MEM/DOS

SYSTÈME D'EXPLOITATION

UN PRODUIT : **MEMSOFT**

3, rue Meyerbeer

06000 NICE - FRANCE

MEMSOFT S.A. au capital de 1.750.000 F

R.C. NICE B 321 250 490 - APE 7703

NICE - PARIS - LOS ANGELES

AVERTISSEMENT

La présente brochure contient en 3 ème partie un complément au manuel MEM/DOS pour l'APPLE //c.

Cette 3 ème partie ne décrit que les différences apparaissant pour l'utilisateur entre la version Apple //e et la version Apple //c.

Le lecteur qui possède un Apple //c devra donc lire d'abord dans la partie Apple //c le chapitre concernant la prise en main du système, et effectuer le démarrage sur son Apple //c, puis lire et exécuter les exemples du manuel d'initiation à MEM/DOS.

Lorsqu'il sera familiarisé avec le système, le lecteur pourra alors se référer au manuel d'utilisation sur l'Apple //e, et se reporter pour certains points précis à la 3 ème partie lorsque des différences avec l'Apple //c seront signalées dans la marge.

MANUEL D'INITIATION

MEMS FT

PROGICIELS DEMAIN C'EST AUJOURD'HUI

MEM/DOS®

UN PRODUIT : **MEMSOFT**

3, RUE MEYERBEER
06000 NICE - FRANCE

MEMSOFT SA NICE – PARIS – LOS ANGELES

SOMMAIRE

CHAPITRE II

UTILISATION DES FICHIERS

CHAPITRE III

UTILISATION DES MASQUES

CHAPITRE IV

LES SOUS-PROGRAMMES

CHAPITRE V

Introduction

QUELS SONT LES OBJECTIFS ET LES AVANTAGES DE MEM/DOS ?

MEM/DOS est un PRECIEUX OUTIL DE DEVELOPPEMENT pour les programmeurs réalisant des logiciels d'application de gestion en basic. Il peut :

. gérer facilement des fichiers à accès par clé,
. saisir des données par masques d'écran,
. structurer des programmes avec des sous-programmes,
. manipuler sans erreur d'arrondi des nombres de 12, 20, 48 chiffres,
. imprimer des états par masques d'impression,
. travailler rapidement grâce à la simplicité de la syntaxe et l'aide d'utilitaires de développement nombreux et puissants.

MEM/DOS est aussi un puissant SYSTEME D'EXPLOITATION de gestion capable de gérer des disques et des disquettes (Disk Operating System). Les programmeurs système, réalisant des configurations micro-informatique, peuvent :

. accéder rapidement et facilement aux informations du disque,
. bénéficier d'une gestion dynamique de disquettes ou disques durs pouvant aller de 140 K à 120 Mégaoctets,
. modifier et adapter, grâce à des utilitaires puissants, le système lui-même en fonction de son environnement et de leurs propres besoins.

MEM/DOS est pour l'entreprise qui s'équipe d'un système micro-informatique l'assurance d'un LOGICIEL DE QUALITE : les fonctionnalités du système d'exploitation, apportent à l'utilisateur confort, rapidité et sécurité.

QUELS SONT LES BUTS DE CETTE NOTICE ?

On ne trouvera dans cette notice que les rudiments nécessaires à l'utilisation de MEM/DOS pour les petites applications de gestion. Sa lecture suppose une connaissance de BASIC et quelques rudiments du DOS.
Après avoir consacré quelques heures à la lecture de cette notice, il sera possible de passer au manuel d'utilisation.

Le premier chapitre permet de prendre contact avec MEM/DOS et de gérer des programmes. Le deuxième est consacré aux fichiers, le troisième aux masques et le quatrième aux sous-programmes. Enfin le chapitre V illustre par un exemple un peu plus important l'utilisation conjointe des fichiers, des masques et des sous-programmes.

Pour des raisons matérielles, les programmes illustrant cette notice sont rassemblés à la fin des chapitres où ils sont présentés ; ces programmes, qui sont numérotés de P0 à P12 et de SP0 à SP5, se trouvent sur la disquette. Il est préférable de les étudier et de les utiliser dans l'ordre.

Chapitre I

PRISE EN MAIN DU MEM/DOS

MANIPULATIONS DE PROGRAMMES BASIC

Dans ce chapitre, vous verrez d'abord comment mettre en route MEM/DOS, ce qui est vraiment facile ; vous ferez ensuite une espèce de survol des notions nécessaires à son utilisation, aussi bien du point de vue de la syntaxe des commandes que de la définition des "objets" manipulés : fichiers et masques. Enfin, vous apprendrez à gérer sous MEM/DOS les programmes écrits en BASIC : vous constaterez qu'il n'y a pas une grande différence avec les commandes habituelles du DOS 3.3 mais qu'il y a déjà quelques possibilités supplémentaires.

1. DEMARRAGE DE MEM/DOS

1.1 Implantation physique

Pour utiliser MEM/DOS, il nous faut :

- un apple II 48 K ou un Apple II e

- un lecteur de disquette avec contrôleur et le DOS 3.3

- une carte MEM/DOS

La carte MEM/DOS peut être fixée sur n'importe quel support ("slot") de votre Apple. Cependant, en cas d'utilisation d'une carte Z 80, la carte MEM/DOS doit être placée sur un support de numéro supérieur à celui de la carte Z 80.
Enfin, on suppose que votre contrôleur de disque est sur le support numéro 6.

1.2 Ce qu'il faut faire une fois pour toutes

La disquette fournie par MEMSOFT ne contient pas le DOS 3.3. Il faut donc, et ceci, une fois pour toutes, "installer" le DOS 3.3 sur votre disquette MEM/DOS. Comme toujours, il vous est fortement conseillé d'effectuer une copie de la disquette MEM/DOS avant toute manipulation. Voici donc la séquence d'opérations à effectuer :

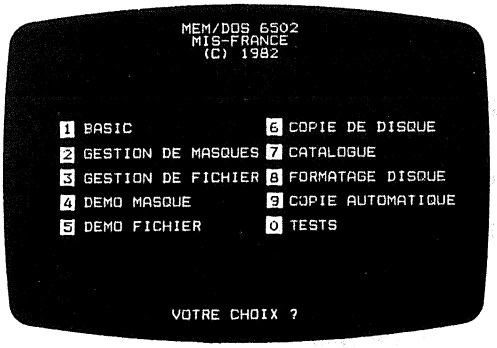
- Faire une copie de la disquette MEM/DOS.
Cette disquette n'est absolument pas protégée et n'importe quel utilitaire de copie conviendra (COPYA du DOS 3.3, par exemple).

- Votre Apple étant sous DOS 3.3, mettre la disquette MEM/DOS et taper

RUN HELLO

Ce programme HELLO permet également de transcoder d'anciens fichiers DOS 3.3 en vue de leur utilisation sous MEM/DOS, mais ce n'est pas ce qui nous intéresse en ce moment. C'est pourquoi on choisit l'option 1 proposée par ce programme.

- Vous assistez alors à un défilement de plusieurs écrans (dont un écran en 80 colonnes, si vous avez une carte 80 colonnes sur votre Apple). Le menu de démarrage ci-dessous s'affiche enfin :



Vous êtes maintenant en possession d'une disquette MEM/DOS qui démarrera toute seule ; d'où...

1.3 Ce qu'il suffit de faire les autres fois

Le démarrage normal du MEM/DOS sur Apple s'effectuera désormais de la façon suivante : introduire la disquette MEM/DOS (qui contient maintenant le DOS 3.3) dans le tourne-disquette et mettre votre Apple sous tension. Le système se charge lui-même et le message suivant apparaît à l'écran quelques instants :

Un écran de 80 colonnes apparaît éventuellement (si vous avez une carte 80 colonnes, bien sûr), puis le menu de démarrage s'affiche, et MEM/DOS est à votre disposition.

1.4 Le menu de démarrage

Les dix fonctions proposées par ce menu sont détaillées en Annexe III mais leur compréhension ne semblera aisée au débutant qu'après avoir pris connaissance du contenu de cette brochure.
Pour l'instant, nous allons utiliser le Basic et il faut donc taper 1. Sur l'écran qui s'affiche, le curseur est au début du mot RUN écrit sur la dernière ligne. Il vous suffit de taper RETURN pour faire disparaître ce RUN et passer à la ligne suivante. Vous êtes maintenant en Basic mais... sous MEM/DOS : vous pouvez constater que vous n'êtes plus sous DOS 3.3 : par exemple CATALOG ne marche pas !!!

Remarque : Si, par erreur, vous tapez RETURN au lieu d'un chifre, le menu de démarrage, vous pouvez revenir à ce menu en tapant : RUN puis RETURN.

2 - LES DIFFERENTS OBJETS QUE L'ON TROUVE SUR UNE DISQUETTE

Le travail d'un système d'exploitation est de gérer une disquette, c'est-à-dire de permettre à l'utilisateur d'y écrire et d'y lire des informations. Avant de voir quelles sont les commandes qui permettent de gérer effectivement ces informations, commençons par voir quels sont les " objets " (objet = groupement d'informations) que l'on peut gérer.

2 - 1 Avant toute chose : la numérotation des lecteurs

Dans le DOS normal, les lecteurs sont repérés par S6,D1, puis S6,D2 ... De même, dans le système PASCAL, ces lecteurs sont repérés par # 4: puis # 5: ... Ils s'appellent A: et B: sous CP/M ! On voit donc que chaque système baptise à sa guise les lecteurs. Dans le cas du MEM/DOS :

le lecteur connecté en S6,D1 porte le numéro 0
le lecteur connecté en S6,D2 porte le numéro 1
le lecteur connecté en S5,D1 porte le numéro 2

2 - 2 Le catalogue d'une disquette

La commande LET"*" a pour effet d'afficher le catalogue du lecteur courant ; on entend par lecteur courant celui que l'on a utilisé en dernier : par exemple, après avoir démarré le système, le lecteur courant est évidemment le lecteur numéro 0 (S6, D1), mais après avoir exécuté une commande telle que :

LET"*,1"

qui agit sur le lecteur 1, le lecteur courant est maintenant le lecteur 1 et la commande

LET"*"

agira sur le lecteur 1 (ce n'est pas la seule commande à agir sur le lecteur courant).

Bien entendu, avec MEM/DOS qui ne dispose que du mini-lecteur numéroté 0, inutile de préciser que le lecteur courant est toujours 0 et qu'on ne doit pas le modifier : si vous tapez LET"*,1" vous ferez apparaître un message d'erreur, ce qui est logique puisqu'il n'y a pas de périphérique correspondant, mais cela va encore plus loin : pour remettre les choses en ordre, il faut taper LET"*,0" car le lecteur courant était devenu le lecteur 1 !!!

2 - 3 Les différents types d'objets

Revenons au catalogue : nous voyons que les identificateurs sont précédés par une des lettres B, F, G, M ou P qui signifient respectivement que l'objet en question est :

B : un Bloc de valeurs Binaires recopiées telles quelles,

F : un Fichier de données,

P : un Programme écrit en BASIC

M : un Masque (pas d'affolement, nous verrons bientôt ce que c'est)

G : un Groupe de masques (appelé parfois masque Global.

2 - 4 Le nom des objets

Des objets peuvent avoir le même nom s'ils sont de type différent. Le nom d'un objet est toujours composé de 20 caractères ; si l'utilisateur en donne moins MEM/DOS complète par des blancs. On peut utiliser tous les caractères et les blancs sont pris en compte. Donc :

SALAIRES BRUTS

est un nom d'objet différent de

SALAIRES BRUTS

2 - 5 Le catalogue partiel

L'utilisateur perspicace aura constaté que le catalogue présente d'abord les objets du type B, puis ceux du type F, etc ... et que les objets du même type sont classés par ordre lexicographique (ordre alphabétique étendu à tous les caractères : FACT 1 sera avant FACT 2). On peut restreindre l'affichage du catalogue aux objets d'un seul type.

```
LET"*.0,PROGRAM"
```

par exemple, si on veut uniquement la liste des programmes du lecteur 0.

3 - PRINCIPES D'UTILISATION DES FICHIERS ET DES MASQUES

Le but de ce paragraphe est simplement d'exposer à ceux des lecteurs qui sont "débutants" des définitions bien connues des informaticiens.

3 - 1 Qu'est-ce qu'un fichier de données ?

Signalons d'abord que "données" est ici une traduction de l'anglais DATA ... qui signifie "information". Donc, fichier de données doit se comprendre fichier d'informations : que ces informations soient données ou calculées ne change rien ! Il s'agit d'une de ces ambiguités génantes mais bien ancrées dans le vocabulaire des informaticiens.

Après cette précision, voyons le terme "fichier" : avant l'avènement de l'informatique une fiche était (d'après le Robert) "un carton sur lequel on inscrit des renseignements en vue d'un classement" et le mot fichier avait le sens courant d'ensemble de fiches ou de boîte contenant un ensemble de fiches. En remplaçant "carton" par "emplacement sur un support magnétique" on a la notion de fiche et de fichier informatique (bizarrement les informaticiens utilisent le mot enregistrement de préférence au mot fiche).

A l'origine les fichiers étaient sur bande magnétique et, les fiches étant l'une derrière l'autre, on était obligé de les parcourir toutes, séquentiellement, si on en cherchait une en particulier. Actuellement, les fichiers sont sur disquettes et on peut accéder plus facilement à n'importe quel enregistrement puisque la disquette est "découpée" en morceaux appelés "secteurs" que l'on peut atteindre directement. Il n'empêche qu'on utilise encore des mots comme "séquentiel", "accès direct", "relatif" pour exprimer des modes de parcours d'un fichier ; on verra en détail dans le chapitre II consacré aux fichiers que MEM/DOS est particulièrement agréable et efficace dans ce domaine.

3 - 2 Qu'est-ce qu'un masque ?

La saisie des données sous mode "conversationnel" classique s'effectue sous forme d'un dialogue entre le programme qui affiche sur l'écran un message précisant ce qu'il attend, et l'utilisateur qui tape en retour au clavier la valeur attendue. Cela va bien tant qu'il y a peu de données à saisir, mais lorsqu'il s'agit de saisir les données relatives à plusieurs enregistrements d'un fichier, ce processus peut paraître désagréable à l'usager et on a imaginé de faire demander en bloc l'ensemble des différentes variables d'un enregistrement (par exemple : nom, prénom, numéro de sécurité sociale, date de naissance) grâce à une technique nouvelle : le masque.

Un ensemble de messages et de "fenêtres" est affiché en bloc sur l'écran : dans chaque fenêtre l'utilisateur entre la valeur de la variable demandée et, lorsque toutes les fenêtres sont remplies, une commande permet d'envoyer l'ensemble des valeurs au programme. Le chapitre III donne tous les détails sur l'utilisation de masques en MEM/DOS.

3 - 3 La notion d'identificateur temporaire

Tout fichier ou masque est répertorié dans le catalogue d'une disquette grâce à son nom (ou identificateur). Mais lorsqu'on veut utiliser dans un programme un de ces objets, on le fait par l'intermédiaire d'un identificateur temporaire (appelé classiquement "numéro logique"): dans un même programme, chaque objet utilisé se voit d'abord attribuer un identificateur temporaire différent qui, par la suite, sera utilisé à la place de son nom.

Cet identificateur temporaire est en fait formé d'un seul caractère... mais quelconque : chiffre, lettre, signe (à l'exception de $). La commande

```
LET"#OPEN:B,FICHIER:ADRESSE"
```

attribue l'identificateur temporaire B au fichier ADRESSE et, dans la suite de ce programme, la commande

```
LET"READ:B"
```

déclenchera une lecture sur le fichier ADRESSE. Ne vous préoccupez pas pour l'instant du détail de ces commandes, nous y reviendrons plus tard.

4 - GESTION DES PROGRAMMES BASIC

Avant d'entamer les chapitres suivants consacrés aux masques et aux fichiers, voyons d'abord comment MEM/DOS gère les programmes écrits en BASIC : c'est voisin de ce que vous avez l'habitude de faire. Il y a simplement une syntaxe légèrement différente et quelques possibilités supplémentaires.

4 - 1 Les messages d'erreurs

A partir de maintenant, vous allez probablement multiplier les petits essais au clavier... ce qui entrainera quelques erreurs. Alors, autant consacrer un paragraphe à ce sujet avant de poursuivre... Il existe deux classes d'erreurs : celles qui n'interrompent pas le déroulement du programme, et que l'on qualifie de "récupérables" et les autres, qui sont baptisées "irrécupérables ou fatales", en ce sens qu'elles interrompent le déroulement normal du programme.

Les erreurs irrécupérables sont des erreurs de programmation (donc détectées à l'interprétation ou à l'exécution), ou des erreurs "systèmes" : disque plein ou protégé en écriture par exemple. Les erreurs récupérables n'interrompent pas le déroulement du programme ; le programmeur peut constater comment la commande s'est exécutée : la variable WS est réservée par MEM / DOS pour y placer un "compte-rendu de transaction" : si une commande s'est bien exécutée, MEM/DOS place la valeur 0 dans cette variable ; en cas de difficulté, MEM/DOS y place une valeur positive. Un bon programmeur fera donc suivre toute commande MEM/DOS d'un test lui permettant de vérifier si WS est devenu positif et prendra des mesures en conséquence.

Remarques :

R.1 : La liste des messages d'erreurs et des valeurs correspondantes de WS pour les erreurs récupérables est donnée en annexe I.

R.2 : Lorsqu'une commande MEM/DOS est exécutée en mode direct (ou interactif : c'est-à-dire lorsque, hors de tout programme, une commande est entrée au clavier) et que l'exécution de cette commande provoque une erreur, le message DIRECT ERROR apparait. Ce message est parfois mal interprété : il ne veut pas dire que la commande en question est interdite en mode direct, il signifie simplement que l'exécution de cette commande n'a pu être menée à bien.

R.3 : Ce message est de plus suivi d'un "message explicite" qui évite à l'utilisateur de faire imprimer WS pour voir ce qui n'a pas bien marché.

R.4 : N'oubliez pas que WS est un identificateur de variable réservé au système. Ne l'utilisez donc surtout pas dans vos programmes.

4 - 2 Sauvegarde d'un programme.

Après avoir tapé un petit programme du genre

```
NEW
10 PRINT " OK "
20 END
```

Il est facile de le sauvegarder sur la disquette courante sous un nom quelconque, SE par exemple, grâce à la commande :

SAVE"SE"

Si on veut préciser le numéro du lecteur, c'est possible :

SAVE"0:SE"

On peut alors constater par la commande LET "*" que SE apparait dans la liste des programmes du catalogue.

4 - 3 Effacement

L'opération qui permet de supprimer un objet de la disquette est déclenchée par la commande DELETE qui veut que l'on précise le type de l'objet à supprimer, puisque des objets de type différent peuvent avoir le même nom :

LET"#DELETE,PROGRAM:SE" ou LET" # D,P : SE"

aura pour effet d'effacer le programme SE du disque courant. Le mot PROGRAM peut être résumé à son initiale P, et DELETE à D.

4 - 4 Sauvegarde après effacement

La commande SAVE "SE" ne marche pas s'il y a déjà un programme appelé SE sur la disquette : cela permet aux étourdis de ne pas effacer par mégarde un programme dont ils avaient oublié le nom ! Mais il existe une variante de la commande SAVE qui permet d'effacer volontairement une version précédente :

SAVE" @ SE" ou SAVE" @ 0:SE" (notez bien la position du @)

4 - 5 Chargement d'un programme

La commande LOAD "SE" a pour effet d'effacer le programme précédemment en mémoire centrale et d'y placer le programme SE.

4 - 6 Chargement en queue

La commande LOAD"#SE" a pour effet de placer le texte du programme SE à la suite du texte résidant en mémoire centrale : c'est très utile pour mettre bout à bout des morceaux de programmes ... à condition d'avoir prévu la numérotation adéquate car ce chargement bout à bout s'effectue sans s'occuper des numéros de lignes : il n'y a pas fusion des deux programmes !!! Par curiosité, vous pouvez exécuter la séquence

```
NEW
LOAD "SE"
LOAD "#SE"
LIST
```

Si vous êtes vraiment curieux, enlevez le premier END et exécutez !!!

4 - 7 Exécution d'un programme

La commande RUN lance l'exécution de ce qui se trouve en mémoire centrale. La commande RUN "TOTO" cherche le programme TOTO sur la mini-disquette, le charge en mémoire centrale et l'exécute. Par contre, si vous essayez d'exécuter un masque ou un fichier, cela ne marchera pas ; on verra aux chapitres II et III comment on se sert de ces objets.

4 - 8 Exécution avec conservation de variables pré-valuées

Lorsqu'on lance l'exécution d'un programme par un RUN, toutes les variables sont remises à zéro. Par contre, la commande

```
LOAD"/ESSAI"
```

a pour effet de conserver les valeurs des variables actuellement en mémoire centrale et de lancer l'exécution avec ces valeurs après avoir chargé ESSAI.

La séquence

```
NEW
10 PRINT "OK"
20 PRINT M
30 END
SAVE "ESSAI"
M = 10
LOAD "/ESSAI"
```

affiche OK puis 10 sur l'écran.

REMARQUE : Il s'agit bien du mot LOAD et pas du mot RUN. Attention ceci ne marche que si ESSAI est d'une taille inférieure au dernier programme chargé normalement.

5 - LE POINT SUR LA FORME DES COMMANDES MEM/DOS.

Il y a deux types de commandes sous MEM/DOS. D'une part celles qui agissent sur un programme (SAVE, LOAD, RUN) que nous venons d'utiliser, et d'autre part... toutes les autres : nous avons utilisé LET"*", nous avons vu sans les détailler LET"READ,B" et LET"#OPEN:B,FICHIER,ADRESSE" et nous en verrons beaucoup d'autres bientôt.

Vous avez pu constater que la syntaxe des premières est très voisine de ce que vous avez l'habitude d'utiliser : la différence essentielle est la présence de guillemets ; toutes les autres utilisent le mot LET. C'est "le" mot réservé du MEM/DOS, en ce sens qu'il intervient souvent pour donner une commande au système ; il n'a plus du tout la signification du BASIC normal où il indiquait, de manière facultative et peu employée, l'instruction d'affectation (LET N = N + 1 par exemple). LET est suivi d'une expression chaîne précisant ce qu'il faut faire ; cette chaîne peut contenir des constantes et des variables ; par exemple, si K est la variable (numérique) dont la valeur est le numéro du lecteur, on aurait pu écrire :

```
LET"*,"+STR$(K)
```

REMARQUES :

R.1 : Le caractère , est un séparateur. Il en existe trois autres, qui sont - ; : tous les quatre sont interchangeables entre eux : dans une commande MEM/DOS vous pouvez utiliser à votre guise l'un ou l'autre.

- : , ; séparateurs interchangeables dans une commande MEM/DOS.

R.2 : Les blancs n'ont pas de signification : ils sont ignorés par le système, sauf dans le nom d'un objet comme on l'a vu plus haut.

R.3 : Les mots réservés READ, OPEN, etc... peuvent être abrégés à leur première lettre : de toutes facons, MEM/DOS n'analyse que la première lettre de chaque mot réservé. Cette remarque ne s'applique pas aux identificateurs de commande SAVE, LOAD, RUN, LET qui eux sont analysés par BASIC. Autrement dit, tout ce qui suit est équivalent :

```
LET"# OPEN-1,FICHIER:FSE"
LET"# O-1,F:FSE"
LET"# OUVRIRLE-1,FICHIER:FSE"
```

R.4 : Les commandes SAVE, LOAD, RUN sont aussi suivies d'une chaîne de caractères : c'est ce qui explique la présence des guillemets ; cela autorise l'usage de variables dans les commandes.

6 - UN PETIT PROBLEME NON RESOLU

Tout ordre de MEM/DOS suivant le THEN doit être précédé d'un : sous peine d'une erreur fatale se traduisant par un message SYNTAX ERROR. Il faut donc écrire par exemple :

```
IF A = B THEN : LET "...
```

C'est très irritant quand on tombe dans cette chausse-trappe ... alors essayez de retenir cette petite imperfection de MEM/DOS.

7 - UN DILEMME PEDAGOGIQUE

Y a-t-il eu d'abord une poule ou un œuf ? Autrement dit : est-ce l'œuf qui vient de la poule ou la poule de l'œuf ? Le dilemme provient de l'indissociabilité des deux protagonistes.

De la même facon, il est difficile de décider s'il vaut mieux expliquer d'abord l'utilisation des fichiers ou celle des masques puisque malheureusement l'exposé de la première notion se fera sans l'aide de la deuxième. Nous avons choisi d'exposer d'abord l'utilisation des fichiers (chapitre II) ; après la lecture du chapitre III consacré aux masques, vous pourrez facilement reprendre les exemples donnés au chapitre II en vous servant de masques. Vous constaterez alors la pleine puissance de MEM/DOS.

Chapitre II :

UTILISATION DES FICHIERS

Ce chapitre commence par donner quelques précisions sur l'organisation des fichiers et les principes logiques de leur utilisation. Ces précisions ne sont évidemment que des rappels pour certains lecteurs qui pourront donc survoler les paragraphes 1 et 2. La suite du chapitre est consacrée à l'utilisation des fichiers en MEM/DOS et elle s'appuie sur quelques petits programmes rassemblés sous forme de fiches au paragraphe 8 ; les programmes sont également présents sur la mini-disquette MEM/DOS. Vous êtes fortement invité à faire tourner ces programmes le moment venu.

1 L'ORGANISATION DES FICHIERS

Si l'on veut s'y retrouver dans une collection d'objets quelconques, il faut un peu d'organisation. Il en va de même pour les fichiers informatisés, d'autant plus que les "fiches" sont ici illisibles à l'œil nu et qu'en cas de manque d'organisation, l'utilisateur sera fortement pénalisé. Il faut donc parfaitement définir ce que contient une fiche, comment on les classe entre elles et comment on utilise le fichier.

1 - 1 Description du contenu d'une fiche

Une fiche (ou enregistrement) est composée d'une ou plusieurs rubriques (on dit aussi caractéristiques ou champs) qui sont repérées chacune par un identificateur de variables. MEM/DOS autorise les variables classiques du BASIC : entier, flottant, chaîne de caractères, mais aussi deux autres types de variables : les variables représentant une date qui seront de type DATE et les variables représentant un entier compris entre 0 et 255 qui seront de type ENTIER COURT ; l'utilisation des variables de type DATE ou ENTIER COURT est exposée en annexe II : ne mélangeons pas tout.

Dans une fiche, on peut également mémoriser des tableaux de toutes ces variables, mais nous n'en parlons pas dans cette notice d'initiation.

1 - 2 Classement des fiches entre elles : organisation du fichier

Il y a deux méthodes possibles, a priori, pour repérer les fiches entre elles :
- Chaque fois qu'on ajoute une fiche dans le fichier, le système lui donne le numéro N + 1 et c'est ce numéro qui la repère : les fiches sont ordonnées (classées) selon ce numéro auquel il faudra associer une variable pour pouvoir le manipuler. Cette variable est parfois appelée "pointeur du fichier".
- On peut par contre être amené à traiter les fiches dans un ordre lié à la valeur d'une variable : les fiches seront alors repérées grâce à cette rubrique que l'on appellera "clé du fichier" ou "index".

Un fichier sera qualifié de relatif si ses fiches sont rangées dans l'ordre chronologique de création et sera qualifié d'indexé si ses fiches sont rangées selon une clé ; on verra plus loin que la clé peut en fait être constituée de plusieurs rubriques ... et même qu'un fichier peut avoir plusieurs clés différentes.

1 - 3 Rangement logique et rangement physique

On vient de voir que, dans un fichier relatif, l'ordre "logique" de rangement des fiches était l'ordre de création. Sur la disquette, les fiches sont effectivement rangées dans cet ordre. On dit que le rangement physique coïncide avec le rangement logique. Cela peut surprendre, mais dans un fichier indexé, une nouvelle fiche est également placée physiquement en bout de fichier quelle que soit la valeur de la clé !!!

L'explication est que MEM/DOS gère en fait un "fichier d'index" associé à tout fichier indexé : c'est une espèce de tableau ou les fiches (repérées par leur adresse sur le disque) sont rangées selon la valeur de la clé : cette technique permet de n'avoir pas à déplacer physiquement toute une partie du fichier chaque fois qu'il faut insérer une nouvelle fiche, ce qui risquerait d'être long.

Remarques :

R.1 : Deux ou plusieurs fiches peuvent avoir la même valeur de clé ; elles sont alors rangées "logiquement" l'une derrière l'autre.

R.2 : Mais l'utilisateur peut demander à MEM/DOS de ne pas accepter de nouvelle fiche correspondant à une valeur de la clé qui existe déjà dans le fichier : il y a une commande spéciale pour cela (cf paragraphe 5).

R.3 : Il est recommandé de réorganiser de temps en temps le fichier pour que l'ordre logique coïncide avec l'ordre physique, ce qui diminue le temps d'accès : cela se fait grâce à la commande REORGANISE.

R.4 : Rien n'interdit d'associer plusieurs fichiers d'index à un même fichier indexé : c'est ce qui permet d'avoir plusieurs clés pour un même fichier.

R.5 : A l'intention du lecteur soucieux de précision, on peut signaler que MEM/DOS range tous les fichiers en utilisant la technique des B-arbres. Cette technique est utilisée par IBM dans tous ses gros ordinateurs sous le nom de VSAM. Le meilleur article à ce sujet est malheureusement en anglais : "The Ubiquitous B-Tree" par D. COMER dans Computing Surveys Vol. 11, numéro 2, Juin 1979 pages 121 à 137.

1 - 4 Fichier MEM/DOS = Structure + fiches

Dans la plupart des systèmes d'exploitation des micros ordinateurs, un fichier ne contient aucun renseignement concernant sa propre nature ou la structure de ses fiches ; ceci est une source inépuisable d'erreurs. Par contre, un fichier MEM/DOS mémorise deux types de renseignements :

- un ensemble d'informations, parfois appelé "bibliothèque", concernant le fichier lui-même : type du fichier, nom du pointeur ou nom et type des variables clés, description de la fiche : nom et type des variables qui la composent. Cette bibliothèque est accessible à l'utilisateur qui peut la visualiser à tout moment.
- un ensemble de 0 à N fiches, c'est-à-dire de valeurs affectées aux diverses variables de chaque fiche.

Cette organisation permet à MEM/DOS de vérifier que l'utilisation d'un fichier est bien conforme à la structure qui lui a été donnée lors de la création et donc d'éviter certaines erreurs d'inattention.

2 LOGIQUE DE L'UTILISATION DES FICHIERS

2 - 1 Actions élémentaires au niveau de la fiche, du fichier

Une fiche peut être créée, remplie, lue, modifiée ou supprimée. Cela va de soi et ne nécessite pas d'autres explications ; par contre, en ce qui concerne les actions au niveau d'un fichier, on va voir (à l'intention des débutants en informatique) à quoi correspondent les opérations de définition, création, ouverture et fermeture d'un fichier.

2 - 2 Organisation des traitements faisant intervenir des fichiers

Pour pouvoir utiliser un fichier, il y a une chose à faire "une fois pour toutes", c'est de préciser au système :

- la description de la fiche : c'est-à-dire la liste de ses rubriques,
- le type d'organisation du fichier : relatif ou indexé, et le nom de l'index ou du pointeur,
- le nom sous lequel ce fichier sera mémorisé sur le disque.

Ces précisions sont données au système lors de la création du fichier ; en MEM/DOS, la création se fait en deux étapes, donc en deux commandes : l'une de description, l'autre de création proprement dite. La syntaxe de ces commandes est donnée plus loin.

On a déjà vu que dans tout programme utilisant un fichier, il faut lui attribuer un identificateur temporaire dont on se sert ensuite plutôt que d'utiliser le nom sous lequel ce fichier est mémorisé sur disque. Cette précision est donnée lors d'une phase appelée ouverture qui a pour effets secondaires d'abord de préparer le système à travailler sur la première fiche mais aussi de réserver quelques octets en mémoire centrale dans lesquels transiteront les informations : ces octets forment ce qu'on appelle un tampon (ou buffer).

Lorsqu'un programme cesse d'utiliser un fichier, on peut récupérer la place mémoire réservée à ce fichier grâce à une opération appelée classiquement fermeture du fichier, par opposition à l'ouverture. Précisons à ce sujet qu'avec MEM/DOS toute opération d'écriture dans un fichier sur disque est exécutée sur le champ, contrairement à ce qui se passe dans la plupart des systèmes où la recopie du buffer est différée aussi longtemps que possible. L'opération de "fermeture" correspond donc à une simple récupération de la place du buffer, c'est pourquoi la commande correspondante est appelée CLEAR plutôt que CLOSE. Le gros avantage qu'apporte MEM / DOS avec cette technique de recopie immédiate, c'est une sécurité accrue : on ne peut pas perdre une écriture disque (ou plusieurs) comme cela arrive avec les autres systèmes en cas de "pépin" ; avec MEM/DOS, un fichier est modifié dès qu'une instruction d'écriture est exécutée ... ce qui est normal après tout !

Remarques :

R.1 : La création d'un fichier entraîne implicitement son ouverture. Donc si, dans un même programme, vous créez un fichier et l'utilisez immédiatement, il ne faut pas essayer de l'ouvrir après l'avoir créé, cela entraînerait une erreur.

R.2 : Les identificateurs temporaires d'un fichier donné ne sont pas forcément les mêmes dans les différents programmes qui l'utilisent.

3 LA SYNTAXE DES COMMANDES

Une commande fichier devra permettre à MEM/DOS de savoir sur quel fichier il doit travailler : grâce à l'identificateur temporaire ; puis de déterminer s'il s'agit d'une commande au niveau du fichier ou d'une fiche : le caractère # indiquera une action au niveau fichier ; enfin, certaines commandes étant identiques pour les fichiers, les programmes et les masques, il faudra préciser le type d'objet en question.

```
LET "# OPEN-1,FICHIER:FSE "
                                    FSE : nom du fichier sur
                                      le disque courant
                          FICHIER : action sur un fichier
                    1 : identificateur temporaire
          OPEN : ouverture d'un objet
          # : action au niveau fichier
```

Rappels :

```
LET"#O:1;F-FSE"
```

aura le même effet puisque seule la première lettre des mots réservés est analysée et que les séparateurs : ; , - sont équivalents.

```
50 PRINT "DONNEZ UN NOM DE FICHIER"
60 INPUT NF$
70 LET "# O:1 ; F-"+NF$
```

aura le même effet sur le fichier dont le nom sera entré au clavier : on ouvrira avec l'identificateur temporaire 1 dans le programme le fichier répertorié sur le disque courant sous le nom lu au clavier.

Si vous essayez maintenant (ou plus tard bien sur !) cette commande en mode direct et que le fichier FSE n'existe pas sur le disque courant, vous aurez un message DIRECT ERROR. Rappelons que cela ne veut pas dire que cette commande est interdite en mode direct.

4 LA COMMANDE CLEAR

Nous consacrons un paragraphe spécial à la commande CLEAR car sa syntaxe est simple, mais son utilisation est un peu difficile à expliquer. La commande

```
LET"# CLEAR:3"
```

a pour effet de rompre le lien entre l'identificateur temporaire 3 et le fichier qu'il identifiait jusqu'alors : accessoirement cela libère les quelques octets du buffer en plus de l'identificateur 3 que l'on peut allouer à un autre objet. Par assimilation avec un terme couramment employé en informatique, on appellera FERMETURE du fichier 3 cette opération. La commande

LET"# CLEAR,$"

a pour effet de "fermer" tous les fichiers et aussi tous les masques (une fermeture qui n'est donc qu'une libération d'un identificateur temporaire et de quelques octets a le même effet sur un masque). Cette instruction de fermeture générale est bien agréable car elle évite de répéter autant de CLEAR qu'il y a d'objets à fermer.

Jusque là, normalement, c'est clair. Ce qui pose problème, c'est de savoir où mettre une telle instruction de fermeture générale. A première vue, on est tenté de la mettre en fin de programme ; mais si, pour une raison quelconque, l'exécution du programme ne se termine pas normalement, les identificateurs temporaires ne sont pas libérés et si un autre programme essaie d'en réutiliser un immédiatement, on est certain de voir apparaître un message d'erreur. La seconde possibilité est donc de mettre cette instruction de fermeture générale en tête de chaque programme plutôt qu'en queue : c'est une solution un peu égoiste du genre "après moi le déluge...". Alors la bonne solution, c'est peut-être d'en mettre une en tête par prudence, et une en queue par politesse !

5 UN EXEMPLE SIMPLE

Avant d'entrer dans les détails de l'utilisation des fichiers, regardez (après avoir lu les remarques ci-dessous) ce petit exemple concernant un fichier relatif : dans un premier programme PO on crée un fichier (page 25) et le programme P1 permet de remplir des fiches, les programmes P2, P3 et P4 montrent deux méthodes différentes de lecture dans ce fichier (pages 27, 28, 29).
Avant que vous ne partiez à la découverte de ces programmes (qui sont rassemblés sous forme de fiches (!) à la fin de ce chapitre), il faut rappeler qu'au chapitre III, on verra l'utilisation des masques qui permettent de gérer agréablement les entrées-sorties : les exemples que vous allez voir sont là pour vous permettre de débuter et de bien comprendre le détail de certains mécanismes. Regardez les fiches de ces cinq programmes sans oublier de lire les remarques AVANT de faire exécuter les programmes correspondants, mais sans chercher à comprendre en détail la syntaxe des instructions, cela sera vu dans quelques pages.

6 LES PRINCIPALES COMMANDES

Après l'étude de ces cinq petits programmes nous pouvons maintenant regarder de manière un peu plus précise (mais toutefois pas dans tous les détails) les différentes commandes.

6 - 1 Description de la fiche et du type de fichier

Dans une seule et même commande, on va décrire la composition de la fiche, c'est-à-dire la liste de ses rubriques, et le type de fichier (relatif ou indexé) ainsi que le nom de la (ou des) variable(s) qui commande(nt) l'accès.

Fichier relatif :

```
LET ">  @  TA% = T%,XR$"
```

C'est le symbole > qui indique à MEM/DOS que cette commande lui décrit un fichier. Ensuite, le caractère @ indique qu'il s'agit d'un fichier relatif. Enfin, le caractère = sépare la variable entière qui précise le pointeur ou numéro (TA%), de toutes celles qui précisent les rubriques (T%, XR$).

Fichier indexé par une clé composée d'une seule variable :

Première différence avec ce qui précède : la disparition du symbole @ . La variable clé peut être de type quelconque : entier, réel ou chaîne ; si la variable clé est une chaîne de caractères, il faut préciser la longueur maximale qu'aura cette chaîne : .

```
LET">NM$12=N%,SC,AD$"
```

Notez bien la syntaxe : la longueur n'est pas entre parenthèses. Notez également que si une rubrique est une chaîne, il n'y a pas à préciser la longueur de cette rubrique : seule la longueur de la chaîne clé est à préciser : les enregistrements ont une longueur variable, ce qui économise de la place sur le disque.

Fichier indexé par une clé composée de plusieurs variables :

```
LET">NM$10,PR$15,AN=N%,SC,AD$"
```

Le caractère = sépare la liste des variables de la clé de la liste des variables de la fiche. Les fiches seront classées selon la première variable de la clé et, en cas d'égalité, selon la deuxième, ... etc : si NM$ est un nom et PR$ un prénom, le fichier sera classé dans l'ordre habituel. En cas d'homonymie parfaite, et si AN est l'année de naissance, l'ainé précèdera le cadet. Rappellons que vous ne devez préciser la longueur d'une variable de la clé que si c'est une chaîne.

Fichier indexé par plusieurs clés :

Les clés sont séparées par le symbole & ; chaque clé peut être composée de plusieurs variables :

```
LET">NM$10,PR$20&SS$13=NM$,PR$,SS$,T%,XR$"
```

On constate que les variables composant les clés sont répétées dans la description de la fiche : en effet, qu'il y ait une ou plusieurs clés, les variables qui composent la ou les clés ne font pas partie de la fiche ; donc, si on accède à une fiche grâce à la valeur d'une clé et si on lit cette fiche, on n'y trouvera la valeur des autres clés de cette fiche... que si on les y a répétées comme rubriques.

Bien entendu, lors de l'utilisation d'un tel fichier, il faudra préciser quelle clé on veut utiliser: si un fichier correspondant à cette description s'est vu attribuer l'identificateur temporaire 4, la commande LET "READ-4,2" déclenchera une lecture par l'intermédiaire de la deuxième clé (SS$).

6 - 2 Commandes au niveau du fichier :

Certaines commandes sont identiques, quelle que soit le type du fichier :

LET "# NEW-3,FICHIER:FSE"

crée et ouvre sous l'identificateur temporaire 3 le fichier qui s'appellera FSE sur la disquette ; la description de ce fichier doit avoir été précisée auparavant.

LET "# OPEN-2,FICHIER:0:FSE"

ouvre le fichier FSE de la disquette 0 sous l'identificateur temporaire 2

LET "# DELETE,FICHIER:FSE"

supprime le fichier FSE de la disquette. Ici, il n'y a pas besoin de passer par un identificateur temporaire. On ne peut pas supprimer un fichier ouvert.

On peut enfin demander à MEM/DOS de nous révéler quelle est la description d'un fichier existant : la séquence

```
10 LET"# OPEN-1,FICHIER:FSE"
20 LET"ENTER-1"
30 LET"VISUALISE"
```

affiche la liste de la ou les clés et des rubriques du fichier FSE. Attention, cette séquence est inutilisable en mode direct.

REMARQUE : Toutes ces commandes ont comme premier caractère qui symbolise une action au niveau d'un "objet" ; on va voir immédiatement que les commandes ayant une action au niveau d'une fiche ne comportent pas ce caractère

6 - 3 Commandes agissant sur les enregistrements d'un fichier relatif :

On a déjà vu certaines de ces commandes :

LET "READ-2"

lit l'enregistrement (du fichier de l'identificateur temporaire 2) dont le numéro a été préalablement précisé.

LET"NEXT-1"

lit l'enregistrement "suivant" : après l'ouverture, c'est le premier enregistrement qui est lu ; s'il n'y a pas d'enregistrement suivant, la variable WS reçoit la valeur 255, alors qu'elle reçoit 0 s'il n'y a pas de problème de lecture.

LET "WRITE-3"

ajoute un enregistrement à la queue du fichier : l'utilisateur n'a pas à se préoccuper de la gestion du numéro de fiche : MEM/DOS s'en occupe.

Il existe d'autres commandes tout aussi utiles :

LET "UPDATE-2"

remplace les valeurs des variables de l'enregistrement sur la disquette par les valeurs qu'ont ces variables en mémoire centrale.

LET "DELETE-4"

efface un enregistrement ; attention : les numéros des fiches restantes ne sont pas modifiés : il se crée un "trou" dans la séquence des numéros des enregistrements et ce trou ne pourra plus être rempli !

6 - 4 Commandes agissant sur les enregistrements d'un fichier indexé :

Les cinq commandes READ, NEXT, WRITE, UPDATE, DELETE agissent également dans le cas d'un fichier indexé avec la même syntaxe, mais il y a quelques différences dans les conditions d'emploi ou les conséquences :

- READ, UPDATE, DELETE exigent que la valeur de la clé soit préalablement fixée,
- DELETE supprime bien un enregistrement, mais la valeur de la clé est réutilisable ultérieurement,
- WRITE suppose que l'utilisateur a précisé les valeurs de toutes les clés s'il y en a plusieurs.

Enfin, il existe une commande ADD qui joue un rôle analogue à celui de WRITE à ceci près que MEM/DOS acceptera les "homonymes" éventuels : on pourra avoir dans le fichier des enregistrements ayant la même valeur de clé : ils seront "logiquement" voisins et en ordre inverse de leur création : le dernier homonyme créé sera le premier lu.

Les programmes P5, P6, P7, P8 et P9 donnent quelques exemples de l'utilisation de fichiers indexés : création, écriture, lecture séquentielle ou à la demande. Ici encore, il ne s'agit que de programmes "pédagogiques". Le programme P8 fait intervenir la commande XTRACT qui permet d'extraire d'un fichier indexé ceux des enregistrements dont la valeur d'une partie de la clé a une certaine valeur. Il existe également une commande BORNE qui fixe une borne supérieure pour la lecture "séquentielle" : cela veut dire qu'il y aura une erreur dès que l'on arrivera à l'enregistrement en question : cela permet de programmer facilement la lecture d'une fraction de fichier. Attention : ces deux commandes sont spécifiques aux fichiers indexés.

Tous ces exemples travaillent sur des fichiers monoclés. En ce qui concerne les fichiers multiclés, il faudrait préciser, pour certaines commandes, le numéro de la clé (par défaut, ce serait la première) :

LET "READ-1,2"

permettrait de lire dans le fichier dont l'identificateur temporaire est 1, en utilisant le tableau d'adresses (ou fichier d'index) relatif à la clé numéro 2.

7 CONCLUSION

Nous n'avons pas fait le tour complet des possibilités de MEM/DOS dans la gestion des fichiers : cette notice est une introduction ... qui doit être poursuivie par la lecture de brochures plus complètes dans lesquelles vous trouverez la totalité des commandes de MEM/DOS, des explications détaillées sur l'organisation et l'implantation réelle des fichiers, ... etc, etc.

8 LES PROGRAMMES P0 à P9

On trouve pour chaque programme :

a) - un énoncé précis de ce que doit faire ce programme, avec les noms des objets manipulés,

b) - un algorithme dans lequel on découpe bien chacune des opérations ; on utilise la structure répétitive :

Itérer

.

.

. traitement 1

.

.

Sortir si condition

.

.

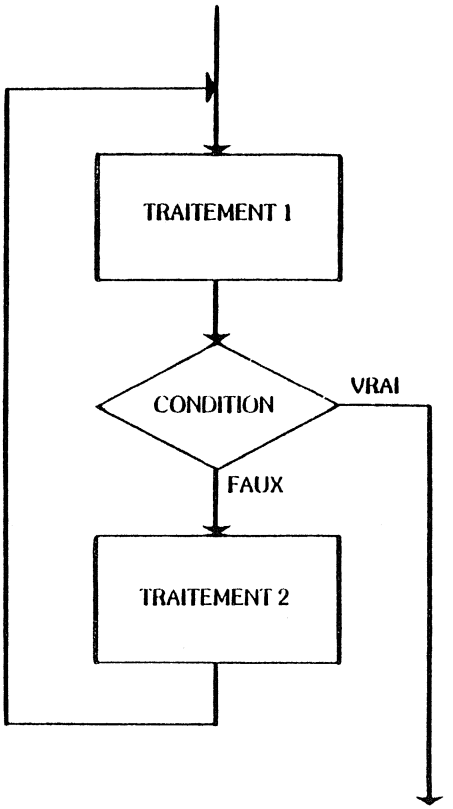
. traitement 2

.

.

Finitérer

qui correspond à l'organigramme ci-contre dans lequel le traitement 1 ou le traitement 2 peut être vide ; dans certains cas, on utilise plusieurs tests de sortie dans la même boucle.



c) - un programme,

d) - des remarques.

P0 : CREATION D'UN FICHIER RELATIF

1) Problème précis :

Créer le fichier relatif FSE dont le numéro de fiche aura comme identificateur NO% et dont chaque fiche comprendra deux rubriques : un nom NM$ et un solde de compte SC. Avant d'essayer de créer FSE, il faudra vérifier qu'il n'existe pas déjà. Ce test s'effectue simplement en essayant d'ouvrir le fichier avant d'essayer de le créer.

2) Algorithme :

. faire le ménage (cf paragraphe 4)
. ouvrir le fichier FSE
. **si** l'ouverture s'est faîte normalement
    alors afficher "le fichier FSE existe déjà"
    sinon décrire la structure d'un enregistrement et le type de FSE
        créer le fichier
finsi
. faire le ménage

3) Programme :

```
10 REM  :  PROGRAMME P0
20 LET "#CLEAR,$"
30 LET "# OPEN:1,FICHIER:0-FSE"
40 IF WS = 0 THEN : PRINT "FSE EXISTE DEJA": GOTO 80
50 LET "> @ NO%=NM$,SC"
60 LET "# NEW:1,FICHIER:0-FSE"
70 IF WS = 0 THEN : PRINT "FSE CREE"
80 LET "# CLEAR,$"
90 END
```

4) Remarques :

1 - Dans ce programme, on a attribué l'identificateur temporaire 1 à FSE et il est créé sur le lecteur 0.

2 - Une erreur de syntaxe dans la ligne 50 peut se traduire par un message d'erreur en ligne 60, car la description de la ligne 50 n'est utilisée que lors de la création.

3 - La commande LET"# DELETE:FICHIER:0-FSE" supprime FSE.

4 - Comprenez par faire le ménage : nettoyer la mémoire !

P1 : AJOUTER UNE OU PLUSIEURS FICHES AU FICHIER FSE

1) Problème précis :

Lire au clavier une suite de NM$ et de SC que l'on place dans le fichier FSE.

2) Algorithme :

. faire le ménage
. ouvrir le fichier FSE
. **si** l'ouverture s'est mal passée **alors** arrêter le programme **finsi**
. **itérer** . lire un nom NM$
. **sortir si** NM$ = "FIN"
    . lire le solde correspondant SC
    . enregistrer NM$ et SC sur le fichier FSE
    . **si** l'opération s'est mal passée **alors** arrêter le programme **finsi**
. **finitérer**
. faire le ménage

3) Programme :

```
10 REM  :  PROGRAMME P1
20 LET "# C,$"
30 LET "# OPEN:1,FICHIER:0-FSE"
40 IF WS THEN : PRINT "FSE N'EXISTE PAS": GOTO 160
50 REM  : ITERER
60 PRINT "DONNEZ UN NOM (FIN=ARRET)"
70 INPUT NM$
80 REM  : SORTIR SI
90 IF NM$ = "FIN" THEN : GOTO 150
100 PRINT "SOLDE CORRESPONDANT SVP ?"
110 INPUT SC
120 LET "WRITE-1"
130 IF WS THEN : PRINT "ERREUR ECRITURE": GOTO 160
140 GOTO 50
150 REM  : FIN ITERER
160 LET "# C,$"
170 END
```

4) Remarques :

1 - Le test IF WS est équivalent à IF WS>0 car en BASIC une expression numérique dont la valeur est nulle est considérée comme fausse.

2 - On n'a jamais eu besoin de préciser le numéro de la fiche NO% : MEM/DOS s'en occupe tout seul. On peut néanmoins faire afficher ce numéro si on veut.

3 - Le programme peut être utilisé plusieurs fois de suite : les fiches sont ajoutées au fur et à mesure au bout du fichier.

P2 : LECTURE DE TOUT LE FICHIER RELATIF

1) Problème précis :

Lire successivement (on dit aussi SEQUENTIELLEMENT) tous les enregistrements du fichier FSE et les afficher sur l'écran au fur et à mesure.

2) Algorithme :

. faire le ménage
. ouvrir le fichier FSE avec l'identificateur temporaire 4 (pourquoi pas !)
. **itérer**
    . lire l'enregistrement suivant du fichier FSE
. **sortir si** il n'y a plus d'enregistrement à lire
    . afficher les caractéristiques de l'enregistrement lu : numéro, nom, solde
. **fin itérer**
. faire le ménage

3) Programme :

```
10  REM  :  PROGRAMME P2
20  LET "# C,$"
30  LET "# OPEN:4,FICHIER:0-FSE"
40  IF WS THEN : PRINT "FSE N'EXISTE PAS": GOTO 150
50  REM  : ITERER
60  LET "NEXT-4"
70  REM  :SORTIR SI
80  IF WS > 0 THEN : PRINT "FIN LECTURE": GOTO 140
90  PRINT "NUMERO D'ENREGISTREMENT : ";NO%
100  PRINT "NOM : ";NM$
110  PRINT "SOLDE DU COMPTE : ";SC
120  PRINT
130 GOTO 50
140 REM  : FIN ITERER
150 LET "# C,$"
160 END
```

4) Remarques :

R.1 :La commande NEXT permet de lire l'enregistrement "suivant" du fichier dont l'identificateur temporaire est précisé ; et la commande OPEN prépare à la lecture du premier enregistrement.

R.2 : On a vu que WS contient un compte-rendu après une opération faisant intervenir la mini-disquette. Après une lecture, si la valeur de ce WS est positive, c'est qu'il n'y avait plus rien à lire.

R.3 : Notez bien que le seul fait d'utiliser le fichier FSE permet au programme de connaître les noms des identificateurs des "rubriques" d'un enregistrement.

R.4 : Le numéro du premier enregistrement est 1 (cela, on le voit en faisant exécuter le programme, pas en regardant le texte ci-dessus).

R.5 : On verra plus loin (P.4) qu'on peut facilement lire séquentiellement une partie du fichier.

P3 LECTURE A LA DEMANDE DE TELLE OU TELLE FICHE

1) Problème précis :

Lire au clavier un numéro d'enregistrement et afficher le contenu de l'enregistrement correspondant du fichier FSE. On s'arrête lorsque le numéro demandé est 0.

2) Algorithme :

. faire le ménage
. ouvrir le fichier FSE
. **itérer**
        lire un numéro d'enregistrement : NO%
. **sortir si** le numéro lu est 0
        lire l'enregistrement correspondant dans le fichier
        afficher les valeurs lues
. **fin itérer**
. faire le ménage

3) Programme :

```
10  REM  : PROGRAMME P3
20  LET "# C,$"
30  LET "# OPEN:4,F:0-FSE"
40  IF WS THEN : PRINT "FSE N'EXISTE PAS": GOTO 180
50  REM  : ITERER
60  PRINT "DONNEZ UN NUMERO (0=FIN)"
70  INPUT N0%
80  REM  : SORTIR SI
90  IF NO% = 0 THEN : GOTO 170
100  LET "READ-4"
110  REM : SORTIR SI
120  IF WS THEN : PRINT "ERREUR LECTURE": GOTO 170
130  PRINT "NOM : ";NM$
140 PRINT "SOLDE DU COMPTE : ";SC
150 PRINT
160 GOTO 50
170 REM  : FIN ITERER
180 LET "# C,$"
190 END
```

4) Remarques :

R.1 : La commande READ permet de lire l'enregistrement dont le numéro est précisé au préalable. C'est ce que certains appellent l'accès "direct".

R.2 : La ligne 120 teste si la lecture s'est bien passée. Avez-vous essayé de lire un enregistrement inexistant ?

P4 LECTURE SEQUENTIELLE D'UNE PARTIE D'UN FICHIER RELATIF

1) Problème précis :

Lire au clavier un numéro d'enregistrement et afficher à l'écran le contenu de cet enregistrement du fichier FSE et de tous les enregistrements suivants.

2) Algorithme :

. ouvrir le fichier FSE
. lire le numéro de départ au clavier
. lire l'enregistrement correspondant sur le fichier FSE
. **itérer**
. **sortir si** on n'a rien lu (fin de fichier)
afficher les valeurs lues
lire l'enregistrement suivant
. **fin itérer**
. fermer le fichier FSE

3) Programme :

```
10 REM  : PROGRAMME P4
20 LET "# C,$"
30 LET "# OPEN:4,F:0-FSE"
40 IF WS THEN : PRINT "FSE N'EXISTE PAS": GOTO 180
50 PRINT "NUMERO DE DEPART"
60 INPUT NO%
70 LET "READ-4"
80 REM  : ITERER
90 REM  : SORTIR SI
100 IF WS THEN : PRINT "FIN LECTURE": GOTO 170
110 PRINT "NUMERO D'ENREGISTREMENT : ";NO%
120 PRINT "NOM : ";NM$
130 PRINT "SOLDE DU COMPTE : ";SC
140 PRINT
150 LET "NEXT-4"
160 GOTO 80
170 REM  :  FIN ITERER
180 LET "# C,$"
190 END
```

4) Remarques :

R.1 : Notez bien la position de l'instruction READ (extérieure à la boucle) qui permet d'initialiser la lecture.

R.2 : Le programme suivant P5 illustre l'utilisation des fichiers indexés : lisez d'abord le paragraphe 6 avant de l'étudier : cela vous permettra de faire le point sur les fichiers relatifs et vous y trouverez également des explications nécessaires à la poursuite de l'étude des exemples.

P5 CREATION D'UN FICHIER INDEXE MONOCLE

(Avez-vous lu le paragraphe 6 ?)

1) Problème précis :

Créer un fichier indexé PERSO dont la clé est composée de deux chaînes de longueur 10 : NM$ et PR$, et dont l'enregistrement comprend deux rubriques : un entier appelé IN% et une chaine appelée AD$.

2) Algorithme :

Sans objet ! On utilise le même principe que dans le programme P0 : on essaie d'ouvrir, ce qui teste l'existence de PERSO !

3) Programme :

```
10 REM  : PROGRAMME P5
20 LET "# C,$"
30 LET "# OPEN:1,FICHIER:0-PERSO"
40 IF WS = 0 THEN : PRINT "PERSO EXISTE DEJA": GOTO 80
50 LET ">NM$10,PR$10=IN%,AD$"
60 LET "# NEW:1,F:PERSO"
70 IF WS = 0 THEN : PRINT "PERSO CREE"
80 LET "# C,$"
90 END
```

4) Remarque :

Un des principes de la méthode ASSIMIL est de s'imprégner de la langue que l'on apprend avant d'essayez de la parler ; autrement dit de pratiquer la version avant le thème. C'est ce que vous avez fait jusqu'ici. Vous pourrez bientôt passer à la seconde phase qui consiste à reprendre ces petits programmes depuis P0, à ne regarder que les algorithmes et à essayer de les programmer par vous-même.

P6 ECRITURE DANS UN FICHIER INDEXE MONOCLE

1) Problème précis :

Enregistrer dans le fichier PERSO une suite de fiches lues au clavier. S'arrêter lorsqu'on lit la valeur "FIN" pour NM$.

2) Algorithme :

. ouvrir le fichier PERSO
. **itérer**
    . lire NM$
. **sortir si** NM$ = "FIN"
    . lire PRS$, IN%, AD$
    . enregistrer la fiche
. **fin itérer**
. fermer le fichier PERSO

3) Programme :

```
10 REM  : PROGRAMME P6
20 LET "# C,$"
30 LET "# OPEN:1,F:0-PERSO"
40 IF WS > 0 THEN : PRINT "PERSO N'EXISTE PAS": GOTO 200
50 REM : ITERER
60 PRINT "DONNEZ UN NOM (FIN=ARRET)"
70 INPUT NM$
80 REM :SORTIR SI
90 IF NM$ = "FIN" THEN : GOTO 190
100 PRINT "DONNEZ LE PRENOM"
110 INPUT PR$
120 PRINT "DONNEZ L'INDICE"
130 INPUT IN%
140 PRINT "DONNEZ L'ADRESSE"
150 INPUT AD$
160 LET "WRITE-1"
170 IF WS > 0 THEN : PRINT "ERREUR ECRITURE": GOTO 200
180 GOTO 50
190 REM : FIN ITERER
200 LET " #C,$"
210 END
```

4) Remarques :

R1 - Si on tolère les homonymes (NM$ et PR$ identiques) il suffit en ligne 160 de remplacer WRITE par ADD.

R2 - Vous pourrez constater, grâce au programme P7, que les fiches sont rangées par ordre des noms et des prénoms, quel que soit leur ordre d'entrée.

R3 - On ne mentionne plus dans l'algorithme les tests qui vérifient que tout s'est bien passé au niveau des transferts sur disquette.

P7 LECTURE SEQUENTIELLE D'UN FICHIER INDEXE MONOCLE

1) Problème précis :

Lire dans le fichier PERSO tous les enregistrements dont les NM$ sont compris entre deux bornes qui seront lues au clavier.

2) Algorithme :

. ouvrir le fichier PERSO
. lire au clavier le dernier NM$ et positionner cette borne
. lire au clavier le premier NM$
. se positionner sur l'enregistrement correspondant (il n'y en aura probablement pas : cf remarque n. 1 ci-dessous)
. **itérer**
    . lire l'enregistrement suivant
. **sortir si** cela s'est mal passé
    . afficher les valeurs lues
. **fin itérer**
. fermer le fichier PERSO

3) Programme :

```
10 REM  : PROGRAMME P7
20 LET "# C,$"
30 LET "# OPEN:1,F:0-PERSO"
40 IF WS > 0 THEN : PRINT "PERSO N'EXISTE PAS": GOTO 220
50 PRINT "DONNEZ LE DERNIER NOM"
60 INPUT NM$
70 LET "BORNE-1"
80 PRINT "DONNEZ LE PREMIER NOM"
90 INPUT NM$
100 LET "READ-1"
110 REM :ITERER
120 LET "NEXT-1"
130 REM :SORTIR SI
140 IF WS > 0 THEN : PRINT "FIN LECTURE": GOTO 210
150 PRINT "NOM     : ";NM$
160 PRINT "PRENOM : ";PR$
170 PRINT "INDICE : ";IN%
180 PRINT "ADRESSE: ";AD$
190 PRINT
200 GOTO 110
210 REM : FIN ITERER
220 LET "# C,$"
230 END
```

4) Remarques :

R.1 : On précise la valeur de NM$, mais pas celle de PR$ donc on va essayer de lire un enregistrement ayant comme clé le NM$ lu et PR$ vide (cf P7 : la clé est composée de deux variables !) : un tel enregistrement n'existe probablement pas ! Donc la lecture de l'instruction 100 ne va pas aboutir, ce que l'on pourrait vérifier par un

```
105 PRINT WS
```

Voilà pourquoi l'algorithme est différent de celui du P4.

R.2 : La même analyse est à faire pour l'arrêt.

R.3 : On aurait, bien sûr, pu lire les PR$ au clavier.

P8 EXTRACTION DES ENREGISTREMENTS DONT UNE PARTIE DE LA CLE SATISFAIT A UNE CONDITION DONNEE

1) Problème précis :

Lire dans le fichier PERSO tous les enregistrements dont la valeur de PR$ est égale à une valeur lue au clavier.

2) Algorithme :

. ouvrir le fichier PERSO
. lire la valeur de PR$
. prévenir que l'on veut extraire à partir de la deuxième rubrique de la clé
. **itérer**
. lire l'enregistrement suivant (ayant la borne valeur de PR$)
. **sortir si** on n'a pas lu... car il n'y avait plus d'enregistrement satisfaisant
. afficher les valeurs lues
. **fin itérer**
. fermer le fichier PERSO

3) Programme :

```
10 REM  : PROGRAMME P8
20 LET "#C,$"
30 LET "#OPEN:1,F:0-PERSO"
40 IF WS > 0 THEN : PRINT "PERSO N'EXISTE PAS": GOTO 200
50 PRINT "DONNEZ LE PRENOM"
60 INPUT PR$
70 LET "XTRACT-1,2"
80 LET "READ-1"
90 REM :ITERER
100 LET "NEXT-1"
110 REM :SORTIRSI
120 IF WS > 0 THEN : PRINT "FIN LECTURE": GOTO 190
130 PRINT "NOM      : ";NM$
140 PRINT "PRENOM : ";PR$
150 PRINT "INDICE : ";IN%
160 PRINT "ADRESSE: ";AD$
170 PRINT
180 GOTO 90
190 REM : FINITERER
200 LET "#C,$"
210 END
```

4) Remarques :

R.1 : On aurait pu extraire de la même façon tous les enregistrements ayant un certain NM$: il aurait suffit, en ligne 70, de le préciser : LET"XTRACT-1,1" sans oublier de modifier les lignes 50 et 60 en conséquence !!! Mais en fait cette "extraction" pourrait alors être faite plus rapidement grâce à la commande BORNE puisque tous les enregistrements ayant le même NM$ sont consécutifs.

R.2 : Les mêmes remarques concernant le non-aboutissement de la lecture de la ligne 80 sont à faire (cf P7).

P9 CREATION D'UN FICHIER AYANT UNE RUBRIQUE DATE

1) Problème précis :

Créer un fichier indexé PERSODATE dont la clé est une chaîne SS$ de 13 caractères et dont chaque enregistrement est formé de deux rubriques : une chaîne NM$ et une date DA$*. Enregistrer ensuite des fiches (on s'arrête lorsque SS$ lu vaut 000)

2) Algorithme :

. décrire la structure de PERSODATE
. créer ce fichier
. **itérer** lire SS$
. **sortir si** la valeur lue est "000"
. lire le NM$ et le DA$ correspondants
. enregistrer la fiche
. **fin itérer**
. fermer le fichier PERSODATE

3) Programme :

```
10 REM  : PROGRAMME P9
20 LET "#C,$"
30 LET "#OPEN:1,FICHIER:0-PERSODATE"
40 IF WS = 0 THEN : GOTO 70
50 LET ">SS$13=NM$,DA$*"
60 LET "#NEW:1,F:0-PERSODATE"
70 REM :ITERER
80 PRINT "NUMERO SS (000=ARRET)
90 INPUT SS$
100 REM : SORTIR SI
110 IF SS$ = "000" THEN : GOTO 190
120 PRINT "DONNEZ LE NOM"
130 INPUT NM$
140 PRINT "DONNEZ LA DATE"
150 INPUT DA$
160 LET "WRITE-1"
170 IF WS > 0 THEN : PRINT "ERREUR ECRITURE": GOTO 190
180 GOTO 70
190 REM : FIN ITERER
200 LET "#C,$"
210 END
```

4) Remarques :

R.1 : Notez bien que la variable de type DATE a comme identificateur DA$ dans l'instruction 150, mais DA$* dans la commande 50. (Cf annexe II).

R.2 : Les lettres D et A sont évidemment des lettres quelconques !!!

R.3 : Essayez quelques dates ... pour voir ce que comprend la machine vous avez peut-être intérêt à rajouter deux instructions.

```
172 LET"READ-1"
173 PRINT"DATE ENREGISTREE:";DA$
```

Chapitre III

UTILISATION DES MASQUES

1 DEFINITION ET UTILISATION DES MASQUES

Dans le vocabulaire courant, un masque est une "chose" que des personnes différentes peuvent se mettre devant le visage : elles auront toutes les mêmes traits, mais à l'endroit des trous (yeux, bouche) ce que l'on verra pourra être différent selon la personne qui porte le masque à cet instant.

En informatique, un masque est un "objet" qui est composé :
- d'un texte fixe (c'est le masque)
- de "fenêtres" (ce sont les trous ci-dessus)
- d'identificateurs de variables (l'équivalent des "yeux", "bouche" ci-dessus) : il y a autant de variables que de fenêtres.

Avant d'utiliser un masque, il faudra le créer (et le mémoriser sur disque le plus souvent). Ensuite, il pourra servir en entrée ou en sortie.

En entrée (pour la saisie des données), le texte fixe du masque est projeté sur l'écran et l'utilisateur entre les données dans les fenêtres (en tapant les valeurs au clavier) puis l'ensemble des variables dont on vient ainsi de préciser les valeurs est utilisé : le plus souvent ces variables constitueront un enregistrement et cet enregistrement sera envoyé dans un fichier.

En sortie, l'utilisation d'un masque permet de faciliter l'affichage sur écran ou les impressions grâce, en particulier, à la notion de FORMAT DE SORTIE (cela n'a rien à voir avec le formattage d'une disquette ; pour ceux qui n'ont pas cette notion, patience jusqu'au paragraphe 5).

L'utilisation des masques sera illustrée à l'aide du fichier PERSO, créé à l'aide du programme P5, qui rappelons-le, a une clé composée de deux variables NM$ et PR$ et a deux rubriques IN% et AD$.

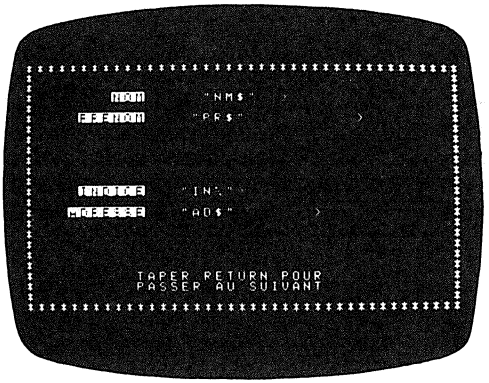
2 COMMENT CREER UN MASQUE

2 - 1 Principe de l'opération

On vient de voir qu'un masque est un objet. Il pourra donc être mémorisé sur le disque sous un certain nom (identificateur) qui figurera au catalogue sous la rubrique M. Cet objet est un ensemble d'informations : le texte fixe, les emplacements des fenêtres, les identificateurs des variables correspondant à chaque fenêtre. La création d'un masque (MSE par exemple) passe par trois étapes :
- on exécute la commande
  LET"# NEW:1,MASQUE:MSE"
  qui a pour effet de passer la main à l'utilisateur (le masque créé s'appellera MSE)
- on nettoie l'écran en tapant SHIFT-CTRL-@
- on décrit alors le masque sur l'écran grâce aux commandes qui seront détaillées ci-dessous,
- on rend la main à MEM/DOS (en tapant ESC) ce qui a pour effet de sauvegarder le masque MSE.

2 - 2 Comment décrire un masque

Le masque MSE est reproduit ci-dessous :



Il comprend 4 "fenêtres" dont les positions sont repérées par les caractères < et > et 4 identificateurs de variables qui apparaissent dans ces fenêtres. Le reste, c'est le "texte" (y compris les caractères * qui forment un cadre ... et qui ne sont évidemment là que pour faire plus joli !).

REMARQUES :

R. 1 : La position et la largeur des fenêtres sont arbitraires, mais la largeur doit être compatible avec le nombre de caractères de la variable qui y figure (et qui est malencontreusement appelée le plus souvent : "longueur") : ici, le NM$ ayant été déclaré à la création du fichier PERSO avec une "longueur" 10, la "largeur" de la fenêtre correspondante devrait être logiquement de 10. Précisons à ce sujet que < et > comptent dans le décompte ... et que, en cas de manque de place l'identificateur peut être écrit à l'extérieur de la fenêtre. Si la fenêtre n'a qu'un caractère de large, il faut la positionner grâce au seul caractère >.

R. 2 : Les identificateurs des variables apparaissent entre guillemets. Ne les oubliez surtout pas ! Ou alors MEM/DOS refusera de mémoriser votre masque lorsque vous taperez ESC (il vous rappelera à l'ordre par un coup de sonnette).

R.3 : Le caractère " ne doit absolument pas apparaître dans le texte du masque : il serait très mal digéré par MEM/DOS ; en effet, on a vu ci-dessus qu'un identificateur de variables entouré de ses " pourrait, en cas de manque de place, apparaître à l'extérieur de sa fenêtre; l'apparition d'un " où qu'il soit dans l'écran, est donc analysé par MEM/DOS comme le début d'un identificateur de variable. Pour la même raison, les caractères < et > ne doivent pas intervenir dans le texte.

R.4 : En fait, les identificateurs des variables peuvent apparaître n'importe où ! Lorsque MEM/DOS analyse un masque, il le balaye ligne par ligne de haut en bas et constitue, au fur et à mesure qu'il les rencontre, une suite de fenêtres et une suite d'identificateurs de variables ; il met en correspondance ces deux suites : la première fenêtre rencontrée avec le premier identificateur, et ainsi de suite.

Maintenant que l'on sait ce que l'on doit faire, voyons comment le faire : essentiellement comment décrire un tel masque dans un écran vide après avoir exécuté la commande

LET "#NEW:1,MASQUE:0-MSE"

Le curseur se déplace horizontalement grâce aux flêches du clavier et verticalement par CTRL-R (haut) et CTRL-Z ou RETURN (bas). Pour répéter un caractère sur une même ligne, il suffit sur APPLE II d'utiliser la touche REP et sur APPLE IIe de maintenir la pression, comme d'habitude. Mais en plus, sur les deux versions, on peut répéter un caractère verticalement vers le bas en plaçant le curseur sur ce caractère et en tapant CTRL-V. Les caractères peuvent être écrits normalement, ou inversés ou clignotants grâce à l'utilisation de CTRL-N, CTRL-B (pour fond Blanc) ou CTRL-F (pour Flash). Enfin, on peut effacer tout l'écran par SHIFT-CTRL- @

Vous pourrez vous entrainer à créer le masque MSE (ou n'importe quel autre). N'oubliez pas de faire ESC quand la description est terminée ... et si vous entendez alors une sonnette ... c'est peut-être que vous avez oublié de mettre les variables entre ".

3 UTILISATION D'UN MASQUE EN SAISIE

Supposons que le masque MSE soit créé et que l'on veuille l'utiliser pour saisir des données et les enregistrer dans le fichier PERSO. Il faut écrire un programme dans lequel on commence par "ouvrir" le masque (en lui attribuant un identificateur temporaire : B par exemple) par LET" #OPEN:B,MASQUE:0-MSE" puis on le projette sur l'écran grâce à la commande

LET"CHARGE-B"

Ensuite on peut saisir plusieurs enregistrements : la commande

LET"INPUT-B"

saisit l'ensemble des variables du masque : lors de l'exécution, l'utilisateur doit remplir les fenêtres du masque et taper ESC lorsqu'il les aura correctement remplies ; la touche RETURN permet simplement de passer à la fenêtre suivante ... et la dernière fenêtre est "suivie" par la première, ce qui permet de faire des corrections et ce qui explique que la touche RETURN ne peut pas servir à la saisie de l'ensemble des variables : c'est la touche ESC qui remplit cette fonction. Le programme P10 illustre ces notions. La commande

LET"PRINT-B"

permet à tout instant de visualiser le texte et les valeurs qu'ont actuellement les variables. C'est une commande particulièrement utile lorsqu'on veut faire modifier par l'utilisateur une ou plusieurs de ces valeurs : on exécute succesivement une commande PRINT et une commande INPUT. La commande

LET"OUTPUT-B"

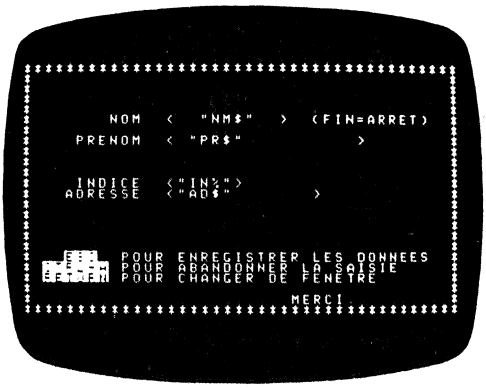
affiche sur l'écran les valeurs des variables à l'emplacement de leurs fenêtres respectives, mais sans modifier le reste de l'écran et donc sans afficher le texte du masque. Enfin, la commande

LET"VISUALISE-B"

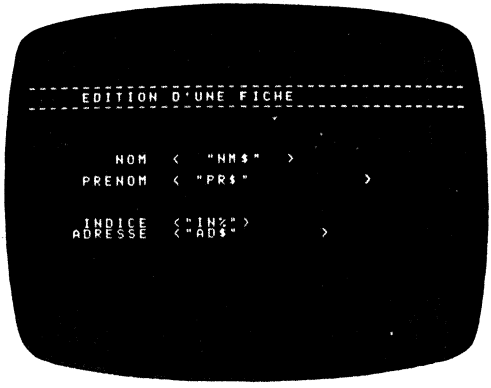
affichera simplement le masque complet : texte, fenêtres et identificateurs des variables.

4 UTILISATION D'UN MASQUE EN SORTIE ECRAN OU PAPIER

Le masque MIMPRIM est utilisé dans le programme P12 de sortie sur imprimante, alors que le programme P11 de sortie sur écran utilise le masque MSORTIE. Ces masqes ont été créés pour vous sur la disquette MEM/DOS afin que vous puissiez essayer directement les programmes P11 et P12.



M SORTIE



M IMPRIM

5 LES FORMATS DE SORTIE

Le BASIC de l'APPLE ne permet pas d'obtenir facilement des affichages "élégants" de tableaux de noms ou de valeurs numériques : il est difficile d'arrondir des valeurs à un nombre donné de chiffres derrière la virgule par exemple.

On appelle format de sortie l'ensemble des renseignements permettant de décrire comment on veut présenter une information : nombre total de cases (ou positions), position de la virgule (donc nombre de chiffres derrière la virgule), calage à gauche ou à droite dans l'emplacement réservé. Cette possibilité de préciser un format de sortie existe sur certains BASIC ; elle est alors souvent appelée PRINT USING.

Les masques de MEMDOS permettent, tant pour l'affichage sur l'écran que pour l'impression, un "formattage" des valeurs numériques : il y a dans MEM/DOS trois formats possibles et le choix s'effectue lorsqu'on précise l'identificateur à afficher dans la fenêtre de l'écran ; par "défaut", c'est le format classique du BASIC (cadrage à gauche, quantités écrites comme d'habitude) ; si la variable est suivie des caractères a * ("FL a *" par exemple) la valeur sera cadrée à droite et les quantités écrites comme d'habitude ; enfin, si la variable est suivie de * seulement ("FL*" par exemple) les valeurs sont cadrés à droite avec un nombre constant de chiffres derrière la virgule : c'est ce que l'on appelle le format "gestion" ; par défaut, il y aura 2 chiffres derrière la virgule, mais ce nombre peut être modifié par la commande

LET" @ -3"

par exemple, si on veut trois chiffres au lieu de deux.

6 CONCLUSION

Nous ne pouvons que reprendre les termes de la conclusion du chapitre II : ce que vous avez vu dans cette notice n'est qu'une introduction à l'utilisation des masques. En particulier en ce qui concerne l'utilisation de tableaux dans les masques (ou les fichiers), les caches dans les masques, les sorties sélectives, l'impression ligne à ligne... etc ... Grâce à ces outils, vous améliorerez à la fois la qualité de vos programmes (lisibilité, rapidité) et votre productivité.

7 LES PROGRAMMES P10 A P12

On trouvera dans les pages suivantes les programmes P10 à P12

P10 UTILISATION D'UN MASQUE EN SAISIE DE DONNEES

(ATTENTION: si vous n'avez pas créé le masque MSE, créez le, ou exécutez le programme PSECOURS qui crée MSE à partir du masque MSECOURS qui existe d'origine).

1) Problème précis :

Grâce au masque MSE, saisir plusieurs enregistrements destinés au fichier PERSO et les y charger. S'arrêter lorsqu'on lit FIN pour NM$.

2) Algorithme :

Ouvrir le masque MSE et le fichier PERSO.
Afficher le texte du masque MSE.
. **itérer**
. lire au clavier les valeurs du masque MSE
. **sortir si** NM$ = "FIN" ou si l'utilisateur abandonne
. écrire l'enregistrement dans le fichier PERSO
. **fin itérer**
. Effacer l'écran
. Fermer tous les objets.

3) Programme :

```
10 REM  : PROGRAMME P10
20 LET "#C,$"
30 LET "# OPEN:M,MASQUE:0-MSE"
40 IF WS THEN : PRINT "ERREUR OUVERTURE MSE": GOTO 160
50 LET "# OPEN:F,FICHIER:0-PERSO"
60 IF WS THEN : PRINT "ERREUR OUVERTURE PERSO": GOTO 160
70 LET "CHARGE-M"
80 REM   : ITERER
90 LET "INPUT-M"
100 REM :SORTIR SI
110 IF NM$ = "FIN" OR WS THEN: GOTO 140
120 LET "WRITE-F"
130 GOTO 80
140 REM:FIN ITERER
150 HOME
160 LET " #C,$"
170 END
```

4)Remarques :

R.1 : Les identificateurs temporaires des objets doivent évidemment être différents, même s'il s'agit d'objets de type différent. Ici, on a choisi M pour le masque et F pour le fichier, mais ce n'est pas obligatoire.

R.2 : L'instruction CHARGE étant placée à l'extérieur de la boucle itérer, quand on tape ESC pour enregistrer le masque puis la fiche et que l'on revient à l'instruction 90, les valeurs des variables restent affichées, ce qui accélère la saisie s'il y a des variables à conserver pour l'enregistrement suivant.

R.3 : Il est dangereux de mettre l'instruction CHARGE dans la boucle itérer : ce n'est pas parce que les valeurs des variables ne seraient plus affichées qu'elles seraient remises à zéro ou à blanc : l'utilisateur pourrait croire qu'il saisit une fiche en laissant des zones vides alors que ces zones auraient la valeur de la fiche précédente ! Par contre, on peut mettre un LET"PRINT-M" dans la boucle.

P11 UTILISATION D'UN MASQUE POUR SORTIE ECRAN

1) Problème précis :

Grâce au masque MSORTIE, afficher séquentiellement tous les enregistrements du fichier PERSO. Après l'affichage d'un enregistrement, attendre que l'utilisateur tape RETURN pour afficher le suivant.

2) Algorithme :

Ouvrir le masque MSORTIE et le fichier PERSO.
. Afficher le masque vide.
. **itérer**
    . lire l'enregistrement suivant de PERSO
. **sortir si** on n'a rien lu
    . afficher les valeurs sur le masque (sur l'écran)
    . attendre que l'utilisateur tape un caractère
. **fin itérer**
. Effacer l'écran
. Fermer les objets

3) Programme :

```
10 REM  : PROGRAMME P11
20 LET "  C,$"
30 LET "# OPEN:1,M:0-MSORTIE"
40 LET "# OPEN:2,F:0-PERSO"
50 REM : ITERER
60 LET "CHARGE-1"
70 LET "NEXT-2"
80 REM :SORTIR SI
90 IF WS > 0 THEN : GOTO 130
100 LET "OUTPUT-1"
110 GET C$
120 GOTO 50
130 REM : FIN ITERER
140 HOME
150 LET "# C,$"
160 END
```

4) Remarques :

R.1 : On lit un caractère (GET) plutôt qu'autre chose (INPUT) afin de ne pas décaler le masque sur l'écran d'une ligne vers le haut. On aurait pu tout aussi bien utiliser un masque uniquement pour cette saisie. C'est ce qui sera fait dans l'exemple du chapitre V.

R.2 : Il serait prudent de rajouter des instructions vérifiant que les ouvertures se sont bien effectuées.

P12 UTILISATION D'UN MASQUE POUR IMPRESSION

1) Problème précis :

Grâce au masque MIMPRIM afficher et imprimer séquentiellement les enregistrements du fichier PERSO. Après chaque impression, on attend la lecture d'un caractère et on s'arrête d'imprimer si ce caractère est A.

2) Algorithme :

. Ouvrir le masque MIMPRIM et le fichier PERSO
. **itérer**
    . lire l'enregistrement suivant de PERSO
. **sortir si** on n'a rien lu
    . afficher le masque sur l'écran avec ses valeurs
    . recopier l'écran sur l'imprimante
    . lire un caractère au clavier
. **sortir si** le caractère lu est "A"
. **fin itérer**
. Effacer l'écran
. Fermer les objets

3) Programme :

```
10 REM  : PROGRAMME P12
20 LET "# C,$"
30 LET "# OPEN:1,M:0-MIMPRIM"
40 LET "# OPEN:2,F:0-PERSO"
50 REM : ITERER
60 LET "NEXT-2"
70 REM : SORTIR SI
80 IF WS THEN: GOTO 190
90 LET "CHARGE-1"
100 LET "OUTPUT-1"
110 PR # 1: POKE 33,1: POKE 35,1: VTAB 1: HTAB 1: LET "?-*":
TEXT : PR # 0
120 HOME
130 PRINT "TAPEZ A POUR ARRETER"
140 PRINT "OU RETURN POUR CONTINUER"
150 GET LU$
160 REM :SORTIR SI
170 IF LU$ = "A" THEN: GOTO 190
180 GOTO 50
190 REM :FIN ITERER
200 HOME
210 LET "# C,$"
220 END
```

* attention, voir les remarques page 44.

4) Remarques :

R.1 : L'instruction 110 nécessiterait un long développement pour être clairement expliquée. Utilisez-la comme une recette de cuisine : elle recopie l'écran sur l'imprimante connectée sur le slot 1.

R.2 : Le masque vide a été placé dans la boucle à cause des messages 130 et 140 qui décalent l'écran.

R.3 : Il existe une commande LET"PRINT-1" qui a le même effet qu'un CHARGE-1 suivi d'un OUTPUT-1. Attention, ce n'est pas parce que cette commande s'appelle PRINT qu'elle imprime : non, cette commande affiche sur l'écran. C'est pour éviter cette confusion qu'on ne l'a pas utilisée dans ce programme, mais on peut le faire.

CHAPITRE IV :

LES SOUS-PROGRAMMES

1 INTRODUCTION

Une des clés de l'efficacité en informatique, comme ailleurs, réside dans la réduction des difficultés par le fractionnement grâce à la technique de la programmation modulaire. Les langages les moins évolués contiennent la notion de sous-programme : en BASIC la fameuse paire GOSUB, RETURN permet d'éviter la répétition d'une suite d'instructions utilisée en plusieurs endroits. MEM/DOS permet d'aller beaucoup plus loin dans la modularisation car il ajoute au BASIC des notions réservées jusqu'alors aux langages les plus modernes (PASCAL, ADA) : paramètres formels et effectifs, variables locales et globales, récursivité !

Avant de décrire l'utilisation des sous-programmes avec MEM/DOS, nous allons rappeller brièvement le principe même de l'utilisation des sous-programmes et quelques définitions.

Un programme est une suite d'instructions que vous pouvez faire exécuter une ou plusieurs fois en changeant éventuellement les conditions d'exécution grâce à la lecture de certaines valeurs. Un sous-programme est, de la même façon, une suite d'instructions qui peut être exécutée une ou plusieurs fois en changeant éventuellement certaines conditions d'exécution ; mais la différence essentielle est qu'ici, l'exécution est déclenchée automatiquement par une instruction spéciale CALL placée au bon endroit dans le programme qui utilise le SP et que l'on appelle le programme appelant ou programme principal. Comme pour un programme, l'utilisation d'un sous-programme se décompose donc en deux phases : la description du traitement qui est faite une fois pour toutes, et l'exécution de ce traitement. Nous allons voir maintenant que les échanges d'informations entre le SP et le programme principal peuvent se faire de plusieurs façons différentes dont il est important de bien comprendre le pourquoi et le comment.

Variables globales et variables locales.

Dans un programme, la valeur des variables peut être fixée par affectation du résultat du calcul d'une expression ou par une lecture (au clavier, dans un fichier ou en mémoire centrale par un PEEK) ; dans un sous-programme, la valeur d'une variable peut aussi provenir du programme appelant : si une variable EX par exemple est utilisée dans un sous-programme et que la valeur 12 lui a été affectée lors de l'exécution du programme principal avant l'appel du SP, cette valeur 12 est utilisée dans le sous-programme où elle peut d'ailleurs être modifiée ; après l'exécution du SP la valeur acquise reste affectée à la variable EX et peut être utilisée dans la suite du programme principal. Une telle variable est appelée globale.

On sent immédiatement le besoin d'une autre espèce de variable : en effet, si par exemple l'appel d'un SP se fait depuis l'intérieur d'une boucle gérée par un indice K variant de 1 à N et si dans le SP la valeur de K est modifiée, vous imaginez facilement ce qui va se passer : avant le premier appel, K vaut 1 mais après exécution du SP, K ne vaut plus 1, donc la boucle du programme principal aura un fonctionnement pour le moins inattendu !!! La notion de variable locale répond à cette préoccupation ; si une variable K est déclarée locale lors de la description du SP (et on verra plus loin comment), alors pour la durée de l'exécution de ce SP, le système réserve un emplacement en mémoire pour cette variable, s'en sert et libère la place en fin d'exécution du SP. S'il existe dans le programme appelant une variable K, il y a une distinction totale entre les deux variables K : aucune valeur ne passe de l'une à l'autre : ce sont deux objets distincts.

MEM/DOS permet cette distinction entre variable locale et variable globale ; nous verrons comment préciser les choses au paragraphe 2.

Paramètres formels et paramètres effectifs.

Si on ne pouvait utiliser que des variables locales ou globales pour la description du traitement à effectuer par le SP, on serait limité : une même variable devrait avoir le même nom lors de la description du SP et lors de son utilisation ; or, pour une certaine variable, on peut être amené à décrire le traitement en lui donnant un nom, X et à utiliser ce SP successivement pour une variable qui s'appelle Y dans le programme appelant puis une autre qui s'appelle Z, c'est ce que vous faites couramment lorsque vous utilisez en Basic une fonction comme SIN(Y) ou SIN(Z). D'où la notion de "paramètre formel" X et de "paramètre effectif" Y ou Z. Un paramètre formel est un objet servant lors de la description de la "forme" du SP et un paramètre effectif est celui sur lequel le traitement est "effectivement effectué". Le mot argument est parfois utilisé comme synonyme de paramètre. Ici encore, MEM / DOS vous permet d'utiliser cette notion de paramètre formel et de paramètre effectif.

2 DEFINITION D'UN SOUS-PROGRAMME ET APPEL

La définition d'un sous programme en MEM/DOS étend le rôle de l'instruction DEF FN : rappelons d'abord qu'en Applesoft l'exécution de la séquence

```
10 DEF FN A(W) = 2 * W + 3
20 PRINT FN A(4)
```

a pour résultat d'afficher la valeur 11. Les deux restrictions principales de cette instruction en Applesoft sont que la fonction doit être définie en une seule ligne et qu'il ne peut y avoir qu'un seul paramètre et qu'il doit être de type numérique.

Sous MEM/DOS, la définition d'un sous-programme se fait de la façon suivante :

```
DEF FN "nom du sous-programme" (liste d'identificateurs de variables)
...
...
...
...
... suite d'instructions
...
...
END FN
```

Le nom du sous-programme est limité à 20 caractères. La liste d'identificateurs de variables précise la suite des paramètres formels suivie de la liste des variables locales.

L'exécution d'un sous-programme est déclenchée par l'instruction :

CALL FN "nom du sous-programme" (liste des paramètres effectifs).

Il faut remarquer qu'il ne faut pas répéter la liste des variables locales à la suite de la liste des paramètres effectifs. C'est inutile, et de plus c'est par cette différence que MEM/DOS sépare les paramètres et les variables locales. Précisons par ailleurs que la transmission des paramètres se fait par adresse : MEM/DOS attribue au paramètre effectif le même emplacement mémoire (la même adresse) que le paramètre formel : donc toute modification de la valeur du paramètre formel affectera, après la fin du sous-programme, le paramètre effectif. Cela entraine également que les paramètres effectifs ne peuvent être que des variables.

Lors de la définition, nous vous conseillons, mais ce n'est pas obligatoire, d'utiliser un séparateur différent pour isoler la liste des paramètres de la liste des variables locales et un autre séparateur pour distinguer les paramètres d'entrée des paramètres de sortie, même si ces séparations n'ont aucun effet réel. On propose

s1, s2, ... = e1, e2, .../l1, l2 ...

les identificateurs s1, s2 représentant les paramètres dont la valeur résulte d'un traitement effectué dans le sous-programme, les identificateurs e1, e2, ... ceux dont la valeur est connue à l'appel du sous-programme et les identificateurs l1, l2, ... désignant les variables locales. Un sous-programme peut n'avoir ni paramètres, ni variables locales ; il se comporte alors comme une SUBROUTINE ; mais un sous-programme ne peut avoir de variables locales s'il n'a pas de paramètres (on peut tourner cette restriction en déclarant un paramètre "bidon").

3 EXEMPLES SIMPLES

L'exemple SP0 permet d'illustrer les distinctions de base entre paramètres formel et effectif, d'entrée et de sortie, les variables locales et globales.

L'exemple SP1 montre que MEM / DOS ne fait pas de distinction entre les paramètres d'entrée et de sortie : il les transmet par adresse et donc, si un paramètre dit d'entrée est modifié dans le sous-programme, sa valeur modifiée est transmise au programme principal. Donc MEM / DOS ne fait pas la différence qui existe en PASCAL entre paramètres "valeur" et paramètres "variables" : tous les paramètres MEM/DOS sont "variables" au sens PASCAL.

L'exemple SP2 montre que des tableaux à une dimension peuvent figurer comme paramètres ; leurs identificateurs sont alors suivis d'un point-virgule et leurs tailles doit être précisées par une instruction DIM dans le programme appelant. Par contre un tableau ne peut pas figurer dans la liste des variables locales.

4 RECURSIVITE ET IMBRICATION

Il n'existe probablement pas de texte sur la récursivité qui ne cite l'exemple de la factorielle. Vous trouverez la version MEM/DOS dans l'exemple SP3. En ce qui concerne l'imbrication, précisons qu'un sous-programme peut en appeler un autre mais qu'un sous-programme ne peut pas être défini dans la définition d'un autre sous-programme ; autrement dit, la notion de localité n'existe pas pour les sous-programmes. Enfin, une des grosses sources d'erreurs difficiles à détecter dans les programmes comportant plusieurs modules provient du pari que font les programmeurs en ce qui concerne le mode de recherche des variables globales lors d'appels imbriqués : l'exemple SP4 montre que cette recherche est faite dynamiquement, c'est-à-dire selon la séquence d'appel ; contrairement au PASCAL où cette recherche est faite statiquement : selon la séquence de définition.

L'exemple SP5 montre la dernière facilité que vous procure MEM/DOS (qui se montre là supérieur au PASCAPPLE) : un sous-programme peut-être paramètre d'un autre sous-programme. On verra au chapitre V que des fichiers peuvent également être paramètres ce qui est strictement interdit en PASCAL.

LES SOUS-PROGRAMMES SP0 à SP5

SP0 : PRESENTATION DES DIFFERENTES NOTIONS

1) Problème précis :

Ecrire un sous-programme FACT qui calcule de manière itérative la factorielle de X (c'est le produit des X premiers entiers). S'en servir dans un programme principal pour calculer la factorielle de différents entiers. Le symbole de la fonction factorielle est le point d'exclamation et par convention 0! = 1.

2) Liste du programme :

```
10 REM  : ITERER
20 INPUT "DONNEZ UN ENTIER POSITIF";N%
30 REM  : SORTIR SI
40 IF N% < = 0 THEN : PRINT "AU REVOIR": GOTO 90
50 PRINT "OK, VOICI SA FACTORIELLE..."
60 CALL  FN "FACT"(RES=N%)
70 PRINT N%;"!=";RES: PRINT
80 GOTO 10
90 REM :FIN ITERER
100 END
110 DEF  FN "FACT"(Y = X%/K)
120 IF X% < 0 THEN : PRINT "ERREUR": GOTO 170
130 Y = 1
140 FOR K = 1 TO X%
150 Y = Y * K
160 NEXT K
170 END  FN
```

3) Remarques :

R1 - RES, N% sont les paramètres effectifs correspondant aux paramètres formels Y et X%. La variable K est une variable locale.

R2 - Les sous-programmes doivent être placés hors du programme principal.

R3 - La valeur d'une factorielle est toujours un entier, mais on le calcule comme un réel car c'est une fonction qui croît très vite :

```
DONNEZ UN ENTIER POSITIF 12
OK, VOICI SA FACTORIELLE...
12!=479001600
DONNEZ UN ENTIER POSITIF 13
OK, VOICI SA FACTORIELLE...
13!=6.2270208E+09
```

SPI : ETUDE DE LA TRANSMISSION DES PARAMETRES

1) Problème précis :

Ecrire un sous-programme qui utilise et modifie les deux paramètres formels S et E et montrer en l'utilisant que les valeurs de ces paramètres sont bien transmises à l'entrée et à la sortie du sous-programme. Vérifier également qu'une variable locale perd sa valeur à la sortie du sous-programme.

2) Liste du programme :

```
10 LOCALE = 55:GLOBALE = 2
20 CALCULEE = 10
30 ENTREE = 100
40 CALL  FN "SE"(CALCULEE = ENTREE)
50 PRINT "SORTIE DU SOUS-PROGRAMME"
60 PRINT "ENTREE=";ENTREE;"  CALCULEE=";CALCULEE
70 PRINT "LOCALE=";LOCALE;"  GLOBALE=";GLOBALE
80 END
90 DEF  FN "SE"(S = E/LOCALE)
100 LOCALE = 1
110 PRINT "S=";S
120 S = S + LOCALE + E
130 PRINT "S=";S
140 FOR K = 1 TO GLOBALE
150 PRINT "E=";E
160 E = E + LOCALE + S
170 NEXT K
180 END  FN
```

3) Remarques :

R1 - La manière efficace d'étudier cet exemple consiste à le faire exécuter à la main, ce qui revient à remplir les cases ci-dessous :

```
S =
S =
E =
E =
SORTIE DU SOUS-PROGRAMME
ENTREE=          CALCULEE=
LOCALE=          GLOBALE=
```

R2 - Faites exécuter et comparez. En cas de différence, retour au paragraphe IV.1.

SP2 : UTILISATION DES TABLEAUX COMME PARAMETRES

1) Problème précis :

Ecrire un sous-programme SOMTAB qui calcule la somme S des N éléments d'un tableau T. L'utiliser pour calculer la somme TT des NB éléments d'un tableau U dans lequel on a placé les NB premiers entiers.

2) Liste du programme :

```
10 INPUT NB
20 DIM U(NB)
30 FOR K = 1 TO NB
40 U(K) = K
50 NEXT K
60 CALL  FN "SOMTAB"(TT = U;,NB)
70 PRINT TT
80 END
1000 DEF  FN "SOMTAB"(S = T;,N/K)
1010 S = 0
1020 FOR K = 1 TO N
1030 S = S + T(K)
1040 NEXT K
1050 END  FN
```

3) Remarques :

R1 - Le tableau doit être dimensionné dans le programme principal.

R2 - Les identificateurs de tableaux U et T sont suivis d'un ;.

R3 - Il y a deux variables K : une globale et une locale : il n'y a pas de collision.

R4 - Dans l'instruction 60 n'appellez surtout pas TOTAL la somme des éléments du tableau U. (TO est un mot réservé du Basic)

SP3 : CALCUL RECURSIF DE N!

1) Problème précis :

La définition de N! donnée dans l'exemple SP0 montre que cette fonction peut être définie récursivement :

```
SI N = 0
   ALORS N! = 1
   SINON N! = N*(N-1)!
FINSI
```

Le problème est donc d'écrire un sous-programme basé sur cette définition récursive.

2) Liste du programme :

```
10 REM  : ITERER
20 INPUT "DONNEZ UN ENTIER POSITIF : ";N%
30 REM  : SORTIR SI
40 IF N% < 0 THEN : PRINT "TANT PIS, J'AVAIS DIT POSITIF !!!": GOTO 100
50 PRINT "MERCI, J'EN CALCULE LA FACTORIELLE.....": PRINT
60 CALL  FN "FACT"(RES=N%)
70 PRINT N%;"!=";RES: PRINT
80 GOTO 10
90 REM  : FIN ITERER
100 END
1000 DEF  FN "FACT"(Y = X%/K%,U)
1010 IF X% < 0 THEN : PRINT "ERREUR": GOTO 1100
1020 IF X% = 0 THEN : Y = 1: GOTO 1100
1030 K% = X% - 1
1040 CALL  FN "FACT"(U=K%)
1050 Y = U * X%
1100 END  FN
```

3) Remarques :

R1 - Les résultats sont identiques à ceux de SP0.

R2 - En général un programme récursif est difficile à mettre au point, et souvent plus long à l'exécution. Cependant pour certains problèmes fondamentalement récursifs, c'est-à-dire manipulant des structures de données récursives (arbres), la programmation récursive est bien agréable.

SP4 : APPELS IMBRIQUES.

1) Problème précis :

Ecrire un sous-programme P qui se contente d'afficher la valeur du paramètre A$. Ecrire un sous-programme Q qui affecte une valeur à une variable locale A$, l'affiche et lance l'exécution de P. Enfin, écrire un programme principal qui affecte une valeur à une variable globale A$ l'affiche et lance successivement l'exécution de P puis Q. La question qui se pose est de savoir ce que va imprimer P lorsqu'il est appelé depuis Q : va-t'il imprimer la valeur de la variable locale A$ de Q ou la valeur de la variable globale A$?.

2) Liste du programme :

```
10 A$ = "PRINCIPAL"
20 PRINT A$
30 CALL  FN "P"
40 CALL  FN "Q"(BID)
50 END
1000 DEF  FN "P"
1010 PRINT "JE SUIS DANS P ... A$=";A$
1020 END  FN
2000 DEF  FN "Q"(BID/A$)
2010 A$ = "QQQQQQQQQQ"
2020 PRINT "JE SUIS DANS Q ... A$=";A$
2030 CALL  FN "P"
2040 END  FN
```

3) Remarques :

R1 - BID est une variable "Bidon" : le sous-programme Q utilisant une variable locale doit avoir au moins un paramètre.

R2 - Un traitement analogue avec le PASCAL de l'Apple aurait donné PRINCIPAL comme troisième impression.

SP5 : SOUS-PROGRAMMES PARAMETRES.

1) Problème précis :

Ecrire un sous-programme qui calcule la somme Z = F(A) + F(B) où A et B sont deux fonctions lues au clavier et où F est également le nom d'une fonction définie par un sous-programme. Ici, F pourra être soit SINUS, soit COSINUS.

2) Liste du programme :

```
100 REM  SP5
110 PRINT "NOM DE LA FONCTION ( SINUS OU COSINUS )"
120 INPUT F$
130 PRINT "ENTREZ LA VALEUR DE A"
140 INPUT A
150 PRINT "ENTREZ LA VALEUR DE B"
160 INPUT B
170 CALL  FN "CALCUL"(Z = A,B,F$)
180 PRINT "RESULTAT = "Z
190 END
200 DEF  FN "CALCUL"(R = X,Y,V$/T)
210 CALL  FN "" + V$ + ""(R,X)
220 CALL  FN "" + V$ + ""(T,Y)
230 R = R + T
240 END  FN
250 DEF  FN "SINUS"(S = P)
260 S = SIN (P)
270 END  FN
280 DEF  FN "COSINUS"(C = P)
290 C = COS (P)
300 END  FN
```

3) Remarques :

R1 - Notez bien la syntaxe des instructions 210 et 220 dans lesquelles on lance l'exécution de la fonction V$ dont le nom est donné dans le programme principal.

R2 - Ce type de programmation est indispensable si on veut par exemple écrire un sous-programme général, de calcul de l'intégrale d'une fonction quelconque (ce qui est impossible en PASCAPPLE).

Chapitre V

EXEMPLE RECAPITULATIF

1 INTRODUCTION

Le but de ce chapitre est de montrer, sur un petit exemple récapitulatif, l'utilisation conjointe des fichiers, des masques et des sous-programmes. Il montre comment on peut gérer les prêts dans une bibliothèque, avec la simplification qu'un emprunteur ne peut sortir qu'un livre à la fois. La structure des données et des traitements qui a été adoptée n'est pas la seule, ni peut-être la meilleure ; elle est exposée sans justification dans le deuxième paragraphe. Le troisième décrit schématiquement le mode d'utilisation ; vous êtes invité à l'essayer sur votre ordinateur avant de lire le quatrième paragraphe dans lequel les différents modules sont expliqués et commentés.

Bien comprendre une application de cette envergure ne peut se faire en quelques minutes : ne vous étonnez donc pas si l'étude de ce chapitre vous prend du temps, c'est tout-à-fait normal. De plus, si un lecteur courageux entreprend de réécrire cette application, pour s'entraîner, ou une autre de difficulté comparable, qu'il ne s'étonne pas s'il lui faut quelques heures, (voire quelques dizaines d'heures) pour la mettre au point. MEM/DOS facilite le travail du programmeur et lui permet d'aboutir plus rapidement à un programme qui tourne, mais il n'annule pas totalement le travail à fournir !

2 STRUCTURE GENERALE DES DONNEES ET DES TRAITEMENTS

Pour la gestion de cette bibliothèque dans laquelle les emprunteurs ne peuvent emprunter qu'un livre à la fois, on a choisi de n'utiliser que deux fichiers et de découper le traitement en plusieurs modules ayant chacun une fonction précise.

2 - 1 Structure des données

Deux fichiers indexés, un pour les livres et un pour les emprunteurs, les renseignements sur les prêts étant reportés dans ces fichiers.

Fichier LIVRES :
La clé est composée d'un code de rayon sur deux caractères et d'un numéro d'ordre dans ce rayon qui est un entier ; les rubriques sont l'auteur, le titre, l'éditeur, la date d'achat et la date d'emprunt si ce livre est sorti.

Fichier EMPRUNTEURS :
La clé est composée d'un nom d'emprunteur sur 15 caractères ; les rubriques sont d'une part le nom suivi du prénom, l'adresse en trois zones et le numéro de téléphone, qui identifient l'emprunteur ; d'autre part les caractéristiques du livre éventuellement emprunté : code de rayon et numéro.

2 - 2 Structure des traitements

La mise en œuvre de cette application nécessite trois opérations distinctes :
- une opération faite une fois pour toutes en conversationnel : créer les masques et écrire les programmes, ce qui est le travail de l'informaticien
- une opération faite également une fois pour toutes, mais par programme: créer les fichiers
- ensuite le bibliothécaire pourra utiliser en permanence le programme général de gestion de sa bibliothèque.

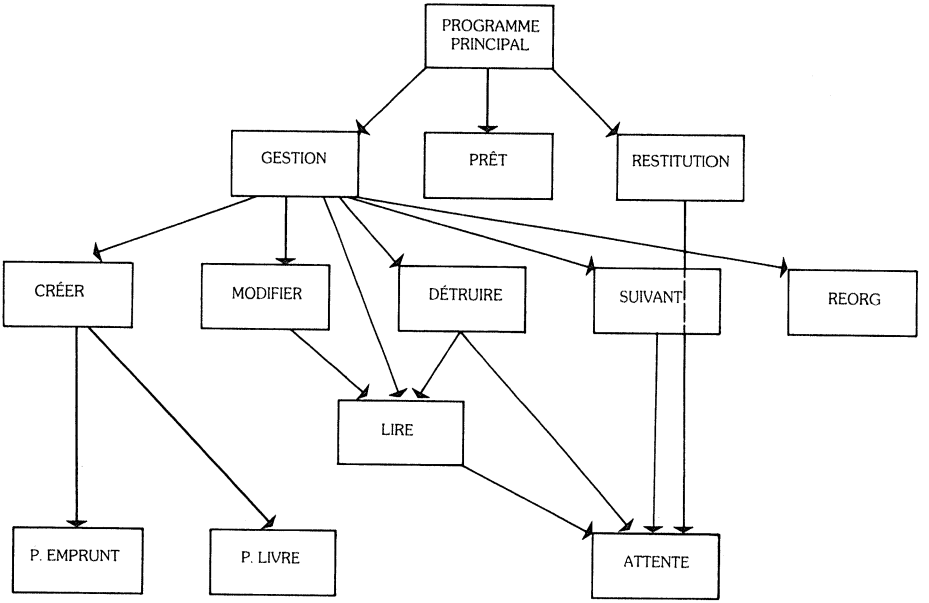
Les masques sont décrits plus loin et leur création ne sera même pas évoquée. Le programme de création des fichiers est tellement simple qu'on va le liquider tout de suite : il se compose des cinq instructions ci-dessous qui ne nécessitent aucun commentaire.

```
100 LET "#C,$"
110 LET ">RA$2,NU%=AU$,TI$,ED$,DA$*,EM$,DE$*"
120 LET "#N,F,F,LIVRE"
130 LET ">EM$15=N0$,A1$,A2$,A3$,TL$,RA$,NU%"
140 LET "#N,G,F,CLIENT"
```

2 - 3 Le "gros" programme de gestion de cette bibliothèque

Il y a quatre entités à gérer : un livre, un emprunteur, un prêt et une restitution. Gérer un livre ou un emprunteur nécessitent les mêmes opérations : créer, lire, modifier ou détruire un enregistrement, parcourir le fichier en séquence, et le réorganiser de temps en temps pour diminuer les temps de recherche. Par contre, les opérations de prêt et de restitution sont très différentes et il y aura donc deux modules distincts. Remarquons enfin que pour des motifs de simplicité d'utilisation, il a été créé un petit module ATTENTE qui a pour seul objet de lire une réponse au clavier. Tous ces modules sont décrits au paragraphe 4.

Le programme principal est simplement un menu. La structure des appels des modules est décrite dans le schéma ci-dessous.



MODE D'UTILISATION

Les fichiers ont été créés sur la mini-disquette DEMO et on y a enregistré quelques livres et emprunteurs. Lorsqu'on sélectionne cette mini-disquette, le programme principal se charge et s'exécute : on voit donc apparaître le menu principal. L'utilisation des différents modules ne présente pas de difficultés ; elle peut être facilitée par les quelques remarques suivantes :

- la plupart des masques sont affichés avec les valeurs qu'ont les variables en mémoire ; cela peut surprendre, mais on a constaté à l'usage que c'était agréable en cas de modification et comme il est très simple d'effacer le contenu d'une fenêtre (CTRL-SHIFT-@), cette option est couramment retenue par les professionnels.

- n'oubliez pas que dans un masque on peut toujours revenir à une fenêtre précédente par CTRL-R et passer à la suivante par RETURN.

- si on veut abandonner un masque en cours d'édition pour une raison quelconque (on s'est trompé d'option, on a changé d'avis, etc ...) il suffit de taper CTRL-A, ce qui a pour effet secondaire de donner la valeur 1 à la variable WS. Les programmes testent cette variable après chaque saisie et, dans le cas où elle est positive, le traitement se poursuit par retour à l'affichage du menu précédent.

- Si dans une fenêtre l'identificateur de variable est suivi d'un point d'interrogation (AB$? par exemple), cette variable n'est pas lue au clavier dans le cas d'un OUTPUT ou d'un PRINT. Cela permet en particulier deux choses : d'une part de faire varier le texte d'un masque en utilisant une variable chaîne dans laquelle le programme place un message circonstancié avant de l'afficher ; d'autre part d'afficher des valeurs calculées ou lues par ailleurs.
- si dans une fenêtre l'identificateur de variable est suivi du caractère : (AB: par exemple) cette variable est à enchaînement automatique : dès que la fenêtre est remplie, MEM/DOS passe à la fenêtre suivante.

4 DESCRIPTION DES DIFFERENTS MODULES

On trouvera dans les pages suivantes la description de chaque module : sa fonction, l'algorithme, la liste des instructions et les masques qu'il utilise. l'étude détaillée de ces modules, simultanément à leur utilisation permettra de mieux comprendre le mode d'emploi des masques, des fichiers et des sous-programmes. Ces modules forment un tout et c'est leur utilisation simultanée qui a un sens. La commande LIST FN donne la liste des sous-programmes. La commande LIST FN "TOTO" donne la liste des instructions du sous-programme TOTO.

Les modules 4400 à 7699 sont très généraux, vous pouvez les réutiliser. Les modules 1000 à 1559 sont spécifiques à l'application.

La puissance de
MEM/DOS

pour ceux qui veulent éviter la programmation :

MEMOBASE

gestionnaire de fiches

- Consultation sélective sur critères multiples et sur plusieurs fichiers avec écritures détaillées possibles, étiquettes...
- L'écran de saisie est dessiné automatiquement tout en restant modifiable
- Chaque fiche de base est constituée d'une page-écran en 40 ou 80 colonnes

Possibilité d'étendre la fiche de base à plusieurs pages-écran

- Fonctionne avec ou sans carte MEM/DOS

MEMOBASE s'intègre à tous les progiciels développés par MEMSOFT :
MEM/PLOT, MEMTEXT, MEMSOFT GESTION...

PROGRAMME PRINCIPAL

Fonction :

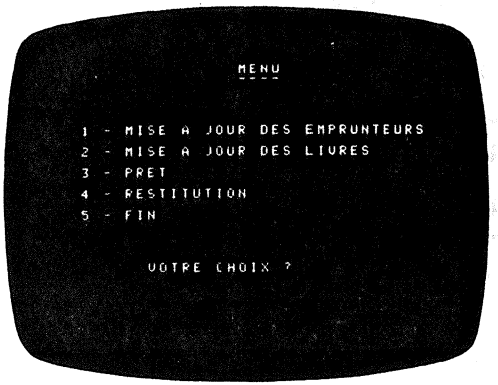
Ouvrir tous les masques et fichiers puis boucler en attente d'une demande et lancer les modules demandés.

Algorithme :

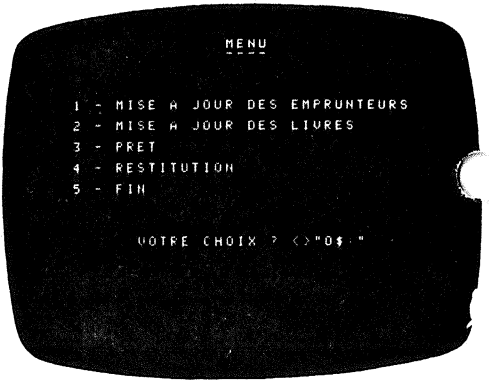
. ouvrir masques et fichiers.
. **itérer**
    afficher le menu principal
    lire le choix de l'utilisateur
. **sortir si** WS › 0 OU SI le choix est "fini"
    appeler le sous-programme correspondant au choix
    de l'utilisateur après avoir éventuellement précisé les paramètres effectifs.
. **fin itérer**

Masque utilisé :

MENU (identificateur temporaire M)



tel qu'il apparaît lors de l'utilisation du programme



tel qu'il a été créé

Liste du programme :

```
1000 LET "# CLEAR-$"
1010 LET "# OPEN,A,MASK,CLE-EMPRUNTEUR"
1011 LET "# OPEN,B,MASK,ART-EMPRUNTEUR"
1012 LET "# OPEN,C,MASK,CLE-LIVRE"
1013 LET "# OPEN,D,MASK,ART-LIVRE"
1014 LET "# OPEN,E,MASK,PRET"
1015 LET "# OPEN,F,FILE,LIVRES"
1016 LET "# OPEN,G,FILE,EMPRUNTEURS"
1017 LET "# OPEN,M,MASK,MENU"
1018 LET "# OPEN,N,MASK,MOUVEMENT"
1019 LET "# OPEN,X,MASK,ATTENTE"
1100 LET "PRINT-M":LET "INPUT-M": IF WS> 0 OR O$ = "5" THEN HOME : END
1110 IF O$ = "1" THEN C$ = "A":M$ = "B":F$ = "G":P$ = "P.EMPRUNT": CALL
FN "GESTION"(C$,M$,F$,P$)
1120 IF O$ = "2" THEN C$ = "C":M$ = "D":F$ = "F":P$ = "P.LIVRE": CALL FN
"GESTION"(C$,M$,F$,P$)
1130 IF O$ = "3" THEN CALL FN "PRET"
1140 IF O$ = "4" THEN CALL FN "RESTITUTION"
1150 GOTO 1100
```

Remarques :

R.1 : Le module GESTION utilise également une variable O$, mais déclarée localement ; MEM/DOS ne la confond donc pas avec la variable O$ du programme principal.

R.2 : Les paramètres effectifs de l'appel du sous-programme GESTION ont les mêmes identificateurs que les paramètres formels (cf la définition de GESTION) ; cela n'est ni obligatoire ni interdit.

R.3 : La fenêtre correspondant à O$ a une largeur 2, alors qu'il n'y a qu'un seul caractère à saisir : cela oblige en quelque sorte l'utilisateur à confirmer son choix par un RETURN.

SOUS-PROGRAMME GESTION

Fonction :

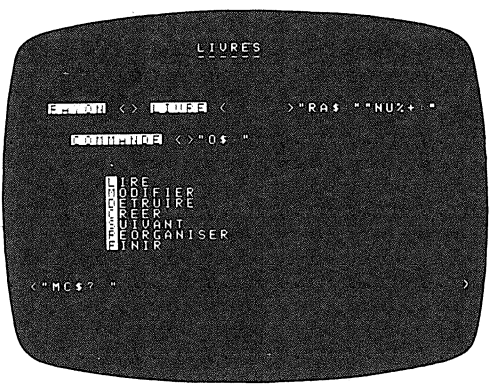
Gérer un fichier : c'est la valeur des paramètres effectifs qui, au moment de l'appel, précisera quel fichier. Cette gestion se fait ensuite par le lancement des modules correspondants.

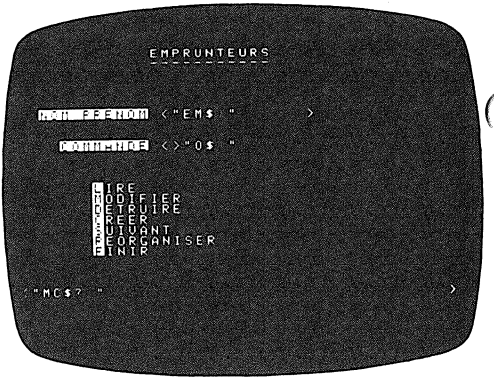
Algorithme :

. **itérer**
afficher le masque dont le nom est contenu dans
le paramètre C$: selon le cas, ce sera le masque
CLE-LIVRE ou CLE-EMPRUNTEUR.
lire l'action demandée et la clé du livre
ou de l'emprunteur.
. **sortir si** abandon en cours de saisie
ou si l'action demandée est "FIN"
lancer le sous-programme correspondant à l'action
demandée ou afficher le message "ordre éronné".
. **fin itérer**

Masques utilisés :

CLE-LIVRE (identificateur temporaire C) OU CLE-EMPRUNTEUR (identificateur temporaire A).





Les masques sont désormais uniquement visualisés sous la forme qu'ils ont à la création.

Liste du programme :

```
4400 DEF  FN "GESTION"(C$,M$,F$,P$/O$,MC$,MA$)
4410 MC$ = "":MA$ = ""
4420 LET "PRINT-" + C$:MC$ = "": LET "INPUT-" + C$
4430 IF O$ = "F" THEN 4499
4440 IF WS THEN 4499
4450 IF O$ = "L" THEN CALL FN "LIRE"(F$,M$): GOTO 4420
4460 IF O$ = "C" THEN CALL FN "CREER"(F$,M$,P$): GOTO 4420
4470 IF O$ = "M" THEN CALL FN "MODIFIER"(F$,M$): GOTO 4420
4480 IF O$ = "D" THEN CALL FN "DETRUIRE"(F$,M$): GOTO 4420
4490 IF O$ = "S" THEN CALL FN "SUIVANT"(F$,M$): GOTO 4420
4493 IF O$ = "R" THEN CALL FN "REORG"(F$): GOTO 4420
4495 MC$ = "ORDRE ERRONE": GOTO 4420
4499 END   FN
```

Remarques :

R.1 : Les deux premières variables du masque LIVRES sont décrites dans l'ordre après les deux fenêtres correspondantes.

R.2 : Le + de la variable NU%+: est une possibilité offerte par MEM/DOS dont nous n'avons pas encore parlé : il signale que l'entier entré doit être positif. Cela constitue une vérification qui peut détecter des erreurs de frappe.

R.3 : Les modules LIRE, CREER, ... sont décrits après les modules PRET et RESTITUTION.

SOUS-PROGRAMME PRET

Fonction :

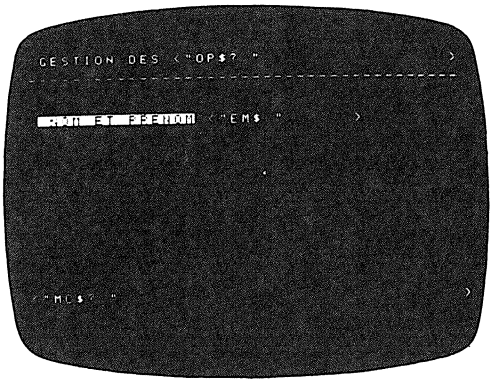
Enregistrer un prêt après avoir vérifier que celui-ci est possible : livre existant et en bibliothèque, emprunteur n'ayant pas de prêt en cours.

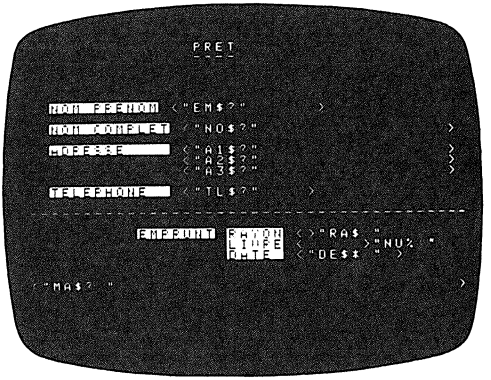
Algorithme :

. **itérer**
  afficher le masque "MOUVEMENT" et le faire
  remplir pour connaître l'emprunteur
  SI abandon en cours de saisie ALORS sortir du module FINSI
  lire sur le fichier " EMPRUNTEURS "la fiche correspondante
. **sortir si** le livre existe et n'est pas sorti
. **fin itérer**
. enregistrer le prêt en mettant à jour les fiches correspondant à l'emprunteur et au livre.

Masques utilisés :

MOUVEMENT (identificateur temporaire N) et PRET (identificateur temporaire E).





Liste du sous-programme :

```
1390 DEF   FN "PRET"
1400 MA$ = "":OP$ = "EMPRUNTS": LET "PRINT-N": LET "INPUT-N":MC$ = "": IF
WS THEN 1499
1410 LET "READ-G": IF WS THEN MC$ = "INEXISTANT": GOTO 1400
1420 IF NU% > 0 OR RA$ > ""OR DE$ > "" THEN MC$ = "A DEJA EMPRUNTE":
GOTO 1400
1430 LET "PRINT-E": LET "INPUT-E":MA$ = "": IF WS THEN: GOTO 1499
1431 EE$ = EM$
1440 LET "READ-F": IF WS THEN MA$ = "LIVRE NON CATALOGUE": GOTO 1430
1450 IF EM$ > "" THEN EM$ = EE$:MA$ ="LIVRE DEJA EMPRUNTE": GOTO 1430
1460 EM$ = EE$: LET "UPDATE-F": LET "UPDATE-G"
1499 END   FN
```

Remarques :

R.1 : Le traitement différencie bien la lecture des renseignements concernant l'emprunteur de ceux qui concernent le livre grâce à deux boucles successives. On ne sort "normalement" de ces boucles que lorsqu'on a saisi un élément (emprunteur ou livre) susceptible d'intervenir dans un prêt, mais grâce à la possibilité de sortir d'un module par un CTRL-A en cours de saisie de masque, l'utilisateur peut, à tout moment, faire avorter une transaction.

R.2 : Ce sous-programme n'a pas de paramètres : on voit clairement l'avantage qu'il y a à découper en modules réalisant chacun une fonction.

SOUS-PROGRAMME RESTITUTION

Fonction :

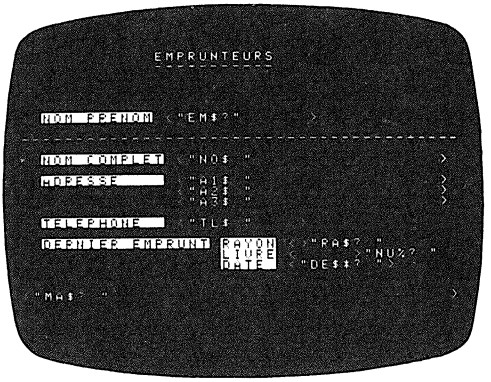
Enregistrer une restitution après avoir vérifié que le livre rendu appartient bien à la bibliothèque et que l'emprunteur l'avait bien sorti.

Algorithme :

. **itérer**
afficher le masque "MOUVEMENT" et le faire remplir pour connaître l'emprunteur
SI abandon en cours de saisie ALORS sortir du module FINSI.
lire sur le fichier "EMPRUNTEURS" la fiche correspondante
. **sortir si** l'emprunteur existe
ET SI et il a bien emprunté un livre
ET SI c'est bien le bon emprunteur
. **fin itérer**
mettre à jour le fichier "livres"
mettre à jour le fichier "emprunteurs"

Masques utilisés :

MOUVEMENT (identificateur temporaire N, cf fiche PRET) et
ART-EMPRUNTEUR (identificateur temporaire B)



Liste du sous-programme :

```
1500 DEF   FN "RESTITUTION"
1505 MA$ = "":OP$ = "RESTITUTION": LET "PRINT-N": LET "INPUT-N":MC$ =
"": IF WS THEN 1599
1510 LET "READ-G": IF WS THEN MC$ = "INEXISTANT": GOTO 1505
1520 IF RA$ = "" AND NU% = 0 THEN MC$ = "PAS DE LIVRE": GOTO 1505
1525 EE$ = EM$:DD$ = DE$
1530 LET "READ-F": IF WS THEN NU% = 0:RA$ = "":DE$ = "": LET "UPDATE-G":
GOTO 1599
1540 IF EM$ <> EE$ OR DD$ <> DE$ THEN MC$ = "ERREUR": GOTO 1505
1545 MA$ = "EST CE BIEN LUI ?": LET "PRINT-B": CALL   FN "ATTENTE"(X$)
1548 IF X$ <> "O" THEN 1505
1550 EM$ = EE$:DE$ = DD$: LET "UPDATE-F"
1560 EM$ = EE$:RA$ = "":NU% = 0: LET "UPDATE-G"
1599 END   FN
```

Remarque : Le sous-programme ATTENTE ne fait que lire au clavier la valeur du paramètre. Il est expliqué plus loin.

SOUS-PROGRAMME LIRE

Fonction :

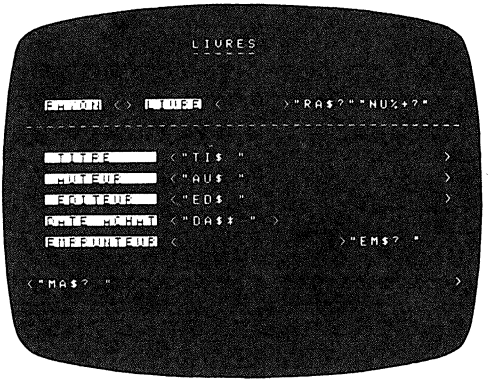
Lire une fiche, l'afficher et attendre que l'utilisateur signale qu'il l'a lue en frappant RETURN ; le nom du fichier et un paramètre, de même que le masque qui sert à afficher.

Algorithme :

. lire la fiche demandée
. **si** elle n'existe pas
. **alors** émettre un message
. **sinon** l'afficher en demandant d'indiquer la fin de lecture par un RETURN
. **finsi**

Masques utilisés :

ART-EMPRUNTEUR (identificateur temporaire B cf module RESTITUTION) ou ART-LIVRE (identificateur temporaire D)



Liste du sous-programme :

```
5000 DEF   FN "LIRE"(F$,M$/X$)
5010 LET "READ-" + F$
5020 IF WS THEN MC$ = "INEXISTANT": GOTO 5099
5030 MA$ = "RETURN=VU": LET "PRINT-" + M$
5040 CALL  FN "ATTENTE"(X$)
5099 END   FN
```

Remarque :

Le message "inexistant" n'est pas affiché lors de l'exécution du sous-programme ; il est renvoyé au programme appelant, qui est en fait ce qu'il veut !

SOUS-PROGRAMME SUIVANT

Fonction :

Lire en séquence une ou plusieurs fiches, avec affichage. Le nom du fichier à parcourir est un paramètre. La première fiche lue correspond à la valeur "actuelle" de la clé ; on ne l'affiche pas puisque l'utilisateur veut la suivante !

Algorithme :

Lire la fiche correspondant à la clé.
. **Itérer**
  lire la suivante
. **sortir si** on est au bout du fichier
  l'afficher sur le masque dont le nom est également
  passé en paramètre
  attendre que l'utilisateur l'ait lue et ait fait
  savoir s'il veut continuer
. **sortir si** l'utilisateur le veut
. **fin itérer**

Masques utilisés :

ART-EMPRUNTEUR ou ART-LIVRE

Liste du sous-programme :

```
6000 DEF   FN "SUIVANT"(F$,M$/X$)
6010 LET "READ-" + F$
6015 LET "NEXT-" + F$
6020 IF WS THEN MC$ = "FIN DE FICHIER": GOTO 6099
6030 MA$ = "RETURN=VU,'S'=SUIVANT": LET "PRINT-" + M$
6040 CALL  FN "ATTENTE"(X$)
6050 IF X$ < > "S" AND X$ > "" THEN 6040
6060 IF X$ = "S" THEN 6015
6099 END   FN
```

Remarques :

R.1 : Logiquement cette option ne devrait être demandée par l'utilisateur qu'après avoir lu au moins une fiche ; même si ce n'est pas le cas, ce sous-programme fonctionne car BASIC initialise à zéro ou à blanc toutes les variables : la lecture commencera donc à la première fiche.

R.2 : S'il n'y a pas de fiche correspondant à la clé, la commande READ positionne le pointeur du fichier sur un enregistrement "fictif" et la commande NEXT positionnera le pointeur sur le premier enregistrement réel suivant. On peut donc facilement retrouver DUPOND ou DUPONT en demandant le suivant de DUPON.

SOUS-PROGRAMME DETRUIRE

Fonction :

effacer une fiche.

Algorithme :

. lire la fiche
. **si** elle n'existe pas **alors** sortir du module **finsi**
. l'afficher pour accord de destruction
. **si** l'utilisateur est d'accord
. **alors** la détruire
. **si non** émettre le message "abandon"
. **finsi**

Liste du sous-programme :

```
6200 DEF   FN "DETRUIRE"(F$,M$/X$)
6210 CALL FN "LIRE"(F$,M$)
6230 MA$ = "OK POUR DETRUIRE (O/N) ?": LET "OUTPUT-" +   M$

6240 CALL FN "ATTENTE"(X$)
6250 IF X$ <> "O" THEN MC$ = "ABANDON": GOTO 6299
6260 LET "DELET-" + F$
6299 END   FN
```

SOUS-PROGRAMME MODIFIER

Fonction :

Modifier une fiche

Algorithme :

. lire la fiche
. **si** elle n'existe pas **alors** sortir du module **finsi**
. l'afficher
. saisir les modifications
. **si** abandon en cours de saisie **alors** sortir du module **finsi**
. enregistrer les modifications dans le fichier

Liste du sous-programme :

```
7200 DEF   FN "MODIFIER"(F$,M$/X$)
7210 CALL FN "LIRE"(F$,M$)
7220 IF WS THEN 7299
7230 MA$ = "MODIFICATION": LET "OUTPUT-" + M$ + "-OUTPUT": LET "INPUT-
" + M$: IF WS THEN MC$ = "ABANDON": GOTO 7299
7240 LET "UPDATE-" + F$
7299 END   FN
```

SOUS-PROGRAMME CREER

Fonction :

Créer, à condition qu'elle n'existe pas, une fiche dont la clé et précisée dans le programme appelant. Comme le nom du fichier est un paramètre, et qu'il faut faire annuler automatiquement certaines variables, on exécute un sous-programme P.EMPRUNT ou P.LIVRE dont le nom est transmis au sous-programme CREER grâce au paramètre P$.

Algorithme :

. essayer de lire dans le fichier.
. **si** la liste existe déjà **alors** sortir du module **finsi**
. appeler le SP qui précise les variables nulles ou vides
. afficher le masque correspondant à la fiche à créer et la saisir
. **si** abandon en cours de saisie **alors** sortir du module **finsi**
. écrire la fiche dans le fichier

Liste du sous-programme :

```
7300 DEF  FN "CREER"(F$,M$,P$)
7310 LET "READ-" + F$: IF WS = 0 THEN MC$ = "EXISTANT": GOTO 7399
7320 IF P$ > "" THEN CALL FN "" + P$
7325 MA$ = "ENTREZ L'ARTICLE": LET "PRINT-" + M$: LET "INPUT-" + M$: IF WS
THEN MC$ = "ABANDON": GOTO 7399
7330 LET "WRITE-" + F$
7399 END  FN
```

LES SOUS-PROGRAMMES P.EMPRUNT ET P.LIVRE

Fonction :

Annuler ou mettre à "blanc" certaines variables.

Listes des sous-programmes :

```
1200 DEF  FN "P.EMPRUNT"
1210 NU% = 0:RA$ = "":DE$ = ""
1220 END  FN
1300 DEF  FN "P.LIVRE"
1310 EM$ = ""
1320 END  FN
```

SOUS-PROGRAMME REORG

Fonction :

Réorganiser un fichier lorsque, à la suite de nombreuses insertions, le temps d'accès s'allonge. Cette fonction est assurée par une commande MEM/DOS.

Liste du sous-programme :

```
7400 DEF  FN "REORG"(F$) IF WS
7410 LET "# REORG-" + F$ : PRINT
7499 END  FN
```

SOUS-PROGRAMME ATTENTE

Fonction :

Lire un caractère au clavier par projection du masque ATTENTE, dont le numéro logique est X.

Liste du sous-programme :

```
7600 DEF  FN "ATTENTE"(X$)
7610 X$ = "": LET "INPUT-X"
7699 END  FN
```

Masque : ATTENTE (identificateur temporaire : X)

On ne le montre pas : vous pouvez sans problème le visualiser vous-même ... Ne vous étonnez pas : il est presque vide : ce n'est pas un gag !

ANNEXE I : LES MESSAGES D'ERREURS

1 - ERREURS RECUPERABLES

Les erreurs de ce type n'arrêtent pas le déroulement du programme. Vous devez donc, lorsque vous écrivez votre programme, tester si une telle erreur est arrivée et agir en conséquence. La variable WS est réservée à MEM/DOS qui y place une espèce de compte-rendu de transaction après l'exécution d'une commande. La valeur 0 indique que tout s'est bien passé ; une valeur différente de 0 indique une difficulté :

1 : erreur lors de l'utilisation d'un masque : il s'agit en fait d'une "erreur" volontaire de l'utilisateur du masque qui tape CTRL-A pour indiquer qu'il abandonne la saisie.

10 : essai d'utilisation d'un objet (fichier, article de fichier, masque) qui n'existe pas sur la disquette courante.

30 : essai de création d'un objet alors qu'il existe déjà un objet du même type ayant le même identificateur sur la disquette courante.

255 : essai de lecture d'un enregistrement alors qu'on est en fin de fichier.

L'utilisation de cette variable WS est simple : après chaque commande, vous devez tester sa valeur et agir en conséquence :

```
100 IF WS > 0 THEN : GOTO 1000
110 REM : TOUT VA BIEN
.....
.....
.....

1000 REM : IL Y A UN PROBLEME
.....
.....
.....
```

Remarque :

Lorsqu'on utilise MEM/DOS en mode conversationnel direct, c'est-à-dire lorsqu'on tape une commande en dehors de tout programme, cette commande s'exécute et WS contient le même compte-rendu de transaction. Cependant, pour vous éviter de tester à chaque fois la valeur de WS, MEM/DOS affiche automatiquement le message DIRECT ERROR si WS > 0. C'est bien agréable, mais le message affiché risque d'être mal interprété : il ne veut pas dire que cette commande est interdite en mode direct !!

2 ERREURS IRRECUPERABLES OU FATALES

Lorsqu'une telle erreur arrive, il n'y a rien à faire, on est prévenu tout de suite : d'une part le programme s'arrête, et d'autre part un message s'affiche sur l'écran. Il suffit donc de savoir interpréter ce message et de trouver la cause de l'erreur ! Voici la liste des messages :

SYNTAX ERROR
La syntaxe de la commande n'est pas bonne

ILLEGAL QUANTITY ERROR
C'est l'identificateur temporaire de la commande qui est en cause : soit vous voulez l'affecter alors qu'il est déjà affecté, soit vous l'utilisez alors qu'il n'a pas été affecté à un objet.

OUT OF MEMORY ERROR
C'est un problème de place en mémoire centrale : il n'y a plus de buffers libres. Cela ne risque pas beaucoup d'arriver avec les petites applications programmables avec BABY MEM /DOS.

TA ERROR
Cette erreur qui signifie que la disquette est pleine risque de se produire rapidement sous BABY MEM/DOS car ici, la disquette est en fait la "mini-disquette" de 12 Koctets. Dans le message, TA est l'abréviation de TRACK ALLOCATION qui, en anglais, veut dire "allocation d'une piste".

DATA ERROR
Cette erreur signale une difficulté de lecture ou d'écriture sur la disquette. Sous BABY MEM/DOS où la mini-disquette est en mémoire centrale, cela risque peu de vous arriver !

Remarque :

Ces erreurs fatales peuvent en fait ne pas l'être si vous utilisez l'instruction ON ERROR ... mais cela, c'est du BASIC classique.

ANNEXE II : L'UTILISATION DES "DATES" ET DES "ENTIERS COURTS".

MEM/DOS reconnaît deux types de variables qui n'existent pas en Basic et qui servent l'une à représenter des dates et l'autre des entiers positifs inférieurs à 256.

L'utilisation des variables de type "entier court" permet de gagner de la place dans les fichiers : si on est certain qu'une variable sera toujours comprise entre 0 et 255, on peut se contenter d'un octet pour la mémoriser, alors qu'il en faut deux pour mémoriser une variable entière normale en Basic.

Pour préciser qu'une variable entière NO% par exemple ne doit occuper qu'un octet, il suffit, dans la description du fichier, de la faire suivre du caractère/:

```
LET "> @ RA%=NM$,NO%/,SC"
```

Remarques :

R1 - Le pointeur d'un fichier relatif ne peut pas être un entier court, mais on peut utiliser un entier court dans une clé d'un fichier indexé.

R2 - Un entier court peut également figurer dans un masque.

R3 - C'est uniquement dans les descriptions de fichiers ou les masques que la variable NO% doit être suivie de/; dans les instructions Basic du programme, cette variable continue de s'appeller NO%.

R4 - Si vous essayez d'affecter à une variable de ce type une valeur hors de l'intervalle 0-255, c'est le reste modulo 256 qui sera affecté : cela veut dire simplement que la valeur 300 par exemple sera réduite à 44.

L'utilisation des variables de type "date" répond à plusieurs objectifs : gagner de la place (MEM/DOS parvient à les coder sur deux octets), mais aussi améliorer la présentation et permettre certaines vérifications. En basic normal, si on veut qu'une suite de six chiffres représente une date, il faut utiliser une variable chaîne (car les entiers sont limités à 32767) et 12 mars 1985 sera codé 120385. Sous MEM/DOS, une variable de type date est représentée dans un fichier par un identificateur de chaîne suivi du caractère *

```
LET "> @ RA%=NM$,NA$*,SC"
```

MEM/DOS affiche la valeur d'une date en séparant les jours, les mois et les années : 12/03/85, mais quand vous entrez la valeur d'une date, vous pouvez entrer les six chiffres à la suite, ou utiliser n'importe quel séparateur.

MEM/DOS a permis l'utilisation de dates impossibles (12/00/84 par exemple) pour deux raisons : lorsqu'une date est inconnue, il faut pourtant pouvoir lui affecter une valeur, et dans un autre domaine, la comptabilité, il est parfois utile de disposer d'un mois 0 ou d'un jour 0 pour faire des cumuls.

Si une date est saisie dans un masque, MEM/DOS fait certaines vérifications et rejette les dates non conformes. Par contre, si vous entrez une date dans une chaîne grâce à un INPUT et que cette chaîne sert de date dans un fichier, vous pouvez avoir des surprises si le contenu de la chaîne ne correspond à aucune date réelle.

ANNEXE III : LES FONCTIONS DU MENU DE DEMARRAGE

1. BASIC

On dispose du Basic... sous MEM/DOS (cf. page 7, paragraphe 1.4).

2. GESTION DE MASQUES

Cette option permet d'effectuer des opérations sur des masques. L'écran projeté est évidemment un masque et vous précisez ce que vous vous voulez faire dans la fenêtre correspondante. Comme d'habitude, on passe d'une fenêtre à la suivante par un RETURN.

3. GESTION DE FICHIERS

Cette option permet d'exécuter toutes les opérations sur un fichier quelconque... exemple, le fichier DEMO (de la disquette MEM/DOS) qui permet de gérer des individus (nom, adresse, date de naissance).

4. DEMO MASQUE

Ce dessin animé est fabriqué à partir d'un masque. Il vous montre une des possibilités d'utilisation des masques... même si ce n'est pas celle couramment utilisée dans les problèmes de gestion.

Pour quitter, taper CTRL C.

5. DEMO FICHIER

Cette option permet de travailler sur le fichier DEMO (nom, adresse, date de naissance).

6. COPIE DE DISQUE

Cette option effectue une copie de disquette. Ne pas oublier que les numéros des lecteurs (ou "drive") sont ceux de MEM/DOS (0, 1...).

7. CATALOGUE

C'est l'affichage à l'écran du catalogue, c'est-à-dire de la liste des masques, fichiers, programmes... on ne peut pas avoir de catalogue partiel.

FORMATTAGE DISQUE

C'est l'initialisation d'une disquette par le système MEM / DOS. A la question "Avec BOOT (O/N) ?", la réponse - NON - laissera plus de place disponible à l'utilisateur mais cette disquette ne démarrera pas.

9. COPIE AUTOMATIQUE

On peut copier tout ou une partie d'une disquette (selon la réponse NON ou OUI à l'interrogation "Voulez-vous une sélection ?").

0. TESTS

Cette option permet d'effectuer des tests de deux sortes :

- 1, 2, 3, : tests sur le contenu de la disquette (l'option 4 donne des informations sur ces trois tests).
- 5 : vérification matérielle du bon état de la carte MEM/DOS (contrôles de parité) : visualisation schématisée de la carte MEM/DOS avec ses cinq ROMS et des nombres sur chacune d'elles.
  En cas d'erreur, affichage d'un masque sur la ROM correspondante.

INDEX

MANUEL D'UTILISATION

MEMS FT

PROGICIELS

DEMAIN C'EST AUJOURD'HUI

SOMMAIRE

IMPORTANT... IMPORTANT... IMPORTANT... IMPORTA

REPORTEZ VOUS AUX SPECIFICITES APPLE IIc

Le système MEM/DOS est entièrement dans les roms de la carte. Néanmoins, il doit, pour fonctionner, utiliser des programmes spécifiques aux périphériques employés.

Ces programmes sont appelés handlers et sont généralement fournis par le constructeur du disque.

La version que nous vous fournissons, permet d'utiliser les lecteurs de disquettes, Disk II Apple en 140K en utilisant le handler approprié fourni en standard avec les lecteurs de disquettes et appelé RWTS dans la documentation Apple.

Pour transférer ce handler sur la disquette MEM/DOS vous devez utiliser la disquette MASTER DOS 3.3, fournie avec le lecteur pour initialiser la disquette MEM/DOS

La procédure est la suivante :

- Démarrez avec la disquette MEM/DOS

- Lorsque le message "BOOT DOS 3.3 & RUN HELLO sur ce disque" apparaît, mettez votre disquette MASTER 3.3 à la place de la disquette MEM/DOS.

- Lorsque vous avez la main en basic, remettez la disquette MEM/DOS et faites RUN HELLO.

- Un menu apparaît à l'écran.

Le choix 1 vous permettra de mettre en place le boot du MEM/DOS et la disquette démarrera ensuite directement.

Cette opération n'est donc à effectuer qu'une seule fois.

NOTE : Pour utiliser la carte MEM/DOS avec d'autres périphériques disque, demandez à votre revendeur habituel ou au fournisseur du disque, si le handler adapté MEM/DOS existe.

(Apple et Disk II sont des marques déposées d'Apple Computer Inc.

CHAPITRE I DESCRIPTION GENERALE DU SYSTEME

Le MEM/DOS est un outil puissant d'aide à la réalisation de logiciels de gestion. Il comprend deux parties principales permettant une programmation rapide, en particulier dans les applioations en temps réel. Cette version sur silicium est entièrement compatible avec les programmes MEM/DOS écrits sous la version précédente. Néanmoins, nous attirons l'attention du lecteur sur le fait que cette nouvelle version possède de nouvelles possibilités. Il faut noter qu'il y a une légère modification quant à l'utilisation de la page 3 de votre APPLE.
ATTENTION : les fonctions SCROLL UP et SCROLL DOWN ont vu leurs adresses modifiées (voir chapitre X : Les utilitaires).

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

1 - 1 LE DISK OPERATING SYSTEM

Il comprend les options suivantes :

- fichiers séquentiels
- fichiers relatifs
- fichiers à accés par clés
- fichiers multiclés

Quelle que soit l'option choisie, aucune modification de variables n'est nécessaire, celle-ci étant faite à la création du fichier, et enregistrée dans un DICTIONNAIRE associé. Le DOS comprend en outre la notion de SOUS-ARTICLES. Les enregistrements sont donc de taille variable.
La gestion du disque est entièrement dynamique et aucun dimensionnement de fichier ou d'enregistrement n'est nécessaire.

1 - 2 LA GESTION DES MASQUES DE SAISIE ET D'IMPRESSION

Cet utilitaire permet de résoudre de façon rapide et souple tous les problèmes de saisie de données et d'impression. Tous les contrôles de validité sont efectués lors de l'utilisation. Les différents types de variables que l'on utilise sont compatibles avec ceux de la gestion de fichier. Un masque est composé de deux parties :

- un texte dont les caractères répétés sont codés pour gagner de la place, aussi bien sur disque qu'en mémoire centrale au moment de l'utilisation.
- un ensemble de fenêtres de saisie. Chaque fenêtre est associée à une variable BASIC et à un ensemble de contrôles à effectuer.

Pour faciliter la saisie, un certain nombre de touches de fonction ont été créées, facilitant la gestion de l'écran.

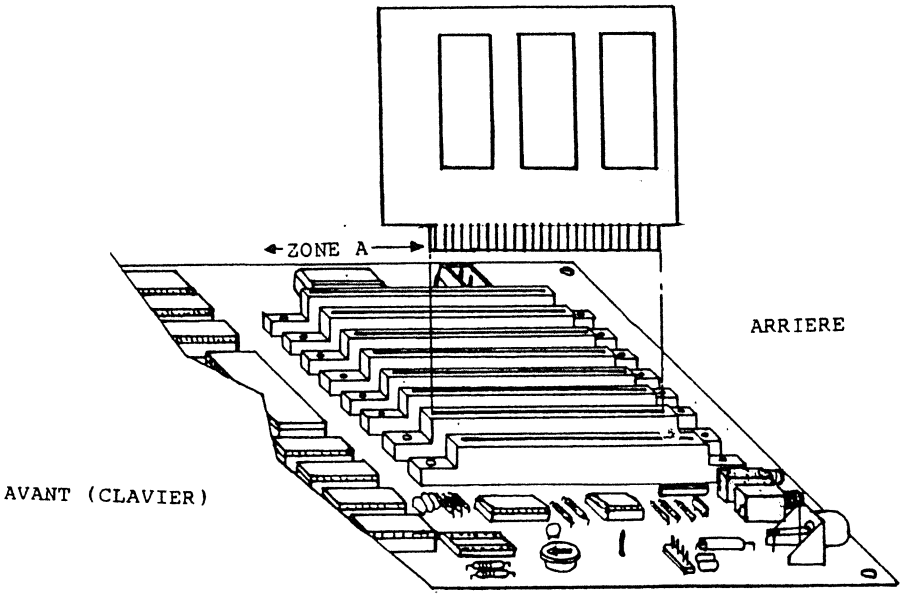
CHAPITRE II MISE EN ROUTE DU MEM/DOS

REPORTEZ VOUS AUX SPECIFICITES APPLE IIc

2 - 1 MISE EN PLACE D'UNE CARTE MEM/DOS

Avant toute manipulation, éteindre votre appareil en basculant votre interrupteur marche - arrêt sur la position arrêt et retirer votre cordon secteur de la prise de courant.

Ouvrez votre appareil. Sur le bac inférieur, vous voyez au fond à droite, un ensemble de huit connecteurs. La carte MEM/DOS est dissymétrique par rapport à l'axe du connecteur.



La zone A, c'est à dire le coté où la carte est plus longue est toujours dirigé vers l'avant de votre appareil. En regardant la plaque microprocesseur, on voit immédiatement que les connecteurs sur lesquels vous allez fixer votre carte sont numérotés de 0 à 7, le slot 0 étant le plus à gauche, le slot 7 étant le plus à droite. Le n° est imprimé sur le bord de le carte microprocesseur. Ces connecteurs sont aussi appelés slots. Choisissez un des slots entre le numéro 1 et le numéro 7. Fixer la carte dans le slot choisi (Par ex., le 3).

ATTENTION : Ne jamais toucher les surfaces dorées des connecteurs. Ces connecteurs sont de haute qualité et de haute précision. Tout dépôt gras risque d'en compromettre sérieusement le fonctionnement.
Au cas où un connecteur serait encrassé, nettoyer doucement avec un chiffon imbibé d'alcool à bruler ou avec un produit spécial contact.

Ne jamais exercer d'effort susceptible de déformer les contacts.

Ne mettez jamais votre carte en flexion, vous risqueriez de couper les pistes.

Vous vous assurerez ainsi une installation parfaite et fiable.

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

2 - 2 MISE EN ROUTE DU MEM/DOS SUR DISQUETTE APPLE

Pour mettre en oeuvre le MEM/DOS , introduisez la disquette système dans le lecteur " Ø " (le DRIVE 1 du controleur sur le port 6) et fermez la porte du lecteur.
Si votre appareil est muni d'une ROM AUTOSTART, mettez la tension sur votre micro-ordinateur et le système se chargera automatiquement. Dans le cas contraire, faites :

```
RESET
6 CTRL/P      RETURN
```

Le système se chargera de lui-même et le message suivant apparait :

```
MEM/DOS     - V4
MICRO INFORMATIQUE SERVICE
P. LAFFITTE
P. NESNIDAL
```

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

2 - 3 FORMATTAGE D'UNE DISQUETTE

La disquette système qui vous est fournie avec le manuel est une disquette spéciale qui contient les utilitaires du système MEM/DOS . Si vous avez deux drives, nous vous conseillons tout d'abord de faire une copie de cette disquette. Pour arriver à recréer une nouvelle disquette, il faut prendre une disquette vierge et pré-inscrire dessus les valeurs nécessaires pour que le système puisse retrouver les pistes et les secteurs. Cette opération s'appelle le formattage.

formattage d'une disquette de type système :

Pour cela, utiliser " l'utilitaire BOOT ".
Voir chapitre XI : utilisation de ce programme.

formattage d'une disquette de type fichier :

Cette disquette ne permet pas de démarrer le système. Utiliser l'instruction : LET"# FORMAT,(numéro du drive)".

Cette instruction de formattage est valable quel que soit le disque ou la disquette implantée correctement sous MEM/DOS . Il n'en va pas de même pour la création de disquettes permettant de démarrer le système. Pour rendre une disquette " autostart " se référer à la notice d'implantation fournie en général par le constructeur de l'unité de disques.

L'utilitaire BOOT permet d'obtenir une disquette qui pourra faire démarrer le MEM/DOS , mais qui ne contiendra aucun programme . Si vous voulez obtenir une disquette avec les utilitaires MASTER dessus, il vous suffit de faire une copie de la disquette MASTER fournie par M.I.S.

CHAPITRE III LE DISK OPERATING SYSTEM

3 - 1 LES FICHIERS

3 - 1 - a Structure des enregistrements

La structure des fichiers du MEM/DOS a été conçuede façon à simplifier la programmation, et surtout à limiter le temps d'exécution du programme BASIC.

Un fichier est un ensemble d'enregistrements. Le nombre d'enregistrements par fichier ne peut dépasser 64 000. La taille du fichier est gérée dynamiquement et il est inutile de définir celle-ci à la création d'un fichier.

Chaque enregistrement d'un fichier est un ensemble de variables BASIC.
Ces variables peuvent être de tous types (à quelques restrictions près pour les clés des fichiers à accès par clé - voir fichier séquentiel indexé chapitre 3.- 2 -c -)

Rappelons les différents types de variables du BASIC :

- les flottants (5 octets) notes XY
- les entiers (2 octets) notes XY%
- les chaines de caractères de longueur 0 à 255 notées XY$

avec les conventions suivantes :

- le nom comporte 1 ou2 caractères X, Y
- le premier X est une lettre
- le second Y (facultatif) est une lettre ou un chiffre

De plus, des tableaux peuvent être définis pour chacun de ces types de variables. Les dimensions et le nombre de dimensions de ces tableaux ne sont limités que par la taille de la mémoire centrale de l'ordinateur.

Exemple : TA$(12,2,1)

Les indices des tableaux commencent à 0.

Dans le DOS, 2 nouveaux types de variables ont été créés :

- les dates notées XY$● dans les ordres MEM/DOS
- les binaires notés XY%/ dans les ordres MEM/DOS

Afin de réduire la place occupée par un enregistrement sur le disque, les valeurs numériques sont codées en binaire, les chaines de caractères en format variable et les tableaux sous forme de matrices creuses.
Il en résulte que si un tableau ne comporte que3 éléments non nuls, seuls ceux-ci seront sauvés sur le disque.

3 - 1 - b Contrôles des erreurs

- erreurs recouvrables

Après chaque ordre du DOS, il est possible de vérifier que l'ordre a bien été exécuté. Pour cela, un octet est réservé (189 sur ITT/APPLE). Cette variable sera appelée WS dans le reste de cette documentation.

Si WS = Ø : tout s'est bien passé.
Sinon, une erreur est apparue lors de l'exécution de l'ordre. Dans ce cas, WS contient le code de l'erreur.

Une erreur de ce type n'interrompt pas le programme.

En général, il n'y a pour chaque ordre qu'un seul type d'erreur recouvrable. Il suffit donc de vérifier que le STATUS est nul pour s'assurer que l'ordre a été exécuté normalement.

Exemple : Lecture d'un enregistrement

WS > Ø l'enregistrement n'existe pas.

CODES ERREURS :	
1	erreur sur masque (abandon...)
10	inexistant
20	priorité trop faible (lecture)
30	existant (en création)
255	fin de fichier

En mode direct, pour éviter au programmeur de tester le status après chaque commande, le message " DIRECT ERROR " apparait, si le STATUS est non nul.

- erreurs non recouvrables

Ces erreurs (qui interrompent le programme) peuvent avoir deux causes :

- Erreur de programmation
  - SYNTAX ERROR
    syntaxe d'un ordre impossible.
  - ILLEGAL QUANTITY ERROR
    erreur sur les numéros logiques (déjà utilisé, non ouvert...)

- BAD SUBSCRIPT ERROR
  tentative de lecture d'un tableau dans un enregistrement avec un dimensionnement inférieur à celui utilisé lors de sa création.

- OUT OF MEMORY ERROR
  plus de buffers libres (pour résoudre ce problème, voir le chapitre 8 POINTEURS DU DOS).

- erreurs système

- TA ERROR
  disque plein (Track Allocation Error)

- DATA ERROR
  erreur de lecture ou d'écriture sur le disque. Cette erreur signale un mauvais fonctionnement du support magnétique. Attention, cette erreur peut provenir d'une protection d'écriture du support.

- FUNCTION ERROR
  erreur détectée dans un contrôle dû à une incohérence des paramètres dans un accès disque.

- NG ERROR
  erreur signalant une incohérence des données en mémoire (faire : LET"#C,8" par programme pour supprimer l'erreur).

3 - 1 - c Fermeture de fichiers

Aucune fermeture n'est nécessaire. A la fin de chaque opération, tous les paramètres nécessaires au fonctionnement du DOS et ayant été modifiés, sont sauvés sur le disque.

La fermeture, qui en général réalise cette opération, mais seulement en fin de traitement, est donc inutile.

En cas d'arrêt imprévu du programme, seule l'opération en cours peut être perdue.

Il est néanmoins possible de récupérer la place occupée en mémoire centrale par l'ouverture d'un fichier. Pour cela, utiliser l'ordre CLEAR.

Il est possible de récupérer la place :

- d'un module particulier (masque ou fichier) désigné par son numéro logique. voir 5 - 1 - b)
- de récupérer toute la place . voir 5 - 1 - c)

Après l'ordre CLEAR d'un fichier, il est nécessaire de le ré-ouvrir pour pouvoir accéder à ses enregistrements.

3 - 2 LES DIFFERENTS TYPES DE FICHIERS

3 - 2 - a Fichiers séquentiels relatifs

Ces fichiers peuvent être manipulés aussi bien en séquentiel qu'en relatif, aussi bien en écriture qu'en lecture.
Il n'y a qu'un seul mode d'ouverture qui permet l'ensemble des opérations.
A la création du fichier, on définit une variable entière BASIC qui sera utilisée comme pointeur vers l'enregistrement.

Un enregistrement d'un fichier séquentiel comprend donc :

- le pointeur XX% permettant d'indiquer son numéro d'ordre.
- la liste des variables.

Remarque : le pointeur est une zone fictive de l'enregistrement et est en réalité le résultat d'un calcul.

La liste des ordres possibles est :

- opérations "dites" séquentielles

Write = écriture d'un nouvel enregistrement en fin de fichier.
Le numéro d'ordre de l'enregistrement créé est renvoyé dans le pointeur.

Next = lecture d'un enregistrement en séquentiel.
Permet de lire l'enregistrement suivant.

Borne = limite le fichier en imposant un numéro d'ordre maximum.
Une fois que le numéro atteint cette valeur, une fin de fichier sera signalée.

Xindex = renvoie dans le pointeur le numéro du prochain enregistrement qui sera créé par Write.

- opérations "dites" relatives

Update = modification d'un article préalablement créé par l'ordre Write.

Read = lecture d'un article en fonction de son numéro d'ordre dans le fichier.

Delet = destruction d'un article dont on indique le numéro d'ordre.

Remarque : La tentative de lecture d'un enregistrement détruit n'aboutira pas.
La possibilité de détruire des enregistrements permet de récupérer la place qu'ils occupaient sur le disque.
Néanmoins, la destruction d'un enregistrement ne modifie pas la numérotation des autres.
Les enregistrements détruits ne seront pas lus par l'ordre NEXT.

Tous les ordres peuvent être utilisés dans n'importe quel ordre. En particulier, on pourra accéder directement à un article par son numéro par READ, lire le fichier séquentiellement à partir de cet enregistrement par NEXT. (Par défaut, NEXT après l'ouverture donne le premier enregistrement du fichier)

3 - 2 - b Fichiers séquentiels indexés

Contrairement aux fichiers relatifs (voir 3 - 3 -b), le moyen d'accès à un enregistrement n'est plus le numéro d'ordre, mais une CLE plus complexe.

Cette clé est définie comme un ensemble de variables BASIC, et il devient possible d'accèder à un enregistrement directement par la valeur de sa clé.

Prenons l'exemple d'un fichier de personnes.
La clé pourra être :

NO$ = le nom
DN$:= la date de naissance
PR$ = le prénom

C'est l'ensemble de ces variables qui composera la clé.

Contraintes sur les clés d'un fichier séquentiel indexé :

La clé doit être de longueur fixe. Il en résulte que les tableaux ne pourront pas être utilisés. De plus, pour les chaines de caractères, il faudra indiquer une longueur maximum possible.
Dans l'exemple ci-dessus, il faudrait préciser la longueur admise pour les deux variables caractères.
Par exemple, NO$ 13,PR$ 8

Les opérations possibles sont :

En séquentiel

Next = lecture de l'enregistrement suivant dans l'ordre des clés.

Borne = fixe une borne maximum au fichier

Xtract = permet de sélectionner des enregistrements suivant un critère d'égalité sur une des variables clé.

En accès par clé

Write = écriture d'un nouvel article (refuse les homonymes)

Add = écriture d'un nouvel article (accepte les homonymes)

Read = lecture d'un enregistrement

Update = mise à jour d'un enregistrement

Delet = destruction d'un enregistrement

3 - 2 - c Fichiers multiclés

Les fichiers multiclés du MEM/DOS sont une extension des fichiers séquentiels indéxés. Un fichier multiclé est un fichier comportant plusieurs groupes de clés, chaque groupe est appelé un MOYEN D'ACCES au fichier. Le nombre maximum de moyens d'accès par fichier est 10. Chaque moyen d'accès est un ensemble de variables formant une clé, identique à la clé unique d'un fichier séquentiel indexé.

Contraintes sur les clés d'un fichier multiclé

Les contraintes sur les clés sont les mêmes que les fichiers séquentiels indexés (3 - 2 - b)

Les opérations possibles sont :

Elles sont identiques à celles des fichiers séquentiels indexés (3 - 2 -b). Cependant, elles nécessitent des paramètres supplémentaires qui sont décrits au paragraphe 3 - 3 - d.

Il est impossible de modifier les clés des autres moyens d'accès.
En lecture, les clés des autres moyens d'accès ne sont pas renvoyées.
Si cette information est nécessaire, répéter dans l'enregistrement les variables clés à connaitre.
La réorganisation du fichier se fera sur l'ensemble des moyens d'accès.

3 - 3 LA CREATION DE FICHIERS

3 - 3 - a syntaxe générale

La création d'un fichier se fait en deux étapes :

- définition de l'enregistrement
- création du fichier

Pour définir l'enregistrement, utiliser l'ordre >
La syntaxe est la suivante :

LET " > vc1,vc2,...,vcn = ve1,ve2.....,vep

avec vcl..n : variables composant la clé
et vel..p : variables composant l'enregistrement

Dans la clé, les variables sont de longueur fixe. Les possibilités sont :

```
XX$NNN  ..... variable alpha de longueur NNN   = 255 (exemple AL$25)
XX      ..... flottant                               (exemple FL)
XX%     ..... entier de -32767 à +32768              (exemple EN%)
XX%/    ..... binaire de Ø à 255                     (exemple BI%/)
XX$*    ..... date format E/S jj/mm/aa, code 2 octets (exemple DA$*)
```

En revanche, dans l'enregistrement, les variables sont de longueur non fixe et peuvent être des tableaux BASIC.

Lorsque la variable est un tableau, indiquer le signe " ; " après le nom.

Exemple : AB est un flottant simple
AB ; est un tableau de flottants

Lorsqu'une variable est définie comme tableau, il n'est pas nécessaire d'en indiquer la dimension. En effet, les tableaux sont enregistrés sur disque sous forme de tableaux creux de dimension quelconque (en fait, limitée par la dimension de la mémoire centrale de l'ordinateur). La dimension du tableau sera faite à l'ouverture du fichier. (voir chapitre 4)

Les variables alphanumériques sont, contrairement au cas des variables clé, de longueur variable de Ø (chaine vide) à 255 caractères. Il est donc inutile d'indiquer leur taille.

La liste des variables possibles est donc :

XX	Flottant	XX ;	Tableau de flottants
XX$	Chaine de caractères	XX$;	Tableau de chaines de caractères
XX%	Entier -32767 32768	XX%/ ;	Tableau d'entiers
XX%/	Binaire 0 255	XX%/ ;	Tableau de binaires
XX$*	Date	XX$* ;	Tableau de dates

Lorsque la définition de l'enregistrement a été faite, il ne reste plus qu'à créer le fichier. L'ordre est NEW.

LET " # NEW, [no logique],FICHIER,[no disque] : [nom du fichier]

A la création d'un fichier, le système détermine automatiquement un coefficient de blocage. Celui-ci définit la taille du bloc de base. Le bloc de base sera la plus petite unité de mémoire disque allouée à un enregistrement. Pour savoir comment le système détermine ce coefficient de blocage, reportez-vous au chapitre
Un ordre permet d'imposer un maximum à cette valeur.
Les valeurs possibles sont :

32 = 1/8 secteur
64 = 1/4 secteur
128 = 1/2 secteur
ou 256 = 1 secteur

L'ordre LET"# > n-" imposera au coefficient choisi par le système de ne pas dépasser n.

EXEMPLE : LET"# > 64"

même si le système estime la taille idéale à 128 ou 256, il choisira 64. S'il trouve 32, il restera à 32.

3 - 3 - b fichiers relatifs

Pour les fichiers relatifs, la clé sera remplacée par le pointeur vers le numéro d'enregistrement? Ce pointeur sera un entier XX%. Pour indiquer que le fichier est relatif, précéder le nom de ce pointeur par : "@ ".

La syntaxe générale est : LET" >@XX% = vel ... vep

EXEMPLE : Création d'un fichier des ventes

Le fichier est un fichier relatif. Le pointeur sera NV%(numéro de la vente). L'enregistrement pourra comprendre :

- la date DA$ de la vente
- le tableau des références pièces vendues :
  - FA% ; numéro de famille des pièces
  - FO% ; numéro du fournisseur
  - NP% ; numéro des pièces
  - PV ; prix de vente

Le programme de création sera :

```
100 LET"#C,$" : S = 189
110 LET" >@NV% = DA$,FA%;,NP%;,PV;
120 LET"#NEW-6,FICHIER,Ø : VENTES" : IF PEEK (S) THEN ? "ERREUR"
130 END
```

3 - 3 - c fichiers séquentiels indexés

La syntaxe est la suivante :

LET " > vc1,vc2,...,vcn = ve1,ve2.....,vep

avec vcl ..n : variables composant la clé
et vel ..n : variables composant l'enregistrement

EXEMPLES :

1) Création d'un fichier stock

Le fichier est un fichier à accès par clé.
La clé est composée de 3 éléments.

FA% = numéro de famille de la pièce
FO% = numéro du fournisseur
NP% = numéro de pièce

L'enregistrement comprend :

PA = prix d'achat
PV = prix de vente
TV%/ = taux de TVA
LI$ = libellé de l'article
QS = quantité en stock
QC = quantité en commande
QM = quantité minimum
QX = quantité critique
UN$ = unité

Pour créer ce fichier, le programme à écrire sera :

```
100 LET"#C,$" : S = 189
110 LET" > FA%,FO%,NP% = PA,PV,TV%/,LI$,QS,QC,QM,QX,UN$
120 LET"#NEW,1,FICH,1 : STOCK" : IF PEEK (S) THEN ? "ERREUR"
130 END
```

Pour s'assurer que tout s'est passé normalement, vérifier que le STATUS est nul.

REMARQUE IMPORTANTE : Si une erreur de syntaxe est détectée dans la définition de l'enregistrement en ligne 110, l'erreur indiquera la ligne du NEW.
Dans cette exemple, ? SYNTAX ERROR IN 120

3 - 3 - d fichiers multiclés

La syntaxe de création d'un tel fichier diffère légèrement de la syntaxe classique. Elle sera :

LET" > vmall,...,vmaln & vma21,...,vma2n & ... = vel,ve2...vep

On indique dans la liste des variables de chaque moyen d'accés, les différents moyens d'accès étant séparés par " & " au lieu de " , ".

EXEMPLE : Création d'un fichier Sécurité Sociale :

- clé 1 = nom NO$
- clé 2 = numéro de SS NS$

L'enregistrement comprend :

NO$: le nom
NS$: le numéro de sécurité sociale
AD$: l'adresse
DN$: la date de naissance
SM$: la situation de famille

```
100 LET"#C,$" : S = 189
110 LET > NO$15 & NS$13= AD$,NO$,NS$,DN$",SM$
120 LET"# NEW,1,FICHIER,Ø : SECU": IF PEEK (S) THEN ? "ERREUR"
```

Pour tous les ordres WRITE et ADD, il faut indiquer l'ensemble des clés (de tous les moyens d'accès). Pout tous les ordres READ, NEXT, UPDATE, BORNE, XTRACT, et DELET, il faut indiquer dans l'ordre un paramètre supplémentaire indiquant le moyen d'accès utilisé. Par défaut, ce paramètre vaut 1 (1er moyen d'accès).

EXEMPLE : Avec le fichier que nous venons de créer :

Création :
```
100 NO$ = "LEROI" : NS$ = "1521114254368
110 LET"WRITE-1
```

Lecture d'après le nom :
```
200 NO$ = "LEROI
210 LET"READ-1
```

Lecture d'après le numéro de SS :
```
300 NS$ = "1521114254368
310 LET"READ-1,2
```

Modification :
```
400 AD$ = " 10, RUE JEAN JAURES
410 LET"UPDATE-1,2
```

3 - 4 L'OUVERTURE D'UN FICHIER

L'ordre à utiliser est OPEN avec la même syntaxe que NEW. Cependant, il est inutile d'indiquer une seconde fois la liste des variables. Au contraire, celle-ci pourra être affichée par les ordres suivants :

```
LET"# Open-[n],FICHIER,[d] : [nom]
LET" ENTER-[n]
LET" Visualise
```

Dans l'exemple 1) (fichier stock), le texte suivant sera imprimé :

```
FA%,FO%,NP%
=
PA,PV,TV%/,LI$,QS,QC,QM,QX,UN$
```

Dans le cas où l'enregistrement contient des tableaux, ceux-ci devront être dimensionnés par l'utilisateur avant l'ouverture du fichier. Sinon, leur dimension par défaut sera 8. Cette valeur par défaut peut être modifiée, indiquer pour cela le nombre d'indices désirés en 770. (Voir chapitre Pointeurs du DOS)

Exemple :

```
]DIM A$(10,3)
]LET"#0,1,F, TEST"
]LET"E,1"
]LET"V"
X,Y
=
A,A$;
```

3 - 5 COPIE DE FICHIERS

Pour recopier un fichier sur un nouveau, il faut pouvoir créer le deuxième fichier avec la même structure de variables que le premier. Pour cela, l'ordre ENTER décrit ci-dessus peut remplacer l'ordre de description d'enregistrement.

Exemple : Création du fichier BBB de même structure que le fichier AAA

```
LET"#OPEN,1,Fi,Ø : AAA
LET" ENTER,1
LET"#NEW,2,Fi,Ø : BBB
```

L'ordre ENTER lit le dictionnaire du 1er fichier qui servira pour créer le second.

CHAPITRE IV LES MASQUES DE SAISIE

4 - 1 STRUCTURE D'UN MASQUE

4 - 1 - a Le texte

A la création d'un masque, le texte est entré librement sur l'écran (voir création de masque page 19)

Les fenêtres sont limitées par les caractères " < " et " > " qui indiquent l'emplacement ou s'effectuera la saisie.

Les noms des variables à saisir seront indiqués entre à un endroit quelconque du masque.

A la fin de cette étape, le masque est analysé, puis compact .

Le volume occupé par le masque peut être calculé de la façon suivante :

- les caractères isolés occupent 1 octet chacun.
- chaque suite de caractères identiques de longueur comprise entre 3 et 255 occupe 2 octets.
- les fenêtres et les variables sont considérées comme des blancs.

De plus, 7 octets seront alloués pour chaque fenêtre de saisie.

4 - 1 - b Les zones de saisie

A chaque fenêtre est associée une variable BASIC et un type.

Les types de variables sont les mêmes que dans le cas du DOS.

C'est-à-dire :

- ALPHANUMERIQUE
- FLOTTANT
- ENTIER
- BINAIRE
- DATE

Chaque type pouvant ou non être un tableau à une dimension.

Les contrôles au moment de la saisie sont de 2 types.

- contrôle au niveau du caractère
  (numérique, entier, ...)
- contrôle au niveau de la validation de zone
  (entier < 32768, dates correctes, ...)

Les contrôles que l'on peut demander sont :

- POSITIF ..... FLOTTANT ou ENTIER
- ENTIER ..... FLOTTANT

D'autres contrôles sont implicites :

- ENTIER ..... ENTIER ou BINAIRE
- VECTORISE ou NON . selon le type défini
- DATE ..... si défini comme date
  (la validité des dates est controlée, y compris années bissextiles)
- NUMERIQUE ..... FLOTTANT, ENTIER ou BINAIRE
- 0 <= X < 256 ..... BINAIRE
- -32767 <X < 32768. ENTIER
- moins de 32 chiffres FLOTTANT

Les zones pourront en outre être définies en ENTREE ou en SORTIE sur le masque.

Une zone en sortie ne sera pas demandée au moment d'une saisie.

4 - 1 - c Caractères transparents

Le caractère " @ " sur fond blanc est considéré comme transparent.
Cela signifie que lors du chargement sur l'écran du texte du masque, les caractères déjà présents sur l'écran resteront.

Exemple :
Si vous indiquez en haut et à droite de l'écran la date du jour, et que vous désirez la conserver tout au long du programme, il suffit de mettre dans tous vos masques des caractères @ sur fond blanc à l'emplacement de cette date.

4 - 2 UTILISATION

Un masque sert à la fois à saisir des données et à les imprimer sur l'écran.

L'impression peut être faite :

- sur l'ensemble des variables
- sur les variables définies en entrée
- sur les variables définies en sortie

En aucun cas, l'impression ne débordera des fenêtres. Si la place définie dans une fenêtre est insuffisante pour écrire la variable (ou une des variables dans le cas d'un vecteur), la zone se terminera par le caractère " ¤".

De même que dans le cas d'un fichier, l'ensemble des variables contenues dans le masque constitue un DICTIONNAIRE et aucune liste de variables n'est nécessaire.

Dans le cas de la saisie d'informations sur le masque, l'utilisateur aura la possibilité de passer de fenêtre en fenêtre par la touche RETURN.

A ce moment, on contrôle la validité de la saisie. La zone est transférée dans la variable BASIC correspondante et réinscrite de façon standard.

La touche de fonction CTRL/R renvoie le spot dans la fenêtre précédente.

Les fonctions de gestion d'écran (voir page suivante) seront possibles, dans les limites de la fenêtre .

Remarque : Si aucune modification n'est faite dans la fenêtre, RETURN ne réalise que le passage à la zone suivante.

Le fait de faire RETURN sur la dernière fenêtre peut ou non provoquer la fin de la saisie.
Dans ce dernier cas, cette action renvoie à la première fenêtre définie en entrée
De même, le retour à la fenêtre précédente sur la première fenêtre envoie à la dernière.

La touche ESC permet de valider l'ensemble des informations. Son utilisation est obligatoire pour valider la saisie si l'on a pas demandé une sortie automatique en fin de masque.
Dans le cas contraire, son utilisation est facultative mais évite de faire RETURN jusqu'à la fin du masque.

Si l'on désire abandonner, enfoncer la touche CTRL/A.
Dans ce cas, la zone en cours n'est pas transmise au BASIC.

La variable STATUS permet de savoir quel a été le mode de sortie du masque.

0..... ok
1..... abandon

4 - 3 GESTION DE L'ECRAN

Les commandes sont de deux types :

- traitement d'une zone
- traitement des caractères

4 - 3 - a Validation d'une zone

CTRL/A ..... abandon (STATUS rendu à 1)
ESCAPE ..... fin normale

En utilisation de masque :

RETURN ..... passage à la zone suivante
CTRL/R ..... retour zone précédente

En création de masque :

RETURN ..... descente du spot
CTRL/R ..... remontée du spot

4 - 3 - b Traitement du caractère

CTRL/B ..... caractères fond blanc
CTRL/N ..... caractères normaux
CTRL/F ..... caractères clignotants
→ ..... déplacement du spot à droite
← ..... déplacement du spot à gauche
CTRL/Q ..... déplacement du spot vers le haut
CTRL/Z ..... déplacement du spot vers le bas
CTRL/@ ..... effacement de la zone
CTRL/I ..... insertion d'un caractère
CTRL/D ..... destruction d'un caractère
CTRL/T ..... retour au début de la zone
CTRL/V ..... répétition verticale du caractère

4 - 4 CREATION DE MASQUES

Pour créer un masque utiliser l'ordre :

LET" #NEW,(n°logique), Masque,(n° drive) : (nom)

Cet ordre vous donnera la main pour entrer le texte.

Lorsque celui-ci est tapé, faire ESCAPE pour rendre la main au programme.

La création de masque peut être faite soit en mode direct, soit en mode programme.

Le programme ne détruit pas le contenu actuel de l'écran. Deux possibilités s'offrent alors :

4 - 4 - a Nouveau masque

Il suffit d'effacer l'écran :

Exemple :

```
100 HOME : LET" #NEW,1,MASQUE,1 : TEST-MASK
```

4 - 4 - b Masque issu d'un précédent

Il suffit de charger le masque précédent.

Exemple :

```
100 LET"#OPEN-A,M,1 : TEST-MASK " : REM OUVRE LE MASQUE
110 LET"V,A" : REM AFFICHE LE MASQUE
120 LET"#NEW-B,M,0 : MASQUE 1 : REM CREE LE NOUVEAU MASQUE
```

Pour indiquer les fenêtres, il suffit d'écrire sur l'écran à l'endroit désiré les caractères "< " et " >". Le premier indique le début de la fenêtre, le second la fin. Le nom de la variable sera inscrit entre " ".

Si la fenêtre ne fait qu' 1 caractère, écrire seulement "> ".
Le nom pourra être écrit soit entre les <..>, soit en dehors. Il suffit que l'ordre des variables soit le même que celui des fenêtres.

Les contrôles à effectuer seront indiqués à la suite du nom.

```
XX ..... flottant
XX% ..... entier
XX%/ ..... binaire
XX+ ..... flottant positif
XX%+ ..... entier positif
XX@ ..... flottant entier
XX@: ..... flottant entier cadre à droite
XX :..... flottant format gestion
XX% :..... entier cadré à droite
XX $..... chaine de caractères
XX$: ..... date
XX$+ ..... chaine de caractères considérée comme numérique avec affichage
           format gestion
```

ajouter :

```
? ..... si la zone est en sortie
: ..... si la zone est en enchainement automatique
; (n) ..... si la zone est un indice de tableau
```

(n) = numéro de l'indice (99 pour liste)

Remarque :

Pour une zone en sortie, il est inutile d'indiquer les contrôles.

Les contrôles sont cumulables.

En cas d'erreur sur le masque, (par exemple nombre de variables ≠ du nombre de fenêtres) le système n'acceptera pas votre masque et vous rendra la main pour le corriger, après vous l'avoir signalé par une cloche.

4 - 5 CREATION D'UN MASQUE DE FAÇON AUTOMATIQUE

Pour générer un masque par programme, il suffit de précéder le numéro du disque par le caractère /. Dans ces conditions le programme de création ne donnera pas la main à l'utilisateur pour modifier le contenu de l'écran avec validation par ESC. Au contraire, le programme créera un masque dont le contenu sera la reproduction de l'écran au moment de l'exécution de l'ordre.

Par exemple :

```
10 HOME : PRINT"MASQUE /": PRINT
20 PRINT"NOM <"CHR $(34) NO $"CHR $(34) " >
30 LET" #NEW,3,M,/1 : AUTOMASQUE
40 REM CHR $(34) = "
```

```
MASQUE
NOM < "NO$"                    >
```

Le masque ci-dessus sera enregistré sur le disque sous le nom : "AUTOMASQUE "

En cas d'erreur sur le masque, le programme rend la main en indiquant une erreur et sans sauver le masque (STATUS = 1)

4 - 6 EXEMPLE

Faire l'ordre :

LET"# CL,$ " : LET "# N,1,M,[illegible] : ART ← STOCK

(fermeture générale puis création d'un masque, éventuellement en modification)

Entrez sur l'écran le texte et les fenêtres. Par exemple :

MASQUE ART-STOCK

```
 NO FOURNE <"FO%?   >NO PIECE <"NP%?" >
-----------------------------------------

 LIBELLE   <"LI$ :"                    >

 QTE MINIM <"QM:">   STOCK  <"QS:" >
                     COMMANDE <"QC: ">

 PRX VENTE <"PV:" >  PRIX ACH <"PA:" >

 CODE TVA  <   > "TV%/:"

-----DETAILS TECHNIQUES-----------------
! <"TB$;:"                          > !
! <"TB$;1:"                         > !
! <"TB$;2:"                         > !
! <"TB$;3:"                         > !
! <"TB$;4:"                         >
```

4 - 7 IMPRESSIONS PARAMETREES

Il est possible de paramètrer des impressions par l'utilisation de masques.
L'ordre permettant cette fonction est LET "?-(c)
(c) étant un caractère de contrôle permettant de répéter la ligne à imprimer sur l'écran.
Les caractères de contrôle doivent être inscrits sur fond blanc, l'un en début de ligne à imprimer, l'autre à la fin de cette ligne.
Ces caractères ne seront bien sûr, pas imprimés.

L'ordre LET"?-(c) imprimera les caractères de l'écran séparés par des caractères (c), puis génèrera un passage à la ligne :

Exemple : l'écran contient :

IMPRESSION [A] TEST [A]

LET"?-A" imprimera :

TEST sur l'imprimante désirée.

Il est nécessaire avant d'exécuter l'ordre LET"?-(c)" de définir l'imprimante sur laquelle on désire l'impression. Pour cela, utiliser les ordres :

```
PR#[n° imprimante]
PRINT CHR$ (9)"80N"
```

(cette dernière commande évite que l'impression se fasse également sur l'écran, évitant ainsi de détruire le contenu de ce dernier.

Pour revenir au mode normal d'impression, faire :

```
PR#Ø
```

Cas particulier : (HARD COPY)

Les caractères de contrôle que l'on doit utiliser sont les lettres de l'alphabet A, B, C,... Z
Nénamoins, le caractère de contrôle * joue un rôle particulier. L'ordre]LET"?-*" réalise la recopie complète de l'écran sur l'imprimante.
De même que dans le cas de l'impression d'une ligne, il faut définir l'imprimante choisie et protèger l'écran.
L'ensemble des ordres réalisant le HARD COPY sera donc :

```
]PR#S : PRINT CHR$ (9)"80N" : LET"?-*" : PR#O
```

Utilisation de masque :

Définir par exemple le masque EDIT suivant :

[A]ARTICLE < "AR" > ACHAT <"PA" > VENTE < "PV" > [A]

Pour lister un catalogue de prix. L'enregistrement du fichier stock contient les variables AR (numéro d'articles), PA (prix d'achat) et PV (prix de vente).
L'impression d'une ligne sera :

```
100 REM LA PIECE A ETE LUE SUR LE FICHIER
110 LET"OUTPUT-M" : REM AFFICHAGE SUR LE MASQUE
120 LET"?-A" : REM EDITION SUR IMPRIMANTE
```

Il peut y avoir plusieurs textes différents sur le même écran, qui peuvent être utilisés simultanément si leurs séparateurs sont différents. (par exemple, un titre et la ligne courante)

Remarque : La ligne d'impression peut représenter plusieurs lignes de l'écran permettant ainsi de s'adapter à n'importe quelle imprimante (80 ou 132 colonnes)

De plus, cet ordre gère le nombre de lignes imprimées.
Le nombre de lignes s'obtient par PEEK (947). Cette valeur augmente de 1 à chaque exécution de l'ordre LET"?-(c). Lorsqu'elle atteint 66, elle repasse à 0. L'ordre LET"?" réalise la fonction " FORM FEED " (haut de page) en utilisant le compteur de ligne PEEK (947).

4 - 8 TABLEAUX ET MASQUES

Il y a deux possibilités pour saisir (ou afficher) des tableaux dans les masques.

- saisie de l'ensemble du tableau dans une seule fenêtre.
- saisie d'un élément du tableau dons une fenêtre.

4 - 8 - a La saisie du tableau complet

Cette saisie ne peut se faire que pour les tableaux à une dimension dont les indices 0 à 8 seront traités. (Si le tableau est de dimension supérieure, les valeurs correspondant aux indices supérieurs ne seront pas affectées par une saisie, si la longueur est inférieure, l'erreur BAD SUBSCRIPT ERROR IN 0 apparaitra.)

Pour saisir ou afficher un tableau de cette manière, indiquer dans le masque XX ; 99, XX étant le nom de la variable et les contrôles.
le nombre de variables du tableau saisies et affichées est 9.
Cette valeur peut être modifiée ; pour cela indiquer la valeur souhaitée en 770.

Exemple :

```
<"AC ; 99" >
```

A la saisie, les différents éléments du tableau doivent être séparés par " ; ".

Par exemple :

La saisie de : ;;12;13;;15 réalise :

```
AC (0) = 0                    AC (6) = 0
AC (1) = 0                    AC (7) = 0
AC (2) = 12                   AC (8) = 0
AC (3) = 13                   AC (9) = 0
AC (4) = 0                    AC (10) = 0
AC (5) = 15
```

A l'affichage, le tableau sera réécrit sous la même forme.

4 - 8 - b La saisie d'une seule zone d'un tableau

Pour cela, indiquer après le nom " ; " puis le numéro de l'indice.

Par exemple :

" AC ; 8 "

Dans ce cas l'indice peut varier de 0 à 98.(par défaut, la valeur de l'indice est 0 : " AC ; " équivaut à " AC ; 0 ")

Dans l'exemple ci-dessus, la saisie de : 145,réalise : AC(8) = 145 sans affecter les autres indices.

4 - 9 PRINT USING

Les masques permettent, tant pour l'affichage sur l'écran que pour l'impression sur papier,un formattage des zones numériques.

Il y a trois possibilités d'impression des variables numériques :

- cadrage à gauche, format flottant
- cadrage à droite, format flottant
- cadrage à droite, format gestion

4 - 9 - a Cadrage gauche, format flottant

C'est le formattage pris par défaut dans les masques. Il correspond au format du BASIC.

123	123
10.0	10
12.25	12.25
10000	10000
99999999999	1E + 11
0	0

4 - 9 - b Cadrage droite, format flottant

Les valeurs sont cadrées à droite de la fenêtre de saisie, (permet de colonner des entiers).

123	123
10.0	10
12.2500	12.25
10000	10000
99999999999	1E+11

4 - 9 - c Cadrage droite format gestion

Ce format permet de colonner des valeurs décimales (par exemple des montants).

```
123             !          123.000!
10.0            !           10.000!
12.2500         !           12.250!
10000           !        10000.000!
99999999999     !1E+11             !
```

Lorsque le formattage gestion est impossible, le cadrage de la zone se fait à gauche, ce qui permet de la repérer immédiatement sur un LISTING.
Ce cas peut se produire lorsque :

- la valeur est sous forme exponentielle (ex 12E-12)
- le nombre de chiffres après le " . " est supérieur au nombre désiré.
- lorsque le nombre de chiffre est supérieur à la dimension de la fenêtre.

Une variable nulle ne sera pas imprimée, sauf s'il s'agit de l'indice 0 d'un tableau. Cela permet d'aérer le résultat de tableaux numériques.

Pour fixer le nombre de chiffres imprimés après le point, utiliser l'ordre :

LET"@- nombre de chiffres

Par défaut, au chargement du système, la valeur est 2.

4 - 9 - d Indication du cadrage dans un masque

La zone sera cadrée à droite si le nom de la variable numérique est suivie de "*".
Exemples :

```
"AC## "
"PO%## "
"EN%/## "
```

Si la zone est un entier ou un binaire, seul le cadrage à droite sera réalisé.

Si la zone est un flottant, le formattage gestion sera réalisé.

Pour cadrer à droite un flottant sans colonner les décimales, le déclarer entier par " @ ".

```
"FL## "   :  format gestion
"FL@##"   :  cadrage à droite
```

4 - 10 LES MASQUES GLOBAUX

Ce type d'objet " G " représente un ensemble de masques sauvés sur le disque sous un seul nom. Les avantages de cette méthode sont multiples :

1) gain de place

Un masque isolé occupera sur le disque

- la place d'un nom dans le catalogue
- un nombre entier de secteurs du disque

Réunis, la place est optimisée.

2) gain de vitesse

Le chargement se fait en un seul accès au catalogue et avec un chargement minimum de blocs en mémoire.

3) raccourcissement des programmes

Un seul objet est à ouvrir en lieu et place d'une liste plus ou moins longue.

En revanche, la mise à jour des masques ainsi réunis est plus délicate. Il est donc conseillé de n'effectuer ce regroupement que lorsque votre logiciel est au point.

- mise en œuvre

Pour sauver sur le disque ces masques globaux, la méthode à employer est la suivante :

1) Ouvrir tous les masques concernés. Par exemple :

```
LET"# 0, 1, M, MASK 1
LET"# 0, 2, M, MASK 2
LET"# 0, 3, M, MASK 3
LET"# 0, 4, M, MASK 4
```

2) Exécuter ensuite l'ordre :

```
LET"# N, G, G, GLOBALMASK1234
```

Ceci aura pour effet :

- de supprimer de la mémoire tous les objets non concernés
  - drives
  - fichiers
- de sauver l'ensemble obtenu

Pour utiliser ultérieurement ces masques regroupés, ouvrez l'ensemble en une seule opération par :

```
LET"# 0, G, G, GLOBALMASK1234
```

Tous les masques sont alors accessibles avec le numéro logique qui leur a été donné à la création.

NOTE :

Vous pouvez charger en mémoire plusieurs globaux. Si ceux-ci contiennent des masques ayant été sauvés dans un même numéro logique, seul le premier chargé sera accessible tant qu'il n'aura pas été fermé par l'ordre :

```
LET"# C, numéro logique
```

EXEMPLE :

```
GLOBAL 1 regroupe 1, 2, 3
GLOBAL 2 regroupe 1, 5

LET"# 0, G, G, GLOBAL 1
LET"# 0, H, G, GLOBAL 2
LET"C, 1"                concerne le " 1 " de GLOBAL 1
LET"# C, 1"
LET"C, 1"                concerne le " 1 " de GLOBAL 2
```

IMPORTANT : Le numéro logique indiqué dans la création ou ouverture de globaux n'a pas d'importance. Dans nos exemples, nous avons toujours choisi G. Cependant, le système n'acceptera pas le numéro logique déjà utilisé. Dans ce cas, utilisez un autre caractère.

Exemple : `LET"# 0, 2, G, GLOBAL`

4 - 11 CREATION D'UN MASQUE EN MEMOIRE

LET"# Image, n° logique, M, d : nom

Création d'un masque sans que celui-ci ne soit sauvé sur le disque.
Cet ordre a deux intérêts :

- création de masques temporaires construits automatiquement
- utilisation de la gestion de masques en DOS 3.2 ou DOS 3.3
- fabrication directe de masques globaux

CHAPITRE V LISTE DES ORDES ET SYNTAXE

Les ordres sont envoyés dans une chaine de caractères par LET.

Il y a deux niveaux d'ordres :

- niveau global : ordres gérant des entités complètes (fichiers, masques)
- niveau action : ordres gérant un élément (enregistrement)

5 - 1 NIVEAU GLOBAL

Les ordres sont précédés d'un " # ".

Let"# Open- n° logique ;[type d'ouv.] ; [n° disque] ; [nom]

Ouverture d'un fichier :

- n° logique : 1 caractère quelconque
  C'est ce numéro logique qui sera utilisé dans tous les ordres au " niveau action" concernant ce fichier.
- type d'ouverture
  - M : masque
  - F : fichiers
- n° du disque : 0, 1,....A, B, ...
- nom : 21 caractères significatifs.

LET"# Clear-[n° logique]

Récupère en mémoire centrale la place réservée à un fichier ou un masque.

LET"# Réorganise- [n° logique]

Réorganise la table d'accès d'un fichier à accès par clé. (Améliore le temps d'accès.)
Lorsque de nouvelles pièces sont créées dans le fichier, elles ne sont pas insérées directement à leur place exacte (dans l'ordre croissant des places) En effet, cette opération ralentirait énormément la création (surtout pour les gros fichiers i.e 10000 pièces.)
L'opération de réorganisation reclasse ces clés.

Les fichiers relatifs ne doivent pas être réorganisés.

LET"# Reorganise-$[n° de disque]

Réorganise le catalogue d'un disque. L'exécution de cet ordre permet d'accélérer les ordres LOAD, SAVE, OPEN, NEW.
En effet, le DOS permettant de gérer de nombreux objets sur disque, l'accès à un module se fait en séquentiel indexé (beaucoup plus rapide qu'une recherche séquentielle)
Il demande donc, comme un fichier de ce type, des réorganisations lorsque de nombreuses créations ont été faites. (SAVE, NEW)

LET"#New-[n° logique],[type],[drive] : [nom]

Création de masque ou de fichier.

MASQUES : Le type est Masque

Si le numéro du disque est précédé de @, le masque écrasera un éventuel masque déjà existant et portant le même nom.

Si le numéro du disque est précédé de /, le programme de création ne donne pas la main à l'utilisateur. (Le masque correspond alors au contenu de l'écran au moment de l'exécution de l'ordre.)

En cas d'erreur dans le masque, le programme rend la main pour correction (sauf en création automatique)

FICHIERS : Le type est Fichier

L'ordre NEW doit être précédé de l'ordre "> " ou de l'ordre ENTER.
" > " indique une description d'enregistrement.
ENTER reprend la description d'enregistrement d'un fichier déjà existant.
(voir création de fichiers)

LET"#Delet-[type] , [n° disque] : [nom]

Cet ordre provoque la destruction d'un objet sur le disque.

Indiquer le type de l'objet :

- programme
- fichier
- masque

LET"#Format-[n° disque]

Pour formatter une disquette; l'introduire dans l'un des lecteurs :

Ø : port numéro 6, drive 1
1 : port numéro 6, drive 2
2 : port numéro 5, drive 1

L'ordre de formattage est : LET "#FORMAT,Ø "

LET"# FORMAT, Ø " s'adressera à la disquette située dans le lecteur Ø
LET"# FORMAT,n " avec n = numéro du lecteur

Ne pas s'étonner de la durée relativement importante du formattage, ceci est normal.

En cas d'erreur, le système affiche le message : ? DATA ERROR
signalant qu'il lui est impossible d'utiliser la disquette.

Il peut y avoir plusieurs causes :

1) la porte du lecteur n'est pas fermée ou il n'y a pas de disquette dans le lecteur concerné.

2) La disquette est protégée en écriture et il est impossible d'écrire.

3) La disquette est usagée et la surface magnétique est endommagée, ce qui interdit son utilisation.

Si aucune erreur n'est détectée, le formattage se déroule normalement. (le voyant lumineux du lecteur s'allume)

A la fin du formattage, le système vous rend la main.

Cette disquette est alors utilisable.

5 - 2 NIVEAU ACTION MASQUE

Lorsqu'un masque á été ouvert, les opérations suivantes sont possibles :

LET"Charge-[n° logique]

Charge sur l'écran le texte du masque.
Cette opération efface le contenu de l'écran, sauf aux endroits où le masque a été déclaré comme transparent. (voir le chapitre Création de masques)

LET"Visualise-[n° logique]

Charge sur l'écran le texte du masque et affiche les fenêtres de saisie. Les fenêtres sont réécrites avec la même syntaxe que celle exigée pour la création. Il est donc possible de créer un nouveau masque ou d'en modifier un en utilisant successivement les ordres visualise et NEW.
Les zones transparentes apparaissent sous la forme de caractères "@" sur fond blanc.

LET"Output-[n° logique][Ø/O]

Affiche sur l'écran les variables du masque.
Le paramètre (optionnel) Ø/O permet d'indiquer quelles sont les variables à afficher.

- Ø : affichage des variables définies en entrée (celles où se déplace le spot en lecture)
- O : affichage des variables en sortie

  Si le paramètre est omis, toutes les variables sont affichées.

Si une variable est trop longue pour entrer dans une fenêtre, elle est tronquée, et un caractère " ⁝ " apparait en fin de fenêtre.

LET"Print-[n° logique][Ø/O]

Cet ordre est le cumul des deux ordres Charge & Output.
La syntaxe est la même que celle de Output.

LET"Input-[n° logique][,n° de zone]

Donne la main à l'utilisateur pour saisir les zones.
Seules les fenêtres déclarées en entrée sont concernées par cet ordre.
Le paramètre (optionnel) permet d'indiquer à quelle fenêtre doit commencer la saisie. Attention ! seules les fenêtres en entrée comptent.
La première fenêtre porte le numéro 1, et ainsi de suite.
La valeur par défaut est 1.

Si le numéro indiqué est supérieur au nombre de fenêtres en entrée, le programme commencera par balayer le masque autant de fois qu'il est nécessaire pour atteindre ce numéro. Si la dernière fenêtre est du type " enchainement automatique ", le programme ressortira alors sans donner la main.

LET"Take, n° logique"

Reprend le contenu de toutes les variables en entrées du masque indiqué telles qu'elles sont actuellement sur l'écran.

EXEMPLE 1 : Remise à zéro

```
LET"C, M"
LET"T, M"
```

Remet toutes les variables du masque à vide ou nulles, puisque l'ordre chargé n'affiche aucune variable.

EXEMPLE 2 : Paramétrage

On saisit des paramètres sur un masque M, puis on sauve l'ensemble :

```
LET"# O, M, M, Ø : QUESTIONS
LET"P, M"
LET"I, M"
LET"# N, P, M, Ø : PARAMETRES
```

pour relire les paramètres, faire :

```
LET"# O, M, M, Ø : QUESTIONS
LET"# O, P, M, Ø : PARAMETRES
LET"C, P"
LET"T, M"
```

Toutes les variables du masque M reprennent les valeurs qu'elles avaient au moment du sauvetage.

5 - 3 NIVEAU ENREGISTREMENT FICHIER

Ces ordres sont possibles sur un fichier ouvert.

LET"Write-[n° logique]

Création d'un nouvel enregistrement dans un fichier.

- Si le fichier est un fichier séquentiel indexé, la création se fait en fonction de la valeur de la clé au moment de l'ordre.

Exemple :

Fichier de clé AA$et AA 10 et d'enregistrement XX$

```
100 LET"#OPEN-1, FICHIER,Ø : FICO
110 AA$ = "MEMSOFT": AA =1980
120 XX$ = "ENREGISTREMENT"
130 LET"WRITE,1
140 IF PEEK(189) THEN PRINT"EXISTANT"
```

L'ordre WRITE n'accepte pas les homonymes. Si la clé de l'enregistrement à créer existe déjà dans le fichier, le status renvoyé sera non nul et l'enregistrement ne sera pas créé.

- Si le fichier est un fichier relatif, le DOS ne tiendra pas compte de la valeur du pointeur. L'enregistrement sera créé à la fin du fichier, avec la première valeur libre pour le pointeur. Cette valeur du pointeur sera renvoyée à l'utilisateur.

LET"Add-[n° logique]

L'ordre ADD est identique à l'ordre WRITE, mais il accepte les homonymes dans le cas des fichiers séquentiels indexés.
Lorsque plusieurs enregistrements de même clé sont créés dans un fichier, l'ordre de rangement est l'ordre inverse de l'ordre de création.
L'enregistrement le plus récent sera donc lu en premier lors d'une lecture séquentielle.

LET"Read-[n° logique]

Lecture d'un enregistrement en fonction de la valeur de la clé (ou du pointeur).

Exemple :

```
200 XX$ = " " : AA$ = "MEMSOFT" : AA = 1980
210 LET"READ-1
220 ?XX$
```

LET"Next-[n° logique]

Lecture de l'enregistrement suivant.

Employé après l'ouverture du fichier, l'ordre NEXT permet de lire le fichier séquentiellement, chaque appel renvoyant à l'utilisateur la clé (ou le pointeur) et l'enregistrement suivants.

Si une lecture a été faite par READ, la lecture séquentielle par NEXT commence à la première clé (ou pointeur) suivant celle (celui) du READ.

REMARQUE : L'ordre NEXT ne tient pas compte de la valeur de la clé indiquée dans les variables BASIC. C'est la valeur obtenue à la dernière lecture qui compte.
Pour lire l'enregistrement suivant pour une valeur donnée de la clé, enchainer READ et NEXT.

Exemple :

Dans un fichier dont la clé est NO$ de longueur de 10 caractères, cherchons le premier enregistrement commençant par " T ".

```
100 LET"#C,$"
110 LET"#0,1,F,NOMS"
120 NO$ = " T "
130 LET"R,1" : IF PEEK (189) = 0 THEN 150
140 LET"N,1"
150 REM ICI LE PREMIER ENREGISTREMENT
160 REM COMMENCANT PAR " T " A ETE LU
```

LET"Update-[n° logique]

Mise à jour d'un enregistrement.
Indiquer les nouvelles valeurs des variables de l'enregistrement, puis exécuter UPDATE.
Il n'est pas nécessaire d'avoir lu l'enregistrement auparavant. (mais dans ce cas la mise à jour est plus rapide)
En cas d'homonymie, si l'un des homonymes vient d'être lu, c'est celui-là qui est modifié. Sinon, c'est toujours le premier (c'est-à-dire le plus récent) qui l'est.

Exemple :

```
300 AA = 1980 : AA$ = "MIS": LET" READ-1
310 XX$ = XX$+" VERSION 2
320 LET"UPDATE-1
```

LET"Delet-[n° logique]

Destruction d'un enregistrement dont on indique la clé (ou le pointeur). Cette opération est possible aussi bien sur les fichiers séquentiels indexés que sur les fichiers relatifs.
Dans le cas d'un fichier séquentiel indexé, il est possible de recréer l'enregistrement détruit, par l'ordre WRITE ou ADD. Ceci est impossible pour un fichier relatif et une valeur du pointeur détruite ne pourra être réutilisée. (Dans le cas des fichiers relatifs, l'intérêt principal de cet ordre est la récupération de la place de l'enregistrement sur le disque.) Dans le cas d'un fichier multiclé, la place de l'enregistrement ne sera récupérée qu'à la prochaine ré-organisation.

LET"Borne-[n° logique]

Fixe une borne maximum au fichier pour les lectures séquentielles.
L'ordre NEXT déclenche une erreur " fin de fichier " (status = 255) si cette borne est atteinte ou dépassée.
Pour utiliser l'ordre BORNE, positionner les clés (ou le pointeur) à la valeur maximum désirée, puis exécuter l'ordre.
La valeur exacte de la borne, si elle correspond à une clé existante, n'est pas lue.
REMARQUE : l'ordre BORNE ne provoque pas de lecture disque, et ne trouble pas le fonctionnement de l'ordre NEXT (qui, rappelons-le, travaille par rapport à la dernière lecture et non par rapport aux valeurs des variables.)
Pour ne lire qu'une partie d'un fichier, il suffit donc de fixer une borne minimum par READ, une borne maximum par BORNE. Une boucle de lecture par NEXT permet d'obtenir cet extrait du fichier comme s'il s'agissait d'un fichier entier.

LET"Xtract-[n° logique],[n° variable]

Cette commande permet d'extraire des éléments d'un fichier en conditionnant l'une des variables composant la clé.

Par exemple, si la clé se compose de :

A1 % numéro de famille
A2 % numéro de fournisseur
A3 % numéro interne de l'article

Le numéro logique de ce fichier est " F ".

Pour lister les éléments du fichier appartenant au fournisseur numéro 2, il faut tout d'abord désigner au système le numéro d'ordre de la variable testée et sa valeur.

```
] LET " XTRACT, F, 2 "
] A1 % = Ø : A2 % = 2 : A3 % = Ø
] LET " READ, F "
? DIRECT ERROR
```

La commande XTRACT positionne le numéro d'ordre de la variable.
La commande READ, qui ici n'aboutit pas, fournit au système une valeur de référence pour la variable A2 %, en l'occurence 2.
De plus, il ne faut pas oublier que ce READ positionne le haut de fichier pour l'instruction LET " NEXT, F " qui extrait les éléments recherchée.

Il est possible de redonner une nouvelle valeur de recherche à la commande NEXT.
Pour cela, il faut exécuter un nouveau READ en positionnant les nouvelles valeurs pour les variables composant la clé.

ATTENTION : ne pas oublier qu'en faisant cela, le haut de fichier est repositionne pour le prochain NEXT.

LET"Xindex-[n° logique]

Pour les fichiers relatifs, cet ordre renvoie dans le pointeur le numéro d'ordre du prochain enregistrement qui sera créé.
Cet ordre permet, entre autre, de créer des chainages entre fichiers avant d'avoir créé tous les enregistrements.

5 - 4 PROGRAMMES

Les ordres sont :

LOAD"[d] : [nom]

Chargement d'un programme.

LOAD"#[d] : [nom]

Chargement d'un programme à la suite de celui déjà existant en mémoire centrale.
IMPORTANT : Les numéros de lignes du second programme doivent être > à ceux du 1er. Dans le cas contraire, modifiez-les au préalable par l'utilitaire RENUMEROTE.

LOAD"/[d] : [nom]

Chargement, puis exécution d'un programme sans détruire les variables.
Attention ! le programme doit être d'une taille inférieure à celle du dernier programme chargé normalement.

SAVE"[d] : [nom]

Sauve le programme en cours.
S'il existe une erreur celle-ci sera signalée dans les status (PEEK (189))

SAVE"@[d] : [nom]

Sauve le programme en cours en écrasant éventuellement une version précédemment enregistrée.

RUN"[d] : [nom]

Charge le programme et lance son exécution.

RUN"#(d) : [nom]

Charge le programme à la suite de celui déjà en mémoire, puis lance l'exécution de l'ensemble.

LET"%-[d]

Imprime le catalogue du disque d.

LET"%-[d],[P/M/F]

Imprime seulement la liste des Programmes, Masques ou Fichiers.

```
]NEW

]10PRINT"TEST"

]SAVE"1:TEST1"

]NEW

]20FORI=0TO10:?"#";:NEXT

]SAVE"1:TEST2"

]LET"*,1"

CATALOG DISK 1

P  TEST1
P  TEST2

]NEW

]LOAD"1:TEST1"

]LOAD"#1:TEST2"

]LIST

10  PRINT "TEST"
20  FOR I = 0 TO 10: PRINT "#";: NEXT


]SAVE"1:TEST1+2"

]RUN
TEST
###########
]LET"*,1"

CATALOG DISK 1

P  TEST1
P  TEST1+2
P  TEST2
```

REMARQUE : Dans le CATALOG, les objets sont imprimés dans l'ordre alphabétique. Les fichiers d'abord, puis les masques et enfin les programmes.

Notes concernant la syntaxe
---

1) Caractères significatifs

Les blancs ne sont pas significatifs, sauf dans le nom d'un objet.
Seul le premier caractère suivant un séparateur (, ; - :) compte (sauf dans le cas du nom d'un objet).
Les autres caractères sont ignorés.

Exemple :

LET"#OPEN-1, FICHIER,Ø : FIC TEST

est équivalant à :

LET"#0 - 1 , F , Ø : FIC TEST

2) Numéro logigue

On peut choisir comme numéro logique n'importe quel caractére ASCII à l'exception du caractére " $ ".
De plus, dans tous les ordres, les caractères suivant le numéro logique sont ignorés jusqu'à un éventuel séparateur.

On pourra écrire :

LET"#OP-STOCK-F,1 : STOCK
LET"READ-STOCK
Le numéro logique étant " S ".

3) Nom d'un objet

Le nom d'un objet est une suite de caractères quelconques. Seuls les 21 premiers caractères sont pris en compte. S'il y en a moins de 21, le nom est complété par des blancs (code ASCII 32).

4) Valeur par défaut du disgue

Si le premier caractère du nom est supérieur à " 9 ", on pourra omettre le numéro du disque, qui gardera la valeur précédente.

Exemple :

LOAD"0 : PRG TEST
SAVE"PRG TEST
SAVE"@PRG TEST

5) Paramétrage des ordres

Les ordres ont été montrés sous la forme :

LET"XXXXXX
ou LOAD"XXXXX....

Mais l'ordre peut être une chaine de caractères quelconque, construite ou non, envoyée par LET, LOAD, SAVE ou RUN.

Exemple :

100 INPUT"NOM DU MASQUE" ; NM$
110 LET"#OPEN-2,MASQ,1 : "+NM$

6) Séparateurs

Dans tous les ordres, les séparateurs sont interchangeables. (, : ; -)

7) Cas particuliers

Tout ordre du MEM/DOS suivant THEN doit être précédé de " : ".

Exemple : IF A = B THEN : LET G

Si l'on omet les " : ", le système répondra :

SYNTAX ERROR

5 - 5 BINAIRES ET PROGRAMMES ASSEMBLEURS

Les modules binaires se comportent d'une façon voisine de celle des programmes BASIC. Les seules différences sont :

1) Les adresses délimitant la partie de mémoire à sauver ont été définies par l'utilisateur et non imposé comme dans le cas d'un programme BASIC.

2) Ils sont notés dans le catalogue sous le code " B " au lieu du code " P "

La syntaxe est la suivante :

Sauvetage :

SAVE "$XXXX, $YYYY, d : nom"

avec : XXXX adresse minimale en héxadécimal

Exemple : SAVE"$1ØØØ,$11ØØ,Ø : BIN"

sauve la partie de la mémoire comprise entre :
$1ØØØ et $1ØFF

Relecture :

LOAD"$XXXX,$YYYY,d : nom"

avec XXXX adresse de départ choisie
YYYY adresse maximale autorisée

Cette deuxième adresse permet de mettre en " chien de garde " évitant d'écraser une autre partie de la mémoire. Même si YYYY est supérieur à l'adresse de fin de chargement, le comportement du LOAD ne sera pas mofifié.

NOTE : Il peut être interessant de pouvoir connaître l'adresse de fin d'un module à l'issue d'un chargement.
Cela est possible, l'adresse de fin du module +1 se trouve placé en ADBLC à l'issue du LOAD avec,

ADBLC	= $3ØE	poids fort de l'adresse
ADBLC + 1	= $3ØF	poids faible de l'adresse

5 - 6 ACCES DIRECT

Cette fonction permet de lire ou écrire un ou plusieurs secteurs, directement sur le disque ou la disquette. La syntaxe est :

```
LET"$d,[C=]c ,[N=]n ,[P=]p ,[T=]t ,[S=]s ,$XXXX
```

Avec XXXX adresse du chargement
- d numéro du drive (Ø 5)
- n nombre de secteurs à charger ou sauver
- p piste
- t tête
- s secteur
- c commande (1 = lire ; 2 = écrire ; 4 = formatter)

Les caractères notés en [ ] sont facultatifs, mais contrôlés s'ils sont présents.

Deux particularités interessantes de cet ordre sont :

1) dans le cas où le nombre n est trop important et que cela conduirait à dépasser la fin du disque, l'ordre est exécuté normalement tant que cela est possible, puis la main est rendue avec un status non nul pour signaler le fait.

2) après son exécution, le programme positionne en :

$3ØA $3ØB $3ØC

Les paramètres piste, tête, secteurs du PROCHAIN secteur à lire.

Une conséquence de ces caractéristiques est la simplicité de réalisation de certains utilitaires.

EXEMPLE : Programme copiant des drives de même type.

```
ready.

 50 himem:16*256
 100 input  " copie depuis ?";d1$
 110 input  " vers         ?";d2$
 120 let"#c,$"
 200 x=7*256*16+15*256+16+5
 210 p=0:s=0:t=0
 215 vtab10:print"lecture  "p" "t" "s
 220 let"$"+d1$+",1,16,"+str$(p)+","+str$(t)+","+str$(s)+"$1000
 225 vtab10:print"ecriture "p" "t" "s
 230 let"$"+d2$+",2,16,"+str$(p)+","+str$(t)+","+str$(s)+"$1000
 240 if  WS     then500
 260 p=peek(x+1):t=peek(x+2):s=peek(x+3):goto215
 500 print"@@@fin
ready.
```

5 - 7 EXEMPLES

```
 100 REM ************************************************
 101 REM * EXEMPLE 1                                     *
 102 REM *                                               *
 103 REM * --------------------                          *
 104 REM * CLEAR-$     >      NEW                        *
 105 REM * --------------------                          *
 110 REM *                                               *
 120 REM *                                               *
 130 REM * CREATION D'UN FICHIER STOCK                    *
 131 REM * --------------------------                    *
 135 REM *                                               *
 140 REM * LE FICHIER EST UN FICHIER SEQUENTIEL INDEXE    *
 150 REM *                                               *
 160 REM * LA CLE :                                      *
 170 REM *  FO%=NUMERO DE FOURNISSEUR                    *
 180 REM *  NP%=NUMERO DE PIECE                          *
 190 REM *                                               *
 200 REM * L'ENREGISTREMENT                              *
 210 REM *  PA=PRIX D'ACHAT                              *
 220 REM *  PV=PRIX DE VENTE                             *
 230 REM *  TV%/=TYPE TVA                                *
 240 REM *  QS=QUANTITE EN STOCK                         *
 250 REM *  QC=QUANTITE EN COMMANDE                      *
 260 REM *  QM=QUANTITE MINIMALE                         *
 270 REM *  LI$=LIBELLE DE LA PIECE                      *
 280 REM *                                               *
 290 REM *                                               *
 300 REM * LE PROGRAMME INDIQUE LA DEFINITION DE         *
 310 REM *  L'ENREGISTREMENT ET CREE LE FICHIER          *
 320 REM ************************************************
 400 LET"#CLEAR-$"
 410 LET">FO%,NP%=PA,PV,TV%,QS,QC,QM,LI$
 420 LET"#NEW,1,FICHIER,0:STOCK
 430 IF    WS   THENPRINT"ERREUR "
READY.
```

```
100 REM ************************************************
105 REM * EXEMPLE 2                                     *
110 REM *                                               *
120 REM *                                               *
130 REM * MISE A JOUR D'UN FICHIER                      *
140 REM *                                               *
150 REM * LE PROGRAMME EST INDEPENDANT                  *
160 REM * DU FICHIER                                    *
170 REM *                                               *
180 REM * SEULS LES MASQUES UTILISES SONT A CHANGER     *
190 REM *                                               *
280 REM *                                               *
300 REM * LE PROGRAMME PERMET LES ORDRES SUIVANTS :     *
310 REM *  LECTURE D'UNE PIECE                          *
320 REM *  CREATION D'UNE PIECE                         *
330 REM *  MODIFICATION D'UNE PIECE                     *
340 REM *  DESTRUCTION D'UNE PIECE                      *
350 REM *  SUIVANT (LECTURE DANS L'ORDRE DES CLES)      *
360 REM *                                               *
370 REM * LE PROGRAMME UTILISE 2 MASQUES                *
380 REM * - CLE-XXXXX = DEMANDE DE LA CLE ET DE L'ORDRE *
390 REM * - ART-XXXXX = DEMANDE OU AFFICHAGE DE LA PIECE*
400 REM *                                               *
410 REM ************************************************
500 LET"#CLEAR-$":       INPUT"NOM DU FICHIER ";N$
510 LET"#OPEN,F,FICHIER,0:"+N$
520 LET"#OPEN,C,MASQUE,0:CLE-"+N$
530 LET"#OPEN,A,MASQUE,0:ART-"+N$
540 LET"CHARGE,C
550 LET"OUTPUT,C":MC$=""
560 LET"INPUT,C":IF   WS   THENMC$="REPRENEZ":GOTO550
570 IFO$="F"THENEND
580 IFO$="L"THEN700
590 IFO$="S"THEN800
600 IFO$="M"THEN900
610 IFO$="C"THEN1000
620 IFO$="D"THEN1100
630 MC$="ORDRE ERRONE":GOTO550
700 LET"READ-F":IF  WS   THEN790
710 MA$="LECTURE:FAITE RETURN":LET"PRINT,A
720 GETA$:IFA$<>CHR$(13)THEN720
730 GOTO540
790 MC$="ARTICLE INEXISTANT":GOTO550
800 LET"READ-F":IF   WS   THEN790
810 LET"NEXT-F":IF  WS   THENMC$="PLUS D'ARTICLE":GOTO550
820 GOTO710
900 LET"READ-F":IF   WS   THEN790
910 MA$="MODIFICATION":LET"PRINT-A":LET"INPUT-A":IF   WS   THEN990
920 LET"UPDATE-F":GOTO540
990 MC$="ABANDON":GOTO540
1000 LET"READ-F":IF  WS   =0THENMC$="ARTICLE EXISTANT":GOTO550
1010 MA$="CREATION":LET"PRINT-A":LET"INPUT-A":IF  WS   THEN990
1020 LET"WRITE-F":GOTO540
1100 LET"READ-F":IF  WS   THEN790
1110 MA$="OK POUR DETRUIRE (O/N)":LET"PRINT-A"
1120 GETA$:IFA$="N"THEN990
1130 IFA$<>"O"THEN1120
1140 LET"DELET-F":GOTO540
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 3                                     *
102 REM *                                               *
103 REM * --------------------                          *
104 REM * CLEAR-$  OPEN  NEXT                           *
105 REM * --------------------                          *
110 REM *                                               *
120 REM *                                               *
130 REM * LECTURE SEQUENTIELLE D'UN FICHIER             *
131 REM *-------------------------------                *
135 REM *                                               *
140 REM * LE FICHIER EST UN FICHIER SEQUENTIEL INDEXE    *
150 REM *                                               *
160 REM * LA CLE :                                      *
170 REM *  FO%=NUMERO DE FOURNISSEUR                    *
180 REM *  NP%=NUMERO DE PIECE                          *
190 REM *                                               *
200 REM * L'ENREGISTREMENT                              *
210 REM *  PA=PRIX D'ACHAT                              *
220 REM *  PV=PRIX DE VENTE                             *
230 REM *  TV%/=TYPE TVA                                *
240 REM *  QS=QUANTITE EN STOCK                         *
250 REM *  QC=QUANTITE EN COMMANDE                      *
260 REM *  QM=QUANTITE MINIMALE                         *
270 REM *  LI$=LIBELLE DE LA PIECE                      *
280 REM *                                               *
290 REM *                                               *
300 REM * LE PROGRAMME LISTE  LE                        *
310 REM *    FICHIER COMPLET                            *
320 REM ************************************************
400 LET"#CLEAR-$"
410 LET"#OPEN,1,FICHIER,0:STOCK
450 LET"NEXT-1": IF    WS    THEN END
460 PRINT FO%,NP%,PA,PV,TV%,QS,QC,QM,LI$:GOTO450
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 4                                     *
102 REM *                                               *
103 REM * --------------------------------              *
104 REM * CLEAR-$  OPEN  NEXT  READ  BORNE              *
105 REM * --------------------------------              *
110 REM *                                               *
120 REM *                                               *
130 REM * LECTURE D'UNE PARTIE D'UN FICHIER              *
131 REM * --------------------------------              *
135 REM *                                               *
140 REM * LE FICHIER EST UN FICHIER SEQUENTIEL INDEXE    *
150 REM *                                               *
160 REM * LA CLE :                                      *
170 REM *  FO%=NUMERO DE FOURNISSEUR                    *
180 REM *  NP%=NUMERO DE PIECE                          *
190 REM *                                               *
200 REM * L'ENREGISTREMENT                              *
210 REM *  PA=PRIX D'ACHAT                              *
220 REM *  PV=PRIX DE VENTE                             *
230 REM *  TV%/=TYPE TVA                                *
240 REM *  QS=QUANTITE EN STOCK                         *
250 REM *  QC=QUANTITE EN COMMANDE                      *
260 REM *  QM=QUANTITE MINIMALE                         *
270 REM *  LI$=LIBELLE DE LA PIECE                      *
280 REM *                                               *
290 REM *                                               *
300 REM * LE PROGRAMME LISTE  LES PIECES D'UN FOURNISSEUR *
310 REM *                                               *
320 REM ************************************************
400 LET"#CLEAR-$"
410 LET"#OPEN,1,FICHIER,0:STOCK
420 INPUT"NOM DU FOURNISSEUR CONCERNE ";FO%
430 NP%=0:LET"READ-1
440 NP%=32700:LET"BORNE-1
450 LET"NEXT-1": IF   WS    THEN END
460 PRINT FO%,NP%,PA,PV,TV%,QS,QC,QM,LI$:GOTO450
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 5                                     *
102 REM *                                               *
103 REM * -------------------------------               *
104 REM * CLEAR-$  OPEN  NEXT  XTRACT                   *
105 REM * -------------------------------               *
110 REM *                                               *
120 REM *                                               *
130 REM * EXTRACTION D'UN SOUS FICHIER                   *
131 REM * -------------------------------               *
135 REM *                                               *
140 REM * LE FICHIER EST UN FICHIER SEQUENTIEL INDEXE    *
150 REM *                                               *
160 REM * LA CLE :                                      *
170 REM *  FO%=NUMERO DE FOURNISSEUR                    *
180 REM *  NP%=NUMERO DE PIECE                          *
190 REM *                                               *
200 REM * L'ENREGISTREMENT                              *
210 REM *  PA=PRIX D'ACHAT                              *
220 REM *  PV=PRIX DE VENTE                             *
230 REM *  TV%/=TYPE TVA                                *
240 REM *  QS=QUANTITE EN STOCK                         *
250 REM *  QC=QUANTITE EN COMMANDE                      *
260 REM *  QM=QUANTITE MINIMALE                         *
270 REM *  LI$=LIBELLE DE LA PIECE                      *
280 REM *                                               *
290 REM *                                               *
300 REM * LE PROGRAMME LISTE  LES PIECES DONT           *
310 REM * LE NUMERO (NP%) EST EGAL  A 1000              *
320 REM ************************************************
400 LET"#CLEAR-$"
410 LET"#OPEN,1,FICHIER,0:STOCK
440 NP%=1000:LET"XTRACT-1
450 LET"NEXT-1": IF  WS    THEN END
460 PRINT FO%,NP%,PA,PV,TV%,QS,QC,QM,LI$:GOTO450
READY.
```

```
 100 REM ******************************************************
 101 REM * EXEMPLE 6                                          *
 102 REM *                                                    *
 103 REM * ---------------------                              *
 104 REM * CLEAR-$      >     NEW                             *
 105 REM * ---------------------                              *
 110 REM *                                                    *
 120 REM *                                                    *
 130 REM * CREATION D'UN FICHIER RELATIF                      *
 131 REM * ----------------------------                       *
 135 REM *                                                    *
 137 REM *  PAR EXEMPLE : FICHIER D'ECRITURES COMPTABLES      *
 138 REM *                                                    *
 140 REM * SA DESCRIPTION EST :                               *
 150 REM *                                                    *
 160 REM * LE POINTEUR                                        *
 170 REM *  NE%=NUMERO D'ECRITURE                             *
 190 REM *                                                    *
 200 REM * L'ENREGISTREMENT                                   *
 210 REM *  DA$*=DATE DE L'ECRITURE                           *
 220 REM *  LI$=LIBELLE                                       *
 230 REM *  CP$=COMPTE CONCERNE                               *
 240 REM *  DE=DEBIT                                          *
 250 REM *  CR=CREDIT                                         *
 260 REM *  NF$=REFERENCE FACTURE                             *
 270 REM *  DP$*=DATE DE L'OPERATION                          *
 280 REM *                                                    *
 290 REM *                                                    *
 300 REM * LE PROGRAMME INDIQUE LA DESCRIPTION               *
 310 REM *  DE L'ENREGISTREMENT ET CREE LE FICHIER            *
 320 REM ******************************************************
 400 LET"#CLEAR-$"
 410 LET">@NE%=DA$*,LI$,CP$,DE,CR,NF$,DP$*
 420 LET"#NEW,2,FICHIER,1:ECRITURES
 430 IF   WS     THEN PRINT"ERREUR
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 7                                     *
102 REM *                                               *
103 REM * ----------------------                        *
104 REM * CLEAR-$       >      NEW                      *
105 REM * ----------------------                        *
110 REM *                                               *
120 REM *                                               *
130 REM * LISTE D'UN FICHIER RELATIF                     *
131 REM * --------------------------                    *
135 REM *                                               *
137 REM *  PAR EXEMPLE : FICHIER D'ECRITURES COMPTABLES  *
138 REM *                                               *
140 REM * SA DESCRIPTION EST :                          *
150 REM *                                               *
160 REM * LE POINTEUR                                   *
170 REM *  NE%=NUMERO D'ECRITURE                        *
190 REM *                                               *
200 REM * L'ENREGISTREMENT                              *
210 REM *  DA$*=DATE DE L'ECRITURE                      *
220 REM *  LI$=LIBELLE                                  *
230 REM *  CP$=COMPTE CONCERNE                          *
240 REM *  DE=DEBIT                                     *
250 REM *  CR=CREDIT                                    *
260 REM *  NF$=REFERENCE FACTURE                        *
270 REM *  DP$*=DATE DE L'OPERATION                     *
280 REM *                                               *
290 REM *                                               *
300 REM * LE PROGRAMME LISTE LE FICHIER COMPLET          *
310 REM *  ( JOURNAL )                                  *
320 REM ************************************************
400 LET"#CLEAR-$"
420 LET"#OPEN,3,FICHIER,1:ECRITURES
430 LET"NEXT-3":IF   WS     THEN END
440 PRINTNE%,DA$,LI$,DE,CR,NF$,DP$
450 GOTO430
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 8                                     *
102 REM *                                               *
103 REM * ----------------------                        *
104 REM * CLEAR-$       >      NEW                      *
105 REM * ----------------------                        *
110 REM *                                               *
120 REM *                                               *
130 REM * CREATION D'UN FICHIER MULTICLE                 *
131 REM * ----------------------------                  *
135 REM *                                               *
137 REM *  PAR EXEMPLE : FICHIER DES COMPTES COMPTABLES  *
138 REM *                                               *
140 REM * SA DESCRIPTION EST :                          *
150 REM *                                               *
160 REM * 1ERE CLE                                      *
170 REM *  CO$10=LE NUMERO DE COMPTE                    *
190 REM *                                               *
200 REM * 2EME CLE                                      *
210 REM *  AL$12=CODE ALPHABETIQUE EQUIVALENT           *
220 REM *                                               *
230 REM * L'ENREGISTREMENT                              *
240 REM *  LI$=LIBELLE DU COMPTE                        *
250 REM *  CR=CREDIT                                    *
260 REM *  DE=DEBIT                                     *
270 REM *  SD=SOLDE                                     *
280 REM *  AL$=CODE ALPHA (REPETE POUR INFORMATION)     *
290 REM *  CO$=COMPTE     (REPETE POUR INFORMATION)     *
295 REM *                                               *
300 REM * LE PROGRAMME INDIQUE LA DESCRIPTION           *
310 REM *  DE L'ENREGISTREMENT ET CREE LE FICHIER       *
320 REM ************************************************
400 LET"#CLEAR-$"
410 LET">CO$10 & AL$12 =LI$,CR,DE,SD,AL$,CO$
420 LET"#NEW,4,FICHIER,1:COMPTES
430 IF    WS    THEN PRINT"ERREUR
READY.
```

```
100 REM ************************************************
101 REM * EXEMPLE 9                                     *
102 REM *                                               *
103 REM * ----------------------------                  *
104 REM * CLEAR-$  OPEN  NEXT  UPDATE                   *
105 REM * ----------------------------                  *
110 REM *                                               *
120 REM *                                               *
130 REM * MISE A JOUR D'UN FICHIER  (MULTICLE)           *
131 REM * ----------------------------                  *
135 REM *                                               *
137 REM *  PAR EXEMPLE : FICHIER DES COMPTES COMPTABLES  *
138 REM *                                               *
140 REM * SA DESCRIPTION EST :                          *
150 REM *                                               *
160 REM * 1ERE CLE                                      *
170 REM *  CO$10=LE NUMERO DE COMPTE                    *
190 REM *                                               *
200 REM * 2EME CLE                                      *
210 REM *  AL$12=CODE ALPHABETIQUE EQUIVALENT           *
220 REM *                                               *
230 REM * L'ENREGISTREMENT                              *
240 REM *  LI$=LIBELLE DU COMPTE                        *
250 REM *  CR=CREDIT                                    *
260 REM *  DE=DEBIT                                     *
270 REM *  SD=SOLDE                                     *
280 REM *  AL$=CODE ALPHA (REPETE POUR INFORMATION)     *
290 REM *  CO$=COMPTE     (REPETE POUR INFORMATION)     *
295 REM *                                               *
300 REM * LE PROGRAMME EFFECTUE LE CALCUL SUIVANT       *
310 REM *  SOLDE (SD) = DEBIT (DE) - CREDIT (CR)        *
315 REM *  ET REECRIT L'ENREGISTREMENT MODIFIE          *
320 REM *                                               *
330 REM *  ICI ,LE FICHIER EST MULTICLE .LA LECTURE SE   *
340 REM *  FAIT SUR LE PREMIER MOYEN D'ACCES            *
350 REM *  QUI CORRESPOND A LA VALEUR PAR DEFAUT        *
360 REM *                                               *
370 REM *  LE PROGRAMME EST DONC LE MEME POUR UN FICHIER *
375 REM *  SIMPLE OU UN FICHIER MULTICLE                *
380 REM ************************************************
400 LET"#CLEAR-$"
420 LET"#OPEN,6,FICHIER,1:COMPTES
430 LET"NEXT-6":IF   WS     THEN END
440 SD=DE-CR: LET"UPDATE-6":GOTO430
READY.
```

```
100 REM ************************************************
105 REM * EXEMPLE 10                                    *
110 REM *                                              *
120 REM *                                              *
130 REM * MISE A JOUR D'UN FICHIER AVEC HOMONYMES       *
140 REM *                                              *
150 REM * LE PROGRAMME EST INDEPENDANT                  *
160 REM * DU FICHIER                                    *
190 REM *                                              *
280 REM *                                              *
300 REM * LE PROGRAMME PERMET LES ORDRES SUIVANTS :     *
310 REM *  LECTURE ,CREATION ,DESTRUCTION,MODIFICATION  *
320 REM *  ET LECTURE SEQUENTIELLE (SUIVANT)            *
360 REM *                                              *
370 REM * LE PROGRAMME UTILISE 2 MASQUES                *
380 REM * - CLE-XXXXX = DEMANDE DE LA CLE ET DE L'ORDRE *
390 REM * - ART-XXXXX = DEMANDE OU AFFICHAGE DE LA PIECE*
400 REM *                                              *
410 REM *  LES DIFFERENCES AVEC LE PROGRAMME N'ACCEPTANT*
420 REM *  PAS LES HOMONYMES SONT:                     *
430 REM *    - POUR CREER UN ARTICLE ,ADD REMPLACE WRITE *
440 REM *    - POUR METTRE A JOUR ,NE PAS FAIRE READ     *
450 REM *      AVANT NEXT .CELA PERMET DE LIRE PAR NEXT  *
460 REM *      LES DIFFERENTS ARTICLES DE MEME CLE .     *
490 REM ************************************************
500 LET"#CLEAR-$":       INPUT"NOM DU FICHIER ";N$
510 LET"#OPEN,F,FICHIER,0:"+N$
520 LET"#OPEN,C,MASQUE,0:CLE-"+N$
530 LET"#OPEN,A,MASQUE,0:ART-"+N$
540 LET"CHARGE,C
550 LET"OUTPUT,C":MC$=""
560 LET"INPUT,C":IF  WS    THENMC$="REPRENEZ":GOTO550
570 IFO$="F"THENEND
580 IFO$="L"THEN700
590 IFO$="S"THEN800
600 IFO$="M"THEN900
610 IFO$="C"THEN1000
620 IFO$="D"THEN1100
630 MC$="ORDRE ERRONE":GOTO550
700 LET"READ-F":IF  WS    THEN790
710 MA$="LECTURE:FAITE RETURN":LET"PRINT,A
720 GETA$:IFA$<>CHR$(13)THEN720
730 GOTO540
790 MC$="ARTICLE INEXISTANT":GOTO550
800 LET"NEXT-F":IF  WS    THENMC$="PLUS D'ARTICLE":GOTO550
820 GOTO710
900 LET"READ-F":IF  WS    THEN790
910 MA$="MODIFICATION":LET"PRINT-A":LET"INPUT-A":IF  WS    THEN990
920 LET"UPDATE-F":GOTO540
990 MC$="ABANDON":GOTO540
1000 LET"READ-F":IF  WS    =0THENMC$="ARTICLE EXISTANT":GOTO550
1010 MA$="CREATION":LET"PRINT-A":LET"INPUT-A":IF  WS    THEN990
1020 LET"ADD-F":GOTO540
1100 LET"READ-F":IF  WS    THEN790
1110 MA$="OK POUR DETRUIRE (O/N)":LET"PRINT-A"
1120 GETA$:IFA$="N"THEN990
1130 IFA$<>"O"THEN1120
1140 LET"DELET-F":GOTO540
READY.
```

```
100 REM ************************************************************
110 REM * EXEMPLE 11                                                *
120 REM *                                                          *
130 REM * TRI DE FICHIER                                           *
140 REM * --------------                                           *
150 REM *                                                          *
160 REM * ------------------------------------------------------   *
170 REM * CLEAR-$  NEXT  REORGANISE  ADD  NEW                      *
180 REM * ------------------------------------------------------   *
190 REM *                                                          *
200 REM * FICHIER D'ORIGINE : FICH ORIG                            *
210 REM *                                                          *
230 REM *  CLE : A$10 ,AA                                          *
240 REM *  ENREGISTREMENT : BB ,CC ,DD                             *
250 REM *                                                          *
260 REM * FICHIER D'ARRIVEE : FICH ARR                             *
270 REM *                                                          *
280 REM *  CLE : CC ,DD                                            *
290 REM *  ENREGISTREMENT : A$ ,BB ,AA                             *
300 REM *                                                          *
310 REM * PRINCIPE DU TRI :                                        *
320 REM *  CREATION DANS UN NOUVEAU FICHIER SEQUENTIEL             *
330 REM *  INDEXE .LE PROGRAMME REORGANISE LE NOUVEAU FICHIER       *
340 REM *  DES QUE LE NOMBRE D'ENREG.  CREES ATTEINT LE NOMBRE      *
350 REM *  D'ENREGISTREMENT DANS LE FICHIER (SAUF AU DEBUT OU       *
360 REM *  LES INSERTIONS SONT RAPIDES )                           *
370 REM *                                                          *
380 REM ************************************************************
400 LET"#CLEAR-$"
410 LET"#OPEN,ORI,FICHIER,0:FICH ORIG
420 LET">CC,DD=A$,BB,AA
430 LET"#NEW,ARR,FICHIER,0:FICH ARR
440 NRANGES=0:NTAS=0
450 LET"NEXT-ORI":IF  WS    THEN PRINT"TRI TERMINE":END
460 LET"ADD-ARR":NTAS=NTAS+1:IF NTAS < NRANGES THEN 450
470 IF NRANGES=0 AND NTAS<30 THEN 450
480 LET"#REORG-ARR":NRANGES=NRANGES+NTAS:NTAS=0:GOTO450
READY.
```

CHAPITRE VI FONCTIONS COMPLEMENTAIRES

6 - 1 CONTENU D'UN DISQUE

Le MEM/DOS permet de connaitre la place libre sur une disquette.

Utilisez pour cela l'ordre :

```
LET " % d "      d = numéro de drive
```

Par exemple :

```
LET " % Ø "
```

Pour connaitre la place disponible, lire l'octet situé à l'adresse 189 (habituellement utilisé comme status)
Cet octet contient le nombre de pistes disponibles sur la disquette.

Remarque : En mode direct si le nombre de pistes libres est non nul, le message ? DIRECT ERROR apparait, mais cela n'a pas d'importance.

Le nombre total de pistes de la disquette est 34.

Si par exemple vous obtenez le résultat

```
? PEEK  WS
17
```

La disquette est occupée à 50 %.

6 - 2 CHANGEMENT DE DISQUETTES

Pour accélérer le fonctionnement du MEM/DOS , certains pointeurs décrivent la disquette logiquement. Si vous changez la disquette sans que le système le sache, vous risquez alors de détruire le contenu de celle-ci.
Pour signaler le changement de disquette, utiliser l'ordre :

```
      LET "# C," + CHR$(d)      d = numéro de drive
   ou LET"# C,$d"               d = numéro de drive
```

De plus, tous les fichiers ouverts sur le drive devront être fermés.

REMARQUE : En mode direct, l'ordre LET "# C," + CHR$(d) est effectué automatiquement, avant chaque ordre.

Par exemple : En mode direct

```
LOAD " Ø : TEST "
```

Changement de disquette

```
SAVE " Ø : TEST "
```

ne détruit pas la disquette

En mode programme

```
10 LOAD " Ø : TEST "
20 INPUT " CHANGEZ LA DISQUETTE ET FAITES RETURN " ; A$
30 SAVE " Ø : TEST
```

détruit la disquette.

Il faut faire :

```
10 LOAD " Ø : TEST "
20 INPUT " CHANGEZ LA DISQUETTE ET FAITES RETURN " ; A$
30 LET " #C, " + CHR$ (Ø)
40 SAVE " Ø : TEST "
```

NOTE : Après LET " #C,$ qui nettoie tous les buffers du DOS, il est possible de changer de disquette sans risque.

CAS PARTICULIER : LET"#C,$$" ferme tous les drives.

L'ordre LET ")M " supprime le CLEAR en mode direct (ce qui permet d'utiliser les fichiers), mais le contrôle n'est plus effectué.
LET ") " rend le CLEAR.

6 - 3 FONCTIONS DE CALCUL SUR 48 CHIFFRES

Le MEM/DOS contient des fonctions de calcul qui sont une aide précieuse pour les applications de gestion comme la comptabilité.

Il y a trois fonctions :

- addition
- soustraction
- arrondi

Ces fonctions manipulent des chaines de caractères.

La précision des calculs est de 48 chiffres

Exemple :

```
LET " = 325.1 + 3.21 "
PRINT WW$
328.31
```

La syntaxe générale est :

```
LET " = op1 ± op2 "
Résultat dans WW$
```

ou

```
LET " = " + 01$ + " + " +02$  (addition)
LET " = " + 01$ + " - " +02$  (soustraction)
LET " = " + 01$               (arrondi)
```

6 - 4 PRECISIONS DES CALCULS

Les calculs se font avec un nombre de décimales fixé par l'ordre :

```
LET "Ω - n "     n = nombre de décimales (1 à 9)
```

Exemple

```
LET "Ω - 2 "
```

Par défaut, au démarrage du système, le nombre de décimales est 2.
Les arrondis sont faits à la valeur inférieure.

Exemples :

```
 5 REM PROGRAMME DE CALCUL
10 INPUT " OPERANDE 1 ? " ; 01$
20 INPUT " OPERANDE 2 ? " ; 02$
30 INPUT " OPERANDE (+/-) " ; OP$
40 LET " = "+  01$ + OP$ + OP$
50 PRINT " RESULTAT = ";WW$

] LET "≤3 + 2.2 " : PRINT WW$
5.2
] LET "Ω - 2 "
] LET " = 3.234 + 1.001 " : PRINT WW$
4.23
] LET " = 3.1234 " : PRINT WW$
3.12
] LET " = 1234567890123456789.00000 "
1234567890123456789
```

Le résultat dans WW3 est toujours présenté sous la forme la moins couteuse en place.

```
    Ø  :  vide
 1.20  :  1.2
```

6 - 5 SAISIE ET AFFICHAGE

Cet outil de calcul ne serait pas utilisable s'il n'était complété par les fonctions complémentaires de saisie et d'affichage.
Ces fonctions existent dans le MEM/DOS .

Dans un masque, définissez une zone comme :

```
" A3 + "
```

Elle sera considérée comme un numérique au niveau de la saisie, et comme alphanumérique pour le stockage.

L'affichage se fait en format gestion, cadré à droite en décimale fixe. Le nombre de décimales est fixé par le même ordre.

```
LET "Ə - n "
```

Note : pour l'affichage gestion, se reporter à ce chapitre dans la documentation.

6 - 6 AZERTY

De nombreux utilisateurs, habitués aux machines à écrire, demandent un clavier du même type sur leur micro-ordinateur.
Le MEM/DOS permet de résoudre ce problème pour l'utilisation de l'ordre :

```
LET " Z "
```

qui échange la signification de certaines touches pour obtenir l'ordre classique AZERTY.

Pour cela, il est nécessaire d'échanger sur le clavier les capuchons des touches :

```
A       Q
W       Z
M       ;
```

et d'ajouter l'ordre LET " Z " dans le programme HELLO qui est executé à la mise en route de votre système.

Remarque : Si vous enfoncez accidentellement la touche RESET, faites

```
CTRL/C RETURN
LET " Z "
```

pour reprendre la main.

Attention ! pour taper LET " Z " faites W au lieu de Z, dans ce cas.

6 - 7 EXECUTE

L'execute est une fonction issue du langage APL qui permet de construire et d'executer par programme un certain nombre d'instructions.

Le MEM/DOS vous offre cette possibilité.

L'opération se fait en deux temps :

1) Enregistrement des instructions par :

```
LET " >          "    pour entrer le texte
LET " +          "    pour en ajouter
```

Note : Il faut remplacer les cotes (") par CHR$ (34) et finir les lignes par CHR$ (13) sauf la dernière.

Exemple :

```
LET " > HOME : VTAB10 " + CHR$(13)
LET " + PRINT " + CHR$ (34) + " BONJOUR "
```

2) Execution : pour executer ces instructions, faites:

```
LET " ! "  : END
```

Après l'éxecution, le buffer est vidé.

Vérification :
L'ordre LET " V " permet de relire le texte des ordres enregistrés par LET " >..." et LET " +..." sans les détruire.

Dans notre exemple après le premier temps,

```
LET " V "
HOME : VTAB10
PRINT " BONJOUR
```

Remarque : Les ordres sont executés comme s'ils étaient tapés en mode direct. En particulier les erreurs s'afficheront même si un ordre ON ERR GOTO a été inséré dans le programme.

Exemple : Programme chargeant et listant un certain nombre de programmes donnés par des DATA, (# indique la fin de la liste.)

```
 5 LET " > " : REM VIDE LE BUFFER
10 READ N$ : IF N$ = "#" THEN 60
20 LET " + LOAD " + CHR$(34) + N$ + CHR$(13)
30 LET " + PRINT " + CHR$(34) + N$ + CHR$(13)
40 LET " + LIST " + CHR$(13)
50 GOTO 10
60 LET " + PR#0 : NEW
70 PR#2 : LET " ! "
90 DATA UTIL, AUTO COPIE, DEMO, HGR,#
```

3) L'exécute et ses réalisations

L'ordre LET " ! " du MEM/DOS permet de modifier l'entrée clavier. Les caractères ne sont plus lus au clavier mais dans un buffer du DOS.

EXEMPLE :

```
10 LET " > XYZ "
20 LET " + ABC "
30 LET " ! "
40 INPUT A$
50 ? A$
RUN
XYZABC
```

Les caractères peuvent être relus par GET ou INPUT.

APPLICATIONS

1) Passage de paramètres d'un programme à un autre :

Vous désirez passer les variables A$ et B$. Faire :

```
        20 LET " > " + A$ + CHR$(13) + B$
        30 RUN " TOTO "
        - - - - - - -
TOTO    LET " ! "
        INPUT A$,B$
        -
        -
        - etc ...
```

Lorsque tous le buffer a été relu, le GET redevient normal.

2) Lecture des dictionnaires

```
10 LET " #0,F,F,FILE "
20 LET " E,F "
30 LET " ! "
40 GET A$ : IF A$ = " 2 " THEN
   ? " FICHIER RELATIF "
```

Dans cet exemple, on teste le premier caractère du dictionnaire, pour savoir si le fichier est relatif.

3) Exécution d'ordre BASIC

On désire exécuter une formule de calcul saisie au clavier.

```
10 INPUT " Y = ? " ; A$
20 LET " > Y = " + A$ + " :GOTO 100 " +CHR$(13)
30 LET " + GOTO 200 "
35 LET"!"
40 END
100 ? " RESULTAT " Y : GOTO 10
200 ? " ERREUR " : GOTO 10
```

Le END rend la main au BASIC qui exécute l'ordre.
Les lignes tapées sont équivalentes à :

```
] Y = (.....) : GOTO 100
] GOTO 200
```

En cas d'erreur, la première ligne n'est pas exécuter entièrement, et le GOTO 100 ne se fait pas.

4) Empêcher les demandes clavier pendant le CATALOG

LET " * " demande d'appuyer sur une touche s'il y a plus de 20 lignes. Pour empêcher ce phénomène, faire :

```
10 LET "  > AAAAAAAA
20 LET " ! "
30 LET " * "
40 IN#Ø
```

Le IN Ø interroge le EXECUTE car on ne sait pas s'il ne reste pas de " A " à prendre.

CHAPITRE VII PARAMETRAGES DU MEM/DOS

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

La carte MEM/DOS présente un certain nombre de particularités au niveau de son implantation et de son adaptabilité.
Elle est théoriquement adaptable :

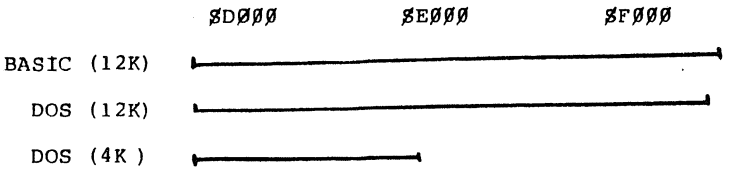
- à toute carte 80 colonnes
- à tout terminal connecté en lieu et place de la vidéo et du clavier d'origine ayant l'écran projeté en mémoire
- à toute mémoire de masse qu'elle peut gérer par tranches allant jusqu'à 16 méga octets.

7 - 1 IMPLANTATION

La mémoire de l'APPLE II se décompose de la façon suivante :

$0000 - $BFFF	RAM
$C000 - $CFFF	Entrées / sorties
$D000 - $F7FF	BASIC (ROM)
$F800 - $FFFF	Moniteur

Une particularité essentielle de l'APPLE II est la possibilité de déconnecter les ROM Basic et MONITEUR pour les remplacer par d'autres mémoires externes.
C'est cette possibilité qu'utilise la carte MEM/DOS . Les programmes assembleurs du DOS sont donc en parallèle avec le Basic et le Moniteur. Le Basic et le Moniteur occupent ensemble une place mémoire égale à 12K. Le MEM/DOS occupe lui 16K. Il a donc fallu décomposer cet espace de 16K en deux parties elles-mêmes superposées.



Les commutations entre le BASIC et les différentes pages du MEM/DOS se font par un logiciel intégré dans la carte. Cette opération est transparente pour l'utilisateur, ce qui signifie que pour lui, le Basic et le DOS fonctionnent sans qu'il ne s'aperçoive de rien.

ORTEZ-VOUS AUX ECIFICITES APPLE IIc

7 - 2 PRINCIPE DE LA COMMUTATION

La commutation se fait de la façon suivante :

Notation PAGE Ø = 1ère page de 4K de $D000 à $DFFF du MEM/DOS

PAGE 1 = 2ème page de 4K de $D000 à $DFFF du MEM/DOS

RACINE = reste du DOS (8K) de $E000 à FFFF

BASIC = 12K de $D000 à $FFFF

Ces quatre parties contiennent bien sûr des informations différentes bien qu'elles soient aux mêmes adresses.

Soit X le numéro du slot où la carte a été installée.
Des commutations se font en écrivant à l'adresse $CXFF

```
LDA # $80
STA$CXFF ..................... BASIC

LDA # $00
STA$CXFF ..................... PAGE 0 + RACINE

LDA # $01
STA$CXFF ..................... PAGE 1 + RACINE
```

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

7 - 3 RESET ET INTERRUPTIONS

La mise en route ou le RESET se font dans les ROM de la carte MEM/DOS . Un programme s'y trouve donc pour gérer les interruptions.

RESET : le programme ramène au RESET de la ROM MONITEUR de l'APPLE. La présence de la carte est donc tranparente.

NMI : L'interruption NMI ramène à l'indirection habituelle en page $3

IRQ : Cette interruption sert à la fois aux interruptions extérieures masquables ainsi qu'au BREAK logiciel.
Lorsque la carte est connectée et sélectée (en page 0 et 1) le BREAK est impossible. L'IRQ ramène donc toujours à l'indirection prévue à cet effet en page $3.

IMPORTANT : Si la carte est sélectée, ce sont les vecteurs d'interruption de la carte qui servent; si elle est désélectée (mode BASIC), ce sont les vecteurs d'origine (ROM MONITEUR). Cependant, au moment d'une interruption, (sauf RESET), l'état reste inchangé et vous pouvez aussi bien être en mode BASIC qu'en mode DOS. Si vous désirez dans vos programmes d'interruptions, utiliser des fonctions MONITEUR vous devez :

1) chercher en quel état vous êtes. Pour cela, lisez un octet dans la page $DXXX à un endroit où les 3 versions sont différentes.

2) commuter vers BASIC par :

```
LDA   $8Ø
STA $CXFF
```

3) avant de finir l'interruption; reconnectez, si besoin est, la PAGE précédente par :

```
LDA   $ØØ ou $Ø1
STA $CXFF
```

7 - 4 L'OCCUPATION RAM DU MEM/DOS

Pour fonctionner, le MEM/DOS utilise les zones RAM suivantes :

a) .$Ø3ØØ - $Ø3CF

b) .$Ø250 - $Ø2FF

La zone a) en page 3 ne doit pas être modifiée. Par contre, la zone en page 2 est composée de variables de travail et peut être détruite entre deux ordres. (Elle l'est d'ailleurs car cette zone sert également de BUFFER pour INPUT.)
D'autre part, le DOS a besoin d'autres places en RAM pour :

1) Créer des objets en mémoire :

- descriptifs de disques
- descriptifs de fichiers
- masques
- ...

Cette zone sera appelée BUFFERS du DOS.

2) Accéder aux périphériques

Il faut prévoir en RAM les programmes permettant de lire, écrire ou formatter un disque. Dans le cas des lecteurs 5 pouces, il s'agit du programme RWTS fourni avec les disquettes.

Ces deux zones RAM ne sont pas en place fixe comme celle en page 2 et 3. L'utilisateur choisit lui-même suivant la version désirée leur implantation. En page 3, on trouve un certain nombre de pointeurs indiquant au DOS l'endroit où se trouvent ces zones.

7 - 5 POSITION DES BUFFERS DU DOS

Note : Les pointeurs sont toujours dans l'ordre poids fort, poids faible.

$3ØØ - $3Ø1 = début des buffers du DOS

$3Ø3 - $3Ø4 = fin de la première partie

$3Ø5 - $3Ø6 = pointeur courant (indique le début de la place non encore occupée dans les buffers)

$3Ø7 - $3Ø8 = fin des buffers du DOS

Pour mettre en place ces pointeurs, choisissez les valeurs mini et maxi que vous réservez à ces adresses et placez-les en :

$3ØØ - $3Ø1
$3Ø7 - $3Ø8

placez en $3Ø3 - $3Ø4 les mêmes valeurs qu'en $3Ø7 - $3Ø8

placez en $3Ø5 - $3Ø6 l'adresse correspondant à celle indiquée en $3ØØ - $3Ø1 plus $124.

```
EXEMPLE : $3ØØ : 9Ø  }
          $3Ø2 : ØØ  }  $9ØØØ
          $3Ø2 :
          $3Ø3 : B6  }
          $3Ø4 : ØØ  }  $B6ØØ
          $3Ø5 : 91  }
          $3Ø6 : 24  }  $9124 = $9ØØØ + $124
          $3Ø7 : B6  }
          $3Ø8 : ØØ  }  $B6ØØ
                        $AFFF sur IIc
```

Dans cet exemple, les parties réservées aux BUFFERS du DOS s'étendent de :

$9ØØØ à $B6ØØ (inclus)
$AFFF sur IIc

7 - 6 LES ACCES AUX PERIPHERIQUES

Le MEM/DOS permet de gérer simultanément des périphériques de types différents. Pour décrire ces périphériques, un certain nombre de tables sont prévues. Ces tables indiquent pour chaque périphérique :

- le nombre de pistes
- le nombre de têtes
- le nombre de secteurs
- l'adresse du programme permettant d'y accéder

MEM/DOS permet d'utiliser dans une même configuration 6 périphériques.
Par exemple : - Un disque dur à 3 drives
- 4 lecteurs 5 pouces

Pour chaque périphérique (numéroté de Ø à 5), dont nous appellerons n le numéro, indiquez :

en $31Ø + n le nombre de pistes - 1

en $316 + n le nombre de têtes

en $31C + n le nombre de secteurs

en $322 + 2 n l'adresse (poids fort, poids faible) du programme d'accès (HANDLER)

Si l'adresse correspondant au HANDLER est remplacée par $ØØ, $ØØ, le périphérique sera considéré comme inexistant.

7 - 7 CARACTERISTIQUES DU PROGRAMME D'ACCES

Le programme gérant le périphérique doit être capable de lire, écrire ou formatter un disque d'après les paramètres suivants :

$3Ø9 numéro du périphérique (permet de différencier les actions pour plusieurs périphériques pour lesquels l'adresse du HANDLER est la même)

$3ØA piste choisie

$3ØB tête choisie

$3ØC secteur choisi

$3ØD commande : 1 - lecture
2 - écriture
3 - formattage

$3ØE poids fort de l'adresse du bloc

$3ØF poids faible de l'adresse du bloc

Il est inutile d'effectuer des contrôles de validité, ceux-ci sont faits par le MEM/DOS avant de faire appel à ces fonctions. Vous trouverez ci-après les versions permettant de faire fonctionner les disquettes 5 pouces APPLE 110K ou 140K en utilisant le programme RWTS.

IMPORTANT : En ce qui concerne le formattage, il est nécessaire que tous les secteurs puissent être lus, même s'ils n'ont pas été remplis. En conséquence, vous constaterez que dans la version 110K, le formattage est suivi de l'écriture de tous les secteurs de la disquette, car, en formattage 110K, la lecture est impossible sans écriture préalable.
Le cas se présente également pour les cartouches du disque CII D140.

```
s......page 0001

a# loc   code           line


1  0000                        * = $b700
2  b700                 ;
3  b700                 drive  = $309
4  b700                 cylind= $30a
5  b700                 tete   = $30b
6  b700                 secteu= $30c
7  b700                 comman= $30d          ; 1 2 ou 4
8  b700                 adblc  = $30e         ;adresse du bloc poids fort / poids faible
9  b700                 ;
0  b700                 ;
1  b700                 ; acces au peripherique ( version 110k )
2  b700                 ;
3  b700                 diob=$b7e8
4  b700                 ;
5  b700  a2 60          vrwts  ldx #$60       ;calcul du slot
6  b702  ad 09 03              lda drive
7  b705  c9 02                 cmp #2
8  b707  90 02                 bcc vrwts1
9  b709  a2 50                 ldx #$50
0  b70b  8e e9 b7       vrwts1 stx diob+1
1  b70e  ae 0d 03              ldx comman
2  b711  8e f4 b7              stx diob+12
3  b714  ae 0a 03              ldx cylind
4  b717  8e ec b7              stx diob+4
5  b71a  a2 01                 ldx #1
6  b71c  4a                    .byte $4a             ; lsr a
7  b71d  90 01                 bcc rwts07
8  b71f  e8                    inx
9  b720  8e ea b7       rwts07 stx diob+2
0  b723  ad 0c 03              lda secteu
1  b726  8d ed b7              sta diob+5
2  b729  a9 00                 lda #0
3  b72b  8d eb b7              sta diob+3            ;voluma
4  b72e  ad 0e 03              lda adblc
5  b731  8d f1 b7              sta diob+9
6  b734  ad 0f 03              lda adblc+1
7  b737  8d f0 b7              sta diob+8
8  b73a  a9 b7                 lda #>diob
9  b73c  a0 e8                 ldy #<diob
0  b73e  20 b5 b7              jsr $b7b5
1  b741  b0 08                 bcs vrw005
2  b743  ad 0d 03              lda comman
3  b746  c9 04                 cmp #4                ; format ?
4  b748  f0 02                 beq vrw001
5  b74a  18                    clc
6  b74b  60             vrw005 rts
7  b74c                 ;
8  b74c  a9 00          vrw001 lda #0
9  b74e  8d 8f b7              sta cylcyl
0  b751  a9 00          vrw004 lda #0
1  b753  8d 8e b7              sta secsec
2  b756  a9 02          vrw003 lda #2
3  b758  8d f4 b7              sta diob+12
4  b75b  a9 08                 lda #8
5  b75d  8d f0 b7              sta diob+8
```

```
rwts.....page 0002

line# loc   code          line

0056  b760  8d f1 b7          sta diob+9
0057  b763  ad 8f b7          lda cylcyl
0058  b766  8d ec b7          sta diob+4
0059  b769  ad 8e b7          lda secsec
0060  b76c  8d ed b7          sta diob+5
0061  b76f  a9 b7             lda #>diob
0062  b771  a0 e8             ldy #<diob
0063  b773  20 b5 b7          jsr $b7b5
0064  b776  b0 15             bcs vrw002
0065  b778  ee 8e b7          inc secsec
0066  b77b  ad 8e b7          lda secsec
0067  b77e  c9 0d             cmp #13
0068  b780  90 d4             bcc vrw003
0069  b782  ee 8f b7          inc cylcyl
0070  b785  ad 8f b7          lda cylcyl
0071  b788  c9 23             cmp #35
0072  b78a  90 c5             bcc vrw004
0073  b78c  18                clc
0074  b78d  60          vrw002 rts
0075  b78e                   ;
0076  b78e  00          secsec .byte 0
0077  b78f  00          cylcyl .byte 0
0078  b790                   ;
0079  b790                   .end

errors = 0000

symbol table

symbol  value
 adblc   030e   comman  030d   cylcyl  b78f   cylind  030a
 diob    b7e8   drive   0309   rwts07  b720   secsec  b78e
 secteu  030c   tete    030b   vrw001  b74c   vrw002  b78d
 vrw003  b756   vrw004  b751   vrw005  b74b   vrwts1  b70b

end of assembly
```

7 - 8 DESCRIPTION DE LA GESTION D'ECRAN

La carte MEM/DOS est prévue pour pouvoir adapter la gestion de masques à n'importe quel type de carte 80 colonnes ou terminal intelligent. Pour cela, un certain nombre de paramètres sont à mettre en place pour décrire l'écran.

$3 68	adresse de l'écran (poids fort, poids faible)
$3 6A	longueur de l'écran (poids fort, poids faible)
$3 6C	nombre de caractères par lignes

Un certain nombre de fonctions sont également à créer pour gérer l'écran. Elles utilisent des pointeurs en page Ø notés :

VV ($DB,$DC)
WW ($DD,$DE)

Le premier (VV) indique la position du curseur (poids faible, poids fort).
Le second (WW) indique le nombre de caractères séparant le curseur du premier caractère de l'écran (poids faible, poids fort).
WW évolue donc entre Ø et la largeur de l'écran - 1.
La position du curseur peut être fictive ou réelle. Dans la version 40 colonnes fournie en standard, VV correspond à l'adresse réelle en mémoire de l'octet se rapportant au caractère de l'écran.
La valeur initiale de VV (caractère en haut à gauche de l'écran) doit correspondre à la valeur indiquée dans le paramètre " adresse de l'écran ".
A chaque opération concernant les masques, le MEM/DOS initialisera VV à cette valeur et WW à zéro.
D'autre part, le MEM/DOS doit savoir se déplacer sur cet écran qui lui est inconnu. Il doit pouvoir aussi lire ou écrire sur cet écran. Il doit finalement pouvoir convertir les caractères lus en ASCII et inversement.

Nous noterons :

CODE ECRAN : la valeur qu'il faut écrire sur l'écran un caractère dont on connait le code ASCII.

Il faut donc fournir au MEM/DOS les fonctions suivantes :

AVANCE : calcule les nouvelles valeurs de VV et WW lorsque l'on avance de 1 caractère sur l'écran. Le carry sera égal à 1 en sortie si l'on dépasse la fin de l'écran. (WW = longueur d'écran) (la longueur de la zone est en NC = ∅26A sur deux octets poids fort, poids faible).

RECULE : calcule les nouvelles valeurs de VV et WW lorsque l'on recule de 1 caractère sur l'écran. Le carry sera égal à 1 en sortie si l'on dépasse le début de l'écran. (Il s'agit du cas où (WW) = ∅ en entrant dans le programme RECULE.)

INCAR : renvoie dans l'accumulateur le code écran du caractère situé en VV.

OUTCAR : écrit le caractère dont le code écran est dans l'accumulateur à l'endroit du curseur pointé par VV.

E A : convertit dans l'accumulateur un code écran en code ASCII.

A E : convertit dans l'accumulateur un code ASCII en code écran.

INVERS : convertit dans l'accumulateur un code écran en code écran du caractère sur fond blanc correspondant.

FLASH : convertit dans l'accumulateur un code écran en code écran du caractère clignotant correspondant. (Si cette fonction n'existe pas, remplacez-la, par exemple, par INVERS.)

Vous devez écrire les programmes assembleurs réalisant ces fonctions et les implanter en mémoire. Pour indiquer au MEM/DOS leurs positions remplissez l'ensemble des JMP prévues à cet effet.

```
$36F    JMP AVANCE
$372    JMP RECULE
$384    JMP INCAR
$387    JMP OUTCAR
$378    JMP EA
$375    JMP AE
$37E    JMP INVERS
$381    JMP FLASH
```

NOTE : Pour simplifier votre programmation, sachez que lorsque le MEM/DOS se branche à l'INCAR ou OUTCAR, le registre y est nul. Donc, si VV est l'adresse réelle du caractère, on pourra choisir :

```
INCAR   LDA (VV), y
        RTS

OUTCAR  STA (VV), y
        RTS
```

que l'on pourra placer directement à la place des JMP correspondants pour accélérer, car dans les deux cas, la longueur est 3 octets. Ci-joint la version 40 colonnes de base contenu dans la ROM.

IMPORTANT : Lors de l'appel de ces fonctions, le MEM/DOS ne reconnecte pas le BASIC (pour gagner du temps), donc vous ne pouvez pas :

- utiliser les fonctions moniteurs
- tester vos programmes en utilisant l'instruction BRK (code $00).

Essayez d'écrire des programmes les plus performants possibles car dans l'exemple d'une insertion de caractères sur tout l'écran en 80 colonnes, on aura :

- 2 000 appels à AVANCE
- 2 000 appels à RECULE
- 2 000 appels à INCAR
- 2 000 appels à OUTCAR

```
ready.

 1000 ; version de base en 40 colonnes
 1010 ;
 1020 ; ae40c
 1030 ; inv40c
 1040 ; fla40c
 1050 ; get40c
 1060 ; ea40c
 1070 ; av40c
 1080 ; rec40c
 1090 ;
 1100 ;
 1110 ;ascii - ecran
 1120 ;
 1130 ae40c  and #63
 1140        ora #128
 1150        rts
 1160 ;
 1170 ; inverse video
 1180 ;
 1190 inv40c and #63
 1200        rts
 1210 ;
 1220 ; flash video
 1230 ;
 1240 fla40c and #63
 1250        ora #64
 1260 rts
 1270 ;
 1280 ;get un caractere avec spot en (vv)
 1290 ;
 1300 get40c ldy #0
 1310        jsr incar
 1320        sta n4
 1330        jsr flash
 1340        jsr outcar
 1350 ge1
 1360 et95   bit 49152
 1370        bpl ge1
 1380        lda 49152
 1390        bit 49168
 1410 geqwer and #$7f
 1420        tax
 1430        lda n4
 1440        jsr outcar
 1450        txa
 1460        rts
 1470 ;
 1480 ;conversion e a
 1490 ;
 1500 ea40c  and #127
 1510        cmp #32
 1520        bcs etq36
 1530        ora #64
 1540 etq36  rts
 1550 ;
 1560 ;
 1570 ;
 1580 ;avance /recule de un caractere
 1590 ;
 1600 av40c  lda ww+1
```

```
1610          cmp nc
1620          bcc avok
1630          lda ww
1640          cmp nc+1
1650          bcc avok
1660          rts
1670 avok     lda vv+1
1680          cmp #7
1690          bne av1
1700          lda vv
1710          cmp #$a7
1720          beq av2
1730          cmp #$cf
1740          bne av1
1750 av2      sec
1760          sbc #216
1770          sta vv
1780          lda vv+1
1790          sbc #3
1800          sta vv+1
1810          av1 lda vv
1820          and #127
1830          cmp #$27
1840          beq av3
1850          cmp #$4f
1860          beq av3
1870          cmp #$77
1880          bne av4
1890 av3      lda vv
1900          clc
1910          adc #88
1920          sta vv
1930          bcc av4
1940          inc vv+1
1950 av4      inc ww
1960          bne etq21
1970          inc ww+1
1980 etq21    inc vv
1990          bne etq7
2000          inc vv+1
2010 etq7     clc
2020          rts
2030 ;
2040 ;recule de 1 caractere"
2050 ;
2060 rec40c   ldx ww+1
2070          bne reok
2080          ldx ww
2090          cpx #1
2100          bne reok
2110          sec
2120          rts
2130 reok     lda vv+1
2140          cmp #4
2150          bne re1
2160          lda vv
2170          cmp #$28
2180          beq re2
2190          cmp #$50
2200          bne re1
2210 re2      clc
2220          adc #128
```

```
2230          sta vv
2240          lda vv+1
2250          adc #3
2260          sta vv+1
2270          jmp re4
2280 re1      lda vv
2290          and #127
2300          beq re3
2310          cmp #$28
2320          beq re3
2330          cmp #$50
2340          bne re4
2350 re3      lda vv
2360          sec
2370          sbc #88
2380          sta vv
2390          bcs re4
2400          dec vv+1
2410 re4      dec ww
2420          ldx ww
2430          cpx #255
2440          bne etq12
2450          dec ww+1
2460 etq12    dec vv
2470          ldx vv
2480          cpx #255
2490          bne etq9
2500          dec vv+1
2510 etq9     clc
2520          rts
2530 ;
2540 .end

ready.
```

7 - 9 MODIFICATIONS DE L'ENTREE CLAVIER

L'accès clavier dans les saisies par masques se fait par l'intermédiaire d'une indirection en page 3. Pour modifier cette fonction, placez dans l'indirection l'adresse de votre programme (poids faible, poids fort)

```
ø37B     JMP  GET
```

Le programme doit renvoyer dans l'accumulateur soit le code ASCII, soit le code ASCII + 128.

NOTE : La fonction AZERTY pourra être utilisée quelque soit la fonction GEØT que vous avez créée.

7 - 10 AJOUT DE NOUVELLES COMMANDES AU MEM/DOS

Vous pouvez, si vous le désirez, ajouter vous-même des fonctions au DOS. Pour cela, il existe une indirection sur deux octets (poids faible, poids fort) indiquant l'adresse du programme analysant les commandes. Si vous voulez ajouter une commande :

1) sauvez l'adresse précédemment placée dans cette indirection.

2) mettez l'adresse de votre programme.

3) à la fin de l'éxécution de votre programme, si la commande ne vous concernait pas, rebranchez-vous à l'adresse que vous avez sauvée si elle n'est pas nulle.

Cette méthode permet de faire fonctionner ensemble plusieurs programmes ajoutant de nouvelles commandes.

NOTE : Dans la version de base, l'indirection contient Ø,Ø. Dans ce cas, le MEM/DOS ignore l'indirection.

L'adresse de l'indirection est :

- ø38A poids faible
- ø38B poids fort

Lorsque vous prenez la main dans votre programme :

- l'accumulateur contient le premier caractère de la commande
- AA(ø272) contient la longueur de l'instruction
- yy(øE1,øE2) contient l'adresse de l'instruction
- le registre y pointe vers le caractère en cours dans l'instruction par rapport à yy.

4 fonctions peuvent vous être utiles :

JSR CARAC : - renvoie dans l'accumulateur le prochain caractère non blanc de la commande et met à jour y.

- si le carry est 1, on a atteint la fin de la commande.

JSR SEPARA : - renvoie le prochain caractère non blanc de la commande suivant un séparateur. (, : -) dans les mêmes conditions.

JSR VALEUR : - renvoie une valeur sur 1 octet lu dans la commande en décimal à partir de l'octet en cours.

JSR GETHX1 : - idem pour une valeur sur deux octets saisis sous la forme øXXXX.
En sortie X = poids faible.
A = poids fort.

EXEMPLE : La commande est : LET"H,valeur,une lettre,héxadécimal"
LET"H,12,A,ø25FØ"

```
cmp   'H
bne   NON
jsr   SEPARA
bcs   ERR
sta   VAL
jst   SEPARA
bcs   ERR
                        sta   LETTRE
                        jsr   SEPARA
                        bcs   ERR
                        jsr   GETHX1
                        stx   HEXA
                        sta   HEXA + 1
                        jmp   OK
                  NON   rts
```

```
ready.

 1000 carac   cpy aa
 1010         bcc ee72
 1020         rts
 1030 ee72    lda (yy),y
 1040         iny
 1050         cmp #'
 1060         beq carac
 1070         clc
 1080         rts
 1090 ;
 1100 ;
 1110 separa jsr carac
 1120         bcc ee80
 1130         rts
 1140 ee80    cmp #'          ;
 1150         beq carac
 1160         cmp #'-
 1170         beq carac
 1180         cmp #58
 1190         beq carac
 1200         cmp #44
 1210         beq carac
 1220         bne separa
 1230 ;
 1240 ;
 1250 valeur  cmp #58
 1260         bcs val1
 1270         cmp #48
 1280         bcc val1
 1290         sbc #48
 1300         sta nuacc
 1310 va333   jsr carac
 1320         bcs val2
 1330         cmp #58
 1340         bcs val2
 1350         cmp #48
 1360         bcc val2
 1370         sbc #48
 1380         sta aa+1
 1390         lda nuacc
 1400         cmp #26
 1410         bcs val1
 1420         asl a
 1430         asl a
 1440         asl a
 1450         adc nuacc
 1460         adc nuacc
 1470         adc aa+1
 1480         bcs val1
 1490         sta nuacc
 1500         jmp va333
 1510 val1    sec
 1520         rts
 1530 val2    clc
 1540         rts

ready.
```

Dans le cas où l'instruction n'est pas pour vous :

- retourner à l'indirection précédente si celle-ci existait.
- sinon, faites simplement RTS

Dans le cas où l'ordre est pour vous :

- exécutez celui-ci
- puis faites : - pla
  - pla
  - RTS

  pour terminer l'ordre correctement.

ANNEXES

```
1000 ;
1010 ; adresses des parametres
1020 ;
1030 zwdeb   = $300
1040 dimdef = $302
1050 zt      = $303
1060 lm      = $305
1070 zf      = $307
1080 drive   = $309
1090 cylind = $30a
1100 tete    = $30b
1110 secteu = $30c
1120 comman = $30d
1130 adblc   = $30e
1140 ;
1150 ;tables des drives
1160 ;
1170 tcylm   = $310          ;012345 ......
1180 ttetem = $316          ;6789ab ......
1190 tsectm = $31c          ;cdef01 ......
1200 trwts   = $322          ;234567 89abcd
1210 ttype   = $32e          ;ef0123 ......
1220 ;
1230 jsrmon = $334          ; jsr combas
1240 ;                  jsr $0000
1250 ;                  lda #0
1260 ;           combi sta $c200
1270 ;                  rts
1280 ;           combas lda #$80
1290 ;                  bne combi
1300 ;
1310 slotcd=jsrmon+10
1320 pagecd=jsrmon+7
1330 combi  =jsrmon+8
1340 combas=jsrmon+12
1350 ;
1360 aplexe = $344
1370 ;
1380 ;
1390 ; indirection retour systeme
1400 ;
1410 jmpsy   = $34a
1420 ;
1430 ;
1440 ;
1450 ;caracteres ecran utilises
1460 ;
1470 zblanc = $350          ; 160
1480 zapost = $351          ; 162;''
1490 zetoil = $352          ; 170;'*
1500 zzero   = $353          ;  48;'0
1510 zneuf   = $354          ; 185;'9
1520 zplus   = $355          ; 171;'+
1530 zadres = $356          ; 128;'@
1540 zdollr = $357          ; 164;'$
1550 zpourc = $358          ; 165;'%
1560 zslash = $359          ; 175;'/
1570 zinfer = $35a          ; 188;'<
1580 zsuper = $35b          ; 190;'>
```

```
1590 zdeuxp = $35c          ; 186;';
1600 zpvirg = $35d          ; 187;';
1610 zinter = $35e          ; 191;'?
1620 zpoint = $35f          ; 174;'.
1630 ztrans = $360          ;   0; transparent
1640 manuel = $361          ;   0
1650 ;
1660 call   = $362          ; !!!!!!!!!!!!!
1670 ;call 941
1680 ;
1690 ; call .byte 0,140;-< sys >-
1700 ; .byte '941'
1710 ; .byte 0,0,0
1720 ;
1730 ;
1740 ;caracteristiques ecran
1750 ;
1760 adrecr = $368          ; 4,0
1770 lngecr = $36a          ; 3,192
1780 nlscrn = $36c          ; 40
1790 cinp   = $36d          ; $1b,$fd
1800 ;
1810 ; fonction indirectes
1820 ;
1830 ;
1840 avance = $36f          ; jmp av40c
1850 recule = $372          ; jmp rec40c
1860 aa     = $375          ; jmp aa40c
1870 ae     = $378          ; jmp ae40c
1880 getind = $37b          ; jmp get40c
1890 invers = $37e          ; jmp inv40c
1900 flash  = $381          ; jmp fla40c
1910 incar  = $384 lda (vv),y
1920 ;              rts
1930 outcar = $387          ; sta (vv),y
1940 ;              rts
1950 indcom = $38a
1960 ;              0
1970 ;              0
1980 ;
1990 ;
2000 ;indirection
2010 ;
2020 imprent=$3ad;vers deb prg ass
2030 ;
2040 ;
2050 ;
2060 ; impressions ( compteur de lignes )
2070 ;
2080 nblgn  = $3b3
2090 ;
2100 ;
2110 nbdeci = $3b4
2120 ;
2130 evamsk = $3b5
2140 ;
2150 azerty = $3b6
2160 ;
2170 ; prg acces azerty
2180 ;
2190 aplaze = $3b7
2200 ;
2210 ; 1er libre en $3be
```

CHAPITRE VIII POINTEURS DU DOS

8 - 1 UTILISATION DE LA MEMOIRE

L'espace mémoire de votre micro-ordinateur est utilisé de la façon suivante :

ADRESSE (HEXADECIMALE)

$0000 – $00FF	page zéro - pointeurs divers
$0100 – $01FF	pointeurs internes
$0200 – $02FF	buffers
$0300 – $03FF	pointeurs
$0400 – $07FF	mémoire écran
$0800 – fin programme	texte du programme BASIC
début variable – fin variable	variables du BASIC
début buffers – fin de buffers	buffers du DOS
début des handlers – fin des handlers	HANDLERS
$C000 – $CFFF	I/O
$D000 – SFFFF	BASIC et MONITOR OU DOS

Pour plus de détails sur les pointeurs, utilisés par le BASIC, se reporter au manuel d'utilisation de celui-ci.
Deux pointeurs sont particulièrement intéressants pour l'utilisateur puisqu'ils peuvent être modifiés.

- le pointeur de fin de programme BASIC
- le pointeur de début des buffers du DOS

Normalement, ces deux pointeurs doivent être les-mêmes, les buffers du DOS commençant là où finit le BASIC.

Adresse des pointeurs (en décimal)

115 : fin du BASIC poids faible
116 : fin du BASIC poids fort

768 : début des buffers du DOS poids fort
769 : début des buffers du DOS poids faible

Pour modifier ces valeurs, faites par exemple :

```
POKE 115, Ø : POKE 116,50 : CLEAR
POKE 769, Ø : POKE 768,50 : LET"#C,$"
```

Les deux ordres CLEAR et LET"#C,$" sont indispensables pour que les modifications soient prises en compte.

NOTE : Il est possible de laisser une place libre entre la fin du BASIC et le début des buffers du DOS.

8 - 2 DIMENSION PAR DEFAUT DES TABLEAUX

Lors de l'ouverture ou de la création d'un fichier, les tableaux qui n'ont pas été dimensionnés au préalable le sont automatiquement.

Ce sont alors des tableaux à 1 dimension dont l'indice maximum est 8.

Cette valeur par défaut peut être changée . Faire :

```
POKE 770,[nombre d'indice désiré]
```

CHAPITRE IX PRINCIPE DE FONTIONNEMENT DU MEM/DOS

9 - 1 - a Structure des objets en mémoire centrale

Les différents types d'objets contenus en memoire sont :

1) les fichiers : type = " F "
2) les masques : type = " M "
3) les descriptions de disque : type = " $ "

Chaque objet est rangé sous la forme :

numéro logique
type
longueur poids faible
longueur poids fort
objet par lui-même

Les objets sont placés consécutivement en mémoire.
Pour retrouver un objet, la méthode est la suivante :

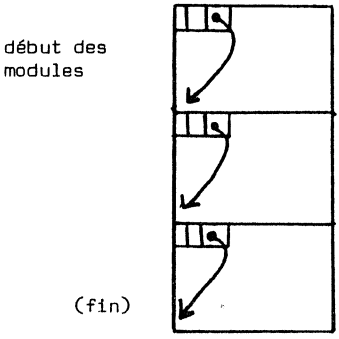
- accès au premier élément
- comparaison du numéro logique

  Si celui cherché : fin
  Sinon, ajouter à l'adresse de départ la longueur totale qui permet d'accèder à l'élément suivant.

Et ainsi de suite...

Deux pointeurs permettent de connaitre les limites des objets :

- les pointeurs de début des modules ($300,$301) + 292 ($124)
- les pointeurs du premier libre



Le premier module commence à l'adresse :

PEEK (768)*256 + PEEK (769) + 1028

Destruction d'un objet :

L'ordre LET"#C,[n° logique]

permet de récupérer la place d'un objet.

Pour cela, les objets suivants en mémoire sont décalés de la longueur de l'objet détruit et le premier libre est diminué de la même valeur.

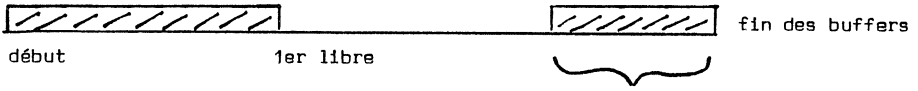
Double utilisation des buffers

La place disponible dans les buffers entre le pointeur du premier libre et la fin des buffers est également :

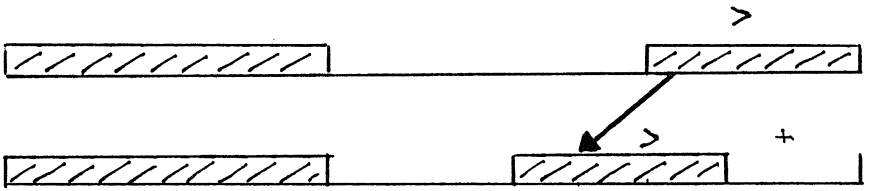
Elle sert :

- à enregistrer les descriptions de fichiers
- à enregistrer des ordres pour l'EXECUTE

Par exemple, l'ordre LET" > ..... "
va transférer la chaine de caractères suivant le " > " dans cet espace, cadrée à droite.



L'ordre LET" +.... " permet de compléter cette chaine. Pour cela, on décale vers la gauche la chaine déjà enregistrée, puis on insère la nouvelle.



Trois ordres permettent d'inscrire des caractères dans ce buffer :

```
LET" > ....."
LET" + ....."
LET"ENTER-[n° logique]
```

Ce dernier ordre décode le dictionnaire d'un fichier déjà enregistré et le place dans le buffer.

Dans le cas ou la place disponible entre le premier libre et la fin des buffers est insuffisante, le programme le signale par le message :

```
? OUT OF MEMORY ERROR
```

9 - 1 - b Structure des fichiers/DCB

1) Description générale

Un fichier MEM/DOS ne sont pas tous semblables puisque chacun a une définition appelerons PISTE LOGIQUE.
Ces pistes peuvent ou non correspondre à des pistes physiques du disque.

- création du fichier

A la création du fichier, le système réserve deux pistes logiques :
- la première est réservée aux clés ou pointeurs
- la seconde est réservée aux enregistrements

Au fur et à mesure du grossissement du fichier, un certain nombre de pistes logiques pourront être allouées dynamiquement au fichier.

Les fichiers MEM/DOS est découpé en un certain nombre d'unités de base que nous d'enregistrement qui lui est propre.
Pour cela, un descriptif est attaché à chaque fichier.
Ce descriptif est appelé D.C.B. (Data Control Bloc). Il comporte un certain nombre d'éléments dont les principaux sont :

- La description de l'enregistrement
  Ces paramètres définissent les différentes clés, leur type,...
  Pour chaque variable de l'enregistrement, 7 octets sont réservés pour sa description.

- La liste des pistes logiques occupées par le fichier.
  Au fur et à mesure que des pistes sont allouées, leurs numéros sont ajoutés à cette liste, classés dans un ordre fictivement croissant pour permettre les recherches par clés.

- Les différents pointeurs permettant dans ce fichier de connaitre les blocs libérés à la suite de destruction ou non encore utilisés.

Une description de cette DCB est donnée en fin de chapitre.

- les clés ou pointeur

Les fichiers peuvent être de deux types :

- fichiers relatifs
- fichiers séquentiels indexés

Dans le cas des fichiers relatifs, la structure est la suivante :

- Les pistes de pointeurs contiennent un chainage sur l'enregistrement.
  Ce chainage est une suite de trois octets, ce qui permet de prévoir des extensions importantes.

Dans chaque bloc réservé aux pointeurs, on peut donc mettre 85 pointeurs.

Chaque pointeur renvoie sur l'enregistrement de la façon suivante :

- 1er paramètre : la piste logique où se trouve le début de l'enregistrement.
- 2ème paramètre : le numéro du bloc dans cette piste logique.

Lorsque vous demandez la lecture de l'enregistrement N, le système va calculer tout d'abord la piste logique où se trouve ce pointeur, puis lira le bloc correspondant. Le pointeur permettra alors de trouver l'enregistrement.

Cette méthode impose de lire deux blocs au moins pour accéder à un enregistrement, mais en revanche, elle n'impose aucune contrainte sur la longueur de l'enregistrement. C'est cet avantage qui permet au MEM/DOS de gérer des fichiers de taille variable.

Pour les fichiers séquentiels indexés, les pointeurs vers les enregistrements sont complétés par la clé. Cette clé peut être éventuellement une suite de sous-clés.

Le COEFFICIENT DE BLOCAGE des clés d'un fichier séquentiel indexé sera donc calculé de la façon suivante :

Soit L le nombre d'octets de la clé,
pour chaque article, on enregistre L + 3 + 2 octets
avec :
L octets pour la clé
3 octets pour pointer sur l'enregistrement
2 octets pour pointer vers le suivant

Le nombre de clés que l'on pourra mettre sur un bloc de 256 octets sera donc la partie entière de 256 / (L + 5)

- fichiers multiclés

Dans le cas d'un fichier comportant plusieurs clés, chaque clé demandera un ensemble de pistes logiques.

- principe de recherche par clé

Les clés sont réparties en deux types :
- les clés triées
- les clés non triées (appelées TAS)

Les clés triées sont rangées par ordre croissant sur les pistes logiques.
Le principe de recherche sur ces clés est la recherche DICHOTOMIQUE.
Il consiste à comparer la clé recherchée avec une clé médiane, ce qui permet d'éliminer la moitié des clés.

Exemple :

La liste des clés enregistrées est la suivante :

A B G K P Q U V W Y Z

recherche de la clé I :
La clé médiane est Q : I est plus petit que Q : les clés supérieures à Q sont éliminées.

La clé médiane est G : I est plus grand que G : on élimine A.....G

La clé médiane est K : I est plus petit que K

Il n'y a plus de clés entre G et K, la clé recherchée n'existe pas.

Intérêt de la méthode :

A chaque essai, (qui demande au plus une lecture disque) on élimine la moitié du fichier. Le temps de recherche est donc de l'ordre de :

LOG (nombre de clés) x constante

Exemple : lecture sur disque dur (temps d'accès moyen 50 millisecondes) le coefficient de blocage est de 25 (clés de 5 octets, par exemple un flottant)

parmi	25 clés,	1 lecture	50 ms
	50 clés,	2 lectures	100 ms
	100 clés,	3 lectures	150 ms
	200 clés,	4 lectures	200 ms
	400 clés,	5 lectures	250 ms
	800 clés,	6 lectures	300 ms
	1600 clés,	7 lectures	350 ms
	3200 clés,	8 lectures	400 ms
	6400 clés,	9 lectures	450 ms
	12800 clés,	10 lectures	500 ms

Chaque fois que le fichier double de taille, le temps de recherche n'augmente que de une lecture disque.

Dans l'exemple (cas assez fréquent) le temps d'accès à un article parmi 10 000 est environ 1/2 seconde.

Problème du TAS : création d'articles

Lors d'une création d'article, le but est d'insérer une nouvelle clé dans la liste. Cependant, si le fichier comporte un nombre important d'articles, cette opération risque de demander un grand nombre de décalages. Par exemple, dans le cas décrit ci-dessus, avec 10 000 articles, pour insérer une nouvelle clé en début de liste il faudrait décaler 10 000 clés, soit 400 blocs de 256 octets Le temps d'insertion serait alors de 20 secondes, ce qui n'est pas admissible en utilisation normale.

De ce fait, la technique employée dans le MEM/DOS pour créer de nouvelles clés est légèrement différente.

Une nouvelle clé n'est pas insérée directement à sa place, mais ajoutée à la fin de la liste dans ce que nous appelerons le TAS.
Ces nouvelles clés ne sont pas triées, mais CHAINEES aux autres clés.

A la fin de la recherche dichotomique, le système connait la clé immédiatement inférieure et la clé immédiatement supérieure. La nouvelle clé du TAS est chainée à la clé immédiatement inférieure. Pour cela, on utilise le pointeur permettant de retrouver le suivant.

Cela modifie dont l'algorithme de recherche d'un article : après avoir cherché la clé dans la liste triée, il faut vérifier qu'aucun chainage ne conduit à la clé recherchée dans le tas.

Le temps d'accès à un article dont la clé est dans le tas est bien entendu moins performant que dans le cas de la liste triée.

Supposons par exemple que le fichier comporte 10 000 articles.
Nous en insérons 10 000 autres.
Pour les 10 000 premiers qui sont dans la liste triée, le temps d'accès est de 500 ms.
Les nouveaux articles, eux, sont dans le tas.

Si la répartition des clés est aléatoire, on peut espérer que chaque nouvelle clé se situera entre deux des clés triées.
Dans ce cas, la détérioration du temps d'accès sera en moyenne de un accès supplémentaire pour la lecture d'une clé du tas.
Le temps d'accès est alors :

500 ms (10 accès) pour une clé de la liste triée
550 ms (11 accès) pour une clé du tas.

Le système comporte un ordre : REORGANISATION qui permet de ranger les clés du tas. Pour cela, toutes les clés sont relues dans l'ordre croissant grâce aux pointeurs vers le suivant et sont réécrites sur de nouvelles pistes de clés.
Lorsque l'opération est terminée, la nouvelle liste de clés utilisées remplace la précédente dans la DCB et les pistes précédentes sont libérées.

REMARQUE : Le problème du tas est particulièrement sensible dans le cas d'un fichier initialement vide. En effet, dans ce cas, aucune clé n'est triée et le temps d'accès est alors en moyenne la moitié du temps d'une recherche séquentielle (le temps de recherche est alors proportionnel au nombre de clés !).

Le temps de réorganisation est facile à calculer : c'est le temps de lecture et d'écriture de toutes les clés. Pour 10 000 articles, il faut manipuler 400 blocs en lecture et écriture soit : 400 x 2 x 50 ms = 40 000 ms

En quarante secondes, le fichier est trié !

NOTE : en multiclés, l'opération de réorganisation est répétée sur chacune des clés.

Problème de la place disponible pour réorganiser

Pour que la réorganisation soit possible, il est nécessaire que le nombre de pistes libres soit suffisant pour recopier la liste des clés.
Au besoin, on peut récupérer des pistes libres en détruisant d'autres fichiers.

Il est important de noter que les différentes réorganisations d'un fichier font que les pistes utilisées pour les clés se déplacent sur le disque et ne restent pas figées. Or ces pistes sont fréquemment utilisées lorsque l'on fait des recherches par clé. Ce principe permet de répartir l'usure de la surface sur tout le disque (dans le cas des disques souples bien sur !)

Le CATALOGUE du disque étant également un fichier à accès par clé, le même phénomène se produit également dans le cas de chargements de programmes ou masques.

- structure de l'enregistrement

L'enregistrement a la même structure que le fichier soit relatif, ou séquentiel indexé.

Une caractéristique de l'enregistrement est le coefficient de blocage. Celui-ci représente le nombre d'octets contenus dans un bloc de base.

Les enregistrements sont de tailles variables, il n'est donc pas possible de définir exactement la taille du bloc de base (élément le plus petit d'un enregistrement).

Pour des raisons d'optimisation, la taille du bloc de base pourra être :

- 32 octets
- 64 octets
- 128 octets
- 256 octets

Sur ces octets définissant le bloc de base, trois sont réservés pour les chainages internes.

Le fonctionnement est le suivant :

1) L'article est plus petit que le bloc de base.
   Il ne demandera alors qu'un seul bloc et le chainage interne indiquera fin d'article.

2) L'article est supérieur au bloc de base.
   Il faudra alors plusieurs blocs de base pour le contenir et ceux-ci seront chainés ensemble. Le chainage interne du premier pointant sur le second et ainsi de suite. Le dernier bloc de base ne contient pas de chainage significatif, comme dans le cas d'un bloc de base unique.

Lors d'une mise à jour d'un article, celui-ci pouvant changer de dimension, tous les blocs de base sont désalloués puis repris en fonction des besoins.

La taille du bloc de base est une constante du fichier (contenuedans la DCB). Celle-ci est déterminée automatiquement lors de la création.

L'algorithme de calcul est le suivant :

En ne tenant compte que des variables de l'enregistrement,

chaque flottant vaut 5 octets
chaque chaine de caractères est approximée à 12 octets
chaque binaire simple vaut 1 octet
chaque binaire double vaut 2 octets
chaque date vaut 2 octets

Les tableaux sont supposés contenir en moyenne trois éléments.

Le coefficient de blocage choisi est celui immédiatement supérieur à la valeur obtenue dans la liste des possibles (32, 64, 128, 256) en tenant compte des 3 octets de chainage par bloc de base.

Intérêt de l'optimisation du bloc de base

Plus le bloc de base est petit, plus l'utilisation de l'espace disque sera optimisée.(moins de place perdue)

Plus le bloc de base est grand, plus le temps d'accès sera petit. (en effet, les différents blocs de base ne se trouvent pas forcément dans le même bloc physique, dans ce cas, il faudra plusieurs lectures disques. De plus, en cas d'écriture, il faut lire les blocs avant de les réécrire.)

- codage des informations

Dans le but de minimiser la place perdue sur le disque, les informations sont compactées.
Pour lire ou écrire un article, le système se fie au dictionnaire (qui décrit l'enregistrement)
Les variables simples sont écrites directement sans séparateur.

Exemple :

un flottant A ..... 5 octets
un binaire simple ..... 1 octet
une date ..... 2 octets
un binaire double ..... 2 octets
une chaine de caractères ... 1 + la longueur de chaine

(l'octet supplémentaire indique la longueur)
les blancs non significatifs sont supprimés (aux extrémités)

Dans le cas des tableaux le codage est le suivant :
Les variables du tableau sont lues dans l'ordre.

- si n indices du tableau correspondent à des variables non nulles ou non vides consécutives, ces variables sont codées comme une suite de variables simples.

- si n variables successives sont nulles ou vides :

  - si n est plus petit que 128
    les variables sont codées sur 2 octets : 0, n

  - si n est plus grand que 127
    les variables sont codées sur 3 octets : 0, (poids fort de n+128)
    (poids faible de n)

- fin de tableau :
  La fin de tableau est repérée par deux 0 consécutifs (configuration impossible dans le codage de valeurs nulles ou vides)

- codage des dates

Les dates sont codées sur 2 octets.

A partir de la date standard 21/02/81 la transformation est :

AAAAAAAM	MMMJJJJJ
01234567	01234567

L'année est codée sur 7 bits (0 à 128)
Le mois est codé sur 4 bits (0 à 16)
Le jour est codé sur 5 bits (0 à 32)

Le codage de l'année se fait en ajoutant 50 aux années inférieures à 50 et en enlevant 50 aux années supérieures à 50.

De ce fait, les années 00 à 49 sont considérées comme 2000 à 2049
les années 50 à 99 sont considérées comme 1950 à 1999

Les années 2000 et plus sont donc bien supérieures aux années 80-90.

Importance de l'ordre

Si vous créez un fichier séquentiel indexé dont une clé (ou une partie d'une clé) est une date, les articles seront automatiquement classés.

Exemple concret de codage d'enregistrement

a) Définition

L'enregistrement est défini par :

```
LET" A = A$,B,C%,A ; , B$ ;
```

Ecrivons sur le fichier l'enregistrement suivant :

```
B = 1
A$ = " TEST "
C% = 10
A(0) = 1, A(6) = 2, les autres éléments sont nuls.
B$ (0) = " TUTU ", les autres éléments sont vides.
```

b) Codage (en hexa décimal)

A$

04 54 45 53 54 — longueur 4 "TEST "

B

81 00 00 00 00 — flottant 1 sur 5 octets

C%

00 0A — entier

A ;

81 00 00 00 00 | 00 05 | 82 00 00 00 00 | 00 00 | 1,5fois0,2
1 | 5 fois 0 | 2 | fin

B$

04 54 55 54 55 | 00 00 — longueur 4 " TUTU ", fin de tableau

L'enregistrement complet est donc codé :

```
: 04 54 45 53 54 81 00 00
: 00 00 00 0A 81 00 00 00
: 00 00 05 82 00 00 00 00
: 00 00
```

Notons que cet enregistrement pourrait avoir plusieurs milliers d'octets de longueur si les tableaux étaient bien remplis.

- data control block

TT	n° logique	1	Ø
	Type de l'objet en face Ø (ou M)	1	1
	Longueur de l'objet Pf-PF	2	2
	n° DRIVE	1	4
	Entête nom sur DRIVE = " F "	1	5
	NOM LØ = 20 -Nombre de pistes libres pour DCBØ	21	6
	Nombre de moyens d'accès	1	27
	Longueur sur le DRIVE PF-Pf	2	28
	Longueur minimum pour le save sur DRIVE PF-Pf (Ø ; 6Ø)	2	30
	Etat du fichier Ø normal ; octet de contrôle	1	32
	dans Magma DEBØ de l'objet	3	33
	Rel/TT de la T.S. Magma DCB PF-Pf	2	36
	Rel/TT de la table des pistes alloués	2	38
	LISTE DES 1ER LIBRES (PF-Pf) rel/T.T.	20	40
	MAT POINTEUR 1ER LIBRE MAM	6	66
MM	Rel prochain moyen d'accès PF-Pf	2	2
	Rel clé recherchée/MM PF-Pf	2	2
	Rel de MaX/MM PF-Pf	2	4
	Rel Xtract/MM PF-Pf	2	6
	Rel T.S. clé/MM PF-Pf	2	8
	Type de moyen d'accès R ou I	1	10
	Rel 1ère piste logique/PP PF-Pf	2	11
	Rel Dère piste logique/PP PF-Pf	2	13
	FIN DE TABLE TRIEE PF-Pf	2	15
	Inused	2	17
	Coefficient de blocage	1	19
	Longueur de l'enregistrement en octet	1	20
	n° enregistrement en MC PF-Pf	2	21
	T.S. pour les clés descriptif 7 octets/variables		23
	Clé recherchée LØ = 21		
	Clé maximum LØ = 21		
	Xtract : Rel de la zone 1	1	
	Longueur de la zone 1	1	
	AUTRES MOYENS D'ACCES		
	T.S. Magma descriptif 7 octets/variables		
PP	LISTE DES PISTES UTILISEES en deux octets		
	Coefficient de blocage enregistrement		
	LISTE DES PISTES / TETES	5	

9 - 1 - c Structure des masques

Les masques du MEM/DOS comprennent deux parties :

1) le texte du masque

Il s'agit du texte qui apparait par l'ordre LET"C,[n° logique]
Ce texte ne contient donc pas les fenêtres de saisie.

Le texte est compacté pour gagner de la place.

La méthode est la suivante :

Les caractères < et > étant interdits dans les masques car étant réservés aux positions des zones de saisie, ils seront utilisés comme caractères de contrôle.

Si un caractère de code écran X est répété n fois, il sera codé :

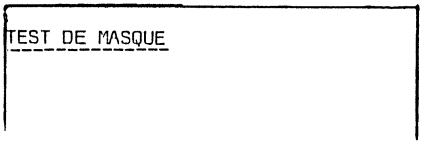
X, CODE("< "), n si n < 256
X, CODE("> "), poids faible de n, poids fort de n dans le cas contraire

Si n est inférieur à 4, il n'y aura pas de codage, car celui-ci n'apporterait pas de gain de place.

La fin de l'écran sera repéréepar un nombre nul de caractères identiques (configuration normalement impossible)

NOTE : CODE(" < ") = 60
CODE(" > ") = 62

Exemple : Codage du masque suivant :



Le texte du masque de cet exemple sera codé :

212	T	148
197	E	133
211	S	147
212	T	148
160	blanc	160
196	D	132
197	E	133
160	blanc	160
205	M	141
193	A	129
211	S	147
209	Q	145
213	U	149
197	E	133

```
160      blanc  )
188      <      }    26 blancs consécutifs
26       26     )
173      -      )
188       <     }    14 tirets consécutifs
14       14     )
160      blancs )
190       >     }    blanc jusqu'à la fin de l'écran
Ø        Ø      )
Ø        Ø      )
```

2) les zones de saisie

Pour chaque variable du masque à saisir, on enregistre 7 octets :

- adresse écran du début de la zone — 2 octets

- longueur de la zone — 1 octet

- nom de la variable — 2 octets
  (codé comme dans le BASIC)

- contrôles à effectuer — 1 octet

- indice dans le cas d'un tableau — 1 octet
  (99 dans le cas des listes)
  (Dans le cas des variables simples, cette dernière valeur est inutilisée)

Le contrôle sur 1 octet se décompose de la façon suivante :

- 2 bits pour indiquer le type
  - ØØ flottant
  - Ø1 binaire
  - 1Ø entier
  - 11 alphanumérique

- 1 bits pour contrôle (*)
  - Ø si date et alphanumérique ou numérique gestion
  - 1 dans les autres cas

- 1 bit pour entrée/sortie (?)
  - 1 : en sortie

- 1 bit pour positif (+)
  - 1 : numérique positif ou alpha numérique considéré comme numérique
  - Ø : dans le cas contraire

- 1 bit pour autoskip (!)
  - 1 : autoskip

- 1 bit pour entier (C)
  - 1 : entier ou binaire ou flottant entier (C)

- 1 bit pour tableaux
  1 : tableau
  Ø : variable simple

L'ensemble du masque est structuré de la façon suivante :

contenu	position
- numéro logique	Ø
- type : " M "	1
- longueur totale poids faible . poids fort	2 3
- longueur de la table poids faible poids fort (7 fois le nombre de variables NV)	4 5
- table (7 * NV octets) — — — —	6
- texte du masque — — — — —	6 + 7* NV

REPORTEZ-VOUS AUX SPECIFICITES APPLE IIc

CHAPITRE X UTILITAIRES STANDARDS

- anal piste et super catalog

Super catalog donne une idée de la place occupée par chaque fichier en nombre de pistes logiques. Toute incohérence est signalée et comptabilisée.

Anal piste a la même fonction, mais affiche la répartition des pistes logiques par fichier, puis la liste des fichiers par piste logique.

- auto copie

Recopie d'un drive sur un autre les masques, globaux et programmes. Le drive d'arrivée n'est pas détruit. S'il s'agit d'une disquette vierge, il faut donc la formatter par l'ordre LET " #F, n° drive ", avant de lancer le programme AUTO COPIE IMPORTANT : Les drives peuvent être de types différents (5', 8', disque dur).

- auto list

Edition de tous les masques et programmes contenus sur une disquette.

- autostart

Simule les fonctions clavier de la ROM autostart dans le cas où votre micro-ordinateur n'en est pas équipé.

```
CTRL/S : bloque un list
ESC/I  : déplacement du spot vers le haut
ESC/J  : déplacement du spot vers la gauche
ESC/K  : déplacement du spot vers la droite
ESC/M  : déplacement du spot vers le bas
```

- boot

Formattage d'un disque en laissant la piste Ø pour le BOOT. Si celui-ci est sur la disquette, il sera recopié en piste Ø. Les disquettes ainsi obtenues peuvent mettre en route le MEM/DOS 6502.

- check rom

Affiche les valeurs trouvées et les valeurs réelles des 3 ROMS du DOS, pour contrôler leur validité.

Les valeurs réelles sont affichées sur fond blanc

- copy

Copie intégralement un drive sur un autre. Les drives doivent être de même type.

IMPORTANT :

- Ce programme fonctionne quelque soit le périphérique.
- Le drive d'arrivée est effacé.

Si le disque d'arrivée a déjà été formatté, indiquer " N " à la question concernant le formattage.
Le cas se produit en particulier si le disque :

- a été déjà formatté (par LET " # F,n ")
- est déjà une copie dont on ne se sert plus.

- debug-menu

C'est le menu général regroupant les diverses fonctions.

- demo-masque

Programme de démonstration de masque MEM/DOS montrant les diverses possibilités de déplacement du curseur.

Pour plus d'informations sur la structure interne des objets et de leur codage sur disque : consulter le "Manuel de maintenance système".

- demo-fichier

Exemple de programme gérant un fichier en accès par clé. Vous constaterez que les deux masques utilisés ont été regroupés sous forme de GLOBAL, ce qui présente deux avantages :

- temps de chargement accéléré
- place restreinte sur disque

- file copy

Copie un fichier d'un disque sur un autre, après affichage du catalogue.

- hello

Ce programme est celui qui est exécuté à la mise en route du système. C'est lui qui se charge de mettre en place les HANDLERS 80 colonnes, si votre micro-ordinateur est équipé d'un tel type d'affichage ; en exécutant les programmes suivants :

- AIIE 80COL : 80 colonnes APPLE IIe*
- TEXT 80COL : 80 colonnes BASIS 108**

Dans tous les cas le programme Hello se débranche sur le programme SYS.MENU.

- interro

Ce programme permet de mettre à jour un fichier quelconque. Pour cela, deux masques sont créés provisoirement (il ne sont pas enregistrés sur disque).
Le premier demande l'opération désirée :

- la clé de recherche définie dans le fichier
- la commande : L = lire
  M = modifier
  C = créer
  D = détruire
  S = lire le suivant

Le deuxième sert à décrire l'article aussi bien en affichage qu'en saisie.

Dans le cas de tableaux, ceux-ci apparaissent en une ligne. Les différents indices sont séparés par " ; ".

- scroll

Affiche les adresses du SCROLL UP et du SCROLL DOWN.
L'adresse du SCOLL DOWN est calculé par :

```
PEEK(116) * 256 + PEEK(115)
```

L'adresse du SCROLL UP est fixe :

```
CALL 64624
```

Ce programme s'applique aux affichages 40 colonnes exclusivement.

- super contrôle

Les caractères de contrôles apparaissent sur fond blanc dans le listing du programme, mais l'exécution s'effectue normalement.

- util

Tous les ordres concernant les masques de saisie ou d'édition MEM/DOS en 40 ou 80 colonnes peuvent être exécutés directement sans affecter la mémoire, en particulier le programme. Cependant, cet utilitaire permet une mise à jour rapide et simple de votre catalogue de masques.
Les commandes possibles sont :

C | Création d'un masque :

L'écran s'éfface, saisissez votre masque puis faites Escape. (CTRL/A pour abandonner). Une cloche signale une erreur dans les fenêtres. Corrigez-la ou abandonnez.

M | Modification ;

Le contenu du masque apparait sur l'écran. Vous pouvez le modifier de la même façon qu'en création.

D | Destruction :

Le masque est affiché pour une dernière validation. Répondez " O " pour détruire.

* | Contenu du disque :

Permet de consulter la liste des masques avant de les manipuler.

L | Ouverture d'un masque :

Son nom s'affiche en bas à droite de l'écran. Il pourra servir de base pour la création d'un deuxième masque par la commande S

H | Impression d'un masque :

S | Création d'un masque à partir d'un autre :

Le module est le masque chargé par " L ".

V | Visualisation :

Simple affichage du masque.

F | Sortie du programme

I | Impression d'un masque :

Imprime un masque sur une imprimante placée en slot 2.

- sys-menu

Ce programme est celui qui est exécuté à la suite du programme Hello.

Il affiche un masque "Menu" regroupant les principaux utilitaires contenus sur la disquette.

Tapez sur le numéro correspondant à l'utilitaire choisi ainsi que sur la touche RETURN, le changement se fera automatiquement.

- verify

Ce programme permet de vérifier la cohérence de vos fichiers au niveau de l'adresse de la DCB (Data Control Block) sur le disque.

CHAPITRE XI
UTILISATION D'UNE PARTIE DES FONCTIONS MEM/DOS

Un certain nombre d'ordres du MEM/DOS pourront être utilisés en DOS 3.2. DOS 3.3 aux conditions suivantes :

- la page 3 est conforme aux spécifications (voir chapitre paramétrage du MEM/DOS
- une place a été réservée pour les buffers MEM/DOS et leurs adresses mises à jour en page 3.

Les ordres suivants sont alors utilisables :

LET"#I, n ,M, d : nom" ..... création d'un masque en mémoire
LET"C, n" ..... chargement du texte
LET"P, n" ..... affichage texte + variables
LET"O, n" ..... affichage variables
LET"V, n" ..... visualisation
LET"I, n" ..... saisie
LET"T, n" ..... saisie globale
LET"#C,n" ..... clear
LET"=xxxx±yyyy" ..... addition/soustraction sur 48 chiffres
LET" > xxxx" ..... remplissage buffers
LET" + xxxx" ..... ajout des buffers
LET" ! " ..... execute
LET"Z" ..... azerty
LET"- P" ..... arrondis
LET"y" ..... inversion vidéo écran
LET"G" ..... cloche
LET"?-" ..... " hard copy "
LET"?-c" ..... masques d'édition
LET"V" ..... visualisation buffers
LET"#C,$" ..... clear de tous les masques

Deux fonctions vont manquer puisqu'elles son liées à la gestion de fichier, il s'agit :

- du sauvetage des masques sauvés par LET"# I"
- de leur relecture

Pour cela, il faudra faire appel aux ordres :

BLOAD
BSAVE

Les limites du module à sauver sont :

```
début = PEEK(768)*256 + PEEK(769) + 292
fin + 1 = PEEK(773)*256 + PEEK(774)
```

Le sauvetage de l'ensemble des masques en mémoire se fera par :

```
X = PEEK(768)*256 + PEEK(769) + 292
L = PEEK(773)*256 + PEEK(774) - X
?CHR$(4) " BSAVE MASQUES, A " X ", L " L
```

La relecture se fera par :

```
X = PEEK(773)* 256 + PEEK(774)
?CHR$(4) " BLOAD MASQUES , A " X
Y = X + PEEK (43616) + PEEK(43617) * 256   sur 48K
POKE 773, INT(Y/256)
POKE 774, Y - 256 * INT(y/256)
```

RESUME DES ORDRES DU DOS

MASQUES

```
LET"#NEW-(n), MASQUE,(d) : (NOM) ..... Création d'un masque (donne la main)
LET"#NEW-(n), MASQUE,/(d) : (NOM) ..... Création de masques (prend l'écran)
LET"#NEW-(n), MASQUE,  (d) : (NOM) ..... Remplacement d'un masque
LET"#DELET-MASQUE,(d) : (NOM) ..... Destruction d'un masque
LET"#OPEN-(n), MASQUE,(d) : (NOM) ..... Ouverture d'un masque
LET"# IMAGE-(m),MASQUE,(d)..... Créé un écran en mémoire centrale
LET"CHARGE-(n) ..... Affichage du texte du masque
LET"VISUALISE-(n) ..... Affichage du texte et des fenêtres
LET"OUTPUT-(n),(t) ..... Affichage des variables du masque
LET"PRINT-(n) ..... Affichage du texte et des variables
LET"INPUT-(n),(z) ..... Saisie des variables du masque
LET"T,n" ..... Reprise de variables d'un masque
```

FICHIERS

```
LET" > vci...vcn = vei...vep ..... Description d'un dictionnaire
LET"+ ..... Ajout au dictionnaire
LET"#NEW-(n),FICHIER,(d) = (NOM) ..... Création d'un fichier
LET"#OPEN-(n),FICHIER,(d) : (NOM) ..... Ouverture d'un fichier
LET"#DELET-FICHIER,(d) : (NOM)..... Destruction d'un fichier
LET"#REORG-(n) ..... Réorganisation d'un fichier
LET"READ-(n) ..... Lecture d'un article
LET"NEXT-(n) ..... Lecture de l'article suivant
LET"UPDATE-(n) ..... Mise à jour d'un enregistrement
LET"DELET-(n) ..... Destruction d'un enregistrement
LET"BORNE-(n) ..... Fixe une borne maximum au fichier
LET"XTRACT-(n),(v) ..... Extraction par criteres sur la clé
LET"XINDEX-(n) ..... Envoie en relatif le prochain numéro
LET"WRITE-(n) ..... Création d'un enregistrement
LET"ADD-(n) ..... Ajout (homonymes autorisés)
LET"ENTER-(n) ..... Traduit en clair le dictionnaire
LET"VISUALISE ..... Imprime ce dictionnaire
LET"# > n" ..... Fixe un maximum au coefficient de blocage
```

PROGRAMMES

```
SAVE"(d) : (NOM) ..... Sauve le programme
SAVE"@(d) : (NOM) ..... Sauve en écrasant
LOAD "(d) : (NOM) ..... Charge un programme
LOAD"#(d) : (NOM) ..... Charge le programme à la suite
LOAD"/(d) : (NOM) ..... Exécute le programme sans CLEAR
RUN"(d) : (NOM) ..... Charge et exécute un programme
LET"#DELET,PROGRAMME,(d) : (NOM) ..... Destruction d'un programme
```

DIRECTORY

LET" :,(d) ..... Directory complète du disque d
LET" :,(d),(M/F/P/) ..... Liste des masques/fichiers/programmes
LET"#REORG-$(d) ..... Réorganise le catalogue du disque d

ORDRES GENERAUX

LET"# FORMAT,(d)..... Formattage sur le drive d
LET"@-nombre décimales ..... Paramétrage du format gestion
LET"#CLEAR,(n) ..... Récupère la place RAM d'un objet
LET"#CLEAR,"+CHR$(d) ..... Fermeture d'un disque
LET"#CLEAR,$ ..... Ferme tous les fichiers, masques, ...
LET"Y ..... Inversion vidéo de tout l'écran
LET"! ..... Exécute d'une chaine entrée par
LET"%(d) ..... Indique le nombre de pistes libres
LET" = oper 1 + - oper 2 ..... Addition et soustraction de chaines
LET"G ..... Emission d'un signal sonore
LET"Z ..... Passage en AZERTY
LET"?-(x) ..... Impression de la ligne repérée par(x)inVersé
LET"?-: ..... Hard Copy
LET"#C,$d." ..... Fermeture du drive d pour chargement disque
LET"#C,$$" ..... Ferme tous les drives

FICHIERS MULTICLES

LET" > vai,...,vai & vbi,...vbj & .....
LET"+ = vei,...., ven ..... Définition d'un fichier multiclé
LET"#NEW-(n),FICHIER,(d) : (NOM) ..... Création d'un fichier (après ....)
LET"#DELET-FICHIER,(d) : (NOM) ..... Destruction d'un fichier
LET"# OPEN-(n),FICHIER,(d) : (NOM).. Ouverture d'un fichier
LET"#REORG-(n) ..... Réorganisation d'un fichier
LET"READ-(n),(my) ..... Lecture d'un enregistrement
LET"NEXT-(n),(my) ..... Lecture séquentielle
LET"UPDATE-(n),(my) ..... Mise à jour d'un enregistrement
LET"DELET-(n),(my) ..... Destruction d'un enregistrement
LET"BORNE-(n),(my) ..... Fixe une borne maximum au fichier
LET"XTRACT-(n),(v),(my) ..... Extraction sur la variable v
LET"WRITE-(n) ..... Création d'un enregistrement
LET"ADD-(n) ..... Ajout (homonymes autorisés)
LET"ENTER-(n) ..... Traduit en clair le dictionnaire
LET"VISUALISE ..... Imprime ce dictionnaire

BINAIRES

LOAD"$XXXX,$YYYY,d: NOM ..... LOAD de module binaire
SAVE"$XXXX,$YYYY,d: NOM ..... SAVE de module binaire

GLOBAUX

LET"#N,G,G, d: NOM ..... Création d'un global
LET"#O,G,G, d: NOM ..... Ouverture d'un global

ACCES DIRECT

LET"$ d,c,n,p,t,s,$XXXX ..... Accès direct

LEGENDES

n : identification temporaire ou numéro logique

d : numéro de drive

nom : nom de l'objet

v : numéro d'1 variable clé dans la définition d'un fichier

z : numéro d'1 zone en entrée dans 1 masque

t : type de zone en impression dans un masque

0 : en entrée (chiffre 0)
O : en sortie (lettre O)
rien : toutes les zones

my : numéro du moyen d'accès en multiclés

x : caractère de contrôle écran pour éditer son masque

SPÉCIFICITÉS APPLE IIC

MEMS FT

PROGICIELS

DEMAIN C'EST AUJOURD'HUI

CHAPITRE 1 . DESCRIPTION GENERALE DU SYSTEME

1.1 LE DISK OPERATING SYSTEM

L'adaptation du système d'exploitation MEM/DOS pour l'Apple //c comporte très peu de modifications fonctionnelles par rapport au MEM/DOS pour Apple //e :

- Les instructions Basic peuvent être entrées aussi bien en minuscules qu'en majuscules.

- La fonction multipostes n'est pas supportée par l'Apple //c .

- La fonction CALL FN ainsi que le handler 80 colonnes sont chargés en mémoire au démarrage de la disquette système.
  Cela diminue la place utilisable en mémoire vive.

  Si l'utilisateur manque de place, il peut :

  - supprimer le CALL FN (s'il n'emploie pas d'ordres CALL FN dans ses programmes) ;

  - supprimer le chargement du handler 80 colonnes si ses programmes sont en 40 colonnes (en plus de celui du CALL FN).

MARCHE A SUIVRE :

1. Pour supprimer le handler 80 colonnes ET le CALL FN, il faut remplacer le programme :

   MEMDOS//c (C)MEMSOFT qui se trouve sur la disquette système par le programme :

   MEMDOS//c 40COL qui se trouve sur la disquette utilitaires.

ATTENTION : le c de MEMDOS//c est toujours en minuscules.
Tout le reste est en majuscules

Pour effectuer ce remplacement :

- mettre la disquette utilitaires dans le drive 0
- taper : LOAD ''MEMDOS//c 40COL
- mettre la disquette système dans le drive 0
- taper : SAVE''àMEMDOS//c (C)MEMSOFT

2. Pour supprimer seulement le CALL FN, il faut remplacer le programme :

   MEMDOS//c (C)MEMSOFT de la disquette système par le programme :

   MEMDOS//c -CALLFN de la disquette utilitaires.

Pour cela :

- mettre la disquette utilitaires dans le drive 0
- taper : LOAD ''MEMDOS//c -CALLFN
- mettre la disquette système dans le drive 0
- taper : SAVE''@MEMDOS//c (C)MEMSOFT

Le programme MEMDOS//c (C)MEMSOFT se trouve également sur la disquette utilitaires. Vous pourrez donc toujours récupérer la version initiale de la disquette système en cas de fausse manipulation.

CHAPITRE 2 . MISE EN ROUTE DU MEM/DOS

Pour l'Apple //c il n'y a pas à connecter de carte, comme c'est le cas pour l'Apple //e .

2.1 CHARGEMENT DU SYSTEME D'EXPLOITATION MEM/DOS

La mise en route de MEM/DOS doit se faire obligatoirement avec la disquette système MEM/DOS A//c .

Introduire la disquette dans le lecteur intégré de l'ordinateur, et mettre en marche l'ordinateur.
Le système d'exploitation est automatiquement chargé en mémoire et le programme initial affiche à l'écran :

```
MEMDOS //c
(C) 1984 MEMSOFT
```

Le message suivant apparaît :

```
Pour accéder aux utilitaires MEMDOS
mettez la disquette UTILITAIRES et
tapez RUN''HELLO
```

Le système d'exploitation est désormais totalement en mémoire, et vous pouvez enlever la disquette système du drive intégré.

Rappelez-vous que sous système MEM/DOS, le drive intégré sera le drive numéro zéro et que le drive externe portera le numéro 1.

Pour travailler, introduisez maintenent la disquette utilitaires MEM/DOS que vous allez d'abord commencer par copier.

2.2 FORMATTAGE D'UNE DISQUETTE

La disquette système qui vous est fournie ne peut pas être copiée. Veuillez donc la protéger contre l'écriture en y mettant un sticker de protection.

Pour formatter vos diquettes de travail (non système), vous pouvez utiliser la commande de formattage MEM/DOS :

```
LET ''#FORMAT, No de drive''
```

ou bien passer par le menu des utilitaires MEM/DOS décrits au chapitre X .

Une disquette formattée ne contiendra pas le système d'exploitation MEM/DOS.
Elle ne permettra donc pas, seule, de démarrer une application sous MEM/DOS.

La disquette système est livrée sans programmes pour que l'utilisateur puisse y mettre ses logiciels.
Pour transférer un logiciel existant sur la disquette système, il faut utiliser le programme AUTO COPIE qui se trouve sur la disquette utilitaires, pour recopier tous les objets sur la disquette système.

Exemple : disquette système en 0
disquette programme en 1

Lancer AUTO COPIE de 1 vers 0 .

Il faut également modifier le programme HELLO pour qu'il enchaine sur le premier programme du logiciel.
Pour cela, faire :

```
LOAD ''HELLO
DEL 60, 99
60 RUN ''nom du programme''
SAVE ''@HELLO''
```

ATTENTION :

Si le premier programme du logiciel change le handler 80 colonnes, supprimer cette opération car le handler 80 colonnes est déjà mis en place au démarrage de la disquette système.

PARAMETRAGE DU MEM/DOS

Le MEM/DOS étant chargé en mémoire centrale, les commutations de ROM n'ont plus aucune utilité. Par contre toute la partie consacrée aux BUFFERS de l'Apple //e reste utilisable.

CHAPITRE X - UTILITAIRES STANDARDS

La disquette utilitaires MEM/DOS Apple //c contient un certain nombre de prgrammes qui sont nécessaires pour copier des disques, transférer des fichiers, créer, modifier, imprimer des masques.

L'utilisateur peut examiner les programmes et les masques qui sont sur cette disquette, et les adapter en partie pour traiter ses applications.

Après avoir chargé le système MEM/DOS en mémoire, le programme initial affiche : ''Pour accéder aux utilitaires MEM/DOS, mettez la disquette UTILITAIRES et tapez ''RUN HELLO''.
Enlever la disquette système et introduisez la disquette des utilitaires dans le lecteur intégré, après avoir placé un sticker de protection.

Tapez la commande RUN ''HELLO'' et appuyez sur RETURN pour charger le programme menu des utilitaires.

Les fonctions de saisie ont été standardisées dans tous les utilitaires :

La touche RETURN :

fait passer le curseur à la rubrique suivante de l'écran, ou à l'étape suivante du programme.
Le contrôle de validité des données saisies se fait après le dernier RETURN de l'écran. Un message d'erreur éventuel, ne se déclenchera donc qu'au dernier RETURN.

La flèche à gauche <--- :

fait remonter le curseur à l'information précédente de l'écran, qui peut alors être modifiée.

Les touches CTRL - A :

entraînent l'interruption du programme en cours et le retour au menu général des utilitaires.

La touche ESCAPE :

pendant la saisie d'un masque, met fin à la saisie et valide cette saisie, en sauvant le masque sur disque.

OPTION 1 - BASIC

Cette option permet d'accéder au basic.

OPTION 2 - GESTION DES MASQUES

Le programme utilitaire de gestion des masques permet de créer, modifier, détruire ou imprimer un masque .
Le programme est chargé par l'option 2 du menu général.

L'écran affiche :	Réponse à donner :
NOM DU MASQUE	Le nom de l'objet sur disque, qui contient la description du masque (précédé de ''M'' dans le catalogue).
DISQUE	Numéro du drive sur lequel se trouve le masque.
ECRAN (40/80)	Taper 40 ou 80, selon que le masque à créer devra s'afficher sur l'écran en mode 40 colonnes ou en mode 80 colonnes. Si le masque existe déjà, ce paramètre n'est pas pris en considération par le programme. Le masque créé en 40 colonnes s'affichera automatiquement en mode 40 colonnes et le masque créé en 80 colonnes s'affichera automatiquement en mode 80 colonnes.

COMMANDE
========

V	Visualisation : affiche le masque
C	Création : création d'un nouveau masque
D	Destruction : destruction sur le disque du masque nommé,
M	Modification : permet de modifier un masque existant sur disque.
L	Lire : charge en mémoire un masque du disque
S	Sauver : sauve, on écrit sur disque le masque actuellement en mémoire
*	Catalogue : affiche à l'écran, la liste des objets du disque
E	Explication : affiche la fonction de chaque commande
H	Hard copy : imprime sur l'imprimante le masque nommé
F	Fin : retour au menu général

Visualisation : V

La commande V visualiser permet d'afficher la description du masque nommé.
La description d'un masque comprend :

- une partie ''texte''
- des fenêtres de saisie
- les variables basic

Le texte d'un masque comprend non seulement du texte explicatif pour l'utilisateur, les titres des rubriques à saisir, mais aussi les encadrements, les traits et d'une manière générale, tout ce qui sera constamment affiché lorsque le masque est envoyé à l'écran par le programme.

Les fenêtres de saisie sont délimitées par < et >. Le curseur est automatiquement envoyé au début de chaque fenêtre, après chaque information saisie par l'utilisateur. Le nombre de caractères entre <et> fixe donc la longueur maxima de la donnée à saisir.
Si la donnée à saisir n'a qu'un seul caractère, la fenêtre sera identifiée par le seul signe >.

Les variables sont indiquées dans la description du masque avec les mêmes noms de variables que dans le programme basic qui exécutera le traitement.
Chaque variable est encadrée par des guillemets .
La relation entre chaque fenêtre de saisie, et chaque variable, est établie uniquement par la séquence de lecture des fenêtres d'une part, et des variables d'autre part :

1 ère fenêtre <------> 1 ère variable
2 ème fenêtre <------> 2 ème variable

La lecture du masque par le programme se fait ligne par ligne, de la gauche vers la droite et du haut vers le bas.

Dans un masque simple, il sera pratique d'indiquer les variables à l'intérieur même des fenêtres de saisie, ou immédiatement après, si la fenêtre est trop petite.

Dans un masque plus élaboré, il sera souvent plus clair de regrouper toutes les variables sur les dernières lignes du masque, là où il n'y a pas de fenêtres de saisie.

Les codes de programmation utilisés dans les masques sont décrits au chapitre IV du manuel d'utilisation.

CREATION : C

Cette commande est utilisée pour créer un masque nouveau.
Saisir le nom du masque, le numéro de drive sur lequel il sera conservé, le nombre de colonnes écran 40 ou 80, et la letrre C pour Création.
Le programme verifie d'abord que le nom donné n'existe pas déjà pour un masque se trouvant sur le disque. S'il existe il affiche le message MASQUE EXISTANT et envoie le curseur sur la rubrique numéro de drive.
En effet, vous pouvez conserver deux masques sous le même nom, à condition que les deux masques ne résident pas sur le même drive.

Le nombre de colonnes écran 40 ou 80 indiqué, fera passer automatiquement l'écran dans le format indiqué.

Le programme affiche un écran vide, avec le curseur en haut à gauche et vous pouvez dessiner le masque.

Voir le chapitre IV pour les codes écran.

Appuyer sur ESCAPE pour valider la saisie du masque, et le sauvegarder sur le disque.

Il est toujours possible d'abandonner la saisie du masque par CTRL/A.

Notez que même si vous n'avez rien écrit sur l'écran de saisie du masque, le fichier masque est crée sur le disque et contient la description d'un écran blanc.

Destruction : D

Permet de détruire, ou, supprimer le masque nommé sur le disque. Le programme visualise d'abord le masque à détruire et affiche en bas le message :

OK POUR DETRUIRE CE MASQUE (O/N)

Taper ''O'' pour confirmer que vous désirez le détruire.

Toute autre touche annule la procédure, et vous retournez au menu.

Lire : L

La commande L Lire, permet de lire un masque sur le disque,et de le charger en mémoire.
A ce stade le masque ne peut pas être modifié, mais il peut être sauvé par la commande S Sauver, soit sous le même nom, mais sur un autre disque, soit sous un autre nom sur le même disque.
Après S on a la main pour modifier le masque si on veut.

L'utilisation successive des commandes L et S permet donc de créer toute une famille de masques différents, mais ayant des parties communes, avec une très grande rapidité. Cette manière de procéder garantit une unité de présentation dans tous les masques d'une même application.

Nous vous conseillons de toujours inclure le nom du masque, dans le dessin du masque lui-même, afin d'éviter des confusions pendant la phase de mise au point du programme. Cette pratique facilite également le dialogue entre le programmeur et les utilisateurs de l'application.

Modification : M

Cette commande permet de modifier un masque déjà résidant sur disque. Le nombre de colonnes de l'écran 40 ou 80, est sans effet sur le masque à modifier.
Saisir le nom du masque, le numéro du drive, et la lettre M pour modifier.

Si le masque nommé n'existe pas sur le disque, le message MASQUE INEXISTANT est affiché, et le curseur revient sur la rubrique ''Nom du masque''.

Le programme charge en mémoire le masque à partir du disque, et l'affiche, avec le curseur en haut à gauche de l'écran, et vous pouvez modifier le masque.
Voir le chapitre IV pour les codes écran.
Appuyer sur ESCAPE pour la saisie du masque, et le sauvegarder sur le disque.

Les modifications saisies en mémoire seulement, sont abandonnées si vous appuyez sur CTRL - A pour finir la saisie.

Sauver : S

La commande S Sauver est utilisée en association avec la commande L pour sauver sur disque, le masque qui est en mémoire.
Charger d'abord le masque en mémoire par la commande L.
Pour sauver, taper S et appuyer sur RETURN.
Le masque est affiché à l'écran. Appuyer sur ESCAPE pour valider la sauvegarde.

Catalogue : *

Affiche la liste des objets se trouvant sur le disque désigné.
Voir description de l'option 7 du menu des utilitaires.

Explication : E

Cette commande affiche à l'écran une brève explication concernant les commandes :

SUR QUEL ORDRE DESIREZ-VOUS DES EXPLICATIONS ?

Répondre en tapant la lettre initiale de la commande et faites RETURN.
Après affichage de l'explication, appuyer sur la touche RETURN pour revenir à l'écran des commandes de gestion des masques.

Hard copy : H

La commande H Hard copy déclenche l'impression sur papier de la description du masque.
Si l'imprimante n'est pas connectée, le masque est affiché à l'écran pendant quelques instants, et on revient au menu.

OPTION 3 - GESTION DE FICHIERS

L'utilitaire gestion de fichiers permet de lire, modifier, détruire les enregistrements d'un fichier existant sur le disque.

L'écran affiche :	Réponse à saisir :
NOM DU FICHIER	Saisir le nom du fichier sur disque. Appuyer sur RETURN.
LECTEUR	Saisir le numéro du drive sur lequel le fichier est résidant. Appuyer sur RETURN.

Si le nom du fichier n'existe pas sur le disque indiqué ou si le numéro de drive est incorrect, le programme émet un bip d'avertissement et retourne au premier masque de saisie de gestion des fichiers.

Si tout est correct, le programme ''ouvre'' le fichier, charge en mémoire le dictionnaire du fichier et affiche un 2 ème masque de saisie.

Le nom du fichier et le numéro de drive sont affichés sur la 1 ère ligne de l'écran, sur fond inversé.

En dessous apparaît le message

SELECTION DE LA CLE

et, sur fond inversé, le nom de la variable clé, et sa longueur.

Dans le fichier DEMO, la variable est MO$ et la longueur 10
Saisir la valeur de la clé
Par exemple : MEMSOFT

COMMANDE
(LECTURE, MODIF, CREER, DETRUIRE, SUIVANT, FIN)
Saisir l'initiale de la commande :

L pour Lire
M pour Modifier
C pour Créer
D pour Détruire
S pour lire l'enregistrement Suivant
F pour Fin

Si la clé saisie n'existe pas, le message

... INEXISTANT est affiché

Si la commande saisie est erronée, le message

ORDRE ERRONE est affiché !

Si tout est correct, le programme affiche l'enregistrement correspondant.
Les noms des variables sont affichés à gauche sur fond inversé et les données sont affichées à droite de chaque variable.
Si la commande est C ou M, vous pouvez modifier toutes les données.
Les données restent en mémoire en passant d'un enregistrement à l'autre, ce qui peut faciliter la création d'une série d'enregistrements ayant des caractéristiques identiques.
Appuyer sur RETURN après avoir traiter un enregistrement.
L'écriture sur disque se fait alors après enregistrement traité.

NB : Les éléments d'une variable tableau, apparaissent sur une même ligne séparés par des points-virgules.

Si la commande est D pour Détruire un enregistrement, le programme afiche un message avant destruction :

OK POUR DETRUIRE (O/N) ?

Répondre O pour OUI
Répondre N pour NON

OPTION 4 - DEMO MASQUES
==========

Cette animation est réalisée à partir d'un masque.
Vous pourvez l'interrompre par CTRL C.

OPTION 5 - DEMO FICHIERS
==========

Cette option vous permet de travailler sur un fichier nommé DEMO (nom, adresse ...)

OPTION 6 - COPIE DE DISQUE

Ce programme réalise la copie d'une disquette complète sur une autre disquette, avec ou sans formattage préalable.

Il nécessite donc la connexion d'un lecteur de disque externe sur l'Apple //c.

Afin de vous éviter des erreurs de manipulation, nous vous conseillons de prendre l'habitude de faire vos copies de disque toujours dans le même sens, par exemple à partir du lecteur intégré (drive 0) vers le lecteur externe (drive 1).

Pour effectuer une copie de disquette, prendre l'option 6 du menu des utilitaires.
Insérer les disquettes dans les drives respectifs.

l'écran affiche :	Réponse à donner :
ORIGINAL EN	Numéro de drive du disque à copier. Valeurs possibles : 0 ou 1 Remarque : MEM/DOS affiche une variable numérique de valeur zéro, comme si c'était un blanc. Appuyer sur RETURN pour continuer.
COPIE SUR	numéro de drive du disque sur lequel les fichiers vont être copiés. Valeurs possibles : 1 ou 0 Le numéro du drive COPIE doit être différent du numéro du drive ORIGINAL, sinon un message d'erreur sera affiché à la fin de la saisie. Appuyer sur RETURN pour continuer : FORMATTAGE DE LA COPIE (O/N) Le programme vous laisse le choix de formater ou de ne pas formater la disquette copie, avant d'effectuer la copie. Valeurs possibles : O pour OUI N pour NON Appuyer sur RETURN pour continuer. Le programme commence alors par formater le disque copie, si on l'a demandé, puis effectue la copie, cylindre par cylindre. Lorsque la copie est terminée, l'écran affiche OK ENCORE UNE COPIE Vous pouvez à ce stade, mettre d'autres disquettes et refaire une copie, ou bien mettre fin au programme copie de disque par CTRL - A.

OPTION 7 - CATALOGUE

Le programme permet d'afficher le catalogue des disquettes, c'est à dire, la liste des objets qui sont enregistrés sur les disquettes.

La liste affichée sera classée automatiquement par types d'objets, et dans l'ordre alphabétique des noms.

Le programme est lancé par l'option 7 du menu des utilitaires.

L'écran affiche :	Réponse à donner :
CATALOGUE DISQUE 1	Taper le numéro du drive sur lequel se trouve la disquette à examiner, soit 0 pour le lecteur intégré, 1 pour le lecteur externe. Appuyer sur RETURN. L'écran affiche les noms des premiers objets du catalogue, précédés du type d'objet : B : fichier Binaire F : fichier de données G : fichier Global M : Masque P : Programme Basic Appuyer sur RETURN pour afficher la suite. Lorsqu'on arrive à la fin du catalogue, on retourne automatiquement à l'affichage du menu général.

OPTION 8 - FORMATAGE DISQUE

Cet utilitaire est nécessaire pour formater un disque.

L'écran affiche :	Réponse à donner :
FORMATAGE DISQUE 1	Saisir le numéro du drive sur lequel se trouve le disque à formater. Sous MEM/DOS les numéros du drive sont différents des numéros de drive sous DOS ou PRO DOS. Le lecteur interne a le numéro 0 (zéro). Le lecteur externe a le numéro 1. Appuyer sur RETURN.
ETES-VOUS SUR (O/N)	Confirmer l'action de formatage par O = OUI. Si vous taper N = NON, le formatage n'est pas exécuté. Appuyer sur RETURN.

OPTION 9 - COPIE AUTOMATIQUE

Ce programme copie d'un drive sur un autre, n'importe quel objet : programme, masque, global.
Le disque de destination n'est pas formaté dans ce programme.

L'écran affiche :	Réponse à donner :
ORIGINAL EN	Saisir le numéro de drive à partir duquel on désire copier. Valeurs possibles : de 0 ou 1 Appuyer sur RETURN
COPIE SUR	Saisir le numéro du drive sur lequel on désire copier. Valeur possible : 1 ou 0 Le drive copie doit être différent du drive original. Nous vous conseillons, afin de vous éviter des erreurs, de toujours faire les copies à partir du lecteur intégré 0, vers le lecteur externe 1.
VOULEZ-VOUS UNE SELECTION (O/N)	Le programme vous laisse le choix de copier tous les objets du disque d'origine. Taper N pour NON sélection Taper O Pour OUI sélection Si sélection : O OUI
TYPE NON A COPIER (O/N)	Le programme lit un par un les noms des objets du catalogue du disque d'origine, et s'arrête chaque fois pour vous demander si l'objet proposé doit être copié. Répondre par O (OUI) ou N (NON). L'ordinateur émet un bip, et vous propose aussitôt l'objet suivant du catalogue, jusqu'au dernier. Les noms des objets sélectionnés sont mis en mémoire.

Si sélection : N NON

Le programme met en mémoire les noms des objets du catalogue sans laisser le choix de sélection à l'utilisateur.

A la fin de la lecture du catalogue, le programme copie automatiquement tous les objets sélectionnés, l'un après l'autre, en affichant au passage leur nom sur l'écran.

OPTION 0 - TESTS
==========

Les choix 1, 2, 3 permettent d'effectuer des tests sur le contenu d'une disquette.

1 ANAL PISTE : contrôle les pistes allouées.
2 SUPER CAT : fait le catalogue des fichiers et contrôle les pistes.
3 VERIFY : vérifie la cohérence de la DCB
4 INFORMATIONS :renseigne sur les choix 1, 2, 3
5 FIN : ramène au menu général