 Appel illégal de fonction

Une fonction est utilisée d'une manière qui n'est pas correcte. Essayez une autre fonction ou vérifiez les paramètres.

41 Bit trace allumé

Un programme en langage machine est en mode TRACE-BIT.

15 Chaîne trop longue

Une chaîne alphanumérique ne peut dépasser la taille (respectable) de 32000 caractères.

8 Commande inconnue

La ligne ou l'étiquette désignée n'existe pas.

- GOTO 100. Il n'y a pas de ligne 100
- LISTAGE (NOM$, 10). LISTAGE n'existe pas ou les paramètres sont incorrects.

13 Confusion de type

Le type de variable de convient pas.

Exemples :

```
A$ = 2
PRINT CHR$(A$)
```

10 Définition multiple

Une procédure est définie plusieurs fois.

```
100  DEF PROC EAZYGEM : RETURN
110  DEF PROC EAZYGEM : PRINT "ICI EAZYGEM " : RETURN
```

6 Dépassement

L'OMIKRON dispose d'un domaine de calcul qui s'étend de 5,11E-4931 à +5,11E+4931. Si le message 6 s'affiche, c'est que vous avez dépassé ces limites. Si le programme gère vos économies, permettez-nous de vous féliciter!

50 Dépassement FIELD

Le FIELD en question dépasse la limite : la capacité maximale de son tampon est dépassée.

61 Disque plein

Il n'y a plus la place nécessaire sur le disque pour la bonne exécution de l'instruction.

11 Division par zéro

L'OMIKRON vient de tomber sur une division par zéro, ce qui le met dans tous ses états.

37 Division par zéro

Un programme en langage machine essaye de diviser par zéro un nombre (DIVU ou DIVS). Il n'y arrivera pas!

34 Erreur Bus

Un programme en langage machine vient d'effectuer une très belle erreur bus.

35 Erreur d'adressage

Un programme en langage machine vous fait une crise d'erreur d'adressage.

2 Erreur de syntaxe

Sans commentaire.

38 Erreur CHK

Une exception CHK vient d'être exécutée dans un programme en langage machine.

57 Erreur TOS #XX

Le système annonce qu'une erreur système s'est produite.

39 Erreur TRAPV

Un programme en langage machine vient de trouver une exception "Trap on overflow".

42 Exception XXX

Un programme en langage machine vient d'exécuter l'exception numéro XXX.

45 EXIT sans structure

EXIT a été exécuté, mais aucune boucle n'est ouverte, aucun sous-programme n'est en route, aucune routine d'erreur n'est en cours. Alors, EXIT d'où ?

55 Fichier déjà ouvert

Tentative d'ouverture d'un fichier déjà ouvert (on ne peut pas!)

56 Fichier fermé

Un fichier doit absolument être ouvert avant qu'on puisse faire quoi que ce soit dessus (Si, si!!).

53 Fichier introuvable

Le fichier demandé n'existe pas sur le chemin indiqué ou sur le chemin par défaut.

26 FOR sans NEXT

Une boucle FOR ne trouve pas de NEXT.

16 Formule trop complexe

La formule exprimée dépasse la capacité de l'OMIKRON.

18 Fonction utilisateur non définie

La fonction appelée n'existe pas.

28 IF sans THEN ou ENDIF

Deux possibilités se présentent :

- Il manque un THEN à un IF.
- Il manque un ENDIF à un IF à plusieurs lignes.

```
90   -Deb
100  IF Prix > 990
110  THEN PRINT "Pas donné"
120  ELSE PRINT "C'est bon marché "
130  GOTO Deb
```

17 Impossible de continuer

Si vous avez modifié le programme après un [CTRL]–[C] ou bien si une erreur est venue interrompre le bon déroulement du programme, il n'est plus possible d'exécuter l'ordre CONT.

62 INPUT après fin

Tentative de lecture d'un fichier séquentiel après la fin de celui–ci.

23 Ligne pleine

Une ligne contient au maximum 255 caractères. La vôtre en contient plus.

47 Matrice irrégulière

La matrice à inverser par MAT INV n'est pas régulière et n'a donc pas d'inverse.

54 Mauvais mode d'accès

Le mode d'accès du fichier spécifié lors de l'ouverture ne convient pas pour l'opération indiquée.

63 Mauvais numéro d'enregistrement

Dans un fichier en accès relatif, on a tenté de se servir d'un enregistrement qui dépasse les capacités de GEMDOS.

64 Mauvais nom de fichier

Des caractères interdits ont été utilisés dans un nom de fichier.

52 Mauvais numéro de fichier

Les numéros de fichiers sont compris entre 1 et 16. En dessous ou au–delà, c'est impossible.

27 NEXT sans FOR

Il manque un FOR à un NEXT.

36 Opcode inconnu

Un programme en langage machine contient une instruction que le 68000 ne connaît pas.

22 Opérande manquant

Il vous manque un opérande dans la ligne.

```
100  DATA 1, 2, : REM Il manque un opérande après la virgule
```

66 Pas de numéros de lignes

Il manque des numéros de lignes à un programme en ASCII chargé par LOAD. Il faut utiliser 'LOAD BLOCK' ([Shift]–[F8]).

19 Pas de RESUME

Sans commentaire.

4 Plus de DATA

Un ordre READ ne trouve plus de DATA.

7 Plus de mémoire

Deux possibilités:

- Plus de place en mémoire. Programme trop grand ou tableaux trop largement dimensionnés.
- Plus de place sur la pile processeur (STACK). La pile est utilisée par :
  - Les boucles ouvertes.
  - Les appels de sous-programmes.
  - Les appels de procédures/fonctions
  - Certains ordres (SORT...)

24 REPEAT sans UNTIL

Une boucle REPEAT ne trouve pas de UNTIL.

33 RESET

Vous avez appuyé sur le bouton RESET. Mais rien de grave si le système n'est pas déjà planté, car l'OMIKRON conserve votre programme en mémoire.

20 RESUME sans erreur

L'ordre RESUME est exécuté sans qu'une erreur ait eu lieu.

14 RETURN sans fonction

Correspond à l'erreur 'RETURN sans GOSUB' pour les fonctions à plusieurs lignes.

3 RETURN sans GOSUB

L'OMIKRON a rencontré l'ordre RETURN sans trouver sa contrepartie (DEF PROC, DEF FN ou GOSUB).

Exemple :

```
100  GOSUB Entre
200  PRINT "Trois fois ";A;" égal ";3*A
300  -Entre
400  INPUT  "Donnez-moi un nombre, patron";A
500  RETURN
```

1 Structure trop longue

Une structure contient plus de 64 Ko de programme (une structure peut être une boucle FOR...NEXT, WHILE...WEND, REPEAT...UNTIL).

31 THEN, ELSE ou ENDIF sans IF ou THEN

Il manque un IF ou un THEN à un THEN, ELSE ou ENDIF.

67 Trop de fichiers

GEMDOS n'accepte qu'un certain nombre de fichiers sur une disquette. Au-delà, il faut reprendre une autre disquette ou effacer des fichiers.

25 UNTIL sans REPEAT

Il manque un REPEAT à un UNTIL.

21 Utilisez EXIT

Certains basics font des saletés plein partout.

```
100  FOR T=1 TO 10
110  INPUT "La bourse ou la vie ? <O>ui ou <N>on ";Rep$
120  IF UPPER$(REP$) = "O" THEN T = 11 : NEXT T
130  NEXT T
```

Franchement, c'est laid. Et c'est interdit en OMIKRON.

46 Utilisez EXIT TO dans les fonctions

Une fonction à plusieurs lignes ne doit pas utiliser un simple EXIT. Prenez un EXIT TO.

40 Violation de privilège

Un programme en langage machine utilise une instruction en mode UTILISATEUR. L'OMIKRON fonctionne toujours en mode SUPERVISEUR.

Il faut donc toujours enlever les passages en mode SUPERVISEUR puisque vous y êtes déjà.

30 WEND sans WHILE

Il manque un WHILE à un WEND

29 WHILE sans WEND

Une boucle WHILE ne trouve pas de WEND.

REFERENCES

LISTES DES INSTRUCTIONS PAR ORDRE ALPHABETIQUE

Les caractères arithmétique +, −, *, /, \, ^ ne sont pas abordés dans cet index. Vous les trouverez dans le chapitre consacré aux opérateurs.

*	*<Chaine>*<Expression_Numerique>*	59
+	*<Chaine>+<Chaine>*	59
ABS	*A=ABS(B)*	50
AES	*AES(Code_Op, Global, Intin, Addrin, Intout, Adrout)*	201
AND	*A=B AND C*	45
ARCCOS	*A= ARCCOS(B)*	53
ARCCOT	*A= ARCCOT(B)*	53
ARCOTH	*A= ARCOTH(B)*	54
ARCSIN	*A= ARCSIN(B)*	53
ARSINH	*A= ARSINH(B)*	54
ARTANH	*A= ARTANH(B)*	54
ASC	*A= ASC(<Chaine>)*	63
ATN	*A= ATN(B)*	53
BACKUP	*BACKUP <Chemin_Acces>*	125
BIN$	*A$= BIN$(<Expression_Numerique>)*	57
BIOS	*BIOS([variable_Retour],<Numero_Fonction> [,Parametre]...)*	199
BIT	*A=BIT(<Num_Bit>,C)*	48
BITBLT	*BITBLT <M> TO X,Y,W,H [,<MODE>]*	162
	BITBLT<X>,<Y>,<W>,<H>TO<X2>,<Y2>,<W2>,<H2> [,<MODE>]	
	BITBLT <X>,<Y>,<W>,<H> TO <M>	
BLOAD	*BLOAD <Nom_Fichier> [,<Adresse>]*	202
BOX	*BOX <X>,<Y> TO <X2>,<Y2>*	155
	BOX X,Y,H,L	
BSAVE	*BSAVE <Nom_Fichier>[,<Adresse>,<Longueur>]*	203
CALL	*CALL <Adresse>[(Parametre,Parametre...)]*	192
CDBL	*A#=CDBL(<Expression_Numerique>)*	39
CHAIN	*CHAIN [MERGE] "Nom_Fichier" [,<Deb_Ligne>]*	121
CHDIR	*CHDIR <Chemin_Acces>*	126
CHR$	*A$= CHR$(<Expression_Numerique>)*	64
CINT	*A%= CINT (<Expression_Numerique>)*	39
CIRCLE	*CIRCLE <X>,<Y>,<Rayon> [,<Angle_Dep>,<Angle_Arr>]*	157
CINTL	*A%L= CINTL (<Expression_Numerique>)*	40
CLEAR	*CLEAR <GEMDOS> [,<Pile>]*	111
CLIP	*CLIP [<X>,<Y>,<Hauteur>,<Largeur>]*	161

CLOSE	*CLOSE [<Numero_Canal>]*	130
CLS	*CLS*	208
CMD	*CMD <Canal>*	78
COMMON	*COMMON <Liste_Variable>*	122
COMPILER	*IF COMPILER THEN*	209
CONT	*CONT <Etiquette>*	91
	CONT <Etiquette> TO <Etiquette>	
COPY	*COPY <Chemin_Acces> TO <Chemin_Acces>*	125
COS	*A= COS(B)*	52
COSEC	*A= COSEC(B)*	53
COSECH	*A= COSECH(B)*	54
COSH	*A= COSEH(B)*	53
COT	*A= COT(B)*	52
COTH	*A= COTH(B)*	53
CSNG	*A!= CSNG (<Expression_Numerique>)*	39
CSRLIN	*CSRLIN*	69
CVD	*CVD (<ChaineS>)*	41
CVI	*CVI (<ChaineS>)*	42
CVIL	*CVIL (<ChaineS>)*	42
CVS	*CVS(<chaineS>)*	41
DATA	*DATA <Expression> [,<Expression>]*	118
DATE$	*DATES=<ChaineS>*	68
DEF FN	*DEF FN <Nom> [(Parametres)...]*	101
DEF PROC	*DEF PROC <Nom> ([Parametres...])*	97
DEF SPRITE	*DEF SPRITE <Numero>,<Type> [,[Adresse][,Sauvegarde]]*	166
DEF USR	*DEF USR=<Valeur>*	193
DEFDBL	*DEFDBL "<Liste_Variables>"*	32
DEFINT	*DEDFINT "<Liste_Variables>"*	32
DEFINTL	*DEFINTL "<Liste_Variables>"*	32
DEFSNG	*DEFSNG "<Liste_Variables>"*	32
DEFSTR	*DEFSTR "<Liste_Variables>"*	32
DEG	*DEG*	51
DET	*X= DET(A!(N,N))*	185
DIM	*DIM <Tableau>(<Valeur>[,<Valeur>]...)[,<Tableau>...]*	112
DRAW	*DRAW <X>,<Y>*	148
	DRAW TO <X>,<Y>	
	DRAW <X>,<Y> TO <X2>,<Y2> [TO <X3>,<Y3>][TO etc...]	
DUMP	*DUMP [Alpha][,Aipha]*	176
EDIT	*EDIT[<ligne>]*	12
ELLIPSE	*ELLIPSE<X>,<Y>,<RayX>,<RayY>[,[Ang_Dep] [,Ang_Arr]]*	158
ELSE	*IF <Conditions> THEN -INSTR- ELSE -INSTR- [ENDIF]*	105
END	*END*	92
ENDIF	*IF <Conditions> THEN -INSTR- [ELSE -INSTR-] [ENDIF]*	105
EOF	*EOF(<Numero_Canal>)*	131
EQV	*A=B EQV C*	46
ERL	*A= ERL*	70
ERR$	*AS= ERRS*	70
ERR	*A= ERR*	70

ERROR	*ERROR <Numero_Erreur>*	139
EXEC	*EXEC <Nom_Programme>[,<Chaine_Commande>]*	122
EXIT	*EXIT [TO <Etiquette>]*	110
EXP	*A= EXP(B)*	55
FACT	*A= FACT(B)*	55
FIELD	*FIELD #<Numero_Canal>,<Longueur>[AS <Variable>]...*	133
FILES	*FILES <Chemin_Acces>*	123
FILESELECT	*FILESELECT(<Chemin_Acces>,<Nom_Fichier>,<Retour>)*	171
FILL	*FILL <X>,<Y>,<Expression_Numerique>*	159
FILL COLOR=	*FILL COLOR= <Expression_Numerique>*	152
FILL STYLE=	*FILL STYLE=<Exp_Numerique>,<Expression_Numerique>*	152
FIX	*A = FIX(B)*	49
FN	*FN <Nom> [(Parametres...)]*	101
FOR	*FOR <Var_Num>= <Exp_Num> TO <Exp_Num> [STEP <Exp_Num>][:-INSTRUCTIONS-] :NEXT <Var_Num>*	109
FORM_ALERT	*FORM_ALERT(<texte>,<Choix>,<Retour>)*	169
FRAC	*A=FRAC(B)*	50
FRE	*FRE(0) FRE(<Chaine$>) FRE("Lecteur")*	35
GEMDOS	*GEMDOS([Variable_Retour],<Num_Fonc> [,Parametre]...)*	200
GET	*GET <Numero_Canal>,<Numero_Enregistrement>*	135
GOSUB	*GOSUB <Etiquette>*	94
GOTO	*GOTO <Etiquette>*	93
HCOPY	*HCOPY*	207
HCOPY TEXT	*HCOPY TEXT*	207
HEX$	*A$= HEX$(<Expression_Numerique>)*	57
HIGH	*HIGH (<Expression_Numerique>)*	202
IF	*IF <Conditions> THEN -INSTR- [ELSE -INSTR-] [ENDIF]*	105
IMP	*A=B IMP C*	47
INKEY$	*A$= INKEY$*	87
INLINE	*INLINE <Chaine>*	197
INPUT	*INPUT [@(Y,X);]["texte";] <Variable>[,<Variable>]*	79
INPUT USING	*INPUT [@(y,X);] ["texte";] <Chaine> USING [<Chaine_Com>] [,<Ret>][,<Long>][,<Car_Rem>][,<Pos_Cur>]*	81
INPUT$	*A$= INPUT$(<Nbr_Car>)* *A$= INPUT$(<Nbr_Car>,<Num_Canal>)*	88/132
INPUT #	*INPUT #<Numero_Canal>,<Variable>[,<Variable>]*	130
INSTR	*A= INSTR(<Chaine_Depart>,<Chaine_Recherchee>)* *A=INSTR(<Exp_Num>,<Chaine_Depart>,<Chaine_Recherchee>)*	64
INT	*A=INT(B)*	49
IPL	*IPL<Expression_Numerique>*	203
JOYSTICK	*JOYSTICK(X)*	208
KEY	*KEY <Touche_Fonction>="texte"*	173
KEY LIST	*KEY LIST*	174
KILL	*KILL <Chemin_Acces>*	124

L	*L<Variable>*	202
LDUMP	*LDUMP [Alpha][,Alpha]*	177
LEFT$	*LEFT$(<Chaine>,<Expression_Numerique>)*	60
LEN	*A= LEN(<Chaine>)*	61
LET	*[LET] <variable = <Expression_Numerique>*	
LIBRARY	*LIBRARY <Proc>,<Chemin_Acces>*	205
LINE COLOR=	*LINE COLOR= <Teinte>*	151
LINE INPUT	*LINE INPUT [@(Y,X);]["texte";] <Variable> [,<Variable>]...*	81
LINE INPUT #	*LINE INPUT #<Numero_Canal>,<Variable> [,<Variable>]...*	130
LINE WIDTH	*LINE WIDTH= <Expression_Numerique>*	165
LINE STYLE	*LINE STYLE= <Numero_Style>*	151
LIST	*LIST [-][<Exp_Numerique>][-<Exp_Numerique>]*	89
LLIST	*LLIST [-][<Exp_Numerique>][-<Exp_Numerique>]*	90
LN	*A= LN(B)*	54
LOAD	*LOAD <Nom_Fichier>*	10/121
LOCAL	*LOCAL <Liste_Variables>*	99
LOCATE	*LOCATE <Ligne>,<Colonne>*	Cf PRINT @
LOC	*LOC(<Numero_Canal>)*	136
LOF	*LOF(<Numero_Canal>)*	131
LOG	*A= LOG(B,C)*	54
LOW	*LOW(<Expression_Numerique>)*	202
LOWER$	*AS= LOWERS(<Chaine>)*	66
LPEEK	*LPEEK (<Adresse>)*	191
LPOKE	*LPOKE <Adresse>, <Valeur>*	190
LPOS	*LPOS(0)*	69
LPRINT	*LPRINT [<Chaine>][,<Chaine>]*	75
LSET	*LSET <Variable_Tampon>= <Chaine>*	133
MAT=1	*MAT A!(X,Y)=1*	182
MAT *	*MAT A!(X,X) * <Expression_Numerique>*	182
MAT *	*MAT A!(N,M) = B!(N,Q) * C!(Q,M)*	183
MAT CLEAR	*MAT CLEAR A!(<Dimension>[,<Dimension>]...)*	181
MAT INV	*MAT A!(N,N) = INV B!(N,N)*	184
MAX	*A=MAX(B,C)*	44
MEMORY	*MEMORY(-1)*	195
	MEMORY(<Expression_Numerique>)	
	MEMORY(<Chaine>)	
MEMORY_BLOCK	*MEMORY_BLOCK <Numero_Bloc>,<Longueur>,<Variable>*	196
MEMORY_MOVE	*MEMORY_MOVE <Depart>,<Longueur> TO <Recept>*	196
MEMORY_MOVEB	*MEMORY_MOVEB <Depart>,<Longueur> TO <Recept>*	196
MERGE	*MERGE <Nom_Fichier>*	10/121
MID$	*AS= MIDS(<Chaine>,<Exp_Num>[,<Exp_Num>])*	61
MID$=	*MIDS(<Chaine>,<Exp_Num>[,<Exp_Num>])=<Chaine_Rempl>*	114
MIN	*A=MIN(B,C)*	44
MIRROR$	*AS= MIRRORS(<Chaine>)*	65
MKD$	*MKDS(<Expression_Numerique>)*	40
MKDIR	*MKDIR <Nom_Dossier>*	126
MKI$	*MKIS(<Expression_Numerique>)*	42
MKIL$	*MKILS(<Expression_Numerique>)*	42
MKS$	*MKSS(<Expression_Numerique>)*	41
MOD	*A= B MOD C*	43

MODE	*MODE <Chaine>*	66
MODE=	*MODE=<Expression_Numerique>*	149
MODE LPRINT	*MODE LPRINT <F>*	207
MOUSEBUT	*A= MOUSEBUT*	67
MOUSEX	*A= MOUSEX*	67
MOUSEY	*A= MOUSEY*	67
MOUSEON	*MOUSEON*	172
MOUSEOFF	*MOUSEOFF*	172
NAME...AS	*NAME <Ancien_Nom> AS <Nouveau_Nom>*	124
NAND	*A=B NAND C*	47
NDC	*NDC [<X>,<Y>,<W>,<H>]*	148
NEW	*NEW[<Chaine>]*	92
NEXT	*FOR <Var_Num>= <Exp_Num> TO <Exp_Num> [STEP <Exp_Num>][:-INSTRUCTIONS-] :NEXT <Var_Num>*	109
NOISE	*NOISE <Masque>,<Frequence>*	188
NOR	*A=B NOR C*	47
NOT	*A=NOT B*	45
OCT$	*AS= OCTS(<Expression_Numerique>)*	57
ON ERROR GOTO	*ON ERROR GOTO <Etiquette>*	137
ON HELP GOSUB	*ON HELP GOSUB <Etiquette>*	142
ON KEY GOSUB	*ON KEY GOSUB <Etiquette>*	142
ON MOUSEBUT GOSUB	*ON MOUSEBUT GOSUB <Etiquette>*	141
ON TIMER GOSUB	*ON TIMER <Secondes>GOSUB <Etiquette>*	143
ON TRON GOSUB	*ON TRON GOSUB <Etiquette>*	175
ON...GOSUB	*ON <Exp_Num> GOSUB <Sous_Prg1>[,<Sous_Prg2>...]*	95
ON...GOTO	*ON <Exp_Num> GOTO <Sous_Prg1>[,<Sous_Prg2>...]*	95
ON..RESTORE	*ON <Expression_Numerique> RESTORE <Etiq1>[,<Etiq2>...]*	119
OPEN	*OPEN <Type>,<Canal>[,<Nom_Fichier>][,<Long_Maxi]*	128
OR	*A=B OR C*	46
OUTLINE	*OUTLINE {ON|OFF}*	154
PALETTE	*PALETTE [Teinte],[Teinte]*	149
PBOX	*PBOX <X>,<Y> TO <X2>,<Y2>*	156
	PBOX <X>,<Y>,<Largeur>,<Hauteur>	
PCIRCLE	*PCIRCLE<X>,<Y>,<Rayon>[,[Angle_Depart][,Angle_Arrivee]]*	158
PEEK	*PEEK(<Adresse>)*	190
PELLIPSE	*PELLIPSE<X>,<Y>,<RayonX>,<RayonY>[,[Angle_Dep][,Angle_Arr]]*	159
PI	*A#= PI*	69
POINT()	*POINT(<X>,<Y>)*	162
POKE	*POKE <Adresse>,<Valeur>*	189
POS	*POS(0)*	69
PRBOX	*PRBOX <X>,<Y> TO <X2>,<Y2>*	157
	PRBOX <X>,<Y>,<Largeur>,<Hauteur>	
PRINT	*PRINT [<Chaine>] ou [<Expression_Num>][<Car_Controle>]*	71
PRINT @	*PRINT @(<Ligne>,<Colonne>);<Sortie>*	76
PRINT USING	*PRINT USING <Masque>,<Chaine>*	73

PRINT # *PRINT #<Canal>,<Sortie>* 76/129
PROC *[PROC] <Nom_Procedure[(liste_Parametres)]>* 97
POLYGON *POLYGON <Tableau>* 154
PPOLYGON *PPOLYGON <Tableau>* 155
PUT *PUT <Numero_Canal>,<Numero_Enregistrement>* 135

RAD *RAD* 52
RBOX *RBOX <X>,<Y> TO <x2>,<Y2>* 156
RBOX <X>,<Y>,<Largeur>,<Hauteur>
READ *READ <Variable>[,<Variable>]...* 117
REM *REM <texte>* 209
RENUM *RENUM [<Nouvelle_Ligne>][,<Ancienne_Ligne>][,<Pas>]* 177
REPEAT *REPEAT [-INSTRUCTIONS-] UNTIL <Conditions>* 107
RESTORE *RESTORE [<Etiquette>]* 119
RESUME *RESUME [<Etiquette>]* 138
RESUME NEXT
RETURN *RETURN* 94/100
RETURN <Parametre>
RIGHT$ *RIGHTS(<Chaine>,<Expression_Numerique>)* 60
RMDIR *RMDIR <Nom_Dossier>* 126
RND *A=RND(B)* 51
RSET *RSET <Variable_Tampon>= <Chaine>* 134
RUN *RUN[<Expression_Numerique>]* 90/121

SAVE *SAVE [<Nom_Fichier>]* 9/121
SAVE,A *SAVE [<Nom_Fichier>],A* 9/121
SCREEN *SCREEN <Ecran>[,Adresse][,Couleurs]* 147
SECH *A= SECH(B)* 54
SEC *A= SEC(B)* 53
SEEK *SEEK <Numero_Canal>,<Deplace>[,<Mode>]* 136
SEGPTR *SEGPTR* 194
SGN *A=SGN(B)* 50
SHL *A=B SHL N* 48
SHR *A=B SHR N* 47
SINH *A= SINH(B)* 53
SIN *A= SIN(B)* 52
SORT *SORT [ASC] <Tableau>* 115
SORT [ASC] <Tableau1> TO <Tableau2>
SPACE$ *AS= SPACES(<Expression_Numerique>)* 65
SPC *AS= SPC(<Expression_Numerique>)* 64/73
SPRITE *SPRITE<Numero>,[<X>],[<Y>],[<Dessin>]*
,<Couleur_Masque>,<Couleur_Dessin>] 167
SQR *A=SQR(B)* 55
STOP *STOP* 91
STR$ *AS= STRS(<Expression_Numerique>)* 63
STRING$ *AS= STRINGS(<Expression_Numerique>,<Caractere>)* 65
SWAP *SWAP <Variable>,<Variable>* 113
SYSTEM *SYSTEM* 11

TAB *PRINT TAB(<Expression_Numerique>)* 65/72
TANH *A= TANH(B)* 53

TAN	*A= TAN(B)*	52
TEXT	*TEXT <X>, <Y>, <texte>, [<Long>], [<Espace_Mots>], <Espace_Car>]*	160
TEXT COLOR=	*TEXT COLOR=<Numero_Teinte>*	153
TEXT HEIGHT=	*TEXT HEIGHT= <Hauteur_En_Pixels>*	153
TEXT ROTATION=	*TEXT ROTATION= <Expression_Numerique>*	165
TEXT STYLE=	*TEXT STYLE=<Effet>*	152
THEN	*IF <Conditions> THEN -INSTRUCTIONS- [ELSE -INSTRUCTIONS-] [ENDIF]*	105
TIME$	*TIMES=<Chaine>*	68
TIMER	*A=TIMER*	68
TROFF	*TROFF*	175
TRON	*TRON*	174
TUNE	*TUNE <Voie>,<Hauteur>[,<Voie>,<Hauteur>...]*	187
UNTIL	REPEAT [-INSTRUCTIONS-] UNTIL <Conditions>	107
UPPER$	*UPPERS(<Chaine>)*	66
USING	*USING <Chaine>*	75
USR()	*USR(<Valeur>)*	193
VAL	*VAL(<Chaine>)*	58/62
VARPTR	*VARPTR(<Variable>)*	195
VERSION	*A= VERSION*	70
VDI	*VDI(Code_Op, Contrl, Intin, Ptsin, Intout, Ptsout)*	201
VOLUME	*VOLUME <Voie>,<Volume>*	187
	VOLUME <Voie>,<Enveloppe>,<Periode_Oscillation>	
WAIT	*WAIT <Secondes>*	208
WEND	*WHILE <Conditions>[:<Instructions>:]WEND*	108
WHILE	*WHILE <Conditions>[:<Instructions>:]WEND*	108
WPEEK	*WPEEK(<Adresse>)*	190
WPOKE	*WPOKE <Adresse>,<Valeur>*	189
WRITE	*WRITE <Texte>*	77
WRITE #	*WRITE #<Canal>,<Texte>*	77/129
WVBL	*WVBL*	203
XBIOS	*XBIOS([Valeur_Retour],<Numero_Fonction> [,Parametre]...)*	200
XOR	*A=B XOR C*	46

INTRODUCTION

LES EDITEURS

SAVE
SAVE A
LOAD
MERGE

STRUCTURE DE LA MEMOIRE

ORGANISATION DES VARIABLES
DEFINT
DEFINTL
DEFSNG
DEFDBL
DEFSTR
ORGANISATION DE LA MEMOIRE
FRE

CONVERSION DE TYPE

CDBL
CSNG
CINT
CINTL
MKD$
CVD
MKS$
CVS
MKIL$
CVIL
MKI$
CVI

OPERATEURS

OPERATEURS DE BASE
+
-
/
*
^
\
MOD
OPERATEURS RELATION D'ORDRE
>
<
=
>=
<=
<>
MIN
MAX
OPERATEURS LOGIQUES
NOT
AND
OR
XOR
EQV
IMP
NAND
NOR
SHR
SHL
BIT

FONCTIONS MATHEMATIQUES

INT
FIX
FRAC
ABS
SGN
RND
DEG
RAD
SIN
COS
TAN
COT
SEC
COSEC
ATN
ARCTAN
ARCSIN
ARCCOS
ARCCOT
SINH
COSH
TANH
COTH
SECH
COSECH
ARSINH
ARTANH
ARCOTH
LN
LOG
EXP
SQR
FACT

LES BASES NUMERIQUES

LE BINAIRE
L'OCTAL
L'HEXADECIMAL
BIN$
HEX$
OCT$
VAL

CHAINES DE CARACTERES

ADDITION DE CHAINES
MULTIPLICATION DE CHAINES
LEFT$
RIGHT$
MID$
LEN
VAL
STR$
ASC
CHR$
INSTR
SPC
SPACE$$
TAB
STRING$
MIRROR$
UPPER$
LOWER$
MODE

VARIABLES SYSTEMES

MOUSEBUT
MOUSEX
MOUSEY
TIMER
TIMES
DATES
PI
CSRLIN
POS
LPOS
ERR
ERL
ERR$
VERSION

INSTRUCTIONS DE SORTIE

PRINT
TAB
SPC
PRINT USING
USING
LPRINT
PRINT#
PRINT @
WRITE
WRITE#
CMD

ENTREES

INPUT
LINE INPUT
INPUT USING
INKEY$
INPUT$

COMMANDES

LIST
LLIST
RUN
STOP
CONT
CONT TO
END
NEW

BRANCHEMENTS

GOTO
GOSUB...RETURN
ON...GOTO
ON...GOSUB

PROCEDURES & FONCTIONS

PROCEDURES
PASSAGE DE PARAMETRES
LOCAL
FONCTIONS SUR UNE LIGNE
FONCTIONS SUR PLUSIEURS LIGNES

CONDITIONS & BOUCLES

CONDITIONS
IF...THEN...ELSE...ENDIF
BOUCLES
REPEAT...UNTIL
WHILE...WEND
FOR...NEXT
EXIT

TABLEAUX & TRIS

CLEAR
DIM
SWAP
MID$
SORT

READ, DATA, RESTORE

READ
DATA
RESTORE
ON...RESTORE

GESTION DU DISQUE

RUN
CHAIN
COMMON
EXEC
FILES
NAME...AS
KILL
COPY
BACKUP
MKDIR
RMDIR
CHDIR

LES FICHIERS

FICHIERS SEQUENTIELS
OPEN
PRINT#
WRITE#
CLOSE
INPUT#
LINE INPUT#
EOF
LOF
INPUT$
FICHIERS RELATIFS (OU DIRECTS)
FIELD
LSET
RSET
PUT
GET
SEEK
LOC

LES TRAITEMENT DES ERREURS

ON ERROR...GOTO
ERR
ERL
ERR$
RESUME
ERROR

MULTITACHE CONDITIONNEL

ON MOUSEBUT GOSUB
ON KEY GOSUB
ON HELP GOSUB
ON TIMER GOSUB

GRAPHISME

SCREEN
NDC
DRAW
MODE-
PALETTE
LINE COLOR-
LINE STYLE-
FILL STYLE-
FILL COLOR-
TEXT STYLE-
TEXT HEIGHT-
TEXT COLOR-
OUTLINE
POLYGON
PPOLYGON
BOX
PBOX
RBOX
PRBOX
CIRCLE
PCIRCLE
ELLIPSE
PELLIPSE
FILL
TEXT
CLIP
POINT
BITBLT
LINE WIDTH
TEXT ROTATION
DEF SPRITE
SPRITE

FONCTIONS GEM DIRECTES

FORM_ALERT
FILE_SELECT
MOUSEON
MOUSEOFF

CHASSES AUX BUGGS

KEY
KEY LIST
TRON
TROFF
ON TRON GOSUB
DUMP
LDUMP
RENUM

MATRICES

RAPPEL
LES MATRICES ET L'OMIKRON
MAT CLEAR
MAT A!(N,N)-1
MAT * X
MAT A!-B! * C!
MAT INV
MAT DET

LE SON

TUNE
VOLUME
NOISE

ACCES AU LANGAGE MACHINE

ACCES DIRECT EN MEMOIRE
POKE
WPOKE
LPOKE
PEEK
WPEEK
LPEEK
LE LANGAGE MACHINE
CALL
DEFUSR
USR
SEGPTR
VARPTR
MEMORY
MEMORY_BLOCK
MEMORY_MOVE
MEMORY_MOVEB
INLINE

ACCES AU SYSTEME

BIOS
XBIOS
GEMDOS
VDI
AES
HIGH
LOW
L
BLOAD
BSAVE
IPL
WVBL

LES BIBLIOTHEQUES

LIBRARY

INSTRUCTIONS DIVERSES

HCOPY
HCOPY TEXT
MODE LPRINT
CLS
WAIT
JOYSTICK
COMPILER
REM

CLAUSES D'UTILISATION

Vous devez **impérativement** nous renvoyer la carte d'enregistrement pour recevoir gratuitement le manuel de programmation de l'OMIKRON (cf ci-dessous). De plus, **aucune** mise à jour, gratuite ou non, ne sera transmise aux acheteurs de l'OMIKRON n'ayant pas retourné leur carte d'enregistrement.

NOUS AVONS BESOIN DE CES CARTES, RENVOYEZ NOUS LES !

L'OMIKRON n'est pas protégé contre la copie et vous pouvez le copier autant de fois que vous le désirez, sous réserve que ces copies ne servent qu'à des fins de sauvegarde pour vous prémunir contre la perte des disquettes originales. Comme un livre ne peut être lu par deux personnes à la fois dans deux lieux différents, ce logiciel ne peut être utilisé à la fois sur deux micro ordinateurs par deux personnes. En aucun cas vous ne devez modifier, adapter ou intégrer ce logiciel dans d'autres produits destinés à la vente sans autorisation préalable de la part d'OMIKRON. Software & Hardware.

Le fait de déchirer les auto-collants qui maintiennent la ou les disquettes originales implique l'entière acceptation des conditions indiquées ci-dessus.

CARTE DE GARANTIE

IMPORTANT : OMIKRON.France offre le manuel de programmation aux acheteurs du basic OMIKRON en France.

1°) Si vous avez acheté la version 2.x ou la version 3.x ;

2°) Si vous nous retournez la carte de garantie entièrement et correctement remplie ;

Vous allez recevoir **GRATUITEMENT** le manuel de programmation. Ce manuel d'environ 200 pages contient de nombreuses astuces qui vous permettront de mieux utiliser votre basic OMIKRON. De plus, il est livré avec une disquette où vous pourrez retrouver les programmes du manuel de référence et du manuel de programmation.

Pour nos amis belges, suisses et tous ceux qui n'ont pas acheté le basic OMIKRON, le prix de ce manuel est de 199 Francs français toutes taxes comprises (soit 48 FS ou 1300 FB sans les taxes).

REMERCIEMENTS

Ce manuel a été mis en page sur un MEGA ST4 et le premier exemplaire a été sorti grâce à une imprimante laser SLM 804. Les logiciels employés pour la rédaction et la mise en page sont : TIMEWORKS 1.12, REDACTEUR 1.97 et 2nd_Word 1.

Tous nos remerciements à la société Double A, 80 Avenue de Laon 51100 Reims et à Alain Laurent pour la mise en page.

Les corrections ont été réalisées par Ghislain Wysocinski.

Le basic OMIKRON a été entièrement écrit en assembleur par Artur Södler et Thomas Kemp.

Le manuel d'origine a été rédigé par Artur Södler, Carsten Kraus et Thomas Kemp.

La traduction a été réalisée par Alix Jail et Dietrich Raisin, qui a aussi programmé la plupart des exemples.

Copyright 1987/1989 par OMIKRON.France, 11 rue Dérodé, 51100 Reims.

COPYRIGHT

Tous droits de traduction, d'adaptation et de reproduction par tous procédés réservés pour tous pays.

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collectives" et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration : "Toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite" (alinéa 1er de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du code pénal.

LE COMPILATEUR

Le basic OMIKRON est l'un des langages les plus rapides et des plus faciles à utiliser. Vous pouvez encore améliorer la vitesse de vos programmes en les compilant. D'après tous les magazines allemands, le compilateur OMIKRON est le plus rapide des Basics compilés. Il génère un code compact et propre.

Voici un tableau qui indique les temps d'exécution interprétés et compilés pour certaines instructions :

INSTRUCTIONS	INTER	COMP	ACCELER.
FOR...NEXT	0.240	0.075	3.2
REPEAT...UNTIL	2.225	0.080	27.8
CALCULS (+-/*)	3.320	0.450	7.4
BOUCLE IF	1.295	0.115	11.3
CHAINES	6.310	2.980	2.1

(Chaque boucle est répétée 10000 fois).

En moyenne, le facteur d'accélération est de 2 à 3 pour les entiers et de 4 à 5 pour les réels simple précision.

VITESSE DE COMPILATION

La vitesse de compilation est d'environ 40 Kilo-octets par minute.

UTILISATION

Le compilateur est extrêmement simple à utiliser : il suffit de choisir le programme à compiler (avec le sélecteur de fichiers) et le programme se compile.

TAILLE DES PROGRAMMES

La taille d'un programme compilé est de 10% à 80% supérieure à la taille du programme interprété.

COPYRIGHT

Vous disposez du droit de compiler et de vendre tous vos programmes sans verser de royalties à OMIKRON.SOFTWARE. Néanmoins, dans le cas d'un langage de programmation (comme le

Pascal, le Forth...), vous devrez demander une autorisation.

La seule condition : vous devez mettre dans la documentation du programme et dans le programme lui-même une indication concernant l'utilisation du basic OMIKRON et du compilateur.

PRIX

Le compilateur est livré dans un boitier en plastique blanc, contenant le manuel d'utilisation, la carte de garantie et la disquette originale. Son prix est de 545 Francs TTC (Pour nos amis suisses et belges : 115 FS et 3000 FB, sans les taxes).

LE COMPILATEUR 68881

Nous disposons aussi d'un compilateur pour le co-processeur arithmétique 68881 diffusé par ATARI. Les gains de temps sont impressionnants, en particulier avec les fonctions trigonométriques (SINUS, COSINUS..), dont la vitesse est multipliée par ... 53! Ce compilateur a été classé comme le plus rapide existant (tous langages confondus) sur l'ATARI ST. De plus – et il s'agit là de l'un de ses points forts – le compilateur 68881 ne nécessite **AUCUN** changement dans le programme source. En effet, les réels double précision de l'OMIKRON correspondent à la norme IEEE extended (soit 10 octets).

PRIX

Le compilateur est livré dans un boitier en plastique blanc, contenant le manuel d'utilisation, la carte de garantie et la disquette originale. Son prix est de 795 Francs TTC (Pour nos amis suisses et belges : 170 FS et 4500 FB, sans les taxes).

LES BIBLIOTHEQUES

Le basic OMIKRON dispose de plusieurs biliothèques qui permettent d'accéder à une base de fonctions spécialisées dans certains domaines. Voici la liste des bibliothèques disponibles ou en passe de l'être :

STATLIB : Spécialisée dans les statistiques. Très nombreuses fonctions accessibles facilement.

NUMLIB : Approximation de fonctions numériques.

COMPLEX : Nombres complexes.

FINANCE : Bibliothèque financière. Plus de 100 fonctions.

Vous pouvez obtenir plus de détails en ce qui concerne ces bibliothèques en nous adressant une demande de renseignements.

DRAW 3.0

DRAW 3.0 est un programme qui nécessite au minimum 1 Méga-octets de mémoire et un écran monochrome. Il dispose de toutes les fonctions classiques des programmes de dessins et d'innovations particulièrement importantes. Il permet l'animation la plus rapide du marché et la plus simple à manipuler.

Il charge pratiquement tous les formats existants (STAD, STAD compressé, DEGAS ELITE en basse et haute résolution, compressé ou non, DOODLE, GEM) et sauvegarde au format STAD compressé, DRAW, GEM et DRAW A4.

- Il reprend toutes les polices de caractères de SIGNUM! et les affiche à l'écran.
- Il peut imprimer sur les 8, 9 et 24 aiguilles en 4 formats différents.
- Il permet de travailler au format A4 (le double de la normale).
- Il possède les fonctions de transformation et de manipulation de blocs les plus évoluées et les plus rapides du marché.

Son prix est de 495 Francs TTC (soit 105 FS ou 2800 FB).

OMIKRON.

OMIKRON.France dispose de plusieurs autres produits qui vous permettent de tirer le maximum de votre basic OMIKRON. Vous pouvez acheter ces produits chez votre revendeur habituel ou directement à notre adresse :

OMIKRON.France
11, Rue Dérodé
F–51100 REIMS
FRANCE

Frais de port

En France métropolitaine :
Envoi en recommandé : Pas de frais de port.
Envoi contre–remboursement : Forfait de 50 francs.

Pour les DOM :
Envoi recommandé : Forfait de 30 Francs.
Envoi contre–remboursement : Forfait de 60f francs.

En Suisse et en Belgique :
Envoi recommandé : Forfait de 30 Frf (soit 7.5 FS ou 200 FB).
Envoi contre–remboursement : Forfait 60 Frf (soit 15 FS/ 400 FB).

Dans le cas où vous nous commandriez un produit, nous vous demandons de bien vouloir remplir le coupon destiné à cet effet :

✂----------

BON DE COMMANDE A RETOURNER A :

OMIKRON.France, 11 Rue Dérodé, F–51100 REIMS FRANCE

Nom : .....

Prénom : .....

Adresse : .....

Code postal : .....

Ville : .....

Pays : .....

Produits : .....

ATARI®

Copyright ©1987 Atari Corporation
Sunnyvale, CA94086
Tous dreits Réservés

C103412-003 REV. A
Printed in Taiwan
1990 3 CHENG SHEN

ATARI (Benelux) B.U. Belgian Branch Boomsesleenweg 128, B-2621 Schelle

Téléchargé sur
Le Vieux Manuel

WWW.MANUELS.ABANDONWARE-FRANCE.ORG