MEMSOFT

MEMSOFT pour APPLE IIGS

MEMSOFT pour APPLE II

MEMSOFT

# MEMSOFT

Réalisé
par

**Philippe NESNIDAL**

**Martine PUJOL**

**Jean-Charles MATHEY**

**Jean DESPATIN**

**Mathieu ROSSI**

Sur un scénario
original
de

**Patrick LAFFITTE**
et
**Philippe NESNIDAL**

Cette documentation
a été réalisée
avec la
participation
de

**Jean-Pierre LAMOITIER**

MEMSOFT
S.A. au capital de 4.925.000 F -
Siège social 3, rue Meyerbeer 0600 NICE FRANCE
R.C. NICE 81 B 320
APE 7703

**ATTENTION : Pour pouvoir bénéficier de la garantie MEMSOFT, vous devez impérativement nous retourner dans le mois qui suit votre achat, le BON de GARANTIE, inséré à la fin de ce manuel, accompagné de la photocopie de la facture d'achat.**

## RECOMMANDATIONS

## IMPORTANTES

L'acheteur ou l'utilisateur doit vérifier, en prenant conseil auprès de son revendeur, que le produit est bien propre à satisfaire ses besoins.

De même, il doit vérifier auprès de son revendeur la compatibilité du produit avec le matériel et les logiciels qu'il possède déjà ou qu'il compte acquérir.
L'acheteur ou l'utilisateur doit satisfaire au respect absolu des conditions et précautions d'utilisation, notamment:

- Usage normal.
- Locaux respectant les normes électriques et de sécurités en vigueur et possédant les aménagements nécessaires pour éviter toute détérioration des programmes.
- Absence de variation ou défaillance du courant électrique.
- Absence totale de toute intervention, de toute modification ou tentative de modification des programmes.
- Absence de variation ou défaillance du courant électrique.
- Copie-sauvegarde journalière de l'ensemble des fichiers dès qu'une modification leur est apportée et contrôle, afin de s'assurer de la conformité de l'opération de copie.
- Conservation, dans un meuble protecteur approprié ou hors des locaux, d'une copie du logiciel et des fichiers de données ainsi que des copies sauvegardes périodiques afin de pouvoir redémarrer le système en cas de destruction de l'ensemble des copies (incendies, inondations, etc...).
- Conservation des logiciels et copies dans un emplacement sec, à température normale (16° à 25° Celsius), hors de tout champ magnétique et électrostatique.
- L'acheteur n'acquiérant qu'une documentation, un support et un droit d'utilisation du logiciel pour son usage personnel et pour un seul ordinateur, toute copie ou tentative de copie, exception faite d'une copie de sauvegarde, lui est strictement interdite (Loi du 03-07-1985).

## DOCUMENTATION

## GARANTIE MEMSOFT pour APPLE II GS

Le produit est garanti, contre tous vices de fabrication, pendant trois (3) mois à compter de la livraison au revendeur.

Cette garantie est strictement limitée à un échange standard. Pour bénéficier de cette garantie, l'acheteur devra impérativement satisfaire aux conditions expresses suivantes :

- avoir retourné à MEMSOFT FRANCE, SAV, 3 rue Meyerbeer, 06000 NICE, d'une part le bon de garantie qui se trouve à la fin du présent manuel, dûment complété et signé, et, d'autre part, une photocopie de la facture d'achat du produit. Cet envoi groupé doit intervenir dans le délai d'un mois à compter de la date d'achat du produit.

- restituer la disquette détériorée ou la clé électronique à MEMSOFT FRANCE, SAV, 3 rue Meyerbeer, 06000 NICE, accompagnée du bon d'échange dûment complété.

Le produit fourni en retour par MEMSOFT FRANCE pourra être une version différente. Les frais de port, aller et retour, sont à la charge de l'acheteur.

Le bénéfice de la garantie est subordonné au respect, par l'acheteur, ses ayants droit et ayants cause, des "RECOMMANDATIONS IMPORTANTES" ci-dessus.

Leur non-respect exclurait de plein droit le bénéfice de la garantie. De plus, toute copie ou tentative de copie du logiciel engagerait la responsabilité de l'acheteur, de ses ayants droit et ayants cause.

En tout état de cause, la responsabilité de MEMSOFT S.A. et de MEMSOFT FRANCE, éventuellement admise, est limitée au maximum au montant du prix de vente hors taxe du produit au revendeur.

Les spécifications du produit, données dans le présent manuel ont le caractère de simples indications qui ne sauraient engager MEMSOFT S.A.

MEMSOFT S.A. et MEMSOFT FRANCE n'ont aucun lien de droit avec le client final.

Les modifications techniques, esthétiques ou de tous ordres, jugées utiles par MEMSOFT S.A., peuvent intervenir à tout moment, sans obligation d'avoir à modifier pareillement les produits commandés, livrés ou à livrer.

## SERVICE CLIENTS

Le bénéfice de ce service est subordonné à la condition expresse que le client ait retourné dans le délai et selon les modalités prévus au paragraphe "GARANTIE", le bon de garantie signé et la copie de la facture d'achat.

Ces services sont les suivants:

1 - ECHANGE DE LA CARTE ELECTRONIQUE EN CAS DE DETERIORATION

Echange standard accordé aux conditions expresses suivantes:

- Restitution préalable à MEMSOFT FRANCE, aux frais du client de la carte électronique détériorée.

- Cette restitution doit être accompagnée d'une part du bon d'échange dûment complété et, d'autre part, du règlement correspondant au tarif ci-dessous, libellé à l'ordre de MEMSOFT FRANCE. Un nouveau bon vous sera adressé avec la carte électronique de remplacement.

2 - FOURNITURE DE LA VERSION LA PLUS RECENTE DU PRODUIT.

Correspond à la fourniture des disquettes dont les fichiers ont évolué, et des feuillets de mise à jour de la documentation pour adapter votre version à une version plus récente qui pourrait être commercialisée, utilisant la même carte électronique. Cette possibilité exclut l'échange avec tout nouveau produit pouvant exister, même destiné au même ordinateur.

Cette adaptation pourra être obtenue à la condition expresse suivante :

Envoi d'une part du bon d'échange dûment complété et, d'autre part, du règlement correspondant au tarif ci-dessous, libellé à l'ordre de MEMSOFT FRANCE. Un nouveau bon vous sera adressé avec la disquette de remplacement.

LIMITATION DE DUREE: Le bénéfice des services ci-dessus est limité au 31 décembre 1989. Il ne pourra être donné aucune suite aux envois reçus après cette date.

TARIF 1987

Echange de la carte électronique ..... 700 F TTC
Echange pour nouvelle version ..... 560 F TTC
Frais de port en sus pour DOM-TOM et étranger.

Pour les années postérieures, l'acheteur devra se renseigner sur les tarifs en vigueur soit auprès de son revendeur, soit auprès de MEMSOFT FRANCE.

# SOMMAIRE

# 1 MEMSOFT : UN CONCEPT

Actuellement, certains programmeurs de gestion passent leur temps sur des problèmes de programmeurs système. Les détails de finition d'un logiciel les entraînent bien loin de leur préoccupation initiale.

Le temps qu'ils y consacrent est si important que c'est au détriment de leur créativité ou de la qualité des programmes.

MEMSOFT apporte de nouvelles solutions au dialogue avec l'utilisateur, à la saisie et à la visualisation d'informations dans des fenêtres, au stockage de données...

Tous les besoins prévisibles ont été directement intégrés dans MEMSOFT : ce qui était la part la plus fastidieuse de la programmation se réduit maintenant à une simple utilisation de MEMSOFT.

Le BASIC MEMSOFT libère le programmeur !

MEMSOFT a été spécialement conçu dans le but de fournir des réponses **complètes et simples** aux attentes des programmeurs d'**applications de gestion.**

**Dialoguer avec l'utilisateur :**

Proposer une intéraction agréable, contrôler la saisie, signaler les erreurs et permettre leur correction, sont des informations primordiales pour le confort de l'utilisateur.

La gestion d'écrans de **MEMSOFT** réalise en un seul ordre la saisie, la modification et la validation des informations contenues dans une fenêtre.

L'esthétique n'est plus une affaire de programmation : les couleurs, les déplacements de fenêtres, l'utilisation de la souris sont totalement intégrés et à tout moment disponibles.

Il ne vous reste plus qu'à définir le nombre de fenêtres que vous voulez utiliser, les rendre visibles ou invisibles et surtout décider de ce que vous voulez y présenter ...

**Guider l'utilisateur :**

Fournir à la demande et à tout moment une explication appropriée pour guider l'utilisateur nécessite un effort important du programmeur.

Le système de gestion d'aides intégré de **MEMSOFT** vous décharge de toute cette programmation.

**Gérer des données :**

Vous avez besoin de manipuler des informations dans des fichiers de structures variées.

La gestion sophistiquée de fichiers de **MEMSOFT** vous évite la programmation des détails d'une recherche, d'une insertion, d'une mise à jour ou d'une destruction dans un fichier. Vous précisez simplement l'opération souhaitée, elle est prévue avec tous ses cas particuliers par **MEMSOFT**. Vous n'aurez jamais à préciser la taille de quoi que ce soit. Les limites seront celles de votre espace disque ...

**Ecrire et mettre au point les traitements :**

Programmer des applications de gestion impose d'éviter les erreurs d'arrondis, de traiter des chaînes, de récupérer toutes les erreurs...
Et tout cela avec un temps de mise au point minimum.

**MEMSOFT** vous propose un environnement de programmation facile (éditeur, mode trace, contrôle syntaxique...) centré sur un **BASIC** structuré permettant de calquer l'architecture de votre programme sur votre analyse, des étiquettes clarifiant l'utilisation éventuelle des "GOTO".

De plus, avec ce Basic, vous disposez d'un éventail complet de fonctions numériques et de manipulations de chaînes de caractères. Les calculs effectués sur quatorze chiffres sont particulièrement adaptés aux applications de gestion.

Enfin, l'installation d'une adresse de branchement en cas d'erreur évite de parsemer votre programme de vérifications diverses liées aux traitements d'erreurs.



**MEMSOFT UNE SOLUTION OUVERTE :**

Un système fermé pourrait freiner la suite de votre évolution. **MEMSOFT** offre au programmeur une réelle transportabilité des programmes et des données vers les machines les plus répandues au monde comme, par exemple, votre **APPLE II GS.**

Ainsi, les programmes ne dépendent plus du tout de la machine et sont utilisables, sans la moindre modification, sur les nouvelles machines dès qu'elles supportent **MEMSOFT.**

Pour l'utilisateur de programmes écrits sous **MEMSOFT,** c'est aussi l'assurance de pouvoir faire évoluer son matériel sans jamais perdre une seule de ses données. En d'autres termes, il n'aura pas à ressaisir les 80 000 fiches clients qui ont nécessité trois ans de frappe forcenée !

# 2 LES TROIS COMPOSANTES DE MEMSOFT

MEMSOFT est composé de trois sous-ensembles que nous allons découvrir plus en détails :

**MEMBASIC : LE BASIC INTERPRETE**

**MEMSCREEN : LA GESTION D'ECRANS**

**MEMFILE : LA GESTION DE FICHIERS**

**MEMBASIC :**

MEMBASIC est un interpréteur BASIC évolué.

MEMBASIC facilite la programmation grâce à des instructions structurées comme DO...LOOP, SELECT...CASE, IF...THEN...ELSE multilignes...

L'indentation des lignes d'instructions et l'utilisation d'étiquettes de branchement rendent plus clair l'architecture de votre programme.

Des noms de variables aussi explicites que vous le souhaitez, permettent de parachever la lisibilité de vos sources.

La puissance et la richesse des fonctions proposées par MEMBASIC, permettent de répondre à tous vos besoins de manipulation de chaînes de caractères, de calculs numériques, évitant toute programmation d'utilitaires de bas niveau.
De plus, les calculs de MEMBASIC sont exacts sur 14 chiffres significatifs. Ainsi, la recherche d'un centime rompant l'équilibre d'une comptabilité en fin d'exercice est un problème appartenant au passé !

Il n'est pas question de détailler ici les 156 instructions de ce Basic. Sachez tout de même que MEMBASIC réalise le contrôle syntaxique immédiat de toute ligne de programmation lors de sa saisie, vous facilitant la maîtrise de cette puissance.

Le programmeur ne passe plus son temps à la recherche d'erreurs cachées parmi de multiples instructions, la phase de tests d'exécution peut être déclenchée à tout moment.
MEMBASIC propose alors des fenêtres de TRACE et d'exécution de programmes, indépendantes de la fenêtre de commandes. Les erreurs d'exécution sont signalées par des messages explicites les commentant et les localisant.

Les économies réalisées en temps de programmation et de mise au point des programmes réduisent considérablement le stress du programmeur !

**PROGRAMMER REDEVIENT UN PLAISIR.**

Vous découvrirez les nombreuses autres possibilités de MEMBASIC dans la section de référence qui lui est consacrée. En particulier, comment produire des programmes ni listables, ni modifiables ... car il ne faudrait pas que d'autres détournent à leur profit le résultat de vos efforts, même si ceux-ci sont réduits au minimum par **MEMSOFT.**

Ces quelques éléments sur MEMBASIC vous donnent sans doute envie de commencer à travailler avec **MEMSOFT.** Ce qui va suivre devrait vous en persuader définitivement...

**MEMFILE :**

La gestion des fichiers augmente la taille des programmes de façon considérable : pour toute action sur un fichier, il est généralement nécessaire de définir les variables du fichier, ses index ...

Les problèmes se multiplient si l'on souhaite accéder aux informations d'une fiche de différentes manières (le nom ou le code postal dans un carnet d'adresses et pas seulement un numéro d'ordre de création qui ne correspond à aucune réalité ...).

Avec MEMFILE, ces difficultés (et bien d'autres...) s'effacent.



Une quinzaine d'ordres suffisent à traiter complètement vos besoins de manipulation de fichiers.

Prenons le cas de votre carnet d'adresses : vous l'extrayez de votre sac, vous l'ouvrez, vous consultez la page des C et vous trouvez très rapidement l'adresse et le numéro de téléphone de votre ami Serge CARON.

Avec MEMFILE, vous programmez toutes ces actions aussi simplement, mais votre carnet d'adresses sera infiniment plus puissant :

- Vous ne serez pas limité par des problèmes de place. Le nombre de personnes que vous inscrirez à la page des C dans votre carnet d'adresses ne sera pas limité par une taille maximum de page.

- Un changement d'adresse d'une de vos connaissances ne nécessite plus de raturer son ancienne adresse, il vous suffit de la modifier.

- Vous pouvez consulter votre carnet non seulement dans l'ordre alphabétique des noms, mais aussi selon l'ordre des dates de naissance pour n'oublier aucune date d'anniversaire. Vous pouvez connaître l'ensemble de vos connaissances dans une ville ...

**PLUS RIEN NE VOUS EST INTERDIT, LA DERNIERE LIMITE A AFFRONTER EST CELLE DE VOTRE IMAGINATION !**

Chaque ordre de MEMFILE correspond à une simple opération que vous effectueriez à la main. MEMFILE élimine la programmation purement informatique (définitions de taille de fichiers, de taille d'enregistrements ... sont prises en charge sans que jamais vous ayez à vous en préoccuper).

Vous définirez, une fois pour toutes, la structure de votre fichier comme vous auriez pu la dessiner sur fiche cartonnée avec l'emplacement du nom, de l'adresse, de la ville, du code postal et du numéro de téléphone.
Etape primordiale de la création d'un fichier, cette définition d'un dictionnaire propre à chaque fichier simplifiera votre programmation de l'ouverture du fichier à sa fermeture en passant par l'ajout, la lecture, la modification ... d'enregistrements.

Ainsi pour lire l'ensemble des données sur Monsieur DUPONT, il suffira d'indiquer que la recherche se fait sur ce nom puis de demander à MEMFILE de lire la fiche.

Cela se fait en une seule ligne de programme, et vous permet d'accéder directement à toutes les informations concernant Monsieur DUPONT.

Les fichiers de MEMFILE sont structurés de manière automatique en arbre balancé (B-TREE) ce qui vous garantit un accès très rapide à vos données.

Vos fichiers ne connaîtront qu'une limite : la capacité physique du disque sur lequel vous travaillez ... et, bien sûr, vous n'avez jamais à pré-définir de taille maximale (ni même minimale ou encore moyenne !) de fichier ou d'enregistrement de fichier.

Un seul fichier de données pourra occuper 10 millions de caractères sur disque dur sans la moindre difficulté, et sans que vous ayez à vous préoccuper de gérer l'espace disque d'aucune manière !

**MEMFILE VOUS SURPRENDRA AGREABLEMENT PAR LA FACILITE ET LA RAPIDITE AVEC LAQUELLE VOUS POURREZ GERER VOS FICHIERS.**

## MEMSCREEN :

Le multi-fenêtrage et la souris se sont aujourd'hui imposés comme une norme minimale.

Pour le programmeur, cela nécessite généralement une programmation extrêmement longue et complexe pour des résultats parfois médiocres.

Avec MEMSOFT le programmeur retrouve le sourire :
**MEMSCREEN INTRODUIT UNE NOUVELLE CONCEPTION DE LA SAISIE ET DE L'AFFICHAGE : LE MASQUE.**

Un MASQUE comprend des textes fixes et des zones de saisie, l'ensemble étant associé à une fenêtre.
A chaque zone sont associés des contrôles de saisie, des formules de calcul, des formats d'affichage.

Ainsi une seule instruction permet d'afficher et d'effectuer une séquence de saisie avec ses contrôles, la correction des erreurs de frappe, le déplacement de zone en zone, le recalcul instantané et permanent des zones de résultat ...



Sans la moindre programmation pour le développeur, les utilisateurs disposent de fonctionnalités de multi-fenêtres et de souris (modification des tailles des fenêtres, déplacement des fenêtres, positionnement du curseur...).

MEMSCREEN rend les fenêtres aussi simples à concevoir pour le programmeur qu'à manipuler pour l'utilisateur !

## DES AIDES EN PERMANENCE :

Avec **MEMSOFT** vous n'êtes jamais seul en train de programmer.

Si une question se pose (quelle est la syntaxe de cet ordre, comment utiliser cette fonction, etc...), une simple pression sur la touche d'aide fait apparaître une nouvelle fenêtre contenant les explications que vous souhaitez.

Vous offrirez ce même niveau de qualité aux utilisateurs des programmes que vous créerez : il leur suffira d'appuyer sur la touche d'aide pour accéder au mode d'emploi de votre application.

Bien entendu, ces aides ne nécessitent aucune programmation pour être réalisées.

Vous ne craindrez plus jamais de laisser utiliser vos programmes par des novices ! Les utilisateurs seront sécurisés par la possibilité d'appeler ces aides en permanence.

Vos programmes se différencieront définitivement par leur fini et leur perfection. Ils seront conformes aux standard de qualité des meilleures applications professionnelles (ergonomie, finition...).

## A VOUS DE JOUER !

Si ce rapide tour d'horizon vous a convaincu, vous comprenez désormais à qui s'adresse **MEMSOFT** : à vous !

**MEMSOFT est destiné à tout utilisateur d'APPLE II GS souhaitant réaliser des applications de qualité sans rencontrer d'insurmontables difficultés.**

Le programmeur professionnel trouvera, bien sûr, avec MEMSOFT, le langage qui épargne de longues heures de travail (passées à réaliser des prouesses techniques pour suppléer ce qu'un langage devrait normalement effectuer).

L'étudiant (et l'enseignant) dispose d'un formidable outil d'apprentissage. Les aides toujours présentes et le contrôle syntaxique accélèrent la prise en main du produit. L'intégration des détails et la maîtrise rapide d'un séquentiel indexé permettent de progresser en permanence en se consacrant à l'essentiel : l'analyse et l'algorithmique.

L'amateur de programmation possède un langage puissant mais suffisamment facile d'utilisation. Il peut enfin achever les programmes qu'il entreprend. Il présentera des réalisations qui n'auront à pâlir devant aucun travail de professionnel.

Enfin, les utilisateurs de logiciels (particuliers, professions libérales, entreprises...) disposent, rapidement, de programmes adaptés réellement à leur besoin et d'une extrême simplicité d'utilisation.

**Vous ne pourrez bientôt plus vous passer de MEMSOFT : outil extraordinairement simple et puissant pour le programmeur, il offre à l'utilisateur un confort et une facilité de travail incomparables.**



# 3 PRESENTATION DU GUIDE

Cet ouvrage décrit précisément chaque fonction de MEMSOFT.
Voici comment trouver très rapidement toutes les informations que vous pourriez être amené à rechercher.

Outre l'introduction que vous parcourez, ce livre est structuré en trois sections.

**Section II : MEMBASIC**

**Section III : MEMSCREEN**

**Section IV : MEMFILE**

Ces 3 sections constituent le guide de référence de MEMSOFT.

Chacune des sections possède une structure similaire.

Le premier chapitre donne les informations générales nécessaires à la compréhension de la section.

Le corps de la section détaille toutes les possibilités des différentes instructions par des "fiches fonctions" classées suivant l'ordre alphabétique.

Chacune des fiches se présente de la façon suivante :

**But de l'instruction** : décrit brièvement le rôle de l'instruction.

**Syntaxe** : donne la syntaxe complète d'une instruction ainsi que ses paramètres.

**Explication** : décrit en détails le fonctionnement de l'instruction.

**Exemple** : présente un exemple de programme faisant appel à la fonction décrite.

Erreurs : fournit la liste des erreurs pouvant être détectées par MEMSOFT lors de l'exécution de cette instruction.

Conseils : apparaît pour donner des conseils relatifs à la meilleure utilisation de la fonction.

Remarque : éclaire sous un jour particulier l'utilisation d'une fonction ou signale l'existence de fonctions complémentaires.

Pour découvrir l'instruction ou la commande MEMSOFT adaptée à votre besoin, vous trouverez à la fin de chaque section une liste récapitulative des instructions classées par domaine d'utilisation.

De plus, à la fin de l'ouvrage, un index général vous permet d'accéder rapidement à l'endroit du livre où chaque point est traité.



# 4 MISE EN OEUVRE DE MEMSOFT

**Mise en place de la protection.**

**La protection se fait avec la carte électronique que vous avez eu avec ce produit.**

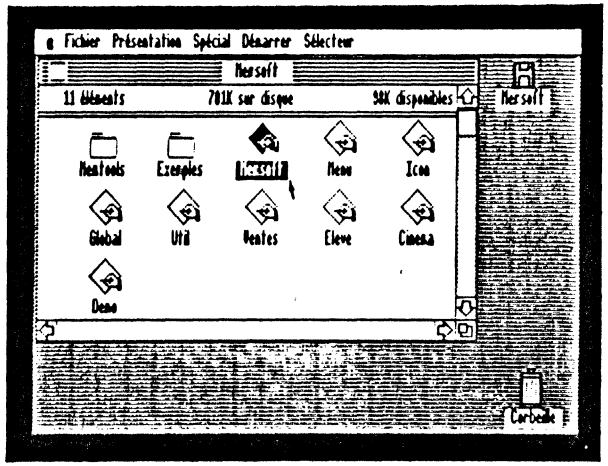
**Pour installer la protection ETEIGNEZ VOTRE ORDINATEUR.**

**Ouvrez le, et placez la carte électronique dans le slot numéroté 4.**

**REMARQUE : le slot 4 est réservé à la gestion de la souris, cependant le fait de mettre la carte électronique dans le slot 4 ne gêne pas cette gestion. En conséquence, ne modifiez pas l'option du tableau de bord pour ce slot.**

**Le démarrage de MEMSOFT est extrêmement simple.**

a) Avec un seul lecteur de disquettes :
Placez la disquette système PRODOS dans le lecteur de disquettes, puis mettez en marche votre APPLE II GS. Lorsque le système a été chargé, remplacer la disquette système PRODOS par la disquette MEMSOFT dans ce lecteur. L'icône de la disquette MEMSOFT apparaît alors à l'écran. Cliquez deux fois sur cette icône, et l'écran suivant apparaît bientôt :



b) avec deux lecteurs de disquettes :
Mettez la disquette système PRODOS dans le lecteur 1 et la disquette MEMSOFT dans le lecteur 2. Lorsque l'icône MEMSOFT apparaît à l'écran, cliquez deux fois sur cette icône.

c) avec un disque dur :
Copiez la disquette MEMSOFT sur votre disque. Pour cela mettez la disquette MEMSOFT dans le lecteur 1, cliquez sur l'icône MEMSOFT, en maintenant le bouton de la souris enfoncé, et amené l'icône MEMSOFT sur l'icône du disque.

Vous pouvez :

* soit lancer MENU pour accéder aux exemples de programmes fournis avec MEMSOFT. (Vous pouvez également lancer séparemment ces différents programmes.)

* soit lancer MEMSOFT, pour créer vous-même de nouveaux programmes.

Pour cela il suffit de sélectionner l'icône MEMSOFT avec la souris, et de déclencher l'exécution de ce programme avec l'option "ouvrir fichier" du bureau ou par un "double clic".

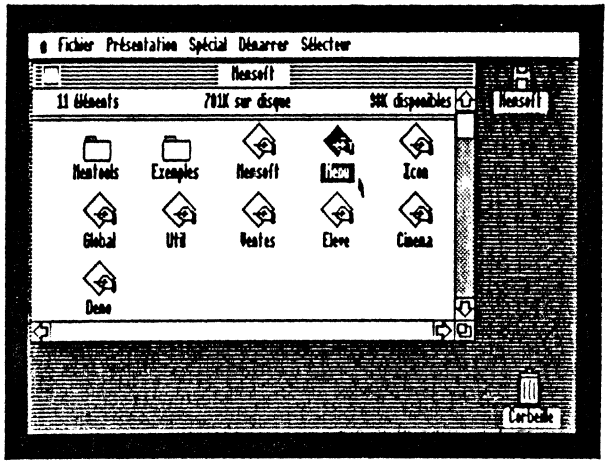
Si la carte de protection n'a pas été installée, lors du chargement de MEMSOFT le message suivant apparaît :

**La carte MEMSOFT manque, enfoncez ESC pour sortir.**

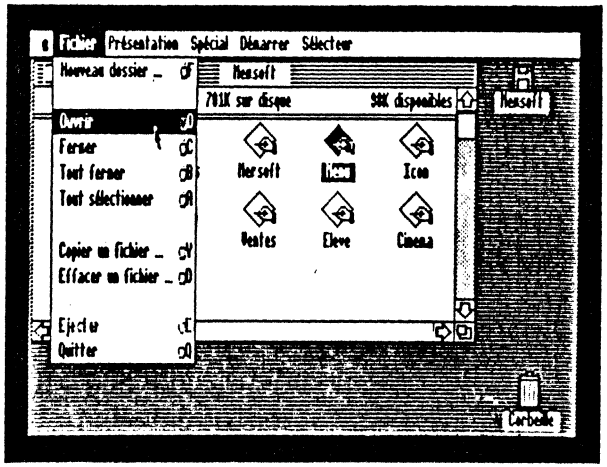


**LES EXEMPLES DE PROGRAMMES MEMSOFT**

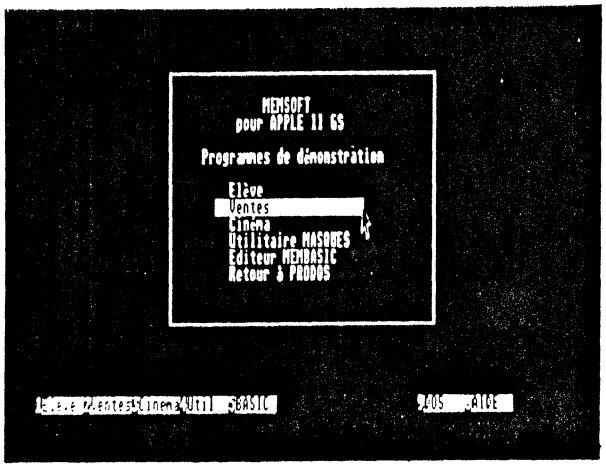
Pour faire fonctionner ces exemples, il suffit de sélectionner l'icône MENU avec la souris :




Exécutez ensuite ce programme avec l'option "Ouvrir" du menu ou par un "double clic" :

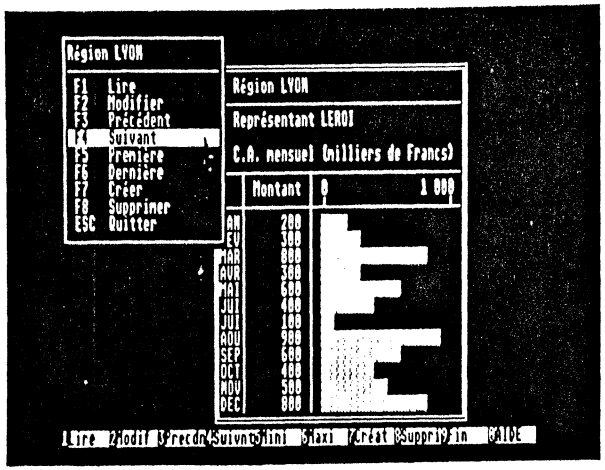


MEMSOFT se charge et lance le programme MENU. Une fois MEMSOFT chargé (si vous avez démarré avec un seul lecteur de disquettes, vous serez obligé de remplacer temporairement la disquette MEMSOFT par la disquette système PRODOS durant le chargement de MEMSOFT), l'écran suivant s'affiche :




Avec la souris (ou les touches de fonction), choisissez un exemple et regardez comment il fonctionne : vous pourrez réaliser vous aussi des programmes comme ceux-ci. Vous verrez un peu plus loin comment les lister. Inspirez-vous de ces programmes s'ils vous ont plu.

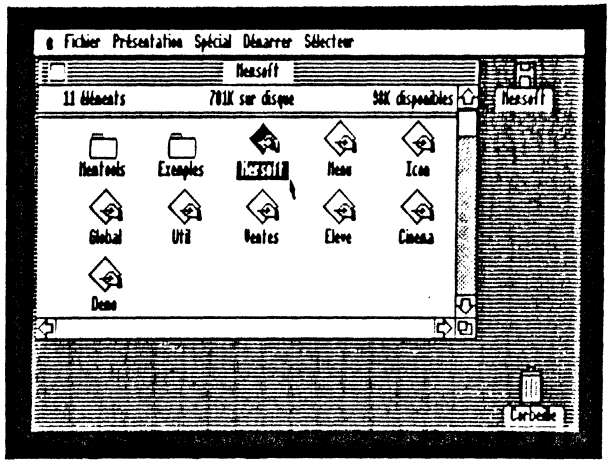
Voici, par exemple, une image de VENTES :



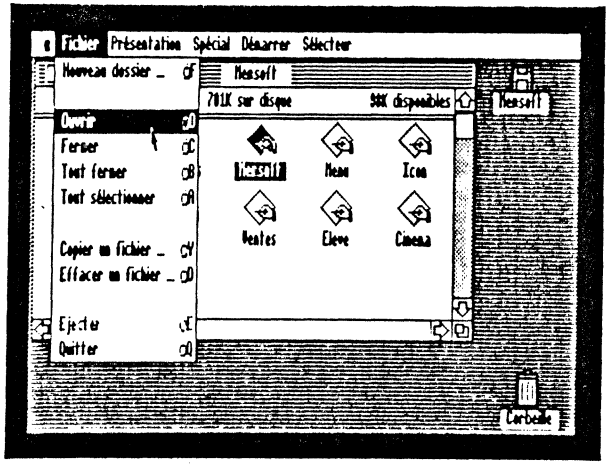



**PROGRAMMER**

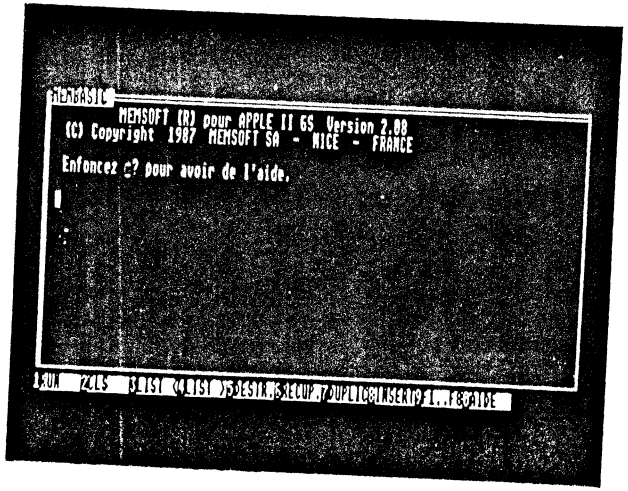
Pour réaliser vos propres programmes, démarrez MEMSOFT en sélectionnant l'icône MEMSOFT avec la souris :




Lancez ensuite cette application avec l'option "Ouvrir" du menu "Fichier" ou par un "double clic" :



MEMSOFT se charge et vous donne la main. Vous voyez l'écran suivant qui est celui de l'éditeur :

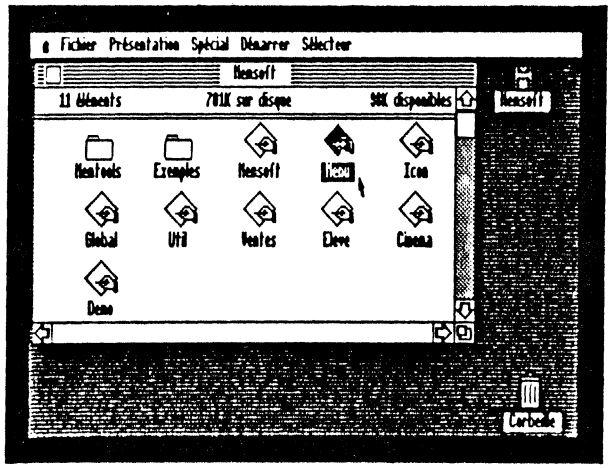



A noter : on peut aussi lancer MEMSOFT depuis le bureau par MENU, comme précédemment, et choisir l'option Editeur MEMBASIC du menu. Le résultat est le même.

Vous trouverez un peu plus loin, un chapitre appelé "VOS PREMIERS PAS" qui indique les quelques commandes indispensables à la réalisation de votre premier programme.



## LE CONTENU DE LA DISQUETTE




### MEMSOFT

C'est l'icône qui permet de charger MEMSOFT. Sélectionnez l'icône, ouvrez le fichier et MEMSOFT se charge.

### MENU

C'est un programme MEMBASIC qui permet de lancer les programmes d'exemples qui sont :

UTIL : Un programme d'aide à la création de masques.

VENTES : Un exemple simple de gestion de chiffres d'affaires par région.

ELEVE : La gestion d'une classe avec sélection sur les notes et édition.

CINEMA : Un exemple de gestion multi-fichiers ... et multi-fenêtres.

UTIL, VENTES, ELEVE, CINEMA

Ces quatres icônes permettent de charger les programmes correspondants, sans passer par le menu.

DEMO

Programme de démonstration de MEMSOFT.

Dossier MEMTOOLS

Ce dossier contient MEMSOFT.OVL, c'est MEMSOFT lui-même, et les aides de MEMSOFT. Ces aides sont accessibles quel que soit le dossier (ou répertoire) où vous vous trouvez. C'est une particularité de MEMTOOLS : les programmes, masques ou aides, non trouvés dans le répertoire courant, sont automatiquement recherchés dans le répertoire MEMTOOLS. Vous pouvez y recopier le programme UTIL et ses écrans pour pouvoir l'utiliser en permanence.

En cas d'installation sur un disque dur, ce dossier doit y être copié dans la racine pour pouvoir utiliser les aides.

Dossier EXEMPLES

Ce dossier contient tous les programmes, masques et fichiers des programmes d'exemple.

ICON. GLOBAL

Les programmes ICON et GLOBAL sont des utilitaires nécessaires à la finition des logiciels MEMSOFT. Le chapitre 7 vous donne toutes les explications nécessaires à leur mise en oeuvre.



VOS PREMIERS PAS ...

Lorsque vous êtes dans l'éditeur MEMBASIC, voici les quelques commandes de base les plus utiles :

Pour lister et consulter nos exemples :

CHDIR "/<nom>" : vous place dans un dossier (ou répertoire) particulier.
Par exemple : CHDIR "/EXEMPLES".

LOAD "<nom>" : charge un programme.
Par exemple LOAD "ELEVE" charge l'exemple ELEVE.

RUN ou
RUN "<nom>" : lance l'exécution d'un programme.

SYSTEM : quitte MEMSOFT et retourne à PRODOS.

Pour créer vos propres programmes :

NEW : efface la mémoire pour vous permettre de commencer un nouveau programme.

MKDIR "/<nom>" : crée un nouveau répertoire sur le disque (par exemple MKDIR "/ESSAI" pour y sauver vos programmes d'essais). Ne pas oublier de s'y placer par CHDIR.

SAVE "<nom>" : sauve un programme que vous avez tapé.

SAVE "@<nom>" : sauve en écrasant une ancienne version.

Pour plus d'informations sur ces commandes, reportez vous à la section II - MEMBASIC.

LES DIFFERENTS MODES

L'APPLE IIGS supporte outre les modes texte et graphique deux nouveaux modes d'affichage qui sont :

- le mode graphique basse résolution : 16 couleurs

- le mode graphique haute résolution : 4 couleurs

Pour plus d'informations se reporter à la documentation de la machine.

MEMSOFT fonctionne exclusivement dans le mode haute résolution 4 couleurs qui seul, permet l'affichage en couleur de 80 caractères dans la largeur de l'écran. Néanmoins, un jeu de couleurs différent peut être utilisé pour la ligne de fonctions disponibles en bas de l'écran.
MEMSOFT gère ce mode haute résolution de deux façons différentes suivant le type d'affichage choisi dans le tableau de bord :

- le type couleur, utilise les 4 couleurs disponibles,

- le type monochrome n'utilisera que deux couleurs (plus une surbrillance) qui rendra l'affichage plus lisible sur un écran noir et blanc.

Pour de plus amples explications concernant le choix des couleurs, se reporter à la description du programme ICON.



# 5 LA GESTION DE FENETRE

**Pourquoi des fenêtres ?**

Une nouvelle dimension apparaît depuis quelques temps dans le monde des logiciels : les fenêtres.

**MEMSOFT** propose une gestion de fenêtres indépendante du programme d'application, mais que celui-ci utilise comme si elle lui était parfaitement intégrée.

L'intérêt?

Tout logiciel **MEMSOFT**, même le plus simple, offre le luxe d'une gestion de fenêtres complète et facile à mettre en oeuvre.

Pour l'utilisateur du logiciel, il en résulte un confort accru.

Les différentes fonctions de votre logiciel utilisent les fenêtres comme des fiches posées sur un bureau, les recouvrant les unes par les autres, faisant revenir à la surface les plus "enfouies", menant plusieurs actions en même temps ... circulant en profondeur dans l'espace de travail.

L'utilisateur pourra entreprendre plusieurs traitements successifs sans perdre le fil de son travail : consultations, modifications seront effectuées comme des traitements annexes, répondant en temps réel aux besoins.

La grande souplesse permise dans la manipulation des fenêtres, dimensionnements, couleurs, déplacements, permet de personnaliser l'utilisation des logiciels.

MEMSOFT l'a voulu ainsi pour votre plus grande satisfaction.

## QUELQUES DEFINITIONS :

Vous entendrez, par la suite, parler d'un certain nombre de notions, il convient de bien s'accorder sur leur sens.

### UNE FENETRE

C'est une partie de l'écran, généralement délimitée par un cadre. Vous devez choisir la surface de l'écran affectée à chaque fenêtre, c'est-à-dire sa taille et sa position.

### UN MASQUE

C'est un espace à deux dimensions pouvant comporter 250 lignes de 250 caractères. Un masque peut donc dépasser la taille de l'écran. Chaque logiciel propose une panoplie de masques à travers lesquels s'effectuent les saisies, consultations, etc... Le contenu de chaque masque est affiché dans la fenêtre propre au masque.

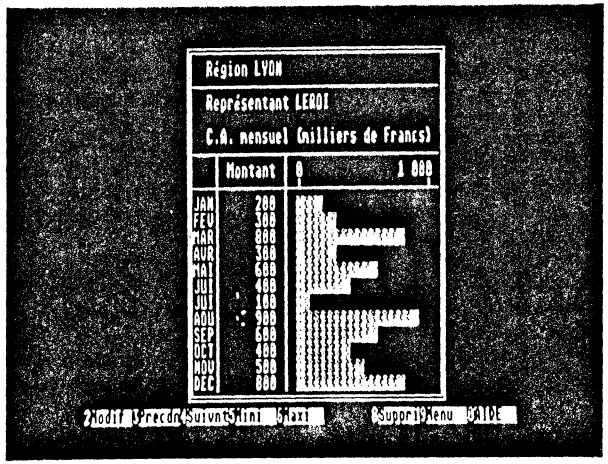
### LA SOURIS

La souris, bien connue des utilisateurs de l'APPLE II GS, est matérialisée à l'écran par un caractère spécial appelé curseur souris. La forme du curseur souris n'est pas toujours la même. Elle dépend de sa position sur l'écran et sur les fenêtres de MEMSOFT.

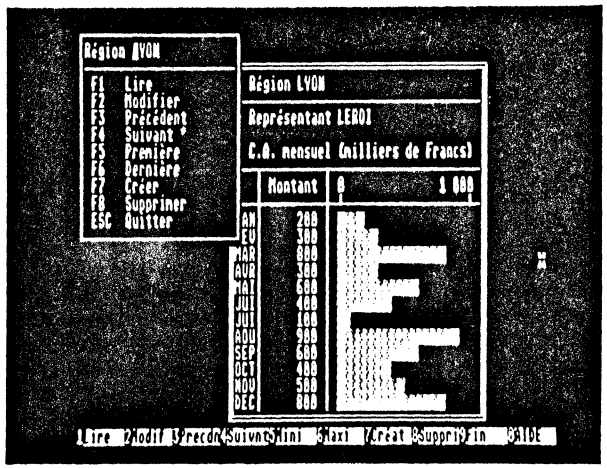


## VOUS AVEZ LA MAIN

Dès que le curseur souris cesse de représenter une montre, vous avez la main : le clavier et la souris agissent immédiatement.




MEMSOFT travaille pour vous : vous n'avez pas la main.



## MANIPULATION DE FENETRES

La souris permet d'exécuter de façon très simple un certain nombre de manipulations de fenêtres :

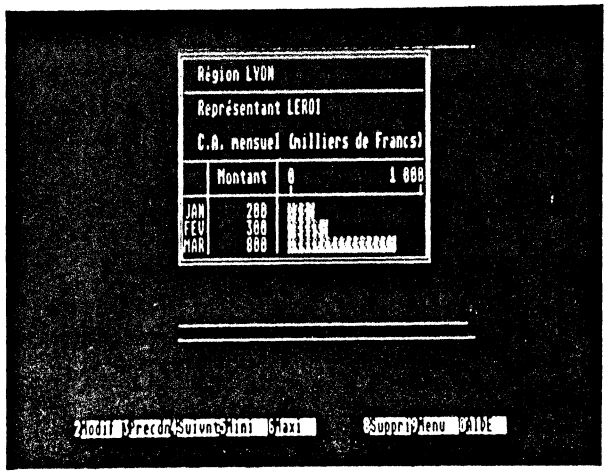
- CHANGEMENT DE TAILLE
- DEFILEMENT
- DEPLACEMENT
- POSITIONNEMENT
- DECLENCHEMENT DE FONCTION
- DEPLACEMENT EN PROFONDEUR

Elle sert, et c'est son usage le plus fréquent, de raccourci dans toutes les opérations de saisie : vous déplacez le curseur directement à l'endroit souhaité.

Nous allons passer en revue les différentes formes du curseur souris et les fonctionnalités associées.

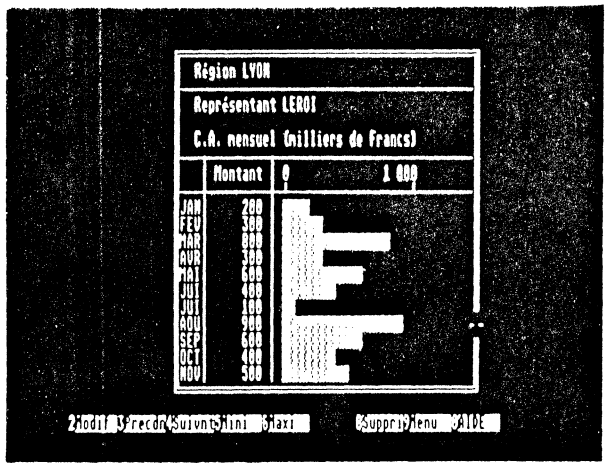


## CHANGEMENT DE TAILLE




Ce type de curseur ◊ est obtenu en déplaçant la souris de façon à pointer sur le coin inférieur droit du cadre d'une fenêtre.

Pour changer la taille de la fenêtre, il suffit de maintenir enfoncé le bouton de la souris, les déplacements de celle-ci entraîneront un déplacement du coin inférieur droit, le coin supérieur gauche restant fixe. Seules des lignes représentant le cadre montrent le déplacement souhaité qui se concrétise dès que l'on relâche le bouton de la souris. Le contenu est alors automatiquement mis à jour pour correspondre au contenu du masque aux endroits visibles.




Ces formes du curseur ⇧ et ⇩ sont obtenues en le plaçant sur l'un des côtés verticaux ou sur le côté inférieur du cadre.

Pour un défilement vers le haut, on place le curseur sur la moitié supérieure d'un côté vertical et l'on clique le bouton gauche de la souris.

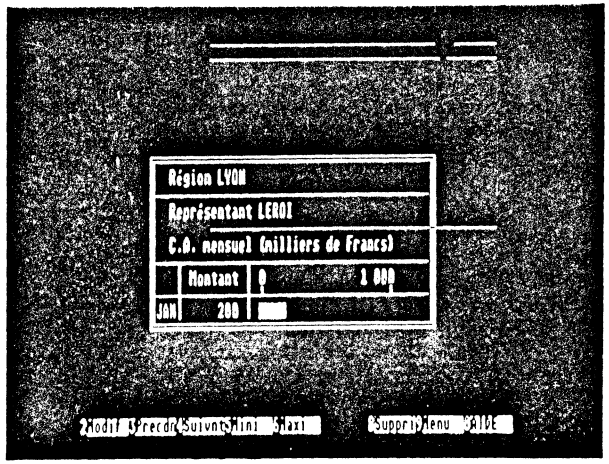
Pour un défilement vers le bas, l'opération est la même mais on place le curseur sur la moitié inférieure d'un des côtés verticaux.

Le texte du masque avancera d'une ligne chaque fois que vous cliquerez. Le défilement s'accélère si l'on maintient le bouton enfoncé.

Le même principe s'applique pour le défilement droite-gauche en pointant la ligne horizontale du bas du cadre.



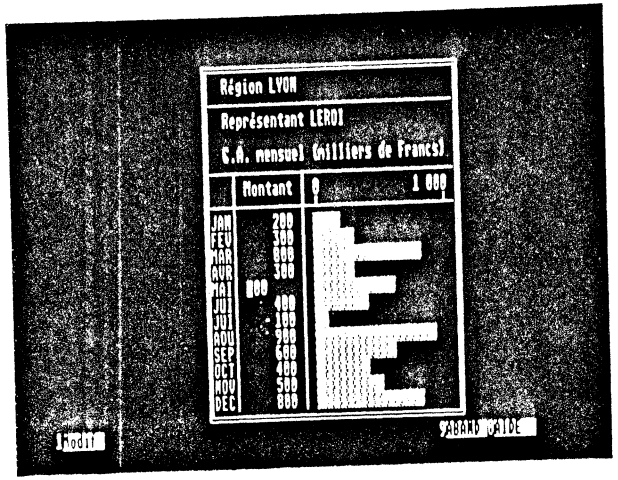





Ce curseur ▽ indique que la fenêtre peut être déplacée sans modification, ni de sa taille, ni de son contenu.

Ce type de curseur est obtenu en déplaçant la souris de façon à pointer sur la ligne supérieure du cadre d'une fenêtre.

Pour déplacer la fenêtre, il suffit de maintenir enfoncé le bouton de la souris. Les déplacements de cette dernière entraîneront un déplacement d'un cadre filiforme représentant la fenêtre déplacée. Le déplacement est effectif dès que l'on relâche le bouton de la souris.




Le curseur est dans la fenêtre active, c'est-à-dire celle dans laquelle l'utilisateur fait ses saisies. Il est possible de demander un déplacement du curseur de saisie.

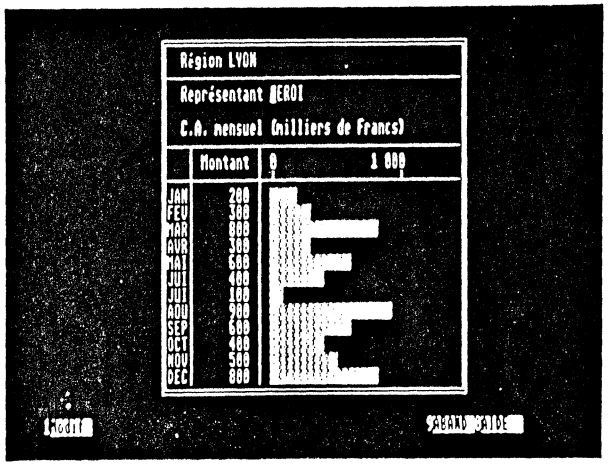
Pour cela, après avoir placé le "curseur souris" à l'endroit désiré, enfoncez puis relâchez le bouton de la souris.

Si le logiciel valide la demande, le curseur de saisie se place à la position du curseur souris.

Par exemple, lors d'une saisie dans un masque MEMSCREEN, on pourra "sauter" de zone en zone librement parmi toutes les zones autorisées.





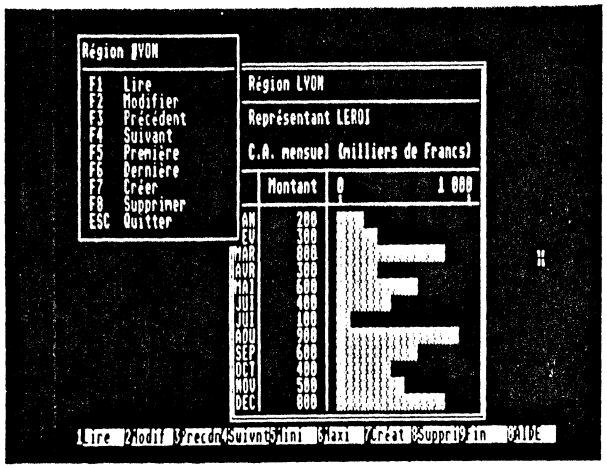



Le curseur n'a cette forme ↯ que s'il est sur la dernière ligne de l'écran où sont situés les messages associés aux touches de fonction.

En enfonçant puis relâchant l'un des boutons de la souris sur un message, la touche de fonction correspondante est sélectionnée. L'effet pour le logiciel est identique à celui que provoquerait l'enfoncement de la touche de fonction.(voir chapitre 6).

Un cas particulier : lors de la consultation des fichiers d'aide, la ligne de bas d'écran est différente et par cette méthode, la souris permet d'avancer ou revenir en arrière dans le texte d'aide ou de retourner au logiciel.

DEPLACEMENT EN PROFONDEUR




Le curseur ⇕ a cette forme lorsqu'il pointe dans une fenêtre qui n'est pas la fenêtre active (celle dans laquelle la saisie a lieu).

La seule action possible est de faire passer cette fenêtre au premier plan.

Pour cela, il suffit d'enfoncer puis de relâcher le bouton de la souris.

Si la fenêtre qui passe au premier plan gêne la saisie, elle sera aussitôt repoussée. Dans ce cas, il faut déplacer l'une des fenêtres, ou le curseur de saisie, et essayer de nouveau.

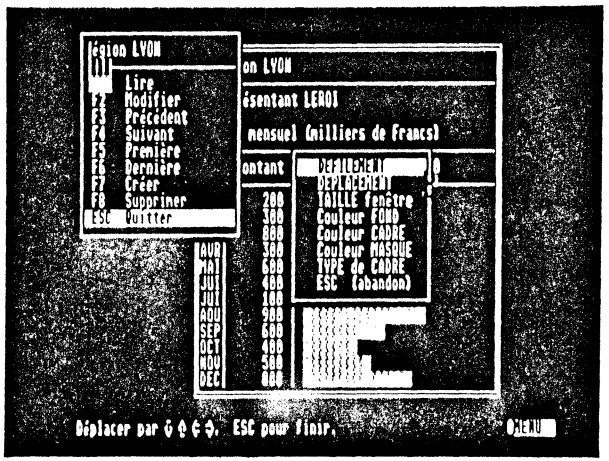


FONCTIONNALITES COMPLEMENTAIRES

Le choix de la couleur du FOND, du CADRE, du MASQUE et du type de CADRE n'est pas directement possible avec la souris : il faut passer dans un mode de contrôle par menu, ce mode sera désormais désigné par : **MODE FENETRE.**

On passe dans ce mode, en maintenant enfoncée la touche OPTION tout en "cliquant". La fenêtre courante est alors désignée par une main très stylisée.

La touche ESC permet de sortir de ce mode.
Les touches ⌘ O affiche le menu :



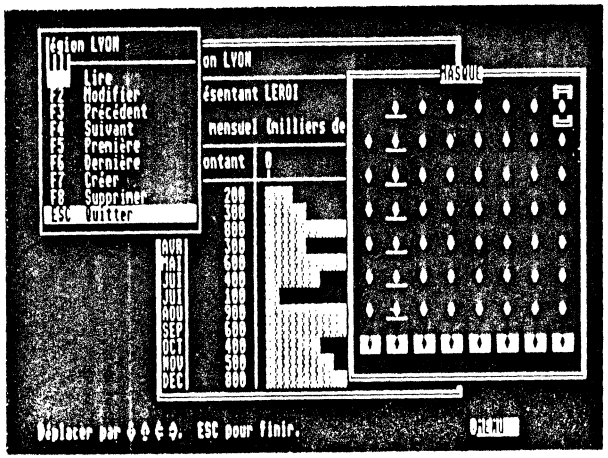

Dans ce mode les touches "+" et "-" permettent de déplacer cette main de fenêtre en fenêtre et donc de désigner comme fenêtre courante n'importe laquelle des fenêtres.

Déplacez la barre du menu avec les flèches et validez par "Retour Chariot", ou "cliquez" avec la souris.

Chaque validation de l'option type de cadre vous propose le type suivant de cadre parmi les huit possibilités.

La validation d'une option de couleur propose une palette.

Choisissez une couleur et validez.




La palette de couleurs de fond est enrichie par l'utilisation de la touche "S" permettant de parcourir une série de palettes différentes.

La touche "I" offre aux possesseur d'un APPLE II GS avec un écran monochrome un affichage des caractères avec une brillance différente, permettant de mettre en relief une partie du texte. Cet effet ne sera pas visible en mode couleur.

La couleur du masque permet de préciser la couleur des caractères saisis dans un masque. En particulier, en création ou en modification de masque, il permet de choisir la couleur de tout caractère du masque. Pour le cadre, il faut utiliser l'option spécifique.



# 6 L'ENVIRONNEMENT

Après avoir chargé MEMSOFT, votre environnement habituel sera, sur quelques points, modifié. En particulier, en ce qui concerne les touches du clavier, le nom et le suffixe d'un fichier, le nom d'accès d'un fichier,.

## LE CLAVIER

Le clavier de l'APPLE II GS se présente comme un clavier de machine à écrire avec un pavé numérique. (Voir la documentation de la machine pour plus de renseignements).
En général, les touches du clavier numérique ont la même fonction que celle du clavier alphanumérique correspondante. Vous pouvez taper les chiffres indifféremment sur l'un ou l'autre clavier. Cependant, sous MEMSOFT, ces touches ont un rôle particulier, elles peuvent être des touches de fonction. Ces touches permettent de réduire au minimum de nombreuses procédures intervenant dans les programmes.

Utilisées en combinaison avec la touche Ó , les touches du clavier numérique définissent les touches de fonction suivantes : F1, F2, F3, F4, F5, F6, F7, F8, F9, F0.
Le rôle des touches de fonction dans MEMBASIC est décrit dans le chapitre I de la section MEMBASIC, sinon, pour les logiciels, le rôle de ces touches est déterminé par programme. (voir instruction FKEY)

Prenons un exemple, pour cela chargez MEMSOFT.

La ligne située au bas de l'écran indique la signification implicite des touches de fonction.
F8, par exemple, permet d'insérer une ligne blanche. Pour obtenir cette fonction il faut donc, comme nous l'avons expliqué précédemment que vous enfonciez simultanément la touche Ó et la touche 8 (POMME 8) du clavier numérique. A ce moment là une ligne blanche s'est insérée à l'endroit où était positionné le curseur.
Avec la souris c'est encore plus simple. En enfonçant puis relâchant le bouton de la souris sur un message, la touche de fonction correspondante est sélectionnée.

Par la suite, lorsque nous parlerons de la touche de fonction **F3**, n'oubliez pas qu'il faudra en fait enfoncer les touches  et **3 (POMME 3)**.

Parfois la combinaison des touches est un peu plus complexe. Pour :

**CTRL F3** enfoncez les touches **CTRL  3 (CTRL POMME 3)**

Une planche de pastilles autocollantes vous a été livrée avec ce produit. Ces pastilles vous permettent de modifier le clavier de votre **APPLE II GS** afin de faire apparaître les commandes reconnues par **MEMSOFT** et accessibles par la frappe simultanée de la touche **POMME** et de la touche concernée. Pour mieux les distinguer des commandes préexistantes, celles-ci sont inscrites en rouge. Un certain nombre d'entre elles ne sont utiles qu'au programmeur (#, @..). Une seule exception concerne la fonction STOP d'interruption d'un programme. Celle-ci est accessible par la frappe simultanée des trois touches **POMME CTRL C.**

Une autre particularité dans l'utilisation du clavier.
En effet, vous pouvez obtenir certains caractères (que vous ne pouvez pas frapper directement) en introduisant le code ASCII propre à chacun.

Voici comment vous allez procéder :
enfoncez la touche **OPTION** (ou **ALT**), puis frappez sur le clavier numérique le code ASCII correspondant au caractère que vous voulez afficher. Lorsque vous relâchez la touche **OPTION**, ce caractère viendra s'afficher à l'écran.

Exemple : enfoncez la touche **OPTION**, et sur le clavier numérique :
143

Le caractère spécial : Å

vient de s'afficher.
Un fichier d'aide donne la liste des caractères supplémentaires et le code associé. Enfoncez **POMME 0** pour la consulter.



**QU'EST-CE QU'UN NOM DE FICHIER OU DE CHEMIN VALIDE ?**

Sous **MEMSOFT**, une spécification de fichier comprend trois éléments : l'identificateur de lecteur, le nom du fichier et le suffixe.

1) LES NOMS DE FICHIERS
Un nom de fichier sous PRODOS peut comporter jusqu'à 15 caractères et il doit commencer par une lettre.
Sous MEMSOFT, un nom de fichier est composé d'un nom qui peut comprendre de 1 à 8 caractères, et d'un suffixe composé d'un point (caractère spécial qui ne peut donc pas être utilisé dans le nom du fichier), suivi d'un suffixe de 1 à 3 caractères.
Par exemple, un masque aura pour suffixe .MSK, par défaut. Ceci a été fait dans un but de clarification, mais il vous est possible de définir un autre suffixe, ou de ne pas en mettre. Par défaut, les suffixes habituellement utilisés sont :

.PRG pour un programme
.MSK pour un masque
.MFK pour un fichier de clés MEMFILE
.MFR pour un fichier d'enregistrements MEMFILE
.AUT pour le fichier d'une séquence enregistrée
.HLP pour un fichier d'aide

ATTENTION : les syntaxes utilisables pour les noms de fichiers ou de répertoires sont sensiblement différentes sous **MEMSOFT** et sous **PRODOS.**
PRODOS n'accepte que les lettres, les chiffres et le caractère point(.). **MEMSOFT** accepte en plus les caractères *, # et @ à l'exception des chiffres en première position. Sous **MEMSOFT**, le point doit être utilisé une seule fois par nom de fichier ou de répertoire, et il marque le début du suffixe. Ces différences font qu'un nom de fichier créé sous **MEMSOFT** pourra ne pas apparaître de manière identique sur le bureau **PRODOS**. Par exemple, sous **MEMSOFT**, vous pouvez créer les fichiers suivants :

VENTE.PRG
VENTE_GS.MFR

A partir du bureau, vous verrez :

VENTE.PRG
VENTEBGS.PRG040

2) LES CARACTERES GENERIQUES

Les caractères spéciaux ? et * peuvent être inclus dans un nom de fichier et son suffixe. Ils permettent d'utiliser certaines commandes avec plus de souplesse.

Le caractère ?

Un point d'interrogation dans un nom de fichier ou son suffixe indique qu'un caractère quelconque peut occuper cette position. Par exemple :

```
DIR "TER?E.PRG"
```

donne la liste de tous les fichiers du répertoire de l'unité par défaut dont le nom comporte cinq caractères, commence par TER suivi d'un caractère quelconque et continue par E, et qui ont .PRG comme suffixe.

Voici un exemple de ce qui pourrait s'afficher :

```
TERME.PRG
TERRE.PRG
TERNE.PRG
```

Le caractère *

Un astérisque dans un nom de fichier ou son suffixe signifie que tout caractère peut occuper cette position et les suivantes. Par exemple :

```
DIR "TER*.PRG"
```

donne la liste de tous les fichiers du répertoire courant dont le nom commence par TER et qui ont .PRG comme suffixe. Le nom de fichier peut comporter de 3 à 8 caractères.

Voilà ce que l'on pourrait obtenir :

```
TER.PRG
TERME.PRG
TERTRE.PRG
TERRASSE.PRG
```

Un autre exemple. Si vous voulez copier tous les fichiers de la disquette **MEMSOFT** sur le disque, il suffit, de taper :

```
COPY "A:*.*" TO "C:"
```



Autre utilisation :

```
DIR "*."
```

affiche tout ce qui, dans le répertoire courant, n'a pas de suffixe, par exemple des sous répertoires.

3) LES NOMS D'ACCES

Le nom d'accès **sous PRODOS** est une série de noms de dossier séparés par un slash /. Ce nom indique à PRODOS le chemin à parcourir pour parvenir au fichier voulu. Ce nom d'accès commence toujours par le nom de la disquette, puis le nom des dossiers (sous répertoires) et enfin le nom du fichier proprement dit.
Sous MEMSOFT, le nom du disque est remplacé par un identificateur (A, B, C ..) suivi du caractère (:). Cet identificateur représente l'unité de disque physique. A, désigne le premier lecteur de disquettes, B le second s'il existe. Si un ou plusieurs disques durs sont connectés à votre APPLE II GS, ils auront comme identificateurs les lettres C, D etc..
Les lecteurs de disquettes au-delà de deux se verront affectés dans l'ordre les identificateurs disponibles.
Par exemple, avec trois lecteurs de disquettes et un disque dur, on obtient :

| | |
|---|---|
| A | pour le premier lecteur de disquettes |
| B | pour le deuxième lecteur de disquettes |
| C | pour le disque dur |
| D | pour le troisième lecteur de disquettes |

Avec cette convention, pour accéder au fichier COMPTES du répertoire COMPTA de l'unique disque dur GESTION, il faut utiliser sous MEMSOFT le chemin d'accès :

```
C:/COMPTA/COMPTES
```

REMARQUE

Les caractères / et \ sont équivalents. La longueur maximum d'un nom d'accès utilisé dans une commande est de 64 caractères, slashs inclus.

Le changement de répertoires se fait avec l'instruction CHDIR (voir Section II COMMANDES, INSTRUCTION et FONCTIONS MEMBASIC).

# 7 UTILITAIRES

## GLOBAL

Cet utilitaire vous permet de regrouper dans un seul et même fichier différents objets. Il peut réunir des masques, des aides, des programmes ..etc.

L'utilisation des globaux apporte les avantages suivants:

- gain de place

- gain de vitesse : près de 30% (en supprimant le temps de recherche et d'ouverture des objets)

- simplicité d'installation : un seul fichier à installer au lieu de l'ensemble des masques, aides et programmes.

L'utilitaire peut être lancé à partir du Bureau en ouvrant le fichier GLOBAL, ou à partir de MEMSOFT en tapant :

```
CHDIR "/MEMTOOLS/GLOBAL
RUN "GLOBAL
```

CRITERES :
Critère de sélection des fichiers sur lesquel doit se porter l'action. Cette sélection peut se faire sur différents lecteurs (lecteurs de disquettes, disque), et sur différents répertoires. Il suffit à chaque ligne de préciser les chemins d'accès. Exemple :

```
A:*.PRG
C:*.*
```

Tous les programmes du lecteur de disquette A:, et tous les fichiers du disque C: constitueront le global.

Suffixes à éliminer :
Parmi tous les fichiers selectionnés par CRITERES, vous avez la possibilité, grâce à cette rubrique, d'éliminer tous ceux dont vous préciserez le suffixe.



Passez dans la zone NOM DU GLOBAL et inscrivez le nom que vous voulez donner au global. Tapez POMME 1 pour valider.

La création s'effectue. S'affichent alors à l'écran, des informations sur le nombre de critères analysés, le nombre de fichiers traités, l'entête de chaque fichier, la recopie des fichiers dans le global.

Pour comprendre à quoi correspond ces informations nous allons regarder comment est structuré un global.

Un global se partage en trois parties distinctes qui permettent à MEMSOFT d'en connaître le contenu.

**Partie 1 : Les 16 premiers octets qui sont :**

2 octets : FR (indiquent que ce fichier est un global).
2 octets : numéro de la version.
2 octets : nombre de fichiers regroupés dans le global.

Les octets restants ne sont pas utilisés.

**Partie 2 : Par fichier, 16 octets qui indiquent :**

12 octets : NOM DU FICHIER, avec son suffixe. Il est suivi d'un 0 si le nom s'écrit sur moins de 12 octets .

4 octets : TAILLE MAXIMALE DU FICHIER.

**Partie 3 : Tous les fichiers concaténés.**

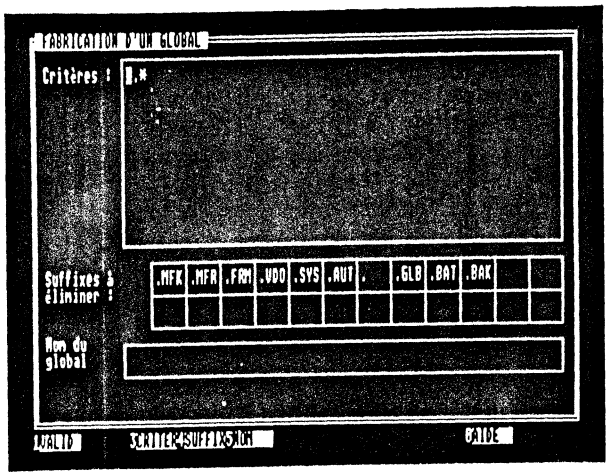
Dans l'ordre où leurs noms ont été indiqués dans la partie 2, tous les fichiers du global sont concaténés les uns à la suite des autres.

Lorsqu'on lance MEMSOFT avec un global, les parties 1 et 2 sont chargées une fois pour toutes en mémoire. Ceci permet de savoir instantanément si un objet appartient au global ou non et, si oui, quelle est sa position dans le fichier global.

Attention ! Si le global regroupe n fichiers, la taille de ces deux parties résidentes en mémoire est :

( n+1 ) * 16 octets

Par exemple, l'utilisation d'un global regroupant 300 fichiers nécessite une place mémoire supplémentaire de 4816 octets.






ICON

Programme permettant de créer des icônes.

A chaque icône correspond un programme lanceur dans lequel ont été enregistrés une fois pour toute les paramètres de l'application.

Le bureau de l'APPLE II GS permet de lancer des programmes en cliquant simplement deux fois sur l'icône qui le représente.

Cette méthode est très pratique et facilement compréhensible pour l'utilisateur. Il serait dommage que, pour lancer sa comptabilité, l'utilisateur clique sur l'icône de MEMSOFT pour être ensuite obligé de taper :

```
CHDIR "/MEMCOMPT
RUN "INCOMPTA
```

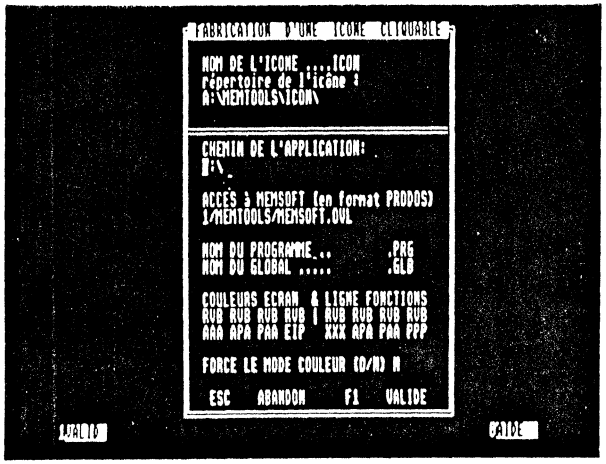
Cet utilitaire va donc permettre de créer une icône portant le nom du logiciel et qui enchaîne directement dans le bon programme MEMSOFT du bon répertoire !.

L'icône doit être de type exécutable, peut être copiée ou déplacée n'importe où sur le disque : elle fonctionnera de la même manière dans tous les cas si vous avez pris la précaution d'indiquer les répertoires de façon complète et non relativement au répertoire de lancement.

L'utilitaire peut être lancé à partir du bureau en ouvrant le fichier ICON, ou à partir de MEMSOFT en tapant :

```
CHDIR "/MEMTOOLS/ICON
RUN" ICON
```

Le masque suivant apparaît :




NOM DE L'ICÔNE :
Répertoire de l'icône :

Indiquez le nom et le répertoire de l'icône choisie. Par exemple, pour créer une icône COMPTA située dans le répertoire principal du disque dur, répondez :

NOM DE L'ICÔNE :.... COMPTA
Répertoire de l'icône :
C:\

NB : Afin de mieux comprendre la syntaxe utilisée pour désigner les lecteurs de disques ou les répertoires, se reporter à la section I, chapitre 6.

Le programme vérifie alors si cette icône existe. Si oui, il demande si vous voulez la modifier. Tapez POMME 1 si oui.

Voici les questions qui vous sont ensuite posées :

CHEMIN DE L'APPLICATION : (en format MEMSOFT)
**/MEMCOMPT**

C'est le répertoire où il faut se placer avant de lancer MEMSOFT. On y cherchera le global.



ACCES A MEMSOFT (en format PRODOS)
**/MEMSOFT/MEMTOOLS/MEMSOFT.OVL**

On indique le nom du disque et le répertoire où doit se trouver MEMSOFT.OVL. Il est recommandé que le répertoire soit toujours MEMTOOLS pour ne pas avoir plusieurs versions de MEMSOFT sur le disque !
Il est possible de ne pas indiquer explicitement un nom de disque. Vous pouvez utiliser pour cela les préfixes 1/ ou 2/. 1/ désigne le disque qui contient l'icône, 2/, désigne le disque système PRODOS qui a servi à démarrer votre APPLE II GS. Par exemple :

2/MEMTOOLS/MEMSOFT.OVL provoquera la recherche de MEMSOFT.OVL dans le répertoire MEMTOOLS du disque courant.

NOM DU PROGRAMME ... **INCOMPTA** .PRG

C'est le nom du premier programme à lancer. Il peut être dans le global.

NOM DU GLOBAL ..... **CPTGLG** .GLB

C'est le global à utiliser pour l'application. Voir le chapitre précédent pour plus d'informations. L'indication d'un global est facultative.

COULEURS ECRAN & LIGNE FONCTIONS
RVB RVB RVB RVB RVB RVB RVB RVB
**AAA APA PAE EIP XXX APA PAA PPP**

Ces paramètres permettent de modifier les couleurs. Lesquelles ?

COULEUR ECRAN
La première colonne de gauche correspond à la couleur du fond de l'écran, la suivante à la couleur de la fenêtre MEMBASIC, les deux suivantes agiront sur les couleurs des masques de votre application.

LIGNE FONCTIONS
Les quatres dernières couleurs concernent la 25ème ligne de l'écran (touches de fonction).
La première de celles-ci n'est pas modifiable, elle correspond à la couleur de la bordure de

l'écran (redéfinissable dans le TABLEAU de bord). La deuxième est celle du texte des touches de fonction. La troisième sert à l'affichage des messages d'erreurs en bas de l'écran (erreur d'imprimante ou erreur disque). La dernère représente la couleur de fond du texte des touches de fonction.

En fait, pour définir une couleur vous faites une combinaison de trois couleurs, le rouge R, le vert V et le bleu B.
L'échelle va de la lettre A à la lettre P, c'est à dire :
la lettre A étant la nuance la plus sombre et P, la nuance la plus claire. Par exemple, avec :

AAA on obtient du noir,
PPP on obtient du blanc,
AAP on obtient un bleu vif,
APA on obtient du vert vif,
PPA on obtient du jaune,
HHH on obtient du gris.

FORCE LE MODE COULEUR ? (O,N)
Il se peut que le TABLEAU de bord de votre APPLE II GS ait selectionné le mode NOIR et BLANC.
Si vous voulez néanmoins que les couleurs qui ont été définies précédemment soient prises en compte dès le chargement de MEMSOFT répondez OUI.

Validez par **POMME 1** ou en cliquant sur la touche **1 VALID.**
Pour finir, vous pouvez soit CREER une autre icône, soit revenir sous MEMBASIC, soit revenir sous PRODOS.

C'est tout. Regardez le bureau maintenant. Une nouvelle icône a été créée : COMPTA. C'est un "losange", icône cliquable que vous pouvez sélectionner, ouvrir et qui enchaînera successivement les programmes MEMSOFT et INCOMPTA.



# SECTION II
# MEMBASIC

## PLAN

# CHAPITRE 1
# GENERALITES

## 1.1 INTRODUCTION

Ce chapitre expose les caractéristiques générales de l'interpréteur MEMBASIC. Les commandes, fonctions et instructions sont présentées individuellement par ordre alphabétique dans le chapitre 2.

## 1.2 LES TOUCHES DU CLAVIER DANS L'EDITEUR

Voici le rôle particulier que jouent les touches du clavier dans l'éditeur MEMBASIC.

**"OPTION Clic-souris"** : fait passer MEMBASIC en mode fenêtre et permet de modifier les couleurs, les types de cadres etc ..

**flèches** : permettent le déplacement du curseur de saisie.

**POMME ↑** : Remonte le curseur de la hauteur d'une demi-fenêtre.

**POMME ↓** : Descend le curseur de la hauteur d'une demi-fenêtre.

**CTRL POMME ↑** : Remonte le curseur en haut du texte.

**CTRL POMME ↓** : Descend le curseur en bas du texte.

**"TAB →|"** : Déplace le curseur de huit caractères vers la droite.

**"SHIFT TAB →|"** : Déplace le curseur de huit caractères vers la gauche.

"POMME I" : En appuyant sur cette touche, MEMBASIC passe en mode insertion.
L'appui sur l'une des touches de direction, sur la touche "RETOUR CHARIOT" ou plus généralement sur toute touche ne correspondant pas à un caractère (sauf la touche DELETE) fait quitter le mode insertion.

"POMME DELETE" : Destruction du caractère sous le curseur.

"CTRL POMME DELETE" ou ⌦ : Destruction des caractères depuis le curseur jusqu'en fin de ligne.

"DELETE" : Destruction du caractère à gauche du curseur.

"POMME <-" : Ramène le curseur en début de ligne.

"CTRL POMME <-" : Amène le curseur en haut d'écran à la première frappe de cette touche, puis en haut de texte à la seconde frappe.

"CTRL -> " : va au prochain mot (à droite).

"CTRL <- " : va au mot précédent (à gauche).

"RETOUR CHARIOT" : valide une ligne MEMBASIC.

"POMME CTRL-C" : L'exécution d'un programme peut être interrompue en appuyant sur la touche Interruption (POMME CTRL C). Si le programme attend une entrée du clavier, il faut en outre appuyer sur la touche "RETOUR CHARIOT" pour revenir dans la fenêtre du programme. Le rôle de cette touche peut être inhibé au moyen de l'instruction BREAK OFF.

La ligne située au bas de l'écran indique la signification implicite des touches de fonction qui est :

| | | |
|---|---|---|
| F0 | AIDE | Accès aux fichiers d'aide MEMBASIC |
| F1 | RUN | Exécution du programme en mémoire |
| F2 | CLS | Efface l'écran d'édition |
| F3 | LIST < | Liste la page précédente du programme |
| F4 | LIST > | Liste la page suivante du programme |
| F5 | DESTR. | Destruction de la ligne d'écran |
| F6 | RECUP. | Récupération de la dernière ligne détruite |
| F7 | DUPLIC | Duplication de la ligne d'écran |
| F8 | INSERT | Insertion d'une ligne blanche |
| F9 | F1..F8 | Aide sur les touches clavier |



Ces fonctions peuvent être activées :

- soit en utilisant conjointement la touche POMME et la touche correspondante du pavé numérique,
- soit, plus simplement en cliquant avec la souris.

Chaque programme peut attribuer une nouvelle signification aux touches de fonction **POMME 1 à POMME 9** au moyen de l'instruction FKEY, la touche **POMME 0** restant affectée à la fonction AIDE. La liste ci-dessus donne la signification de ces touches en édition de programme.

## 1.3 NOTATIONS UTILISEES

Pour la description de la syntaxe, les règles suivantes sont utilisées :

- Les mots représentant des commandes, instructions, fonctions et, d'une manière générale, les mots-clés du langage sont écrits en majuscules.

  Exemple : 100 GOTO 1000

- Tous les signes de ponctuation contenus dans la syntaxe (virgule, point virgule, guillemets, deux-points, parenthèses, etc...) sont obligatoires à l'exception des caractères "inférieur à" et "supérieur à" (< et >) ou des crochets.

- Les expressions en minuscules, encadrées par les caractères < et > doivent être remplacées par les termes idoines choisis par l'utilisateur.

  Exemples :

  GOTO <étiquette>
  ABS (<expression-numérique>)

  Pour l'instruction GOTO, <étiquette> pourra, par exemple, être remplacé par 'ITER', ce qui donnera :

  GOTO 'ITER'

  Pour la fonction ABS, <expression-numérique> pourra être remplacé par une expression numérique; par exemple on pourra écrire :

  ABS (B*B-A)

- Les termes placés entre crochets sont facultatifs.

  Exemple : RESTORE [<numéro-de-ligne>]

  Cela signifie que l'on peut écrire

  RESTORE
  ou
  RESTORE <numéro-de-ligne>

- La notation suivante signifie que l'utilisateur doit choisir entre plusieurs options. Par exemple, l'utilisateur peut écrire :

  <étiquette>
  GOTO
  <numéro-de-ligne>

- Les termes entre crochets suivis de points de suspension peuvent être répétés plusieurs fois dans les limites de la longueur d'une ligne logique.

  Exemple :

  PRINT <expression> [ , <expression> ] ...

REMARQUES :

Dans un programme MEMBASIC, un nom de variable et un mot-clé qui se suivent doivent être séparés par un espace. Il en est de même pour un mot-clé précédant un nom de variable.

Par contre dans une expression, les opérateurs et les parenthèses jouent le rôle de séparateurs. Les espaces, autres que ceux figurant dans une constante chaîne, sont sans signification pour MEMBASIC.

Les espaces à l'intérieur d'un mot-clé ou d'un nom de variable, sont interdits et sont détectés par une erreur de syntaxe.

Par exemple P RINT n'est pas autorisé.



## 1.4 LIGNE LOGIQUE ET LIGNE PHYSIQUE

Un programme est composé de lignes logiques.

Une ligne logique ne peut pas excéder la longueur de 250 caractères. La longueur de la ligne physique (partie visible de la ligne logique) est fixée par la largeur de la fenêtre et donc choisie par l'utilisateur. (247)

Lors de l'introduction d'une ligne, lorsqu'on arrive à la fin de la ligne physique, si l'utilisateur continue à introduire des caractères, MEMBASIC décale l'ensemble du texte à condition de ne pas avoir atteint la limite de la ligne logique.

L'appui sur le caractère "RETOUR CHARIOT" marque la fin de la "ligne logique".

Dans ce qui suit, nous ne nous intéresserons qu'aux lignes logiques.

Une ligne logique doit obligatoirement commencer par un numéro de ligne qui est utilisé pour :

- Assurer le reclassement automatique des lignes si elles ne sont pas introduites dans l'ordre séquentiel.

- Atteindre cette ligne par branchement (GOTO, IF, etc...) bien que l'on puisse leur préférer les étiquettes.

## 1.5 VALIDATION D'UNE LIGNE

Lors de l'introduction d'une ligne au clavier, il est possible de corriger des erreurs de frappe ou des oublis en utilisant les touches de direction ou la souris. La ligne n'est prise en compte que lorsque l'utilisateur a appuyé sur la touche "RETOUR CHARIOT" ou - (parfois appelée "RETURN").

L'appui sur cette touche a, en fait, cinq fonctions :

- Validation de la totalité de la ligne logique introduite.

- Analyse syntaxique complète de la ligne logique.

- L'effacement et la réécriture de la ligne après suppression des espaces inutiles, mais en conservant le "colonnage" (appelé indentation) du début de la ligne et du point d'exclamation éventuel annonçant un commentaire.

- Cet affichage est effectué avec cadrage du numéro de ligne et après conversion des formes simplifiées de syntaxe et suppression des espaces inutiles (par exemple ? devient PRINT).

  Dans le cas particulier où le remplacement des abréviations par les mots-clés conduit à un allongement excessif de la ligne, le commentaire peut être décalé vers la droite.

- Passage à la ligne suivante.

Si une erreur de syntaxe est détectée, un message d'erreur est affiché dans une fenêtre annexe réservée à cet effet, visible uniquement dans ce cas. De plus la (les) partie(s) erronée(s) de la ligne est (sont) marquée(s) d'une couleur différente. Malgré l'erreur, la ligne est néanmoins mémorisée.

## 1.6 LES FENETRES UTILISEES

Lorsque MEMBASIC est chargé en mémoire, une seule fenêtre apparaît à l'écran : celle qui est utilisée pour l'introduction des commandes et des lignes de programme. Cette fenêtre est appelée fenêtre d'édition. MEMBASIC travaille cependant avec deux autres fenêtres : une fenêtre d'exécution qui n'apparaît que lors de l'exécution d'un programme qui affiche des données, et la fenêtre de trace qui n'est utilisée que si la trace d'un programme a été demandée.

A tout moment l'utilisateur peut déplacer les fenêtres ou modifier leurs tailles en utilisant la souris.



## 1.7 MODES DE FONCTIONNEMENT

Après chargement, l'interpréteur MEMBASIC attend une entrée au clavier. Il peut alors travailler selon trois modes différents :

- Le mode "immédiat" qui est très proche de celui d'une calculette. Il est également appelé mode "commande".

- Le mode "programme", mode dans lequel l'exécution des programmes est effectuée.

- Le mode "fenêtre" qui est utilisé pour déplacer des fenêtres ou pour changer leur taille.

### 1.7.1 Le Mode "Immédiat"

Dans ce mode, où aucune ligne ne doit commencer par un numéro de ligne, toute ligne introduite donne immédiatement lieu à l'analyse syntaxique et à son exécution.

Ce mode est utilisé en général :

- Soit pour introduire une commande,

- Soit pour demander des informations sur le contenu des variables d'un programme en cours de mise au point,

- Soit pour utiliser l'ordinateur comme une calculette,

- Soit pour utiliser les commandes éditeur de MEMBASIC.

Dans ce mode immédiat, certaines instructions ne peuvent pas être exécutées. MEMBASIC peut alors afficher un message d'erreur.

Exemple : DATA, SELECT ne peuvent pas être exécutées en mode immédiat.

Les commandes d'édition et de mise au point sont :

- AUTO ON/OFF numérotation automatique
- CONT reprise d'une exécution
- DEBUG ON/OFF passage en mode mise au point ou sortie de ce mode
- DELETE destruction de lignes

- LIST liste du programme en cours
- LOAD chargement d'un programme
- LOCNUMBER numérotation par adresses
- NEW effacement du programme en cours
- RENUMBER Renumérotation du programme
- RUN exécution du programme
- SAVE sauvegarde d'un programme
- TRACE ON/OFF demande ou arrêt du mode trace

Toutes ces commandes sont détaillées au chapitre 2.

### 1.7.2 Le Mode "Programme"

Toutes les lignes logiques commençant par un numéro de ligne de 1 à 64000, entrées en mode immédiat, ne sont pas exécutées, mais sont stockées en mémoire centrale. Elles ne sont exécutées par l'interpréteur MEMBASIC que si l'on frappe:

- Soit la commande RUN
- Soit GOTO <numéro de ligne>
  ou GOTO <étiquette>

Le mode de fonctionnement de MEMBASIC, lorsqu'il exécute ces lignes pré-enregistrées, s'appelle mode "Programme" ou mode "Exécution".

### 1.7.3 Le mode "Commande fenêtre"

Ce mode permet diverses manipulations sur les fenêtres. A tout moment, en appuyant sur la touche "OPTION CLIC-SOURIS", le système passe en mode "fenêtre" s'il était en mode commande ou mode immédiat. Un pictogramme représentant une main apparaît dans le coin supérieur gauche de la fenêtre active.

La section I "FENETRE" a été spécialement prévue pour expliquer ces manipulations de fenêtres.

On revient au mode précédent en appuyant à nouveau sur la touche "OPTION CLIC-SOURIS".

## 1.8 L'EDITEUR PLEIN ECRAN

Les lignes de programme peuvent être introduites en minuscules ou en majuscules, l'interpréteur transformant automatiquement les minuscules des mots-clés en majuscules et les majuscules des noms de variables en minuscules. Un espace est obligatoire avant et après chaque mot-clé du langage.



L'utilisateur dispose de toutes les touches décrites au chapitre 1.2 pour entrer ses lignes ou commandes.

La commande LIST permet d'afficher, dans l'ordre numérique croissant, les lignes du programme dans la fenêtre de travail dans les conditions suivantes :

- Les variables sont affichées en minuscules.
- Les mots-clés sont affichés en majuscules.
- Les constantes chaînes et les étiquettes sont affichées telles qu'elles ont été introduites.
- Les espaces d'indentation en début de ligne sont gardés.
- Les espaces inutiles à l'intérieur de la ligne sont supprimés.
- Les espaces précédant un commentaire commençant par ! sont gardés.
- Les espaces à l'intérieur des commentaires sont conservés.
- Les remarques, qu'elles soient introduites au moyen du mot-clé REM ou au moyen du point d'exclamation sont toujours affichées précédées du point d'exclamation.

REMARQUES :

Dans la pratique, pour les commentaires et pour l'indentation, MEMBASIC mémorise le numéro de colonne, ce qui est plus souple pour l'utilisateur.

Cette technique offre une excellente lisibilité du programme.

En utilisant la souris sur le bord inférieure de la fenêtre, on peut obtenir un défilement latéral du programme permettant de lire ou modifier des lignes plus larges que la fenêtre.

En outre, en utilisant les touches POMME I et POMME DELETE, l'utilisateur peut modifier une ligne, en

insérant ou supprimant des caractères. L'appui sur la touche "RETOUR CHARIOT" entraîne la prise en compte de la ligne logique ainsi modifiée.

## 1.9 L'ALPHABET UTILISE

Le jeu de caractères utilisé par MEMBASIC est le jeu ISO 646 version américaine ASCII étendu à un jeu 8 bits comportant 128 caractères nationaux ou semi-graphiques.

Cependant les mots-clés et les noms de variables utilisent un sous-ensemble du jeu ISO 646 : lettres, chiffres et le caractère "souligné".

Pour une meilleure lisibilité, toutes les variables sont affichées en minuscules et tous les mots-clés en majuscules.

## 1.10 FORME DES LIGNES

Une ligne logique doit avoir la forme générale suivante:

<numéro-de-ligne> <instruction> [ : <instruction> ] ...
ou
<numéro-de-ligne> <étiquette> [ : <instruction> ] ...

L'utilisateur peut introduire les lignes dans n'importe quel ordre, MEMBASIC se chargeant automatiquement de les replacer dans l'ordre croissant des numéros de lignes.

Une ligne peut contenir plusieurs instructions séparées par ":". Cependant, certaines instructions MEMBASIC utilisent le caractère ":" comme séparateur, il convient donc de ne pas confondre ces deux significations.
dans l'exemple :

```
100 INPUT PROMPT "Nom et prenom " : nom$ , prenom$
```

la ligne 100 contient une seule instruction.

## 1.11 LES OBJETS MANIPULES

MEMBASIC permet d'utiliser des variables de types :

- entier
- réel
- chaîne de caractères



Ces variables peuvent être de type "scalaire" (variables simples) ou de type tableau (variables indicées).

A chaque type "scalaire" de variable correspond un type de constante.

MEMBASIC accepte donc des constantes de types:

- entier
- réel
- chaîne de caractères

En outre MEMBASIC accepte un autre type d'objet : les étiquettes.

## 1.12 LES VARIABLES

Une variable est caractérisée par son nom, par son type et par sa valeur.

### 1.12.1 Noms de Variables

Un nom de variable peut comporter de 1 à 200 caractères dont le premier est une lettre, les caractères suivants étant au choix de l'utilisateur :

- des lettres,
- des chiffres,
- le caractère "souligné".

Si l'utilisateur introduit un nom de variable comportant des lettres majuscules, MEMBASIC les convertit automatiquement en minuscules.

EXEMPLE :

```
A
XMAX              sont des noms de variables
TAUX_DE_TVA

3AB               n'est pas un nom de variable
```

Une restriction importante toutefois : un nom de variable ne peut être l'un des mots réservés du langage. Voir l'annexe A qui en donne la liste.

### 1.12.2 Types de Variables

Le type d'une variable est indiqué par un suffixe :

le suffixe % indique que la variable est de type entier,
le suffixe $ indique que la variable est de type chaîne.

En l'absence de suffixe, les variables sont de type REEL.
Les variables chaînes se subdivise en deux types :

- les chaînes dites "de tailles fixes"

Ce sont des chaînes pour lesquelles une place fixe est attribuée en mémoire et dont la longueur ne peut dépasser une certaine valeur. La longueur de la chaîne (telle qu'elle est rendue par la fonction LEN) pourra alors évoluer entre 0 et la taille maximale attribuée. Ces chaînes nécessitent une déclaration explicite, indiquant la longueur maximum souhaitée, ce qui permet à MEMBASIC de leur allouer une place mémoire.

- les chaînes "de tailles variables"

C'est le type utilisé implicitement en l'absence de déclaration. Les chaînes peuvent avoir une longueur variant de 0 à 255 caractères, mais il n'y a pas pour autant attribution d'une place mémoire de 255 octets ! Au contraire, MEMSOFT pour APPLE II GS gère et optimise en permanence la place utilisée.

Quelle type de chaîne choisir ? Le raisonnement à appliquer est très simple: si vous connaissez la longueur maximum des chaînes que vous utilisez et si ce maximum peut être atteint ou approché pour l'ensemble des chaînes simultanément à un moment du déroulement du programme, il est alors préférable d'utiliser les chaînes de tailles fixes pour lesquelles l'optimisation du système sera meilleure.

Par exemple, si un important tableau de chaînes constitue l'utilisation principale des chaînes, si chaque élément du tableau ne peut dépasser 80 caractères et doit pouvoir les atteindre, utilisez sans hésiter les chaînes de tailles fixes.

Les chaînes de tailles fixes ont également pour elles un dernier avantage : elles seules peuvent être passées de programme en programme grâce à l'instruction CHAIN.



Consultez cette instruction au chapitre 2 pour plus d'informations.

### 1.12.3 Tableaux

Un tableau est un ensemble d'éléments (variables élémentaires) de même type, référencés au moyen d'un nom unique. Chaque élément de tableau est désigné par son nom suivi entre parenthèses par un ou plusieurs indices.

MEMBASIC accepte des tableaux ayant 1, 2, 3 ou 4 indices.

Il faut indiquer pour un élément de tableau autant d'indices que le tableau en contient dans l'instruction de déclaration :

EXEMPLE :

```
100 DIM n$(10), a(10,20), b(3,5,3)
.
.
... n$(I)
.
.
... a(I,J)
.
.
... b(M,N,P)
```

La valeur maximale que peut prendre un indice est limitée sur le plan de la syntaxe à 32767, et est limitée par la place disponible en mémoire.

La valeur implicite maximale des indices est 10. Toutefois, cette valeur implicite peut être modifiée au moyen de l'instruction OPTION DIM. Cette valeur implicite ne peut être utilisée que pour les tableaux à une seule dimension, qui seuls pourront se passer de déclaration explicite.

La valeur implicite minimale des indices est 1. Cependant l'instruction OPTION BASE permet de changer cette valeur implicite.
La valeur minimale d'un indice peut être négative.

En outre l'utilisateur peut spécifier la valeur minimale et la valeur maximale que peut prendre chaque indice au moyen de l'instruction DIM.

EXEMPLE :

```
100 OPTION DIM 30
110 DIM annee(1983 TO 1987),ca(1983 TO 1987)
...
200 FOR i=1 TO 30
210   v(i)=i
220   NEXT i
```

Le dimensionnement implicite a été utilisé pour le tableau V. Par contre les tableaux ANNEE et CA ont été dimensionnés explicitement.

### 1.12.4 Indices

Si l'évaluation d'un indice donne une valeur non entière, celle-ci est convertie à l'entier le plus proche par arrondi.

Par exemple:

```
A(3.9)  est équivalent à A(4)
A(3.5)  est équivalent à A(4)
A(3.49) est équivalent à A(3)
A(3.1)  est équivalent à A(3)
```

Un indice peut être du type entier ou réel mais pas du type chaîne.
A l'exécution, si un indice prend une valeur en dehors de la plage utilisée, l'erreur N° 2001 est détectée.

CONSEIL

L'utilisation d'indices entiers améliore la vitesse d'exécution.

### 1.12.5 Conversions

MEMBASIC accepte des expressions "mixtes", c'est à dire faisant intervenir des variables et constantes de type réel et entier. Dans ce cas il effectue automatiquement les conversions nécessaires avant d'effectuer les opérations.



### 1.12.6 Index

Dans ce qui suit, nous appellerons INDEX une expression numérique dont la valeur est convertie en entier par arrondi avant d'être utilisée comme paramètre d'une fonction ou d'une instruction.

### 1.12.7 Représentation des Variables

Les variables entières sont stockées sur 2 octets et leur plage de variation va de -32768 à +32767.

Les variables réelles sont représentées sous forme décimale flottante permettant une précision de 14 chiffres significatifs. Pour cela 8 octets sont utilisés pour chaque valeur réelle.

```
Premier octet : 1 bit    0 si nombre positif
                         1 si nombre négatif

                7 bits   exposant (en binaire)
                         0 pour une valeur nulle.
                         64 pour un nombre compris
                         entre 1 et 9.

Octets 2 à 8  : Représentation DCB de la mantisse.
                (DCB signifie Décimal Codé en Binaire)
```

## 1.13 EXPRESSIONS ET OPERATEURS

MEMBASIC accepte 4 catégories d'opérateurs :

- Opérateurs arithmétiques (portant sur des nombres)
- Opérateurs de relations
- Opérateurs logiques
- Opérateurs sur chaînes

Ces opérateurs sont utilisés, en liaison avec les parenthèses, les fonctions, les variables et constantes, pour la constitution d'expressions.

Ces opérateurs ont un ordre de priorité, mais à priorité égale, les opérations sont effectuées dans l'ordre de gauche à droite. L'utilisateur peut imposer un ordre différent en introduisant des parenthèses.

### 1.13.1 Opérateurs et Expressions Numériques

MEMBASIC accepte les opérateurs numériques suivants par ordre de priorité décroissante :

| | |
|---|---|
| ^ | élévation à la puissance |
| * et / | multiplication et division |
| + et - | addition et soustraction |

L'opérateur "Modulo" n'existe pas mais la fonction MOD permet d'accomplir la même fonctionnalité.

Les fonctions sont prioritaires par rapport à l'opérateur puissance.

Des parenthèses permettent à l'utilisateur d'imposer l'ordre d'exécution des opérations.

EXEMPLE :

```
Y = A * (B + C)
```

L'addition B + C est effectuée avant la multiplication par A.

ERREURS :

L'évaluation d'expressions numériques peut entraîner un dépassement de capacité. Dans ce cas l'erreur No 1002 est détectée.

L'erreur No 3001 est détectée en cas de tentative de division par zéro.

### 1.13.2 Opérations sur Chaînes de Caractères

Le seul opérateur portant sur des chaînes de caractères est l'opérateur de concaténation qui est noté au choix de l'utilisateur & ou +.

La concaténation de deux chaînes se fait en mettant les chaînes bout à bout afin d'en constituer une troisième. Dans cette opération aucun "espace" supplémentaire n'est ajouté.

Par exemple :

```
"JEAN" & "PAUL" donne "JEANPAUL"
```



Pour obtenir une chaîne comportant un espace, on devra donc écrire :

```
"JEAN " & "PAUL
```

ou

```
"JEAN" & " " & "PAUL
```

ERREURS :

La concaténation de deux chaînes peut tenter de produire une chaîne dont la longueur excède 255. Dans ce cas l'erreur 1051 est détectée.

SOUS-CHAINES :

Il est également possible d'extraire des sous-chaînes, à partir d'une chaîne existante. Deux méthodes sont disponibles :

- L'utilisation des fonctions LEFT$, MID$, RIGHT$ (voir ces fonctions au Chapitre 2).
- L'utilisation de la notation sous-chaîne.

Si a$ est une chaîne simple, a$(3:6) représente la sous-chaîne extraite de A$ en prenant les caractères de rang 3 à 6.

Exemple :

```
100 a$ = "LES GIBOULEES DE MARS"
110 PRINT a$(5:13)
RUN
GIBOULEES
```

Pour construire des expressions chaînes, on dispose donc

- de l'opérateur concaténation & ou +,
- des constantes et variables chaînes,
- des fonctions sur chaînes décrites au Chapitre 2.

Les fonctions standard qui donnent un résultat de type chaîne ont un nom se terminant par $.

EXEMPLE :

```
CHR$, LCASE$, UCASE$, STR$, etc...
```

### 1.13.3 Comparaison de Chaînes de Caractères

MEMBASIC compare des chaînes selon la méthode suivante :

La comparaison entre deux chaînes se fait caractère par caractère à partir du début de chaque chaîne. Un caractère est dit plus petit qu'un autre si son code est plus petit que celui de l'autre caractère.

Dès que l'on trouve dans l'une des chaînes un caractère plus petit que le caractère correspondant de l'autre, la première chaîne est déclarée comme étant la plus petite.

Si la fin d'une chaîne est atteinte avant qu'une différence ne soit apparue, alors la chaîne la plus courte est considérée comme la plus petite.

Dans cette comparaison, tous les caractères sont pris en compte y compris les espaces éventuels.

EXEMPLE :

```
     A codé 65 est plus petit que B codé 66
mais
     Z codé 90 est plus petit que a codé 97
```

### 1.13.4 Opérateurs de Relations

Les opérateurs de relation sont utilisés pour comparer des valeurs de même type; le résultat d'une comparaison est une valeur "booléenne" VRAI ou FAUX qui peut être utilisée pour des instructions de branchement.

Le tableau suivant donne la liste de ces opérateurs et leur signification :

| Relation testée | Opérateur |
|---|---|
| égalité | = |
| inférieur à | < |
| supérieur à | > |
| différent de | <> |
| inférieur ou égal | <= |
| supérieur ou égal | >= |

Ces opérateurs peuvent porter sur des opérandes de type numérique ou de type chaîne.



EXEMPLE :

```
IF taux < 10 THEN ...
IF a$=b$ THEN ...
```

REMARQUE :

Faire attention au fait que le caractère = est utilisé à la fois comme opérateur de relation et comme symbole d'affectation :

```
     IF taux=1 THEN tva=18.6
           |           |
  Comparaison        Affectation
```

### 1.13.5 Opérateurs Logiques

MEMBASIC dispose d'opérateurs logiques qui permettent de construire des comparaisons sur plusieurs relations. Ces opérateurs sont AND, NOT et OR.
Leur signification est donnée par le tableau suivant :

| 1r opérande | 2e opérande | X AND Y | X OR Y |
|---|---|---|---|
| faux | faux | faux | faux |
| faux | vrai | faux | vrai |
| vrai | faux | faux | vrai |
| vrai | vrai | vrai | vrai |

Pour l'opérateur unaire NOT, la signification est :

```
NOT "vrai" donne "faux"
NOT "faux" donne "vrai"
```

## 1.14 ETIQUETTE

Une étiquette est représentée dans un programme sous forme d'une constante caractère entre apostrophes.

Une telle étiquette ne peut être placée qu'en début de ligne.

EXEMPLE :

```
200 'ITER'
```

Une ligne contenant une étiquette peut être éventuellement suivie d'une instruction (séparée alors par le caractère ":").

EXEMPLE :

```
200 'ITER' : IF x<10 GOTO 30
```

Une étiquette peut être utilisée dans des instructions de branchement comme GOTO, ON...GOTO, etc...

**Génération automatique d'étiquettes :**

Il est possible de remplacer automatiquement les branchements à des numéros de lignes, par des branchements à des étiquettes dans un programme MEMBASIC.
Il suffit pour cela d'enfoncer **OPTION POMME 3** : une étiquette est ajoutée au début de chaque ligne dont le numéro figure dans une instruction de branchement, et le branchement est modifié pour utiliser l'étiquette.
Le libellé de l'étiquette est en fait le numéro de la ligne.

Cette modification présente deux intérêts :

- permettre au programmeur de garder les mêmes repères dans son programme malgré l'utilisation de RENUMBER,
- éviter de supprimer une ligne qui est appelée par une instruction de branchement.

Il est également possible de modifier les branchements à des numéros de lignes commençant par des étiquettes, par des branchements à ces étiquettes. Pour cela, enfoncez OPTION POMME 3.

## 1.15 SOUS-PROGRAMMES

L'instruction GOSUB permet de dérouter le programme vers un "sous-programme". Le sous-programme est désigné par le numéro de la ligne où il débute ou par une étiquette.

L'instruction RETURN du sous-programme ramène à l'instruction suivant le GOSUB dans le programme principal.



Un sous-programme appelé par GOSUB peut lui même appeler un autre sous-programme.

L'instruction ON ... GOSUB permet un appel à un sous-programme parmi une liste suivant la valeur d'un index.

## 1.16 DETECTION ET GESTION DES ERREURS

Si le programme est écrit sans "dispositif de récupération des erreurs", toute erreur détectée par MEMBASIC donne lieu à un message d'erreur et en général à l'arrêt du programme.

Cependant, il est possible de construire un "dispositif de récupération des erreurs" en utilisant diverses instructions spécifiques dont l'instruction WHEN EXCEPTION.

Cette instruction est utilisée sous la forme :

```
                          <numéro de ligne>
WHEN EXCEPTION GOTO
                          <étiquette>
```

Si une erreur survient, le programme est dérouté sur une séquence de lignes destinées à "traiter" l'erreur détectée.

Elle dispose pour cela des fonctions EXLINE et EXTYPE et de l'instruction RETRY :

- EXLINE contient le numéro de la ligne qui a provoqué l'erreur ( sauf dans le cas d'un programme protégé )

- EXTYPE contient le numéro de l'erreur

- L'instruction RETRY provoque le branchement à l'instruction qui a provoqué l'erreur.

## 1.17 COMMANDES PRODOS

Un programme écrit en MEMBASIC peut exécuter un certain nombre de commandes agissant sur son environnement PRODOS.

Les instructions et fonctions qui sont réservées à cet effet ont donc un fonctionnement lié à celui de PRODOS et peuvent différer selon les versions du produit MEMSOFT, sur d'autres systèmes d'exploitation.

### 1.17.1 Gestion des disques et des répertoires

PRODOS gère plusieurs disques, chaque disque étant organisé en structure arborescente.

MEMBASIC propose un certain nombre de commandes permettant la gestion des disques sous PRODOS. On peut changer de répertoire, en créer un nouveau, demander le répertoire en cours, définir une liste de noms d'accès vers des répertoires. DIR donne la liste des fichiers d'un répertoire.

Instructions :

- CHDIR — Changement de répertoire
- CURDRIVE — Changement de disque implicite
- DIR — Liste d'un répertoire
- MKDIR — Création de répertoire
- PATH — Définition de chemins implicites
- RMDIR — Suppression de répertoire

Fonctions :

- CHDIR$ — Donne le répertoire en cours
- CURDRIVE$ — Donne le disque en cours
- DISKSIZE — Donne la taille du disque
- FILESIZE — Taille des fichiers
- FREE — Donne la place libre sur le disque
- PATH$ — Donne les chemins implicites

Toutes ces instructions et fonctions sont présentées au chapitre 2.

### 1.17.2 Les fichiers PRODOS

MEMBASIC propose un ensemble complet d'instructions spécifiques permettant de gérer les fichiers PRODOS. Certaines seront utiles pour effectuer quelques travaux de gestion du disque sans quitter MEMSOFT pour APPLE II GS (comme RENAME, COPY, KILL), d'autres serviront à interfacer vos logiciels MEMSOFT avec d'autres logiciels PRODOS.

Instructions générales :

- COPY — Copie de fichier
- KILL — Destruction d'un fichier
- RENAME — Changement de nom de fichier



Disque dur

- MEMBACKUP — Sauvegarde de fichiers
- MEMCOMPARE — Vérification des sauvegardes
- MEMRESTORE — Restauration des fichiers

Lecture et Ecriture

- OPEN — Ouverture d'un fichier
- POINTER # — Choix d'enregistrement
- INPUT #
- LINE INPUT # — Lecture dans un fichier
- INPUT$
- WRITE #
- PRINT # — Ecriture dans un fichier
- CLOSE — Fermeture d'un fichier

Toutes ces instructions et fonctions sont présentées au chapitre 2, voici néanmoins une synthèse des possibilités d'utilisation :

L'instruction OPEN définit le type d'accès au fichier :

"I": Le fichier est ouvert en lecture seule et en séquentiel. Cette méthode est particulièrement bien adaptée à la lecture des fichiers textes. On pourra lire les données grâce aux instructions INPUT #, LINE INPUT #, INPUT$.

"A": Le fichier est ouvert en mode "ajouts". On désire compléter un fichier déjà existant sans modifier les informations qui sont présentes. On pourra écrire dans le fichier grâce aux instructions PRINT # ou WRITE #.

"O": Le fichier est créé. Si le fichier existait déjà, il est détruit. On pourra écrire dans le fichier grâce aux instructions PRINT # ou WRITE #.

"R": Le fichier est un fichier structuré avec des enregistrements de taille fixe. La taille de chaque enregistrement peut être définie dans l'instruction d'ouverture, sinon, la valeur implicite 128 est utilisée. L'instruction POINTER # permet de choisir un enregistrement

qui pourra être lu grâce aux instructions INPUT #, LINE INPUT #, INPUT$ ou écrit grâce aux instructions PRINT # ou WRITE #.

Les instructions de lecture des données dans le fichier:

INPUT #

Cette instruction lit des données dans le fichier et les traite comme le fait l'instruction INPUT, hormis l'élimination des espaces à gauche. Les différentes données d'un même INPUT # sont séparées par ",". Une "fin de ligne" termine la liste.

LINE INPUT #

Pour permettre de lire des données contenant des ",", LINE INPUT # lit dans une seule chaîne de caractères la donnée jusqu'au premier "fin de ligne".

INPUT$

Pour les cas difficiles ... INPUT$ permet de lire le nombre exact de caractères désiré, quels qu'ils soient.

Les instructions d'écriture des données dans le fichier:

PRINT #

Cette instruction écrit les données dans le fichier exactement dans le même format que celui utilisé par PRINT. Par exemple, le séparateur de données "," produira uniquement une suite d'espaces permettant le "colonnage" de la donnée.

WRITE #

Pour avoir effectivement la notion de champs, il est préférable d'utiliser WRITE # qui inscrit réellement les "," séparatrices dans le fichier et met les chaînes entre guillemets. Ce format est exactement celui reconnu par INPUT #.



Les fichiers doivent être fermés en fin d'utilisation grâce à l'instruction CLOSE. Cela permet de sauvegarder sur le disque les informations modifiées dans le cas d'écritures. La fermeture d'un fichier permet également d'en ouvrir un nouveau. En effet, le nombre maximum de fichiers PRODOS ouverts simultanément ne peut dépasser 10.

### 1.17.3 Les communications "série"

Il est essentiel à tout système de pouvoir communiquer avec l'extérieur. La méthode la plus répandue est la communication série RS232 C.

Pour cela, le port de communication est ouvert comme le serait un fichier grâce à une forme spéciale de l'instruction OPEN (qui est OPEN "COM..."). Les données sont ensuite reçues ou émises avec les mêmes instructions (décrites en 1.17.2) que dans le cas des fichiers: INPUT #, LINE INPUT #, INPUT$, PRINT #,WRITE #.

Les problèmes rencontrés généralement lors de transmissions série sont essentiellement des problèmes de synchronisation. En effet, l'émetteur de données est souvent "aveugle" et lorsque le récepteur désire que la transmission soit temporairement suspendue (par exemple, pour sauvegarder sur disque les données reçues), il ne sait pas toujours comment faire.

Il est conseillé pour résoudre totalement ce problème d'utiliser l'un des protocoles prévus par le TABLEAU de BORD de votre APPLE II GS.

## 1.18 SUFFIXES IMPLICITES UTILISES PAR MEMSOFT

MEMSOFT pour APPLE II GS utilise des suffixes implicites qui sont les suivants :

| | |
|---|---|
| .AUT | Séquences claviers enregistrées |
| .HLP | Fichiers d'aide |
| .MFK | Clés de fichiers MEMFILE |
| .MFR | Articles (ou enregistrements) MEMFILE |
| .MSK | Masques MEMSCREEN |
| .PRG | Programmes MEMBASIC |

La signification des fichiers ayant les suffixes .MSK est expliquée dans le manuel MEMSCREEN.

La signification des fichiers .MFK et .MFR est donnée dans le manuel MEMFILE.

## 1.19 SAUVEGARDE ET CHARGEMENT D'UN PROGRAMME

Les programmes peuvent être sauvegardés selon trois modes différents :

- Mode "Normal" ou "précompilé"
- Mode "fichier fleuve" utilisable avec n'importe quel éditeur
- Mode protégé dans lequel le programme est stocké sous forme comprimée exécutable. Un programme stocké sous cette forme ne peut plus être ni modifié ni listé.

### 1.19.1 Sauvegarde d'un programme

Pour sauvegarder un programme sous forme d'un fichier fleuve, il faut utiliser une forme particulière de l'instruction LIST.

La sauvegarde sous forme normale ou sous forme protégée s'obtient au moyen de la commande SAVE.

Dans tous les cas, on ne pourra sauvegarder un programme en le substituant à une version précédente portant le même nom que si on le précise explicitement dans la commande en faisant précéder l'expression donnant le nom du fichier par le caractère "@".

Exemples : Pour sauvegarder le programme présent en mémoire centrale sous le nom ESSAI.PRG, dans le répertoire courant, on écrira :

```
SAVE "ESSAI"        sauvegarde sous forme normale
SAVE "$ESSAI"       sauvegarde sous forme protégée
LIST "ESSAI"        sauvegarde sous forme texte
```

Pour une sauvegarde ultérieure après modification, on écrira respectivement :

```
SAVE "@ESSAI"
SAVE "@$ESSAI"
LIST "@ESSAI"
```



### 1.19.2 Chargement d'un programme et lancement de l'exécution

Les commandes LOAD, RUN et CHAIN permettent de charger un programme en vue de son exécution.

Alors que LOAD ET RUN, habituellement utilisés comme commandes, peuvent également être utilisés comme instructions, CHAIN, dont le seul rôle est le passage de programme à programme, n'est utilisé que comme instruction.

L' exécution de LOAD, RUN ou CHAIN provoque d'abord la fermeture des fichiers et masques qui étaient éventuellement ouverts, puis le chargement du programme demandé.

LOAD n'entraîne pas le démarrage automatique de son exécution.

RUN entraîne le démarrage automatique de l'exécution du programme chargé.

CHAIN entraîne le démarrage de l'exécution et permet de passer des données au nouveau programme. L'instruction PROGRAM établit (dans le nouveau programme) la correspondance entre les données passées et les variables.

Ces instructions tiennent compte des chemins indiqués dans la commande PATH pour chercher et charger le programme demandé.

Voici un exemple illustrant l'emploi de LOAD, RUN et CHAIN:

```
LOAD "test"
  Charge le programme "test.prg" après fermeture des
  fichiers éventuellement ouverts. Il faudra exécuter
  RUN en mode immédiat pour en lancer l'exécution.

RUN "test"
  Est équivalent à l'enchaînement LOAD "test" puis
  RUN.

CHAIN "test" WITH (chaine$,5)
  Est équivalent au RUN décrit ci-dessus, mais les
  paramètres chaine$ et 5 sont passés au programme
  "test.prg" si celui-ci débute par une instruction
  PROGRAM telle que :
  100 PROGRAM test (nouvelle_chaine$, numerique)
```

Le tableau suivant résume les principales caractéristiques de ces instructions et commandes dont le détail est donné dans le chapitre 2.

| | Ferme les fichiers | Démarre l' exécution | Mode d' utilisation | Passage de Paramètres |
|---|---|---|---|---|
| LOAD | OUI | NON | Commande | NON |
| RUN | OUI | OUI | Comm/Inst | NON |
| CHAIN | OUI | OUI | Instruc. | OUI |

## 1.20 FICHIERS D'AIDE

Plusieurs fichiers d'aide sont disponibles avec le système MEMSOFT pour APPLE II GS :

- Des fichiers standard qui contiennent des informations sur la syntaxe des instructions utilisées dans MEMBASIC, MEMFILE et MEMSCREEN.

- Des fichiers d'aide que le développeur peut facilement construire pour aider l'utilisateur de programmes d'application fonctionnant sous MEMSOFT.

### 1.20.1 Fichiers d'Aide Standard

En cas d'hésitation sur la façon d'employer certaines fonctionnalités de MEMBASIC, par exemple les fenêtres, l'éditeur, certaines commandes ou instructions, consulter ces fichiers selon le procédé suivant :

- Cliquez sur la touche de fonction **AIDE** Une fenêtre contenant le menu des différentes aides possibles apparaît à l'écran.

- En cliquant sur lès flêches situées sur la dernière ligne de l'écran les différentes pages du fichier d'aide défilent dans la fenêtre, dans le sens choisi. La touche "POMME <-" reprend l'aide depuis le début.

A tout moment, en cliquant sur "ESC" sur la dernière touche de l'écran, l'utilisateur quitte le fichier



d'AIDE pour revenir au programme en cours de création ou d'exécution.

La fin des fichiers d'AIDE contient souvent un menu qui permet d'appeler un autre fichier d'AIDE.

### 1.20.2 Fichiers d'Aide "Application"

Dans un programme en cours d'exécution, les fichiers d'AIDE appelés au moyen de la touche **F0 (POMME 0)** sont en priorité les fichiers qui ont le même nom que celui du masque MEMSCREEN utilisé, puis celui portant le nom du programme en cours et enfin le fichier d'Aide FRSOS.HLP

Les fichiers d'AIDE ont toujours le suffixe .HLP qui permet de les différencier des autres fichiers.

Dans la pratique, les fichiers d'AIDE sont des masques MEMSCREEN. Le détail de leur construction est donc expliqué dans le manuel MEMSCREEN.

## 1.21 SEQUENCES CLAVIER PRE-ENREGISTREES

Les séquences clavier pré-enregistrées permettent de mémoriser puis de rejouer des séquences de touches tapées au clavier. Ces séquences sont sauvegardées sur disque avec un suffixe .AUT. Comme un piano mécanique rejoue ses bandes perforées, les séquences enregistrées pourront être utilisées à chaque fois qu'une opération est répétitive. On les retrouvera pour répéter les procédures d'accès à un logiciel particulier, pour 'dessiner' des cadres ou autres figures semi-graphiques en création de masque, pour réaliser très rapidement des démonstrations de logiciels etc...

L'enregistrement et la relecture de séquences claviers enregistrées se font à n'importe quel moment, quel que soit le programme et ne demandent aucune programmation. C'est donc une fonctionnalité ajoutée à tout logiciel sans aucun effort.

### 1.21.1 Enregistrement d'une séquence

La procédure à suivre pour enregistrer une séquence est la suivante :

- Enfoncer **CTRL POMME 3** pour lancer l'enregistrement. Une fenêtre apparaît alors qui demande le nom du fichier destiné à recevoir la séquence.

- Entrer le nom du fichier SANS SUFFIXE puis enfoncer **"RETOUR CHARIOT"** pour confirmer le nom choisi ou **"ESC"** pour abandonner.

  Le nom devra être précedé du caractère "@" si le fichier existe déjà. Le nom du fichier pourra contenir des spécifications de disque et de répertoire.

  A partir du moment où le nom du fichier à été confirmé puis accepté, les touches tapées ensuite sont enregistrées.

- L'enregistrement s'arrête avec l'enfoncement des touches **CTRL POMME 4** qui ferment le fichier. Le fichier créé a pour suffixe .AUT.

  Au cours de l'enregistrement, il est possible de prévoir des 'pauses' qui, lorsque la séquence sera rejouée, auront une durée de 1 seconde chacune. Une pause se programme en enfonçant **CTRL POMME 6**. A chaque appui de ces touches correspondra une seconde d'attente à la relecture. Cette possibilité est particulièrement intéressante pour la préparation de démonstrations.

### 1.21.2 Répétition d'une séquence

Rejouer une séquence est encore plus simple :

- Enfoncer **CTRL POMME 1** pour lancer la relecture. Une fenêtre apparaît alors qui demande le nom de la séquence à jouer.

- Entrer le nom du fichier SANS SUFFIXE puis enfoncer **"RETOUR CHARIOT"** pour confirmer le nom choisi ou **"ESC"** pour abandonner.

  Le nom du fichier pourra contenir des spécifications de disque et de répertoire. S'il n'en contient pas et si le fichier n'est pas trouvé dans le répertoire en cours, la recherche se poursuivra dans les répertoires enregistrés par l'instruction **PATH**.

  A partir du moment où le nom du fichier a été confirmé puis accepté, les touches enregistrées sont rejouées. En fin de fichier, le clavier redevient la source d'arrivée de caractères.

  **"CTRL POMME 2"** permet à tout moment d'arrêter la séquence.

  L'instruction REPLAY permet de rejouer des séquences enregistrées sans intervention manuelle. Cette instruction est décrite au chapitre 2.

### 1.21.3 L'utilisation de la touche OPTION

La touche **OPTION** (ou **ALT**) joue un rôle particulier. Associée à une lettre du clavier alphabétique, elle provoque la relecture automatique de la séquence de nom ALT<lettre>.

C'est donc un moyen encore plus rapide pour rejouer jusqu'à 26 séquences particulières de noms :

ALTA.AUT à ALTZ.AUT.

Ces séquences peuvent être créées comme indiqué précédemment au chapitre 1.21.1.

### 1.21.4 Familles de séquences enregistrées

En maintenant enfoncée la touche **OPTION** puis en enfonçant une lettre de l'alphabet, ou rejoue directement une séquence enregistrée de nom prédéfini. Ces séquences appartiennent à une famille de 26 fichiers (un par lettre de l'alphabet). Il est possible de changer la famille utilisée :

taper **CTRL POMME 1**
préciser **#x** comme nom de fichier
taper **RETOUR CHARIOT**
taper **ESC**

x étant une des 26 lettres.

Ce seront les séquences de nom **xLTA.AUT à xLTZ.AUT** qui seront exécutées. Par défaut x vaut A (voir chapitre 1.12.3 ).

### 1.21.5 Enregistrement d'une démonstration de logiciel

Les fonctionnalités qui vont être présentées permettent de réaliser simplement une véritable démonstration d'un logiciel:

Enregistrement d'une démonstration :

- Séquences sans fin

  Il est possible de réaliser des séquences qui recommencent au début dès qu'elles se terminent. Pour cela, finir l'enregistrement de la façon suivante :

  - taper **CTRL POMME 1**
    la fenêtre ENREGISTREMENT apparaît

  - taper **CTRL POMME 3**
    (l'enregistrement est arrêté par cette commande).

- Arrêts temporaires enregistrés :

  Il est possible de prévoir des points d'arrêts dans les démonstrations. Ces points d'arrêts permettront éventuellement d'introduire des données particulières lors de la relecture.

  Pendant l'enregistrement, taper **CTRL POMME 2** là où un arrêt est souhaité.

  Lors de la relecture, la séquence enregistrée s'interrompra jusqu'à enfoncement de **CTRL POMME 3**.

  Tous les caractères saisis avant **CTRL POMME 3** sont transmis à l'application. Cet état du clavier est caractérisé par un indicateur "a" dans le coin inférieur droit de l'écran.

  Pendant l'enregistrement, pour permettre également d'entrer des données particulières, l'enregistrement est suspendu jusqu'à enfoncement de **CTRL POMME 3**.



  L'indicateur "a" en bas à droite permet de savoir que l'enregistrement est suspendu.

- Demande de saisies libres finies par **"RETOUR CHARIOT"**

  **CTRL POMME 7** suivi de **"RETOUR CHARIOT"** fonctionne comme **CTRL POMME 2** suivi de **CTRL POMME 3**. L'indicateur est alors "s" en cours de réexecution et "S" en cours de saisie.

  La seule différence entre les deux techniques est que **"RETOUR CHARIOT"** est effectivement envoyé au logiciel, alors que les **CTRL** ne le sont jamais.

  Cette méthode est préférable dans le cas fréquent où le **"RETOUR CHARIOT"** représente la validation de la saisie et doit être lié à la reprise de l'enregistrement ou de la réexécution d'une séquence.

- Temporisation

  Un **CTRL POMME 6** enregistré provoquera une pause de une seconde. Vous pouvez ainsi permettre au spectateur de constater les étapes importantes du traitement ou de lire les commentaires. L'indicateur en bas de l'écran devient "p".

- Visualisation de commentaires

  **CTRL POMME 8** fait apparaître une fenêtre de commentaires où il est possible d'enregistrer des explications. Ne pas oublier de prévoir le temps de lecture de ces commentaires.

  **CTRL POMME 8** permet de supprimer la fenêtre de commentaires et de retourner à l'enregistrement de la démonstration et l'exécution du programme.

Pendant qu'une démonstration est rejouée :

- Suspension temporaire de la séquence rejouée

  **CTRL POMME 6** permet de suspendre une réexécution de séquence à tout moment (par exemple pour répondre à une question ...). L'indicateur en bas de l'ecran devient t.
  **CTRL POMME 6** une deuxième fois permet de reprendre ensuite l'exécution.

- Imbrications de séquences et retour à l'appelant

Une séquence enregistrée peut appeler une autre séquence enregistrée (par CTRL POMME 1 ou par OPTION <lettre>) et ce jusqu'a huit appels imbriqués . Par exemple des séquences répétitives peuvent annimer l'écran au cours d'une explication orale prévue lors de l'enregistrement.

CTRL POMME 4 permet de finir une séquence et de continuer la séquence appelante.

### 1.21.6 Edition des séquences enregistrées

Il peut être agréable de pouvoir compléter ou corriger des séquences enregistrées. Les deux sont possibles :

- Il est possible de rallonger une séquence en spécifiant à l'enregistrement le nom du fichier précédé de "&".

- Les modifications de séquences enregistrées pourront être faites grâce à un éditeur de texte. Il suffit de savoir que les codes spéciaux sont codés sous la forme : caractère de code 254 suivi d'un caractère représentatif. De même les caractères "Line Feed" (de code 10) sont filtrés à la réexécution ce qui permet de récupérer des fichiers texte sans transformations.

### 1.21.7 Transformation de fichiers texte en masques

Pour récupérer des textes existant dans un masque, il suffit de rejouer le fichier pendant la création ou la mise à jour du masque :

Par exemple, pour charger le texte contenu dans le fichier TEXTE.TXT dans le masque TEXTE.MSK :

```
exécuter : LET"#new,masque,TEXTE"
           le masque apparaît
taper    : CTRL POMME 1
           la fenêtre de réexécution apparaît
entrer   : texte.txt
```

le texte est inséré comme s'il était tapé au clavier.



# CHAPITRE 2
# COMMANDES, FONCTIONS ET INSTRUCTIONS

Le Chapitre 2 décrit l'ensemble des commandes, fonctions et instructions disponibles avec l'interpréteur MEMBASIC.

L'ordre utilisé est l'ordre alphabétique car il permet de retrouver très rapidement une information recherchée.

Pour chaque commande, fonction, ou instruction le même plan est utilisé. Ce plan est donné page suivante.

Chaque commande, fonction, ou instruction nouvelle commence à une nouvelle page pour faciliter vos recherches.

Pour le cas particulier de l'instruction LET, consultez également les sections III "MEMSCREEN" et IV "MEMFILE".

NOM et TYPE du MOT CLE

BUT

Il s'agit d'une brève description du but de ce mot-clé

SYNTAXE

Ce paragraphe décrit la syntaxe de la commande, fonction ou instruction de façon exhaustive. La syntaxe est décrite selon la méthode exposée dans le Chapitre 1.

EXPLICATION

Ce paragraphe décrit en détail le fonctionnement de cette commande, fonction ou instruction.

EXEMPLE

Ce paragraphe contient une portion de programme utilisant l'item faisant l'objet du paragraphe et montrant donc sa mise en oeuvre.

ERREURS

Il s'agit d'un petit paragraphe donnant la liste des erreurs qui peuvent être détectées par l'interpréteur MEMBASIC lors de l'exécution du programme.

CONSEILS

Paragraphe non-systématique donnant des conseils relatifs à l'utilisation de l'item décrit.

REMARQUE

Paragraphe également non-systématique permettant des rapprochements entre des commandes, fonctions ou instructions légèrement différentes.



# ABS FONCTION

BUT

Cette fonction renvoie la valeur absolue du paramètre qui doit être de type numérique (entier ou réel).

SYNTAXE

ABS(<expression numérique>)

EXPLICATION

Cette fonction renvoie une valeur de même type que celui du paramètre.

EXEMPLE

```
100 IF ABS(X)> 100 THEN 200
```

ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas de type numérique.

# ACOS FONCTION

## BUT

Cette fonction rend l'angle dont le cosinus est égal à la valeur obtenue par l'évaluation du paramètre.

## SYNTAXE

ACOS (<expression numérique>)

où <expression numérique> doit être comprise dans l'intervalle [-1,1].

## EXPLICATION

L'angle résultant est exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

ACOS rend une valeur comprise entre [0,PI] ou [0,180].

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT ACOS (.5)
RUN
60
```

## ERREURS

L'erreur No 3007 est détectée si <expression numérique> n'est pas comprise dans l'intervalle [-1,1].

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# ANGLE FONCTION

## BUT

Evaluer l'angle entre l'axe des abscisses positives et le vecteur joignant l'origine au point de coordonnées indiquées.

## SYNTAXE

ANGLE (<expr. numérique>,<expr. numérique>)

## EXPLICATION

L'angle résultant est exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT ANGLE (1,1),ANGLE (1,10)
RUN
45          84.289406863
```

## ERREURS

L'erreur No 3008 est détectée si on tente d'évaluer ANGLE (0,0).

L'erreur No 25101 est détectée si l'un des paramètres n'est pas numérique.

# ASIN FONCTION

## BUT

Cette fonction rend l'angle dont le sinus est égal à la valeur obtenue par l'évaluation du paramètre.

## SYNTAXE

ASIN (<expression numérique>)

où <expression numérique> doit être comprise dans l'intervalle [-1,1].

## EXPLICATION

L'angle résultant est exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

ASIN rend une valeur comprise entre [-PI/2,PI/2] ou [-90,90].

## EXEMPLE

```
100 OPTIONS ANGLE RADIANS
110 PRINT ASIN (1)
RUN
1.5707963268
```

## ERREURS

L'erreur No 3007 est détectée si <expression numérique> n'est pas comprise dans l'intervalle [-1,1].

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# ASK INSTRUCTION

## BUT

ASK permet de connaître les paramètres d'édition en cours utilisés pour les impressions non formatées.

## SYNTAXE

ASK <élément ask> <variable numérique>

où <élément ask> peut être MARGIN ou ZONEWIDTH.

## EXPLICATION

L'exécution de cette instruction se déroule de la façon suivante : la variable numérique reçoit pour valeur la longueur de la ligne d'édition ( pour la forme ASK MARGIN ) ou la longueur de la zone d'édition ( pour la forme ASK ZONEWIDTH ).

La longueur des zones d'édition est utilisée pour le formatage des champs obtenus par le séparateur "," dans l'instruction PRINT.

Les valeurs implicites pour MARGIN et ZONEWIDTH sont respectivement 132 et 23.

## EXEMPLE

```
400 ASK MARGIN larg_lin
410 ASK ZONEWIDTH larg_champ
420 PRINT larg_lin,larg_champ

RUN
132                 23
```

## ERREUR

L'erreur No 25102 est détectée si la variable devant recevoir la valeur de MARGIN ou ZONEWIDTH n'est pas de type numérique.

## REMARQUE

Voir l'instruction PRINT pour des informations complémentaires sur MARGIN et ZONEWIDTH.

Voir l'instruction SET qui permet de donner une valeur à ces paramètres.



# ATN FONCTION

## BUT

Cette fonction rend l'angle dont la tangente est égale à la valeur obtenue par l'évaluation du paramètre.

## SYNTAXE

ATN (<expression numérique>)

## EXPLICATION

L'angle résultant est exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT ATN (1)
RUN
45
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# AUTO COMMANDE

## BUT

Cette commande demande à l'interpréteur de numéroter et indenter automatiquement les lignes au fur et à mesure de la saisie.

## SYNTAXE

AUTO ON
AUTO OFF

## EXPLICATION

La forme AUTO ON indique à MEMBASIC qu'il doit effectuer la numérotation automatique des lignes, alors que la forme AUTO OFF suspend cette numérotation automatique.

En mode AUTO ON, MEMBASIC laisse l'utilisateur frapper le premier numéro de ligne et l'instruction associée. Lorsque l'utilisateur enfonce "RETOUR CHARIOT", MEMBASIC propose de lui-même un nouveau numéro par incrément fixe. L'incrément implicite est 10 et peut être modifié par l'instruction STEP.

En mode AUTO ON, l'indentation d'une nouvelle ligne est calquée sur celle de la ligne précédente.

A tout moment, il est possible de quitter la séquence de numérotation. Pour cela, il suffit de déplacer le curseur verticalement ou de frapper "RETOUR CHARIOT" sur un numéro de ligne seul.

Si l'utilisateur frappe un nouveau numéro de ligne suivi d'une instruction, MEMBASIC reprend la numérotation automatique à partir du nouveau numéro introduit par l'utilisateur.

Si le numéro de ligne produit par numérotation automatique, correspond à une ligne existante, l'ancienne version de la ligne est affichée et il y a passage en mode édition pour modification éventuelle.



## EXEMPLE

```
AUTO ON
100 ! Début
110 F=1 ......
```

# BREAK INSTRUCTION

## BUT

Placer des points d'arrêt dans le programme en cours de mise au point.

## SYNTAXE

BREAK

BREAK ON

BREAK OFF

## EXPLICATION

Quand le mode DEBUG est actif, l'exécution de l'instruction BREAK provoque l'interruption de l'exécution, l'affichage sur la console du message

BREAK en ligne...

et le passage en mode commande.

La commande CONT permet de reprendre l'exécution à partir du point où elle a été interrompue.

Les formes BREAK OFF et BREAK ON modifient l'action de la touche CTRL/C qui peut être utilisée pour interrompre le programme comme le ferait une instruction BREAK.

La forme BREAK OFF rend la frappe sur la touche CTRL/C inactive.

La forme BREAK ON rend la frappe sur la touche CTRL/C active.

## ERREUR

Néant.



# CASE INSTRUCTION

## BUT

Cette instruction est utilisée pour construire des séquences structurées de programme.

## SYNTAXE

CASE <cas> , <cas> ...
ou
CASE ELSE

où <cas> peut prendre 3 formes :

expression
expression TO expression
IS relation expression

## EXPLICATION

Une instruction SELECT doit avoir été exécutée préalablement à une instruction CASE.

La valeur obtenue lors de l'évaluation de l'expression associée à l'instruction SELECT est comparée successivement aux diverses plages de valeur contenues dans les différentes instructions CASE associées à SELECT.

Dès que l'une des plages contient la valeur de l'expression, les instructions du bloc CASE correspondant sont exécutées, puis il y a passage à l'instruction suivant immédiatement l'instruction END SELECT.

Si aucune concordance n'est trouvée, alors deux cas se présentent : ou bien il existe un bloc "CASE ELSE" et dans ce cas les instructions qu'il contient sont exécutées, ou bien il n'y a pas de tel bloc et l'erreur 10004 est détectée.

## EXEMPLE

```
200 SELECT CASE A$
210 CASE "A" TO "Z"
220   PRINT "Majuscule"
230 CASE "a" TO "z"
240   PRINT "Minuscule"
250 CASE IS = "é", IS = "è", IS = "à"
260   PRINT "Lettre accentuée"
270 CASE ELSE
280   PRINT "Ce n'est pas une lettre"
270 END SELECT
```

## ERREURS

L'erreur No 10004 est détectée si l'instruction SELECT est exécutée et qu'aucun bloc CASE ne peut être sélectionné.

L'erreur No 10752 est détectée si l'instruction CASE est rencontrée en dehors d'un bloc SELECT CASE ... END SELECT.

## CONSEIL

Ces instructions sont très commodes pour une programmation structurée très lisible. Ne pas hésiter à les utiliser, et faire une indentation des instructions incluses dans un bloc pour améliorer la présentation, ce qui n'augmente pas l'encombrement du programme en mémoire.



# CAUSE EXCEPTION INSTRUCTION

## BUT

Cette instruction permet de "simuler" une erreur.

## SYNTAXE

CAUSE EXCEPTION <index>

## EXPLICATION

La valeur de l'index est évaluée et l'erreur dont le numéro est égal à la valeur de l'index est produite.

Cela permet, en simulant une erreur donnée, de mettre au point la routine d'anomalie.

Il est aussi permis de créer des erreurs spécifiques à l'application.

## EXEMPLE 1

```
.
CAUSE EXCEPTION  3001 ! simule division par zéro
.
```

## EXEMPLE 2

```
.
IF recettes < depenses CAUSE EXCEPTION 100
.
```

où l'erreur No 100 est spécifique à l'application.

## ERREURS

L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.

L'erreur No 25101 est détectée si l'index n'est pas de type numérique.

# CEIL FONCTION

## BUT

CEIL réalise un arrondi supérieur à la nième décimale du paramètre.

## SYNTAXE

CEIL (<expression numérique>)
ou
CEIL (<expression numérique> , <index> )

## EXPLICATION

La forme CEIL à un seul paramètre donne le plus petit entier supérieur ou égal à la valeur numérique du paramètre.

La forme CEIL à deux paramètre CEIL ( x,n ) équivaut à:

CEIL ( x * 10 ^ n ) / 10 ^ n

L'index n doit être positif.

## EXEMPLE

```
PRINT CEIL(3.1); CEIL(3.9); CEIL(-3.1); CEIL(-3.9)
 4  4 -3 -3

PRINT CEIL(3.123,2); CEIL(3.567,2); CEIL(-3.123,1)
 3.13 3.57 -3.1
```

## ERREUR

L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.

L'erreur No 25101 est détectée si l'expression ou l'index n'est pas de type numérique.



# CHAIN INSTRUCTION

## BUT

La commande CHAIN lance l'exécution d'un programme préalablement chargé depuis la mémoire auxiliaire en lui passant éventuellement des paramètres.

## SYNTAXE

CHAIN <spécification de fichier>
[ WITH ( <paramètre> [, <paramètre> ... ] ]

Les paramètres sont, soit des noms de variables, soit des constantes numériques ou chaînes.

## EXPLICATION

CHAIN provoque :

- la fermeture des fichiers qui sont ouverts,
- l'effacement de la table des variables après extraction des éventuels paramètres,
- le chargement, puis l'exécution du programme indiqué,
- l'affectation éventuelle à de nouvelles variables, des valeurs passées en paramètres.

<spécification de fichier> est une expression chaîne contenant le nom et éventuellement le chemin d'accès au programme. Le suffixe implicite utilisé, si la spécification de fichier n'en fournit pas, sera .PRG. Dans le cas où aucun chemin particulier n'est proposé dans la spécification de fichier, les différents chemins enregistrés grâce à l'instruction PATH seront utilisés successivement (voir cette instruction pour plus de détails). Enfin si le programme ne se trouve dans aucun des chemins spécifiés, il sera cherché dans le répertoire particulier /MEMTOOLS.

Dans le cas où le programme à exécuter n'est pas trouvé, le programme de départ se poursuit à l'instruction suivant l'instruction CHAIN.

Les paramètres ne seront passés au programme suivant que si celui-ci débute par l'instruction PROGRAM qui définit comment recevoir les données passées. L'instruction PROGRAM indique les noms des variables qui sont destinées à recevoir les paramètres. Ce sont des noms de variables numériques entières ou flottantes, ou des noms de variables chaînes. Les variables chaînes définies par l'instruction PROGRAM sont automatiquement de taille fixe, leur longueur maximale étant définie par le paramètre passé. Les variables tableaux sont indiquées de façon particulière pour que le contrôle du nombre d'indices puisse être effectué (voir l'instruction PROGRAM pour plus d'informations).

Les paramètres sont passés par valeur. La correspondance entre les paramètres et les variables destinées à les recevoir dans le programme chaîné est, suivant le type du paramètre :

- Constante numérique ou variable simple numérique:

  La variable réceptrice doit être numérique simple, flottante ou entière ( elle ne peut être entière que si la valeur passée ne provoque pas de dépassement).

- Constante chaîne :

  La variable réceptrice doit être de type chaîne (de nom terminé par $). Ce doit être une chaîne de taille fixe de longueur maximum égale à la taille de la constante chaîne passée en paramètre.

- Variable simple chaîne de taille variable :

  La variable réceptrice doit être de type chaîne (de nom terminé par $). Ce doit être une chaîne de taille fixe de longueur maximum 50 ou de la taille de la chaîne passée en paramétre si celle-ci excède 50.

- Variable simple chaîne de taille fixe :

  La variable réceptrice doit être de type chaîne (de nom terminé par $). Ce doit être une chaîne de taille fixe de mêmes caractéristiques que celle passée en paramètre.



- Tableaux :

  Les tableaux numériques et les tableaux de chaînes fixes sont passés dans des variables tableaux réceptrices de types équivalents et de même nombre d'indices.

  LES TABLEAUX DE CHAINES DE TAILLES VARIABLES NE PEUVENT PAS ETRE PASSES AU PROGRAMME CHAINE.

Si le nombre de paramètres de l'instruction PROGRAM du programme chaîné est inférieur au nombre de paramètres passés par CHAIN, les données supplémentaires sont perdues. (C'est également ce qui se passe pour l'ensemble des données si le programme chaîné ne comporte pas d'instruction PROGRAM.)

Si le nombre de paramètres de l'instruction PROGRAM du programme chaîné est supérieur au nombre de paramètres passés par CHAIN, les variables complémentaires sont initialisées à 0 ou vide. (C'est également ce qui se passe pour l'ensemble des paramètres si le second programme est lancé par RUN ou CHAIN sans paramètre.)

## EXEMPLE

```
100 valeur_a_passer = 121.1
110 CHAIN "SUIVANT" WITH ( valeur_a_passer )
```

## ERREURS

L'erreur No 28010 est détectée si le programme spécifié n'a pas été trouvé.

L'erreur No 25802 est détectée si l'on tente de passer en paramètre un tableau de chaînes de tailles variables.

Les autres erreurs possibles lors du passage des paramètres se produisent après lancement du programme chaîné et seront signalées lors de l'exécution de l'instruction PROGRAM.

## REMARQUE

L'instruction CHAIN sans paramètre est équivalente à l'instruction RUN dans sa forme :
RUN <spécification de fichier>.

# CHDIR INSTRUCTION

## BUT

Cette instruction change le répertoire courant.

## SYNTAXE

CHDIR <expression chaîne>

où <expression chaîne> doit représenter un chemin vers un répertoire existant.

## EXPLICATION

Les différents répertoires d'un disque sont structurés en arbre. Le répertoire courant est celui qui sera utilisé par toutes les commandes MEMSOFT qui accèdent au disque si aucun répertoire particulier n'est spécifié. L'instruction CHDIR change le répertoire courant.

Le répertoire particulier situé à la racine de l'arbre est atteint par le chemin noté "/". Un répertoire situé sous la racine est atteint par le chemin "/<nom-répertoire>". L'arbre sera parcouru en ajoutant "/<nom>" à la chaîne en construction pour désigner chaque noeud du chemin.

Si le premier "/" est omis, le chemin indiqué est ajouté au chemin en cours pour devenir le nouveau chemin en cours.

## EXEMPLE

```
1000 CHDIR "A:/MEMSOFT/BASE"
```

Le répertoire courant devient le sous-répertoire BASE du répertoire MEMSOFT.

## ERREUR

L'erreur no 28110 est détectée si le répertoire désigné n'existe pas.

L'erreur No 25102 est détectée si le paramètre n'est pas une chaine.



# CHDIR$ FONCTION

## BUT

Donne le répertoire en cours.

## SYNTAXE

CHDIR$

## EXPLICATION

La fonction sans paramètre CHDIR$ renvoie une chaîne contenant le nom du répertoire en cours. La chaîne ainsi produite est conforme à la syntaxe exigée par la commande CHDIR, elle pourra donc servir ultérieurement pour une sélection de répertoire.

## EXEMPLE

```
PRINT CHDIR$
C:/MEMSOFT/TEST
```

Indique que le répertoire en cours est le sous-répertoire TEST du répertoire MEMSOFT.

```
CHDIR CHDIR$
```
ne change pas le répertoire en cours.

## ERREUR

Néant.

# CHR$ FONCTION

## BUT

Cette fonction donne une chaîne de UN caractère. Ce caractère correspond au code du jeu de caractères utilisé. La valeur numérique de l'expression, donnée comme paramètre, est évaluée et CHR$ donne une chaîne de longueur 1 contenant le caractère de même code que la valeur numérique de l'expression.

Si la valeur numérique de l'expression n'est pas entière, elle est arrondie à l'entier le plus proche.

## SYNTAXE

CHR$(<expression numérique>)

## EXPLICATION

Cette fonction permet de mettre dans des chaînes de caractères à la fois des caractères qui peuvent être introduits au clavier et des caractères qui ne peuvent pas être introduits au clavier, par exemple des caractères de contrôle.

## EXEMPLE 1

Pour obtenir un signal sonore, il suffit d'écrire en MEMBASIC:

```
PRINT CHR$(7);
```

car le caractère de code 7 correspond au signal sonore.

## EXEMPLE 2

Pour mettre dans une chaîne le caractère "retour chariot", il suffit d'écrire:

```
retour$ = CHR$(13)
```

car le caractère de code 13 correspond au "RETOUR CHARIOT".



## EXEMPLE 3

Pour afficher à l'écran le jeu de caractères d'un micro-ordinateur on peut par exemple écrire la séquence suivante :

```
100 FOR I = 32 TO 255
110   PRINT I , CHR$(I)
120 NEXT I
```

## ERREURS

L'erreur No 25101 est détectée si le paramètre n'est pas de type numérique.

L'erreur No 4000 est détectée si la valeur du paramètre est négative ou supérieure à 255.

L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.

## REMARQUE

Voir l'instruction ORD qui donne le code d'un caractère contenu dans une chaîne de longueur 1.

## CONSEIL

Faire attention au fait qu'avec certaines imprimantes , le jeu de caractères disponible n'est pas exactement le même que le jeu de caractères disponible à l'écran et que les caractères de contrôle ont des effets particuliers. Cela signifie que des instructions PRINT CHR$(....) peuvent parfaitement fonctionner avec l'écran et ne pas donner le résultat escompté sur imprimante .

# CLEAR KEYBOARD INSTRUCTION

## BUT

Vide les caractères d'avance dans la mémoire tampon du clavier.

## SYNTAXE

```
CLEAR KEYBOARD
```

## EXPLICATION

Si l'utilisateur enfonce des touches du clavier avant d'y être invité, un certain nombre de caractères est conservé et serviront aux prochaines entrées. CLEAR KEYBOARD permet d'éliminer ces caractères pour s'assurer de la validité d'une réponse particulièrement importante.

## EXEMPLE

```
100 CLEAR KEYBOARD
110 INPUT PROMPT "Répondre O pour détruire " : a$
```

## ERREUR

Néant.



# CLOSE INSTRUCTION

## BUT

Cette instruction ferme un fichier PRODOS ou un fichier de communication, ou l'ensemble de ces fichiers.

## SYNTAXE

Deux formes de l'instruction CLOSE existent:

```
CLOSE
```

ou

```
CLOSE <numéro de fichier>
```

Dans cette deuxième forme, <numéro de fichier> est une expression numérique indiquant le numéro attribué au fichier lors de son ouverture.

## EXPLICATION

L'instruction OPEN permet d'associer à un fichier PRODOS un numéro utilisé pour écrire ou lire dans ce fichier. De même, l'instruction OPEN "COM..." associe un numéro à un port d'entrées/sorties série. L'instruction CLOSE permet de rompre ce lien et aussi, si elle s'applique à un fichier ouvert en écriture, de sauver sur disque les modifications apportées.

La forme CLOSE sans paramètre ferme tous les fichiers PRODOS et les fichiers de communication ouverts. La forme CLOSE <numéro de fichier> ferme un fichier PRODOS ou un fichier de communication particulier.

La fermeture d'un fichier PRODOS ou d'un fichier de communication permettra d'en ouvrir un nouveau (rappelons que le nombre de fichiers ouverts simultanément a une limite qui dépend de l'installation du système). En tout état de cause, le nombre maximum de fichiers PRODOS ou de communication ouverts simultanément est 10.

L'instruction CLOSE n'a aucune action sur les fichiers MEMFILE. Pour ceux-ci, la fermeture se fait par l'instruction MEMFILE : LET "#CLEAR...".

## EXEMPLE

```
100 CLOSE     ! ferme tous les fichiers
110 CLOSE 4   ! ferme le fichier de numéro 4
```

## ERREUR

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier PRODOS ou fichier de communication ouvert.

## REMARQUE

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications séries.



# CLS INSTRUCTION

## BUT

Cette instruction efface le contenu de la fenêtre de travail.

## SYNTAXE

CLS

## EXPLICATION

CLS efface le contenu de la fenêtre de travail et place le curseur à l'origine de la fenêtre, c'est à dire au coin supérieur gauche.

## EXEMPLE

```
100 CLS
110 INPUT PROMPT "NOM DU FICHIER A LIRE" : N$
...
```

## ERREUR

Néant.

## REMARQUE

La fenêtre de travail qui sera effacée par l'instruction CLS ne sera pas toujours la même : ce sera la fenêtre d'édition si la commande est exécutée en mode immédiat, la fenêtre d'exécution si l'instruction est exécutée en mode programme.

# COMSTAT FONCTION

## BUT

Cette fonction donne l'état actuel du port série associé à un fichier de communication.

## SYNTAXE

COMSTAT ( <numéro de fichier> )

## EXPLICATION

Il peut arriver que les communications entre un périphérique ou autre ordinateur et votre programme MEMSOFT ne fonctionne pas correctement, en général à cause d'un problème de câble ou d'un problème de protocole. La fonction COMSTAT peut alors être utile pour comprendre la cause de la non communication. La valeur donnée doit être analysée bit par bit. Pour plus d'informations, se référer au manuel de référence de votre micro-ordinateur.

Vérifiez également votre câble et votre protocole en lisant attentivement les explications données au chapitre 1.17.

La signification bit par bit de la réponse de COMSTAT est ( bit de droite = bit 0):

bit 1 = 1 si un caractère a été reçu.
bit 0 = 1 si le port RS232 est prêt à émettre.

## EXEMPLE

```
100 OPEN "COM2",1
110 PRINT "L'état de la ligne est : "; COMSTAT(1)
```

## ERREUR

L'erreur No 28503 est détectée si le numéro de fichier ne correspond pas à un fichier de communication ouvert.



## REMARQUE

N'utilisez COMSTAT qu'en cas de dernier recours car la compréhension des informations fournies nécessite une connaissance approfondie des connexions séries RS232.

# CONT COMMANDE

## BUT

Cette commande permet de reprendre l'exécution d'un programme interrompu exactement à l'endroit où l'interruption a eu lieu. Cette commande CONT est sans effet si l'interruption est due à une erreur fatale.

## SYNTAXE

CONT

## EXPLICATION

Lorsqu'un programme est interrompu, soit par la rencontre d'une instruction BREAK, soit par l'enfoncement de la touche "POMME CTRL C", l'interpréteur MEMBASIC affiche le numéro de la ligne où l'interruption s'est produite. Si l'utilisateur souhaite poursuivre l'exécution il lui suffit de frapper :

```
CONT
```

pour relancer l'exécution.

Par contre, CONT est sans effet sur un programme interrompu par suite d'une erreur.

Cette instruction de reprise est particulièrement utile en phase de mise au point. Elle permet d'interrompre le programme, de vérifier le contenu des variables, puis de reprendre le déroulement normal.

## EXEMPLE

L'exécution d'un programme, interrompue par l'instruction BREAK, provoque l'affichage suivant :

```
BREAK en ligne 1000
```

En frappant CONT suivi du caractère "RETOUR CHARIOT", l'exécution reprend au point où elle avait été interrompue.



## ERREUR

L'erreur N° 29002 est détectée si l'exécution du programme ne peut pas être reprise.

## REMARQUE

Si une instruction WHEN EXCEPTION GOTO ... a été exécutée pour gérer les anomalies, CONT ne sert plus puisque les deux cas cités ne provoquent plus l'arrêt du programme.

# COPY INSTRUCTION

## BUT

Copie d'un fichier d'un répertoire vers un autre répertoire ou sous un autre nom. La copie peut, si on le désire, être invisible dans le catalogue.

## SYNTAXE

COPY <expression chaîne> TO <expression chaîne>

Les deux expressions chaînes représentent des spécifications de fichier.

La deuxième expression chaîne commence par le caractère "!" si l'on désire l'invisibilité dans le catalogue. Attention, invisibilité seulement quand vous faites un catalogue à partir de MEMSOFT. Si vous êtes dans le bureau, ce fichier sera visible.

Elle commence par le caractère "@" si l'on désire que le fichier d'arrivée écrase un autre fichier de même nom.

## EXPLICATION

Le fichier spécifié dans la première expression chaîne est copié avec les spécifications données dans la deuxième expression chaîne.

Si le fichier d'arrivée existe déjà, il sera écrasé par le nouveau si on précède son nom du caractère "@".

Si le nom du fichier d'arrivée commence par "!", le fichier copié ne sera pas visible dans le catalogue du disque ( instruction DIR ), ceci n'est valable que si vous êtes sous MEMSOFT. Par contre, si vous connaissez le nom du fichier, toutes les instructions MEMSOFT sont utilisables. Par exemple, s'il s'agit d'un masque, il pourra être ouvert et utilisé normalement. Les fichiers non visibles ne peuvent pas être copiés depuis le bureau.

Aucun suffixe implicite n'est pris. Il faut donc, pour les deux fichiers préciser explicitement les suffixes.



On peut copier plusieurs fichiers à la fois en utilisant dans les noms des fichiers les caractères spéciaux ? et *.
Un ? dans un nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans un nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

Si le nom du fichier à copier est un nom de répertoire, c'est le répertoire complet qui sera copié. Si le nom du fichier d'arrivée est un nom de répertoire, les fichiers à copier seront copiés dans ce répertoire.

En mode immédiat, la liste des fichiers copiés sera affichée dans la fenêtre courante.

## EXEMPLES

```
COPY "original.msk" TO "nouveau.hlp"
  copie d'un fichier particulier

COPY "orig*.*" TO "nouv*.*"
  copie d'un ensemble de fichiers :
  ORIGABC.XYZ est copié en NOUVABC.XYZ

COPY "a:*.*" TO "monrep"
  copie depuis la disquette vers un répertoire.
```

## ERREURS

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

L'erreur No 28010 est détectée si le fichier d'origine n'existe pas.

L'erreur No 28030 est détectée si le fichier d'arrivée existe déjà.

L'erreur No 28040 est détectée si le fichier d'arrivée est impossible à créer.

# COS FONCTION

## BUT

Cette fonction rend le cosinus d'un angle.

## SYNTAXE

COS (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat est une valeur comprise dans l'intervalle [-1,1].

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT COS (60)
RUN
.5
```

## ERREURS

L'erreur No 3090 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# COT FONCTION

## BUT

Cette fonction rend la cotangente d'un angle.

## SYNTAXE

COT (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat obtenu est l'inverse de la tangente soit celui de la division du cosinus par le sinus.

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT COT (30)
RUN
1.7320508076
```

## ERREURS

L'erreur No 3091 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# CSC FONCTION

## BUT

Cette fonction rend la cosécante d'un angle.

## SYNTAXE

CSC (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat obtenu est celui de l'opération :

1 / SIN

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT CSC (90)
RUN
1
```

## ERREURS

L'erreur No 3091 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# CURDRIVE INSTRUCTION

## BUT

L'instruction CURDRIVE permet d'imposer le disque courant.

## SYNTAXE

CURDRIVE <expression chaîne>

## EXPLICATION

C'est le premier caractère de la chaîne qui indiquera le nouveau disque courant. Cette première lettre devra donc être A,B,C....

La lettre spécifiant le disque peut être indiquée aussi bien en majuscule qu'en minuscule.

Il est fréquent que les disques A et B soient des lecteurs de disquettes et que le disque C soit un disque dur. Dans ce dernier cas le disque B peut ne pas exister.

## EXEMPLE

```
CURDRIVE "A:"
```
Le disque courant devient le disque A.

## ERREURS

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

L'erreur No 28900 est détectée si le disque spécifié est invalide.

## REMARQUE

L'instruction CURDRIVE change le disque en cours mais le répertoire en cours sur chaque disque n'est pas modifié.

# CURDRIVE$ FONCTION

## BUT

La fonction CURDRIVE$ donne une chaîne de deux caractères indiquant le disque en cours.

## SYNTAXE

CURDRIVE$

## EXPLICATION

Le premier caractère de la chaîne rendue est la lettre représentant le disque : A,B,C ... Le deuxième caractère est toujours ":".

## ERREUR

Néant.

## REMARQUE

La chaîne obtenue est directement utilisable par l'instruction CURDRIVE ou la fonction FREE.



# CURLINE FONCTION

## BUT

Cette fonction sans paramètre est un compteur de lignes d'édition pour les instructions MEMSCREEN LET "?...." et LET "HARDCOPY...".

## SYNTAXE

CURLINE

## EXPLICATION

Après chaque ligne éditée par l'une des deux instructions MEMSCREEN LET "?..." ou LET "HARDCOPY" la valeur rendue par cette fonction augmente de 1.

Il existe une valeur maximale que CURLINE ne peut dépasser et qui, lorsqu'elle est atteinte, provoque la réinitialisation du compteur à 0.

Ce nombre maximum, que CURLINE ne peut atteindre, peut être connu et modifié. Il est lu par la fonction MAXLINE. Il peut être modifié par SET MAXLINE.

La valeur initiale du compteur CURLINE peut aussi être fixée par SET CURLINE.

## EXEMPLE

```
100 ! édition de 20 lignes par page de 40 lignes
110 SET CURLINE 0  ! initialise CURLINE
120 SET MAXLINE 40 ! fixe la page à 40 lignes
550 IF CURLINE >= 20 THEN LET "?"
...
```

## ERREUR

Néant.

# DATA INSTRUCTION

## BUT

Cette instruction permet d'incorporer, dans le programme, des données qui seront lues au moyen de l'instruction READ.

## SYNTAXE

DATA <constante> [ , <constante>]...

## EXPLICATION

Les instructions DATA ne sont pas exécutables et peuvent se trouver n'importe où dans le programme. Une instruction DATA peut contenir autant de constantes que la longueur de la ligne le permet. Un programme peut contenir autant d'instructions DATA qu'il y a de place en mémoire.

Les constantes contenues dans DATA peuvent être de type entier, réel ou chaîne de caractères; les expressions ne sont pas autorisées dans cette liste. Les chaînes de caractères doivent être encadrées par des guillemets si elles contiennent un séparateur (virgule, deux points, point d'exclamation) ou si les espaces à gauche ou à droite doivent être conservés. Si elles ne contiennent pas de séparateur, les guillemets sont facultatifs.

Les constantes contenues dans des instructions DATA sont lues au moyen de l'instruction READ. Le type des constantes ainsi lues, doit correspondre au type des variables de la liste contenues dans l'instruction READ correspondante.

Si l'utilisateur désire que les données contenues dans une instruction DATA puissent être relues plusieurs fois au cours de l'exécution du programme, il doit utiliser l'instruction RESTORE.

Les instructions DATA sont locales à l'unité de programme. Par conséquent, le contenu d'une instruction DATA ne peut pas être lu par les instructions READ d'une autre unité de programme.



## EXEMPLE

```
100 REM initialisation
110 READ A,B,N$
120 READ D$
...
200 DATA 10,20,DUPONT
210 DATA "19 Décembre 1998"
```

## ERREUR

L'erreur No 29004 est détectée en cas de tentative d'utilisation de DATA en mode immédiat.

## CONSEILS

1. Cette méthode est très pratique pour des données dont on modifie rarement la valeur.

2. Au lieu de disséminer les instructions DATA dans le programme, il est souvent préférable de les rassembler au début ou en fin de programme, ce qui permet d'obtenir une meilleure lisibilité du programme.

3. En début de chaque groupe de DATA qui pourra être relu par RESTORE, on pourra placer une étiquette explicitant le contenu des DATA, le programme en sera plus clair. Par exemple :

```
100 'taux de tva'
110 DATA 7.5, 18.6, 33.33
........
200 RESTORE 'taux de tva'
210 FOR i=1 TO 3
220     READ taux_tva(i)
230 NEXT
```

# DATE FONCTION

## BUT

DATE donne la date du jour exprimée sous la forme numérique AAJJJ, JJJ représentant le nombre de jours depuis le début de l'année.

## SYNTAXE

DATE

## EXPLICATION

Cette fonction utilise la date fournie par le système d'exploitation. Il est donc souhaitable d'avoir fourni une date correcte au système d'exploitation, sinon DATE renvoie toujours la même date qui est sans signification.

## ERREUR

Néant.

## REMARQUE

Voir DATE$ qui fournit une date formatée.



# DATE$ FONCTION

## BUT

DATE$ donne la date du jour exprimée sous la forme chaîne "AAAAMMJJ".

"AAAA" représente l'année complète sur 4 chiffres.
"MM" représente le mois de "01" à "12".
"JJ" représente le jour de "01" à "31".

## SYNTAXE

DATE$

## EXPLICATION

La date fournie par la fonction sans paramètre DATE$ est celle fournie par le système d'exploitation. Il est donc souhaitable d'avoir fourni une date correcte au système d'exploitation, sinon DATE$ renvoie toujours la même date qui est sans signification.

## ERREUR

Néant.

## REMARQUE

Le format utilisé par la fonction DATE$ pour donner la date du jour est le même que celui utilisé par MEMSCREEN pour rendre les dates saisies dans un masque. Ce format respectant l'ordre chronologique, les dates données par les saisies de masques MEMSCREEN pourront être directement comparées entre elles et avec la date du jour fournie par DATE$.

# DEBUG INSTRUCTION

## BUT

L'instruction DEBUG permet de rendre le mode DEBUG actif ou inactif.

## SYNTAXE

```
DEBUG ON
ou
DEBUG OFF
```

## EXPLICATION

L'instruction DEBUG ON rend le mode DEBUG actif et permet l'utilisation des instructions BREAK et TRACE.

L'instruction DEBUG OFF rend le mode DEBUG inactif, ce qui signifie que les instructions BREAK et TRACE n'agissent plus.

Dans ce cas, si une instruction BREAK ou TRACE est rencontrée, elle est ignorée et le programme passe à l'instruction suivante.

## EXEMPLE

```
100 DEBUG ON
.
300 GOSUB 1000
.
1000 DEBUG OFF
```

## ERREUR

Néant.

## REMARQUE

Voir également BREAK ON et BREAK OFF qui rendent actif ou inactif l'arrêt du programme par la touche POMME CTRL C.



# DECLARE STRING INSTRUCTION

## BUT

Cette instruction permet de déclarer la longueur maximum que peuvent prendre une ou plusieurs variables chaînes simples.

## SYNTAXE

```
DECLARE STRING [*(<exp1>)] <variable> [*(<exp2>)]
               [,<variable> [*(<expn>)]]...
```

où <exp1>, <exp2> ... <expn> sont des expressions entières.

Dans le cas où les paramètres longueur sont des constantes entières, il n'est pas nécessaire de les faire figurer entre parenthèses.

## EXPLICATION

La déclaration DECLARE STRING indique la longueur maximum des chaînes dont le nom figure dans la liste selon le procédé suivant :

- Le paramètre facultatif *(<exp1>) indique la longueur maximum des chaînes de la liste pour lesquelles aucune longueur maximum n'est déclarée.

- Si une variable est déclarée spécialement avec une longueur maximum sous la forme <variable> *<exp> , alors cette longueur est prise à la place de la longueur implicite.

La longueur d'une variable chaîne ne peut dépasser 255 caractères.
L'instruction DECLARE STRING n'est pas utilisable avec des tableaux de chaînes.

## EXEMPLE 1

```
110 DECLARE STRING *10 a$,b$,c$ *15
```

Dans cet exemple, les variables a$ et b$ ont une longueur maximum de 10 alors que la variable c$ ne peut excéder la longueur 15.

## EXEMPLE 2

```
100 INPUT PROMPT "Donner la longueur max de a$" : n
110 DECLARE STRING a$*(n)
```

## ERREURS

L'erreur 1051 est détectée si, lors de l'exécution, il se produit une tentative d'affectation d'une chaîne de longueur supérieure à celle attribuée à la variable.

L'erreur 26002 est détectée si la chaîne a déjà été déclarée.

## REMARQUE

L'instruction DIM permet de dimensionner des tableaux de chaînes en spécifiant la longueur maximale des chaînes.

## CONSEIL

Cette instruction, en limitant la longueur des chaînes, permet, d'une part de détecter des erreurs de programmation, d'autre part de réduire les temps d'exécution.

Son utilisation est donc à conseiller chaque fois que cela est possible.



# DEG FONCTION

## BUT

Cette fonction convertit un nombre de radians en degrés.

## SYNTAXE

DEG (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians.

## EXPLICATION

La fonction DEG rend le nombre entré converti en degrés.

## EXEMPLE

```
100 RADIANS = .7853981633974
110 PRINT DEG (RADIANS)
RUN
45
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# DELETE COMMANDE

## BUT

Cette commande est utilisée pour détruire des lignes logiques de programmes.

## SYNTAXE

DELETE <numéro> [,<numéro>] ...

DELETE <numéro> TO <numéro>[,<numéro> TO <numéro>]

DELETE TO <numéro>

DELETE <numéro> TO

Dans tous les cas, l'instruction pourra se terminer par une constante chaîne représentant une spécification de fichier.

## EXPLICATION

La commande DELETE permet de supprimer des lignes du programme en mémoire selon les modalités suivantes :

- La forme <numéro> signifie que la ligne correspondante sera supprimée.

- La forme <numéro> TO <numéro> indique la séquence de lignes à supprimer. Si le premier numéro n'est pas inférieur au second, aucune ligne n'est supprimée.

- La forme <numéro> TO indique que toutes les lignes sont à détruire à partir de celle indiquée.

- La forme TO <numéro> indique que toutes les lignes sont à détruire jusqu'à celle indiquée.

Ces quatre formes peuvent se combiner ainsi que le montre l'exemple 1 ci-dessous.

Utilisée avec un nom de fichier, DELETE permet de supprimer des lignes après les avoir sauvegardées sous forme texte dans le fichier spécifié.



## EXEMPLE 1

DELETE 100 TO 150,200,500 TO 600,1000 TO

détruit les lignes 100 à 150, 200, 500 à 600 et toutes les lignes à partir de 1000.

## EXEMPLE 2

DELETE 100 TO 300 "attente.lin"

détruit les lignes 100 à 300 après les avoir sauvegardées dans le fichier attente.lin.

## ERREUR

Néant.

## REMARQUE

Si une spécification de fichier de sauvegarde des lignes détruites est utilisée, le suffixe PRG est pris si aucun suffixe n'est indiqué.

# DIM INSTRUCTION

## BUT

L'instruction DIM permet de déclarer des tableaux et de donner la plage de variation des indices.
Elle permet également de définir des tableaux de chaînes en fixant leur longueur maximale.

## SYNTAXE

```
     variable(max1[,max2]...)
DIM                                [*(<expression>)]..
     variable(borne1 TO borne2,)
```

max1, max2, borne1, borne2 sont des expressions numériques.

Un nom de variable déclaré comme tableau ne peut pas être utilisé comme nom de variable simple.

## EXPLICATION

La forme DIM variable(max1,max2...) permet de déclarer des tableaux dont les indices varient de la valeur minimum implicite (qui est souvent 1) à la valeur maximum déclarée qui doit être positive.

L'instruction OPTION BASE permet de faire partir les indices de la valeur 0 ou d'une autre valeur positive ou négative.

La forme DIM variable(borne1 TO borne2) permet de déclarer que les indices d'un tableau peuvent varier de la valeur borne1 jusqu'à la valeur borne2.

Les valeurs de borne1 et borne2 peuvent être négatives ou positives mais borne1 doit être inférieure à borne2.

Cette instruction doit être exécutée avant toute référence au tableau concerné ou à des éléments du tableau.

Le redimensionnement d'un tableau n'est pas autorisé, les indices doivent être compris entre -32768 et +32767.



Dans le cas des tableaux de chaînes de caractères, il est possible de spécifier la longueur maximale des chaînes qui le composent en faisant suivre le nom de *(<expression>). L'expression indique alors la taille de l'élément. Si l'expression n'est pas une constante, elle devra être entre parenthèses.

## EXEMPLES

```
100 DIM TABLEAU(10,10), CA(1980 TO 1985)
```

Dimensionne TABLEAU et CA. L'indice minimum est fixé par OPTION BASE pour TABLEAU alors qu'il est donné explicitement pour CA.

```
110 DIM CHAINE$(18) * (80)
```

Définit un tableau de chaînes d'indice maximum 18, d'indice minimum spécifié par OPTION BASE, chaque chaîne ne pouvant excéder la longueur 80.

## ERREURS

L'erreur No 26001 est détectée en cas de redimensionnement d'un tableau.

Au cours de l'exécution, si un indice sort des limites indiquées, l'erreur No 2001 est détectée.

L'erreur 25002 est détectée si un tableau est utilisé sans indice.

L'erreur 25001 est détectée s'il y a déjà une variable simple de même nom que le tableau que l'on veut utiliser.

## REMARQUE

Voir l'instruction OPTION DIM qui permet de donner la valeur implicite de l'indice maximum des tableaux utilisés sans dimensionnement explicite.

# DIR INSTRUCTION

## BUT

Obtenir le contenu d'un répertoire.

## SYNTAXE

DIR [(expression chaîne>]

où <expression chaîne> doit représenter un chemin vers le répertoire syntaxiquement valable pour le système d'exploitation.

## EXPLICATION

La forme DIR sans paramètre permet d'obtenir le contenu du répertoire courant.

La forme DIR <expression chaîne> permet d'obtenir le contenu de n'importe quel répertoire spécifié avec un eventuel critère de sélection.

L'expression chaîne peut contenir les caractères spéciaux ? et * dans le nom du fichier qui permettent d'utiliser la commande DIR avec plus de souplesse.
Un point d'interrogation dans un nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un astérisque dans un nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

DIR affiche, pour chaque fichier, sa date de création ou de dernière modification, sa taille en octets.

En fin d'affichage, DIR donne le nombre de fichiers listés ainsi que la taille totale occupée par ces fichiers sur le disque.

L'affichage se fait sur l'écran sauf si une instruction PRINTER <imprimante> ou PRINTER <nom de fichier> a été exécutée.

Le listage du répertoire à l'écran peut être suspendu ou ralenti :



- La touche SHIFT suspend le listage.
  Tant que l'une des touches SHIFT est enfoncée, l'exécution de DIR est arrêtée.

  On peut alors :

  - avancer d'une ligne en enfonçant puis relachant la touche CTRL (sans relacher la touche SHIFT),
  - passer en mode "gestion des fenêtres" en enfonçant la touche "OPTION Clic-Souris" pour consulter d'autres fenêtres,
  - stopper le listage en enfonçant POMME CTRL C.

  Au relachement de la touche SHIFT, le listage du répertoire reprend normalement.

## EXEMPLES

| | |
|---|---|
| DIR "B:*.MEM" | affiche tous les fichiers du répertoire en cours de l'unité B dont le suffixe est MEM |
| DIR "AZE?TY.ABC" | affiche tous les fichiers du répertoire en cours du disque par défaut, dont le nom comporte 6 caractères, commence par AZE suivi d'un caractère quelconque puis de TY et dont le suffixe est ABC |
| DIR "AB*.M*" | affiche tous les fichiers dont le nom commence par AB et le suffixe par M |

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

# DISKSIZE FONCTION

## BUT

Rend la place totale disponible en octets sur le disque spécifié.

## SYNTAXE

DISKSIZE ( <expression chaîne> )

## EXPLICATION

Rend la capacité totale (en octets) du disque dont la lettre de désignation est donnée dans le premier caractère de l'<expression chaîne>.

Elle ne dépend pas du répertoire actuellement utilisé.

La taille indiquée est donnée avec une précision de plus ou moins 512 octets.

## EXEMPLE

```
105 total=DISKSIZE (CURDRIVE$)
108 libre=FREE (CURDRIVE$)
110 PRINT "Place totale disque"; total
120 PRINT "Place libre"; libre
130 PRINT "Place occupée"; total - libre
140 PRINT "soit"; (total - libre) * 100 /total ;"%"
```

## ERREURS

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

L'erreur No 28900 est détectée si le disque spécifié est invalide.



# DO...LOOP INSTRUCTION

## BUT

Les instructions DO et LOOP sont destinées à construire des boucles de traitement.

## SYNTAXE

```
DO
ou
DO UNTIL <condition>
ou
DO WHILE <condition>
```

la boucle finit par :

```
LOOP
ou
LOOP UNTIL <condition>
ou
LOOP WHILE <condition>
```

Une boucle DO doit contenir une instruction LOOP et une seule.

## EXPLICATION

Ces différentes formes permettent de construire des boucles avec des conditions de sortie diverses :

```
DO
...
LOOP
```

```
DO UNTIL ...
...
LOOP
```

```
DO
...
LOOP UNTIL
```

```
DO WHILE ...
...
LOOP
```

```
DO
...
LOOP WHILE
```

Il est également possible de combiner deux critères de sortie au moyen des mots-clés UNTIL et WHILE, l'un accompagnant une condition dans DO et l'autre une condition dans LOOP.

Les critères sont exprimés de la façon suivante :

UNTIL <condition> : la boucle continue JUSQU'A ce que la condition devienne vraie.

WHILE <condition> : la boucle continue TANT QUE que la condition est vraie.

Lorsque la boucle se termine sur le DO, MEMBASIC cherche le LOOP associé et poursuit l'exécution du programme à l'instruction suivante.

Il est possible de sortir d'une boucle DO au moyen de l'instruction EXIT DO qui provoque le passage à l'instruction qui suit immédiatement l'instruction LOOP associée.

Il est possible d'aller directement à l'instruction LOOP grâce à l'instruction SKIP DO.

## EXEMPLE

```
100 INPUT PROMPT "Montant ": montant
110 infla=.08
120 annee=0
130 montant_final=montant
140 DO
150    montant_final=montant_final * ( 1+infla )
160    annee=annee+1
170 LOOP UNTIL montant_final >= montant * 2
```

## ERREUR

L'erreur No 10701 est détectée si une instruction LOOP est rencontrée sans qu'une instruction DO ait été exécutée au préalable.



# ELSE MOT-CLE

## BUT

Ce mot-clé est utilisé dans les instructions IF...THEN...ELSE, ON GOTO, ON GOSUB, CASE. Se reporter aux pages relatives aux instructions IF, ON GOSUB, ON GOTO, CASE.

## SYNTAXE

Voici un rappel syntaxique des différents cas :

Forme IF monoligne :

```
IF <condition> THEN <instr.> ELSE <numéro lig>
                                  <étiquette>
                                  <instruct.>
```

Forme IF multilignes :

```
IF <condition> THEN
.....
ELSEIF
.....
ELSE
.....
ENDIF
```

dans ON GOTO/GOSUB ...

```
ON <expres> GOTO/GOSUB <liste> ELSE <affect.>
                                    <numligne>
                                    <étiquet>
```

dans SELECT CASE

```
SELECT CASE ...
CASE ...
.....
CASE ELSE
.....
END SELECT
```

## ERREUR

Néant.

# END INSTRUCTION

## BUT

END indique la fin de la séquence d'instructions du programme.

## SYNTAXE

```
END
```

## EXPLICATION

Cette instruction provoque : l'arrêt de l'exécution du programme, la fermeture des fichiers qui étaient encore ouverts juste avant qu'elle ne soit exécutée, le passage en mode commande.

## EXEMPLE

```
940  INPUT PROMPT "Voulez-vous continuer": r$
1000 IF r$="NON" THEN END
...
```

## ERREUR

Néant.



# END IF INSTRUCTION

## BUT

END IF indique la fin du dernier bloc d'une instruction IF.

## SYNTAXE

```
END IF
```

## EXPLICATION

Voir IF pour plus de détail.

L'instruction END IF est associée à la forme IF multilignes :

```
IF <condition> THEN
  ....
  ....
ELSEIF <condition> THEN
  ....
  ....
ELSE
  ....
  ....
END IF
```

## ERREUR

L'absence de cette instruction pour finir un bloc IF...THEN multilignes provoque une exécution anormale du programme pouvant conduire à une erreur détectée ou à des résultats erronés.

# END SELECT INSTRUCTION

## BUT

END SELECT marque la fin du dernier bloc CASE associé à une instruction SELECT.

## SYNTAXE

END SELECT

## EXPLICATION

Une instruction SELECT est suivie d'un ou plusieurs blocs CASE et le dernier doit être suivi de l'instruction END SELECT.

## EXEMPLE

Voir instruction CASE.

## ERREUR

L'erreur No 10752 est détectée si cette instruction est rencontrée alors que l'exécution ne se trouve pas à l'intérieur d'un bloc SELECT/CASE.



# EPS FONCTION

## BUT

Evaluer l'erreur maximale sur un nombre.

## SYNTAXE

EPS (<expression numérique>)

## EXPLICATION

Soit X l'expression numérique,

EPS(X) rend le maximum de (X-X',X''-X,Z)

où X'et X'' sont le prédécesseur et le successeur de X et Z la plus petite valeur positive représentable.

EPS(0) est donc la plus petite valeur positive représentable.

## EXEMPLE

```
PRINT EPS (4)
1.E-13
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# ERASE ALL COMMANDE/INSTRUCTION

## BUT

Efface toutes les variables du programme et les boucles et sous-programmes en cours et ferme tous les masques et fichiers en cours.

## SYNTAXE

ERASE ALL

## EXPLICATION

Toutes les valeurs précédentes. des variables sont effacées et le contexte d'exécution des boucles FOR/NEXT, DO/LOOP et SELECT/CASE et de sous-programmes GOSUB/RETURN est remis à vide. Tous les fichiers et masques sont fermés.

Une tentative d'utiliser NEXT, LOOP, RETURN, END SELECT ... après ERASE ALL provoquera la même erreur qu'en cas d'absence du FOR, DO, GOSUB ou SELECT associé.

Après ERASE ALL les tableaux peuvent être redimensionnés.

## EXEMPLE

```
a=1
ERASE ALL
PRINT a
 0
```

## ERREUR

Néant.



# EVALUATE FONCTION

## BUT

Cette fonction convertit en une valeur numérique l'expression arithmétique simple contenue dans une expression chaîne.

## SYNTAXE

EVALUATE ( <expression chaîne> )

L'expression chaîne contiendra l'expression simple sous la forme :

<nombre> [ <opérateur> <nombre> ]

L'opérateur pourra être "+" "-" "*" ou "/".

## EXPLICATION

L'expression chaîne est évaluée et les calculs effectués au fur et à mesure. Les espaces sont ignorés.

L'évaluation est interrompue si un caractère est non interprétable. Le résultat rendu est restitué tel qu'il est à ce stade de l'évaluation.

## EXEMPLE 1

```
100 INPUT PROMPT "Expression simple " : a$
110 PRINT EVALUATE ( a$ )
```

## EXEMPLE 2

```
100 a$="123.34"
110 b$="10.12"
120 PRINT EVALUATE ( a$ + "+" + b$ )
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne .

## REMARQUE

Voir la fonction VAL qui comme EVALUATE évalue une chaîne et rend une valeur numérique, mais qui n'accepte pas de calculs.



# EXECUTE INSTRUCTION

## BUT

EXECUTE permet d'exécuter une chaîne de caractères contenant une instruction.

## SYNTAXE

EXECUTE <expression chaîne>

## EXPLICATION

L'instruction EXECUTE permet d'exécuter une chaîne de caractères contenant une instruction ou une suite d'instructions exécutables séparées par ":".

La chaîne ne doit pas contenir de ligne numérotée, mais uniquement des instructions exécutables.

Par exemple :

```
EXECUTE "A=1 : b$=using$ (F$,x)"   est valide

EXECUTE "100 CASE IS 10"           est invalide
```

La chaîne à exécuter ne doit pas contenir de fonction EXECUTE, celle-ci ne permettant pas la récursivité.

```
EXECUTE " EXECUTE a$ "             est invalide
                                   et dangereux !
```

## EXEMPLE

```
100 PRINT "DONNEZ VOTRE FORMULE"
110 INPUT a$
120 EXECUTE " Y= " & a$
130 PRINT Y
```

## ERREURS

L'erreur No 9001 est détectée si l'EXECUTE tente de modifier le programme par l'introduction d'une ligne numérotée.

Une erreur de syntaxe est détectée si l'expression chaîne ne représente pas une instruction exécutable correcte.

Des erreurs d'exécution peuvent également être détectées : dépassement de capacité, indice sortant des limites d'un tableau, etc...

## REMARQUES

Pour l'exécution de formules très simples, voir EVALUATE.

Si l'instruction EXECUTE est utilisée en mode commande (mode immédiat), d'éventuelles autres instructions suivant EXECUTE sur la ligne ne seront pas exécutées. Il en est de même pour les EXECUTE imbriqués.

## CONSEILS

Cette instruction est très utile pour simplifier le paramétrage des applications. Une formule peut alors remplacer un nombre important de cas à prévoir.



# EXIT INSTRUCTION

## BUT

Cette instruction permet de sortir d'une boucle et de passer à l'instruction qui suit immédiatement la fin de la boucle.

## SYNTAXE

```
EXIT DO
ou
EXIT FOR
```

## EXPLICATION

L'instruction EXIT permet de sortir d'une boucle : boucle DO...LOOP ou boucle FOR...NEXT.
Il suffit que cette instruction se trouve à l'intérieur de la portée d'une boucle pour que son exécution provoque le branchement à l'instruction qui suit immédiatement la fin de la boucle. Une telle instruction évite l'utilisation d'un GOTO suivi d'un numéro de ligne ou d'une étiquette.

## EXEMPLE

```
100 FOR I=1 TO N
.
140     IF ABS(X) < .00001 THEN EXIT FOR
.
160 NEXT I
.
300 DO
.
350     IF ... THEN EXIT DO
...
400 LOOP
```

## ERREURS

L'erreur No 10702 est détectée si l'instruction EXIT DO est exécutée alors que le programme n'est pas dans une boucle DO....LOOP.

L'erreur No 10703 est détectée si l'instruction EXIT DO est exécutée et qu'aucun LOOP n'est trouvé.

L'erreur No 10903 est détectée si l'instruction EXIT FOR est exécutée alors que le programme n'est pas dans une boucle FOR ... NEXT.

L'erreur No 10904 est détectée si l'instruction EXIT FOR est exécutée et qu'aucun NEXT n'est trouvé.

## REMARQUE

Il est déconseillé d'utiliser l'instruction GOTO pour sortir définitivement d'une boucle DO ou d'une boucle FOR car MEMBASIC garde des informations sur la boucle en cours d'exécution et la sortie d'une boucle au moyen de GOTO provoque une perte de place en mémoire.



# EXKEY FONCTION

## BUT

Rend le code de la touche clavier enfoncée par l'utilisateur pour sortir d'un masque MEMSCREEN durant une saisie.

## SYNTAXE

EXKEY

## EXPLICATION

Pendant une saisie par masque MEMSCREEN, l'utilisateur peut finir la saisie de différentes façons. EXKEY est affecté en conséquence.

Si l'utilisateur abandonne et enfonce ESC :

EXKEY rend -1

Si l'utilisateur sort zone par "RETOUR CHARIOT" sur la dernière zone ou par les flèches haute et basse ou par les touches PgUp et PgDn :

EXKEY rend 0

Dans ce cas, la fonction EXWAY permettra de distinguer les différents types de sortie (voir EXWAY).

Si l'utilisateur sort par l'une des touches de fonction POMME 1 à POMME 9 :

EXKEY rend une valeur de 1 à 9 qui est le numéro de la touche.

Il est impossible de sortir par une touche pour laquelle aucun texte n'est visible.

Les saisies par masques sont décrites dans la section III (MEMSCREEN).

## ERREUR

Néant.

## REMARQUES

Voir l'instruction FKEY qui permet de définir le texte des touches de fonctions.

Voir également la fonction EXZONE qui permet de savoir sur quelle zone s'est effectuée la sortie.



# EXLINE FONCTION

## BUT

EXLINE est une fonction sans paramètre qui donne le numéro de la ligne où s'est produit l'erreur.

## SYNTAXE

EXLINE

## EXPLICATION

Quand une anomalie est détectée, le module de traitement de l'anomalie peut connaître le numéro de la ligne où s'est produite cette anomalie en faisant référence à la fonction EXLINE.

Cette fonction ne peut pas être utilisée si le programme a été sauvegardé sous une forme protégée. Dans ce cas, le résultat rendu par la fonction EXLINE est imprévisible.

## EXEMPLE

```
.
1000 IF EXLINE = 100 THEN ...
.
.
.
1100 RETRY
```

## ERREUR

Néant.

## REMARQUE

Voir la fonction EXTYPE qui donne le numéro de l'erreur détectée.

# EXP FONCTION

## BUT

Calculer une fonction exponentielle.

## SYNTAXE

EXP (<expression numérique>)

## EXPLICATION

Soit X l'expression numérique,

EXP(X) donne le nombre e élevé à la puissance X.

e est la base des logarithmes naturels.

## EXEMPLE

```
PRINT EXP (0)
1
```

## ERREURS

L'erreur No 1002 est détectée s'il y a dépassement de capacité dans l'évaluation.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# EXTEXT$ FONCTION

## BUT

Cette fonction donne le contenu du message d'erreur.

## SYNTAXE

EXTEXT$(<numéro d'erreur>)

## EXPLICATION

Quand une anomalie provoque le branchement vers la routine d'exception, cette routine peut obtenir le contenu du message d'erreur correspondant en faisant référence à la fonction EXTEXT$.

Cette version française de MEMBASIC permet d'obtenir le message en Français.

## EXEMPLE

```
510 PRINT EXTEXT$(EXTYPE) ! ARRET DU PROGRAMME
520 END
```

## ERREUR

Néant.

## REMARQUE

Pour des raisons d'encombrement mémoire, le fichier des messages d'erreurs est stocké sur une mémoire auxiliaire et peut ne pas être présent lors de l'exécution. Dans ce cas, EXTEXT$ rend une chaîne vide.

# EXTYPE FONCTION

## BUT

EXTYPE est une fonction sans paramètre qui donne le numéro de l'anomalie détectée.

## SYNTAXE

EXTYPE

## EXPLICATION

Quand une anomalie est détectée, le module de traitement de l'anomalie peut connaître le numéro d'anomalie en faisant référence à la fonction EXTYPE.

## EXEMPLE

```
100 a$="B ne doit pas être nul"
110 INPUT PROMPT "entrez A et B " : a,b
120 WHEN EXCEPTION GOTO 300
130 c=a/b
140 PRINT "A/B=";c
150 GOTO 100
...
300 IF EXTYPE=3001 THEN PRINT a$
...
```

## ERREUR

Néant.



# EXWAY FONCTION

## BUT

Cette fonction permet, en cas de sortie de la saisie d'un masque par une touche de déplacement, de connaître cette touche.

## SYNTAXE

EXWAY

## EXPLICATION

Lors de l'utilisation des instructions LET "INPUT" ou LET "KEYBOARD" de MEMSCREEN, il est possible de terminer la saisie par un clic souris ou par l'enfoncement de l'une des touches suivantes :

- touches de fonction de POMME 1 à POMME 9
- ESC
- flèche haute ou flèche basse
- RETOUR CHARIOT
- POMME ↑ ou POMME ↓

La fonction EXKEY renseigne partiellement sur le mode de sortie : elle rend une valeur comprise entre 1 et 9 si une touche de fonction a été enfoncée, -1 pour ESC, 0 dans tous les autres cas.

EXWAY complète l'information obtenue dans ce dernier cas (EXKEY=0). Elle rend les valeurs suivantes :

* 0 pour RETOUR CHARIOT
* 1 pour la flèche basse
* -1 pour la flèche haute
* 2 quand on clic en bas, et à droite de la zone de saisie, ou POMME ↑,
* -2 quand on clic en haut, et à gauche de la zone de saisie ou POMME ↓.

## EXEMPLE

```
LET "Input,M,0"
IF EXKEY = 0 THEN
     IF EXWAY < 0 THEN 'haut'
     IF EXWAY > 0 THEN 'bas'
END IF
```

## ERREUR

Néant.

## REMARQUE

Les instructions LET "INPUT" et LET "KEYBOARD" permettent une option /K qui interdit à l'utilisateur de sortir par toutes les touches rendant le code EXKEY = 0. La fonction EXWAY n'a aucune signification dans ce cas.



# EXZONE FONCTION

## BUT

A l'issue d'une saisie de données par masque MEMSCREEN, EXZONE rend le numéro de zone de saisie où la sortie s'est effectuée.

## SYNTAXE

EXZONE

## EXPLICATION

L'utilisateur peut sortir d'une saisie dans un masque MEMSCREEN à tout moment. La fonction EXZONE donne le numéro de la zone EN ENTREE qui était saisie à l'instant de la sortie. L'instruction MEMSCREEN LET "INPUT..." permet de reprendre la saisie à un endroit quelconque du masque, en particulier à la zone où s'est terminée la dernière saisie, de numéro EXZONE.

Les saisies par masques sont décrites dans la section III (MEMSCREEN).

## ERREUR

Néant.

## REMARQUE

La fonction EXKEY permet de savoir quelle touche a provoqué la sortie.

# FILESIZE FONCTION

## BUT

Rend le cumul des tailles des fichiers correspondants au critère.

## SYNTAXE

FILESIZE (<expression chaîne>)

## EXPLICATION

L'expression chaîne représente le critère de sélection des fichiers concernés par le calcul.
Cette fonction rend en résultat une valeur de type numérique, correspondant au nombre d'octets utilisés par les fichiers spécifiés.

L'expression chaîne peut contenir les caractères spéciaux ? et *.
Un ? dans un nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans un nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

Si le nom du fichier est un nom de répertoire, c'est le répertoire complet qui sera pris en compte pour le calcul.

## EXEMPLE

```
500 IF FILESIZE ("COMPTES.MF*")>500000 THEN
510             GOSUB 'File Copy'
520 END IF
```

## ERREURS

L'erreur No 28010 est détectée si aucun fichier ne correspond au critère.

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.



## CONSEILS

Cette fonction peut servir pour savoir s'il est utile d'exécuter LET "#GARBAGE..." pour compacter un fichier. Elle peut également servir à estimer le nombre de disquettes nécessaires à une sauvegarde (voir MEMBACKUP).

## REMARQUE

La taille indiquée ne tient pas compte de la place perdue par le système d'exploitation pour ranger les fichiers (par exemple, un fichier de 10 octets prendra au moins 512 octets puisque c'est la taille d'un secteur).
Il faut donc prendre des précautions avant d'affirmer, après utilisation de FILESIZE, qu'un ensemble de fichiers tient sur une disquette !.

# FIND COMMANDE

## BUT

Cette commande permet de retrouver une suite de caractères dans un programme.

## SYNTAXE

FIND <type> [<région>] "<caractères>"

<type> peut être '&', '<', '>' ou 'vide'.

<région> représente la partie du programme dans laquelle on effectue la recherche.

<caractères> est la suite de caractères à rechercher à l'exception du caractère '"' (guillemet).
Le caractère '?' peut remplacer n'importe quel caractère, y compris le guillemet.

## EXPLICATION

Cette commande permet d'afficher à l'écran les lignes du programme contenant la suite de caractères recherchée suivant le type de recherche et la région.

<région> caractérise la partie de programme concernée par la commande.
Sa syntaxe est :

<lmin> TO <lmax>

où <lmin> et <lmax> sont des numéros de ligne ou des étiquettes. Ils peuvent également prendre des valeurs particulières :

- '+' pour le numéro de ligne maximum,
- '-' pour le numéro de ligne minimum,
- '*' pour la dernière ligne validée par RETOUR CHARIOT.

<type> peut prendre les valeurs suivantes :

- '&' permet d'afficher à l'écran toutes les lignes contenant la suite de caractères dans la région de programme spécifiée,
- '<' permet d'afficher à l'écran les lignes contenant la suite de caractères dans la page précédente du programme,



- '>' permet d'afficher à l'écran les lignes contenant la suite de caractères dans la page suivante du programme,
- 'vide' permet d'afficher à l'écran la première ligne du programme contenant la suite de caractères.

## EXEMPLE

```
10 aa = 1
15 tester$ = "oui"
20 ab = 2
30 'etiq1' : ac = 3
40 ad = 4

FIND "1"
10 aa = 1

FIND & "1"
10 aa = 1
30 'etiq1' : ac = 3

FIND - TO 'etiq1' "a? ="
10 aa = 1
20 ab = 2
30 'etiq1' : ac = 3
```

## ERREUR

Néant.

# FKEY INSTRUCTION

## BUT

Cette instruction permet de modifier la signification des touches de fonction POMME 1 à POMME 9, affichées en bas de l'écran.

## SYNTAXE

FKEY <chaîne vide>
ou
FKEY <expression chaîne>

où <expression chaîne> indique la nouvelle signification des touches de fonction sous la forme :

<index><caractères> [+<index><caractères>] ...
ou
[+<index><caractères>] [-<index>] ...

## EXPLICATION

FKEY permet de donner une signification aux neuf touches de fonction POMME 1 à POMME 9. A chaque touche de fonction, on associe un texte de 0 à 6 caractères qui sera affiché sur la 25ème ligne de l'écran.

Lors de la saisie d'un masque MEMSCREEN, l'utilisateur pourra sortir en enfonçant l'une des touches de fonction parmi celles qui sont affichées sur la 25ème ligne. La fonction EXKEY permettra alors de connaître la touche sélectionnée.

La première forme FKEY <chaîne vide> provoque l'effacement de la 25ème ligne de l'écran à l'exception du message de la touche POMME 0 puis la désactivation des touches POMME 1 à POMME 9. La touche POMME 0 reste toujours active pour les fichiers d'aides.

La seconde forme permet de redéfinir le message associé à chaque touche de fonction et d'activer les touches correspondantes.

- Si aucun des signes + ou - n'est présent en tête de liste, seules les touches dont l'index figure dans la liste prennent une signification, La 25ème ligne est donc entièrement redéfinie.

- Si le signe + (ou le signe -) est présent, la 25ème ligne sera seulement modifiée. Le signe + indique que la touche spécifiée doit être ajoutée à la liste ou modifiée si elle existe déjà. Le signe - indique que la touche doit être supprimée de la liste. Dans ce cas il suffira d'indiquer le numéro de la touche.

Chaque message est représenté par les caractères qui suivent l'index dont 6 au maximum sont pris en compte.

## EXEMPLE

```
100 FKEY "2SUIVNT+1PRECDNT"
110 FKEY "+3LIRE+4MODIF"
120 FKEY "-2+5CREER"
.
.
500 FKEY ""
```

L'instruction FKEY en ligne 100 donne une signification aux touches de fonction POMME 2 et POMME 1 et ôte toute signification aux autres touches.
L'instruction FKEY en ligne 110 ajoute une signification aux touches fonctions POMME 3 et POMME 4.
L'instruction FKEY en ligne 120 supprime la touche de fonction POMME 2 et ajoute une signification à la touche de fonction POMME 5.
L'instruction FKEY en ligne 500 annule le rôle de l'ensemble des touches fonctions POMME 1 à POMME 9.

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

## REMARQUE

Quelle que soit la forme employée, la fonction POMME 0 sera toujours affichée avec le texte "AIDE". Le chapitre 1 donne des informations sur les fichiers d'aides.

La fonction EXKEY permet de connaître le code de la touche de fonction utilisée.



# FOR INSTRUCTION

## BUT

Construction d'une boucle dont l'exécution est "contrôlée" par la valeur d'une variable.

## SYNTAXE

FOR <variable>=<expres> TO <expres> [STEP <expres>]

où <expres> représente une expression numérique et <variable> l'identificateur d'une variable simple numérique.

## EXPLICATION

L'instruction FOR, associée à l'instruction NEXT, permet de construire des boucles. La variable numérique qui suit immédiatement le mot FOR est appelée variable de contrôle de la boucle. Elle prendra des valeurs successives à partir d'une valeur initiale suivant un pas. Les deux ou trois expressions numériques suivantes sont appelées paramètres de la boucle, le premier représente la valeur initiale, le second la valeur finale et le troisième, optionnel, le pas ou incrément. Si l'option STEP <expression numérique> est absente, alors la valeur du pas est égale à 1. Toutes les expressions sont évaluées avant que la première des expressions ne soit affectée à la variable de contrôle, ce qui signifie que la variable de contrôle conserve sa valeur d'origine pendant l'évaluation des trois expressions.

Si la valeur initiale est inférieure à la valeur finale et que le pas est positif, alors le contenu des instructions de la boucle est exécuté au moins une fois.

Si la valeur initiale est plus grande que la valeur finale et que le pas est négatif, alors le contenu des instructions de la boucle est exécuté au moins une fois.

Si aucune de ces deux conditions n'est satisfaite, alors l'ensemble des instructions de la boucle n'est pas du tout exécuté, il y a passage immédiat à l'instruction qui suit l'instruction NEXT.

Lorsque l'on arrive à l'instruction NEXT, la variable de contrôle est augmentée de la valeur du pas, puis il y a retour aux conditions de test afin de savoir si les instructions de la boucle seront exécutées une fois supplémentaire ou non.

Attention : la valeur finale est évaluée une fois pour toutes en début dxe boucle. Elle n'est pas réévaluées à chaque passage sur un NEXT.

Des boucles FOR/NEXT peuvent être emboîtées mais pas enchevêtrées :

| Autorisé: | Interdit: |
|---|---|
| FOR I=... | FOR I=... |
| FOR J=... | FOR J=... |
| NEXT J | NEXT I |
| NEXT I | NEXT J |

Les instructions EXIT FOR et SKIP FOR peuvent modifier le comportement normal de la boucle FOR/NEXT.

EXIT FOR permet de sortir de la boucle avant que la comparaison de la variable de boucle avec la valeur finale ne soit fausse. SKIP FOR permet de passer à l'addition du pas à la variable de contrôle et au retour en haut de boucle avant que le déroulement normal du programme n'amène à l'instruction NEXT.

## EXEMPLE

```
 90 DIM a(10 TO 20)
100 ! Initialisation
110 FOR i=10 TO 20
120   A(i)=1
130 NEXT i
```

## ERREUR

L'erreur No 10901 est détectée si une instruction NEXT est rencontrée lors de l'exécution sans que l'instruction FOR correspondante ait été exécutée.



## REMARQUE

Lorsque la boucle se termine de façon normale (sans EXIT FOR), la variable de contrôle contient la première valeur qui a rendue fausse la comparaison avec la valeur finale. Dans l'exemple ci-dessus, en sortie de boucle, la variable I vaut 21.

# FP FONCTION

## BUT

FP donne la partie décimale d'une valeur numérique.

## SYNTAXE

FP (<expression numérique>)

## EXPLICATION

FP renvoie la partie décimale de la valeur numérique obtenue par l'évaluation du paramètre.

## EXEMPLE

```
100 INPUT x
110 y = FP(x)
120 IF y <> 0 THEN
130           PRINT "X N'EST PAS ENTIER";
140           PRINT "SA PARTIE DECIMALE VAUT "; y
150 END IF
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas de type numérique.



# FREE FONCTION

## BUT

Donne des informations sur la place disponible en mémoire centrale ou sur disque.

## SYNTAXE

FREE

ou

FREE (<expression chaîne>)

où <expression chaîne> indique le disque concerné.

## EXPLICATION

FREE donne la place libre en mémoire centrale en octets.

FREE ("") donne, en octets, la plus petite place mémoire restée libre depuis le chargement de MEMSOFT.

FREE ("<disque>") donne la place restant disponible sur le disque exprimée en octets.
"<expression chaîne>" est une chaîne dont le premier caractère est seul significatif et spécifie le disque concerné ( A,B,C...).

Le nombre d'octets libres est donné à 512 près.

## EXEMPLES

```
100 PRINT "Il reste ";
110 PRINT FREE(CURDRIVE$);" octets sur le disque "
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

L'erreur No 28900 est détectée si le disque spécifié est invalide.

# GET KEYBOARD INSTRUCTION

## BUT

Attente d'une touche frappée au clavier.

## SYNTAXE

GET KEYBOARD <variable>

## EXPLICATION

Si la variable est de type chaîne, cette instruction attend qu'une touche correspondant à un caractère du jeu supporté soit appuyée.

La variable reçoit alors le caractère correspondant.

Si la variable est numérique, cette instruction accepte toutes les touches du clavier, et rend la valeur du code correspondant. Si l'utilisateur appuie sur une touche ne correspondant pas à un caractère du jeu supporté, la variable reçoit une valeur supérieure à 256 (voir les codes clavier donnés par le constructeur pour les touches spéciales).

GET KEYBOARD fait clignoter un curseur dans la fenêtre d'exécution, qui devient donc visible si elle ne l'était pas.

## ERREUR

Néant.

## REMARQUE

La forme GET KEYBOARD <variable numérique> rend un code qui peut dépendre de la machine. Elle n'est donc à utiliser qu'en dernier recours. La transportabilité de cette forme n'est pas assurée.



# GOSUB INSTRUCTION

## BUT

GOSUB effectue un branchement vers un sous-programme interne.

## SYNTAXE

GOSUB <numéro de ligne>
GOSUB <étiquette>

## EXPLICATION

L'exécution de l'instruction GOSUB provoque le branchement vers le sous-programme interne qui commence à la ligne ou l'étiquette indiquée dans l'appel.

L'instruction RETURN permet le retour à l'instruction qui suit immédiatement l'instruction GOSUB qui a provoqué le branchement.

Un sous-programme peut avoir plusieurs points d'entrée. Il peut également contenir plusieurs instructions RETURN.

Un sous-programme peut appeler un autre sous-programme, il peut y avoir une cascade d'appels successifs de sous-programmes. La limitation n'est pas dans le langage lui-même mais dans la place mémoire.

Si un GOSUB provoque le branchement vers une ligne contenant uniquement un commentaire, alors l'exécution se poursuit à la ligne suivante.

Un sous-programme interne peut être placé n'importe où dans le corps du programme. Cependant il est souhaitable pour rendre la mise au point plus facile et pour améliorer la lisibilité du programme, de le séparer des instructions du programme principal.

L'appel récursif de sous-programmes au moyen de GOSUB est possible.

## EXEMPLE

```
1000 IF ordre$="I" THEN GOSUB 'Imprime'
```

## ERREUR

L'erreur No 10600 est détectée si GOSUB fait référence à un numéro de ligne inexistant ou à une étiquette inexistante.

## CONSEIL

Pour permettre une meilleure lisibilité des programmes, il est préférable de bien séparer les instructions du programme principal de celles des sous-programmes et d'utiliser des étiquettes plutôt que des numéros de lignes.



# GOTO INSTRUCTION

## BUT

Cette instruction effectue un branchement à la ligne indiquée.

## SYNTAXE

```
      <numéro de ligne>
GOTO
      <étiquette>
```

## EXPLICATION

L'exécution de l'instruction GOTO provoque le branchement vers la ligne spécifiée par son numéro ou par une étiquette.

Contrairement au GOSUB, aucun mécanisme ne permet de revenir à l'instruction de départ.

## EXEMPLE

```
100 IF reponse$="FIN" THEN GOTO 'fin'
120 GOTO 'Suite'
130 GOTO 200
140 ...
150 'Suite'
```

## ERREUR

L'erreur No 10600 est détectée si le numéro de ligne ou l'étiquette n'existe pas dans l'unité de programme contenant l'instruction.

## REMARQUES

En mode commande, GOTO relance l'exécution du programme à partir de la ligne spécifiée sans que les variables déjà affectées ne soient effacées.

Voir également ON ... GOTO qui permet un branchement suivant une liste en fonction d'une valeur d'index.

# HELP INSTRUCTION

## BUT

Cette instruction permet de définir le fichier d'aide qui sera pris en compte au prochain enfoncement de la touche AIDE.

## SYNTAXE

HELP <expression chaîne>

où <expression chaîne> comporte le nom du fichier d'aide. Ce nom peut comporter des spécifications de disque et de répertoire.

## EXPLICATION

L'instruction HELP permet de définir un fichier d'aide explicite.
Le fichier d'aide dont le nom a été défini sera affiché à l'enfoncement de la touche AIDE. Si le fichier n'est pas trouvé, il sera cherché dans les répertoires indiqués dans l'instruction PATH en cours. S'il n'est toujours pas trouvé, on recherche les fichiers d'aide associés au masque ou au programme (voir paragraphe 1.6 de la section III "MEMSCREEN").

Pour ne plus avoir d'aide explicite, utiliser la commande HELP avec une chaîne vide.

## EXEMPLE

```
HELP "/REP/ESSAI.HLP"
```

## ERREUR

Néant.

## REMARQUE

Voir également la fonction HELP$ qui indique le nom du fichier d'aide en cours.



# HELP$ FONCTION

## BUT

Cette fonction sans paramètre rend le nom du fichier d'aide explicite en cours.

## SYNTAXE

HELP$

## EXPLICATION

Cette fonction rend le nom du fichier d'aide qui a été défini par l'instruction HELP.

## EXEMPLE

```
PRINT HELP$
/REP/ESSAI.HLP
```

## ERREUR

Néant.

## REMARQUES

Voir l'instruction HELP qui permet de définir le nom du fichier d'aide en cours.

# HTAB FONCTION

## BUT

La fonction sans paramètre HTAB donne le numéro de la colonne, comptée à partir du bord gauche de la fenêtre d'exécution sur laquelle se trouve le curseur.

## SYNTAXE

HTAB

## EXPLICATION

HTAB donne le numéro de la colonne sur laquelle se trouve le curseur.

## EXEMPLE

```
100 ! passe à la ligne à la colonne 60
110 IF HTAB>=60 THEN PRINT
```

## ERREUR

Néant.

## REMARQUES

Voir la fonction VTAB qui donne la colonne du curseur.

Voir l'instruction HTAB qui permet de positionner le curseur sur une colonne donnée.



# HTAB INSTRUCTION

## BUT

L'instruction HTAB permet de positionner le curseur sur une colonne donnée.

## SYNTAXE

HTAB <index>

## EXPLICATION

L'expression numérique est évaluée et si nécessaire arrondie à l'entier le plus proche. Cette valeur est utilisée pour positionner le curseur dans la fenêtre d'exécution.

## EXEMPLE

```
100 HTAB 30
```

## ERREURS

L'erreur No 25101 est détectée si le paramètre n'est pas numérique.

L'erreur No 4000 est détectée si le paramètre de l'instruction HTAB est hors limites.

L'erreur No 1902 est détectée si l'index dépasse la valeur entière maximale (32767).

## REMARQUES

Voir la fonction HTAB qui donne la colonne du curseur.

Voir l'instruction VTAB pour positionner le curseur sur une ligne donnée.

# IF INSTRUCTION

## BUT

Cette instruction permet d'effectuer soit des branchements conditionnels, soit d'exécuter des instructions selon qu'une condition est satisfaite ou non.

## SYNTAXE

L'instruction IF peut prendre deux formes :

- La forme très connue suivante :

```
IF <relation> THEN <clause> ELSE <clause>
```

où <clause> peut prendre l'une des formes suivantes :

- <instruction> [ : <instruction> ] ...
- <numéro de ligne>
- <étiquette>

Plusieurs IF ... THEN ... ELSE peuvent être imbriqués.

- La forme structurée suivante :

```
IF <expression de relation> THEN

     Bloc d'instructions

ELSE

     Bloc d'instructions

END IF
```

Dans ce cas un bloc d'instructions est constitué par une ou plusieurs lignes d'instructions exécutables. Ce bloc d'instructions peut inclure d'autres blocs IF.

## EXPLICATION

Le fonctionnement général de cette instruction est le suivant:



L'expression de relation est d'abord évaluée. Si elle donne un résultat booléen VRAI alors la ou les instructions qui suivent le mot-clé THEN sont exécutées. Ensuite il y a passage à l'instruction suivante. Si l'expression de relation donne la valeur booléenne FAUX, alors la ou les instructions qui suivent le mot-clé THEN ne sont pas exécutées mais deux cas se présentent:

- Ou bien il n'y a pas l'option ELSE et dans ce cas il y a passage automatique à l'instruction suivante.

- Ou bien l'option ELSE existe et dans ce cas la ou les instructions suivant immédiatement le mot-clé ELSE sont exécutées.

Remarquons que la seconde forme, qui permet d'écrire une instruction IF sur plusieurs lignes logiques, est préférable à la première dès l'instant où l'on souhaite soit placer de nombreuses instructions après THEN ou ELSE, soit encore lorsque un bloc THEN ou un bloc ELSE doit à son tour contenir des instructions de contrôle du type instruction IF ou DO LOOP ou SELECT CASE.

Le mot-clé ELSEIF permet d'emboîter une instruction IF à l'intérieur d'un bloc ELSE, selon le schéma suivant :

```
IF <condition 1> THEN
     ...
     séquence à exécuter si <condition 1> VRAI
     ...
ELSEIF <condition 2> THEN
     ...
     séquence à exécuter si <condition 2> VRAI
     ...
ELSEIF <condition 3> THEN
     ...
     séquence à exécuter si <condition 3> VRAI
     ...
ELSE
     ...
     séquence D à exécuter dans les autres cas
     ...
END IF
```

## EXEMPLE 1

```
100 IF cle$<"A" THEN GOTO 'numer' ELSE GOTO 'alpha'
```

## EXEMPLE 2

```
100 IF cle$ < "A" THEN
110                     GOTO 'numer'
120             ELSE
130                     GOTO 'alpha'
140             END IF
```

## ERREURS

L'erreur No 10802 est détectée si END IF n'est pas trouvé après ELSE ou ELSEIF.

L'erreur No 10801 est détectée si END IF n'est pas trouvé après un IF...THEN multilignes.

## REMARQUES

L'expression de relation peut prendre plusieurs formes. Elle peut être une expression booléenne traditionnelle, mais aussi une expression numérique quelconque considérée comme VRAIE si non nulle, fausse si nulle; Elle pourra même être une expression chaîne. Dans ce cas, elle sera considérée comme VRAIE si de longueur non nulle.
Par exemple, les formes suivantes sont équivalentes:

```
IF LEN(a$) > 0 THEN
IF LEN(a$) THEN
IF a$ THEN
```

## CONSEIL

Si vous utilisez IF ... THEN ... ELSE ... END IF sur plusieurs lignes, il est souhaitable d'indenter les lignes de façon à permettre une meilleure lisibilité du programme.



# IMAGE INSTRUCTION

## BUT

L'instruction IMAGE permet de définir un format que pourra utiliser une instruction PRINT USING.

## SYNTAXE

IMAGE :<format>

La syntaxe de <format> est décrite à l'instruction PRINT USING.

La ligne MEMBASIC qui contient l'instruction IMAGE ne doit contenir aucune autre instruction. Elle peut par contre commencer par une étiquette qui facilitera son identification.

## EXPLICATION

Si l'exécution du programme l'amène sur une ligne contenant une instruction IMAGE, le programme passera à l'instruction suivante comme dans le cas de l'instruction DATA.

L'instruction PRINT USING utilisant le format contenu dans la ligne IMAGE pourra être située aussi bien après qu'avant la ligne IMAGE.

## EXEMPLES

```
100 IMAGE :Ceci est la page numéro ### du document
200 'prix' : IMAGE :Le prix est ********.## F
```

## ERREUR

Les erreurs sont détectées lors de l'exécution de PRINT USING.

# INPUT INSTRUCTION

## BUT

Permet de lire des données introduites au clavier.

## SYNTAXE

```
INPUT  <liste variables>
  ou
INPUT PROMPT <expression chaîne>: <liste variables>

avec <liste variables> = <var> [ , <var> ... ]
```

## EXPLICATION

L'exécution de l'instruction INPUT provoque :

- si PROMPT est présent, l'affichage de la valeur de l'expression chaîne suivant PROMPT,
- la suspension de l'exécution du programme jusqu'à ce qu'une réponse satisfaisante soit donnée à la liste de variables.

Le nombre de données introduites doit être égal au nombre de variables de la liste. En outre, il doit y avoir correspondance entre type des données et type des variables à affecter.

Si ces deux conditions ne sont pas satisfaites, l'entrée est annulée et l'utilisateur est invité à la recommencer à partir du début.

Une donnée numérique peut être affectée aussi bien à une variable numérique qu'à une variable chaîne.

Par contre une donnée de type chaîne ne peut être affectée qu'à une variable chaîne.

Si l'ensemble de l'entrée de données est acceptable alors chaque donnée est affectée à chaque variable dans l'ordre de gauche à droite de la liste. Les indices de la liste de variables sont évalués juste avant l'affectation.



Ainsi :

```
100 DIM A(3)
110 I=2
120 INPUT I,A(I)

run
3,5  (valeurs entrées)
```

entraîne l'affectation de 3 à I, puis de 5 à A(3) et non à A(2)

## EXEMPLE

```
100 INPUT PROMPT "QUEL EST VOTRE NOM":n$
 .
 .
210 INPUT i, j, a(i,j)
```

## ERREUR

L'exécution de l'instruction INPUT peut provoquer des erreurs; dans chaque cas, MEMBASIC demande de recommencer l'entrée des données à partir du début.

# INPUT # INSTRUCTION

## BUT

Permet de lire des données depuis un fichier PRODOS ou un port de communication série.

## SYNTAXE

INPUT #<numéro fichier>, <liste variables>

<numéro fichier> est une expression numérique entière donnant le numéro du fichier à utiliser.

<liste variables> est la liste des variables à affecter de forme : <variable> [ , <variable> ... ]

## EXPLICATION

Le système tente de lire sur le fichier (PRODOS ou communication) un nombre de données égal au nombre de variables de la liste. En outre, il doit y avoir correspondance entre type des données et type des variables à affecter. Les données lues sont supposées sous forme caractère et non sous forme codée : Une donnée numérique peut être affectée aussi bien à une variable numérique qu'à une variable chaîne. Par contre une donnée de type chaîne ne peut être affectée qu'à une variable chaîne.

Le système reconnaît comme séparateur de données les caractères "," et "fin de ligne" codé par la suite des deux caractères de code 13 (CR) et 10 (LF). Si une donnée chaîne est entre guillemets, les "," situées entre les guillemets ne sont pas considérées comme séparateurs. Les espaces à gauche et à droite ne sont pas supprimés (s'ils doivent l'être, utiliser LTRIM$ et RTRIM$).

Les données sont affectées à chaque variable dans l'ordre de gauche à droite de la liste.

Par exemple:

```
100 DIM a(3)
110 i=2
120 INPUT #1,i,a(i)
```



entraîne l'affectation de la première donnée lue à i puis de la seconde à a(<valeur lue pour i>).

Si la fin de fichier est rencontrée alors que toutes les variables n'ont pas été affectées, la fonction STATUS rendra 255 et les variables pas encore affectées conservent leurs valeurs précédentes. Si la lecture se termine normalement, STATUS rendra 0.

## EXEMPLE

```
100       OPEN "R",1,"amis.fic",30
110 'autre':
120       INPUT PROMPT "numéro d'enregistrement ":n
130       POINTER #1,n
140       INPUT #1,nom$,prenom$,age,adresse$
150       PRINT nom$,prenom$,age,adresse$
160       GOTO 'autre'
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier ouvert.

L'erreur No 28514 est détectée si le mode d'ouverture du fichier n'autorise pas la lecture.

L'erreur No 25101 est détectée si le numéro de fichier n'est pas une expression numérique.

L'erreur No 28518 est détectée si une donnée est trop longue.

## REMARQUES

Voir aussi LINE INPUT # et INPUT$ qui sont d'autres façons de lire des données.

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications séries.

# INPUT$ FONCTION

## BUT

Permet de lire un nombre exact d'octets depuis un fichier PRODOS ou un port de communication série.

## SYNTAXE

INPUT$ ( <nombre d'octets> ,<numéro de fichier> )

<nombre d'octets> est une expression numérique entière dont le résultat doit être entre 1 et 255.

<numéro de fichier> est une expression numérique entière donnant le numéro du fichier à utiliser.

## EXPLICATION

Le système tente de lire sur le fichier (PRODOS ou communication) le nombre d'octets demandé. Le résultat rendu est une expression chaîne.

Cette méthode n'est à conseiller que lorsque INPUT # et LINE INPUT # ne suffisent pas.

Si la fin de fichier est rencontrée alors que toutes les variables n'ont pas été affectées, la fonction STATUS rendra 255 et les variables pas encore affectées conservent leurs valeurs précédentes. Si la lecture se termine normalement, STATUS rendra 0.

## EXEMPLE

```
100 OPEN "COM2", 1     ! ouvre un port série
110 PRINT INPUT$(10,1) ! lit 10 octets
120 CLOSE
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier ouvert.

L'erreur No 28514 est détectée si le mode d'ouverture du fichier n'autorise pas la lecture.

L'erreur No 25101 est détectée si le numéro de fichier n'est pas une expression numérique.



Une erreur de 28559 à 28570 est détectée en cas d'erreur de transmission.

## REMARQUES

Voir aussi LINE INPUT # et INPUT # qui sont d'autres moyens de lire des données.

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications séries.

# INT FONCTION

## BUT

Cette fonction donne un résultat numérique donnant la plus grande valeur entière inférieure ou égale à celle de l'expression numérique donnée comme paramètre.

## SYNTAXE

INT (<expression numérique> [, index ] )

## EXPLICATION

Si le deuxième paramètre est précisé, le résultat est la plus grande valeur ayant le nombre de décimales égal à l'index et inférieure ou égale au paramètre.

INT (x,n) est donc équivalent à :

INT (x*10^n) / 10^n

## EXEMPLE

```
100 A = 3.9
110 B = -3.1
120 PRINT INT(A); INT(B)
RUN
 3   -4
```

## REMARQUE

Faire attention que le calcul est fait selon l'ordre des mathématiques et que, lorsqu'une expression est négative, en prendre la partie entière, ne signifie pas supprimer ses décimales. Pour d'autres formes d'arrondi voir les fonctions CEIL, ROUND et TRUNCATE.

## ERREURS

L'erreur No 25101 est détectée si l'expression ou l'index n'est pas de type numérique.

L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.



# IP FONCTION

## BUT

IP donne la partie entière du paramètre.

## SYNTAXE

IP (<expression numérique>)

## EXPLICATION

IP évalue la valeur numérique du paramètre et restitue sa partie entière (accompagnée du signe).

```
IP ( 3.9) donne  3
IP ( 3.1) donne  3
IP (-3.1) donne -3
IP (-3.9) donne -3
```

IP(X) est équivalent à SGN(X) * INT(ABS(X))

## EXEMPLE

```
100 INPUT x
200 IF x = IP(x) THEN PRINT x; "est entier "
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas de type numérique.

# KILL FONCTION

## BUT

KILL permet de détruire un fichier sur disque.

## SYNTAXE

KILL <spécification de fichier>

<spécification de fichier> est une expression chaîne donnant le nom du fichier à détruire, et éventuellement son disque et son répertoire.

## EXPLICATION

Le fichier spécifié est supprimé de son répertoire.

Il ne faut pas oublier de préciser le suffixe du fichier. Par exemple, un masque MEMSCREEN créé sans suffixe devra néanmoins être indiqué sous sa forme complète (avec le suffixe .MSK) pour être détruit par KILL. Le chapitre 1.18 donne les suffixes implicites qui peuvent être rencontrés.

On peut détruire plusieurs fichiers en même temps en utilisant dans le nom du fichier les caractères spéciaux ? et *.
Un ? dans un nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans un nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

Si le nom du fichier est un nom de répertoire, ce sont tous les fichiers de ce répertoire qui sont supprimés. Le répertoire, lui, reste existant (il faut utiliser l'instruction RMDIR pour le supprimer).

En mode immédiat, les fichiers détruits sont listés dans la fenêtre en cours et une confirmation est demandée pour toute destruction de répertoire complet.

Si le fichier que l'on désire détruire n'existe pas, la fonction STATUS rendra 10. Si la destruction se déroule normalement, STATUS rendra 0.



## EXEMPLE

```
KILL "tempo.tst"
KILL "/REP1/*.*"
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

## REMARQUES

L'instruction KILL peut être utilisée pour détruire des fichiers MEMFILE, mais il faut pour cela détruire les DEUX fichiers de suffixes .MFR et .MFK.

Par exemple:

```
100 ! Détruisons le fichier MEMFILE TEST
110 KILL "test.mfr"
120 KILL "test.mfk"
```

ou

```
110 KILL "test.mf*"
```

Rappelons que l'instruction LET "#DELET.." de MEMFILE a été spécialement conçue pour détruire les fichiers MEMFILE.

# LBOUND FONCTION

## BUT

La fonction LBOUND donne la valeur minimale que peut prendre l'indice d'un tableau.

## SYNTAXE

LBOUND ( <nom de tableau> [ ,<index> ] )

où <index> représente la position de l'indice.

## EXPLICATION

LBOUND donne la plus petite valeur que peut prendre l'indice d'un tableau à une dimension ou l'indice indiqué d'un tableau à plusieurs dimensions. Le second paramètre <index> qui indique le numéro d'indice n'est nécessaire que pour les tableaux à plusieurs dimensions.

## EXEMPLE

```
100 DIM annee(1985 TO 2000), ca(1985 TO 2000,5)
 .
 .
 .
200 x= LBOUND(annee)
210 y= LBOUND(ca,2)   ! 2ème indice
```

## ERREURS

L'erreur No 4008 est détectée si l'index est plus petit que 1 ou supérieur au nombre maximum d'indices.

L'erreur No 25005 est détectée si le paramètre n'est pas un nom de tableau sans indices mais un nom de variable simple.

L'erreur No 4901 est détectée si l'index est négatif.



L'erreur No 25101 est détectée si l'index n'est pas un numérique.

## REMARQUE

Voir la fonction UBOUND qui donne la plus grande valeur que peut prendre un indice.

# LCASE$ FONCTION

## BUT

LCASE$ restitue une chaîne ne comportant que des minuscules.

## SYNTAXE

LCASE$ (<expression chaîne>)

## EXPLICATION

LCASE$ produit une chaîne de même longueur que la chaîne donnée comme paramètre. La seule différence entre la chaîne de sortie et la chaîne d'entrée réside dans le fait que les majuscules éventuelles de la chaîne d'entrée sont transformées en minuscules dans la chaîne de sortie.

## EXEMPLE

```
500 PRINT "DONNER VOTRE REPONSE"
510 INPUT a$
520 a$ = LCASE$(a$)
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.



# LEFT$ FONCTION

## BUT

LEFT$ permet d'extraire une sous-chaîne à partir des premiers caractères de la chaîne donnée comme paramètre.

## SYNTAXE

LEFT$ (<expression chaîne>,<expression numérique>)

## EXPLICATION

L'expression chaîne est évaluée ainsi que l'expression numérique.

La valeur obtenue par l'évaluation de l'expression numérique est, si nécessaire, convertie sous forme d'un entier arrondi au plus proche. Soit n cette valeur. LEFT$ retourne alors les n premiers caractères de la chaîne obtenue par l'évaluation du premier paramètre.

## EXEMPLE

```
100 INPUT A$
110 B$=LEFT$(A$,4)
120 PRINT "Les 4 premiers caractères de a$ sont";B$
```

## ERREURS

L'erreur No 25102 est détectée si le premier paramètre n'est pas une chaîne.

L'erreur No 25101 est détecté si le second paramètre n'est pas numérique.

L'erreur No 4901 est détectée si le second paramètre est négatif.

L'erreur No 1902 est détectée si le second paramètre est hors limites.

REMARQUE

Dans le cas particulier où l'expression numérique indiquant le nombre de caractères à extraire est nulle, la chaîne produite sera la chaîne vide.



# LEN FONCTION

## BUT

La fonction LEN donne la longueur de l'expression chaîne donnée comme paramètre.

## SYNTAXE

```
LEN (<expression chaîne>)
```

## EXEMPLE

```
100 INPUT a$
110 IF LEN(a$) = 0 THEN PRINT "La chaîne est vide"
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas du type chaîne.

# LET INSTRUCTION

## BUT

L'instruction LET permet d'affecter une valeur à une variable simple.

LET est également utilisé pour toutes les instructions MEMSCREEN et MEMFILE.

## SYNTAXE

LET <variable> = <expression>

Le mot-clé LET est facultatif et on peut également écrire : <variable> = <expression> .

La forme :

LET "<chaîne de caractères>"

exécute une commande MEMSCREEN ou MEMFILE selon le contenu de la chaîne passée en paramètre. Se référer aux sections III "MEMSCREEN" et IV "MEMFILE" pour la syntaxe de LET dans ce cas.

## EXPLICATION

Dans la forme LET affectation, la variable et l'expression doivent être toutes deux de type numérique ou de type chaîne.

Si la variable située à gauche du signe égal est de type numérique, entier ou réel, la valeur de l'expression est convertie au type entier ou réel afin de concorder avec celui de la variable, avant que l'affectation proprement dite ne soit effectuée.

Les explications sur la forme LET "<chaîne>" sont données dans les sections III "MEMSCREEN" et IV "MEMFILE".

## EXEMPLE 1

```
10 LET a=b+c
20 a=b+c        ! équivaut à la ligne 10
```



## EXEMPLE 2

```
LET "#CLEAR,$" ! Commande MEMSCREEN ou MEMFILE.
```

## ERREURS

L'erreur No 25101 ou l'erreur No 25102 est détectée si variable et expression ne sont pas de même type dans la forme affectation du LET.

D'autres erreurs peuvent être détectées dans les forme MEMSCREEN et MEMFILE de LET.

## CONSEIL

Le mot-clé LET est facultatif dans la forme LET affectation. Nous vous conseillons de le réserver aux formes LET <expression chaîne> de MEMSCREEN et MEMFILE pour clarifier les programmes.

# LINE INPUT # INSTRUCTION

## BUT

Permet de lire des données pouvant inclure des "," depuis un fichier PRODOS ou un port de communication série.

## SYNTAXE

LINE INPUT #<numéro fichier>, <variable chaîne>

<numéro fichier> est une expression numérique entière donnant le numéro du fichier à utiliser.

## EXPLICATION

Le système tente de lire sur le fichier (PRODOS ou communication) une donnée de type chaîne et l'affecte à la variable chaîne désignée.

Le système reconnaît, comme seul séparateur de données, la "fin de ligne. Si une donnée chaîne est entre guillemets, ceux-ci seront conservés. Les espaces à gauche et à droite ne sont pas supprimés (s'ils doivent l'être, utiliser LTRIM$ et RTRIM$).

Si la fin de fichier est rencontrée, la fonction STATUS rendra 255. Si la lecture se termine normalement, STATUS rendra 0.

## EXEMPLE

```
 90 ! Liste séquentielle
100       OPEN "I",1,"liste.txt"
110 'suite':
120       LINE INPUT #1,a$
130       PRINT a$
140       GOTO 'suite'
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier ouvert.

L'erreur No 28514 est détectée si le mode d'ouverture du fichier n'autorise pas la lecture.



L'erreur No 25101 est détectée si le numéro de fichier n'est pas une expression numérique.

L'erreur No 28518 est détectée si une donnée est trop longue.

## REMARQUES

Voir aussi INPUT # et INPUT$ qui permettent également de lire des données.

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications séries.

# LIST COMMANDE

## BUT

La commande LIST permet d'obtenir la liste partielle ou complète du programme en mémoire.

## SYNTAXE

LIST [<page>] [<rég.> [,<rég.>]..] ["[@]<fichier>"]

<page> peut être '<' ou '>'.

<rég.> représente la partie du programme à lister. Cela peut être une désignation de ligne (<ligne>) ou d'un ensemble de lignes décrit sous la forme <ligne> TO <ligne>.

Le paramètre <ligne> peut être :
- un numéro de ligne,
- une étiquette,
- '+' pour indiquer le numéro de ligne maximum,
- '-' pour indiquer le numéro de ligne minimum,
- '*' pour la dernière ligne validée par RETOUR CHARIOT.

<fichier> représente une spécification de fichier. Cette option s'utilise si l'on veut sauver le résultat de l'instruction LIST dans un fichier.

## EXPLICATION

La forme LIST signifie que l'ensemble du programme doit être listé.

La forme LIST <ligne> provoque l'affichage de la ligne dont le numéro est indiqué.

La forme LIST <ligne> TO <ligne> provoque l'affichage de la portion de programme comprise entre les deux lignes indiquées. Les exemples illustrent différentes variantes de cette forme.

Si le paramètre <page> est spécifié, la portion de programme visible dans la fenêtre MEMBASIC est remplacée par la page précédente (option '<') ou par la page suivante (option '>').



Si une ligne a été modifiée dans la page en cours, c'est cette même page qui sera réaffichée par l'instruction LIST < ou LIST >.

Si une spécification de fichier est indiquée (entre guillemets), alors le résultat est envoyé dans le fichier en question au lieu d'être affiché à l'écran.

Si aucun suffixe n'est spécifié, le suffixe implicite est .PRG.

Comme pour la commande SAVE, le caractère "@" doit précéder la spécification de fichier si le fichier existe déjà.

Si l'imprimante a préalablement été activée, au moyen de l'instruction PRINTER, alors LIST sans spécification de fichier envoie le résultat vers l'imprimante.

Le listage du programme à l'écran peut être suspendu ou ralenti:

- La touche SHIFT suspend le listage.
  Tant que l'une des touches SHIFT est enfoncée, l'exécution du LIST est arrêtée.

  On peut alors :

  - avancer d'une ligne en enfonçant puis relachant la touche CTRL (sans relacher la touche SHIFT),

  - passer en mode "gestion des fenêtres" en enfonçant la touche "OPTION CLIC-Souris" pour consulter d'autres fenêtres,

  - stopper le listage en enfonçant POMME CTRL C.

  Au relachement de la touche SHIFT, le listage reprend normalement.

## EXEMPLES

```
LIST 200 TO 500 ! affiche les lignes 200 à 500
LIST TO 1000    ! affiche les lignes 1 à 1000
LIST 400 TO     ! affiche toutes lignes depuis 400
```

```
LIST 800, 1000 TO 1200, 'etiq' TO
```

Cet exemple liste la ligne 800, puis les lignes 1000 à 1200, puis toutes les lignes à partir de l'étiquette 'etiq'.

```
LIST 3000, 4000 TO 6000 "PART34.LST"
```

Cette forme sauve sur disque sous forme texte la ligne 3000 puis les lignes 4000 à 6000.

```
PRINTER 1 ! liste le programme sur imprimante
LIST
PRINTER 0
```

## ERREUR

L'erreur No 28030 est détectée si le fichier spécifié existe déjà et que LIST a été utilisé avec une spécification de fichier non précédée du caractère "@".

## REMARQUE

En mode commande, les touches POMME 3 et POMME 4 effectuent respectivement les instructions LIST < et LIST >.



# LOAD INSTRUCTION/COMMANDE

## BUT

La commande LOAD permet de charger un programme stocké sur mémoire auxiliaire.

## SYNTAXE

```
LOAD "<spécification de fichier>"
```

## EXPLICATION

LOAD peut être utilisé soit en mode commande, soit en mode programme pour charger des programmes sauvegardés sous l'un des trois modes possibles : normal, protégé, texte ( voir SAVE et LIST ).

En mode programme, LOAD suspend l'exécution du programme en cours. La place qu'il occupe en mémoire est récupérée pour loger le programme qui sera chargé.

Tous les fichiers et masques qui étaient éventuellement ouverts sont d'abord fermés, la table des variables est effacée, le programme figurant comme paramètre est chargé.

En l'absence de suffixe, MEMBASIC charge le programme ayant le suffixe .PRG.

## EXEMPLE

```
LOAD "CONVERT"
```

## ERREUR

L'erreur No 28010 est détectée si le nom de programme donné ne peut pas être atteint.

# LOCNUMBER COMMANDE

## BUT

La commande LOCNUMBER permet de renuméroter un programme avec les adresses de début d'instruction comme numéros de ligne.

## SYNTAXE

LOCNUMBER

## EXPLICATION

En cas d'erreur dans l'exécution d'un programme protégé, la fonction EXLINE permet de connaître l'adresse de l'instruction à laquelle a été détectée l'erreur. Avec LOCNUMBER la numérotation du programme correspond aux adresses des instructions. Il est donc possible de déterminer facilement la ligne où s'est produite l'erreur par la fonction EXLINE.

## EXEMPLE

```
10 a = 15
20 PRINT a
30 FOR i = 1 TO a : PRINT b(i) : NEXT
40 PRINT i

LOCNUMBER
LIST

0 a = 15
6 PRINT a
12 FOR i = 1 TO a : PRINT b(i) : NEXT
38 PRINT i
```

L'exécution de ce programme en mode protégé engendrera une erreur à l'adresse 24. Celle-ci correspond au débordement de l'indice du tableau dont le dimensionnement par défaut est 10.

## ERREUR

Néant.



## REMARQUE

La modification d'un programme renuméroté avec LOCNUMBER peut poser quelques problèmes pour les lignes de commentaires.
En effet, trois lignes de commentaires successives auront les mêmes numéros de ligne, celles-ci ne comportant pas de code. Seule la première de ces lignes pourra être modifiée.
L'exécution de RENUMBER permettra de nouveau la modification de toutes les lignes du programme.

# LOG FONCTION

## BUT

Cette fonction donne le logarithme népérien d'un nombre.

## SYNTAXE

LOG (<expression numérique>)

où <expression numérique> doit être supérieur à 0.

## EXPLICATION

Le logarithme népérien est le logarithme en base e.

## EXEMPLE

```
100 e = 2.718281828459
110 PRINT LOG (e)
RUN
1
```

## ERREURS

L'erreur No 3004 est détectée si <expression numérique> est inférieure ou égale à 0.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# LOG10 FONCTION

## BUT

Cette fonction donne le logarithme décimal d'un nombre.

## SYNTAXE

LOG10 (<expression numérique>)

où <expression numérique> doit être supérieur à 0.

## EXPLICATION

Le logarithme décimal est le logarithme en base 10.

## EXEMPLE

```
PRINT LOG10 (100)
2
```

## ERREURS

L'erreur No 3004 est détectée si <expression numérique> est inférieure ou égale à 0.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# LOG2 FONCTION

## BUT

Cette fonction donne le logarithme en base 2 d'un nombre.

## SYNTAXE

LOG2 (<expression numérique>)

où <expression numérique> doit être supérieur à 0.

## EXEMPLE

```
PRINT LOG2 (2)
1
```

## ERREURS

L'erreur No 3004 est détectée si <expression numérique> est inférieure ou égale à 0.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# LOOP INSTRUCTION

## BUT

Instruction qui termine une boucle DO.

## EXPLICATION

VOIR INSTRUCTION DO.

# LTRIM$ FONCTION

## BUT

La fonction LTRIM$ supprime les espaces de début de chaîne.

## SYNTAXE

LTRIM$ (<expression chaîne>)

## EXPLICATION

Cette fonction donne une sous-chaîne extraite de la chaîne paramètre par suppression des espaces éventuels du début de la chaîne. Dès qu'un caractère différent de l'espace apparaît, les espaces suivants ne sont plus supprimés :

LTRIM$ ("  IL FAIT BEAU") donne "IL FAIT BEAU"

## EXEMPLE

```
110 A$ = LTRIM$(N$)
120 NOM$ = RTRIM$(A$)
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

## REMARQUE

Voir la fonction RTRIM$ qui supprime les espaces situés à la fin d'une chaîne.



# MAT INSTRUCTION

## BUT

MAT permet d'initialiser un tableau ou de copier un tableau dans un autre.

## SYNTAXE

MAT <nom tableau> = <expression>

ou

MAT <nom tableau> = <nom tableau original>

## EXPLICATION

La première forme de MAT évalue l'expression puis remplit tous les éléments du tableau avec la valeur obtenue. Le tableau peut être de n'importe quel type (entiers, flottants, chaînes fixes ou variables). L'expression doit être d'un type compatible (numérique dans les deux premiers cas et chaîne dans les deux derniers).

L'intérêt principal de cette forme est l'initialisation rapide et simple d'un tableau. Par exemple, la remise à zéro se fera en une seule instruction et sera presque instantanée, ce qui ne serait pas le cas avec une boucle FOR ... NEXT.

La seconde forme permet de copier un tableau dans un autre. Les deux tableaux doivent être strictement de même type et de mêmes dimensions.

La copie est beaucoup plus rapide en utilisant MAT qu'avec une boucle de copie.

## EXEMPLE

```
100 OPTION BASE 1
110 DIM a(10), b(3,5), c(3,5)
120 MAT a=1          ! remplit de 1 le tableau a
130 x=2              ! initialise un scalaire
140 MAT c=x          ! remplit c avec l'express. x
150 MAT b=c          ! copie de c dans b
```

Le programme classique équivalent est:

```
100 OPTION BASE 1
110 DIM a(10), b(3,5), c(3,5)
120 FOR i= 1 TO 10    ! remplit de 1 le tableau a
121     a(i)=1
122 NEXT
130 x=2               ! initialise un scalaire
140 FOR i=1 TO 3      ! remplit c avec l'express. x
141     FOR j=1 TO 5
142         c(i,j)=x
143     NEXT
144 NEXT
150 FOR i=1 TO 3      ! copie de c dans b
151     FOR j=1 TO 5
152         b(i,j)=c(i,j)
153     NEXT
154 NEXT
```

## ERREURS

L'erreur No 25101 est détectée si le premier paramètre est un tableau numérique et si le second paramètre est une expression chaîne.

L'erreur No 25102 est détectée si le premier paramètre est un tableau de chaînes et si le second paramètre est une expression numérique.

L'erreur No 25103 est détectée si les paramètres sont des tableaux de types incompatibles.

L'erreur No 25104 est détectée si les paramètres sont des tableaux de dimensions différentes.

## REMARQUE

Notez bien qu'aucune parenthèse n'est demandée pour préciser que l'un des paramètres est un tableau. En effet, un tableau et une variable simple ne pouvant avoir le même nom, aucune ambiguïté n'est possible.



# MAX FONCTION

## BUT

MAX donne la valeur du plus grand de deux paramètres.

## SYNTAXE

MAX (<expression numérique>,<expression numérique>)

## EXPLICATION

La fonction MAX évalue les valeurs numériques des deux paramètres et restitue la plus grande valeur trouvée.

## EXEMPLE

```
100 INPUT a, b
110 PRINT MAX(a,b) ;  "est la plus grande valeur"
```

## ERREUR

L'erreur No 25101 est détectée si l'un des paramètres n'est pas de type numérique.

## REMARQUE

Voir la fonction MIN qui permet d'obtenir le minimum.

# MAXLEN FONCTION

## BUT

La fonction MAXLEN donne la longueur maximum allouée à une variable chaîne.

## SYNTAXE

MAXLEN (<variable chaîne>)

La variable chaîne peut être une variable simple ou un tableau.

## EXPLICATION

MAXLEN donne pour résultat :

- ou bien la longueur maximum qui a été allouée à une variable chaîne au moyen de l'une des instructions DECLARE STRING et DIM,

- ou bien la valeur 255 si aucune longueur maximum n'a été fixée.

## EXEMPLES

```
100
 .
 .
300 m=MAXLEN(a$)
320 PRINT "a$ a une longueur maximale fixée à ";m
```

## ERREUR

L'erreur No 25006 est détectée si la variable donnée comme paramètre n'est pas de type chaîne.



# MAXLINE FONCTION

## BUT

Cette fonction rend le nombre de lignes par page de l'imprimante fixé par SET MAXLINE.

## SYNTAXE

MAXLINE

## EXPLICATION

Le nombre de lignes par page de l'imprimante est utilisé par les instructions LET "?..." et LET "HARDCOPY..." de MEMSCREEN.

Il doit être fixé auparavant par l'instruction SET MAXLINE. La valeur implicite de MAXLINE est 66.

## EXEMPLES

```
PRINT MAXLINE
 66
```

## ERREUR

Néant.

# MAXNUM FONCTION

## BUT

Cette fonction sans paramètre donne la valeur du plus grand nombre représentable en MEMBASIC qui est :

9.9999999999999E+63

## SYNTAXE

MAXNUM

## EXEMPLE

```
IF x >= .9 * MAXNUM THEN 1000
```

## REMARQUE

Cette fonction peut être utilisée afin d'éviter des débordements numériques.

## ERREUR

Néant.



# MEMBACKUP INSTRUCTION

## BUT

Sauvegarde de fichiers d'un répertoire d'un disque dur vers une ou plusieurs disquettes.

## SYNTAXE

MEMBACKUP <expression chaîne> TO <express. chaîne>

La première expression chaîne représente le répertoire à sauvegarder suivi d'un éventuel critère de sélection.

La deuxième expression chaîne indique le lecteur de disquette concerné par la sauvegarde, sous la forme <lettre du lecteur> suivi de ":".

## EXPLICATION

Le fichier spécifié dans la première expression chaîne est copié sur une ou plusieurs disquettes placées successivement dans le lecteur spécifié dans la seconde expression chaîne.

On peut sauvegarder plusieurs fichiers en utilisant dans le nom de fichier les caractères spéciaux ? et *.
Un ? dans le nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans le nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

Si le nom du fichier à copier est un nom de répertoire, c'est le répertoire complet qui sera sauvegardé.

En mode immédiat, la liste des fichiers sauvegardés sera affichée dans la fenêtre courante.

Si la disquette est de capacité insuffisante pour contenir tous les fichiers ou les fichiers les plus gros, de nouvelles disquettes seront automatiquement demandées. Dans ce cas, il est possible que certains fichiers soient coupés et se retrouvent sur deux disquettes ou plus.

Toutes les disquettes utilisées doivent avoir été formatées au préalable (Cette opération ne peut pas être réalisée depuis MEMSOFT).

La séquence des opérations est la suivante :

- Demande de la disquette Numéro 1.

- Si la racine de la disquette n'est pas vide, il demande de confirmer son effacement, sinon il vous redonne la main sans rien avoir fait et réitère l'opération précédente.

- Remplissage de la disquette avec les fichiers à copier.

- Si la copie est incomplète, une nouvelle disquette est réclamée (la disquette numéro 2) et ainsi de suite.

Tous les messages apparaissent en bas d'écran, à la place des touches de fonction.

Sur chacune des disquettes de sauvegarde, un fichier identificateur est automatiquement ajouté. Ce fichier dont le nom est MEMBCKUP.@@@, est un fichier texte en ASCII, il est donc lisible très simplement (il suffit d'ouvrir le fichier sous PRODOS et de demander à le voir).

Le fichier MEMBACKUP.@@@ contient les lignes suivantes :

DISK #<x>
Numéro de la disquette.

Directory : <ddddd>
Nom du répertoire sauvegardé (seulement sur la première disquette).

Warning : <fffff> is incompletly saved
Nom d'un fichier qui n'a pas été entièrement copié.

LAST DISK.
Si cette disquette est la dernière.

Ce fichier servira lors d'une éventuelle restauration à éviter toute erreur de manipulation (voir MEMRESTORE).



A noter :

- Il ne peut y avoir que deux lignes "Warning ..." puisque deux fichiers au maximum peuvent être incomplets : le premier et le dernier de la disquette.

- Une seule disquette peut porter la mention "LAST DISK" !

En mode immédiat, la liste des fichiers sauvegardés sera affichée dans la fenêtre courante.

## EXEMPLES

```
MEMBACKUP "*.MF*" TO "A:"
```

Sauvegarde d'un ensemble de fichiers du répertoire implicite sur des disquettes placées successivement dans le lecteur A:.

```
MEMBACKUP "/COMPTA/EX86" TO "a:"
```

Sauvegarde d'un répertoire complet si EX86 est un sous répertoire de COMPTA.

## ERREURS

L'erreur No 25102 est détectée si l'un des paramètres n'est pas une chaîne.

L'erreur No 28010 est détectée s'il n'y a aucun fichier à sauvegarder.

L'erreur No 28051 est détectée si l'utilisateur a abandonné l'opération en enfonçant sur la touche ESC.

## CONSEILS

MEMBACKUP ne formate pas les disquettes et en cas de manque de disquettes, toute votre sauvegarde sera à refaire ! Il est donc conseillé, avant de commencer vos sauvegardes, de formater un nombre suffisant de disquettes. La fonction FILESIZE peut vous aider à évaluer le nombre de disquettes nécessaires.

Utilisez l'instruction MEMCOMPARE conjointement avec MEMBACKUP pour vérifier la validité de votre sauvegarde. Il vaut mieux s'apercevoir immediatement qu'une sauvegarde n'est pas bonne plutôt que d'attendre le moment d'une restauration !

## REMARQUE

Si les disquettes utilisées pour la sauvegarde, contiennent des répertoires, ceux-ci ne seront pas effacés. La disquette ne sera pas remplie au maximum par les fichiers sauvegardés.



# MEMCOMPARE INSTRUCTION

## BUT

Vérification des disquettes de sauvegarde d'un répertoire de disque dur.

## SYNTAXE

MEMCOMPARE <express. chaînel> TO <express. chaîne2>

<expression chaînel> : lecteur de disquette ou seront placées successivement les disquettes à verifier sous la forme <lettre du lecteur> suivi de ":" et optionellement d'un critére de selection.

<expression chaîne2> : définit le disque dur et le répertoire sur lequel s'effectuera la comparaison.

## EXPLICATION

Cette instruction permet de vérifier une sauvegarde effectuée avec l'instruction MEMBACKUP ou restaurée avec l'instruction MEMRESTORE.

MEMCOMPARE demande d'introduire dans le lecteur les disquettes de sauvegardes les unes à la suite des autres. Chacune des disquettes porte un numéro qui est rappelé dans le message et qui a été défini lors de la sauvegarde. Ce numéro se trouve dans le fichier MEMBCKUP.@@@ (voir MEMBACKUP). Si une disquette invalide ou hors de séquence a été placée dans le lecteur, le problème est automatiquement détecté et un message apparait pour le signaler.

Tous les messages s'affichent en bas d'écran.

On peut également ne comparer que certains fichiers parmi ceux se trouvant sur la disquette d'origine. Pour cela, il suffit d'utiliser dans la première expression chaîne un critère définissant le nom des fichiers concernés à l'aide des caractères spéciaux ? et *.
Un ? dans le nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans le nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

En mode immédiat, la liste des fichiers comparés sera affichée dans la fenêtre courante.

## EXEMPLES

MEMCOMPARE "A:" TO "C:/COMPTA/EX86"

Compare tous les fichiers se trouvant sur les disquettes de sauvegardes introduites dans le lecteur A:, avec ceux du répertoire /COMPTA/EX86 du drive C:.

MEMCOMPARE "A:*.MF*" TO "C:/COMPTA/EX86"

Compare les fichiers MEMFILE (les *.MFR et *.MFK) parmi tous les fichiers pouvant se trouver sur les disquettes du lecteur A:.

## ERREURS

L'erreur No 28010 est détectée si il n'y a aucun fichier à comparer.

L'erreur No 28051 est détectée si l'utilisateur a abandonné l'opération en enfonçant sur la touche ESC.

L'erreur No 28100 est détectée si il y a eu une erreur dans la comparaison de deux fichiers.

## CONSEILS

Utilisez l'instruction MEMCOMPARE conjointement avec MEMBACKUP pour vérifier la validité de votre sauvegarde. Il vaut mieux s'apercevoir immédiatement qu'une sauvegarde n'est pas bonne plutôt que d'attendre le moment d'une restauration !

Bien sûr, après une restauration il serait également souhaitable que vous utilisiez cette instruction qui vous permettrait de vérifier vos copies.



# MEMRESTORE INSTRUCTION

## BUT

Restaure sur disque dur les fichiers ayant été sauvegardés par MEMBACKUP.

## SYNTAXE

MEMRESTORE <express. chaînel> TO <express. chaîne2>

<expression chaînel> : lecteur de disquette ou seront placées successivement les disquettes à restaurer sous la forme <lettre du lecteur> suivi de ":" et optionellement d'un critère de sélection.

<expression chaîne2> : définit le disque dur et le répertoire sur lequel s'effectuera la restauration. Il suffit de mettre le caractère @ en début d'expression pour que les fichiers déjà existants soient écrasés.

## EXPLICATION

Cette instruction permet de restaurer une sauvegarde effectuée avec l'instruction MEMBACKUP.

MEMRESTORE demande d'introduire dans le lecteur les disquettes de sauvegardes les unes à la suite des autres. Chacune des disquettes porte un numéro qui est rappelé dans le message et qui a été défini lors de la sauvegarde. Ce numéro se trouve dans le fichier MEMBCKUP.@@@ (voir MEMBACKUP). Si une disquette invalide ou hors de séquence a été placée dans le lecteur, le problème est automatiquement détecté et un message apparait pour le signaler.

Tous les messages s'affichent en bas d'écran.

On peut ne restaurer que certains fichiers parmi ceux se trouvant sur la disquette d'origine. Pour cela, il suffit d'utiliser dans la première expression chaîne un critère définissant le nom des fichiers concernés à l'aide des caractères spéciaux ? et *.
Un ? dans le nom de fichier indique qu'un caractère quelconque peut occuper cette position.

Un * dans le nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

On peut restaurer les fichiers sur n'importe quel répertoire du disque dur, même différent de celui sauvegardé, mais si le répertoire concerné contient déjà des fichiers de même noms, il faut précéder la seconde expression chaîne de @ pour demander le remplacement.

En mode immédiat, la liste des fichiers restaurés sera affichée dans la fenêtre courante.

## EXEMPLES

```
MEMRESTORE "A:" TO "C:/COMPTA/EX86"
```

Restaure tous les fichiers se trouvant sur les disquettes de sauvegardes introduites dans le lecteur A:, avec ceux du répertoire /COMPTA/EX86 du drive C:.

```
MEMRESTORE "A:COMPTES.MF*" TO "C:/COMPTA/EX86"
```

Restaure uniquement certain fichiers (COMPTES.MFR et COMPTES.MFK qui constituent le fichier MEMFILE COMPTES) parmi tous les fichiers pouvant se trouver sur les disquettes du lecteur A:.

## ERREURS

L'erreur No 28010 est détectée si il n'y a aucun fichier à restaurer.

L'erreur No 28051 est détectée si l'utilisateur a abandonné l'opération en enfonçant sur la touche ESC.

L'erreur No 28030 est détectée si un fichier à restaurer existe déjà et qu'il n'y a pas de caractères @ en début de seconde expression chaîne.

## CONSEIL

Utilisez l'instruction MEMCOMPARE conjointement avec MEMRESTORE pour vérifier la validité de votre restauration. Il vaut mieux s'apercevoir immédiatement qu'une restauration n'est pas bonne...



# MID$ FONCTION

## BUT

MID$ permet d'extraire une sous-chaîne.

## SYNTAXE

MID$ ( <chaîne>,<valeur départ> [,<longueur>] )

Où <chaîne>, <valeur départ> et <longueur> sont des expressions, la première de type chaîne, les deux autres de type numérique.

## EXPLICATION

MID$ extrait à partir du caractère de rang "valeur départ" une sous-chaîne de la chaîne de départ.

<longueur> indique la longueur de la sous-chaîne à extraire.

Si le troisième paramètre est absent, ou s'il est trop grand, la sous-chaîne se termine au dernier caractère de la chaîne de départ.

Si <valeur départ> est supérieure à la longueur de la <chaîne> ou si <longueur> est nulle alors MID$ extrait une chaîne vide.

## EXEMPLE

```
10 a$="ABCDEFGHIJ"
20 PRINT a$
30 PRINT MID$(a$,3,4)
RUN
ABCDEFGHIJ
CDEF
```

## ERREURS

L'erreur No 25102 est détectée si le premier paramètre n'est pas une chaîne.

L'erreur No 25101 est détectée si le deuxième ou le troisième paramètre n'est pas numérique.

L'erreur No 4901 est détectée si l'un des deux paramètres numériques est négatif.

L'erreur No 1902 est détectée si l'un des deux paramètres numériques est hors limites.

## REMARQUE

Pour extraire une sous-chaîne, on peut écrire également :

```
a$(3:6) au lieu de MID$(a$,3,4)
```

et d'une manière générale :

```
a$(d:d+l-1) au lieu de MID$(a$,d,l)

t$(i,j)(d:d+l-1) au lieu de MID$(t$(i,j),d,l)
```



# MIN FONCTION

## BUT

MIN donne la valeur du plus petit de deux paramètres.

## SYNTAXE

MIN (<expression numérique>,<expression numérique>)

## EXPLICATION

La fonction MIN évalue les valeurs numériques des deux paramètres et restitue la plus petite valeur trouvée.

## EXEMPLE

```
100 INPUT a, b
110 PRINT MIN(a,b) "est la plus petite valeur"
```

## ERREUR

L'erreur No 25101 est détectée si l'un des paramètres n'est pas de type numérique.

## REMARQUE

Voir la fonction MAX qui permet d'obtenir le maximum.

# MKDIR INSTRUCTION

## BUT

MKDIR permet de créer un sous-répertoire.

## SYNTAXE

MKDIR "<expression chaîne>"

Où <expression chaîne> doit représenter un chemin acceptable par le système d'exploitation.

## EXPLICATION

L'instruction MKDIR crée un sous-répertoire à condition que le dernier indiqué soit syntaxiquement et techniquement acceptable par le système d'exploitation.

## EXEMPLE

```
100 PRINT "Chemin du sous-répertoire à créer " ;
110 INPUT a$
150 MKDIR a$
```

## ERREURS

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

L'erreur No 28130 est détectée si le répertoire spécifié existe déjà.

## REMARQUES

Voir les instructions CHDIR et RMDIR qui permettent respectivement de changer de sous-répertoire et de supprimer un sous-répertoire vide.

Voir également l'instruction PATH.



# MOD FONCTION

## BUT

MOD donne la valeur d'une expression modulo une autre expression.

## SYNTAXE

MOD (<expression numérique>,<expression numérique>)

## EXPLICATION

Cette fonction est exécutée de la façon suivante :

- Evaluation des deux expressions numériques données comme paramètres.
- Détermination du modulo selon la formule :

  MOD(x,y) = x - y * INT(x/y)

## EXEMPLE

```
PRINT MOD(5,3); MOD(-5,3)
2  1
```

## ERREURS

L'erreur No 25101 est détectée si l'un des paramètres n'est pas de type numérique.

L'erreur No 3001 est détectée si la valeur du second paramètre est nulle.

# NEW COMMANDE

## BUT

NEW permet d'effacer le programme se trouvant en mémoire centrale en vue de l'introduction d'un nouveau programme.

## SYNTAXE

NEW

## EXPLICATION

Cette instruction efface complètement le programme qui était résident en mémoire centrale. Elle remet à zéro l'ensemble des tables utilisées par l'interpréteur MEMBASIC. La table des variables est également effacée. Les fichiers qui étaient encore ouverts avant l'exécution de cette instruction sont fermés.

## ERREUR

Néant.



# NEXT INSTRUCTION

## BUT

NEXT est utilisée conjointement avec l'instruction FOR pour marquer la fin de la "portée" de la boucle FOR.

## SYNTAXE

NEXT [ <variable numérique> ]

## EXPLICATION

Voir le détail de fonctionnement de la boucle FOR/NEXT dans le paragraphe EXPLICATION relatif à l'instruction FOR.

Le rappel du nom de variable de boucle est facultatif. Il ne sert que pour imposer à MEMBASIC d'effectuer le contrôle. On comprendra dès lors que cela coûte un peu de temps et de place ...

Il ne doit y avoir qu'un seul NEXT par boucle FOR/NEXT. S'il est nécessaire de passer à l'élément suivant de la boucle à plusieurs endroits du corps de la boucle, utiliser l'instruction SKIP FOR.

## EXEMPLE

Voir l'instruction FOR.

## ERREUR

L'erreur No 10902 est détectée si la variable est spécifiée et qu'elle n'a pas le même nom que la variable de contrôle de l'instruction FOR associée.

# ON...GOSUB INSTRUCTION

## BUT

Cette instruction permet d'effectuer un branchement vers un sous-programme.

## SYNTAXE

```
                                        <numéro ligne>
ON <expression> GOSUB <liste> ELSE      <étiquette>
                                        <Instr.>
```

Où <liste> a la forme suivante :

```
<numéro de ligne>    [ , <numéro de ligne> ] ....
  ou <étiquette>           ou <étiquette>
```

<expression> est une expression numérique.

## EXPLICATION

L'expression est évaluée et sa valeur est arrondie, si nécessaire, à l'entier le plus proche.

Si le résultat ainsi obtenu est supérieur à 0 et inférieur ou égal au nombre d'éléments de la liste, alors il y a branchement à la ligne ou à l'étiquette correspondante dont le rang dans la liste est égal au résultat de l'évaluation de l'expression. Le RETURN associé à ce sous-programme fera revenir à l'instruction qui suit immédiatement l'instructon ON...GOSUB.

Si le résultat est supérieur au nombre d'éléments de la liste, ou si le résultat est négatif ou nul, alors deux cas sont possibles :

- ou bien l'option ELSE est présente et dans ce cas elle est exécutée,
- ou bien elle est absente et le programme se poursuit à l'instruction suivante.



## EXEMPLES

```
100 INPUT PROMPT "1:LIRE 2:MODIF autre:FIN ": choix
110 ON choix GOSUB 'lire' , 'modifier' ELSE 'fin'
```

## ERREUR

L'erreur No 25101 est détectée si l'expression n'est pas de type numérique.

## CONSEIL

Pour une programmation soignée, il est préférable d'utiliser l'option ELSE.

# ON...GOTO INSTRUCTION

## BUT

Cette instruction permet un branchement conditionnel.

## SYNTAXE

```
                                        <numéro ligne>
ON <expression> GOTO <liste> ELSE       <étiquette>
                                        <Instr.>
```

Où <liste> a la forme suivante :

```
<numéro de ligne>    [ , <numéro de ligne> ] ....
  ou <étiquette>           ou <étiquette>
```

## EXPLICATION

L'expression est évaluée et sa valeur est arrondie, si nécessaire, à l'entier le plus proche.

Si le résultat ainsi obtenu est supérieur à 0 et inférieur ou égal au nombre d'éléments de la liste, alors il y a branchement à la ligne ou à l'étiquette correspondante dont le rang dans la liste est égal au résultat de l'évaluation de l'expression.

Si le résultat est supérieur au nombre d'éléments de la liste, ou si le résultat est négatif ou nul, alors deux cas sont possibles :

- ou bien l'option ELSE est présente et dans ce cas elle est exécutée,

- ou bien elle est absente et le programme se poursuit à l'instruction suivante.

## EXEMPLES

```
100 INPUT PROMPT "1:LIRE 2:MODIFIER 3:FIN ": choix
110 ON choix GOTO 'lire','modifier','fin' ELSE 100
```



## ERREUR

L'erreur No 25101 est détectée si l'expression n'est pas de type numérique.

## CONSEIL

Pour une programmation soignée, il est préférable d'utiliser l'option ELSE.

# OPEN INSTRUCTION

## BUT

Cette instruction ouvre un fichier PRODOS. Une forme particulière de OPEN décrite dans les pages qui suivent ouvre un fichier de communication.

## SYNTAXE

OPEN <option>,<num. fichier>,<spécif. de fichier>
[,<long>]

<option> est une expression chaîne dont le premier caractère indique le mode d'ouverture du fichier:
"I" pour lire seulement,
"A" pour ajouter en fin de fichier,
"O" pour écrire dans un fichier après création,
"R" pour lire ou écrire des enregistrements n'importe où dans le fichier.

<numéro de fichier> est une expression numérique entière dont la valeur sera associée au fichier.

<spécification de fichier> est une expression chaîne contenant le nom du fichier à ouvrir avec la syntaxe PRODOS. Ce nom pourra contenir des spécifications de disque ou de répertoire.

<longueur> est un paramètre optionnel de valeur implicite 128. Il sert uniquement lorsque l'option "R" a été choisie pour définir la longueur d'un enregistrement.

## EXPLICATION

L'instruction OPEN permet d'associer à un fichier PRODOS un numéro utilisé pour écrire ou lire dans ce fichier.

Le nom du fichier peut contenir des spécifications de disque ou de répertoire. S'il n'en contient pas, le fichier sera cherché dans le répertoire courant. Si l'option demandée est l'option "I" (lecture seule) et si le fichier n'a pas été trouvé dans le répertoire courant, il sera recherché dans les répertoires donnés par l'instruction PATH.



Si le fichier spécifié n'existe pas, la fonction STATUS rendra 10 et l'ouverture n'aura pas lieu, mais le programme ne sera pas interrompu. Si l'ouverture se passe bien, STATUS rendra 0.

Suivant l'option choisie, les actions qui pourront être effectuées sur le fichier diffèrent :

"I": Le fichier est ouvert en lecture seule et en séquentiel. Cette méthode est particulièrement bien adaptée à la lecture des fichiers textes. On pourra lire les données grâce aux instructions INPUT #, LINE INPUT #, INPUT$.

"A": Le fichier est ouvert en mode "ajouts". On désire compléter un fichier déjà existant sans modifier les informations qui sont présentes. On pourra écrire dans le fichier grâce aux instructions PRINT # ou WRITE #.

"O": Le fichier est créé. Si le fichier existait déjà, il est détruit. On pourra écrire dans le fichier grâce aux instructions PRINT # ou WRITE #.

"R": Le fichier est un fichier structuré avec enregistrements. La taille de chaque enregistrement peut être définie dans l'instruction d'ouverture, sinon, la valeur implicite 128 est utilisée. L'instruction POINTER # permet de choisir un enregistrement qui pourra être lu grâce aux instructions INPUT #, LINE INPUT #, INPUT$, ou écrit grâce aux instructions PRINT # ou WRITE #.

Les fichiers doivent être fermés en fin d'utilisation grâce à l'instruction CLOSE. Cela permet de sauvegarder sur le disque les informations modifiées dans le cas d'écritures. La fermeture d'un fichier permet également d'en ouvrir un nouveau (rappelons que le nombre de fichiers ouverts simultanément a une limite qui dépend des paramètres d'installation du système). En tout état de cause, le nombre maximum de fichiers PRODOS ou communication ouverts simultanément est 10.

## EXEMPLE

```
100 OPEN "I",1,"essai.txt"    ! lecture
110 OPEN "A",2,"historique"   ! ajouts
120 OPEN "O",100,"tempo.$$$"  ! écriture
130 OPEN "R",5,"fiches",80    ! accès direct
```

## ERREURS

L'erreur No 28502 est détectée si le numéro de fichier correspond à un autre fichier PRODOS ou fichier de communication déjà ouvert.

L'erreur No 28501 est détectée si le nombre de fichiers PRODOS et de fichiers de communications ouverts simultanément excède le maximum permis (10).

L'erreur No 28505 est détectée si le numéro de fichier n'est pas correct (entier).

L'erreur No 28506 est détectée si le mode d'ouverture est incorrect (erreur dans la syntaxe).

## REMARQUES

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS.

L'instruction OPEN ne sert pas à ouvrir les fichiers MEMFILE. Pour ceux-ci, l'ouverture se fait grâce à l'instruction MEMFILE : LET "#OPEN...".



# OPEN "COM..." INSTRUCTION

## BUT

Cette instruction ouvre un fichier de communication.

## SYNTAXE

OPEN <expression chaîne> , <numéro de fichier>

<expression chaîne> permet de sélectionner le port choisi sous la forme :

"COM1" ou "COM2"

avec :
  v = vitesse en bauds. Les valeurs possibles sont: 110,150,300,600,1200,2400,4800,9600

<numéro de fichier> est une expression numérique entière dont la valeur sera associée au fichier.

## EXPLICATION

L'instruction OPEN permet d'associer à un fichier de communication un numéro utilisé pour écrire ou lire dans ce fichier.

Tous les paramètres de transmission (vitesse, parité, bits de données, bits d'arrêt, protocole) doivent avoir été spécifiés à partir du tableau de bord avant de lancer MEMSOFT.

La valeur implicite de la vitesse est 300 bauds.

La transmission peut se faire aussi bien en lecture (grâce aux instructions INPUT #, LINE INPUT # et la fonction INPUT$) qu'en écriture (grâce aux instructions PRINT # ou WRITE #).

## EXEMPLE

```
100 OPEN "COM1,2"
```

Ouvre un fichier de communication sur le port 2.

## ERREURS

L'erreur No 28502 est détectée si le numéro de fichier correspond à un autre fichier PRODOS ou de communication déjà ouvert.

L'erreur No 28501 est détectée si le nombre de fichiers PRODOS et de fichiers de communications ouverts simultanément excède le maximum permis (10).

L'erreur No 28505 est détectée si le numéro de fichier n'est pas correct (entier).

L'erreur No 28506 est détectée si le mode d'ouverture est incorrect (erreur dans la syntaxe).

## REMARQUE

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les fichiers de communications.



# OPTION INSTRUCTION

## BUT

Cette déclaration, selon la forme utilisée, permet de déclarer la valeur implicite de départ des indices, ou l'indice maximum implicite, ou l'unité utilisée pour les angles dans les fonctions trigonométriques.

## SYNTAXE

Quatre formes sont possibles :

```
OPTION BASE <index>
OPTION DIM <index>
OPTION ANGLE DEGREES
OPTION ANGLE RADIANS
```

## EXPLICATION

La forme OPTION BASE indique la valeur implicite de départ des indices.

La forme OPTION DIM indique la valeur implicite maximum des indices pour les tableaux non dimensionnés.

Les instructions OPTION BASE et OPTION DIM doivent être exécutées avant toute référence à des tableaux. Elles ont un effet uniquement sur l'unité de programme concernée (action "locale").

Une unité de programme ne peut contenir qu'une seule instruction OPTION BASE ou OPTION DIM.

Les formes OPTION ANGLE permettent de déterminer l'unité des angles dans les fonctions trigonométriques.

## EXEMPLE 1

```
100 OPTION BASE 0
110 DIM a(10,10)
```

## EXEMPLE 2

```
100 OPTION BASE 1980
110 DIM ca(1990)          !indices de 1980 à 1990
```

## EXEMPLE 3

```
100 OPTION DIM 30
110 OPTION BASE 1
120 n = 30
130 FOR i=1 TO 30
140   a(i)=0
150   NEXT i
```

## EXEMPLE 4

```
100 OPTION ANGLE DEGREES
110 PRINT COS (180)
RUN
-1
```

## EXEMPLE 5

```
100 OPTION ANGLE RADIANS
110 PRINT COS (3.14159265359)
RUN
-1
```

## ERREURS

L'erreur No 25101 est détectée si l'index n'est pas une valeur numérique.

L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.



# ORD FONCTION

## BUT

ORD donne le code du premier caractère de la chaîne paramètre.

## SYNTAXE

ORD (<expression chaîne>)

## EXPLICATION

ORD donne le code du premier caractère de la chaîne donnée comme paramètre.

Ainsi :

ORD("A") donne 65
ORD("a") donne 97

Dans le cas particulier d'une chaîne vide, ORD rend la valeur 0.

## EXEMPLE

```
100 INPUT a$
110 IF ORD(a$) < 65 THEN
120   PRINT "Le premier caractère n'est pas lettre"
130   END IF
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

## REMARQUE

Voir l'instruction CHR$ qui produit une chaîne à partir d'un code de caractère.

# PATH INSTRUCTION

## BUT

PATH indique un ou plusieurs chemins pour la recherche de fichiers.

## SYNTAXE

PATH "<chemin> [ ; chemin ] ..."

Où <chemin> doit satisfaire à la syntaxe imposée par le système d'exploitation. <chemin> peut contenir une spécification de disque.

## EXPLICATION

Habituellement, la recherche d'un fichier sans spécification de répertoire ne se fait que dans le répertoire courant, répertoire qui peut être changé au moyen de l'instruction CHDIR.

Cependant l'instruction PATH de MEMBASIC permet, dans le cas où un fichier n'est pas trouvé dans le répertoire courant, de poursuivre la recherche dans les répertoires dont le chemin figure dans la liste.

La recherche se fait en respectant l'ordre des chemins de la liste.

Dès qu'un fichier est trouvé, la recherche est arrêtée.

Les opérations concernées par cette recherche grâce à la liste de chemins fournie par PATH sont :

- Le chargement d'un programme MEMBASIC par LOAD
- L'exécution d'un programme MEMBASIC par RUN
- L'ouverture des masques MEMSCREEN
- La lecture des fichiers d'AIDES (voir Chap. 1)
- La lecture des fichiers de séquences claviers
- La lecture des fichiers PRODOS.

Toutes les opérations d'écriture ou de mise à jour n'utilisent pas les chemins de recherche de PATH.



Rappelons également qu'il est possible de spécifier explicitement pour chaque fichier dans son nom le chemin désiré, même pour une sauvegarde ou une ouverture en vue d'une modification.

## EXEMPLE

```
110 PATH "C:/COMPTA/STE1; A:/PIERRE/TRAVAIL"
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas une chaîne.

# PATH$ FONCTION

## BUT

Donne la liste des chemins enregistrée par PATH.

## SYNTAXE

PATH$

## EXPLICATION

La liste de chemins enregistrée grâce à l'instruction PATH peut être relue. La fonction sans paramètres PATH$ produit une chaîne de caractères contenant la liste de chemins. Une chaîne vide indique qu'aucun chemin n'a été enregistré.

## EXEMPLE

```
100 PRINT "Les chemins de recherche sont : " ;
110 PRINT PATH$
```

## ERREUR

Néant.



# PI FONCTION

## BUT

Cette fonction sans paramètre donne la valeur du nombre (PI).

## SYNTAXE

PI

## EXPLICATION

La valeur de PI est donnée avec 14 chiffres significatifs.

## EXEMPLE

```
PRINT PI
3.1415926535898
```

## ERREUR

Néant.

# POINTER #  INSTRUCTION

## BUT

POINTER # permet d'indiquer le numéro d'enregistrement dans un fichier PRODOS utilisé en mode direct.

## SYNTAXE

POINTER # <numéro de fichier> , <numéro d'enreg.>

<numéro de fichier> est une expression numérique entière dont la valeur sert à désigner le fichier concerné.

<numéro d'enregistrement> est une expression numérique entière dont la valeur indique un numéro d'enregistrement.

## EXPLICATION

Le fichier dont on a précisé le numéro doit être un fichier PRODOS ouvert en mode "R" (accès direct). L'instruction OPEN d'ouverture peut alors contenir une indication de taille d'enregistrement (qui implicitement vaut 128).

Le numéro d'enregistrement moins 1, multiplié par la longueur d'enregistrement, donne une position exacte dans le fichier. Le rôle de l'instruction POINTER # est d'y placer la "position courante" dans le fichier. Les prochaines lectures ou écritures se feront alors à partir de cet endroit.

Il est possible de placer la "position courante" après la fin du fichier. Dans ce cas, toute écriture provoquera une extension du fichier pour rendre l'écriture possible. Il n'y a, par contre, aucune garantie quand au contenu des enregistrements créés pour compléter. Si une tentative de lecture est faite après avoir positionné la position courante après la fin de fichier, une erreur sera détectée.

Les lectures à partir de la "position courante" se feront grâce aux instructions INPUT #, INPUT$, et LINE INPUT #.



Les écritures à partir de la "position courante" se feront grâce aux instructions PRINT # ou WRITE #.

## EXEMPLES

```
100 OPEN "R",1,"fiches",72 ! ouvre le fichier
110 POINTER #1,5 ! position courante au 5ème enreg.
120              ! soit : 72*4 ème octet
130 LINE INPUT A$
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier ouvert.

L'erreur No 28514 est détectée si le fichier n'a pas été ouvert avec l'option "R" (voir OPEN).

# POS FONCTION

## BUT

POS indique la position d'une chaîne dans une autre.

## SYNTAXE

POS ( <expression chaîne>,<expression chaîne> )
ou
POS ( <express. chaîne>,<express. chaîne>,<index> )

## EXPLICATION

Soient A$ et B$ les valeurs obtenues par évaluation des deux premiers paramètres. POS donne le rang (compté à partir de 1) du premier caractère de A$ à partir duquel B$ est contenu.

Si B$ n'est pas contenu dans A$, POS renvoie la valeur zéro.

Par convention, POS(A$,"") renvoie 1.

Si le paramètre <index> est présent, la recherche ne se fait plus à partir du début de A$, mais à partir de la position indiquée par la valeur numérique de l'index. Le paramètre <index> ne doit pas prendre de valeur négative.

## EXEMPLES

```
POS ("BELLE MARQUISE","LLE") donne 3
POS ("BELLE MARQUISE","DUC") donne 0
POS ("AB CD X CD","CD",5)    donne 9
```

## ERREURS

L'erreur No 25102 est détectée si les deux premiers paramètres ne sont pas tous deux du type chaîne de caractères.

L'erreur No 25101 est détectée si le troisième paramètre est présent et n'est pas du type numérique.



L'erreur No 1902 est détectée si l'index dépasse la valeur entière 32767.

L'erreur No 4901 est détectée si l'index est négatif.

# PRINT INSTRUCTION

## BUT

PRINT permet d'afficher des informations à l'écran ou sur imprimante.

## SYNTAXE

PRINT <liste d'édition>

La liste d'édition est composée d'éléments d'éditions séparés les uns des autres par une virgule ou un point virgule.

Exemple : PRINT A;B,X+3;" FIN"

Les éléments d'édition peuvent être des expressions ou des références à la fonction TAB.

## EXPLICATION

L'exécution de cette instruction consiste à produire une chaîne de caractères terminée par une fin de ligne et à envoyer cette chaîne vers le périphérique d'édition en cours (écran ou imprimante).

Le périphérique d'édition peut être sélectionné au préalable par l'instruction PRINTER.

La chaîne à envoyer est produite par évaluation successive de chaque élément dans l'ordre de gauche à droite.

**Valeurs Numériques :**

Les valeurs numériques sont évaluées afin de produire une chaîne commençant par un espace si la valeur est positive et par le signe - si la valeur est négative et suivi par la représentation décimale de la valeur absolue, puis par un espace final.

Les valeurs décimales sont représentées avec le point décimal flottant et selon les cas (valeur très grande ou très petite) un exposant peut être utilisé.



Le champ d'édition doit avoir une longueur supérieure ou égale à 23 (ce qui est sa valeur implicite) pour que toutes les valeurs numériques puissent être éditées avec 14 chiffres significatifs.

Les valeurs inférieures en valeur absolue à 1 sont éditées sans caractère "0" avant le point décimal.

**Valeurs chaînes :**

Les chaînes sont éditées avec tous les caractères qui la composent.

**Rôle des séparateurs et de la fonction TAB :**

La ligne d'impression finale est supposée être une chaîne de caractères dont le premier a le rang 1 et le dernier la valeur de MARGIN.

Un numéro de colonne courante est attribué à PRINT.

Au départ ce numéro est égal à 1. Chaque caractère édité provoque l'incrémentation de 1 de ce numéro de colonne.

Chaque "RETOUR CHARIOT" redonne la valeur 1 à ce numéro de colonne.

MARGIN représente la plus grande valeur possible pour le numéro de colonne. La valeur implicite est 132. Une autre valeur peut lui être affectée au moyen de l'instruction SET MARGIN (voir cette instruction).

ZONEWIDTH représente la longueur du champ pour les valeurs numériques. Sa valeur implicite est 23. Elle peut être modifiée au moyen de l'instruction SET ZONEWIDTH (voir l'instruction SET à ce sujet).

Normalement l'édition de 2 valeurs numériques consécutives, séparées dans la liste de sortie par une virgule se fait dans deux champs consécutifs, chacun ayant la longueur donnée à ZONEWIDTH.

La fonction TAB permet, avant l'édition de la valeur qui la suit, d'affecter la valeur de son paramètre ou numéro de colonne selon les modalités suivantes :

- si la valeur du paramètre de TAB dépasse MARGIN, la valeur attribuée est MOD(valeur-1, MARGIN)+1.

- si la valeur du paramètre est négative ou nulle, la valeur prise est 1.

- dans les autres cas, la valeur du paramètre est gardée.

Si le numéro de colonne courant est inférieur à celui du paramètre de TAB: il y a insertion d'espaces jusqu'au caractère de rang (n-1) inclus et positionnement au caractère de rang n.

Si le numéro de colonne a une valeur supérieure, alors il y a passage à la ligne suivante qui commencera par n-1 espaces.

Un séparateur "virgule" fait passer au champ d'édition suivant par adjonction d'espaces. Mais si le numéro de colonne est déjà dans le dernier champ, alors il provoque un passage à la ligne et le positionnement sur le premier caractère du premier champ de la ligne suivante.

Le séparateur "point virgule" provoque l'édition de la valeur suivante sans passage au champ suivant.

PRINT avec liste vide réalise un interligne.

## EXEMPLE

```
PRINT "Prix " , prix ; TAB(30) ; "Remise " , remise
```

## REMARQUE

L'instruction PRINT USING permet d'obtenir des éditions formatées.



# PRINT # INSTRUCTION

## BUT

PRINT # permet d'écrire des informations dans un fichier PRODOS ou un fichier de communication.

## SYNTAXE

PRINT #<numéro de fichier> , <liste d'édition>

Le numéro de fichier est une expression numérique entière donnant le numéro du fichier PRODOS ou communication vers lequel on désire écrire.

La liste d'édition est composée d'éléments d'édition séparés les uns des autres par une virgule ou un point virgule.

Exemple : PRINT #1, A;B,X+3;" FIN"

Les éléments d'édition peuvent être des expressions ou des références à la fonction TAB ou SPC.

## EXPLICATION

L'instruction PRINT # fonctionne comme l'instruction PRINT, mais les informations sont envoyées vers le fichier PRODOS ou de communication spécifié et non sur l'écran ou l'imprimante standard.

Le fichier PRODOS ou communication utilisé doit avoir été ouvert avec l'instruction OPEN (ou OPEN "COM...").

Si le fichier est un fichier PRODOS, il ne doit pas avoir été ouvert avec le mode "I" qui ne permet pas l'écriture.

La fonction PRINT # n'écrit pas de séparateur entre les données. La relecture par INPUT # ne permettra donc pas d'identifier les différents champs de données. Pour isoler les champs, il faut utiliser WRITE # au lieu de PRINT #.

Par exemple PRINT #1,"test","fichier"

envoie sur disque ..... test fichier

avec WRITE #, on obtient .. "test","fichier"

En revanche, si la chaîne envoyée contient une virgule, INPUT # la considérera comme un séparateur.

En fin de PRINT #, si la liste d'édition ne se termine pas par ";", une "fin de ligne" est envoyée.

La fonction INPUT$ permet de relire les enregistrements écrits par PRINT # lorsque l'on en connaît le format. Cela évite de prendre sur le fichier la place des séparateurs de champs.

## EXEMPLE

```
100 OPEN "O",1,"TEST"
110 FOR i=0 TO 10
120     PRINT #1,i
130 NEXT
140 CLOSE
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier PRODOS ou communication ouvert.

L'erreur No 28514 est détectée si le fichier a été ouvert en mode lecture seule ("I").

## REMARQUES

Voir également WRITE # qui est plus adapté aux écritures d'enregistrements à champs que PRINT #.

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications série.



## CONSEIL

Utilisez de préférence ensemble les couples :

```
   PRINT # ... LINE INPUT #
ou WRITE # ... INPUT #
ou PRINT # ... INPUT$
```

pour retrouver les données à la lecture telles qu'elles étaient à l'écriture.

# PRINT USING INSTRUCTION

## BUT

Permet d'obtenir des éditions formatées sur écran ou imprimante.

## SYNTAXE

```
                <expression chaîne>
PRINT USING                        : <liste de sortie>[;]
                <ligne image>
```

où <ligne image> est un numéro de ligne ou une étiquette. La ligne correspondante contient une instruction IMAGE définissant le format.

<expression chaîne> doit représenter une liste de formats.

<liste de sortie> représente une suite d'expressions séparées par des virgules.

Le caractère ';' en fin d'instruction permet d'éviter un retour à la ligne automatique.

## EXPLICATION

PRINT USING sert à obtenir des éditions formatées sur le périphérique de sortie en cours ( sélectionné par l'instruction PRINTER ). Le format peut être fourni sous deux formes:

- Une ligne du programme a été reservée à cet effet et elle contient une instruction IMAGE suivie du format sous forme de constante chaîne. L'instruction IMAGE doit être seule sur la ligne. Il est inutile que l'exécution du programme passe par les instructions IMAGE pour qu'elles soient utilisables. Cependant, le passage du programme sur une ligne IMAGE ne provoquera pas d'erreur, mais un passage à l'instruction suivante comme dans le cas d'une remarque.

- Une expression chaîne est spécifiée qui contient le format.



Dans les deux cas, le format à spécifier obéit à la même syntaxe.

Le format indiqué contient aussi bien des textes fixes que des zones destinées à recevoir les expressions. Si le nombre de zones est inférieur au nombre d'expressions à éditer, le format sera repris depuis le début après passage à la ligne pour l'édition des zones supplémentaires après passage à la ligne.

Les descriptions de zones suivent les règles suivantes:

**Expressions numériques:**

Les valeurs numériques sont arrondies au plus proche selon le nombre de décimales demandé.

Une valeur numérique contient jusqu'à trois champs différents : la partie entière, la partie décimale, l'exposant.
Le champ exposant doit être spécifié dans le format pour que la valeur numérique soit éditée sous la forme mathématique XXXX.XXX E+YY. On note l'exposant dans le format par "^^^^". Le nombre de "^" indique la place resérvée à l'exposant qui ne doit pas être inférieure à 3 ( 1 pour "E", 1 pour le signe, 1 chiffre). Les champs représentant les parties entières et décimales seront figurés avec des caractères '#', '%' ou '*' (un seul caractère générique par champ). S'il y a une partie décimale, elle sera séparée de la partie entière par un '.' dans le format.

Les "0" inutiles à gauche de la partie entière ne seront écrits que s'ils remplacent des caractères "%" du format, sauf dans le cas d'une valeur à afficher nulle. Dans ce dernier cas un "0" au moins sera affiché. Si un "0" inutile à gauche correspond à un caractère "*" dans le format, "*" sera affiché en lieu et place du "0".

Il est également possible de rendre obligatoire l'édition du signe ainsi que le caractère "$".

Le tableau suivant donne les caractères générés en fonction des combinaisons de caractères "$" et "-" du format qui seront placés à la place des "0" inutiles à gauche:

| début du format | | | Généré | |
|---|---|---|---|---|
| Premiers | Dernier | Exemple | si>=0 | si<0 |
| - | $ | -$ | " $" | "-$" |
| $ | - | $$$- | "$ " | "$-" |
| - | rien | | " " | "-" |
| + | $ | +++$ | "+$" | "-$" |
| $ | + | $$$+ | "$+" | "$-" |
| + | rien | ++++ | "+" | "-" |
| $ | rien | $$$$ | "$" | "$-" |
| rien | rien | rien | "" | "-" |

Expressions chaînes :

Les expressions chaînes sont cadrées à droite si le premier caractère du format est ">"; elles sont cadrées à gauche si le premier caractère du format est "<"; centrées dans les autres cas.

Si l'expression doit être centrée et si le nombre d'espaces est impair, l'espace restant après partage est mis à droite.

Les autres caractères de la chaîne seront notés dans le format par des caractères "#".

## EXEMPLES

```
100 PRINT USING 200: ville$,numero
...
200 IMAGE :Ville ############# ,code postal ######
210 ! Ville centrée ...

100 f$="Voici un chèque de *******.## du <########"
110 PRINT USING f$: 123 ,"10-10-84", 22 ,"15-10-84"
RUN
Voici un chèque de ****123.00 du 10-10-84
Voici un chèque de *****22.00 du 15-10-84

100 nom$="Paul"
110 PRINT USING 'en-tête': nom$
200 'en-tête' : IMAGE :Cher Monsieur <############
RUN
Cher Monsieur Paul
```



```
100 A$="De notre agence à LOS ANGELES : $******.##"
110 PRINT USING A$: 2342.1
RUN
De notre agence à LOS ANGELES : $**2342.10
```

## ERREURS

L'erreur No 25102 est détectée dans la forme <chaîne format> si l'expression format n'est pas une chaîne.

L'erreur No 8201 est détectée si le format n'est pas correct.

L'erreur No 8202 est détectée si aucun champ d'édition ne peut être trouvé dans le format.

L'erreur No 8902 est détectée si la ligne spécifiée dans la forme <ligne format> ne commence pas par IMAGE.

L'erreur No 10600 est détectée si la ligne spécifiée dans la forme <ligne format> n'existe pas.

## REMARQUE

Voir la fonction USING$ qui rend une chaîne formatée à partir d'une expression.

# PRINTER FONCTION

## BUT

La fonction PRINTER rend le numéro du périphérique de sortie en cours.

## SYNTAXE

PRINTER

## EXPLICATION

Le numéro de périphérique peut prendre les valeurs suivantes :

| | |
|---|---|
| 0 | Fenêtre d'exécution en cours à l'écran |
| 1 | Imprimante en slot 1 |
| 2 | Imprimante en slot 2 |

## EXEMPLES

```
100 IF PRINTER=0 THEN PRINT "J'écris sur l'écran.."
```

## ERREUR

Néant.



# PRINTER INSTRUCTION

## BUT

Cette instruction indique que les instructions DIR, PRINT, LIST et les éditions par masques MEMSCREEN sont destinées à un nouveau périphérique de sortie.

## SYNTAXE

PRINTER <numéro de périphérique>
ou

PRINTER <expression chaîne>

<numéro de périphérique> peut prendre les valeurs suivantes :

| | |
|---|---|
| 0 | Fenêtre d'exécution en cours à l'écran |
| 1 | Imprimante en slot 1 |
| 2 | Imprimante en slot 2 |

<expression chaîne> représente un nom de fichier PRODOS éventuellement précédé du caractère @.

## EXPLICATION

Les instructions PRINT et LIST ainsi que les messages d'erreurs et les éditions MEMSCREEN s'affichent sur le périphérique de sortie en cours sélectionné par l'instruction PRINTER.
Suivant la forme choisie, le périphérique sera l'imprimante ou l'écran, ou un fichier PRODOS.

L'imprimante pourra être connectée soit sur la carte série intégrée, soit sur une carte série compatible Pascal, version 1.1.

Si le périphérique choisi est l'écran, l'affichage sera effectué dans la fenêtre d'exécution de MEMSOFT.

Si le périphérique de sortie est un fichier, les informations à éditer ne seront pas envoyées à une imprimante mais stockées dans le fichier spécifié. Si le fichier existe déjà, les informations seront ajoutées en fin de fichier à la suite du texte déjà existant. Si l'on désire que le fichier soit

réinitialisé, on fera précéder le nom du fichier du caractère @.

La fermeture du fichier (ou de l'imprimante) est effectuée lors du prochain ordre PRINTER (même s'il est effectué sur le même périphérique ou fichier).

## EXEMPLE

```
100 PRINTER 1        ! connecte l'imprimante
110 PRINT "Hello"    ! envoie "Hello" sur imprimante
120 PRINTER 0        ! ferme l'imprimante
130 PRINTER "@TEST"  ! ouvre et vide le fichier TEST
140 DIR              ! met le catalogue dans TEST
150 PRINTER 0        ! ferme le fichier TEST
160 PRINTER "TEST"   ! réouvre le fichier TEST
170 PRINT "Fin"      ! ajoute "Fin" dans TEST
180 PRINTER 0        ! ferme le fichier TEST
```

Note : les lignes 120 et 150 sont inutiles : elles ne sont présentes que pour plus de clarté.

## ERREUR

L'erreur No 4000 est détectée si le numéro de périphérique est erroné.



# PROGRAM COMMANDE/INSTRUCTION

## BUT

La commande PROGRAM donne un nom au programme et définit les variables destinées à recevoir des paramètres au lancement du programme.

## SYNTAXE

PROGRAM <nom> [ ( <paramètre> [, <paramètre> .. ])]

<nom> est une suite de caractères indiquant à titre de commentaire le nom du programme.

Les paramètres sont des noms de variables simples ou des noms de tableaux. Les noms de tableaux doivent être suivis de () (,) ou (,,) en fonction du nombre d'indices : 1 2 ou 3.

## EXPLICATION

Un programme peut être lancé par RUN ou par CHAIN. RUN et CHAIN provoquent:

- la fermeture des fichiers ouverts,
- la préparation éventuelle des données passées en paramètres,
- l'effacement de la table des variables après extraction des éventuels paramètres,
- le chargement, puis l'exécution du programme indiqué.

L'instruction PROGRAM indique les noms des variables qui sont destinées à recevoir les paramètres. Ce sont des noms de variables numériques entières ou flottantes, ou des noms de variables chaînes. Les variables chaînes définies par l'instruction PROGRAM sont automatiquement de taille fixe, leur longueur maximale étant définie par le paramètre passé. Les variables tableaux sont indiquées de façon particulière (nom suivi de () (,) ou (,,)) pour que le contrôle du nombre d'indices puisse être effectué.

L'instruction PROGRAM doit obligatoirement se situer en début de programme, avant toute affectation de variable.

Les seules instructions autorisées avant PROGRAM sont :

- des commentaires,
- OPTION DIM ou OPTION BASE,
- WHEN EXCEPTION ...

Les paramètres sont passés par valeur. La correspondance entre les paramètres et les variables destinées à les recevoir dans le programme chaîné est, suivant la donnée reçue :

- Constante numérique ou variable simple numérique:

  La variable réceptrice est numérique simple, flottante ou entière (entière si la valeur passée ne provoque pas de dépassement).

- Constante chaîne :

  La variable réceptrice est de type chaîne (de nom terminé par $). C'est une chaîne de taille fixe de longueur maximum égale à la taille de la constante chaîne passée en paramètre.

- Variable simple chaîne de taille variable:

  La variable réceptrice est de type chaîne (de nom terminé par $). C'est une chaîne de taille fixe de longueur maximum 50 ou de la taille de la chaine passée en paramètre si celle-ci excède 50.

- Variable simple chaîne de taille fixe :

  La variable réceptrice est de type chaîne (de nom terminé par $). C'est une chaîne de taille fixe de mêmes caractéristiques que celle passée en paramètre.

- Tableaux :

  Les tableaux numériques et les tableaux de chaînes fixes sont passés dans des variables tableaux réceptrices de types équivalents et de même nombre d'indices.



  LES TABLEAUX DE CHAINES DE TAILLES VARIABLES NE PEUVENT PAS ETRE PASSES AU PROGRAMME CHAINE.

Si le nombre de paramètres de l'instruction PROGRAM du programme chaîné est inférieur au nombre de paramètres passés par CHAIN, les données supplémentaires sont perdues.

Si le nombre de paramètres de l'instruction PROGRAM du programme chaîné est supérieur au nombre de paramètres passés par CHAIN, les variables complémentaires sont initialisées à 0 ou vide. (C'est également ce qui se passe pour l'ensemble des paramètres si le second programme est lancé par RUN ou CHAIN sans paramètre.)

## EXEMPLE

```
100 PROGRAM SUITE ( valeur_recue, table(,) )
```

## ERREURS

L'erreur No 25805 est détectée si un paramètre n'est pas du même type que la donnée reçue lorsque cela est nécessaire.

L'erreur No 25806 est détectée si un paramètre n'est pas d'un type compatible avec la donnée reçue.

L'erreur No 25804 est détectée si le nombre de dimensions d'un tableau en paramètre ne correspond pas avec celui du tableau récepteur.

L'erreur No 25803 est détectée si l'instruction PROGRAM est exécutée après une affectation de variable.

# RAD FONCTION

## BUT

Cette fonction convertit un nombre de degrés en radians.

## SYNTAXE

RAD (<expression numérique>)

où <expression numérique> est un nombre exprimé en degrés.

## EXPLICATION

La fonction RAD rend le nombre entré converti en radians.

## EXEMPLE

```
PRINT RAD (180)
3.14159265359
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# RANDOMIZE INSTRUCTION

## BUT

Cette instruction permet de changer le point de départ d'une séquence de nombres pseudo aléatoires.

## SYNTAXE

RANDOMIZE

## EXPLICATION

L'instruction RANDOMIZE s'utilise en association avec la fonction RND.
Elle permet de réinitialiser le générateur de nombres pseudo-aléatoires.

Si le générateur de nombres n'est pas réinitialisé, la fonction RND renvoie la même séquence de nombres à chaque exécution du programme.
Ceci est utile pour tester le bon fonctionnement d'un programme en cours d'élaboration.
Lorsque le programme est au point, ajoutez l'instruction RANDOMIZE en début de programme pour modifier cette séquence.
(Cf fonction RND).

## ERREUR

Néant.

# READ INSTRUCTION

## BUT

L'instruction READ permet de lire des données qui sont incorporées au programme au moyen de l'instruction DATA.

## SYNTAXE

READ <variable> ,<variable>...

READ IF MISSING THEN <numéro ligne> : <variable> [,...]

READ IF MISSING THEN EXIT DO : <variable> [,<variable>]..

READ IF MISSING THEN EXIT FOR : <variable> [,<variable>]..

## EXPLICATION

Cette instruction est utilisée conjointement avec une ou plusieurs instructions DATA.
Tout se passe comme si les diverses données contenues dans les instructions DATA d'une unité de programme constituaient une sorte de fichier "fleuve". Au début de l'exécution du programme, la première instruction READ exécutée fera lire les données contenues dans la première instruction DATA de l'unité (début du fichier fleuve) jusqu'à ce que toutes les variables contenues dans l'instruction READ aient reçues une valeur. Si l'instruction DATA contient trop de données, celles-ci seront mises en réserve pour une lecture ultérieure; si au contraire, elle ne contient pas assez de données, les données de l'instruction DATA suivante seront lues.

Si une instruction READ est exécutée et que les données non encore lues dans les instructions DATA sont en nombre insuffisant pour satisfaire la liste de variables de l'instruction READ, alors deux cas se présentent :

- L'option IF MISSING est présente et alors il y a branchement au numéro de ligne indiquée ou



sortie de la boucle de lecture en cours par EXIT DO ou EXIT FOR,

- L'option est absente et l'erreur No 8001 est détectée.

## EXEMPLE

```
100 READ IF MISSING THEN 500: A,B,C,D
.....
500 PRINT "pas assez de données, modifiez le programme"
510 END
```

## ERREURS

L'erreur No 8001 est détectée s'il n'y a pas assez de données pour satisfaire la liste contenue dans READ.

L'erreur No 8101 est détectée si l'exécution de l'instruction READ conduit à affecter une valeur non numérique à une variable numérique.

L'erreur No 1006 est détectée si READ lit une donnée numérique qui provoque un dépassement de capacité.

l'erreur 1051 est détectée si READ lit une donnée chaîne qui provoque un dépassement de la longueur maximale autorisée pour cette chaîne.

# REM INSTRUCTION

## BUT

L'instruction REM permet d'insérer des commentaires dans un programme.

## SYNTAXE

REM <commentaire>
ou
! <commentaire>

## EXPLICATION

Pour insérer des commentaires dans un programme, on peut utiliser l'instruction REM soit en début de ligne, soit dans une ligne. Dans ce cas REM doit être précédée du séparateur ":".

La forme utilisant le point d'exclamation peut être utilisée comme l'instruction REM ou comme "final" d'une instruction.

Le commentaire est une suite de caractères affichables se terminant à la fin de la ligne. Par contre MEMBASIC n'affiche que la forme avec point d'exclamation.

Une instruction de branchement peut renvoyer vers une ligne contenant un commentaire. Dans ce cas, l'exécution se poursuit à la ligne suivante.

Les programmes sauvegardés sous forme protégée éliminent les commentaires. Donc quels que soient le nombre et la taille des commentaires dans la version "source" du programme, le programme, dans sa version protégée, sera le même.

## EXEMPLE

```
100 a=1 ! Commentaire après une instruction
110 ! Commentaire seul sur une ligne
120 a=1 : REM même chose sous la forme REM
130 REM ligne de commentaire seul sous la forme REM
```



## ERREUR

Néant.

## REMARQUE

Lorsque le programme est relisté, c'est toujours la forme point d'exclamation qui est utilisée.

# REMAINDER FONCTION

## BUT

REMAINDER donne le reste de la division entière des deux paramètres.

## SYNTAXE

REMAINDER(<express.numérique>,<express. numérique>)

## EXPLICATION

Les deux expressions numériques sont évaluées, puis REMAINDER calcule le reste de la division entière de la première valeur divisée par la seconde.

REMAINDER (5.6, 2.1) donne 1.4

## EXEMPLE

```
100 INPUT x,y
120 z=REMAINDER(x,y)
130 PRINT "Reste de la division de X par Y : "; z
```

## ERREUR

L'erreur No 25101 est détectée si l'un des paramètres n'est pas de type numérique.



# RENAME INSTRUCTION

## BUT

L'instruction RENAME permet de renommer le fichier spécifié dans la première expression chaîne en lui donnant le nom proposé dans la deuxième expression chaîne.

## SYNTAXE

RENAME <expression chaîne> TO <expression chaîne>

Les deux expressions chaînes représentent des spécifications de fichier.

## EXPLICATION

Si la deuxième expression ne correspond pas à un fichier déjà existant et si ses spécifications de disque sont compatibles avec celles de la première expression, le nom du fichier est changé.

Les suffixes de l'ancien nom et du nouveau nom doivent être spécifiés explicitement, l'instruction RENAME ne pouvant faire d'hypothèse sur le suffixe à utiliser.

Si les deux spécifications de fichier n'indiquent pas un même répertoire, le fichier ne sera pas seulement renommé mais aussi déplacé.

On peut renommer plusieurs fichiers en même temps en utilisant dans le nom du fichier les caractères spéciaux ? et *.
Un ? dans un nom de fichier indique qu'un caractère quelconque peut occuper cette position.
Un * dans un nom de fichier indique que tout caractère peut occuper cette position ou les suivantes.

Si le nouveau nom est précédé de @, un éventuel fichier existant portant ce nom sera détruit.

Les différents objets utilisés par MEMSOFT peuvent être renommés par cette instruction. La seule précaution à prendre est, dans le cas des fichiers MEMFILE, de renommer simultanément le fichier des clés et le fichier des enregistrements, puisque

chaque fichier MEMFILE se retrouve sous la forme de deux fichiers PRODOS de suffixe .MFK et .MFR.

## EXEMPLES

Renommer le fichier MEMFILE de nom STOCK en STOCK2:

```
RENAME "stock.mfr" TO "stock2.mfr"
RENAME "stock.mfk" TO "stock2.mfk"
```

ou

```
RENAME "stock.mf*" TO "stock2.*"
```

Renommer le programme PRG1 en PRG2 :

```
RENAME "prg1.prg" to "prg2.prg"
```

## ERREURS

L'erreur No 28020 est détectée si le changement de nom est impossible, soit parce qu'un fichier portant le nouveau nom existe déjà et que @ ne précède pas ce nom, soit parce que le fichier d'origine n'existe pas.

L'erreur No 25102 est détectée si l'un des paramètres n'est pas de type chaîne.

## REMARQUE

Si la nouvelle spécification de fichier correspond à un fichier déjà existant que l'on désire supprimer, le détruire avant d'effectuer le changement de nom avec l'instruction KILL :

KILL <spécification de fichier>



# RENUMBER COMMANDE

## BUT

La commande RENUMBER permet de renuméroter les lignes d'un programme.

## SYNTAXE

RENUMBER

## EXPLICATION

RENUMBER entraîne la renumérotation de l'ensemble du programme à partir de 100 avec un pas implicite de 10.

Le pas peut être modifié grâce à l'instruction STEP.

Cette renumérotation est effectuée de telle façon que les références aux numéros de ligne sont rectifiées afin de ne pas modifier l'ordre d'exécution des instructions.

## ERREUR

Néant.

## REMARQUE

Les numéros de ligne inexistants rencontrés dans des instructions du programme sont transformés en un numéro impossible ( supérieur à 65000 ).

# REPEAT$ FONCTION

## BUT

La fonction REPEAT$ permet de construire une chaîne de caractères par répétition de la chaîne donnée comme paramètre.

## SYNTAXE

REPEAT$ (<expression chaîne>,<index>)

## EXPLICATION

Cette fonction construit une chaîne par répétition de la chaîne donnée comme premier caractère.

La valeur de l'index indique le nombre de fois que la chaîne donnée par le premier paramètre doit être répétée.

Si la valeur de l'index est nulle, le résultat obtenu est la chaîne vide.

## EXEMPLE

```
90 ! POUR SOULIGNER UN TITRE
100 a$ = "TITRE"
110 b$ = REPEAT$("-", LEN(a$))
120 PRINT a$
130 PRINT b$
RUN
TITRE
-----
```

## ERREURS

L'erreur No 4901 est détectée si l'index a une valeur négative.

L'erreur No 1051 est détectée si la longueur de la chaîne résultat dépasse 255.

L'erreur No 25102 est détectée si le premier paramètre n'est pas une chaîne.



# REPLAY INSTRUCTION

## BUT

L'instruction REPLAY permet de rejouer une séquence clavier enregistrée.

## SYNTAXE

REPLAY <expression chaîne>

## EXPLICATION

Cette instruction permet de commander par programme la relecture d'une sequence clavier enregistrée.

La séquence a été enregistrée par les touches de fonction "CTRL POMME 3" et "CTRL POMME 4" suivant la méthode décrite au chapitre 1.21.1.

La séquence rejouée a le même effet que l'enfoncement de la suite de touches dont les codes ont été enregistrés dans le fichier.

La séquence ne sera jouée qu'à la prochaine instruction effectuant une saisie de caractère.

Si le fichier spécifié n'existe pas, aucune séquence n'est rejouée.

La recherche du fichier tient compte des chemins indiqués par l'instruction PATH. Le fichier spécifié ne doit pas comporter de suffixe car le suffixe est toujours .AUT pour les séquences clavier.

La touche "CTRL POMME 2" permet d'arrêter la séquence à tout moment.

## EXEMPLE

```
100 INPUT PROMPT "Fichier à rejouer ": f$
110 REPLAY f$ !si le fichier est trouvé ...
120 INPUT a$  !La donnée A$ est lue dans le fichier
130 .....
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

## REMARQUE

Le même effet peut être obtenu manuellement en utilisant la touche de fonction "CTRL POMME 1" (Voir chapitre 1.21.2).



# RESTORE INSTRUCTION

## BUT

RESTORE permet de relire des données au moyen de l'instruction READ.

## SYNTAXE

```
RESTORE
ou
        <numéro de ligne>
RESTORE
        <étiquette>
```

## EXPLICATION

RESTORE sans paramètre, positionne le pointeur de données sur la première instruction DATA du programme.

RESTORE <numéro de ligne> ou <étiquette> positionne le pointeur de données sur la première instruction DATA dont le numéro de ligne est égal ou supérieur à celui de la ligne spécifiée dans l'instruction RESTORE.

## EXEMPLE

```
100 READ a,b,c
110 READ x,y,z
...
400 RESTORE
410 READ
...
900 DATA 1,2,3
910 DATA 4,5,6
920 END
```

## ERREUR

L'erreur No 10600 est détectée si le numéro de ligne ou l'étiquette n'existe pas.

# RETRY INSTRUCTION

## BUT

Cette instruction est utilisée dans la "routine d'anomalie" pour demander une nouvelle exécution de l'instruction qui a provoqué l'erreur.

## SYNTAXE

RETRY

## EXPLICATION

Une routine d'anomalie peut se terminer de différentes manières :

- Retour à l'instruction qui a provoqué l'erreur :

  Utilisation de RETRY.

- Arrêt de l'exécution :

  Utilisation de END.

## EXEMPLE

```
100 WHEN EXCEPTION GOTO 'Obligatoire'
110 INPUT PROMPT " Reponse OBLIGATOIRE " : a
...
500 'Obligatoire'
510 PRINT "POMME CTRL C INTERDIT"
520 RETRY
```

## ERREURS

L'erreur No 29003 est détectée en cas de tentative d'exécution de RETRY sans qu'une erreur ait été détectée.

L'erreur No 29005 est détectée en cas de tentative d'utilisation de RETRY en mode immédiat.



# RIGHT$ FONCTION

## BUT

La fonction RIGHT$ extrait une sous-chaîne à partir des derniers caractères de la chaîne passée en paramètre.

## SYNTAXE

RIGHT$ (<expression chaîne>,<index>)

## EXPLICATION

Les deux paramètres, notés par exemple A$ et I, sont évalués.

La sous-chaîne comportant les I derniers caractères de A$ est extraite et renvoyée.

Si l'index est nul, RIGHT$ donne une chaîne vide.

Si la valeur de l'index est supérieure à la longueur de A$, la chaîne extraite est A$.

## EXEMPLE

```
100 A$ = "MEMBASIC EST FACILE"
110 B$ = RIGHT$(A$,6)
120 PRINT A$
130 PRINT B$
RUN
MEMBASIC EST FACILE
FACILE
```

## ERREURS

L'erreur No 25102 est détectée si le premier paramètre n'est pas une chaîne.

L'erreur No 25101 est détectée si le second paramètre n'est pas numérique.

L'erreur No 4901 est détectée si la valeur de l'index est négative.

REMARQUE

On peut extraire des sous-chaînes selon une autre méthode au moyen d'index (voir Chapitre 1) :

RIGHT$(A$,N) équivaut à A$ ( LEN(A$)-N+1 : LEN(A$))



# RMDIR INSTRUCTION

## BUT

L'instruction RMDIR permet de supprimer un répertoire vide.

## SYNTAXE

RMDIR <expression chaîne>

Où <expression chaîne> doit représenter un chemin vers un répertoire existant.

## EXPLICATION

RMDIR supprime le répertoire indiqué à condition que ce répertoire soit vide, c'est à dire qu'il ne contienne ni fichiers, ni sous-répertoires.

## EXEMPLE

```
500 RMDIR "C:COMPTA/ESSAI"
```

## ERREURS

L'erreur No 28130 est détectée si le répertoire n'est pas vide ou s'il n'existe pas.

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

# RND FONCTION

## BUT

Cette fonction donne le prochain nombre pseudo-aléatoire dans une séquence prédéfinie entre 0 et 1.

## SYNTAXE

RND

## EXPLICATION

RND renvoie le prochain nombre pseudo-aléatoire dans une séquence de nombres pseudo-aléatoires uniformément répartis dans l'intervalle [0,1[.

Si aucune instruction RANDOMIZE n'est exécutée, la fonction RND génère toujours la même séquence de nombres pseudo-aléatoires à chaque exécution d'un programme.

Si une instruction RANDOMIZE a été exécutée, le point de départ de la séquence de nombres est changé de façon non prévisible puis utilisé pour la liste de nombres générés par RND.

Pour obtenir des nombres pseudo-aléatoires compris entre 0 et n, utilisez la formule :

```
INT (RND*(n+1))
```

## EXEMPLE

```
100 ! Tirage simulant 10 lancers de 6 dés
110 RANDOMIZE
120 FOR I = 1 TO 10
130     FOR J = 1 TO 6
140         LANCER = INT(RND*6) + 1
150         PRINT LANCER;"  ";
160     NEXT : PRINT
170 NEXT
```



## ERREUR

Néant.

# ROUND FONCTION

## BUT

ROUND effectue un arrondi sur la position souhaitée.

## SYNTAXE

ROUND (<expression numérique>,<index>)

## EXPLICATION

L'expression numérique est évaluée et l'arrondi est effectué au plus proche sur la Nième décimale, N étant le résultat de l'évaluation de l'index. Cette valeur doit être positive.

ROUND(3.1415926,4) donne 3.1416
ROUND(3.1415926,0) donne 3

## EXEMPLE

```
100 INPUT PROMPT"NOMBRE DE DECIMALES A GARDER ":n
110 IF n<0 THEN PRINT "ERREUR": GOTO 100
120 y = ROUND(x,n)
```

## ERREUR

L'erreur No 25101 est détectée si l'un des deux paramètres ne sont pas de type numérique.



# RTRIM$ FONCTION

## BUT

Cette fonction supprime les espaces situés à droite de la chaîne passée en paramètre.

## SYNTAXE

RTRIM$ (<expression chaîne>)

## EXPLICATION

Le paramètre est évalué. RTRIM$ restitue une chaîne extraite du paramètre par suppression des espaces situés à la fin du paramètre. Par contre les espaces situés à l'intérieur de la chaîne sont gardés.

## EXEMPLE

```
100 A$ = "RTRIM$ EST UTILE                    "
110 B$ = " POUR SUPPRIMER LES ESPACES A DROITE"
120 PRINT RTRIM$(A$) ; B$
RUN
RTRIM$ EST UTILE POUR SUPPRIMER LES ESPACES A
DROITE
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.

## REMARQUE

Voir la fonction LTRIM$ pour supprimer les espaces en début de chaîne.

# RUN COMMANDE/INSTRUCTION

## BUT

La commande RUN lance l'exécution du programme en mémoire ou d'un programme préalablement chargé depuis la mémoire auxiliaire.

## SYNTAXE

RUN <spécification de fichier>
ou
RUN <numéro de ligne>
ou
RUN <étiquette>
ou
RUN

## EXPLICATION

La forme RUN provoque :

- la fermeture des fichiers qui étaient éventuellement ouverts.
- le lancement de l'exécution du programme en mémoire à partir du début.
- l'effacement de la table des variables.

La forme RUN <numéro de ligne> agit comme précédemment à ceci près que l'exécution débute à la ligne indiquée.

La forme RUN <spécification de fichier> provoque :

- la fermeture des fichiers qui étaient éventuellement ouverts.
- l'effacement de la table des variables.
- le chargement, puis l'exécution du programme indiqué.

Dans la forme RUN <spécification de fichier> le paramètre est une expression chaîne contenant le nom et éventuellement le chemin d'accès au programme. Le suffixe implicite utilisé, si la spécification de fichier n'en fournit pas, sera .PRG. Dans le cas



où aucun chemin particulier n'est proposé dans la spécification de fichier, les différents chemins enregistrés grâce à l'instruction PATH seront utilisés successivement (voir cette instruction pour plus de détails). Enfin si le programme ne se trouve dans aucun des chemins spécifiés, il sera cherché dans le répertoire particulier /MEMTOOLS.

Bien que RUN soit habituellement utilisé comme une commande, RUN peut être utilisé à l'intérieur d'un programme, ce qui permet d'enchaîner les programmes entre eux. Dans ce cas, si le programme n'est pas trouvé, le programme de départ se poursuit à l'instruction suivant le RUN.

## EXEMPLE

```
RUN
RUN 100
RUN "gestion"
RUN 'étiquette'
```

## ERREURS

L'erreur No 10600 est détectée si la ligne spécifiée n'existe pas dans la forme RUN <numéro de ligne> ou RUN <étiquette>.

L'erreur No 28010 est détectée si le programme spécifié n'a pas été trouvé dans la forme RUN <spécification de fichier>.

L'erreur No 25801 est détectée si l'on tente de lancer un programme protégé par RUN <étiquette> ou RUN <numéro de ligne>.

## REMARQUE

Voir l'instruction CHAIN qui remplace avantageusement la forme RUN <spécification de fichier> puisqu'elle permet le passage de données d'un programme à un autre.

# SAVE COMMANDE/INSTRUCTION

## BUT

La commande SAVE sauvegarde un programme sur disque dur ou sur disquette.

## SYNTAXE

SAVE "[@] [$] <spécification de fichier>"

## EXPLICATION

Le programme MEMBASIC résident en mémoire sera sauvegardé sur mémoire auxiliaire, soit sous le répertoire courant, soit sous le répertoire indiqué.

Trois modes de sauvegarde d'un programme sont possibles : le mode texte qui prend plus de place sur disque, qui est plus long à sauvegarder et recharger mais qui est compatible avec les éditeurs de textes du système d'exploitation; le mode "normal" que seul MEMBASIC comprend et qui est moins encombrant et plus rapide à recharger et enfin le mode "compacté et protégé" qui est le moins encombrant et encore plus rapide à charger.

La forme SAVE "spécification de fichier" entraîne la sauvegarde du programme en mémoire centrale vers une mémoire auxiliaire. Le programme ainsi sauvegardé pourra être relu par MEMBASIC mais pas par un éditeur de textes.

Si la forme SAVE "$spécification de fichier" est utilisée, alors le programme est stocké sous une forme compactée et protégée qui n'autorisera plus ni modification, ni affichage, ni TRACE. Cette forme est beaucoup moins encombrante car tous les commentaires, espaces d'indentation, étiquettes et numéros de ligne sont supprimés. Sous cette forme, toute détection d'erreur ne peut plus indiquer le numéro de ligne où l'erreur s'est produite. La fonction EXLINE rend l'adresse de l'instruction ayant provoqué l'erreur (cf LOCNUMBER).
Dans ce cas, MEMBASIC affiche un message prévenant l'utilisateur que le programme stocké est protégé.



La forme SAVE "@ [$] spécification de fichier" permet la sauvegarde sous le même nom qu'une version antérieure, le $ jouant le même rôle de compactage et protection que précédemment.

En l'absence de suffixe, MEMBASIC ajoute le suffixe .PRG.

## EXEMPLE 1

```
SAVE "ESSAI.BAS" ! Première sauvegarde
```

## EXEMPLE 2

```
SAVE "$TEST"          ! Sauvegarde avec protection
```

Le suffixe .PRG est ajouté et le fichier stocké n'est plus listable.

## EXEMPLE 3

```
SAVE "@C:SOURCE/ANCIEN"  ! Remplace ANCIEN.PRG
```

Le programme est sauvegardé sur le disque C dans le répertoire SOURCE avec le nom ANCIEN.PRG sous forme non protégée.

## ERREURS

L'erreur No 28110 est détectée si le répertoire ne peut pas être atteint.

L'erreur No 28030 est détectée si le fichier existe déjà et que le caractère "@" a été omis.

## REMARQUES

Voir l'instruction LOAD qui charge un programme.

Voir également la commande LIST qui permet de sauvegarder tout ou partie d'un programme sur disque sous forme d'un fichier texte.

## CONSEIL

N'utiliser la forme protégée que pour des programmes au point et garder toujours séparément au préalable une version "source".
Pour un programme qui sera exploité sous sa forme protégée, il n'y a pas lieu de se préoccuper de l'encombrement engendré par les commentaires, les étiquettes et la forme du programme (IF monoligne et IF multilignes donneront le même encombrement).



# SEC FONCTION

## BUT

Cette fonction rend la sécante d'un angle.

## SYNTAXE

SEC (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat obtenu est celui de l'opération :

$$1 / \cos$$

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT SEC (0)
RUN
 1
```

## ERREURS

L'erreur No 3091 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# SELECT INSTRUCTION

## BUT

Cette instruction permet de démarrer une série de "bloc case" et d'évaluer l'expression dont la valeur sera utilisée pour les tests.

## SYNTAXE

SELECT CASE <expression>

## EXPLICATION

L'expression, qui peut être de type numérique ou chaîne, est évaluée, puis la valeur ainsi obtenue est utilisée pour sélectionner le "bloc case" qui sera exécuté.

Voir l'instruction CASE pour des explications complémentaires.



# SET INSTRUCTION

## BUT

SET permet de fixer un certain nombre de paramètres utilisés par MEMSOFT :

MARGIN : la longueur de la ligne,

ZONEWIDTH : longueur du champ d'édition,

CURLINE : ligne d'édition pour MEMSCREEN,

MAXLINE : Nombre de lignes par page d'imprimante.

## SYNTAXE

```
          MARGIN
SET       ZONEWIDTH    <index>
          CURLINE
          MAXLINE
```

<index> est une expression numérique dont la valeur est arrondie à l'entier le plus proche.

## EXPLICATION

SET MARGIN permet d'indiquer à MEMBASIC la longueur en caractères de la ligne, SET ZONEWIDTH celle du champ pour les instructions de sortie non formatées PRINT.

SET CURLINE permet d'indiquer à MEMSCREEN le numéro de ligne initial dans les impressions, SET MAXLINE le nombre de lignes de l'imprimante.

Cette instruction prend effet immédiatement même si une ligne était en cours d'affichage.

La valeur implicite de ZONEWIDTH est de 23 caractères. Celle de MARGIN est 132.

La valeur implicite de MAXLINE est 66.

CURLINE n'a pas de valeur implicite, mais une valeur de départ qui est 0.

## EXEMPLE

```
100 SET MARGIN 132
110 SET ZONEWIDTH 15
120 SET MAXLINE 60
```

## ERREURS

L'erreur No 4901 est détectée si la valeur de l'index est négative pour les paramètres MAXLINE et CURLINE.

L'erreur 4006 est détectée si l'on tente de rendre MARGIN inférieur à ZONEWIDTH.

L'erreur 4007 est détectée si l'on tente de rendre ZONEWIDTH supérieur à MARGIN.

L'erreur 25101 est détectée si le paramètre n'est pas numérique.

## REMARQUES

L'instruction ASK permet de connaître la valeur des deux paramètres d'édition MARGIN et ZONEWIDTH.
Pour plus de détails sur ces deux paramètres, voir l'instruction PRINT qui les utilise.

Pour plus de détails sur CURLINE et MAXLINE voir les fonctions CURLINE et MAXLINE.



# SGN FONCTION

## BUT

SGN renvoie -1, 0 ou 1 selon que l'expression donnée comme paramètre est négative, nulle ou positive.

## SYNTAXE

SGN (<expression numérique>)

## EXPLICATION

SGN renvoie la valeur -1 si la valeur du paramètre est négative.

SGN renvoie la valeur 0 si cette valeur est nulle.

SGN renvoie la valeur +1 si cette valeur est positive.

## EXEMPLE

```
200 ON SGN(marge)+2 GOTO 'perte','zéro','bénéfice'
```

## ERREUR

L'erreur No 25101 est détectée si le paramètre n'est pas de type numérique.

# SIN FONCTION

## BUT

Cette fonction rend le sinus d'un angle.

## SYNTAXE

SIN (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat est une valeur comprise dans l'intervalle [-1,1].

## EXEMPLE

```
100 OPTION ANGLE DEGREES
110 PRINT SIN (45)
RUN
.707106781187
```

## ERREURS

L'erreur No 3090 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.



# SIZE FONCTION

## BUT

SIZE donne la taille d'un tableau ou de chacun de ses indices.

## SYNTAXE

SIZE ( <nom de tableau> )
ou
SIZE ( <nom de tableau> ,<index> )

## EXPLICATION

SIZE (<nom de tableau>) donne le nombre total d'éléments que peut comporter le tableau.

SIZE (<nom de tableau>,<index>) donne le nombre de valeurs que peut prendre l'indice indiqué par <index>.

## EXEMPLE

```
100 DIM a(-3 TO 5, 10 TO 14)

     SIZE(a)   donne 45
     SIZE(a,1) donne  9
     SIZE(a,2) donne  5
```

## ERREURS

L'erreur No 4004 est détectée si la valeur de l'index est inférieure à 1 ou supérieure au nombre de dimensions du tableau.

L'erreur No 25005 si le premier paramètre n'est pas un nom de tableau.

L'erreur No 25101 si l'index n'est pas de type numérique.

# SKIP INSTRUCTION

## BUT

Permettre un branchement rapide vers la fin d'une boucle FOR ou DO pour passer à l'élément suivant.

## SYNTAXE

```
        DO
SKIP
        FOR
```

## EXPLICATION

L'instruction SKIP est utilisée pour passer directement à l'instruction terminale d'une boucle FOR (qui est NEXT) ou DO (qui est LOOP).

Cette instruction joue donc le rôle de l'instruction GOTO mais offre une meilleure lisibilité.

```
FOR I=...
.
.
.
SKIP FOR
.
.
.
NEXT I
```

```
DO
.
.
.
SKIP DO
.
.
.
LOOP
```

## EXEMPLE 1

```
FOR I...
   .
   .
  IF...THEN SKIP FOR
   .
   .
NEXT I
```



## EXEMPLE 2

```
DO...
   .
   .
  IF...THEN SKIP DO
   .
   .
LOOP UNTIL...
```

## ERREURS

L'erreur No 10702 est détectée si SKIP DO est rencontrée alors qu'aucun bloc DO n'est actif.

L'erreur No 10903 est détectée si SKIP FOR est rencontrée alors qu'aucun bloc FOR n'est actif.

L'erreur No 10904 est détectée si SKIP FOR est rencontrée alors qu'aucun NEXT ne correspond au bloc FOR actif.

# SPC FONCTION

## BUT

SPC est une fonction particulière qui ne peut être utilisée qu'avec l'instruction PRINT et qui permet d'imprimer une suite d'espaces.

## SYNTAXE

SPC (<index>)

<index> représente le nombre d'espaces à imprimer.

## EXPLICATION

SPC est utilisé exclusivement par PRINT.

## EXEMPLE

```
100 FOR i=1 TO 20
110 PRINT SPC(i);"*"
120 NEXT i
```

## ERREUR

L'erreur No 25101 est détectée si l'index n'est pas numérique.

L'erreur No 4000 est détectée si l'index est hors limites.



# SQR FONCTION

## BUT

Cette fonction rend la racine carrée d'un nombre.

## SYNTAXE

SQR (<expression numérique>)

où <expression numérique> est un nombre supérieur ou égal à 0.

## ERREURS

L'erreur No 3005 est détectée si <expression numérique> est négatif.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# STATUS FONCTION

## BUT

La fonction STATUS donne le résultat de la dernière opération d'Entrée/Sortie effectuée sur fichier ou masque.

## SYNTAXE

STATUS

## EXPLICATION

Cette fonction sans paramètre donne une valeur numérique correspondant à un numéro d'erreur récupérable détectée en utilisant certaines instructions MEMSCREEN ou MEMFILE ou sur les fichiers PRODOS.

La valeur zéro correspond à une fin normale sans anomalies.

Pour plus de détails, voir les sections III "MEMSCREEN" et IV "MEMFILE".



# STEP COMMANDE

## BUT

STEP permet de modifier la valeur du pas implici utilisé dans les instructions AUTO et RENUMBER.

## SYNTAXE

STEP <index>

<index> est une expression numérique entière pos tive.

## EXPLICATION

La nouvelle valeur du pas est enregistrée. El sera utilisée en lieu et place de 10 dans l commandes AUTO et RENUMBER.

## ERREURS

L'erreur No 1902 est détectée si le paramètre e trop grand.

L'erreur No 4000 est détectée si le paramètre e négatif ou nul.

# STR$ FONCTION

## BUT

STR$ convertit une valeur numérique en chaîne de caractères.

## SYNTAXE

STR$ (<expression numérique>)

## EXPLICATION

La fonction STR$ convertit <expression numérique> en chaîne de caractères.

## ERREUR

L'erreur 25101 est détectée si le paramètre n'est pas de type numérique.

## REMARQUE

La fontion VAL est l'inverse de la fonction STR$.



# SYSTEM INSTRUCTION

## BUT

SYSTEM fait quitter MEMBASIC pour revenir au niveau système d'exploitation.

## SYNTAXE

SYSTEM

## EXPLICATION

SYSTEM fait revenir au niveau système d'exploitation, par exemple au bureau.

## EXEMPLE

```
500 IF r$ = "OK" THEN SYSTEM
```

## ERREUR

Néant.

# TAB FONCTION

## BUT

TAB est une fonction particulière qui ne peut être utilisée qu'avec l'instruction PRINT et qui permet de positionner le curseur ou la tête d'impression à la colonne indiquée.

## SYNTAXE

TAB (<index>)

## EXPLICATION

TAB est utilisé exclusivement par PRINT.

Si la valeur numérique de l'index est inférieure à 1, on prend 1.

Si la valeur est supérieure à MARGIN, on prend pour valeur MOD (<index>-1,MARGIN)+1 après passage à la ligne.

## EXEMPLE

```
100 FOR i=1 TO 20
110 PRINT TAB(i);"*"
120 NEXT i
```

## ERREUR

L'erreur No 25101 est détectée si l'index n'est pas numérique.



# TAN FONCTION

## BUT

Cette fonction rend la tangente d'un angle.

## SYNTAXE

TAN (<expression numérique>)

où <expression numérique> est un nombre exprimé en radians ou en degrés suivant l'option de calcul en cours (cf OPTION ANGLE).

## EXPLICATION

Le résultat est celui de la division du sinus par le cosinus.

## EXEMPLE

```
PRINT TAN (0)
0
```

## ERREURS

L'erreur No 3090 est détectée si <expression numérique> est hors limites.

L'erreur No 25101 est détectée si le paramètre n'est pas un numérique.

# TIME FONCTION

## BUT

TIME donne l'heure comptée en secondes.

## SYNTAXE

TIME

## EXPLICATION

Cette fonction sans paramètre donne l'heure sur 24 heures en nombre de secondes,l'heure étant comptée à partir de minuit.

## EXEMPLE

```
100 a=time
110 FOR I=1 TO 1000
 .
 .
 .
200 NEXT
210 PRINT "Je travaille depuis ";(TIME-a);" sec."
```

## ERREUR

Néant.

## REMARQUE

Voir TIME$ qui donne l'heure formatée.



# TIME$ FONCTION

## BUT

TIME$ donne l'heure.

## SYNTAXE

TIME$

## EXPLICATION

Cette fonction sans paramètre donne l'heure sur 24 heures sous la forme :

HH:MM:SS

L'heure étant comptée à partir de minuit.

## EXEMPLE

```
100 PRINT TIME$
110 FOR I=1 TO 1000
 .
 .
 .
200 NEXT
210 PRINT TIME$
```

## ERREUR

Néant.

## REMARQUE

Voir TIME qui donne l'heure en secondes.

# TRACE INSTRUCTION

## BUT

Sélectionne ou enlève le mode TRACE dans lequel les lignes exécutées sont listées.

## SYNTAXE

TRACE ON
ou
TRACE OFF

## EXPLICATION

La forme TRACE ON rend le mode TRACE actif alors que la forme TRACE OFF le rend inactif.

Lorsque le mode TRACE est actif, les lignes sont listées au fur et à mesure de leur exécution. Pour ne pas perturber le fonctionnement normal du programme, les lignes ne sont listées ni dans la fenêtre éditeur, ni dans la fenêtre d'exécution, mais dans une fenêtre réservée à cet effet et nommée fenêtre de TRACE. Cette fenêtre est rendue visible lorsque TRACE ON est exécuté et disparaît avec TRACE OFF.

Les lignes sont listées autant de fois qu'elles contiennent une instruction exécutée. L'instruction elle-même apparaît dans la ligne avec un attribut différent, ce qui permet de l'identifier aisément.

Si la fenêtre de TRACE comporte plus de lignes que ce que l'on peut voir à l'écran, il est possible de consulter ces lignes à tout moment en utilisant l'option DEFILEMENT du mode "manipulation de fenêtres" décrite dans la section I "FENETRE".

TRACE permet également le pas à pas et le défilement rapide.
Lorsque le mode TRACE est actif et que le programme n'attend pas de données au clavier, deux actions sont possibles :



- l'enfoncement de la touche OPTION suspend l'affichage des lignes tracées dans la fenêtre de trace. Cela permet un accès plus rapide à la partie de programme à mettre au point. Les lignes sont à nouveau tracées dès que la touche OPTION est relachée.

- La touche SHIFT suspend l'exécution du programme.
  Tant que l'une des touches SHIFT est enfoncée, l'exécution du programme est arrêtée.

  On peut alors :

  - avancer d'une instruction en enfonçant puis relachant la touche CTRL (sans relacher la touche SHIFT),

  - passer en mode "gestion des fenêtres" en enfonçant la touche "OPTION CLIC-Souris" pour consulter plus facilement la liste des lignes tracées,

  - stopper le programme en enfonçant POMME CTRL C (sauf si le programme comporte une gestion des exceptions).

  Au relachement de la touche SHIFT, le programme reprend son exécution et sa trace normalement.

## ERREUR

Néant.

# TRUNCATE FONCTION

## BUT

Cette fonction effectue une troncature de la valeur donnée comme paramètre.

## SYNTAXE

TRUNCATE (<expression numérique>,<index>)

## EXPLICATION

Les deux expressions sont évaluées et la valeur numérique de la première expression est tronquée selon la valeur de l'index.

La valeur de l'index doit être positive.
La troncature est effectuée sur la Nième décimale (les décimales suivantes sont supprimées).

TRUNCATE(3.333333333,4) donne 3,3333

## EXEMPLE

Pour garder deux décimales, on pourra écrire :

```
...
300 ttc = ht * (1 + taux_de_tva)
310 somme_due = TRUNCATE(ttc,2)
```

## ERREURS

L'erreur No 25101 est détectée si l'un des paramètres n'est pas de type numérique.

L'erreur No 1902 est détectée si l'index prend une valeur hors limites.



# UBOUND FONCTION

## BUT

Cette fonction rend la valeur maximale d'un indice.

## SYNTAXE

UBOUND (<nom de tableau>,<index>)
ou
UBOUND (<nom de vecteur>)

## EXPLICATION

La fonction UBOUND donne la plus grande valeur que peut prendre l'indice indiqué du tableau.

Le second paramètre est facultatif dans le cas où le tableau n'a qu'une seule dimension.

## EXEMPLE

```
100 DIM A(-3 TO 5, 1900 TO 2000),B(-10 TO -1)
110 PRINT UBOUND(A,1); UBOUND(A,2), UBOUND(B)
RUN
 5  2000 -1
```

## ERREURS

L'erreur No 4009 est détectée si la valeur de l'index est plus petite que 1 ou supérieure au nombre d'indices du tableau.

L'erreur No 25101 est détectée si l'index n'est pas numérique.

L'erreur No 25005 est détectée si le premier paramètre n'est pas un nom de tableau sans indice.

# UCASE$ FONCTION

## BUT

UCASE$ restitue une chaîne ne comportant que des majuscules.

## SYNTAXE

UCASE$ (<expression chaîne>)

## EXPLICATION

UCASE$ produit une chaîne de même longueur que la chaîne donnée comme paramètre. La seule différence entre la chaîne de sortie et la chaîne d'entrée réside dans le fait que les minuscules éventuelles de la chaîne d'entrée sont transformées en majuscules dans la chaîne de sortie.

## EXEMPLE

```
500 PRINT "DONNER VOTRE REPONSE"
510 INPUT a$
520 a$ = UCASE$(a$)
```

## ERREUR

L'erreur No 25102 est détectée si le paramètre n'est pas de type chaîne.



# USING$ FONCTION

## BUT

Donne une chaîne obtenue en formatant le résultat d'une expression.

## SYNTAXE

USING$ (<expression chaîne>,<expression>)

où <expression chaîne> est un format. Voir l'instruction PRINT USING où sont detaillés les différents formats numériques possibles.

## EXPLICATION

Cette fonction rend une chaîne formatée à partir d'une valeur numérique ou chaîne. Le format doit être conforme aux spécifications données dans l'instruction PRINT USING, mais le nombre de zones d'édition est obligatoirement 1.

## EXEMPLES

```
100 jour=10
110 a$=USING$("Nous sommes le ##" , jour)
120 PRINT a$; " du mois"
```

## ERREURS

L'erreur No 25102 est détectée si le premier paramètre n'est pas une chaîne.

L'erreur No 8201 est détectée si la chaîne de format est invalide.

L'erreur No 8202 est détectée si il n'y a pas de zone d'édition dans la chaîne de format.

## REMARQUE

La fonction USING$ peut être utile pour afficher des données formatées dans les zones d'affichage des masques MEMSCREEN.

# VAL FONCTION

## BUT

Cette fonction convertit la valeur numérique représentée par une chaîne en un nombre.

## SYNTAXE

VAL (<expression chaîne>)

## EXPLICATION

L'expression chaîne est évaluée, les espaces en début et en fin de chaîne sont ignorés. L'évaluation s'arrête dès qu'un caractère non interprétable est repéré.

Si la valeur chaîne, ainsi obtenue, représente une valeur numérique, VAL restitue cette valeur sous forme d'un nombre.

Si la valeur est trop petite en valeur absolue pour être représentée, VAL renvoie la valeur zéro.

## EXEMPLE

```
100 PRINT VAL("  3.14159  ")
RUN
 3.14159
```

## ERREUR

L'erreur No 1004 est détectée si le nombre est trop grand (en valeur absolue) pour être représentable en machine.

## REMARQUE

Voir EVALUATE qui est une forme plus évoluée de VAL acceptant des calculs simples.



# VERSION$ FONCTION

## BUT

La fonction VERSION$ rend une chaîne indiquant le nom de la machine et la version de MEMSOFT.

## SYNTAXE

VERSION$

## EXPLICATION

La fonction ne comporte pas de paramètre. La chaîne rendue contient :

<nom de la machine>,<nom du produit>:<version>

En utilisant la fonction POS pour rechercher les caractères "," ou ":", il sera facile d'extraire une partie de la chaîne : le nom du produit, par exemple.

## EXEMPLE

```
PRINT VERSION$
APPLE II GS,MEMSOFT GS:2.08
```

## ERREUR

Néant.

# VTAB FONCTION

## BUT

La fonction VTAB donne la ligne où est situé le curseur dans la fenêtre d'exécution.

## SYNTAXE

VTAB

## EXPLICATION

La fonction ne comporte pas de paramètre. La ligne 1 représente la première ligne de la fenêtre d'exécution.

## EXEMPLE

```
100 PRINT "Ceci est la ligne "; VTAB
```

## ERREUR

Néant.

## REMARQUES

Voir la fonction HTAB qui donne la colonne du curseur.

Voir l'instruction VTAB qui fixe la ligne du curseur.



# VTAB INSTRUCTION

## BUT

L'instruction VTAB fixe la ligne du curseur dans la fenêtre d'exécution.

## SYNTAXE

VTAB <index>

## EXPLICATION

La valeur de l'index représente le numéro de la ligne où l'on désire placer le curseur. La première ligne de la fenêtre d'exécution est numérotée 1.

La colonne du curseur n'est pas modifiée.

## EXEMPLE

```
100 VTAB 1     ! place le curseur en ligne 1
```

## ERREURS

L'erreur No 25101 est détectée si l'index n'est pas numérique.

L'erreur No 4000 est détectée si l'index est hors limites.

## REMARQUES

Voir la fonction VTAB qui donne la ligne du curseur.

Voir l'instruction HTAB qui permet de fixer la colonne du curseur.

# WHEN INSTRUCTION

## BUT

WHEN permet de traiter une erreur détectée lors de l'exécution du programme.

## SYNTAXE

WHEN EXCEPTION GOTO <Numéro de ligne> <étiquette>

ou

WHEN EXCEPTION BREAK

## EXPLICATION

La forme WHEN EXCEPTION GOTO... signifie que si une erreur est détectée, il doit y avoir branchement à la ligne indiquée.

La forme WHEN EXCEPTION BREAK signifie que si une erreur est détectée, le programme doit être interrompu et l'éditeur MEMBASIC activé.

L'enfoncement par l'utilisateur de la touche **POMME** CTRL C destinée à interrompre le programme rentre dans le cadre des erreurs concernées par cette instruction.

## EXEMPLE

```
100 WHEN EXCEPTION GOTO 'ERREUR'
...
900 'ERREUR':
```

## ERREUR

Néant.

## REMARQUE

Voir les fonctions EXTYPE, EXLINE et les instructions RETRY, CAUSE EXCEPTION pour gérer l'erreur.



# WRITE # INSTRUCTION

## BUT

WRITE # permet d'écrire des informations dans un fichier PRODOS ou un fichier de communication.

## SYNTAXE

WRITE #<numéro de fichier> , <liste d'expressions>

Le numéro de fichier est une expression numérique entière donnant le numéro du fichier PRODOS ou communication vers lequel on désire écrire.

La liste d'expressions est composée d'éléments séparés les uns des autres par une virgule.

Exemple : WRITE #1, A , B , X+Y ," FIN"

## EXPLICATION

L'instruction WRITE # envoie les différents éléments vers le fichier PRODOS ou de communication spécifié avec les conventions suivantes:

- Les différents éléments sont séparés par ",",

- Les chaînes sont écrites entre guillemets,

- une "fin de ligne" est envoyée en fin de liste.

Ce format est tel que la relecture par INPUT # reproduira de façon strictement identique les champs et leurs contenus.

Le fichier PRODOS ou communication utilisé doit avoir été ouvert avec l'instruction OPEN (ou OPEN "COM...").

Si le fichier est un fichier PRODOS, il ne doit pas avoir été ouvert avec le mode "I" qui ne permet pas l'écriture.

## EXEMPLE

```
100 OPEN "O",1,"ESSAI"
110 FOR i=0 TO 10
120     WRITE #1, i, nom$(i), prenom$(i)
130 NEXT
140 CLOSE
```

## ERREURS

L'erreur No 28503 est détectée si le numéro de fichier ne correspond à aucun fichier PRODOS ou communication ouvert.

L'erreur No 28514 est détectée si le fichier a été ouvert en mode lecture seule ("I").

## REMARQUES

Voir également PRINT # qui est plus adapté aux écritures de fichiers textes.

Voir le chapitre 1.17 pour plus d'informations sur les fichiers PRODOS et les communications série.

## CONSEIL

Utilisez de préférence le couple :

```
   PRINT # ... LINE INPUT #
ou WRITE # ... INPUT #
ou PRINT # ... INPUT$
```

pour retrouver les données à la lecture telles qu'elles étaient à l'écriture.



# ANNEXE A

## LISTE DES MOTS RESERVES DE MEMBASIC

Un langage comporte des "mots-clés" qui permettent de construire les instructions du programme. Les variables d'un programme ne doivent pas avoir le même nom que celui de certains mots-clés du langage qui sont appelés "mots réservés".
Certains interpréteurs considèrent que tous les mots-clés sont des mots réservés, ce qui constitue une restriction parfois gênante.

Avec MEMBASIC, les mots-clés qui sont réservés sont les mots NOT, AND, OR, ELSE, PRINT, TAB et REM ainsi que les noms des fonctions.
En effet, cela est nécessaire pour éviter toute ambiguïté syntaxique.

Par exemple, si ABS n'était pas un mot réservé, il y aurait ambiguïté lors de l'analyse syntaxique de

```
PRINT ABS(x)
```

car ce pourrait être aussi bien un élément de tableau non déclaré qu'une référence à la fonction ABS.

Pour que vos programmes restent compatibles avec les versions ultérieures de MEMSOFT, les noms des fonctions que l'on peut espérer être ajoutées sont inclus dans la liste des mots réservés.

La liste complète des mots réservés est la suivante :

| | | | |
|---|---|---|---|
| ABS | ACOS | AND | ANGLE |
| ASIN | ATN | CEIL | CHDIR$ |
| CHR$ | COMSTAT | COS | COSH |
| COT | CSC | CURDRIVE$ | CURLINE |
| DATE | DATE$ | DEG | DISKSIZE |
| ELSE | EPS | EVALUATE | EXKEY |
| EXLINE | EXP | EXTEXT$ | EXTYPE |
| EXWAY | EXZONE | FILESIZE | FP |
| FREE | HELP | HELP$ | HTAB |
| INPUT$ | INT | IP | LBOUND |
| LCASE$ | LEFT$ | LEN | LOG |
| LOG10 | LOG2 | LTRIM$ | MAX |
| MAXLEN | MAXLINE | MAXNUM | MEMBACKUP |

| | | | |
|---|---|---|---|
| MEMCOMPARE | MEMRESTORE | MID$ | MIN |
| MOD | NOT | OR | ORD |
| PATH$ | PI | POS | PRINT |
| PRINTER | RAD | REM | REMAINDER |
| REPEAT$ | RIGHT$ | RND | ROUND |
| RTRIM$ | SEC | SGN | SIN |
| SINH | SIZE | SPC | SQR |
| STATUS | STR$ | TAB | TAN |
| TANH | TIME | TIME$ | TRUNCATE |
| UBOUND | UCASE$ | USING$ | VAL |
| VERSION$ | VTAB | XOR | |



# ANNEXE B

## LISTE DES NUMEROS ET MESSAGES D'ERREUR

**Dépassement de capacité :**

1002 : Dépassement de capacité dans l'évaluation d'une expression numérique
1004 : Dépassement de capacité dans l'évaluation de la fonction VAL
1051 : Dépassement de la longueur maximale d'une chaîne lors d'une évaluation
1106 : Dépassement de la longueur maximale d'une chaîne lors d'une affectation
1902 : Dépassement de capacité dans l'évaluation d'une expression entière

**Erreurs d'indice dans un tableau :**

2001 : Valeur d'un indice de tableau hors limites
2002 : Nombre de dimensions supérieur au nombre permis
2090 : Tableau de taille supérieure à 64 K
2901 : Dépassement de la valeur maximale possible pour une dimension (32767)
2902 : Mauvais type pour un indice
2903 : Valeur de OPTION BASE supérieure à OPTION DIM en cours
2904 : Valeur de OPTION DIM inférieure à OPTION BASE en cours

**Erreurs mathématiques :**

3001 : Division par zéro
3002 : Nombre négatif élevé à une puissance non entière
3003 : Zéro élevé à une puissance négative
3004 : Logarithme d'un nombre négatif ou nul
3005 : Racine carrée d'un nombre négatif
3007 : Argument de ACOS ou ASIN non compris dans l'intervalle [-1,+1]
3008 : Tentative d'évaluer ANGLE (0,0)
3090 : Dépassement de capacité dans SIN, COS
3091 : Dépassement de capacité dans TAN, COT

**Erreurs de paramètres dans une fonction :**

4000 : Paramètre d'une fonction hors limites
4004 : Paramètre de la fonction SIZE hors limites
4006 : La valeur donnée à MARGIN est inférieure à celle de ZONEWIDTH
4007 : La valeur donnée à ZONEWIDTH est supérieure à celle de MARGIN ou inférieure à 1
4008 : Paramètre de la fonction LBOUND hors limites
4009 : Paramètre de la fonction UBOUND hors limites
4801 : La valeur donnée à ZONEWIDTH est inférieure à la valeur minimale possible
4901 : Paramètre d'une fonction hors limites car négatif
4902 : Paramètre de AUTO négatif ou impossible

**Erreurs de mémoire :**

5901 : Mémoire insuffisante et projection sur disque impossible
5995 : Erreur système : tentative d'accès au delà du premier libre
5997 : Erreur système : mauvaise table
5998 : Le programme actuellement en mémoire est détérioré
5999 : Mémoire détériorée

**Erreurs d'ENTREES/SORTIES :**

8001 : Fin des données dans READ sans IF MISSING
8101 : READ attend une valeur numérique et c'est une chaîne
8201 : Format invalide dans l'instruction PRINT USING
8202 : Pas de champ de variable dans le format de PRINT USING
8901 : RESTORE après la fin des données
8902 : La ligne référencée par le PRINT USING n'est pas une ligne IMAGE

**Erreurs dans des séquences automatiques :**

9001 : Impossible de modifier le programme par un EXECUTE
9002 : Impossible d'exécuter l'instruction REPLAY



**Erreurs dans les structures :**

10001 : Indice hors limites dans ON...GOTO ou ON...GOSUB et pas de ELSE
10002 : RETURN sans GOSUB ou ON ... GOSUB
10004 : Aucun CASE sélectionné et pas d'option CASE ELSE
10600 : Ligne ou étiquette non trouvée dans GOTO, GOSUB, ON ... GOTO ou ON ... GOSUB .. etc ..
10701 : LOOP rencontré sans DO
10702 : EXIT DO ou SKIP DO en dehors d'une boucle DO ... LOOP
10703 : LOOP non trouvé lors d'un EXIT DO ou SKIP DO ou boucle vide
10751 : SELECT CASE non suivi du premier CASE
10752 : CASE ou END SELECT rencontré sans SELECT CASE
10753 : END SELECT non trouvé en fin de boucle SELECT...CASE
10754 : CASE, CASE ELSE ou END SELECT non trouvé dans un SELECT CASE
10801 : END IF non trouvé après un IF ... THEN multi-lignes
10802 : END IF non trouvé après un ELSE ou ELSEIF
10901 : NEXT sans FOR
10902 : La variable du NEXT ne correspond pas à celle du FOR
10903 : EXIT FOR ou SKIP FOR en dehors d'une boucle FOR ... NEXT
10904 : NEXT non trouvé lors d'un EXIT FOR ou SKIP FOR ou boucle vide

**Erreurs : mauvais usage d'une fonction**

24001 : TAB ou SPC utilisés en dehors d'un PRINT
24901 : Usage d'une commande ou fonction non implémentéé
24902 : Erreur de syntaxe

**Erreurs de type :**

25001 : Une variable simple est indicée
25002 : Un tableau est utilisé sans indice
25003 : Mauvais nombre d'indices pour un tableau
25004 : Trop d'indices pour un tableau
25005 : Un nom de tableau sans indice est attendu
25006 : Nom de variable chaîne attendu dans MAXLEN
25101 : Un numérique est attendu et l'évaluation donne une chaîne
25102 : Une chaîne est attendue et l'évaluation donne un numérique

25103 : Tableaux de types incompatibles
25104 : Tableaux de dimensions différentes
25105 : Tableau non dimensionné dans SIZE, LBOUND, UBOUND ou MAT
25201 : La variable de boucle de FOR n'est pas une variable simple
25202 : La variable de boucle de FOR n'est pas numérique
25301 : Les deux éléments d'une comparaison ne sont pas de même type
25801 : RUN ou GOTO ligne de programme protégé
25802 : Tableau de chaînes dynamiques interdit comme paramètre de CHAIN
25803 : Instruction PROGRAM rencontrée après une affectation de variable
25804 : Variables de dimensions différentes dans CHAIN et PROGRAM
25805 : Variables de types différents dans CHAIN et PROGRAM
25806 : Variables de types incompatibles dans CHAIN et PROGRAM

**Erreurs de dimensionnement :**

26001 : Tentative de redimensionnement d'un tableau
26002 : Tentative de re-déclaration d'une variable chaîne.

**Erreurs à l'utilisation des masques MEMSCREEN et des fichiers MEMFILE :**

27001 : Commande MEMSOFT non reconnue dans la chaîne suivant le LET
27002 : Dictionnaire erroné
27003 : Identificateur temporaire non trouvé
27006 : Identificateur temporaire non trouvé dans un ordre LET "#CLEAR..."
27007 : Identificateur temporaire déjà utilisé
27009 : Spécification de disque incorrecte
27011 : Trop de masques MEMSCREEN ouverts en même temps
27013 : Trop de fichiers MEMFILE ouverts en même temps
27110 : Masque MEMSCREEN inexistant
27130 : Masque MEMSCREEN déjà existant
27140 : Masque MEMSCREEN d'une autre version
27150 : Masque MEMSCREEN de structure incorrecte
27161 : Variable alphanumérique pour une zone numérique d'un masque MEMSCREEN
27162 : Variable numérique pour une zone alphanumérique d'un masque MEMSCREEN



27165 : Erreur dans l'affectation d'une variable dans un ordre LET "TAKE..."
27166 : Erreur disque lors d'une opération d'ouverture ou d'écriture d'un masque
27167 : Trop de masques ouverts pour en ouvrir un nouveau
27168 : Erreur de syntaxe dans un masque
27171 : Option non valable dans un masque MEMSCREEN dans l'instruction LET
27500 : Le fichier de clés MEMFILE existe déjà
27501 : Le fichier des clés MEMFILE est impossible à créer
27502 : Le fichier des enregistrements MEMFILE existe déjà
27503 : Le fichier des enregistrements MEMFILE est impossible à créer
27505 : Disque plein
27509 : Erreur dans le dictionnaire d'un fichier MEMFILE
27510 : Le fichier des clés MEMFILE est impossible à ouvrir
27511 : Les fichiers des clés et des enregistrements sont d'âges différents
27512 : Le fichier des enregistrements MEMFILE est impossible à ouvrir
27514 : N'est pas un fichier MEMFILE
27517 : Le fichier n'a pas été fermé à la dernière utilisation
27519 : Dictionnaire trop long dans la création d'un fichier MEMFILE
27520 : Le fichier n'a pas été ouvert sur le lecteur spécifié
27530 : Fichier MEMFILE ouvert, ne peut être détruit
27532 : Fichier des clés inexistant
27533 : Le fichier des clés est impossible à détruire
27535 : Le fichier des enregistrements est impossible à détruire
27540 : Fichier ouvert : ne peut être renommé
27542 : Le fichier des enregistrements MEMFILE est inexistant
27544 : Fichier impossible à renommer
27552 : Les homonymes sont refusés
27553 : Erreur dans la création d'un enregistrement MEMFILE
27555 : Erreur dans l'insertion d'une clé MEMFILE
27557 : Limite de taille de fichier atteinte pour MEMSOFT DEMONSTRATION
27561 : Continuation d'écriture impossible
27571 : Le fichier MEMFILE ne possède pas de fichiers d'enregistrements

27572 : Fiche MEMFILE inexistante
27573 : Incohérence de données dans un fichier MEMFILE
27575 : Option non valable pour un fichier MEMFILE dans la commande LET
27577 : Trop de clés pour un fichier MEMFILE
27579 : Mémoire insuffisante pour une recherche de clé dans MEMFILE
27581 : Numéro de clé invalide dans un fichier MEMFILE
27583 : Erreur de données dans un fichier MEMFILE
27587 : Type de variable incorrect dans le dictionnaire d'un fichier MEMFILE
27589 : Type de variable incorrect pour la clé relative d'un fichier MEMFILE
27591 : Fichier créé sans clé relative
27593 : Erreur disque
27594 : Limite de fichier atteinte lors d'une lecture séquentielle
27595 : Clé trop longue
27596 : Fiche déjà détruite
27597 : Mémoire insuffisante pour MEMFILE

**Erreurs PRODOS :**

28010 : Fichier non trouvé
28020 : RENAME impossible
28030 : Fichier déjà existant
28040 : Fichier impossible à créer dans COPY
28051 : Abandon de l'utilisateur dans une opération disque
28100 : Erreur dans la comparaison de deux fichiers
28110 : Répertoire non trouvé
28120 : Répertoire non destructible
28130 : Répertoire déjà existant
28201 : Accès refusé par PRODOS
28211 : Tentative de lecture après une fin de fichier
28499 : Autre erreur PRODOS

**Erreurs fichiers PRODOS**

28501 : Trop de fichiers PRODOS déjà ouverts
28502 : Numéro logique déjà existant
28503 : Numéro logique inexistant
28504 : Fichier à détruire inexistant
28505 : Numéro logique non correct dans OPEN
28506 : Mode d'ouverture incorrect dans OPEN
28507 : Fichier impossible à détruire
28508 : Erreur en lecture de fichier
28509 : Fin de fichier en lecture
28510 : Ligne trop longue dans LINE INPUT #
28511 : Opération interdite sur fichier



28512 : Une variable est attendue pour LINE INPUT #
28513 : La variable de LINE INPUT # doit être de type chaîne
28514 : Opération interdite sur fichier en fonction du mode d'ouverture
28515 : Erreur de clôture de fichier
28516 : Fichier inexistant
28517 : Erreur dans l'affectation de INPUT # ou LINE INPUT #
28518 : Donnée trop longue dans LINE INPUT # ou INPUT #
28519 : Fin de fichier rencontrée lors de INPUT #, INPUT$ ou LINE INPUT #
28520 : Erreur dans POINTER #
28521 : Longueur trop grande dans OPEN

**Erreurs de communication :**

28570 : Périphérique inexistant
28900 : Erreur disque

**Erreurs en mode immédiat :**

29001 : READ utilisé en mode immédiat
29002 : Reprise par CONTINUE impossible
29003 : Reprise par RETRY impossible
29004 : Lecture de données DATA en mode immédiat
29005 : RETRY impossible en mode immédiat

**Interruption de programme :**

30001 : Programme interrompu volontairement par l'utilisateur
30005 : POMME CTRL C pendant l'attente d'écriture d'un article

# INDEX THEMATIQUE

## LISTE DES MOTS CLES CLASSES PAR UTILISATION

Voici les différentes commandes, instructions et fonctions MEMBASIC classées par type d'utilisation. Chaque mot-clé est suivi d'un commentaire sur son utilisation. Pour en savoir plus, se reporter au chapitre 2 où tous les mots-clés sont décrits, classés par ordre alphanumérique.

## CALCULS NUMERIQUES ET TRIGONOMETRIQUES

**Instructions :**

| | |
|---|---|
| OPTION ANGLE | DEGREES : angles exprimés en degrés<br>RADIANS : angles exprimés en radians |
| RANDOMIZE | Point de départ d'une séquence de nombres pseudo-aléatoires |

**Fonctions :**

| | |
|---|---|
| ABS | Donne la valeur absolue |
| ACOS | Arc cosinus |
| ASIN | Arc sinus |
| ATN | Arc tangente |
| CEIL | Troncature supérieure |
| COS | Cosinus |
| COT | Cotangente |
| CSC | Cosécante |
| DEG | Conversion d'un angle en degrés |
| EVALUATE | Evaluation chaîne avec calculs |
| EXP | Exponentielle |
| FP | Partie fractionnaire |
| INT | Partie entière |
| IP | Partie décimale |
| LOG | Logarithme népérien |
| LOG10 | Logarithme décimal |
| LOG2 | Logarithme en base 2 |
| MAX | Maximum |
| MAXNUM | Le nombre le plus grand |
| MIN | Minimum |
| MOD | Modulo |
| PI | Nombre PI |
| RAD | Conversion d'un angle en radians |
| REMAINDER | Reste de la division |
| RND | Nombre pseudo-aléatoire |
| ROUND | Arrondi au plus proche |

| | |
|---|---|
| SEC | Sécante |
| SGN | Donne le signe |
| SIN | Sinus |
| SQR | Racine carrée |
| TAN | Tangente |
| TRUNCATE | Troncature |
| VAL | Evaluation chaîne sans calculs |

## LES CHAINES DE CARACTERES

**Instructions :**

| | |
|---|---|
| DECLARE | Déclaration de chaîne |

**Fonctions :**

| | |
|---|---|
| CHR$ | Chaîne à partir d'un code caractère |
| LCASE$ | Tout en minuscules... |
| LEFT$ | Extraction début de chaîne |
| LEN | Longueur d'une chaîne |
| LTRIM$ | Enleve les espaces à gauche |
| MAXLEN | Longueur maximale d'une chaîne |
| MID$ | Extraction de sous-chaîne |
| ORD | Code du premier caractère de chaîne |
| POS | Recherche de sous-chaîne |
| REPEAT$ | Répétition de chaîne |
| RIGHT$ | Extraction fin de chaîne |
| RTRIM$ | Enlève les espaces à droite |
| STR$ | Conversion numérique vers chaîne |
| UCASE$ | Tout en majuscules... |
| USING$ | Formate des données |

## L'HEURE ET LA DATE

| | |
|---|---|
| DATE | Donne la date en numérique |
| DATE$ | Donne la date formatée |
| TIME | Donne l'heure en numérique |
| TIME$ | Donne l'heure formatée |



## REPERTOIRES ET FICHIERS PRODOS

**Instructions :**

| | |
|---|---|
| CHDIR | Changement de répertoire |
| CLOSE | Fermeture fichier PRODOS/communication |
| COPY | Copie de fichier |
| CURDRIVE | Changement de disque implicite |
| DIR | Liste d'un répertoire |
| FILESIZE | Taille des fichiers |
| INPUT # | Lecture fichier PRODOS/communication |
| KILL | Destruction d'un fichier |
| LINE INPUT # | Lecture fichier PRODOS/communication |
| MKDIR | Création de répertoire |
| OPEN | Ouverture de fichier PRODOS |
| OPEN "COM" | Ouverture de fichier communication |
| PATH | Définition de chemins implicites |
| POINTER # | Choix enregistrement de fichier PRODOS |
| PRINT # | Ecriture fichier PRODOS/ communication |
| RENAME | Changement de nom de fichier |
| RMDIR | Suppression de répertoire |
| WRITE # | Ecriture fichier PRODOS/ communication |

**Fonctions :**

| | |
|---|---|
| CHDIR$ | Donne le répertoire en cours |
| COMSTAT | Donne l'état des lignes série |
| CURDRIVE$ | Donne le disque en cours |
| DISKSIZE | Donne la taille du disque |
| FREE | Donne la place libre sur le disque |
| INPUT$ | Lit un nombre d'octets dans le fichier PRODOS/communication |
| PATH$ | Donne les chemins implicites |

## DISQUE DUR

| | |
|---|---|
| MEMBACKUP | Sauvegarde des fichiers |
| MEMCOMPARE | Vérification des sauvegardes |
| MEMRESTORE | Restauration des fichiers |

## LA GESTION DES ERREURS

**Instructions :**

| | |
|---|---|
| WHEN | Branchement vers la gestion d'erreur |
| RETRY | Retentative |
| CAUSE | Provoque une erreur |

**Fonctions :**

| | |
|---|---|
| EXLINE | Ligne de l'erreur |
| EXTEXT$ | Message d'une erreur |
| EXTYPE | Numéro d'erreur |

## LES STRUCTURES

| | |
|---|---|
| CASE | Cas de SELECT/CASE |
| CASE ELSE | Cas de SELECT/CASE |
| DO | Début boucle DO/LOOP |
| ELSE | Voir IF,READ,ON... |
| END | Fin programme |
| END IF | Fin bloc IF |
| END SELECT | Fin SELECT/CASE |
| EXIT | Sortie de boucle |
| FOR | Début boucle FOR/NEXT |
| GOSUB | Appel sous-programme |
| GOTO | Branchement |
| IF | Début bloc IF |
| LOOP | Fin boucle DO/LOOP |
| NEXT | Fin boucle FOR/NEXT |
| ON GOSUB | GOSUB liste |
| ON GOTO | GOTO liste |
| RETURN | Retour sous-programme |
| SELECT | Début SELECT/CASE |
| SKIP | Provoque la boucle |
| THEN | Voir IF |
| UNTIL | Condition pour DO ou LOOP |
| WHILE | Condition pour DO ou LOOP |

## LES DONNEES ET ENTREES/SORTIES

| | |
|---|---|
| ASK | Lit les paramètres d'édition |
| CLEAR KEYBOARD | Vide le tampon clavier |
| CLS | Efface l'écran |
| CURLINE | Compteur de lignes d'édition |
| DATA | Définition de données |
| GET KEYBOARD | Attente clavier |
| HTAB | Curseur horizontal |
| INPUT | Entrée de données |



| | |
|---|---|
| MAXLINE | Nombre de lignes par page |
| MISSING | Option de READ |
| PRINT | Edition non formatée |
| PRINTER | Connexion imprimante |
| PRINT USING | Edition formatée |
| READ | Lecture données |
| REPLAY | Rejoue une séquence enregistrée |
| RESTORE | Change les données lus par READ |
| SET | Fixe les paramètres d'édition |
| SPC | Impression d'espaces |
| TAB | Tabulation pour PRINT |
| VTAB | Curseur vertical |

## LES TABLEAUX

**Instructions :**

| | |
|---|---|
| DIM | Dimensionnement de tableau |
| OPTION BASE | Minimum implicite des indices |
| OPTION DIM | Maximum implicite des indices |
| MAT | Initialisations et copies |

**Fonctions :**

| | |
|---|---|
| LBOUND | Donne le minimum des indices |
| UBOUND | Donne le maximum des indices |
| SIZE | Donne la taille d'un tableau |

## LES COMMANDES EDITEUR

| | |
|---|---|
| AUTO ON/OFF | Numérotation automatique |
| BREAK | Arrêt programme |
| CONT | Reprise d'une exécution |
| DEBUG | Passage en mode mise au point |
| DELETE | Destruction de lignes |
| ERASE ALL | Efface les variables |
| LIST | Liste |
| LOAD | Chargement d'un programme |
| LOCNUMBER | Renumérotation par adresses |
| NEW | Effacement du programme en cours |
| RENUMBER | Renumérotation du programme |
| RUN | Exécution du programme |
| SAVE | Sauvegarde d'un programme |
| STEP | Choix d'un pas pour AUTO et RENUMBER |
| TRACE ON/OFF | Demande ou arrêt du mode trace |

## ENCHAINEMENTS

| | |
|---|---|
| CHAIN | Exécution d'un programme avec passage de paramètres |
| PROGRAM | Récupération de paramètres |

## DIVERS

**Instructions :**

| | |
|---|---|
| EXECUTE | Exécute une chaîne |
| FKEY | Message des touches de fonctions |
| HELP | Définit le fichier d'aide cherché |
| REM | Début de remarque |
| SYSTEM | Quitte MEMSOFT |

**Fonctions :**

| | |
|---|---|
| EXKEY | Code de touche sortie d'un masque |
| EXWAY | Sens de sortie d'un masque |
| EXZONE | Zone de sortie d'un masque |
| STATUS | Code erreur MEMSCREEN ou MEMFILE |
| VERSION$ | Nom et numéro de la version de MEMSOFT |



# SECTION III
# MEMSCREEN

## P L A N

# CHAPITRE 1
# GENERALITES

## 1.1 INTRODUCTION

De même que MEMBASIC utilise trois fenêtres pour fonctionner, l'une pour l'édition du programme, l'autre pour son exécution, une troisième pour en contrôler le fonctionnement, les programmes MEMBASIC peuvent aussi utiliser d'autres fenêtres grâce à MEMSCREEN.

Les masques MEMSCREEN résolvent les problèmes de saisie et de consultation de données. Les différents masques utilisés par un programme MEMBASIC se superposent sur l'écran et l'utilisateur peut à sa guise les déplacer, changer leur taille ... Pour toutes ces manipulations de fenêtres, consulter la section I "FENETRE".

MEMSCREEN offre aussi un outil d'aide à la réalisation de modules d'édition.

Les exemples qui sont donnés dans cette section comportent parfois des portions de programmes MEMBASIC. Dans ce cas le fonctionnement des instructions MEMBASIC ne sera pas détaillé. Se reporter à la section II "MEMBASIC" pour une meilleure compréhension des exemples.

## 1.2 SYNTAXE

Toutes les instructions MEMSCREEN ont la forme suivante:

LET <expression chaîne>

Le mot-clé étant toujours LET, c'est le contenu de l'expression chaîne qui déterminera l'instruction à exécuter. L'expression chaîne peut être aussi bien une constante chaîne qu'une expression évaluée, ce qui donne une assez grande souplesse dans la programmation des masques.

Cependant, dans la plupart de nos exemples nous prendrons la forme LET "..." qui est la forme la plus simple, mais ceci ne veut pas dire que les formes plus compliquées (LET suivi d'une expression chaîne évaluée) ne sont pas possibles.

La chaîne de caractères contient des mots-clés comme "PRINT" ou "CHARGE" ... Ces mots-clés sont dans certains cas précédés de "#" comme dans "#OPEN" ou "#UPDATE".

Pour ces mots-clés, la première lettre seule est significative (ou première lettre après "#") et elle peut être écrite en majuscule ou en minuscule indifféremment. Par exemple :

"#O" , "#o", "#open" et "#OPEN"
sont équivalents.

de même:

"Print" , "P" , "p" et "PRINT"

Si l'instruction comporte plusieurs mots, ils pourront être séparés par "-" , ":" , "," ou ";".

Si l'expression comprend le nom d'un fichier, il devra respecter la syntaxe imposée par le système d'exploitation.

L'erreur No 27001 est détectée si l'expression chaîne qui suit LET n'est pas reconnue valide.

Le chapitre 2 détaille, pour chaque forme de l'instruction LET, sa syntaxe et son fonctionnement.

## 1.3 IDENTIFICATEUR TEMPORAIRE

Les masques MEMSCREEN sont des entités indépendantes du programme MEMBASIC qui les utilise. Ils sont créés et sauvés sur disque et pourront être utilisés dans différents programmes.

Lorsque qu'un programme MEMBASIC désire utiliser un masque MEMSCREEN, il doit tout d'abord l'OUVRIR. Cela signifie qu'une copie du masque est chargée dans la mémoire de l'ordinateur. Le masque MEMSCREEN est alors disponible. Il peut être utilisé pour des affichages ou des saisies.

Pour éviter de rappeler le nom du masque à chaque utilisation, on associe à chaque masque lors de son ouverture un IDENTIFICATEUR TEMPORAIRE.



L'identificateur temporaire est un caractère du jeu ISO à l'exception de $ et des caractères de code inférieur ou égal à 32. Les lettres minuscules ou majuscules représentent les mêmes identificateurs temporaires.

| | |
|---|---|
| 1 | est un identificateur temporaire valide, |
| A ou a | aussi. |

Par exemple : on ouvre le masque "TEST" et on lui attribue l'identificateur temporaire "X". Pour désigner ce masque par la suite, seul "X" (ou "x") sera nécessaire. LET "PRINT,X" affiche le masque, LET "INPUT,X" provoque la saisie des valeurs des différentes zones...

## 1.4 CREATION OU MODIFICATION D'UN MASQUE

La seule différence entre la création et la modification est l'état initial de la fenêtre en cours de définition sur l'écran.

Dans le cas de la création, cette fenêtre est vide, dans celui de la modification, elle contient les informations précédentes du masque que l'on veut modifier. La fenêtre elle-même reprend son état antérieur en cas de modification.

Un masque est associé à une fenêtre d'écran.

Pour chaque masque, il faut définir :

- les spécifications de la fenêtre :
  - taille
  - cadre
  - position
  - couleur
    .....
- le texte du masque,
- les zones de saisie et d'affichage.

Le masque, quelle que soit la taille de la fenêtre, pourra comporter jusqu'à 250 lignes de 250 caractères.

La fenêtre qui représente la partie visible du masque sera, elle, limitée à la taille physique de l'écran.

L' instruction de création de masque est :

LET "#NEW,M,<nom du masque>"

ou avec la forme simplifiée ...

LET "#N,M,<nom du masque>"

Celle de modification est :

LET "#UPDATE,M,<nom du masque>"

ou avec la forme simplifiée ...

LET "#U,M,<nom du masque>"

Le chapitre 2 détaille la syntaxe et l'action de ces instructions ainsi que les erreurs qui peuvent survenir.

### 1.4.1 La Fenêtre d'un masque.

La fenêtre définie lors de la création ou la modification d'un masque sera reproduite lors de l'utilisation du masque.

Même si l'utilisateur peut alors modifier ses caractéristiques, elles ne seront pas sauvegardées et resteront locales au programme. Dès que le masque sera fermé, puis réouvert, il reprendra sa fenêtre implicite.

Pour définir taille, position, cadre, et couleur de cadre de la fenêtre du masque, consulter la section I "FENETRE". Il est possible de changer :

- la taille
- la position
- le cadre
- la couleur du cadre

### 1.4.2 Le texte d'un masque et les touches du clavier

Quelles que soient les caractéristiques de la fenêtre, la taille maximale du masque reste 250 X 250 caractères.

Le défilement du texte dans la fenêtre (vertical ou horizontal) sera automatique de façon à rendre toujours possible la saisie.



Lors de la création du masque, un second écran contiendra soit des messages guidant l'opérateur, soit les informations concernant les zones du masque.

Pour rentrer le texte du masque, il suffit de déplacer le curseur avec les flèches de direction sur l'espace 250 X 250 caractères jusqu'à l'endroit désiré et de taper le texte. Il est bien sur possible d'utiliser la souris pour se déplacer rapidement à une position visible du masque.

La couleur caractère par caractère est redéfinissable en utilisant l'option **"Couleur MASQUE"** dans le mode "FENETRE". Cette option, qui sert à l'utilisateur pour définir la couleur de ses saisies, est utilisée ici pour définir la couleur des prochains caractères à taper.

**Les touches du clavier**

Voici l'effet des différentes touches du clavier lors de la phase de création du texte d'un masque.

**"OPTION CLIC-Souris"** : Passe en mode modification de fenêtre.

**flèches** : déplacent le curseur de un caractère dans le sens de la flèche.

**"POMME ↑** : Remonte le curseur de la hauteur d'une demi-fenêtre.

**"POMME ↓** : Descend le curseur de la hauteur d'une demi-fenêtre.

**"CTRL POMME ↑"** : Remonte le curseur en haut du texte.

**"CTRL POMME ↓"** : Descend le curseur en bas du texte.

**"TAB→|** Déplace le curseur de huit caractères vers la droite.

**"SHIFT TAB→|"**: Déplace le curseur de huit caractères vers la gauche.

**"INSERT"** : Passe en mode insertion dans lequel tous les caractères tapés sont insérés. L'appui sur l'une des touches de direction ou sur "RETOUR CHARIOT" fait quitter le mode insertion.

"POMME DELETE" : le caractère situé à la position du curseur est détruit.

"BACK SPACE" : Destruction du caractère à gauche du curseur.

"POMME <-" : Ramène le curseur en début de ligne.

"CTRL POMME <-" : Amène le curseur en haut d'écran à la première frappe de cette touche, puis en haut de texte à la seconde frappe.

"CTRL -> " : va au prochain mot (à droite).

"CTRL <- " : va au mot précédent (à gauche).

"CTRL DELETE" ou ⌦ : détruit les caractères depuis le curseur jusqu'en fin de ligne.

"ESCape" abandonne la création ou la modification du masque. Par sécurité, MEMSCREEN demande confirmation.

Les touches de fonctions POMME 0 à POMME 9 ont ici le fonctionnement suivant:

F0 Appel des fichiers d'aides MEMSCREEN.
F1 Sort en validant la création ou la modification.
F2 Efface le masque. Valide s'il n'y a pas de zone.
F3 Donne un titre au masque.
F4 ...inactif à cette phase...
F5 Détruit une ligne.
F6 ...inactif à cette phase...
F7 Duplication de la ligne du curseur en dessous.
F8 Insertion d'une ligne vide.
F9 Recherche de zone.

N'oubliez pas que pour obtenir la touche de fonction POMME 3 par exemple, il faut enfoncer les touches **POMME** et la touche 1 du clavier numérique.
Toute opération provoquant la perte d'une ou plusieurs zones de saisie ou d'affichage sera refusée pour des raisons de sécurité. Pour utiliser quand même la fonction concernée, détruire d'abord les zones gênantes. Pour cela, voir le rôle et le mode d'emploi de la touche de fonction **POMME 6** au chapitre suivant. Par exemple, **POMME 2** n'effacera pas l'écran si il y a au moins une zone qui a été définie.



Le nombre de caractères utilisables est de 254 mais ils ne sont pas tous disponibles directement au clavier.
Pour obtenir les caractères non dessinés sur le clavier (caractères graphiques, par exemple) il suffit d'enfoncer la touche **OPTION** et de taper le code du caractère sur le clavier numérique. La liste des caractères disponibles et leurs codes est donnée dans un fichier d'aide associé obtenu par l'enfoncement de la touche **AIDE**.

**Principe de destruction et copie des lignes**

La touche **POMME 5** détruit une ligne sur l'écran. Cette destruction ne sera possible que si la ligne ne contient pas de zone. Dans le cas contraire, la destruction est refusée et un message le signale dans la fenêtre annexe.

La ligne détruite n'est pas définitivement perdue, elle est temporairement conservée par MEMSCREEN. Il est possible de la récupérer en enfonçant **OPTION POMME 5**. Cela permet de 'rattraper' une erreur de manipulation, mais aussi de déplacer une ligne dans le masque.

Si l'on désire seulement copier cette ligne, il faut utiliser **SHIFT POMME 5** qui enregistre la ligne sans la détruire. La ligne sera recopiée à un autre endroit autant de fois que désiré par **OPTION POMME 5**. Attention ! cette méthode ne permet pas de copier des lignes contenant des zones (marquées par [), l'enregistrement de la ligne par **SHIFT POMME 5** étant dans ce cas refusé. Voir au chapitre suivant comment copier des zones.

### 1.4.3 Les zones d'un masque

La définition d'une zone se fait en deux temps :

- Position et longueur

- Caractéristiques.

**Position et longueur**

Le début de la zone sur l'écran sera noté :

[

Les autres caractères de la zone seront notés :

]

Par exemple :

[]]]]] représente une zone de 6 caractères.

Si la zone est de longueur 1, seul [ est à mettre.

Il est possible par la suite de revenir sur cette zone et de changer le nombre de ] pour allonger ou raccourcir la zone. En revanche, il n'est pas permis de détruire [ ce qui serait équivalent à la destruction de la zone. Pour détruire une zone, la procédure à suivre est la suivante:

- Placer le curseur sur l'un des crochets []]]]]

- Enfoncer la touche **POMME 6** qui détruit la zone.

Caractéristiques

Les caractéristiques de la zone doivent être entrées dans la fenêtre de contrôle. Pour cela :

- Placer le curseur sur l'un des crochets []]]]]

- Enfoncer la touche **POMME 4**.

La fenêtre de contrôle apparaît alors et le curseur est placé dans la première de ses zones. On peut également cliquer sur la fenêtre de contrôle, si celle-ci est visible, au lieu d'enfoncer la touche **POMME 4**.

Voici les questions posées :

```
Nom..
Cond.
 En sortie N (O/N)        Longueur: 002
 Numérique N (O/N)        Re calcul N (O/N)

 A effacer N (O/N)        A droite. N (O/N)
 Date .... N (O/N)        'O'ou'N'. N (O/N)
 Espaces.. N (O/N)        Majuscul. N (O/N)
```

Il faut répondre à ces questions. En fonction des réponses apportées à "En sortie" et "Numérique", les autres questions pourront changer.



Pour une zone alphanumérique (réponse N à Numérique), les questions sont celles posées ci-dessus.

- A effacer (O/N) : permet de choisir si la dernière valeur de la variable doit être affichée ou non à l'écran lors d'une saisie.
  Sinon, la zone de saisie sera remise à blanc.

- Date (O/N) : permet de spécifier si la variable à saisir est une date. Si la réponse est O (oui), un contrôle sera effectué à la saisie et les dates invalides seront refusées.
  La date devra être saisie sous l'un des formats suivants :

  ```
  12-08-1986          010186
  3/02/58     ou      01;01;86
  ```

  Quel que soit le format choisi à la saisie (séparateur -, /,;), la date sera toujours affichée sous la forme : 12-10-80.

- Espaces (O/N) : permet de conserver ou non les espaces saisis à gauche d'une donnée alphanumérique. Les espaces à droite sont toujours éliminés.

- A droite (O/N) : permet le cadrage de la zone à droite.

- 'O' ou 'N' (O/N) : permet de n'accepter que les caractères O (Oui) ou N (Non) à la saisie. La saisie est refusée tant que la réponse n'est pas O ou N (ou o et n).

- Majuscul. (O/N) : permet de transformer la chaîne en majuscules. Cette possibilité est à conseiller pour les variables de clé d'un fichier MEMFILE.
  Attention : les caractères accentués sont considérés comme des caractères semi-graphiques et ne sont donc pas transformés en majuscules.
  Exemple :

  ```
  Dupont   devient   DUPONT
  élève    devient   éLèVE
  ```

Pour une zone numérique (réponse O à Numérique), les questions suivantes apparaissent à l'écran :

```
Nom..
Cond.
 En sortie N (O/N)        Longueur: 002
 Numérique O (O/N)        Re calcul N (O/N)

 A effacer N (O/N)        A droite. N (O/N)
 Affiche 0 N (O/N)        Exposant. N (O/N)
 Positif.. N (O/N)        Decimale. O (0-9)
```

- A effacer (O/N) : permet de choisir si la dernière valeur de la variable doit être affichée ou non à l'écran lors d'une saisie.
  Sinon, la zone de saisie sera remise à blanc.

- Affiche 0 (O/N) : permet d'afficher ou non les valeurs nulles à l'écran.
  Si la réponse est N (Non), les valeurs nulles ne seront pas représentées à l'écran.
  Si la réponse est O (Oui), les valeurs nulles seront représentées par des zéros avec le nombre de décimales indiqué.

- Positif (O/N) : permet de n'accepter que les nombres positifs à la saisie. Si la réponse est O (Oui), les valeurs négatives sont refusées.

- A droite (O/N) : permet le cadrage de la zone à droite.

- Exposant (O/N) : permet d'accepter les nombres sous forme exponentielle (Ex : 12E+8). Si le nombre peut être converti, il le sera. Sinon, il restera sur l'écran sous forme exponentielle.

- Décimale (0-9) : permet de fixer le nombre de décimales désiré de 0 à 8. Une saisie avec un nombre supérieur de décimales sera refusée. Le nombre s'affichera toujours avec le format choisi. Si le nombre comporte plus de décimales (par exemple s'il provient d'une affectation dans MEMBASIC), il sera arrondi au plus proche.



**Exemples : nombre de décimales = 2**

```
5.1   sera affiché   5.10
5.12  ............   5.12
5.123 ............   5.12
5.127 ............   5.13
```

Le choix 9 représente un format libre, c'est à dire que tous les formats sont possibles et que les nombres seront affichés avec le format saisi.

**Copie de zones**

Lorsqu'une zone vient d'être créée à l'écran par la frappe du crochet gauche [ , ses caractéristiques sont vides, c'est-à-dire que les deux lignes NOM et COND ne sont pas remplies et que toutes les questions ont comme réponse N (NON). Pour remplir ces caractéristiques, il peut être parfois plus rapide de ne pas partir systématiquement d'un tableau vide. MEMSCREEN permet pour cela de copier les caractéristiques d'une zone sur une autre.

La méthode à suivre est la suivante :

1 - Placer le curseur sur la zone d'origine.
2 - Enfoncer la touche **SHIFT POMME 4** pour enregistrer sa position.
3 - Déplacer le curseur vers la zone à renseigner
4 - Au lieu de taper **POMME 4** qui donne le tableau actuel (donc vide la première fois), enfoncer **OPTION POMME 4** qui propose pour la nouvelle zone les caractéristiques de celle pointée précédemment.
5 - Apporter les modifications nécessaires puis valider par **POMME 1** ou annuler par **ESCape**.

Les étapes 3, 4 et 5 pourront être répétées autant de fois que la zone d'origine peut servir de modèle.
Cette méthode est particulièrement agréable pour introduire des lignes d'éléments de tableau.

### 1.4.4 Zones en entrée

La ligne Nom indique la variable dans laquelle doit être affecté le résultat de la saisie.

Ce peut être :

- une variable simple :
  - prix
  - age
  - nom$

- un élément de tableau :
  - tva (1)
  - tva (code_tva)
  - tva (VAL(a$))
  - valeur (i,j)

La ligne **Cond** est une expression booléenne de forme MEMBASIC.

Après la saisie, la formule est évaluée.
Si le résultat est faux, (c'est-à-dire 0), la saisie est refusée.

Exemples:

```
Nom.. PRIX
Cond. PRIX > 1000
```

autres exemples de conditions :

```
(PRIX < 1000) OR (PRIX > 2000)
```

réalise le OU : accepte la zone si

```
PRIX < 1000
ou PRIX > 2000
```

```
(PRIX < 1000) AND (PRIX > 500)
```

réalise le ET : accepte la zone si

```
500 < PRIX < 1000
```

Si la condition est fausse, un signal sonore signalera l'erreur lors de la saisie dans le masque et la saisie sera redemandée (voir LET "INPUT..." au chapitre 2 pour réaliser la saisie). La même chose se produit si l'un des contrôles du programme n'est pas réalisé.



Par exemple :

- nombre de décimales incorrect,
- positivité,
- chaîne représentant une date invalide,
- réponse O ou N seule acceptée.

Notes :

1 - La zone condition est facultative

2 - Un contrôle syntaxique est effectué à la saisie.

En cas d'erreur détectée à la saisie, la correction ou l'abandon de la formule fausse sont les deux seuls choix possibles. L'abandon rendra à TOUTES les caractéristiques de la zone leurs valeurs précédentes.
Malgré ces contrôles, certaines erreurs peuvent se produire lorsque le masque est utilisé en saisie, (par exemple : dépassement de capacité, division par zéro, chaîne trop longue...)

### 1.4.5 Recalcul

Après chaque saisie, il est possible de provoquer un recalcul et un réaffichage de toutes les zones en sortie du masque.

Cela peut être particulièrement utile si l'une des zones d'affichage dépend de la valeur saisie (voir les formules de calcul dans les zones de masque dans le paragraphe suivant).

Par exemple :

si la zone saisie est prix_ht représentant un prix hors taxes et que, ailleurs dans le masque, une zone en sortie affiche :

```
prix_ht * 1.186
```

Si l'on demande l'option Recalcul sur la zone prix_ht, après saisie de cette zone, le réaffichage fera apparaître immédiatement une valeur juste du total TTC (PRIX_HT * 1.186)

### 1.4.6 Zones en sortie

La ligne Nom, peut représenter soit le nom d'une variable à afficher, indicée ou non, soit une expression numérique ou chaîne au format MEMBASIC.

Exemples :

- PRIX
- ADRESSE$
- code_postal & " et " & ville$
- LARGEUR * HAUTEUR

Condition

La condition permet de savoir si la zone doit être ou non affichée.
Si elle est "vraie" (résultat non nul), la zone sera affichée.
Sinon, elle sera mise à blanc.

La zone condition est facultative.
Dans le cas où elle est omise, l'affichage sera systématique.

EXEMPLES :

Affichage d'un message constant suivant une condition :

```
Name : "BONNE AFFAIRE"
Cond : REMISE > 10
```

Affichage d'une valeur calculée suivant une condition :

```
Name : - MONTANT
Cond : MONTANT < 0
```

Changement d'un code en valeur :

```
Name : (TVA=1) * 18.6 + (TVA=2) * 33.33
Cond : TVA > 0

ce qui affichera : 18.6  si TVA = 1
                   33.33 si TVA = 2
                   Rien  si TVA = 0
```



Changement d'un code en message :

```
Name : MID$("NONOUI",1+3*REP,3)
Cond : REP = 0 OR REP = 1

ce qui affichera : "NON" si REP = 0
                   "OUI" si REP = 1
                   Rien si REP vaut autre
                   chose que 0 ou 1
```

### 1.4.7 Les barres semi-graphiques

Cette option permet d'afficher les valeurs numériques sous la forme d'une barre horizontale de caractères semi-graphiques.

Plusieurs caractères semi-graphiques sont possibles.

Il est possible de réserver la largeur de la zone pour des valeurs toujours positives ou bien de la partager entre des valeurs positives et négatives (0 étant au milieu de la zone).

L'effet sera le suivant :

1. Représentation de valeurs positives uniquement :

   - si valeur<=0 : barre vide : zone blanche
   - si valeur>=100 : barre pleine : sur la largeur indiquée par des crochets [][][] La barre est remplie du caractère choisi.
   - si 100>valeur>0 : barre variable : largeur proportionnelle à la valeur.

2. Représentation de valeurs numériques quelconques :

   - si valeur>=100 : demi-barre de droite pleine
   - si valeur<=-100 : demi-barre de gauche pleine
   - si valeur=0 : barre vide : zone blanche
   - si 100>valeur>0 : demi-barre de droite proportionnellement remplie

- si -100<valeur<0  demi-barre de gauche
                    proportionnellement remplie.

## 1.5 MASQUES ET IMPRIMANTE

MEMSCREEN offre un large choix de commandes et instructions pour obtenir des copies papier de tout ou partie d'un masque.

### 1.5.1 Copie manuelle d'une fenêtre sur imprimante

En mode **"FENETRE"**, la fenêtre désignée par le pictogramme représentant une 'main', sera recopiée sur imprimante en enfonçant la touche **"H"**.

Le masque entier est imprimé à l'exception des lignes blanches qui sont limitées à 3.

REMARQUE : Ces méthodes de copie d'écran s'appliquent aussi aux 3 fenêtres de l'éditeur MEMBASIC. On pourra ainsi faire une copie papier des lignes tracées ...

### 1.5.2 Copie par programme d'un masque sur imprimante

Une instruction a été incluse dans MEMSCREEN pour permettre l'impression d'une partie rectangulaire de taille quelconque en une seule fois.
Cette instruction LET "HARDCOPY..." est détaillée au chapitre 2.

Quelques exemples :

```
LET "HARDCOPY,A"
```

Imprime l'intégralité du masque d'identificateur temporaire "A".

```
LET "HARDCOPY,A,3,9"
```

Imprime seulement les lignes 3 à 9.

```
LET "HARDCOPY,A,3,9,1,79"
```

Imprime toujours les lignes 3 à 9, mais seulement entre les colonnes 1 et 79.



### 1.5.3 Les copies sélectives sur imprimante

Enfin, MEMSCREEN contient une instruction éditant une ligne de masque en fonction d'un caractère repère qu'elle contient. Par exemple, on pourra copier sur imprimante la première ligne rencontrée dans un masque commençant par un caractère spécial "A".
Cette méthode permet d'éditer des lignes de masque sans avoir à nommer explicitement leur position, ce qui offre une grande souplesse d'utilisation.
Voir pour plus d'informations l'instruction LET "?...." au chapitre 2.

### 1.5.4 Les masques d'édition

Les deux méthodes décrites en 1.5.2 et 1.5.3 permettent de réaliser des éditions de façon très simple.
Il suffit de concevoir un masque particulier, non destiné à être visible en fonctionnement normal, dont chaque ligne représente une ligne de l'imprimante.
Avec la méthode décrite en 1.5.3, le masque suivant peut servir à l'édition d'une liste de nom :

```
A     Liste des noms et adresses                page []]]
B*--------------------*-------------------------------*
C| []]]]]]]]]]]]]]]] | []]]]]]]]]]]]]]]]]]]]]]]]]]] |
```

L'édition de la ligne A puis de la ligne B puis d'un certain nombre de lignes C et enfin d'une ligne B permettent d'obtenir une page d'édition. Il suffit entre chaque édition de ligne, de remplir les zones par LET "OUTPUT...".
La fenêtre n'a pas besoin d'être visible pour servir à l'édition puisque, ni l'instruction LET "OUTPUT...", ni l'instruction LET "?..." ni l'instruction LET "HARDCOPY" ne demande une fenêtre visible.

Enfin ,la fonction MEMBASIC CURLINE permet de gérer le nombre de lignes imprimées et de provoquer les sauts de page à bon escient.

## 1.6 FICHIERS D'AIDE

Plusieurs fichiers d'aide sont disponibles avec le système MEMSOFT :

- Des fichiers standard qui contiennent des informations sur la syntaxe des instructions utilisées dans MEMBASIC, MEMFILE et MEMSCREEN.

- Des fichiers d'aide que le développeur peut facilement construire pour aider l'utilisateur de programmes d'application fonctionnant sous MEMSOFT.

### 1.6.1 Fichiers d'aide standard

En cas d'hésitation sur la façon d'employer certaines fonctionnalités de MEMSOFT, par exemple les fenêtres, l'éditeur, certaines commandes ou instructions, il est possible à tout moment faire appel à ces fichiers selon le procédé suivant :

- Appuyer sur la touche de fonction **POMME 0 (AIDE)**. Une fenêtre contenant l'aide la plus appropriée apparaît à l'écran.

- Grâce à la souris, en cliquant sur les flèches haute et basse de la 25 ème ligne de l'écran, on peut faire défiler le contenu de l'aide dans un sens ou dans l'autre.

A tout moment, en appuyant sur la touche **"ESCape"**,ou en cliquant sur ESC, l'utilisateur quitte le fichier d'Aide pour revenir à l'opération en cours.

La fin des fichiers d'AIDE contient souvent un menu qui permet d'appeler un autre fichier Aide.

Les fichiers d'aide standard se trouvent sur les disquettes fournies. Leurs noms sont de la forme : FR??????.HLP. Il ne peuvent être utilisés que s'ils sont présents dans le répertoire en cours ou dans l'un des répertoires donnés par l'instruction MEMBASIC PATH, ou dans le répertoire /MEMTOOLS.

### 1.6.2 Fichiers d'Aide "Application"

Dans un programme en cours d'exécution, le fichier d'aide appelé au moyen de la touche **POMME 0 (AIDE)** est en priorité le fichier défini par l'instruction HELP. Si aucun fichier n'a été défini par cette instruction, le



fichier d'aide recherché est d'abord celui qui porte le même nom que le masque MEMSCREEN utilisé, puis celui portant le nom du programme en cours et enfin le fichier d'Aide FRSOS.HLP.

Les fichiers d'Aide ont toujours le suffixe .HLP qui permet de les différencier des autres fichiers.

Par exemple, si le programme ESSAI.PRG contient les lignes :

```
100 LET "#OPEN,1,M,MA1"
110 LET "INPUT,1"
```

Si l'utilisateur enfonce la touche **POMME 0 (AIDE)** pendant l'exécution de la ligne 110, le fichier d'aide recherché sera :

- d'abord le fichier MA1.HLP,
- s'il n'est pas trouvé, le fichier ESSAI.HLP,
- enfin FRSOS.HLP.

Le mode d'emploi de la consultation des fichiers d'aide "Application" est le même que celui des fichiers d'aide standard décrits au chapitre précédent.

### 1.6.3 Création de fichiers d'aide

Le format des fichiers d'aide est celui des masques MEMSCREEN. En fait, tout masque MEMSCREEN peut servir de fichier d'aide. Il faut juste savoir que :

- La consultation se fera "page" à "page" (par page, on entend hauteur de la fenêtre implicite définie avec le masque).

- L'utilisateur ne pourra pas faire défiler le texte latéralement, il est donc inutile de mettre plus de texte en largeur que ce qui est visible dans la fenêtre.

La fenêtre qui est définie lors de la création du masque d'aide servira lors de la consultation de l'aide. C'est exactement dans ce format que l'aide sera visualisée.

NOTE : Bien que l'aide apparaisse dans une fenêtre, il est impossible, lors de la consultation, de modifier sa taille ou sa position. Une tentative d'enfoncer la touche **"OPTION CLIC-Souris"** fait quitter la consultation

de l'aide. Ce phénomène qui peut paraître curieux, s'explique par le fait que, pour des raison d'économie de place, les fichiers d'aides ne sont pas chargés complètement en mémoire, mais lus au fur et à mesure sur le disque.

Deux possibilités spécifiques aux masques d'aides :

- La possibilité de demander un saut de page n'importe où dans le texte.

  Pour cela, insérer une ligne contenant comme premier et seul caractère celui qui s'obtient en enfonçant **OPTION**, puis en tapant **14** sur clavier numérique et enfin en relâchant **OPTION**.

- La possibilité d'enchaîner à la fin de cette aide vers d'autres aides par l'intermédiaire d'un menu.

  Pour cela, indiquer en fin de texte, la liste des fichiers d'aides possibles et des touches à enfoncer pour les obtenir sous la forme :

  <lettre>:<nom de fichier sans suffixe>

  Il doit y avoir un enchaînement de ce type par ligne. Ces informations ne seront pas visibles lors de la consultation du fichier d'aide.

  La touche associée peut être n'importe quelle touche donnant un caractère du code ISO à l'exception de **"ESCape"** qui sert au retour et de **"RETOUR CHARIOT"** réservé au passage à la page suivante. Les touches "spéciales" commes les flèches, les touches de fonctions ... ne sont pas autorisées. Les lettres minuscules ou majuscules ont le même effet.

  Le suffixe du fichier n'est pas à indiquer. Ce sera toujours .HLP.

  Remarque : ♦ s'obtient en enfonçant **OPTION**, puis en tapant **4** sur le clavier numérique et enfin en relachant **OPTION**.



Par exemple, voici un petit fichier d'aide :

```
Quelle aide voulez-vous consulter ?

A - L'aide sur les zernisops
B - L'aide sur les dexuriozes
C - L'aide sur les varbusions

Cliquez la lettre correspondante.
♦a:zernisop
♦b:dexurioz
♦c:varbusio
```

L'utilisateur pourra alors obtenir l'aide appropriée en cliquant sur l'une des trois lettres A, B ou C.

## 1.7 SEQUENCES CLAVIER PRE-ENREGISTREES

Les séquences clavier pré-enregistrées permettent de mémoriser puis de rejouer des séquences de touches tapées au clavier. Ces séquences sont sauvegardées sur disque avec un suffixe .AUT.

L'enregistrement et la relecture de séquences clavier enregistrées se font à n'importe quel moment, quel que soit le programme et ne demandent aucune programmation.

Le chapitre 1.21 de la section II "MEMBASIC" explique comment créer et rejouer des séquences clavier. Se reporter à cette section pour plus de précisions.

# CHAPITRE 2
# INSTRUCTIONS MEMSCREEN

Le Chapitre 2 décrit l'ensemble des instructions disponibles avec MEMSCREEN.

L'ordre utilisé est l'ordre alphabétique car il permet de retrouver très rapidement une information cherchée.

Pour chaque instruction le même plan est utilisé. Ce plan est donné page suivante.

Chaque instruction nouvelle commence à une nouvelle page pour faciliter vos recherches.

Pour les instructions qui ne sont pas de la forme LET "....", consulter la section II "MEMBASIC".

Si une instruction recherchée ne se trouve pas dans cette section, consulter également la section IV "MEMFILE".

# LET "......" INSTRUCTION

## BUT

Il s'agit d'une brève description du but de l'instruction.

## SYNTAXE

Ce paragraphe décrit la syntaxe de l'instruction de façon exhaustive. La syntaxe est décrite selon la méthode employée dans la section II "MEMBASIC".

## EXPLICATION

Ce paragraphe décrit en détail le fonctionnement de cette instruction.

## EXEMPLE

Ce paragraphe contient une portion de programme utilisant l'item faisant l'objet du paragraphe et montrant donc sa mise en oeuvre.

## ERREURS

Il s'agit d'un petit paragraphe donnant la liste des erreurs qui peuvent être détectées par l'interpréteur MEMBASIC lors de l'exécution du programme.

## STATUS

Paragraphe non-systématique donnant l'effet de l'instruction sur la fonction STATUS.

## REMARQUE

Paragraphe non-systématique permettant des rapprochements entre des commandes, fonctions ou instructions différentes.



# LET "#CLEAR..." INSTRUCTION

## BUT

Supprime un masque de la mémoire.

## SYNTAXE

LET "#CLEAR,<IT>"

<IT> est un identificateur temporaire désignant un masque.

## EXPLICATION

Cette instruction supprime dans la mémoire le masque qui a été ouvert avec l'identificateur temporaire spécifié.

Ce masque n'est pas détruit sur disque; la destruction sur disque se fait par l'instruction : LET "#DELETE...".

Le masque supprimé en mémoire disparaît également de l'écran s'il était visible.

La suppression d'un masque en mémoire libère de la place.

Pour utiliser à nouveau le masque, il faut le réouvrir en utilisant l'instruction :
LET "#OPEN...".

## EXEMPLE

LET "#CLEAR,1" supprime de la mémoire et fait disparaître de l'écran le masque d'identificateur temporaire 1.

## STATUS

Si l'on tente de supprimer de la mémoire un masque qui n'a pas été ouvert au préalable, STATUS rend 2, sinon STATUS rend 0.

## ERREUR

Néant.

## REMARQUES

L'instruction LET "#CLEAR,<IT>" peut également servir à fermer un fichier (voir la section IV "MEMFILE").

Un masque peut être rendu invisible sur l'écran sans être fermé. Voir l'instruction LET "MODE...".



# LET "#CLEAR,$" INSTRUCTION

## BUT

Supprime de la mémoire tous les masques ouverts.

## SYNTAXE

LET "#CLEAR,$"

## EXPLICATION

Tous les masques qui ont été ouverts par l'instruction LET "#OPEN..." sont effacés de la mémoire et disparaissent de l'écran s'ils étaient visibles.

Ces masques ne sont pas détruits sur le disque mais sont seulement supprimés en mémoire centrale.

La suppression des masques en mémoire centrale libère de la place.

Pour utiliser à nouveau ces masques, il faut les réouvrir par l'instruction LET "#OPEN...".

## STATUS

Rend toujours 0.

## ERREUR

Néant.

## REMARQUES

LET"#CLEAR,$" ferme aussi tous les fichiers qui sont ouverts (voir la section IV "MEMFILE").

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "#DELETE..." INSTRUCTION

## BUT

Détruit un masque sur disque.

## SYNTAXE

LET "#DELETE,M,<NOM>"

Le nom devra respecter la syntaxe du système d'exploitation; il pourra contenir des spécifications de disques ou de répertoires. Si aucun suffixe n'est spécifié, le suffixe implicite est .MSK.

## EXPLICATION

L'instruction LET "#DELETE..." détruit le masque spécifié sur le disque.

Il n'est pas nécessaire pour cela que le masque ait été ouvert au préalable. Néanmoins, si cela était le cas, le masque reste ouvert après la destruction et peut donc être encore utilisé dans votre programme bien qu'il n'existe plus sur disque.

## EXEMPLE

LET "#DELETE,M,/SUB/MASK1" détruit le masque MASK1 du sous-répertoire SUB du répertoire principal (racine) sur le disque implicite.

## STATUS

Si le masque désigné n'existe pas, la fonction STATUS rendra 10, sinon 0.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".



# LET "#NEW..." INSTRUCTION

## BUT

Créer un masque

## SYNTAXE

LET "#NEW,M, [@] <nom>"

Le nom devra respecter la syntaxe du système d'exploitation; il pourra contenir des spécifications de disque ou de répertoire.

Si le nom est précédé de l'option @, le masque créé pourra porter le nom d'un objet déjà existant sur le disque. Cet objet sera alors détruit. Si aucun suffixe n'est spécifié, le suffixe implicite est .MSK.

## EXPLICATION

L'instruction LET "#NEW..." donne la main à l'utilisateur qui doit définir son masque.
Il définit :

- la taille de la fenêtre attribuée au masque,

- le texte du masque,

- les zones du masque avec leurs caractéristiques.

Se reporter au chapitre 1 où la phase de création du masque est détaillée avec des exemples.

Lorsque la création du masque est terminée, le programme continue et la fonction STATUS permet de savoir comment s'est passée la création.

Si la fonction STATUS vaut 0, le masque a été validé par l'utilisateur, sinon, la création du masque a été annulée (par enfoncement de ESCape) et, dans ce cas, le masque n'a pas été sauvé sur disque.

Dans le cas où la création se passe normalement, le masque est sauvé sur le disque avec le nom qui a été indiqué.

## EXEMPLE

```
LET "#NEW,M,A:BIBLI/MASK"
```

Cette instruction crée le masque MASK dans le sous-répertoire BIBLI du répertoire en cours sur le disque A.

On peut faire suivre cette instruction de création par IF STATUS... pour tester si le masque a été créé ou non.

## STATUS

Rend 0 si la création s'est bien passée.
Rend 1 en cas d'abandon.
Rend 30 si l'objet est déjà existant et qu'il n'y a pas "@" devant le nom.
Rend 50 si une zone n'est pas correcte syntaxiquement : pas de nom de variable pour une zone en entrée, erreur dans un nom de variable ...

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUES

Cette instruction ne permet pas de modifier un masque déjà existant; pour cela, utiliser l'instruction LET "#UPDATE...".

Pour utiliser le masque après sa création, l'ouvrir avec l'instruction LET "#OPEN...".

Ne pas oublier le "@" avant le nom si un fichier déjà existant doit être écrasé.



# LET "#OPEN..." INSTRUCTION

## BUT

Ouvre un masque, c'est-à-dire le charge en mémoire. Le masque peut alors être utilisé pour faire des saisies ou des affichages.

## SYNTAXE

```
LET "#OPEN [<option>],<IT>,M,<NOM>"
```

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire est un caractère du code ISO à l'exception des caractères $ et ^ et des caractères de code inférieur ou égal à 32.

Le nom devra respecter la syntaxe du système d'exploitation, il pourra contenir des spécifications de disques ou de répertoires. Si aucun suffixe n'est spécifié, le suffixe implicite est .MSK.

L' <option> peut être /V ou /U. Elle indique le mode implicite des masques.

## EXPLICATION

LET "#OPEN..." ouvre le fichier contenant le masque, charge le masque en mémoire centrale. Le masque, bien que non visible, est alors utilisable pour les saisies et les affichages.

Avec l'option /V le masque reste visible après utilisation en saisie ou affichage même s'il ne sert plus, avec /U ou sans option, le masque est rendu invisible lorsque un autre masque est utilisé. Cette option pourra être modifiée ultérieurement par l'instruction LET "MODE...".

L'identificateur temporaire qui est défini lors de cette instruction d'ouverture devra être utilisé dans les instructions d'affichage et de saisie ultérieures.

L'identificateur temporaire choisi devra être différent de ceux déjà attribués à d'autres masques ou fichiers (voir MEMFILE).

## EXEMPLE

```
LET "#OPEN,X,M,MAS.C1"
```

ouvre le masque MAS.C1 et lui attribue l'identificateur temporaire X.

## STATUS

Rend 0 si l'ouverture s'est bien passée, 10 si le masque désigné n'existe pas.

## ERREURS

L'erreur No 27007 est détectée si l'identificateur temporaire choisi est déjà utilisé par un masque ou par un fichier (voir MEMFILE).

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

Si le masque n'a pas été trouvé avec les spécifications demandées, il est recherché également dans les répertoires spécifiés par l'instruction PATH de MEMBASIC (Voir l'instruction MEMBASIC PATH).



# LET "#UPDATE..." INSTRUCTION

## BUT

Permet de modifier un masque qui a été créé au préalable par l'instruction LET "#NEW,M,<NOM>".

## SYNTAXE

```
LET "#UPDATE,M,<NOM>"
```

Le nom qui permet de désigner le masque à modifier doit être conforme à la syntaxe du système d'exploitation utilisé. Il pourra contenir des spécifications de disques ou de répertoires. Si aucun suffixe n'est spécifié, le suffixe implicite est .MSK.

## EXPLICATION

L'instruction LET "#UPDATE..." ouvre le fichier contenant le masque, le charge en mémoire, et le propose à la modification.

La phase de modification du masque est à peu près identique à la phase de création à l'exception près que le masque initial, au lieu d'être vide comme lors de la création, contient les informations précédentes du masque.

Se reporter au chapitre 1 où la phase de modification du masque est détaillée.

A l'issue de la modification l'utilisateur pourra abandonner la modification ou la valider et, suivant le cas, le masque conservera son contenu précédent ou prendra le contenu modifié.

En cas de validation de la modification, le nouveau masque remplace l'ancien sur le disque.

Si la modification est validée, la fonction STATUS rendra 0, sinon 1.

## EXEMPLE

```
LET "#UPDATE,M,M1"
IF STATUS THEN PRINT "Masque non modifié"
```

Cet exemple propose la modification du masque M1 et signale un abandon.

## STATUS

Rend 0 si le masque a été modifié, 20 si le masque désigné n'existe pas, 1 en cas d'abandon.
Rend 50 si une zone n'est pas correcte syntaxiquement : pas de nom de variable pour une zone en entrée, erreur dans un nom de variable ...

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUES

Si l'on désire créer un nouveau masque proche d'un masque déjà existant, la méthode la plus simple consiste à copier le masque déjà existant proche du masque à réaliser sur un autre fichier ayant pour nom celui du masque à créer et d'utiliser l'instruction de modification de masque sur ce nouveau fichier.

Pour cela, il suffit de savoir que le suffixe implicite, donné par MEMSCREEN au masque est .MSK.

Le masque est contenu en un seul fichier du système d'exploitation. L'instruction COPY de MEMBASIC permet d'effectuer cette copie de fichiers.

Exemple : COPY "ANCIEN.MSK" TO "NOUVEAU.MSK"



# LET "?,..." INSTRUCTION

## BUT

Rechercher et recopier une ligne de masque sur le périphérique de sortie implicite.

## SYNTAXE

LET "?,<repère>"

Le repère est un caractère du jeu normalisé ISO supérieur à 32.

## EXPLICATION

Sur le masque qui a été utilisé en dernier par une instruction MEMSCREEN, la première ligne trouvée commençant par le caractère repère est recopiée sur le périphérique de sortie en cours (qui a été défini par l'instuction PRINTER), à l'exception du caractère repère qui n'est pas recopié.

ATTENTION ! C'est sur le dernier masque concerné par une instruction MEMSCREEN que la recherche est effectuée même si le masque n'est pas visible.

## EXEMPLE

LET "?,A" imprime la première ligne commençant par un A du dernier masque utilisé.

Dans l'exemple du masque suivant, la ligne recopiée sera : "Giboulées de Mars".

```
XVoici un test de masque et d'éditions
AGiboulées de Mars
BAvril
Acette ligne n'est pas imprimée
```

## ERREUR

Néant.

## REMARQUES

Si plusieurs lignes commencent par le caractère repère choisi, seule la première est recopiée.

Si le caractère repère n'est trouvé sur aucune ligne du masque, l'instruction a pour effet de réaliser un interligne.



# LET "CHARGE..." INSTRUCTION

## BUT

Afficher le texte du masque et le texte seul.

## SYNTAXE

LET "CHARGE [<option>],<identificateur temporaire>"

<option> peut être /V ou /U.

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire est celui qui est attribué lors de l'ouverture du masque.

Rappelons que l'identificateur temporaire est un caractère du code ISO à l'exception de $, de ^ et des caractères de code inférieur ou égal à 32.

## EXPLICATION

Lorsque l'instruction LET "CHARGE..." est exécutée, la fenêtre contenant le masque devient apparente si elle ne l'était pas et est placée au premier plan sur l'écran.

De plus, cette instruction chargeant le texte seul, les zones de saisie ou d'affichage ne sont pas affichées et sont remplacées par des blancs.

L'option, qui peut être /V ou /U, indique si l'on désire que le masque reste visible lorsqu'il ne sera plus utilisé ou si l'on désire qu'il disparaisse.

OPTION /V :

Le masque restera visible même lorsque l'on utilisera d'autres masques par la suite. Néanmoins, il ne restera pas forcément au premier plan, car l'utilisation des instructions LET "MODE...", LET "CHARGE..." ou LET "PRINT..." utilisées avec un autre masque, font passer celui-ci au premier plan.

OPTION /U :

Le masque disparaîtra dès que le programme fera appel à un autre masque. Il restera sur l'écran tant qu'aucun appel à un autre masque ne sera effectué. Même après avoir disparu, il sera ramené sur l'écran si une opération de saisie est effectuée sur ce masque par l'instruction LET "INPUT ...".

OPTION IMPLICITE :

L'option implicite est celle qui a été indiquée à l'ouverture. Si aucune option n'a été précisée à l'ouverture, c'est l'option /U (invisible) qui est choisie.

## EXEMPLE

```
LET"#OPEN,1,M,MASK"
LET"CHARGE,1"
```

Ces deux instructions réalisent :

- l'ouverture du masque MASK avec l'identificateur temporaire 1,
- l'affichage du texte de ce masque sur l'écran.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire utilisé ne correspond à aucun masque ou correspond à un fichier (voir MEMFILE).



# LET "HARDCOPY..." INSTRUCTION

## BUT

Recopier sur le périphérique de sortie une partie d'un masque.

## SYNTAXE

```
LET"HARDCOPY [<option>] ,<IT>[,<lmini>[,<lmaxi>
                              [,<cmini> [,<cmaxi>]]]]
```

<IT> est un identificateur temporaire désignant un masque.

Les paramètres optionnels <lmini> et <lmaxi> indiquent les numéros de ligne minimale et maximale qui sont éditées.
Ils peuvent varier de 1 à 250.

Les paramètres optionnels <cmini> et <cmaxi> indiquent les numéros de colonne minimale et maximale qui sont éditées.
Ils peuvent varier de 1 à 250.

<option> peut être /K ou rien.

## EXPLICATION

Sans paramètre optionnel, tout le masque est copié.
Si <lmini> est spécifiée les lignes à partir de <lmini> incluse sont copiées.
Si <lmaxi> est spécifiée la copie s'arrête à <lmaxi> incluse.
Si <cmini> est spécifiée, les lignes ne sont copiées qu'à partir de la colonne <cmini> incluse.
Si <cmaxi> est spécifiée les lignes ne sont copiées que jusqu'à la colonne <cmaxi> incluse.

La copie se fait sur le périphérique de sortie en cours spécifié par l'instruction MEMBASIC PRINTER.

L'option /K permet d'éditer la ligne sans RETOUR CHARIOT. Ceci permet d'avoir plusieurs largeurs de caractères sur une même ligne.

## EXEMPLE

```
LET"HARDCOPY,A,1,250,1,80"
```

copie toutes les lignes en tronquant à 80 colonnes.

## ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire utilisé ne correspond à aucun masque ou correspond à un fichier (voir MEMFILE).

## REMARQUES

Les 250 lignes possibles du masque ne sont pas recopiées; seules les lignes existantes le sont.

Si la valeur de <lmaxi> est absente ou supérieure à la dernière ligne existante, les lignes inutiles sont ignorées.

La copie sur papier ne reproduit bien entendu pas les couleurs.



# LET "INPUT..." INSTRUCTION

## BUT

Saisir des zones du masque qui ont été définies en entrée.

## SYNTAXE

```
LET "INPUT [/K], <IT>, [ <zentrée>
                         [,<zmini> [,<zmaxi>] ] ]"
```

/K est un paramètre optionnel interdisant la fin de la saisie par d'autres moyens que l'enfoncement d'une touche de fonction ou de la touche ESC.

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire désigne le masque à l'aide duquel on veut faire une saisie.
Il doit correspondre à celui qui a été donné au masque lors de son ouverture.

Le paramètre <zentrée> indique à quelle zone on doit commencer la saisie (à partir de 1). La valeur particulière 0 indique qu'aucune zone n'est à saisir.

Les paramètres optionnels <zmini> et <zmaxi> indiquent les numéros des zones minimale et maximale entre lesquelles doit s'effectuer la saisie. Seules les zones en entrée sont prises en compte, la première zone en entrée étant numérotée 1.

Ces paramètres ne sont autorisés que si le paramètre <zentrée> a été renseigné et n'est pas nul. La zone maximale doit être supérieure ou égale à la zone minimale, et la zone d'entrée comprise entre les deux.

## EXPLICATION

Si le masque n'était pas visible, il le devient et l'utilisateur a la main dans la première zone de saisie ou dans la première zone de saisie indiquée dans l'instruction. Si le numéro de la première zone à saisir est nul, aucune zone ne sera saisie. On attendra uniquement que l'utilisateur enfonce

une des touches de fonction autorisée ou ESC, RETOUR CHARIOT ou une touche de déplacement.

Si cette instruction n'est pas précédée d'une instruction LET "CHARGE..." ou LET "PRINT..." le masque, bien que redevenant visible, ne passera pas au premier plan, sauf si la saisie le nécessite (c'est-à-dire, si l'endroit sur l'écran où se trouve la zone de saisie est recouvert par une autre fenêtre).

Lors de l'exécution de cette instruction, l'utilisateur saisit successivement les différentes zones en entrée spécifiées en commençant, soit au début du masque, soit à la zone d'entrée si elle est précisée.
Il pourra passer d'une zone à la suivante par la touche "RETOUR CHARIOT".
Il pourra remonter effectuer des corrections dans les zones déjà saisies par la touche POMME ↑ ou descendre plus bas dans l'écran par la touche POMME ↓.
L'utilisateur pourra choisir directement la zone à modifier en la désignant avec le curseur de la souris et en enfonçant puis relâchant le bouton de celle-ci.

Contrôles :

Si des zones minimale et maximale ont été spécifiées, il sera impossible de placer le curseur en dehors de ces zones.
Après la saisie de chaque zone, les contrôles demandés sont effectués et la saisie ne peut se poursuivre tant que la valeur entrée n'est pas validée.

Sortie :

La sortie n'est possible que si tous les contrôles de toutes les zones à saisir sont vérifiés. Sinon, une tentative de fin de saisie ramènera l'utilisateur dans les zones invalides jusqu'à correction complète. La seule exception est la sortie par abandon (ESC) qui permet une sortie dans tous les cas.



L'utilisateur sortira définitivement de la saisie par l'une des méthodes suivantes :

- Validation par RETOUR CHARIOT, flèche basse ou POMME ↓ de la dernère zone demandée ou enfoncement de la flèche haute ou de POMME ↑. La fonction EXKEY rend alors 0. La fonction EXWAY permet de distinguer ces 5 cas, en rendant :

  0 pour RETOUR CHARIOT
  1 pour flèche basse
  -1 pour flèche haute
  2 pour POMME ↓
  -2 pour POMME ↑

  Le même résultat est obtenu avec une souris si l'utilisateur tente de sélectionner une zone située en dehors de la partie de la fenêtre autorisée pour la saisie.

  A noter: l'option /K interdit à l'utilisateur l'emploi, pour quitter la saisie, des 3 possibilités évoquées ci-dessus. Cela évite de les tester pour retourner à la saisie si l'on désire imposer une validation par touche de fonction.

- Enfoncement de la touche ESCAPE, ce qui signifie que l'utilisateur désire abandonner cette saisie; ce cas pourra être détecté par la variable STATUS qui vaudra 1, alors qu'elle vaudra 0 dans tous les cas de sortie normale, EXKEY rend -1,

- Enfoncement d'une quelconque touche de fonction pour laquelle un texte est visible en bas d'écran.
  Dans ce dernier cas, pour pouvoir savoir quelle touche de fonction l'utilisateur a enfoncé pour terminer la saisie, la fonction EXKEY indique un numéro de 1 à 9 correspondant à la touche enfoncée.
  Si l'utilisateur possède une souris, il pourra choisir une touche de fonction en pointant le message associé à cette touche sur la dernière ligne de l'écran et en enfonçant et relâchant un des deux boutons.

La fonction MEMBASIC EXZONE :

La fonction sans paramètre EXZONE indique le numéro de zone en sortie ou en entrée, d'où la saisie a été stoppée.
Cela permet de recommencer la saisie au même endroit si l'on veut que le reste du masque soit saisi.

Recalcul :

Si l'option recalcul a été demandée sur la zone, après saisie, l'ensemble des zones en sortie du masque est réaffiché pour tenir compte, dans les formules de calcul, de la nouvelle valeur.

Si l'utilisateur est sorti par "RETOUR CHARIOT" sur la dernière zone, EXKEY prendra la valeur 0.

## EXEMPLES

```
LET "#OPEN,1,M,MASK"
LET "PRINT,1"
LET "INPUT,1"
```

Ces trois instructions permettent successivement :

- d'ouvrir le masque MASK sous l'identificateur temporaire 1,
- d'afficher le texte sur l'écran ainsi que le contenu actuel des variables,
- de saisir de nouvelles valeurs aux zones en entrée.

## STATUS

Rend 1 en cas d'abandon (ESC), 0 sinon.

## EXKEY

Rend le code de la touche de sortie du masque :

```
-1 ....... ESCape
 0 ....... RETOUR CHARIOT, flèches ou POMME ↓,
           ou POMME ↑ , suivant EXWAY
 1 à 9 ... touches POMME 1 à POMME 9
```



## EXWAY

Rend un code indiquant, si EXKEY vaut 0, de quelle façon exacte la sortie a eu lieu.

```
 0 ....... RETOUR CHARIOT
 1 ....... flèche basse
-1 ....... flèche haute
 2 ....... pour POMME ↓
-2 ....... pour POMME ↑
```

## EXZONE

Rend le numéro de la zone en entrée d'où l'on a stoppé la saisie.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun masque ouvert ou correspond à un fichier MEMFILE.

Si le nom ou la formule de contrôle d'une zone provoque une erreur, l'erreur MEMBASIC correspondante sera détectée. Se reporter à la section II "MEMBASIC" pour ces erreurs.

## REMARQUES

Il est préférable avant d'effectuer une saisie par l'ordre LET "INPUT..." d'afficher le masque et les valeurs précédentes des variables par l'instruction LET "PRINT..." pour éviter toute surprise à l'utilisateur.

En effet, si vous utilisez l'instruction LET "CHARGE..." avant de faire la saisie, l'utilisateur ne verra que des zones blanches, ce qui ne signifie pas que les variables ont une valeur nulle avant la saisie, mais simplement qu'elles n'ont pas été affichées.

En conséquence, si l'utilisateur interrompt la saisie par l'utilisation d'une des touches de fonction, les zones du bas de masque qui n'auront pas été saisies, bien que vides sur l'écran, conserveront leurs valeurs précédentes.

Si le masque est déjà présent sur l'écran et que vous ne désirez pas le faire passer au premier plan, utilisez l'instruction LET "OUTPUT..." pour afficher la valeur actuelle des zones avant la saisie.



# LET "KEYBOARD" INSTRUCTION

## BUT

Attendre une validation ou annulation sans saisir de zone de masque.

## SYNTAXE

**LET "KEYBOARD [/K]"**

**/K est un paramètre optionnel interdisant la fin de la saisie par d'autres moyens que l'enfoncement d'une touche de fonction ou de la touche ESC.**

## EXPLICATION

Cette instruction provoque un test clavier non lié à l'utilisation d'un masque particulier.

On teste uniquement si l'utilisateur a enfoncé l'une des touches de fonction autorisée ou ESC, RETOUR CHARIOT ou une touche de déplacement vertical.

Le test du clavier ne bloque pas le programme définitivement dans l'attente d'une réponse : au bout d'un temps relativement bref sans activité de l'utilisateur, un résultat particulier est rendu permettant au programme de s'aiguiller vers d'autres actions.

Les différentes possibilités sont :

- Enfoncement de RETOUR CHARIOT, flèche basse ou flèche haute, POMME ↑ ou POMME ↓ : la fonction EXKEY rend alors 0. La fonction EXWAY permet de distinguer ces 3 cas, en rendant :

  0 pour RETOUR CHARIOT
  1 pour flèche basse
  -1 pour flèche haute
  2 pour POMME ↓
  -2 pour POMME ↑

A noter: l'option /K interdit à l'utilisateur l'emploi, pour quitter saisie, des 3 possibilités évoquées ci-dessus. Cela évite de les tester pour retourner à la saisie si l'on désire imposer une validation par touche de fonction.

- Enfoncement de la touche ESCAPE, ce qui signifie que l'utilisateur désire abandonner cette saisie; ce cas pourra être détecté par la variable STATUS qui vaudra 1, alors qu'elle vaudra 0 dans tous les cas de sortie normale, EXKEY rend -1,

- Enfoncement d'une quelconque touche de fonction pour laquelle un texte est visible en bas d'écran :
  Dans ce cas, pour connaitre la touche de fonction enfoncée par l'utilisateur, la fonction EXKEY indique un numéro de 1 à 9 correspondant à la touche enfoncée.
  Si l'utilisateur utilise la souris, il pourra choisir une touche de fonction en pointant le message associé à cette touche sur la dernière ligne de l'écran et en enfonçant et relâchant l'un des deux boutons.

- Pas d'action clavier :
  La fonction EXKEY indique alors le code -2. C'est au programme de "boucler" s'il le désire jusqu'à obtention d'une réponse satisfaisante.

## EXEMPLES

```
100 DO
110    IF LEFT$(time$,4) = "12:0" THEN
120       PRINT " Il est midi docteur ..."
130       EXIT DO
140    END IF
110    LET "Keyboard"
120 LOOP WHILE EXKEY=-2
```

## STATUS

Rend 1 en cas d'abandon (ESC), 0 sinon.



## EXKEY

Rend le code de la touche de sortie du masque :

```
-2 ..... Pas de touche enfoncée
-1 ..... ESCape
 0 ..... RETOUR CHARIOT ou POMME ↑ ou POMME
         ↓ suivant EXWAY
 1 à 9 ... touches POMME 1 à POMME 9.
```

## EXWAY

Rend un code indiquant, si EXKEY vaut 0, de quelle façon exacte la sortie a eu lieu.

```
 0 ..... RETOUR CHARIOT
 1 ..... flèche basse
-1 ..... flèche haute
 2 ..... pour POMME ↓
-2 ..... pour POMME ↑
```

## REMARQUES

Attention ! aucune garantie n'est apportée sur le temps de réponse de l'instruction dans le cas où aucune touche n'a été enfoncée.

# LET "MODE..." INSTRUCTION

## BUT

Change l'option par défaut /V ou /U.

## SYNTAXE

LET "MODE [<option>] , <IT>"

<IT> est un identificateur temporaire désignant un masque.

<option> peut être /V ou /U.

## EXPLICATION

LET "MODE/V..." rendra le masque visible en permanence.

LET "MODE/U..." rendra le masque invisible lorsqu'il ne sera pas utilisé.

## ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun masque ouvert ou correspond à un fichier MEMFILE.

## STATUS

Rend une valeur non significative.



# LET "OUTPUT..." INSTRUCTION

## BUT

Cette instruction sert à afficher le contenu des variables d'un masque.

## SYNTAXE

LET "OUTPUT [<option>],<IT>[,<typzone> [,<zmini> [,<zmaxi>]]]"

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire doit correspondre à celui qui a été donné au masque lors de son ouverture.

<option> peut être /V ou /U.

<typzone> peut être A,I ou O.

Ce paramètre optionnel permet de savoir si l'affichage doit s'effectuer sur toutes les variables (A), sur les variables en entrée (I), ou sur les variables en sortie (O).

Les paramètres optionnels <zmini> et <zmaxi> représentent les numéros des zones minimale et maximale qui devront être affichées.

Ces paramètres ne sont autorisés que si une option A,I ou O est indiquée.

Les numéros de zones sont comptés à partir de 1 en tenant compte de toutes les zones.

## EXPLICATION

L'instruction LET "OUTPUT..." ne nécessite pas que le masque soit visible pour être effectuée. Il peut même être particulièrement agréable, (en particulier pour les masques d'édition), que le masque ne soit pas visible bien que l'on effectue des affichages.

Avant l'affichage de chaque zone concernée, la formule de contrôle que vous avez rentrée dans les

caractéristiques de chaque zone est évaluée et la zone n'est affichée que si la condition est réalisée.

Par exemple si la zone à afficher est la variable SOMME, et la condition est SOMME>0, si SOMME vaut 50 la zone sera affichée, si SOMME vaut -10, la zone sera blanche.

L'option, qui peut être /V ou /U, indique si l'on désire que le masque reste visible lorsqu'il ne sera plus utilisé ou si l'on désire qu'il disparaisse.

OPTION /V :

Le masque restera visible même lorsque l'on utilisera d'autres masques par la suite. Néanmoins, il ne restera pas forcément au premier plan, car l'utilisation d'instructions LET "CHARGE..." ou LET "PRINT..." utilisées avec un autre masque, font passer ce dernier au premier plan.

OPTION /U :

Le masque disparaîtra dès que le programme fera appel à un autre masque. Il restera sur l'écran tant qu'aucun appel à un autre masque ne sera effectué. Même après avoir disparu, il sera ramené sur l'écran s'il est l'objet d'une instruction de saisie.

OPTION IMPLICITE :

L'option implicite est celle qui a été indiquée à l'ouverture. Si aucune option n'a été précisée , c'est l'option /U (invisible) qui est choisie.

## EXEMPLES

```
LET "#OPEN,3,M,M1"
LET "CHARGE,3"
LET "OUTPUT,3"          ! -affiche tout
LET "OUTPUT,3,A,2,6" ! -affiche les zones en entrée
                        !  et en sortie de 2 à 6
```

Ces instructions permettent successivement :
- d'ouvrir le masque M1,
- de charger le texte seul de ce masque,



- d'afficher le contenu des variables.

LET "OUTPUT,3,A,2,6" ! affiche les zones de 2 à 6 est un exemple d'édition partielle.

## STATUS

Non significatif.

## ERREURS

Si l'identificateur temporaire choisi ne correspond à aucun masque ouvert, l'erreur No 27003 est détectée.

Si le nom ou la formule de contrôle d'une zone provoque une erreur, l'erreur MEMBASIC correspondante sera détectée. Se reporter à la section II "MEMBASIC" pour ces erreurs.

## REMARQUE

Dans le cas où l'instruction LET "OUTPUT..." suit une instruction LET"CHARGE...", comme dans l'exemple que nous avons vu précédemment, il est préférable de la remplacer par l'instruction LET"PRINT..." qui équivaut à l'ensemble de ces deux opérations.

# LET "PRINT..." INSTRUCTION

## BUT

Afficher le texte d'un masque et le contenu des variables d'un masque.

## SYNTAXE

```
LET "PRINT [<option>],<IT>[,<typzone>[,<zmini>
                                    [,<zmaxi>]]]"
```

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire doit correspondre à celui qui a été donné au masque lors de son ouverture.

<option> peut être /V ou /U.

<typzone> peut être A, I ou O.

Ce paramètre optionnel permet de savoir si l'affichage doit s'effectuer sur toutes les variables (A), sur les variables en entrée (I) ou sur les variables en sortie (O).

Les paramètres optionnels <zmini> et <zmaxi> indiquent les numéros des zones minimale et maximale qui seront affichées.
Les numéros de zones sont comptés à partir de 1 en tenant compte de toutes les zones.

## EXPLICATION

L'instruction LET "PRINT..." est équivalente à LET "CHARGE..." suivie de LET "OUTPUT...".

Le masque devient visible s'il ne l'était pas.

Avant l'affichage de chaque zone concernée, la formule de contrôle que vous avez rentrée dans les caractéristiques de chaque zone est évaluée et la zone n'est affichée que si la condition est réalisée.

Par exemple si la zone à afficher est la variable SOMME, et la condition est SOMME>0. Si SOMME vaut



50 la zone sera affichée, si SOMME vaut -10, la zone sera blanche.

L'option, qui peut être /V ou /U, indique si l'on désire que le masque reste visible lorsqu'il ne sera plus utilisé ou si l'on désire qu'il disparaisse.

OPTION /V :

Le masque restera visible même lorsque l'on utilisera d'autres masques par la suite. Néanmoins, il ne restera pas forcément au premier plan, car l'utilisation d'instructions LET "CHARGE..." ou LET "PRINT..." utilisées avec un autre masque, font passer ce dernier au premier plan.

OPTION /U :

Le masque disparaîtra dès que le programme fera appel à un autre masque. Il restera sur l'écran tant qu'aucun appel à un autre masque ne sera effectué. Même après avoir disparu, il sera ramené sur l'écran s'il est l'objet d'une instruction de saisie.

OPTION IMPLICITE :

L'option implicite est celle qui a été indiquée à l'ouverture. Si aucune option n'a été précisée , c'est l'option /U (invisible) qui est choisie.

## EXEMPLE

```
LET "#OPEN,A,M,M1"
LET "PRINT,A"
```

Ces deux instructions permettent :
- d'ouvrir le masque,
- de charger le texte du masque et d'afficher le contenu des variables.

## STATUS

Rend un résultat non significatif.

ERREURS

Si l'identificateur temporaire choisi ne correspond à aucun masque ouvert, l'erreur No 27003 est détectée.

Si le nom ou la formule de contrôle d'une zone provoque une erreur, l'erreur MEMBASIC correspondante sera détectée. Se reporter à la section II "MEMBASIC" pour ces erreurs.



# LET "SCROLL..." INSTRUCTION

## BUT

Fait défiler ou efface des lignes d'un masque et insère une ligne blanche.

## SYNTAXE

LET "SCROLL [<option>],<IT>,<lmini>,<lmaxi>"

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire doit correspondre à celui d'un masque ouvert.

<option> peut être /U /D ou /C .

<lmini> et <lmaxi> sont des numéros de lignes. Ces paramètres varient de 1 à 250.

## EXPLICATION

Suivant l'option choisie, cette instruction fait défiler un certain nombre de lignes du texte vers le haut ou fait défiler des lignes vers le bas ou efface ces lignes.

Les lignes concernées sont les lignes de <lmini> à <lmaxi> incluses.

```
LET "SCROLL/U..."   défilement vers le haut
LET "SCROLL/D..."   défilement vers le bas
LET "SCROLL/C..."   effacement
```

L'intérêt de cette instruction est de faire apparaître, grâce à quelques lignes de programme, une liste de valeurs dans un masque, et cela avec une seule ligne de variables. Il suffit de réserver un espace libre dans le masque, la dernière ligne comprenant une zone. Le programme utilisateur pourra être :

```
100 FOR i=0 TO 10
110   ........        ! détermine a$ à afficher
120   LET "OUTPUT,1" ! affiche a$ en ligne 7
130   LET "SCROLL,1,3,7" ! défilement
140   NEXT
```

ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire utilisé ne correspond à aucun masque ouvert ou correspond à un fichier MEMFILE.



# LET "TAKE..." INSTRUCTION

## BUT

Remet à jour les variables dans MEMBASIC en accord avec le contenu actuel d'un masque.

## SYNTAXE

LET "TAKE [<option>],<IT>"

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire doit correspondre à celui d'un masque ouvert.

<option> peut être /Z ou rien.

## EXPLICATION

Dans la forme sans option /Z, toutes les variables en entrée du masque sont mises à jour en fonction de leur valeur dans le masque.

Avec l'option /Z une remise à zéro a lieu aussi bien sur l'écran qu'en mémoire.

## EXEMPLE

```
100 ! nouvelle entrée
110 LET "TAKE/Z,1" ! vide zones et variables
120 LET "INPUT,1" ! saisie sur zones vides
...
...
500 LET "INPUT,1" ! saisie des valeurs
510 x=1    ! détruit une des valeurs saisies en 500
520 LET "TAKE,1"  ! restitue cette valeur
```

## ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire utilisé ne correspond à aucun masque ouvert ou correspond à un fichier MEMFILE.

# LET "Y..." INSTRUCTION

## BUT

Permet de faire déplacer verticalement, par l'utilisateur, une barre horizontale en fond inverse dans un rectangle défini d'un masque.

## SYNTAXE

```
LET "Y [<option>],<IT> [,<pos>,<lmin>,<lmax>
                                    ,<cmin>,<cmax>]"
```

<IT> est un identificateur temporaire désignant un masque.

L'identificateur temporaire doit correspondre à celui d'un masque ouvert.

<option> peut être /K ou rien.

Les paramètres <lmin>, <lmax>, <cmin> et <cmax> permettent de définir le rectangle (lignes et colonnes minimales et maximales) dans lequel la barre inverse pourra se déplacer.

Le paramètre <pos> représente la ligne sur laquelle sera positionnée initialement la barre inverse relativement au périmètre défini. La valeur 1 représente toujours la première ligne du rectangle.

## EXPLICATION

Une barre inverse, de la largeur du rectangle défini par les paramètres, se déplace verticalement dans ce rectangle à l'utilisation des flèches haute et basse ou de la souris.

Les lignes ne comportant que des caractères graphiques ou blancs ne sont pas prises en compte lors du déplacement de la barre.

L'option /K empêche la sortie du rectangle par les flèches haute et basse (cf instruction LET "Input ...").

Le STATUS sera à 1 en cas d'abandon (ESC), à 0 sinon.



EXKEY rend : -1 si sortie par ESC.
0 si sortie par RETOUR CHARIOT, ou par les flèches haute et basse aux limites du rectangle, ou par le choix d'une ligne par la souris.
1 à 9 si sortie par les touches de fonction POMME 1 à POMME 9.

Si EXKEY vaut 0, EXWAY indique la touche qui a provoqué la sortie (-1, 0, 1, 2 ou -2) selon les cas. (cf instruction LET "Input ...").

Dans tous les cas EXZONE rend la position de la barre au moment de la sortie relativement au rectangle ( 1 pour la première ligne du rectangle).

## EXEMPLE

```
100 LET "#Open,M,Mask,menu"
110 LET "Y,M,2,5,7,3,8"
```

Fait déplacer la barre entre les lignes 5 et 7 (soit 3 lignes). La barre commence en colonne 3 et finit en colonne 8. La position initiale de la barre est la deuxième ligne du rectangle (soit la ligne 6 du masque).

## ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire utilisé ne correspond à aucun masque ouvert ou correspond à un fichier MEMFILE.

# INDEX THEMATIQUE

## LISTE DES INSTRUCTIONS

Voici les différentes instructions de MEMSCREEN. Chaque instruction est suivie d'un commentaire sur son utilisation. Pour en savoir plus, se reporter au chapitre 2 où toutes ces instructions sont décrites en détail, classées par ordre alphanumérique.

# SECTION IV
# MEMFILE

## P L A N

# CHAPITRE 1
# DESCRIPTION GENERALE

## 1.1 INTRODUCTION

MEMFILE est un gestionnaire de fichiers. Il permet de sauver et de retrouver sur disque de grandes quantités de données classées sous forme de fiches ou enregistrements.

Un fichier MEMFILE peut se décrire simplement de la façon suivante :

- D'une part l'ensemble de toutes les fiches
- D'autre part, un ensemble de clés permettant de retrouver les fiches.

Deux familles d'opérations peuvent être effectuées sur les fichiers MEMFILE : les opérations qui agissent globalement sur le fichier, et celles qui agissent sur les enregistrements du fichier.

Le chapitre 2 détaille toutes les instructions MEMFILE.

La plupart des instructions MEMFILE affecte le résultat de la fonction MEMBASIC STATUS. En général, STATUS rendra 0 si l'opération s'est bien passée, un code d'erreur sinon.

Les exemples qui sont donnés dans cette section comportent parfois des portions de programmes MEMBASIC. Les instructions MEMBASIC qui sont alors utilisées ne seront pas détaillées. Se reporter à la section II "MEMBASIC" pour une meilleure compréhension des exemples.

## 1.2 OPERATIONS GLOBALES

Voici les différentes opérations dites globales sur un fichier :

- **Création :** Définit la structure d'une fiche et prépare le fichier sur le répertoire sélectionné du disque. Le fichier ne contient alors aucune fiche.

- Destruction : Le fichier est effacé du catalogue du répertoire et les fiches qu'il pouvait contenir sont perdues.

- Ouverture : Avant d'utiliser un fichier, il est indispensable de le déclarer au système. Cette opération s'appelle "ouverture du fichier".

  MEMFILE renseigne alors le programme sur la structure d'une fiche et se prépare à effectuer des opérations de lecture ou d'écriture sur celui-ci.

- Fermeture : Les opérations d'écriture ou de mise à jour d'un fichier sont 'tamponnées'. Autrement dit, les données à écrire sur un fichier transitent par une zone de mémoire intermédiaire qui n'est recopiée sur le disque que lorsqu'elle est saturée. Il se produit donc assez souvent un décalage dans le temps entre la demande d'écriture et sa réalisation effective. En fin d'utilisation d'un fichier, afin d'éviter de perdre les écritures 'en cours', il y a lieu de demander la "fermeture du fichier".

  A noter que, lorsqu'un programme MEMBASIC termine son exécution, tous les fichiers ouverts sont automatiquement fermés. (Voir END)

## 1.3 LE DICTIONNAIRE

Le dictionnaire sert à décrire la structure d'une fiche pour un fichier déterminé. Il est défini au moment de la création du fichier et est conservé de manière permanente dans le fichier afin d'être relu au moment des ouvertures.

C'est lui qui assure le lien entre les données sauvegardées sur disque et celles existantes en mémoire centrale. En effet, il contient les noms des variables MEMBASIC qui recevront les données transférées depuis ou vers le disque.

Il est défini au moment de la création du fichier par l'ordre LET ">... , complété par des ordres LET "+... éventuels.



Il est relu implicitement par MEMBASIC à chaque ouverture de fichier par l'ordre LET "#OPEN...
Il peut alors être visualisé par l'ordre:

```
LET "ENTER,<Identificateur temporaire>"
```

suivi de l'ordre:

```
LET "VISUALISE".
```

### 1.3.1 Les différentes zones d'un dictionnaire

La fiche est constituée de zones de données de deux types :

#### 1.3.1.1 Les zones de type clé :

Elles contiennent des informations qui permettent de retrouver rapidement la fiche dans le fichier. Pour cela, les données associées à ces zones sont rangées de manière ordonnée dans le fichier.

Il importe de bien choisir les zones de ce type afin d'obtenir des moyens d'accès efficaces, mais de ne pas multiplier leur nombre pour de ne pas surcharger inutilement le fichier (ce qui augmente le temps de création des nouveaux articles).

Une fiche peut ne posséder qu'un seul moyen d'accès. On dit alors que le fichier est monoclé. Cette clé de recherche unique peut elle-même être constituée de plusieurs variables.
Par exemple, le moyen d'accès du fichier personnel sera formé des variables NOM$ et PRENOM$. Le dictionnaire débutera donc par "NOM$,PRENOM$ =".
Le classement des fiches du personnel sera réalisé suivant l'ordre alphabétique des noms et, pour des noms identiques, suivant celui des prénoms.

Les types de variables composant une clé sont ceux de MEMBASIC (alphanumérique, flottant, entier). A noter que l'on peut utiliser, dans une clé, des variables alphanumériques sans préciser leurs tailles maximales.
A l'inverse, on peut fixer la taille maximum des variables alphanumériques en faisant suivre leurs noms par leurs longueurs maximales.

```
Par exemple : NOM$30,PRENOM$20
```

Dans ce cas, chaque "NOM$", éventuellement complété par des blancs, occupera 30 caractères dans le fichier et chaque "PRENOM$" en occupera 20.

Le nombre de moyens d'accès à une fiche peut être multiple. On parle alors d'un fichier **multiclés.** Mais, comme dans le cas précédent, chaque moyen d'accès (ou clé) peut être formé de plusieurs variables de différents types.
Dans les ordres opérant sur des fiches existantes, (READ,NEXT,UPDATE,DELETE...), il devient nécessaire de préciser le numéro de moyen d'accès choisi (toujours égal à 1 par défaut).

Un exemple de fichier multiclés peut être un fichier produits dont la clé 1 sera le code du produit CODE% de type entier et la clé 2 représentera le libellé du produit LIBELLE$ sous la forme d'une chaîne de caractères de taille variable. Le dictionnaire débutera alors par :

"CODE% & LIBELLE$ = "

Le symbole "&" sert de séparateur entre deux zones de clés.

Un cas particulier de clé est un simple index croissant de type entier ou flottant. Cet index, qui représente le numéro d'entrée de l'article dans le fichier, est appelé clé relative.

### 1.3.1.2 La zone de type enregistrement :

C'est une zone qui ne sert pas de moyen d'accès, mais qui, en contre-partie, peut contenir de grandes quantités de données. Cette zone se définit sous la forme d'une liste de variables MEMBASIC séparées des zones de clés par le symbole "=".

Les variables de l'enregistrement peuvent être des variables simples de tout type, mais aussi des tableaux. Dans ce cas, il seront sauvegardés sur disque sous forme de matrice creuse pour économiser la place.

ATTENTION : Il ne faut pas répéter les variables de clés dans les variables d'enregistrement.



### 1.3.2 Définition d'un dictionnaire

Le dictionnaire d'un fichier est mémorisé sur disque avec le fichier. Il n'est donc à spécifier qu'une seule fois : lors de la création du fichier. Lors de ses utilisations, le dictionnaire est automatiquement relu et chargé en mémoire au moment de l'ouverture. De même, grâce au dictionnaire,les instructions de lecture et d'écriture n'auront à faire référence à aucune liste de variables.

L'instruction LET ">....." définit un dictionnaire. La chaîne commençant par ">" donne la liste des variables utilisées. Cette instruction devra être exécutée avant la commande de création du fichier.

La syntaxe du dictionnaire est la suivante :

- Dans le cas d'une **clé unique** :

LET ">vc1, vc2,...,vcn = ve1, ve2,..., vep"

vc1,vc2..vcn sont les variables composant la clé unique .
ve1,ve2..vep sont les variables de l'enregistrement

Le caractère "=" sépare la clé de l'enregistrement.

- Dans le cas de **plusieurs clés** :

LET ">vc11,...vc1n & vc21...,vc2p & ... = ve1...vek

vc11,...,vc1n sont les variables de la 1ère clé.
vc21,...,vc2p correspondent à la deuxième clé.
...
ve1 ,...,vek correspondent à l'enregistrement.

Le caractère "&" permet de séparer deux zones de clés.
Les noms de variables représentant des variables tableaux seront suivis de ';'.
Les tableaux ne sont autorisés que dans la partie enregistrement.
Comme dans MEMBASIC, les noms de variables pourront avoir 200 caractères et être écrits indifféremment en minuscules ou en majuscules.

**Exemple de fichier monoclé : Création d'un fichier stock**

Le fichier est un fichier à accès par clé.

La clé est composée de 2 éléments :

famil%  = numéro de famille de la pièce,
npiece% = numéro de pièce.

L'enregistrement comprend les variables suivantes :

pachat   = prix d'achat,
pvente   = prix de vente,
tva%     = taux de TVA,
libelle$ = libellé de l'article,
qstock   = quantité en stock,
qcmd     = quantité en commande,
qmin     = quantité minimum

Pour créer ce fichier, le programme à écrire sera :

```
100    LET "#CLEAR,$"
109 ! Début du dictionnaire
110    LET ">famil%,npiece%=pachat,pvente,"
119 ! suite et fin du dictionnaire
120    LET "+tva%,libelle$,qstock,qcmd,qmin"
129 ! Création du fichier
130    LET "#NEW,F,STOCK"
139 ! Test si opération de création ok
140    IF STATUS THEN PRINT "Fichier non créé"
150    END
```

Pour s'assurer que tout s'est passé normalement, vérifier que STATUS rend 0.

**Exemple de fichier multiclés** : Création d'un fichier Sécurité Sociale :

Clé 1 :

nom$ : le nom, de taille variable.

Clé 2 :

numsecu : le numéro de Sécurité Sociale, de taille fixe (13 chiffres).

L'enregistrement comprend :

adresse$ : l'adresse
datenaiss$ : la date de naissance



Le programme de création sera :

```
100    LET "#CLEAR,$"
109 ! Définition du dictionnaire
110    LET ">nom$&numsecu=adresse$,datnaiss$"
119 ! Création du fichier
120    LET "#NEW,F,SECU"
130    IF STATUS THEN print "Fichier non créé"
140    END
```

Pour définir une clé relative, on prendra comme première clé de fichier (clé index) une variable de type numérique dont la valeur ne pourra qu'être augmentée à chaque nouvelle création. Si la variable est de type entier, elle ne pourra prendre que 32767 valeurs différentes, ce qui limite implicitement le nombre d'articles possibles avec cette méthode. Par contre, si la variable est flottante, la taille du fichier et le nombre d'articles ne sont plus limités que par la taille du disque, comme dans le cas des fichiers ne comportant pas de clé relative.

La syntaxe du dictionnaire dans ce cas particulier devient :

LET ">@vcI & vc21,..., vc2n &... = vel,..., vep

vcI est la clé index.
vc21,...,vc2n composent la clé 2.
...
vel,...,vep composent l'enregistrement.

**Exemple de fichier avec clé relative** : Création d'un fichier des ventes :

La clé relative sera nvente (numéro de la vente).

L'enregistrement pourra comprendre :

- la date vdate$ de la vente
- le tableau des références pièces vendues :
  - famil% : numéro de famille des pièces
  - fourni% : numéro du fournisseur
  - npiece% : numéro des pièces
  - pvente : prix de vente

Le programme de création sera :

```
100    LET "#CLEAR,$"
109 ! Debut de définition du dictionnaire
110    LET ">@nvente=vdate$,famil%;,"
119 ! Suite et fin du dictionnaire
120    LET "+fourni%;,npiece%;,pvente;"
129 ! Création du fichier
130    LET "#NEW,F,VENTES"
140    IF STATUS THEN PRINT "Erreur !"
150    END
```

## 1.4 ACTIONS AU NIVEAU DE LA FICHE

### 1.4.1 Opérations d'écriture

L'écriture d'un enregistrement ne doit se faire que lorsque toutes les variables clé et enregistrement sont prêtes.
Deux modes sont possibles suivant que l'on autorise ou pas les clés homonymes (ordres ADD ou WRITE).
Dans le cas où les homonymes sont permis, on peut préciser si l'insertion de la fiche s'effectue en tête (option /FIRST) ou en queue (option /LAST) de la liste d'homonymes correspondante.

Contrairement à l'écriture, la mise à jour (ordre UPDATE) d'une fiche ne modifie que la partie enregistrement de celle-ci.

Pour réaliser une mise à jour, il suffit donc de préciser l'une des clés et l'enregistrement.
Si l'on désire mettre à jour à la fois les zones de clés et l'enregistrement, il est nécessaire de détruire la fiche et de la recréer.

### 1.4.2 Opérations de lecture

MEMFILE permet deux modes de lecture :

- La lecture d'une fiche (ordre READ), suite à une recherche sur une des clés.
  Dans ce cas et s'il existe des homonymes, on ne peut atteindre que la première fiche de la liste des homonymes (option /FIRST), ou bien la dernière (option /LAST)



- La lecture séquentielle (ordre NEXT), ou séquentielle inverse (ordre NEXT option /PREVIOUS).

Pour chacune des clés d'un fichier, MEMFILE gère un curseur courant qui :

- est placé au début du fichier à l'ouverture,
- est mis à jour à chaque opération d'écriture de type WRITE ou de lecture de type READ,
- est avancé d'une clé par suite d'un ordre NEXT exécuté normalement,
- est reculé d'une clé par suite d'un ordre NEXT/P.

Le curseur associé à chaque clé peut être replacé en tête (ordre ZERO/FIRST) ou en fin de fichier (ordre ZERO/LAST).
A noter que les lectures séquentielles peuvent être bornées inférieurement et supérieurement grâce à des repères placés par l'ordre BORNE.

## 1.5 SYNTAXE

Toutes les instructions MEMFILE ont la forme suivante:

LET <expression chaîne>

Le mot-clé étant toujours LET, c'est le contenu de l'expression chaîne qui déterminera l'instruction à exécuter. L'expression chaîne peut être aussi bien une constante chaîne qu'une expression évaluée, ce qui donne une assez grande souplesse de programmation.

Cependant, dans la plupart de nos exemples nous prendrons la forme LET "..." qui est la forme la plus simple, mais ceci ne veut pas dire que les formes plus compliquées (LET suivi d'une expression chaîne évaluée) ne sont pas possibles.

La chaîne de caractères contient des mots-clés comme "READ" ou "UPDATE" ... Ces mots-clés, sont dans certains cas, précédés de "#" comme dans "#OPEN" ou "#DELETE".

Pour ces mots-clés, la première lettre seule est significative (ou première lettre après "#") et elle peut être écrite indifféremment en majuscule ou en minuscule.

Par exemple :

"#N" , "#n", "#new" et "#NEW"
sont équivalents.

de même pour :

"Next" , "N" , "n" et "NEXT"

Si l'instruction comporte plusieurs mots, ils pourront être séparés par "-", ":" ou ",".

Si l'expression comprend le nom d'un fichier, celui-ci devra respecter la syntaxe imposée par le système d'exploitation.

L'erreur No 27001 est détectée si l'expression chaîne qui suit LET n'est pas reconnue valide.

Le chapitre 2 détaille, pour chaque forme de l'instruction LET, sa syntaxe et son fonctionnement.

## 1.6 IDENTIFICATEUR TEMPORAIRE

Les fichiers MEMFILE sont des entités indépendantes du programme MEMBASIC qui les utilise. Ils sont créés et sauvés sur disque et pourront être utilisés dans différents programmes.

Lorsque qu'un programme MEMBASIC désire utiliser un fichier MEMFILE, il doit tout d'abord l'OUVRIR. Cela signifie que les paramètres du fichier sont chargés dans la mémoire de l'ordinateur. Le fichier MEMFILE est alors disponible. Il peut être utilisé pour des lectures, création, mise à jour.

Pour éviter de rappeler le nom du fichier à chaque utilisation, on associe à chaque fichier lors de son ouverture un IDENTIFICATEUR TEMPORAIRE.

L'identificateur temporaire est un caractère du jeu ISO à l'exception de $ et des caractères de code inférieurs ou égal à 32. Les lettres minuscules ou majuscules représentent les mêmes identificateurs temporaires.

| | |
|---|---|
| 1 | est un identificateur temporaire valide, |
| A ou a | aussi. |



Par exemple : on ouvre le fichier "FIC1" et on lui attribue l'identificateur temporaire "f". Pour désigner ce fichier par la suite, seul "f" (ou "F") sera nécessaire. LET "NEXT,F" par exemple, lit l'article suivant dans le fichier.

# CHAPITRE 2
# INSTRUCTIONS MEMFILE

Le Chapitre 2 décrit l'ensemble des instructions disponibles avec MEMFILE.

L'ordre utilisé est l'ordre alphabétique car il permet de retrouver très rapidement une information cherchée.

Pour chaque instruction le même plan est utilisé. Ce plan est donné page suivante.

Chaque instruction nouvelle commence à une nouvelle page pour faciliter vos recherches.

Pour les instructions qui ne sont pas de la forme LET "....", consulter la section II "MEMBASIC".

Si une instruction recherchée ne se trouve pas dans cette section, consulter également la section III "MEMSCREEN".

# LET "......" INSTRUCTION

## BUT

Il s'agit d'une brève description du but de l'instruction.

## SYNTAXE

Ce paragraphe décrit la syntaxe de l'instruction de façon exhaustive. La syntaxe est décrite selon la méthode employée dans la section II "MEMBASIC".

## EXPLICATION

Ce paragraphe décrit en détail le fonctionnement de cette instruction.

## EXEMPLE

Ce paragraphe contient une portion de programme utilisant l'item faisant l'objet du paragraphe et montrant donc sa mise en oeuvre.

## ERREURS

Il s'agit d'un petit paragraphe donnant la liste des erreurs qui peuvent être détectées par l'interpréteur MEMBASIC lors de l'exécution du programme.

## STATUS

Paragraphe non-systématique donnant l'effet de l'instruction sur la fonction STATUS.

## REMARQUE

Paragraphe non-systématique permettant des rapprochements entre des commandes, fonctions ou instructions différentes.



# LET "#CLEAR..." INSTRUCTION

## BUT

Cette instruction ferme un fichier.

## SYNTAXE

LET "#CLEAR,<identificateur temporaire>"

## EXPLICATION

Le fichier dont on a indiqué l'identificateur temporaire est fermé.

Un fichier est fermé également par :

- LET "#CLEAR,$"
- LET "#CLEAR,$<disque>"
- LET "#CLEAR,$$"
- END
- RUN "..."

La non fermeture d'un fichier peut le détériorer (Voir le chapitre 1).

En fait, les fichiers MEMFILE étant supportés par des fichiers du système d'exploitation de l'ordinateur, ce sont les contraintes de ce dernier qui apparaissent.

## EXEMPLE

LET "#CLEAR,A" ferme le fichier d'identificateur temporaire A.

## STATUS

Rend 1 si l'on tente de fermer un fichier qui n'est pas ouvert.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

L'instruction LET "#CLEAR..." est également utilisée pour supprimer de la mémoire des masques (voir la section III "MEMSCREEN").



# LET "#CLEAR,$" INSTRUCTION

## BUT

Ferme tous les fichiers ouverts.
Ferme également les masques MEMSCREEN ouverts.

## SYNTAXE

LET "#CLEAR,$"

## EXPLICATION

Tous les fichiers ouverts par LET "#OPEN..." sont fermés.
Cette instruction efface également de la mémoire les masques ouverts (voir la section III "MEMSCREEN") et le tampon des dictionnaires.
C'est la seule différence entre LET "#CLEAR,$" et LET "#CLEAR,$$"

La non fermeture d'un fichier peut le détériorer (Voir le chapitre 1).

En fait, les fichiers MEMFILE étant supportés par des fichiers du système d'exploitation de l'ordinateur, ce sont les contraintes de ce dernier qui apparaissent.

## STATUS

Non significatif.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

L'instruction LET "#CLEAR,$" assure que l'on peut changer de disquette sur toutes les unités.
(Voir aussi à ce sujet LET "#CLEAR,$<disque>" ou LET "#CLEAR,$$").

# LET "#CLEAR,$$" INSTRUCTION

## BUT

Ferme tous les fichiers ouverts.

## SYNTAXE

LET "#CLEAR,$$"

## EXPLICATION

Tous les fichiers ouverts par LET "#OPEN..." sont fermés.
Cette instruction n'efface pas de la mémoire les masques ouverts (voir MEMSCREEN), ni le tampon des dictionnaires.
C'est la seule différence entre LET "#CLEAR,$" et LET "#CLEAR,$$"

La non fermeture d'un fichier peut le détériorer (Voir le chapitre 1).

En fait, les fichiers MEMFILE étant supportés par des fichiers du système d'exploitation de l'ordinateur, ce sont les contraintes de ce dernier qui apparaissent.

## STATUS

Non significatif.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

L'instruction LET "#CLEAR,$$" assure que l'on peut changer de disquette sur toutes les unités.
(Voir à ce sujet LET "#CLEAR,$<disque>" ou LET "#CLEAR,$").



# LET "#CLEAR,$<d>" INSTRUCTION

## BUT

Ferme tous les fichiers ouverts sur une unité de disque.

## SYNTAXE

LET "#CLEAR,$<disque>"

<disque> aura la forme :

A: B: C: ...

## EXPLICATION

Tous les fichiers qui ont été ouverts sur le disque spécifié sont fermés.

La non fermeture d'un fichier peut le détériorer (Voir le chapitre 1).

En fait, les fichiers MEMFILE étant supportés par des fichiers du système d'exploitation de l'ordinateur, ce sont les contraintes de ce dernier qui apparaissent.

## STATUS

Non significatif.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

L'instruction LET "#CLEAR,$<disque>" assure que l'on peut changer la disquette (ou disque amovible) sur l'unité <disque>.
(Voir aussi à ce sujet LET "#C,$" et LET "#C,$$").

# LET "#DELETE,F..." INSTRUCTION

## BUT

Cette instruction détruit un fichier MEMFILE sur disque.

## SYNTAXE

LET "#DELETE,F,<nom>"

Le nom devra respecter la syntaxe des noms de fichiers MEMSOFT (se reporter au chapitre 6 de la section 1). Il pourra contenir des spécifications de disques ou de répertoires. Par contre, le suffixe est imposé par MEMFILE et ne doit donc pas être spécifié.

## EXPLICATION

L'instruction LET "#DELETE,F,<nom>" détruit le fichier sur le disque. Il est nécessaire pour cela que le fichier n'ait pas été ouvert au préalable.

Chaque fichier MEMFILE est composé de deux fichiers simples sur le disque, de suffixes .MFK et .MFR. Les deux fichiers : <nom>.MFK et <nom>.MFR sont détruits.

## EXEMPLE

LET "#DELETE,F,/SUB/FIC1"

détruit le fichier FIC1 du sous-répertoire SUB du répertoire principal sur le disque implicite.

## STATUS

Si le fichier désigné n'existe pas, la fonction STATUS rendra 10, sinon 0.

## ERREURS

L'erreur No 27530 est détectée si le fichier est ouvert lors de la destruction.



Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "#GARBAGE..." INSTRUCTION

## BUT

Compactage d'un fichier MEMFILE.

## SYNTAXE

LET "#GARBAGE,F,<nom> [> <chemin>]"

Le nom devra respecter la syntaxe des noms de fichiers MEMSOFT (se reporter au chapitre 6 de la section 1). Il pourra contenir des spécifications de disques ou de répertoires. Par contre, le suffixe est imposé par MEMFILE et ne doit pas être spécifié.

## EXPLICATION

De nombreuses opérations de destruction ou de mise à jour d'un fichier MEMFILE peuvent créer des pertes de place à l'intérieur du fichier.
Dans ce cas, afin de réduire au minimum la taille de ce fichier sur le disque, il est utile d'effectuer l'opération de nettoyage.

L'instruction LET "#GARBAGE..." se déroule en trois phases consécutives :

1 - Création d'un fichier temporaire dont le nom est obtenu à partir du nom original en remplaçant la dernière lettre par le caractère $.

2 - Remplissage du fichier temporaire à partir des informations lues dans l'original. Durant cette étape, une jauge apparaît en bas de l'écran qui matérialise le déroulement du processus.

3 - Recopie du fichier temporaire sur l'original.

Si l'on désire que le fichier temporaire soit créé sur un autre disque ou simplement dans un répertoire particulier, il suffit de le préciser en indiquant le chemin en fin d'instruction derrière le symbole >.



Cette mention est facultative et, si elle n'est pas mentionnée, c'est le chemin par défaut qui est choisi par le système.

## EXEMPLE

LET "#GARBAGE,F,VENTES"

compacte le fichier VENTES dans le répertoire courant.

LET "#GARBAGE,F,ECRITURES>B:/POUBELLE"

compacte le fichier ECRITURES en créant un fichier temporaire dans le répertoire /POUBELLE du disque B:.

## STATUS

Rend 0 si l'opération s'est effectuée normalement.
Rend 10 si le fichier n'existe pas.

## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

Eviter d'attribuer à un fichier MEMFILE un nom se terminant par le caractère $, car cette instruction le détruirait.

# LET "#NEW..." INSTRUCTION

## BUT

Création d'un fichier MEMFILE.

## SYNTAXE

LET "#NEW,F,<nom>"

Le nom devra respecter la syntaxe des noms de fichiers MEMSOFT (se reporter au chapitre 6 de la section 1). Il pourra contenir des spécifications de disques ou de répertoires. Par contre, le suffixe est imposé par MEMFILE et ne doit pas être spécifié.

## EXPLICATION

La création d'un fichier nécessite que le dictionnaire du fichier ait été défini au préalable.

Le dictionnaire du fichier précise :

- le nombre de moyens d'accès au fichier,
- le type de chaque moyen d'accès (relatif, clé fixe, clé variable...),
- la description des différents champs et leur nom.

L'instruction LET ">..." permet de définir ce dictionnaire.

Le chapitre 1 explique la syntaxe des dictionnaires.

## EXEMPLE

```
LET ">a=b,c"
LET "#NEW,F,TEST"
```

créent le fichier TEST avec une clé numérique (a) et un enregistrement de deux champs (b et c).

## STATUS

Rend 0 si la création s'est effectuée normalement.
Rend 30 si le fichier décrit existe déjà.



## ERREURS

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "#OPEN..." INSTRUCTION

## BUT

Ouverture d'un fichier MEMFILE.

## SYNTAXE

LET "#OPEN,<IT>,F,<nom>"

<IT> identificateur temporaire qui est un caractère du jeu ISO à l'exception des caractères $ et ^ qui sont réservés et des caractères de code inférieur ou égal à 32.
L'identificateur temporaire choisi ne doit pas correspondre à celui d'un autre fichier ou d'un masque (voir MEMSCREEN).

Le nom devra respecter la syntaxe des noms de fichiers MEMSOFT (se reporter au chapitre 6 de la section 1). Il pourra contenir des spécifications de disque ou de répertoires. Par contre, le suffixe est imposé par MEMFILE et ne doit pas être spécifié.

## EXPLICATION

Le fichier est ouvert et pourra être utilisé aussi bien en lecture, écriture et mise à jour. L'identificateur temporaire défini ici sera utilisé dans les instructions ultérieures.

Ne pas oublier de fermer le fichier en fin de traitement (voir LET "#CLEAR...") s'il a subi des modifications.

## EXEMPLE

LET "#OPEN,1,F,TESTF"

ouvre le fichier MEMFILE TESTF et lui attribue l'identificateur temporaire 1.

## ERREURS

L'erreur No 27007 est détectée si l'identificateur choisi est déjà attribué.



L'erreur No 27013 est détectée si trop de fichiers MEMFILE sont ouverts en même temps. Le nombre maximum de fichiers est limité à 20.

L'erreur No 27512 est détectée si un fichier est impossible à ouvrir.

L'erreur No 27587 est détectée si le dictionnaire contient une variable de type incorrect, par exemple un tableau qui a le nom d'une variable simple dans le programme utilisateur.

L'erreur No 27589 est détectée si le dictionnaire contient une variable de type incorrect pour la clé relative.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

A chaque fichier MEMFILE correspondent deux fichiers sur le disque avec les suffixes :
- .MFK pour les clés,
- .MFR pour les enregistrements.

# LET "+..." INSTRUCTION

## BUT

Complète une définition de dictionnaire.

## SYNTAXE

LET "+<constante chaîne>"

## EXPLICATION

La chaîne qui suit le signe "+" est ajoutée à celle déjà contenue dans le tampon.

## EXEMPLE

```
LET "a="
LET "+b,c"
LET "VISUALISE"
a=b,c
LET "+,d"
LET "VISUALISE"
a=b,c,d
```

## ERREUR

Néant.

## REMARQUE

Voir au chapitre 1 les explications sur le contenu du tampon.



# LET ">..." INSTRUCTION

## BUT

Met une chaîne dans le tampon des dictionnaires.

## SYNTAXE

LET ">><Constante chaîne>"

## EXPLICATION

Le tampon des dictionnaires est une zone mémoire qui sert lors de la création de fichiers.
On y entre le descriptif d'un enregistrement qui sera utilisé lors de la création (instruction LET "#NEW,F...").
Voir au chapitre 1 les explications sur ce qu'il faut mettre dans le tampon.

Deux formes sont possibles :

```
LET ">"       vide le tampon,
LET ">..."    vide le tampon et y met la chaîne qui
              suit le signe ">".
```

## ERREUR

Aucune erreur n'est détectée.

Si le contenu du tampon n'est pas un descriptif de dictionnaires valides, une erreur sera détectée lors de la création du fichier par l'instruction LET "#NEW,F,...".

## REMARQUE

Si l'on veut compléter le tampon, utiliser l'instruction LET "+...".

# LET "ADD..." INSTRUCTION

## BUT

Ecriture d'un nouvel article dans un fichier MEMFILE avec possibilité d'homonymie.

## SYNTAXE

LET "ADD [<option>],<identificateur temporaire>"

L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

<option>, qui est expliquée plus loin, peut être /F ou /L.
L'option implicite est /F.

## EXPLICATION

L'article est créé avec les valeurs des variables dans MEMBASIC.
Toutes les clés doivent être spécifiées.

S'il existe déjà des articles de même clé (homonymes), l'option /F ou /L permet de décider si le nouvel article sera inséré au début ou à la fin de la liste.
L'option implicite ou /F : l'article créé est inséré au début de la liste des homonymes.
L'option /L : l'article créé est inséré en fin de liste.

Dans le cas d'un fichier relatif, le numéro donné par l'utilisateur n'est pris en compte que s'il est supérieur aux numéros déjà utilisés. Sinon, l'article sera créé avec le premier numéro libre.
Cela permet de créer des trous dans la numérotation, en particulier de copier un fichier dans lequel on a détruit des articles. Si on désire une incrémentation automatique du numéro, il faudra toujours, par sécurité, positionner le pointeur à 0 avant l'écriture.
L'homomymie sur une clé relative n'est pas acceptée et provoque toujours l'attribution d'un nouveau numéro.



## STATUS

Non significatif.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "BORNE..." INSTRUCTION

## BUT

Permet de fixer une borne supérieure ou inférieure aux lectures séquentielles d'un fichier MEMFILE suivant l'une des clés.

## SYNTAXE

LET "BORNE [<option>],<IT> [,<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

<option> peut être /F ou /L ou /D.
L'option implicite est /L.

Le paramètre optionnel <clé> indique sur quelle clé doit s'effectuer le bornage.
Le numéro de clé implicite est 1.

## EXPLICATION

L'instruction de lecture séquentielle LET "NEXT..." déclenchera une erreur "fin de fichier" si l'une des bornes est atteinte :

- Borne supérieure dans le cas d'une lecture par LET "NEXT...",
- Borne inférieure dans le cas d'une lecture par LET "NEXT/P...".

La borne supérieure est choisie par l'instruction LET "BORNE...".
Elle prend la valeur des variables au moment de l'exécution de l'instruction.

La borne inférieure est choisie par l'instruction LET "BORNE/F...".

Dans le cas d'un fichier multiclés, on peut fixer une borne par clé.



L'instruction LET "NEXT..." ne déclenchera de fin de fichier que sur la borne fixée sur la clé correspondante.

Par exemple :

Avec un fichier de clé A :

```
A = 100
LET "BORNE,F"
A = 0
LET "READ,F"
DO
LET "NEXT,F"
LOOP UNTIL STATUS
```

lit toutes les clés de 0 jusqu'à 100 inclus dans l'ordre croissant.

et

```
A = 100
LET "BORNE/F,F"
A = 500
LET "READ,F"
DO
LET "NEXT/P,F"
LOOP UNTIL STATUS
```

lit toutes les clés de 500 jusqu'à 100 inclus dans l'ordre décroissant.

La borne sert également à l'instruction LET "ZERO..." de "rembobinage".

LET "ZERO..." ou LET "ZERO/F..." ramène à la borne fixée par LET "BORNE/F..." (ou en début de fichier).

LET "ZERO/L..." ramène à la borne fixée par LET "BORNE/L..." (ou en fin de fichier).

On peut retirer les bornes par l'option /D.

LET "BORNE/F/D,..." retire la borne inférieure sur la clé spécifiée.

LET "BORNE/L/D,..." retire la borne supérieure sur la clé spécifiée.

Par exemple :

LET "BORNE/F/D,F,2" retire la borne inférieure de la clé 2 du fichier ouvert avec l'identificateur temporaire F.

## STATUS

Non significatif.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

Aucune instruction, en dehors de LET "NEXT..." et LET "ZERO..." n'est influencée par LET "BORNE...".

Par exemple, dans un fichier d'identificateur temporaire F dont la clé est la variable A :

```
A = 100
LET "BORNE,F"
```

L'ordre LET "NEXT..." ne permettra pas de lire des éléments de clé supérieure à 100.

Par contre :

```
A = 200
LET "READ,F"
```

fonctionnera comme dans les autres cas.



# LET "DELETE..." INSTRUCTION

## BUT

Détruit un enregistrement d'un fichier MEMFILE.

## SYNTAXE

LET "DELETE [<option>],<IT> [<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

<option> peut être /F ou /L.
L'option implicite est /F.

Le numéro de clé varie de 1 au nombre de clés.
La valeur implicite est 1.

## EXPLICATION

La clé est recherchée suivant le numéro de clé spécifié.
Si la clé n'est pas trouvée, la fonction STATUS rend 10. Sinon, la clé et l'article correspondant sont supprimés du fichier.

En cas d'homonymie :

- si la dernière opération de lecture précédant LET "DELETE..." a lu la clé cherchée par la destruction, c'est cette clé qui sera détruite.

- sinon une recherche est effectuée et en cas d'homonymie, l'option /F ou /L décidera à quel bout commence la recherche.
  /F en début de la liste d'homonymes, /L en fin de la liste d'homonymes.

## STATUS

Rend 10 si la clé recherchée n'existe pas.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".



# LET "ENTER..." INSTRUCTION

## BUT

Relecture du dictionnaire d'un fichier.

## SYNTAXE

LET "ENTER,<identificateur temporaire>"

L'identificateur temporaire doit correspondre à celui qui a été attribué au fichier lors de l'ouverture.

## EXPLICATION

Le dictionnaire est reconstitué d'après les informations codées.
Ce dictionnaire peut n'être pas totalement indentique avec celui indiqué à la création du fichier (voir LET ">...") mais ce n'est qu'une question de présentation due au fait qu'il est reconstitué.

Ce dictionnaire relu dans le tampon peut :

- être visualisé par : LET "VISUALISE"
- être utilisé pour une nouvelle création de fichier comme s'il avait été mis dans le buffer par LET ">...".

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "NEXT..." INSTRUCTION

## BUT

Lecture séquentielle d'un article dans un fichier MEMFILE dans le sens croissant ou décroissant.

## SYNTAXE

LET"NEXT [<option>],<IT> [,<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

Le paramètre optionnel <clé> permet de spécifier la clé qui doit servir pour déterminer la séquence. Le numéro de clé varie de 1 au nombre de clés. Sa valeur implicite est 1.

<option>, qui est expliquée plus loin, peut être /K, /R, /A ou /P.

## EXPLICATION

L'article est recherché suivant la clé spécifiée (ou implicitement, suivant la première clé). La recherche se fait suivant la dernière lecture et non suivant la valeur actuelle des variables clés dans MEMBASIC.

Par exemple, si la clé est A :

```
A = 5
LET "READ,1"
A = 3
LET "NEXT,1"
```

Cette dernière instruction lira la clé suivante de 5 et non celle de 3.

Attention : LET "NEXT..." tient compte des bornes placées par LET "BORNE".



## OPTIONS

1. Sens.

L'option /P permet d'inverser le sens de lecture.
Le sens de lecture implicite est croissant.
Avec l'option /P, la lecture est décroissante.

2. Enregistrement.

Si la clé existe, il est possible de lire ou de ne pas lire l'article.

/K lit la clé seule (KEY)
/R lit l'enregistrement seul (RECORD)
/A lit toutes les clés dans le cas d'un fichier multiclés (ALL)

L'option implicite est /A/R : lecture de toutes les clés et de l'enregistrement.

## EXEMPLE

LET "NEXT,1" lit la clé et l'enregistrement suivant du fichier d'identificateur temporaire 1.

LET "NEXT/P,2" lit la clé et l'enregistrement précédent du fichier d'identificateur temporaire 2.

LET "NEXT/K,3" lit la clé suivante du fichier d'identificateur temporaire 3.

## STATUS

Rend 255 si l'on atteint la fin du fichier.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".



# LET "READ..." INSTRUCTION

## BUT

Lecture d'un article dans un fichier MEMFILE suivant une clé.

## SYNTAXE

LET "READ [<option>],<IT> [,<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

Le paramètre optionnel <clé> permet de spécifier la clé qui doit servir à la recherche.
Le numéro de clé varie de 1 au nombre de clés. Sa valeur implicite est 1.

<option>, qui est expliquée plus loin, peut être /K, /R, /A, /F ou /L.

## EXPLICATION

L'article est recherché suivant la clé spécifiée (ou implicitement, suivant la première clé).
La clé de recherche sera constituée de la concaténation des valeurs des variables définies pour cette clé dans le dictionnaire du fichier (voir chapitre 1).
Si la clé est relative, la recherche se fera par numéro, sinon elle se fera grâce à une recherche en arbre (B tree).
Si la clé cherchée n'existe pas, la fonction STATUS rendra 10, sinon elle rendra 0.

## OPTIONS

1. Homonymes.

L'option /F ou /L permet de décider, si la clé recherchée existe en plusieurs exemplaires dans le fichier, si l'on doit lire la première ou la dernière de la liste.
Cela dépend en général de l'opération suivante qui peut être une lecture séquentielle croissante ou décroissante.

Implicite ou /F lit le premier homonyme (FIRST)

/L lit le dernier (LAST)

2. Enregistrement.

Si la clé existe, il est possible de lire ou de ne pas lire l'article.

/K lit la clé seule (KEY)
/A lit toutes les clés dans le cas de fichier multiclés (ALL)
/R lit l'enregistrement seul (RECORD)

L'option implicite est /A/R.

Note : Si le fichier est monoclé, /K/R, /A/R et /R sont équivalents puisqu'il n'y a pas d'autre clé à lire que celle qui a été fournie.

## EXEMPLE

LET "READ,1" lit clé et enregistrement du fichier d'identificateur temporaire 1.

LET "READ/L,2" lit clé et enregistrement du fichier d'identificateur temporaire 2. En cas d'homonyme, c'est le dernier entré qui est lu en premier.

LET "READ/K,3" lit les clés du fichier d'identificateur temporaire 3 (si le fichier est monoclé teste seulement l'existence de l'article).

## STATUS

Rend 10 si la clé demandée n'existe pas.



## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "UPDATE..." INSTRUCTION

## BUT

Met à jour un enregistrement d'un fichier MEMFILE.

## SYNTAXE

LET "UPDATE [<option>],<IT> [,<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

Le paramètre optionnel <clé> permet de spécifier la clé qui doit servir à la recherche.
Le numéro de clé varie de 1 au nombre de clés. Sa valeur implicite est 1.

<option>, qui est expliquée plus loin, peut être /F ou /L.

## EXPLICATION

Si une clé est spécifiée, la mise à jour se fera suivant cette clé, sinon la clé implicite choisie sera 1.
Les valeurs des variables de clés doivent être prêtes dans MEMBASIC ainsi que les nouvelles valeurs des variables de l'enregistrement.
L'article, s'il est trouvé, est mis à jour sur le disque.S'il n'est pas trouvé, la fonction STATUS rendra 10 et aucune mise à jour ne sera effectuée.

S'il existe plusieurs articles ayant la clé indiquée pour la mise à jour (homonyme), deux cas sont possibles :

- La dernière opération effectuée sur le fichier avant le LET "UPDATE..." a permis la lecture de l'un de ces articles.
  L'instruction LET "UPDATE..." agira alors sur cet article qui vient d'être lu.

- La dernière opération effectuée n'a pas lu un article correspondant à cette clé.
  Une recherche de la clé indiquée sera alors effectuée avant la mise à jour, et l'article



mis à jour sera soit le premier avec l'option /F, soit le dernier avec l'option /L de la liste de ces homonymes.
L'option implicite est /F.

## STATUS

Rend 0 si la mise à jour s'est bien passée, 10 si l'article à mettre à jour n'existe pas.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

## REMARQUE

L'instruction LET "UPDATE..." peut s'utiliser aussi bien sur une clé alphanumérique que sur une clé relative. Le fonctionnement est exactement le même dans les deux cas.

# LET "VISUALISE" INSTRUCTION

## BUT

Visualise le tampon contenant les dictionnaires de fichiers.

## SYNTAXE

LET "VISUALISE"

## EXPLICATION

Le dictionnaire d'un fichier se trouve dans le tampon après l'une des instructions LET ">...", LET "+..." ou LET "ENTER...".
L'instruction LET "VISUALISE" imprime sur le périphérique de sortie en cours (déterminé par l'instruction PRINTER de MEMBASIC), le contenu de ce tampon.

## ERREUR

Néant.

## REMARQUE

Le tampon est effacé par l'une des instructions LET "#CLEAR,$" ou LET ">".



# LET "WRITE..." INSTRUCTION

## BUT

Ecriture d'un nouvel article dans un fichier MEMFILE sans possibilité d'homonynie.

## SYNTAXE

LET "WRITE,<identificateur temporaire>"

L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

## EXPLICATION

L'article est créé avec les valeurs des variables dans MEMBASIC.
Toutes les clés doivent être spécifiées. Si l'une des clés existe déjà, l'écriture sera refusée et le STATUS prendra la valeur 30.
Dans le cas d'un fichier relatif, le numéro donné par l'utilisateur n'est pris en compte que s'il est supérieur aux numéros déjà utilisés. Sinon, l'article sera créé avec le premier numéro libre. Cela permet de créer des trous dans la numérotation en particulier de copier un fichier dans lequel on a détruit des articles. Si on désire une incrémentation automatique du numéro, il faudra toujours positionner le pointeur à 0 avant l'écriture.

## STATUS

Rend 0 si l'écriture s'est bien passée, 30 si l'article existe déjà.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".

# LET "XINDEX..." INSTRUCTION

## BUT

Donne la valeur de la prochaine clé relative qui sera créée par défaut dans un fichier relatif.

## SYNTAXE

LET "XINDEX,<IT>"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

## EXPLICATION

Si la valeur de la clé relative n'est pas spécifiée lors d'une écriture dans un fichier relatif, l'article sera créé avec le premier numéro libre.

L'instruction XINDEX permet de connaitre cette valeur. Elle sera lue dans la variable BASIC définissant cette clé.

## ERREUR

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.



# LET "ZERO..." INSTRUCTION

## BUT

Positionnement sur la borne inférieure ou supérieure d'un fichier MEMFILE.

## SYNTAXE

LET "ZERO [<option>],<IT> [,<clé>]"

<IT> est un identificateur temporaire désignant un fichier.
L'identificateur temporaire doit être celui qui a été attribué au fichier lors de l'ouverture.

Le paramètre optionnel <clé> permet de spécifier la clé qui doit servir pour déterminer la séquence.

<option> peut être /F ou /L.
L'option implicite est /F.

## EXPLICATION

Cette instruction permet de se repositionner sur une borne supérieure ou inférieure d'un fichier pour une clé donnée.
La prochaine instruction LET "NEXT..." ou LET "NEXT/P" tiendra compte de ce positionnement.

L'option /F ou /L détermine le sens du positionnement.

L'option implicite ou /F : positionnement sur la borne inférieure du fichier.
La borne est celle qui a été définie par l'instruction LET "BORNE/F...". Si aucune borne inférieure n'a été définie, on sera positionné en début de fichier.

L'option /L : positionnement sur la borne supérieure du fichier.
La borne est celle qui a été définie par l'instruction LET "BORNE/L...". Si aucune borne supérieure n'a été définie, on sera positionné en fin de fichier.

## STATUS

Non significatif.

## ERREURS

L'erreur No 27003 est détectée si l'identificateur temporaire choisi ne correspond à aucun fichier ouvert ou correspond à un masque MEMSCREEN.

L'erreur No 27581 est détectée si un numéro de clé est invalide.

Des erreurs non récupérables sont possibles en cas de problème de support ou de mémoire. Consulter la liste des erreurs en Annexe B de la section II "MEMBASIC".



# ANNEXE
# LES OPTIONS

## L'OPTION /K/A/R

Cette option est utilisée dans les instructions LET "READ..." et LET "NEXT...".

L'option /K permet de ne lire que la clé spécifiée.

L'option /A permet de lire toutes les clés.

L'option /R permet de ne lire que l'enregistrement.

L'option implicite est /A/R.

## L'OPTION /F/L

Cette option peut avoir différents sens selon l'instruction dans laquelle elle est utilisée.

1. Dans les instructions :

```
LET "READ..."
LET "UPDATE..."
LET "ADD..."
LET "DELETE..."
```

Elle permet de dire comment doit se faire la recherche ou l'insertion en cas d'homonymie.

/F la recherche ou l'insertion se fait en début de liste d'homonymes.

/L la recherche ou l'insertion se fait en fin de liste d'homonymes.

Noter que dans le cas des mises à jour (LET "UPDATE..."), l'option n'est pas utilisée si la lecture précédant le LET "UPDATE..." a permis de désigner l'homonyme concerné.

L'option implicite est /F dans tous ces cas.

2. Dans l'instruction :

LET "BORNE..."

/F permet de désigner la borne inférieure.

/L permet de désigner la borne supérieure.

L'option implicite est /L.

3. Dans l'instruction :

LET "ZERO..."

/F positionnement sur la borne inférieure du fichier ou en début de fichier s'il n'y a pas de borne.

/L positionnement sur la borne supérieure du fichier ou en fin de fichier s'il n'y a pas de borne.

L'option implicite est /F.



# INDEX THEMATIQUE
# LISTE DES INSTRUCTIONS

Voici les différentes instructions MEMFILE.
Chaque instruction est suivie d'un commentaire sur son utilisation. Pour en savoir plus, se reporter au chapitre 2 où toutes les instructions sont décrites en détail, classées par ordre alphanumérique.

# INDEX GENERAL MEMSOFT

Cet index donne par ordre alphanumérique un certain nombre de moyens d'accès à la documentation. La notation utilisée pour décrire l'endroit où est située l'information est la suivante :

- La section, donnée par son nom :

  Intro (section I - installation, généralités)
  Basic (section II - MEMBASIC)
  Screen (section III - MEMSCREEN)
  File (section IV - MEMFILE)

- Position dans la section :

  Il s'agit de :

  - soit la référence du chapitre et éventuellement du sous-chapitre,
  - soit "index" pour référencer l'index thématique de chaque section,
  - soit [ ... ] pour indiquer le nom de la fonction à consulter dans le chapitre 2 de la section.

  Par exemple :

  * Basic [ GOTO ] signifie :

    Section II - MEMBASIC, chapitre 2,
    Voir l'instruction GOTO

  * Basic 1.1 signifie :

    Section II - MEMBASIC, chapitre 1.1