!cj
TABLE DES MATIERES
!fj

1. PRESENTATION DE LA CARTE M/DOS 6502:

La carte M/DOS 6502 est un nouveau système d'exploitation disponible sur APPLE. Elle constitue une aide puissante pour le réalisateur de logiciels de gestion. Elle permet en effet une gestion de fichiers beaucoup plus complète qu'en DOS standard. Il est possible, par exemple de créer des fichiers pouvant contenir jusqu'& 10 clés, dont toutes les tables d'index et les procédés de recherche par clé sont assurés automatiquement par le système. La carte M/DOS 6502 donne en outre la possibilité au programmeur de créer des masques pour l'entrée et la sortie de variables. L& encore, l'affichage des textes, les contr&les portant sur la longueur, la localisation et le type des variables sont effectués par le système. Avec les masques, l'utilisateur (et le programmeur!) disposent d'une gestion d'écran particulièrement souple et efficace.

Les avantages du M/DOS 6502 sont alors aisés à discerner: gain de place, puisque tous les programmes du M/DOS 6502 sont directement sur carte et n'ont pas à &tre chargés en RAM utilisateur. Gain de temps, également, aussi bien pour le programmeur qui n'a plus à se soucier de la gestion de ses tables d'index ou des contr&les à effectuer lors de la saisie de variables, que pour l'utilisateur qui verra ses temps de recherche substanciellement diminués et sa gestion de l'écran grandement facilitée. La carte M/DOS 6502 est particulièrement indiquée pour les programmes de gestion, gros consommateurs de fichiers volumineux et complexes et volontiers amateurs de gestions d'écran sophistiquées. De plus, en n'obligeant pas le programmeur à avoir recours à des routines de tri et de gestion de pointeurs en assembleur, ou encore à des utilitaires spéciaux pour la saisie des données, la carte M/DOS 6502 permet de diminuer fortement le temps de réalisation des programmes, et par conséquent, leur co&t. De plus, la grande facilité d'emploi des fichiers et des masques M/DOS 6502 rend la maintenance d'un programme beaucoup plus légère.

2. COMPATIBILITE DE LA CARTE M/DOS 6502 AVEC D'AUTRES CARTES OU PERIPHERIQUES:

2.1. CARTES LANGAGES:

Le système M/DOS 6502 n'est utilisable qu'avec le Basic. Il y a en outre, pour le moment, une incompatibilité physique entre la carte M/DOS 6502 et la carte langage. Elles ne devront donc pas &tre connectées ensemble. Une version prochaine du sytème M/DOS 6502 permettra de donner la priorité à la carte langage et d'éviter les manipulations de cartes.

Il est & noter que le BASIC ne subit aucune modification, si ce n'est qu'il faut changer les instructions de gestion de fichiers et les instructions "système" spécifiques au M/DOS 6502. En ce qui concerne les masques, pas de problème de conversion, puisque ces derniers n'existaient pas en DOS standard.

2.2. COMPATIBILITE ENTRE LE DOS 3.2 ou 3.3 ET LE M/DOS 6502:

Si les programmes écrits en BASIC APPLESOFT restent & quelques modifications près compatibles avec M/DOS 6502, il n'en va pas de m&me pour les fichiers stockés sur un support formatté pour le DOS standard. Il est en effet impossible en utilisation normale de recopier des fichiers crées sous DOS 3.2 ou 3.3 sur un support formatté pour M/DOS 6502, qu'il s'agisse d'un fichier de texte, d'un fichier binaire ou d'un fichier programme. Il faut pour cela avoir recours à un utilitaire (non vendu avec la carte) qui permet l'emploi simultané des commandes DOS et M/DOS 6502. Il suffit alors de lire un fichier DOS et de le réécrire selon les normes M/DOS 6502.

Bien entendu, la carte M/DOS 6502 ne supprime absolument pas le DOS standard. Pour passer de l'un & l'autre, il suffit d'enlever la disquette ou le disque formatté pour M/DOS 6502 et de placer dans le lecteur un support standard. Après remise en route du système (PR£s avec s=numéro du slot sur lequel se trouve votre carte contr&leur), vous pourrez de nouveau travailler sur le DOS standard, sans avoir besoin pour cela d'enlever la carte M/DOS 6502 de son SLOT.

2.3. MEMOIRES PERIPHERIQUES:

En ce qui concerne les mémoires de masse, il n'y a pas de problèmes: M/DOS 6502 est compatible avec tous les types de disquettes ou disques. Il faudra cependant, pour des mémoires de masse autres que les disquettes APPLE, écrire des programmes spécifiques permettant communiquer avec l'unité de disques, et de lancer le système. Ces programmes, ainsi que certains pointeurs décrivant l'unité de disque devront &tre implantés correctement pour pouvoir &tre utilisés par M/DOS 6502. On pourra consulter, pour plus de détails, le chapitre sur les principes et les adresses du M/DOS 6502.

2.4. CARTES 80 COLONNES:

M/DOS 6502 est en principe utilisable avec toute carte 80 colonnes ou terminal intelligent. Il faut là aussi implanter certains pointeurs décrivant l'écran et des routines de gestion de cet écran. Le chapitre sur les principes et les adresses du M/DOS 6502 donne plus de détails à ce sujet.

La société MIS a développé des programmes assurant la compatibilité du système M/DOS 6502 et de la carte SUP'R' TERMINAL. Vous pouvez vous procurer la disquette contenant ces programmes chez votre fournisseur habituel.

Vous pouvez aussi vous procurer une disquette contenant un programme simulant un écran 80 colonnes. Ce programme permet de gérer deux pages d'écran de 40 colonnes chacunes. Il n'est pas possible de visualiser simultanément les deux pages, mais l'on passe très facilement de l'une à l'autre, d'o& l'appellation de 80 colonnes simulées.

2.5. SYSTEMES MULTIPOSTES:

Ces systèmes, o& plusieurs micros partagent les ressources d'un disque dur, lequel est géré par un micro multiplexeur, peuvent &tre utilisés sous M/DOS 6502. Chacun des postes de travail ainsi que le multiplexeur

devront &tre équippés d'une carte M/DOS 6502. L'annexe décrit les options de la syntaxe M/DOS 6502 utiles en mode multipostes pour éviter les "conflits" entre les différents postes.

Pour toutes ces implantations particulières, une bonne connaissance du système M/DOS 6502 est indispensable. Nous vous conseillons de vous orienter vers des matériels pour lesquels les routines de compatibilité et des programmes d'implantation sont déjà disponibles à la vente. En cas de problèmes n'hésitez pas à consulter votre fournisseur habituel.

!cj
CHAPITRE II: INSTALLATION D'UNE CARTE M/DOS 6502:
!lj

1. MONTAGE DES REPROMS SUR LA CARTE M/DOS 6502:

Chaque reprom possède une petite encoche à l'une de ses extrémités. Cette encoche doit toujours &tre placée en haut de la carte, du c&té opposé aux contacts de branchement sur un connecteur (slot) APPLE.

Les huit reproms doivent &tre placés, en regardant la carte du c&té des composants avec les contacts de connection tournés vers le bas, de gauche à droite dans l'ordre suivant:

D81 D01 D80 D00 F8 F0 E8 E0

ATTENTION: Manipulez les reproms avec précautions. Après l'insertion des reproms dans leurs supports, vérifiez que toutes les broches de chacune d'elles sont bien en place et bien enfoncées.

2. MISE EN PLACE D'UNE CARTE M/DOS 6502:

Avant toute manipulation vous devez IMPERATIVEMENT:

--> éteindre votre appareil en basculant l'interrupteur marche-arr&t sur la position arr&t

--> débrancher de la prise de courant votre cordon secteur.

Vous pouvez maintenant ouvrir votre appareil en en retirant le couvercle. Sur le bac inférieur, au fond (donc à l'opposé du clavier), se trouve un ensemble de 8 connecteurs (parfois appelés slots). Ceux-ci sont numérotés de 0 à 7, de la gauche vers la droite. Regardez: les numéros sont imprimés. Choisissez un connecteur inutilisé, le 3 par exemple (le 1 est généralement utilisé pour une imprimante et le 6 pour la carte d'interface avec les lecteurs de disquettes). Vous remarquez que la zone de contacts à introduire dans le connecteur est presque situé à une extrémité de la carte et partage donc cette dernière en deux partie de longueurs inégales. Fixez maintenant votre carte sur le connecteur que vous avez choisi, de telle sorte que l'extrémité de la carte la plus

éloignée des contacts soit dirigée vers le clavier conformément au schéma suivant:

ATTENTION: Ne jamais toucher avec les doigts les surfaces dorées des contacts du connecteur et de la carte. En effet, tout dép&t gras risquerait de compromettre sérieusement le fonctionnement de ces contacts de haute qualité et de haute précision. S'ils étaient encrassés, nettoyez les avec un chiffon imbibé d'alcool à br&ler ou avec un produit spécial pour les contacts.

Ne jamais exercer d'efforts susceptibles de déformer les contacts.

Ne jamais exercer de flexionssur votre carte, vous risqueriez d'en couper les pistes.

CHAPITRE III : GENERALITES SUR LA SYNTAXE M/DOS 6502

Ce chapitre décrit quelques points fondamentaux de la syntaxe du M/DOS et de sa logique.

1. LA LOGIQUE DES INSTRUCTIONS:

Les instructions du système M/DOS sont utilisables aussi bien dans des programmes qu'en mode direct.
Il existe 4 types d'ordres (LET, SAVE, RUN, LOAD) qui sont suivis d'une commande contenue dans une cha&ne de caractères, et par conséquent entourée de guillemets. On peut se dispenser de fermer les guillemets quand il n'y a qu'un ordre M/DOS dans une ligne de programme.

exemples:

```
LET"£0-1,F,0:VD1    qui ouvre le fichier "VD1" stocké sur le drive 0

LOAD"0:PROG         qui charge le programme "PROG" stocké lui aussi
                    sur le drive 0
```

Le fait que les ordres soient contenus dans des cha&nes de caractères permet de les paramétrer en suivant pour cela la syntaxe BASIC concernant la concaténation des cha&nes.

Dans l'exemple d'ouverture du fichier VD1, on aurait pu écrire:

```
LET A$      avec: A$="£0-1,F,0:VD1
```

La cha&ne de caractères peut très bien &tre la somme de plusieurs sous-cha&nes. Dans l'exemple précédent, l'instruction pourrait prendre la forme suivante:

```
LET"£"+A$+"-1,F,0:"+B$     avec: A$="0"  et B$="VD1"
```

La cha&ne qui suit le guillemet comprend des caractères ou des mots représentant une information significative pour le système (0,1,F,0,VD1 dans l'exemple de A$), et des séparateurs qui les isolent.
Les séparateurs suivants sont utilisables et interchangeables:"," "-" ":" ";"

exemple: LET"£0-1,F,0:VD1" peut aussi s'écrire LET"£0;1-F:0;VD1

Les blancs ne sont pas pris en compte dans les instructions sauf s'ils sont insérés dans le nom d'un objet. Seul le premier caractère suivant un séparateur est significatif, sauf évidemment lorsque le système attend un nom. On pourra donc utiliser les formulations explicites ou abrégées, en se rappelant que ces dernières tiennent toujours en 1 lettre.

```
exemple:    OPEN       ou      O
            NEW        ou      N
            FICHIER    ou      F
            OUTPUT     ou      O
```

Dans la suite de ce manuel, nous utiliserons la formulation complète lors de la présentation des instructions, et l'abrégée pour les exemples. Nous avons également choisi par convention de faire précéder le numéro d'unité logique par "-" et le nom de l'objet par ":".
!np

2. IDENTIFICATION DES OBJETS:

Leur nom est une suite d'au plus 21 caractères quelconques.
Les différents types d'objets sont identifiés, dans les catalogues du disque et dans certaines instructions, par les lettres clés suivantes:

P pour les programmes
B pour les modules binaires
M pour les masques
G pour les masques globaux
F pour les fichiers

Pour utiliser les masques et les fichiers, il faudra, après, bien s&r, les avoir créés, les ouvrir en leur affectant un numéro d'unité logique qui permettra au système de les identifier. Si, par exemple, un fichier a été ouvert sous l'unité logique 1, tout ordre faisant référence à l'unité logique 1 concernera ce fichier tant qu'il n'aura pas été fermé. On peut choisir, comme identificateur d'unité logique, n'importe quel caractère à l'exception du $.

3. NUMEROTATION DES DISQUES:

Elle commence en M/DOS à zéro. On établira donc la correspondance suivante:

numéro du disque :

0	contr&leur sur slot 6, premier lecteur
1	contr&leur sur slot 6, deuxième lecteur
2	contr&leur sur slot 5, premier lecteur.....

Si dans une instruction nécessitant un numéro de drive, celui-ci est omis, le système prendra par défaut la dernière valeur précisée dans une instruction.

```
exemple: LOAD"0:TRI
         SAVE"TRI     sauve sur le disque 0
```

ATTENTION: Pour que le système puisse comprendre l'absence de numéro de disque, il faut que le premier caractère du nom de l'objet concerné par l'opération ait un code ASCII supérieur à 9.

4. DETECTION DES ERREURS: LA VARIABLE STATUS

On distingue deux catégories d'erreurs: celles qui n'interrompent pas l'exécution du programme, et celles qui provoquent un arr&t (BREAK).

4.1. ERREURS RECOUVRABLES:

Après l'exécution d'un ordre M/DOS, un code permettant de détecter une éventuelle erreur rendant l'opération impossible est placé à l'adresse 189. Cette variable stockée en 189 en décimal (égale donc à PEEK(189)) sera appelée par la suite variable STATUS.
!np
Les différentes valeurs possibles du STATUS sont:

0 : L'opération s'est déroulée normalement: pas d'erreur.
1 : Abandon d'un masque (en création ou en saisie).
10 : Inexistant (objet ou enregistrement d'un fichier).
20 : Lecture impossible: la carte a une priorité trop faible.
30 : Existant déjà (en écriture d'enregistrement ou sauvegarde sur

disque).
255: Fin de fichier.

exemple: 10 SAVE"0:TEST
         20 ? PEEK(189)

Si un programme du nom de "TEST" existe déjà sur le support, le système affichera 30, 0 si, au contraire, tout se passe bien.

Ces erreurs ne provoquent pas d'interruption du programme. Il sera donc souvent utile de tester le contenu de la variable STATUS, afin de savoir si l'opération s'est bien passée. Comme il n'y a en général, pour chaque ordre, qu'un seul type d'erreur recouvrable, il suffira donc de vérifier que le STATUS est nul pour s'assurer que l'instruction a été exécutée normalement.
En revanche, en mode direct, si PEEK(189) est différent de zéro, le message DIRECT ERROR est affiché.

4.2. ERREURS NON RECOUVRABLES:

Elles peuvent provenir d'une erreur de programmation ou d'une erreur système. Elles entra&nent une interruption du programme.
On pourra se référer, pour plus de détail, à l'annexe qui décrit les messages erreur.

5. BUFFERS DU M/DOS:

Il s'agit d'une zone RAM réservée au système. Ces buffers contiennent des pointeurs décrivant logiquement le disque, les fichiers et les masques ouverts à un instant donné.

En principe l'emplacement des buffers du DOS est fixé par le système. On peut toutefois le modifier en changeant le contenu de certaines adresses (cf chapitre sur les adresses du M/DOS).

Si un programme utilise trop de masques et de fichiers (surtout si ceux-ci sont multiclés), il risque de manquer de place dans les buffers. Dans ce cas, le programme sera interrompu par le message OUT OF MEMORY ERROR. On pourra alors récupérer de la place en fermant les masques et les fichiers dont le programme n'a plus besoin.
!np
Il est conseillé d'utiliser autant que possible les masques d'entrée/sortie. Ceux-ci prennent en effet de la place dans les buffers du DOS, à laquelle l'utilisateur n'a de toute fa&on pas accès, mais lui permettent de gagner la place que prendraient les instructions d'entrée/sortie dans les zones RAM réservées au programme et à ses variables.

6. PARTICULARITE DU "THEN" SOUS M/DOS:

Lorsque "THEN" doit &tre suivi d'un ordre M/DOS, il faudra faire précéder ce dernier de ":".

exemple: IF A=B THEN : LET"£0-1,F,0:FILE

Si A=B on ouvre le fichier "FILE"

En revanche le ":" n'est pas nécessaire après "THEN" lorsque celui-ci est suivi d'une instruction BASIC standard.

exemple: IF A=B THEN GOTO 30 est tout à fait correct.

CHAPITRE IV: INSTRUCTIONS GENERALES

1. MISE EN ROUTE DU SYSTEME:

Il faut pour mettre en route le système M/DOS utiliser une disquette de type "système" permettant de le faire démarrer. La disquette MASTER, comprenant les programmes utilitaires, qui vous est livrée avec la carte, est de ce type . Vous n'aurez cependant pas forcément besoin de l'utiliser chaque fois que vous voudrez mettre en route le système: en effet, n'importe quelle disquette peut mettre en oeuvre M/DOS pourvu qu'elle ait été formattée et initialisée par l'utilitaire BOOT (cf paragraphe sur le formattage des disquettes pour plus de détails).

Pour charger M/DOS, placez une disquette de type "système" dans le lecteur 0 (contr&leur sur le slot No 6, drive No 1) et fermez la porte. Mettez maintenant votre appareil sous tension. Si celui-ci est muni d'une ROM autostart, le système se chargera de lui-m&me et le message suivant s'affichera sur l'écran:

```
M/DOS 6502 - V4
MICRO INFORMATIQUE SERVICE
P. LAFFITTE
P. NESNIDAL
```

Si de plus un programme baptisé "HELLO" figure sur la disquette, il s'exécutera à la suite de ce message.

Si vous n'avez pas de ROM autostart, il vous suffit de taper:

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

Cette séquence provoque le chargement de M/DOS, l'affichage du message précédent et éventuellement l'exécution du programme "HELLO".

Si votre appareil est déjà sous tension lorsque vous voulez mettre en route M/DOS, tapez PR£6.

2. FORMATTAGE DES DISQUETTES:

Pour pouvoir utiliser une disquette sous M/DOS, il faut que celle-ci ait été formattée au préalable. Le formattage consiste à inscrire sur la disquette les valeurs nécessaires au système pour retrouver les pistes et les secteurs. Il peut se faire de deux fa&ons: formattage d'une disquette de type "fichier" ou de type "système".

2.1. FORMATTAGE D'UNE DISQUETTE DE TYPE "FICHIER":

Une telle disquette ne permet pas de mettre en route le système M/DOS. Il faudra donc, pour l'utiliser, s'&tre servi auparavant d'une disquette de type "système" pour charger M/DOS. Les disquettes de type "fichier" permettent, par rapport à celle de type "système", de gagner la place prise sur ces dernières par le programme de mise en route du système (BOOT) qui y occupe la piste 0.
!np

L'instruction de formattage est la suivante:

LET"£FORMAT,(d)

d étant le numéro du lecteur dans lequel se trouve la disquette à formatter. La numérotation des lecteurs est la suivante:

numéro 0 : contr&leur sur le slot 6, drive 1
numéro 1 : contr&leur sur le slot 6, drive 2
numéro 2 : contr&leur sur le slot 5, drive 1

exemple: LET"£F,0 formatte la disquette située dans le lecteur No 0.

Bien s&r cette instruction est un ordre M/DOS. Vous ne pourrez l'utiliser que si le système est dèjà chargé. Elle est valable quel que soit le support implanté correctement sous M/DOS 6502.

Ne vous étonnez pas si la durée du formattage est relativement longue: c'est normal. Le voyant lumineux du lecteur s'allume pendant l'opération; une fois qu'elle est terminée, le système rend la main à l'utilisateur.

Si le formattage échoue, le système envoie le message: ?DATA ERROR. Cet échec peut provenir d'une des causes suivantes:

--> Le lecteur ne contient pas de disquette, ou il n'est pas fermé.
--> La disquette est protégée en écriture.
--> La disquette est endommagée et donc inutilisable.

2.2. FORMATTAGE ET INITIALISATION D'UNE DISQUETTE DE TYPE "SYSTEME":

Rappelons que ce type de disquettes, contrairement à celles de type "fichier", permettent de démarrer M/DOS. Elles doivent donc, non seulement &tre formattées, mais aussi contenir un programme qui charge le système, et qui sera stocké sur la piste 0. Les disquettes APPLE peuvent &tre formattées et initialisées grace à l'utilitaire dénommé BOOT figurant sur la disquette MASTER. Pour d'autres types de disques, il faudra se référer à la notice d'implantation fournie par le constructeur de l'unité de disques, et écrire un programme de chargement spécifique.

3. INSTRUCTIONS DE COMMUNICATION AVEC LES UNITES DE DISQUE:

3.1. CATALOGUE DU DISQUE:

LET"*,(d) donne le catalogue complet du disque numéro d

LET"*,(d),(B/F/G/M/P) donne le catalogue des modules binaires (B), des fichiers (F),des masques (G pour les globaux, M pour les masques simples), des programmes (P).

LET"£REORG-$(d) réorganise le catalogue du disque. L'accès aux différents modules contenus sur le disque se faisant en séquentiel indexé, la réorganisation permet d'accélérer l'exécution des ordres LOAD,SAVE,OPEN,NEW.

!np

3.2. PLACE DISPONIBLE SUR UN DISQUE:

LET"%(d)" (d):numéro du drive

permet de conna&tre le nombre de pistes restant disponibles sur la disquette. Cette instruction provoque le stockage de ce nombre à l'adresse 189 qu'il faudra lire ensuite.

exemple: LET"%0" : ? PEEK(189) affiche le nombre de pistes libres.

En mode direct, si le nombre de pistes libres est différent de 0, l'ordre LET"%(d) entra&ne l'affichage du message: ?DIRECT ERROR. Ce message n'a pas d'importance; l'utilisateur n'en tiendra donc pas compte et pourra aller lire le contenu de l'adresse 189 pour obtenir le renseignement désiré.

exemple:
```
  LET"%0
? DIRECT ERROR
  ?PEEK(189)
17
```

Chaque disquette contenant 34 pistes, dans l'exemple précédent le taux d'occupation est donc de 50%.
ATTENTION: au formattage, le nombre de pistes est stocké sur un octet, dont le contenu est décrémenté de 1 chaque fois qu'une nouvelle piste est occupée. Cette instruction ne pourra donc fonctionner correctement que pour des unités de disques ne comportant pas plus de 256 pistes.
Cependant l'utilitaire DEBUG.MENU permet de conna&tre avec précision l'état d'occupation du disque quel qu'en soit le modèle.

3.3. SAUVEGARDE SUR DISQUE D'UN PROGRAMME RESIDANT EN MEMOIRE CENTRALE:

3.3.1. PROGRAMMES EN BASIC:

SAVE"(d):(NOM) (d) numéro du drive
sauve le programme en cours sous le nom indiqué. S'il existe déjà sur le disque un programme homonyme, le système renvoie un message erreur pour indiquer qu'il ne peut pas réaliser l'opération demandée.

SAVE"&(d):(NOM)
sauve le programme en cours en écrasant éventuellement un programme figurant sur le disque sous le m&me nom. En revanche,un masque ou un fichier portant un nom identique ne sera pas écrasé.

3.3.2. PROGRAMMES BINAIRES:

SAVE"$XXXX,$YYYY,(d):(NOM)

XXXX:adresse minimale en hexadécimal.
YYYY:adresse maximale

exemple: SAVE"$1000,$1100,0:BIN" sauve sous le nom de "BIN" la partie de la mémoire comprise entre les adresses $1000 et $10FF (=1100-1 en hexadécimal).
!np
3.4. CHARGEMENT EN MEMOIRE D'UN PROGRAMME STOCKE SUR DISQUE:

3.4.1. PROGRAMMES EN BASIC:

LOAD"(d):(NOM) charge le programme en effa&ant le programme précédent.

LOAD"&(d):(NOM) charge le programme "NOM" à la suite de celui qui réside déjà en mémoire centrale. Les numéros de lignes du dernier programme

chargé doivent &tre supérieurs à celui de la dernière ligne de son prédécesseur. On pourra, le cas échéant, se servir de l'utilitaire RENUMEROTE pour modifier les numéros de lignes d'un programme qui sera ensuite chargé grace à cette instruction. Cette instruction pourra notamment servir à charger des sous-programmes communs à plusieurs modules de traitement.

3.4.2. PROGRAMMES BINAIRES:

LOAD"$XXXX,$YYYY,(d):(NOM)

XXXX: adresse de départ en hexadécimal
YYYY: adresse maximale autorisée

La deuxième adresse joue un r&le du butoir. Elle ne sera jamais dépassée, évitant ainsi d'écraser une autre partie de la mémoire qu'on veut préserver. Elle peut &tre supérieure à l'adresse de fin de chargement du programme, le LOAD se faisant dans ce cas tout à fait normalement.

A l'issue du LOAD, l'adresse de fin du module +1 est stockée en ADBLC avec:
ADBLC =$30E poids fort de l'adresse
ADBLC+1 =$30F poids faible de l'adresse.

3.5. EXECUTION D'UN PROGRAMME:

L'instruction RUN permet couramment de lancer l'exécution du programme résidant en mémoire. On notera en outre l'existence des trois instructions suivantes:

RUN"(d):(NOM) charge à partir du disque et exécute le programme NOM

LOAD"/(d):(NOM) charge et exécute sans réinitialiser les adresses réservées au variables, ce qui permet de conserver les valeurs de ces dernières. Le programme ainsi chargé doit occuper une place en mémoire inférieure à celle du dernier programme chargé normalement.

exemple: Soient les deux programmes suivants:

PROG1

```
10 INPUT "ENTREZ N:";N
20 PRINT "MERCI"
30 END
```

PROG2

```
10 FOR I=1 TO 5 : PRINT I+N
20 END
```

!np
Supposons que l'utilisateur ait, en exécutant PROG1, affecté la valeur 10 à la variable N. LOAD"/0:PROG2 fera afficher:
11
12
13
14
15

RUN"£(d):(NOM) charge le programme à la suite de celui figurant déjà en mémoire et exécute l'ensemble. Les numéros de lignes du programme ainsi chargé doivent &tre supérieurs à celui de la dernière ligne du précédent programme.

3.6. DESTRUCTION D'UN OBJET STOCKE SUR DISQUE:

LET"£DELETE,(F/G/M/P/B),(d):(NOM)

F pour un fichier
G pour un masque global
M pour un masque simple
P pour un programme
B pour un module binaire

exemple: LET"£D,P,0:PROG détruit le programme PROG sur le drive 0.

3.7. FERMETURE DES DRIVES ET CHANGEMENT DE DISQUES:

Pour accélérer le fonctionnement du M/DOS 6502, certaines zones de mémoire réservées (les buffers du DOS) contiennent des variables de travail qui décrivent logiquement disques, fichiers , masques... Si vous changez de disque sans en informer le système, les pointeurs ne correspondront plus à la nouvelle disquette, qui pourrait alors &tre détruite.

Il faudra donc, avant de changer de disquette, utiliser l'instruction LET"£C,"+CHR$(d) qui ferme le drive numéro d ainsi que tous les fichiers ouverts sur ce drive.

exemple:

```
10 LOAD"0:TEST"
20 INPUT "CHANGEZ DE DISQUETTE ET TAPEZ RETURN";A$
30 SAVE"0:TEST"
```

détruira la disquette.Il faut faire:

```
10 LOAD"0:TEST"
20 LET"£C,"+CHR$(0)
30 INPUT "CHANGEZ DE DISQUETTE ET TAPEZ RETURN";A$
40 SAVE"0:TEST"
```

!np
Une autre alternative pourra &tre, avant de changer de disquette, de nettoyer tous les buffers du DOS par l'instruction LET"£C,$. Cette solution entra&ne aussi une fermeture des masques qui n'est pas forcément nécessaire dans la solution précédente.

Ces procédures sont surtout valables lorsque qu'on travaille par l'intermédiaire de programmes. En effet, en mode direct, l'instruction LET"£C,"CHR$(d) est exécutée avant chaque ordre donné par l'utilisateur. Il n'y aura donc pas besoin, dans ce cas, de fermer le drive avant de changer de disquette. Il sera toutefois préférable de fermer les fichiers.

exemple: la séquence:

```
LOAD"0:TEST
changement de disquette.
SAVE"0:TEST
```

est ici tout à fait correcte, et ne détruira donc pas la disquette.

Il est possible, grace à l'ordre LET")M", d'emp&cher en mode direct la

fermeture systématique du drive. Cela permettra notamment d'utiliser les instructions de gestion de fichier (lecture, écriture...) qui ne marchent pas normalement en mode direct du fait de l'exécution automatique du CLEAR. L'instruction LET")" rend le CLEAR, le drive est de nouveau systématiquement fermé avant l'exécution d'un ordre en mode direct.

On notera enfin l'existence d'une instruction qui ferme tous les drives: LET"£C,$$

3.8. ACCES DIRECT SUR UN DISQUE:

L'instruction d'accès direct à un disque ou une disquette permet:

--> de formatter un disque
--> de lire directement un ou plusieurs secteurs du disque.
--> d'écrire un ou plusieurs secteurs sur ce disque.

Sa syntaxe est la suivante:

LET"$d,(C=)c,(N=)n,(P=)p,(T=)t,(S=)s,$XXXX

avec: XXXX adresse de début du chargement en hexadécimal
c opération à réaliser (1 pour lecture, 2 pour écriture, 3 pour formattage)
d numéro du drive concerné (0 à 5)
n nombre d'opérations à effectuer (forcément égal à 1 pour un formattage)
p numéro de la piste sur laquelle doit se trouver le premier secteur à lire ou écrire
t numéro de t&te
s numéro du premier secteur de la piste No p concerné par l'opération.

Les paramètres entre ( ) sont facultatifs, mais contr&lés s'ils figurent dans l'instruction.
!np
exemple:

LET"$0,C=1,N=5,P=3,T=0,S=0,$1000 et LET"$0,1,5,3,0,0,$1000 ont un effet identique: le chargement en mémoire, à partir de l'adresse $1000, des 5 premiers secteurs de la troisième piste de la disquette placée dans le lecteur 0 (slot 6, drive 1).

Si le nombre de secteurs à sauver est trop grand, compte tenu de la place restant disponible, l'écriture est effectuée normalement jusqu'à la fin du disque, puis la main est rendue à l'utilisateur avec un status différent de 0.

Après l'exécution de cette instruction, l'utilisateur pourra trouver aux adresses $30A, $30B, $30C les valeurs des paramètres piste, t&te et secteurs (p,t,s dans la définition de la syntaxe) du prochain secteur à lire ou écrire.

Cette instruction pourra faciliter la réalisation de certains utilitaires.

exemple: Programme copiant deux disques placés sur des drives de m&me type.

```
100 HIMEM:16*256
110 INPUT "NUMERO DU DRIVE D'ORIGINE:";D1$
```

```
120 INPUT "NUMERO DU DRIVE D'ARRIVEE:";D2$
130 LET"£C,$
140 X=777
150 P=0 : T=0 : S=0
160 VTAB 10 : PRINT " LECTURE:";P;"  ";T
170 LET"$"+D1$+",1,16,"+STR$(P)+","+STR$(T)+","+STR$(S)+",$1000"
180 VTAB 10 :PRINT "ECRITURE:";P;"  ";T
190 LET"$"+D2$+",2,16"+STR$(P)+","+STR$(T)+","+STR$(S)+",$1000"
200 IF PEEK(189) <>0 THEN 250
210 P=PEEK(X+1):T=PEEK=(X+2):S=PEEK(X+3):GOTO 160
250 PRINT "FIN":END
```

4. GESTION DU CLAVIER:

4.1. CLAVIER QWERTY:

La carte M/DOS fonctionne sur le clavier standard APPLE, à savoir le clavier américain ou "QWERTY".

4.2. CLAVIER AZERTY:

L'instruction LET"Z échange la signification de certaines touches pour permettre à l'utilisateur de disposer d'un clavier du type AZERTY. Il faudra échanger les capuchons des touches:

A	et	Q
W	et	Z
M	et	;

Le masque utilitaire AZERTY stocké sur la disquette MASTER montre les touches à inverser.
!np
Si, par erreur, vous tapez sur la touche RESET, le système reviendra au clavier du type QWERTY. Dans ce cas, faites:

CTRL/C RETURN
LET"Z --> Attention: tapez W si vous avez échangé les capuchons!

pour revenir au clavier AZERTY.

L'ordre LET"Z pourra &tre inséré dans le programme HELLO si on désire utiliser de fa&on permanente un clavier AZERTY.

4.3. SIGNAL SONORE:

L'instruction LET"G lorsqu'elle est exécutée provoque l'émission d'un signal sonore.

4.4. INVERION VIDEO DE L'ECRAN

LET"Y"

!1j

1. GENERALITES SUR LES FICHIERS

La carte M/DOS 6502 offre de nombreux avantages par rapport au DOS dans l'utilisation et la gestion des fichiers. Le principal de ces avantages est que M/DOS 6502 assure lui-m&me les procédures de recherche, de stockage et de destruction des enregistrements, ainsi que la gestion des pointeurs et des tables d'index.

En ce qui concerne la structure des fichiers, M/DOS 6502 offre un éventail plus large. Le DOS permettait la création de fichiers à accès séquentiel et à accès direct. M/DOS 6502 ne dispose pas de fichiers à accès séquentiel à proprement parler, mais de fichiers à accès direct pouvant &tre utilisés séquentiellement. S'y ajoute la possibilité d'avoir des fichiers séquentiels indexés (o& le pointeur de l'accès direct est remplacé par une clé, donnée directement par l'utilisateur) et des fichiers multiclés (qui possèdent plusieurs clés, comme leur nom l'indique).

1.1. NOMBRE DE FICHIERS UTILISABLES SIMULTANEMENT

Alors que le DOS ne gérait qu'un nombre limité de fichiers à la fois (3 en mode normal, 16 au maximum avec l'option MAXFILES), M/DOS 6502 inaugure un système d'ouverture par unités logiques, permettant d'utiliser un nombre variable de fichiers. Ce nombre n'est, en effet, limité que par la taille des BUFFERS DU DOS qui contiennent, entre autres, des descriptifs des fichiers ouverts. Evidemment, un fichier occupera d'autant plus de place dans les BUFFERS DU DOS que sa structure est complexe. Il est possible en cas de blocage d'augmenter la taille des BUFFERS en mordant sur la zone RAM qui sert au stockage des variables du programme. Dans ce cas, il ne faudra pas oublier de modifier le HIMEM (adresse maximale réservée aux variables du programme).

1.2. TAILLE DES FICHIERS ET DES ENREGISTREMENTS

La taille des fichiers est limitée à 64000 enregistrements, chacun pouvant &tre de longueur variable. Nul besoin donc d'indiquer une longueur fixe des enregistrements pour les fichiers M/DOS 6502, contrairement à ce qui se passe pour les fichiers à accès direct en DOS.

!np
1.3. STRUCTURE DES ENREGISTREMENT ET DEFINITION DES MOYENS D'ACCES

Quel que soit le type du fichier M/DOS 6502, la structure des enregistrements et des moyens d'accès est définie lors de la création en associant à chaque variable (que ce soit d'une rubrique, d'une clé ou d'un pointeur) un nom de variable BASIC. Prenons l'exemple d'un fichier dont la clé s'appellle CL$ et dont les enregistrements sont composés des variables V1% et V2%. Au moment de l'exécution d'instructions comme READ ou WRITE, le traitement se fait en prenant les valeurs de CL$, V1% et V2% au moment de l'exécution. La lecture va consister à affecter à V1% et V2% les valeurs lues en fonction de CL$ dans le fichier, l'écriture au contraire à stocker sur disque le contenu de ces m&mes variables.

Rappelons qu'en DOS, l'instruction READ n'était qu'une demi-instruction, dans ce sens qu'il fallait rencontrer un INPUT XX pour affecter la valeur lue à XX. L'ensemble des noms de toutes les variables qui composent, pour un fichier donné, le ou les moyens d'accès et l'enregistrement, est appelé DICTIONNAIRE.

1.4. VALIDATION DES OPERATIONS SUR LES FICHIERS

Toutes les opérations qui changent le contenu du fichier (destruction, modification, écriture) sont effectuées directement sur le disque au moment de l'exécution de l'instruction, et non au moment de la fermeture du fichier: ceci a deux conséquenses.

- en cas d'arr&t brusque du système, seule l'opération en cours peut ne pas &tre validée.
- il n'est pas indispensable de fermer un fichier après s'en &tre servi, la fermeture n'ayant pour effet que de libérer de la place en mémoire centrale. De plus, l'arr&t du système provoque une fermeture automatique de tous les objets en mémoire. La fermeture d'un objet particulier se fait par l'instruction: LET"CLEAR-<u.l>".

1.5. NOUVEAUX TYPES DE VARIABLES

Un autre avantage de M/DOS 6502 est l'existence de 2 nouveaux types de variables pouvant &tre utilisés dans la définition des enregistrements, aussi bien comme clés que comme variables: il s'agit des dates et des binaires.

!lm10
La variable "date" prend le format suivant: XX$*. Elle ne peut contenir qu'une cha&ne de caractères de la forme JJ/MM/AA, telle que JJ, MM et AA forment une date cohérente. AA va de 50 à 99 (1950-1999) et de 00 à 49 (2000-2049).

La variable "binaire" prend le format suivant: XX%/. Elle ne peut contenir qu'un entier compris entre 0 et 255 inclus (c'est-&-dire dont la représentation binaire tient dans un seul octet).

!lm5
!np
ATTENTION. Ces 2 variables ne sont acceptées que dans un fichier créé par M/DOS 6502. Dans un programme BASIC, DA$* et BI%/ deviennent respectivement la cha&ne DA$ et l'entier BI%. Si l'on essaye d'écrire dans le fichier une date incohérente, le sytème reformattera cette date en y ajoutant des 0 pour la rendre plausible (1249 --> 01/02/49). Si c'est impossible, un blanc sera inscrit à la place. De m&me, si l'on essaye d'introduire comme binaire un entier >255 ou négatif, c'est la valeur absolue de l'entier modulo 256 qui sera prise en compte (ex : -340 donne 84).

1.6. UTILISATION DE TABLEAUX DANS LES ENREGISTREMENTS D'UN FICHIER

autre nouveauté: les variables d'un enregistrement (mais non d'une clé) peuvent &tre des tableaux. Il n'est pas necessaire d'indiquer leur dimension au moment de la création du fichier car M/DOS 6502 les enregistre en taille variable sous forme de matrice creuse (cf paragraphe sur le codage des informations dans les fichiers pour plus de détails). Il suffit simplement dans le cas de tableaux de placer le signe ";" après

la variable. Exemple:

```
EN%;  tableau d'entiers
DA$*; tableau de dates.
```

!np

ATTENTION: Le système code les variables du tableau en fonction du dimentionnement choisi dans le programme lors de l'écriture ou de la modification de l'enregistrement, d'o& la possibilité de ne pas adopter le m&me dimentionnement dans les différents enregistrements. Mais, au moment de la lecture, le tableau ne peut &tre lu qu'avec un dimentionnement supérieur ou égal à celui sous lequel il avait été écrit. Si, par exemple, vous avez écrit 10 valeurs pour un tableau donné, le système relira les 10 valeurs et ne pourra bien évidemment pas les ranger dans un tableau dimensionné à 8. Dans ce cas le programme serait interrompu et le message BAD SUBSCRIPT ERROR s'afficherait.

En outre, le dimensionnement du tableau devra &tre précisé avant l'ouverture du fichier. Une absence d'instruction DIM, provoquera implicitement un dimensionnement à 10.

1.7. UTILISATION DES FICHIERS EN MODE DIRECT:

Nous tenons à rappeler ici, qu'en mode direct, le système effectue systématiquement une fermeture des drives entre deux ordres, ce qui emp&che normalement, sous ce mode, l'utilisation des instructions de gestion des fichiers. L'ordre LET")M" permet d'éviter cette fermeture automatique des drives. Réportez vous au chapitre 3, paragraphe 3.7, pour plus de détails à ce sujet.

!cj

-

!lj
2. STRUCTURE, CREATION ET UTILISATION DES DIFFERENTS TYPES DE FICHIERS

La procédure de création d'un fichier M/DOS 6502 n'obéit pas aux m&mes principes que celle des fichiers DOS. Avec le DOS, l'instruction OPEN a valeur de création quand le fichier n'existe pas et d'ouverture quand le fichier existe. M/DOS 6502 exige en revanche une phase préalable de création du fichier, au cours de laquelle on définit la structure de l'enregistrement et les noms des variables. Les structures possibles sont au nombre de 2: fichiers séquentiels relatifs et fichiers séquentiels indexés; ces derniers se subdivisant en fichiers à une clé et en fichiers multiclés. Nous allons à présent examiner en détail pour chacun de ces 3 types de fichiers la procédure de création et les possibilités d'utilisation. Nous vous proposerons un certain nombre d'exemples pratiques (programmes, jeux d'essai) que nous vous conseillons de faire tourner sur votre machine pour mieux vous familiariser avec certaines notions délicates. Nous nous sommes efforcés de prendre des exemples courts, mais qui utilisent un maximum d'options; rien ne vous interdit bien s&r de les développer et de les transformer.

2.1. LES FICHIERS SEQUENTIELS RELATIFS

2.1.1 CREATION

Le nom de "séquentiel relatif" vient du fait que ces fichiers peuvent &tre utilisés selon les 2 modes d'accès.

A chaque fichier de ce type correspond une variable entière appelée pointeur, qui permet d'accéder aux enregistrements, lesuqels sont définis au moment de la création du fichier par une liste de variables. Chaque enregistrement est identifié par une valeur du pointeur égale à son rang de création. Les pointeurs sont gérés par le système et ne peuvent pas &tre modifiés par l'utilisateur. Comme pour tous les fichiers M/DOS 6502, la taille maximum est de 64000 enregistrements. Les enregistrements sont gérés par le système en longueur variable, laquelle n'a donc pas à &tre précisée par l'utilisateur comme dans le DOS.

La création d'un fichier séquentiel relatif se fait comme toujours en M/DOS 6502 en 2 étapes.

a/ définition de l'enregistrement à l'aide de l'instruction:

```
LET "> §PO% = VAR1,VAR2,...VARn
```

> indique que l'on crée le dictionnaire

§ indique qu'il s'agit d'un fichier séquentiel relatif et non d'un fichier séquentiel indexé.

PO% est le nom du pointeur. Ce dernier doit toujours &tre défini sous forme d'entier.

= est le séparateur obligatoire entre le pointeur (la ou les clés dans le cas de fichiers séquentiels indexés) et la liste des variables.
!np
VAR1,VAR2...VARn est la liste des variables: il n'y a aucune contrainte quant au nombre et au type des variables, à condition que ces dernières soient d'un type reconnu par M/DOS 6502 (entier, réel, cha&ne, date et binaire, en variable simple ou en tableau). Seul, le séparateur est obligatoire.

Cette instruction crée ce qu'on appelle le dictionnaire, qui sert à définir la structure et le type du fichier. Le dictionnnaire peut &tre édité en clair après ouverture du fichier par l'instruction: LET "ENTER-<u.l>" et visualisé par l'instruction: LET "VISUALISE" (sans No d'unité logique). Ces deux instructions sont d'ailleurs valables pour tous les types de fichiers M/DOS 6502. Si le dictionnaire est trop long, on peut écrire la suite par l'instruction: LET"+<suite du dictionnaire>" juste après le LET">...". Ceci sert lors de la création en mode direct, car les ordres M/DOS 6502 sont limités à 55 caractères.

b/ création proprement dite du fichier à l'aide de l'instruction:

LET "£NEW-<u.l>,FICHIER,<d>:<nom du fichier>

Cette instruction ne peut venir qu'après une définition d'enregistrement (LET">...). Il est à noter que la commande LET"£NEW... est la m&me pour les 3 types de fichiers. L'impression d'un message erreur à ce niveau peut avoir plusieurs causes: unité logique non fermée, fichier existant déjà ou erreur de syntaxe dans l'instruction LET ">...

Exemple pratique: créons un fichier qui référence des prisonniers dans un pénitencier (cela vous changera des gestions de stocks...)

L'enregistrement se décompose de la fa&on suivante:

un pointeur, MA%, qui sera aussi le matricule du détenu.

Une liste de variables comprenant:
NO$: nom du détenu
DE$*: date d'entrée au pénitencier
DU%/: durée de la peine en mois (255 suffiront!)

```
10  LET "£CL,$  (pour s'assurer que toutes les u.l sont fermées)
20  LET ">$MA%=NO$,DE$*,DU%/
30  LET "£N-1,F,0:SINGSING"
40  LET "£CL,$" (on referme l'u.l.)
```

Exécutez ce programme et assurez vous en tapant LET "*,<d>,F que le fichier existe bien sur la disquette.
!np
2.2.2. UTILISATION

Venons-en à présent à l'utilisation du fichier, et, dans un premier temps, procédons au remplissage et à la relecture.

le programme va consister à écrire 3 nouvelles références dans le fichier et à tout relire pour voir ce qui a été écrit. Pour cela, nous utiliserons les instructions:

LET "£OPEN-<u.l> ouverture du fichier
LET "WRITE-<u.l> écriture d'un nouvel enregistrement à la fin du fichier.
LET "READ-<u.l> lecture directe d'un enregistrement
LET "NEXT-<u.l> lecture séquentielle d'un enregistrement.

De plus, nous serons ammenés à tester la variable STATUS pour la fin de fichier (PEEK(189)=255). Se référer au chapitre III pour l'emploi de la variable STATUS.

```
5   HOME: LET"£CL,$": LET "£0-1,F,0,SINGSING
15  REM: ECRITURE DE 3 NOUVELLES REFERENCES
20  FOR I=1 TO 3
30  INPUT "NOM: ";NO$
40  INPUT "DATE D'ENTREE: ";DE$
50  INPUT "DUREE DE LA PEINE: ";DU%
60  PRINT "POINTEUR=";MA%
70  LET"W-1
80  NEXT I
100  MA%=1: LET"READ-1
120  PRINT"NOM: ";NO$
130  PRINT"DATE D'ENTREE: ";DE$
140  PRINT"DUREE DE LA PEINE: ";DU%
150  LET "N-1" : IF PEEK (189) <>255 THEN 120
160  LET"£CL,$" : END
```

En 60, l'impression du pointeur permet de vérifier que l'écriture s'accomplit automatiquement en fin de fichier.

L'instruction READ est une instruction "RELATIVE". Le programme lit l'enregistrement correspondant au numéro du pointeur. Ici, nous l'avons remis à 1 (ligne 100) pour relire depuis le début du fichier.

En revanche, NEXT est une instruction "SEQUENTIELLE" qui lit les enregistrements les uns à la suite des autres sans tenir compte de la valeur du pointeur. La lecture commence toujours à partir du premier enregistrement, sauf si l'on a fait auparavant un READ, auquel cas elle commencera à partir de l'enregistrement qui suit celui qu'on vient de lire. (Ici, le premier NEXT lit l'enregistrement No 2).

Exécutez 3 ou 4 fois le programme. Essayez d'introduire de fausses dates comme 1249, PAQUES 82, 32/05/60, etc... De m&me, essayez de donner des valeurs >255 ou négatives (mais entières, sinon il y a erreur au niveau INPUT) pour la durée. Observez bien le résultat lors de l'affichage des enregistrements.

Maintenant que vous avez créé une dizaine de références sur votre fichier, venons-en à la modification, voire la destruction de certaines d'entre elles. Pour cela, nous utiliserons les instructions:

```
LET "XINDEX-<u.l.>": donne au pointeur la valeur du numéro d'ordre du
                     prochain enregistrement créé par l'ordre WRITE.
LET "DELETE-<u.l.>": détruit l'enregistrement correspondant à la
                     valeur du pointeur.
LET "UPDATE-<u.l.>": réécrit l'enregistrement correspondant à la
                     valeur du pointeur.
```

PROGRAMME DE DESTRUCTION ET DE MODIFICATION D'ENREGISTREMENTS

```
10 HOME:LET"£0-1,F,0:SINGSING"
20 LET"X-1": FI%=MA%-1
30 PRINT "NOMBRE D'ENREGISTREMENTS DANS LE FICHIER: ";FI%
40 INPUT "NO DE L'ENREGISTREMENT A DETRUIRE: ";MA%
50 LET "D-1"
60 IF PEEK(189)<> 0 THEN PRINT "ENREGISTREMENT INEXISTANT"
70 IF PEEK(189) = 0 THEN PRINT "OPERATION BIEN EFFECTUEE"
```

```
80 INPUT "NO DE L'ENREGISTREMENT A MODIFIER: ";MA%
90 LET"R-1": IF PEEK(189) <>0 THEN 60
100 INPUT "NOUVELLE PEINE:";DU%
110 LET"U-1": LET"C-1": END
```

Exécutez plusieurs fois le programme: essayez de détruire ou de modifier des enregistrements déjà détruits ou inexistants. Vous noterez que, dans ce cas, la valeur de PEEK(189) n'est pas 255 (fin de fichier en lecture séquentielle) mais 10.

L'instruction XINDEX nous sert a conna&tre le nombre d'enregistrements qui ont été créés dans le fichier: ce nombre vaut MA%-1, puisque XINDEX affecte à MA% le numéro d'ordre du prochain enregistrement créé par WRITE. Ce nombre comprend les enregistrements qui ont été détruits, car les pointeurs de ces derniers, bien que n'étant plus réutilisables, existent toujours.

Les instructions DELETE et UPDATE sont des instructions relatives. Elles détruisent (DELETE) ou modifient (UPDATE) l'enregistrement dont le No d'ordre est stocké dans MA%.
!np

ATTENTION:

- l'opération DELETE détruit un enregistrement, mais ne renumérote pas les enregistrements suivants. Le système récupère sur le disque l'espace laissé vacant. Il est donc impossible de réécrire un enregistrement qui a été détruit. Si l'on veut lire séquentiellement un fichier, le système sautera automatiquement les enregistrements détruits.

- Les pointeurs sont fixés une fois pour toutes: il n'y a JAMAIS de "renumérotation" ou de "réorganisation" des fichiers relatifs. L'ordre LET"REORGANIZE.." est réservé pour les fichiers à clés.

- L'instruction UPDATE ne fait pas de lecture: elle se contente de réécrire les valeurs des variables stockées en mémoire centrale. C'est pourquoi, si on désire ne modifier qu'une variable parmi une liste, il est préférable de faire un READ avant UPDATE pour mettre toutes les variables à la bonne valeur, puis de modifier la ou les variables souhaitées.

2.2 LES FICHIERS SEQUENTIELS INDEXES

Le principe de fonctionnement des fichiers séquentiels indexés est le m&me que celui des fichiers séquentiels relatifs. Ils peuvent &tre traités en accès séquentiel ou en accès direct. La différence réside dans le moyen d'accès: au lieu d'un pointeur fixé par le système lors de la création d'un enregistrement, nous avons une clé, qui peut comprendre plusieurs variables, et qui est fixée par l'utilisateur.

Les instructions sont assez semblables: lecture en séquentiel ou par clé, écriture, destruction et modification. Deux choses à signaler cependant:

- Les fonctions de recherche et de tri par clé sont plus développées et plus complexes qu'avec les fichiers relatifs.

- L'écriture et la lecture en séquentiel ne se passent pas "physiquement" de la m&me fa&on: nous y reviendrons.

2.2.1 CREATION

Nous avons choisi comme exemple de traiter un fichier d'automobiles dont les enregistrements comportent:

Une clé de 3 variables:
- AN%/: année du modèle
- SO$: fabricant (10 caractères)
- DA$*: date du début de fabrication.

Une variable:
- DE$: désignation du modèle.

!np

L'ordre de création se fait de la fa&on suivante, en 2 étapes.

```
LET "> AN%/,SO$10,DA$* = DE$
LET "NEW-1,F,0:AUTO"
```

Deux différences avec les fichiers relatifs:

-Pas de signe §, lequel est destiné à éviter une confusion entre un fichier relatif et un fichier séquentiel indexé dont l'unique variable de la clé serait un entier.

-La clé peut comporter plusieurs variables. Elle doit &tre de longueur fixe et ne peut contenir de tableaux. Si elle comprend une variable alphanumérique, cette dernière doit &tre suivie par un nombre indiquant sa longueur (inférieure à 256). La valeur par défaut est 1 caractère.

2.2.2 LECTURE ET ECRITURE

Venons-en à présent au remplissage du fichier. Comme pour les fichiers relatifs, nous écrirons quelques références, puis nous relirons l'ensemble du fichier. Pour cela, nous utiliserons les instructions :

- LET "WRITE-<u.l>": écriture d'un enregistrement
- LET "NEXT-<u.l>" : lecture séquentielle

PROGRAMME D'ECRITURE SUR UN FICHIER SEQUENTIEL INDEXE

```
5   REM: ECRITURE DE NOUVELLES REFERENCES
10  LET "£CL,$" :HOME: S=189: LET"£0-1,F,0:AUTO
20  INPUT "COMBIEN DE CAS VOULEZ-VOUS INTRODUIRE: ";I
30  IF I=0 THEN 100
40  FOR J=1 TO I
45  INPUT "ANNEE DU MODELE: ";AN%
50  INPUT "FABRICANT: ";SO$
53  INPUT "DATE DE FABRICATION: ";DA$
55  INPUT "REFERENCE: ";DE$
60  LET "W-1"
65  IF PEEK(S)<>0 THEN PRINT "ERREUR":GOTO 45
70  NEXT J
80  REM: RELECTURE DU FICHIER
100 LET "£CL,$":LET"£0-1,F,0:AUTO"
110 LET "N-1" : IF PEEK(S)=255 THEN 200
120 PRINT AN%;"-";SO$;"-";DA$,DE$: GOTO 110
200 LET "£CL,$" : END
```

!np

Pour écrire un enregistrement, nous utilisons l'ordre "WRITE": le programme écrit en fin de fichier un enregistrement dont la clé et les variables correspondent aux valeurs introduites par l'utilisateur.

Cependant, vous constaterez, si vous essayez d'introduire 2 enregistrements ayant la m&me clé, que le programme vous refusera le second. En effet, pour introduire deux clés homonymes, il faut utiliser l'ordre ADD et non WRITE. Les 2 s'emploient de la m&me fa&on. Si le fichier contient plusieurs clés homonymes, l'ordre de lecture READ provoquera un positionnement sur la dernière clé introduite. Pour les lire toutes, il faut faire une boucle avec NEXT.

ATTENTION: Un fichier séquentiel indexé est toujours classé par ordre de clés croissantes, en tenant compte du rang de la variable dans la clé (tri sur la première variable, puis sur la deuxième en cas d'égalité, etc...); mais l'ordre WRITE (ou ADD) classe la clé du nouvel enregistrement logiquement et non physiquement. En fait, la clé est bien écrite en fin de fichier: c'est le système qui gère un ensemble de pointeurs et qui reclasse "logiquement". Cependant, pour améliorer le temps d'accès, surtout dans le cas de gros fichiers, il est souhaitable de reclasser PHYSIQUEMENT les clés des enregistrements par ordre croissant. Il faut pour cela utiliser l'ordre LET "£REORGANIZE-<u.l>" à chaque fois que l'on introduit un nombre substantiel de nouveaux enregistrements (cf. paragraphe 3. sur la structure des fichiers sous M/DOS 6502)

Pour lire les enregistrements, on utilise comme pour les fichiers relatifs 2 instructions: l'ordre LET "READ-<u.l>, qui lit l'enregistrement dont la clé correspond aux valeurs stockées en mémoire au moment de la lecture, et l'ordre LET "NEXT-<u.l>". Ce dernier lit séquentiellement les enregistrements à partir du début du fichier, à condition qu'il n'y ait eu aucun ordre changeant la valeur des pointeurs (READ, WRITE, ADD, REORGANIZE, XTRACT...). Dans le cas contraire, la lecture séquentielle part de l'enregistrement qui vient d'&tre lu ou écrit. C'est pourquoi, si l'on veut &tre s&r de bien commencer au début du fichier, on ferme et on réouvre le fichier pour remettre à 0 les pointeurs.

Faites l'essai dans le programme en rempla&ant la ligne 100 par REM. introduisez une référence dont la clé ne soit pas la plus petite du fichier. Vous constaterez que les enregistrements ne sont imprimés qu'& partir de cette clé.
!np
2.2.3 PROCEDES DE RECHERCHE CONDITIONNELLE

Abordons maintenant un point nouveau: la recherche par clé. M/DOS 6502 dispose à cet effet de l'ordre LET "XTRACT-<u.l>,n" (o& n est le No de variable de la clé sur laquelle s'effectue le tri) qui permet d'extraire d'un fichier tous les enregistrements pour une valeur donnée d'une des variables de la clé. La recherche se déroule de la manière suivante:

- fixer une valeur pour la variable sur laquelle s'effectue le tri et une borne inférieure pour les autres.
- faire XTRACT, puis READ avec test sur PEEK(189)
- lire séquentiellement les enregistrements avec NEXT. Une fin de fichier sera détectée au premier enregistrement ne correspondant plus au critère.

La procédure n'étant pas vraiment simple, nous vous conseillons d'exécuter le programme "TRI" donné ci-dessous, avec, comme jeu d'essai, le fichier "AUTO" contenant les références suivantes:

année (AN%/)	fabricant (SO$10)	date (DA$*)	référence (DE$)
36	CITROEN	01/01/36	2CV/36
60	PEUGEOT	01/01/60	404/60
75	CITROEN	01/06/75	GS/75
75	NISSAN	01/08/74	N/75
75	PEUGEOT	01/01/75	504/75
75	PEUGEOT	01/02/75	604/75
75	PEUGEOT	01/03/75	404/75
75	PEUGEOT	01/09/75	G9/75
75	RENAULT	01/04/75	R5/75
75	VW	01/01/75	GOLF/75
78	PEUGEOT	15/07/78	104/78
79	TALBOT	01/01/79	1307/75
80	PEUGEOT	01/07/80	305/80
82	RENAULT	01/08/82	R30/82

PROGRAMME "TRI"

```
10  LET"£CL,$":HOME:S=189
15  LET "£O-1,F,AUTO"
20  AN%=75:SO$="A":DA$="01/01/75"
25  LET "£X-1,1"
30  LET "R-1"
35  IF PEEK(S)=0 THEN PRINT AN%;" ";SO$;" ";DA$,DE$
40  LET "N-1": IF PEEK(S)=255 THEN 60
45  PRINT AN%;" ";SO$;" ";DA$,DE$
50  GOTO 40
60  LET "£CL,$":END
!np
```

Sa fonction est la suivante: sélectionner et imprimer toutes les références de l'année 75. Les variables "fabricant" et "date" ont été fixées à leur valeurs minimales, respectivement "A" et "01/01/75".

Faites un essai; si vous ne vous &tes pas trompé, vous devez obtenir les 8 valeurs qui répondent au critère.

Refaites tourner le programme en rempla&ant la ligne 20 par: AN%=75:SO$="PEUGEOT":DA$="01/03/75". Vous constaterez alors que les 4 premières valeurs n'apparaissent plus; les 2 premières, parce que "CITROEN" et "NISSAN" < "PEUGEOT", les 3 et 4ème, bien que "PEUGEOT", parce que 01/01/75 et 01/02/75 < 01/03/75. Les clés n'ont donc pas la m&me priorité, et leur rang entre en ligne de compte dans le classement.

Faites encore un essai: remplacez les lignes 20 et 25 par:

```
20  AN%=78:SO$="PEUGEOT":DA$="01/07/75"
30  LET "X-1,2" (tri sur la deuxième variable)
```

Vous n'obtiendrez que 2 références ("PEUGEOT", 78 et 80). La première clé reste donc prioritaire par rapport à la troisième. Cette dernière n'est consultée qu'en cas d'égalité de AN% et de la borne inférieure. Essayez

la m&me chose avec AN%=75.

Nous avons donc vu que XTRACT permettait d'effectuer une recherche en fixant un borne inférieure. Il existe également un ordre permettant de fixer une borne supérieure, qui génère une fin de fichier lors d'une lecture séquentielle à un endroit choisi du fichier. Il s'agit de l'ordre: LET "BORNE-<u.l>" qui fixe une borne à l'enregistrement dont la clé est immédiatement supérieure ou égale aux valeurs contenues en mémoire au moment de l'exécution de LET "BORNE.."

Exemple: un fichier de 3 enregistrements dont la clé comprend 3 variables.

V1	V2	V3
15	A	1
15	A	4
15	B	9

valeurs de la clé prises en compte par BORNE			enregistrements lus par NEXT
15	A	4	le 1er
15	A	3	le 1er
15	A	10	le 1er et le 2ème
14	Z	10	aucun
15	C	1	tous

!np

Comme pour XTRACT, il est donc souhaitable de fixer les valeurs de toutes les variables de la clé immédiatement avant BORNE pour éviter les surprises. Vous noterez que, là aussi, le tri se fait en fonction du rang de la variable dans la clé et que, d'autre part, si les valeurs de la borne correspondent à un enregistrement réel, ce dernier ne sera pas lu par NEXT.

Exemple pratique: reprenez le programme "TRI" dans sa version initiale. Rajoutez la ligne:

```
33  AN%=75: SO$="PEUGEOT":DA$="1/7/75":LET "B-1"
```

Vous obtiendrez toutes les voitures construites en 75 sauf les 3 dernières (RENAULT et VW > PEUGEOT, PEUGEOT 1/9/75 > PEUGEOT 1/7/75)

Faites des essais en changeant la valeurs des variables dans la ligne 33 (AN%=74 et 76, DA$="1/3/75", etc...)

Pour annuler l'effet de XTRACT, il suffit de faire: LET"XTRACT,0"

CONSEIL PRATIQUE: si, dans ce programme, nous faisons une extraction sur la première variable, c'est pour mieux montrer le fonctionnement de XTRACT. Dans la pratique, on n'utilisera pas XTRACT pour faire un tri sur la première variable, mais la séquense d'instructions READ (positionnement des pointeurs) - NEXT (lecture séquentielle) - BORNE (pour indiquer l'endroit o& la lecture doit s'arr&ter) qui permet de gagner du temps, surtout en cas de gros fichiers.

2.2.4 DESTRUCTION ET MODIFICATION

Enfin, pour détruire ou modifier un enregistrement, nous utiliserons les ordres LET"DELETE-<u.l>" et LET"UPDATE-<u.l>". Leur utilisation est simple: il faut mettre en mémoire la valeur de la clé, puis exécuter l'ordre approprié. Si la recherche aboutit, (la clé existe, PEEK(S)=0), l'enregistrement est détruit (DELETE) ou réécrit avec les nouvelles valeurs de la liste des variables (UPDATE).

ATTENTION: comme pour les fichiers relatifs, l'ordre UPDATE n'entra&ne pas de READ. Si on ne veut modifier que quelques valeurs de la liste, il est souhaitable d'en faire un avant. D'autre part, l'ordre UPDATE ne permet de modifier que les variables de l'enregistrement et non celles de la clé. Pour ces dernières, il faut faire:

- donner les valeurs de la clé à modifier.
- READ (mise en mémoire de toutes les valeurs (clé et variables))
- DELETE
- donner les nouvelles valeurs de la clé
- WRITE

!np

2.3 LES FICHIERS MULTICLES

2.3.1 PRESENTATION

Les fichiers multiclés sont des fichiers séquentiels indexés dont les enregistrements disposent de plusieurs clés, appelées dans ce cas "moyens d'accès". Le nombre maximum de clés possibles est fixé à 10.
L'utilisation des fichiers multiclés étant rigoureusement identique à quelques détails près à celle des fichiers à une clé, nous conseillons de bien prendre connaissance du paragraphe précédent consacré aux fichiers séquentiels indexés avant de lire celui-ci. Il est à noter que les fichiers à une clé ne forment pas une catégorie à part comme les fichiers relatifs, mais ne sont qu'un cas particulier de fichiers multiclés.

Le principe général est le suivant: chaque enregistrement du fichier possède plusieurs moyens d'accès, mais, quel que soit le traitement (sauf la création d'un enregistrement avec WRITE ou ADD), on ne peut utiliser qu'un seul moyen d'accès à la fois, dont on précise le numéro après l'instruction. Ainsi, LET "NEXT-1,3" signifie qu'on va lire séquentiellement selon le troisième moyen d'accès, c'est à dire dans l'ordre croissant de la troisième clé. La valeur par d&faut du moyen d'accès est 1 (c'est ce que nous faisions au paragraphe précédent). LET "NEXT-1" provoquera une lecture séquentielle selon le premier moyen d'accès.

Les clés respectent absolument la m&me syntaxe que précédemment. Dans la création du fichier, elles sont séparées par un "&".

LET ">V11,V12..V1n & V21,..&..Vxy = LV1,LV2,..."

Chaque clé possède plusieurs variables, mais attention: les fichiers multiclés occupent beaucoup de place. Il convient, dans la mesure du possible de s'en tenir à un nombre raisonnable de clés et de variables par clé.

2.3.2 UTILISATION

Pour créer un nouvel enregistrement par WRITE ou ADD (ce dernier acceptant les clés homonymes), il faut stocker en mémoire toutes les valeurs des variables de TOUTES les clés. Il n'est pas possible de créer un enregistrement selon un seul moyen d'accès. Exemple: création et remplissage d'un fichier "dossier de presse" à 2 clés:

```
  1ère clé                  2ème clé
                variable
RU$10 -> rubrique       DA$* -> date du journal      JO$ -> nom du journal
AU$10 -> auteur
```

```
création du fichier "PRESSE":  LET ">RU$10,AU$10 & DA$* = JO$"
                               LET "£NEW-1,F,0:PRESSE"
```

```
!np
10  REM: PROGRAMME D'ECRITURE
20  LET "£CL,$":HOME:S=189
30  LET "£0-1,F,0:PRESSE"
40  INPUT "COMBIEN DE CAS VOULEZ-VOUS INTRODUIRE: ",I
50  IF I=0 THEN 100
60  FOR J=1 TO I
70  INPUT "RUBRIQUE: ";RU$
75  INPUT "AUTEUR  : ";AU$
78  INPUT "DATE    : ";DA$
80  INPUT "NOM     : ";JO$
85  LET "A-1" : IF PEEK(S)<>0 THEN 70
90  NEXT J
100 LET "£CL,$":END
```

Nous vous suggérons de prendre comme jeu d'essai le fichier suivant:

```
RUB.       ! AUTEUR     ! DATE     ! NOM DU JOURNAL
-------------------------------------------------
SANTE      ! A. PIKOUZ  ! 01/04/76 ! LE MONDE
POLITIQUE  ! J. LEMAIRE ! 15/09/77 ! LE FIGARO
POLITIQUE  ! J. LEMAIRE ! 28/10/77 ! LE FIGARO
SOCIETE    ! A. NONIME  ! 01/04/76 ! LE MONDE
POLITIQUE  ! G. LAVESTE ! 23/07/82 ! FRANCE-DIMANCHE
SANTE      ! J. LEMAIRE ! 31/05/78 ! LE FIGARO
```

Pour toutes les autres fonctions, un seul point à retenir: lorsqu'on a choisi un moyen d'accès, il est impossible d'avoir accès aux variables des autres clés. On sera donc souvent obligé de reprendre dans la liste des variables les composantes des différents moyens d'accès. Dans notre exemple, si on lit le fichier par rubrique, on ne pourra jamais savoir la date. Si l'on fait tourner le programme suivant:

```
 !lm10

 5  REM : PROGRAMME DE LECTURE SEQUNTIELLE
10  LET "£CL,$":HOME:S=189
15  LET "£0-1,F,0:PRESSE"
20  INPUT "MOYEN D'ACCES (1/2): ";I$
30  LET "N-1,"+I$:IF PEEK(S)=255 THEN 100
40  PRINT RU$;" ";AU$;" -- ";DA$;" -- ";JO$
50  GOTO 30
100 LET "£CL,$":END
!lm5
```

on obtiendra selon l'option choisie:

premier moyen d'accès

```
POLITIQUE G. LAVESTE --  -- FRANCE-DIMANCHE
POLITIQUE J. LEMAIRE --  -- LE FIGARO
POLITIQUE J. LEMAIRE --  -- LE FIGARO
SANTE     A. PIKOUZ  --  -- LE MONDE
SANTE     J. LEMAIRE --  -- LE FIGARO
SOCIETE   A. NONIME  --  -- LE MONDE
!np
```

deuxième moyen d'accès

```
   -- 01/04/76 -- LE MONDE
   -- 01/04/76 -- LE MONDE
   -- 15/09/77 -- LE FIGARO
   -- 28/10/77 -- LE FIGARO
   -- 31/05/78 -- LE FIGARO
   -- 23/07/82 -- FRANCE-DIMANCHE
```

On notera que l'ordre NEXT est paramétré. C'est une option qui sera souvent utilisée dans le cas de fichiers multiclés.

Donnons à présent un exemple de recherche par clé. La démarche est tout à fait similaire à celle que nous pouvions avoir avec les fichiers à une seule clé.

PROGRAMME DE TRI

```
10  LET "£CL,$":HOME:S=189
20  LET "£0-1,F,0:PRESSE"
30  INPUT "QUEL MOYEN D'ACCES (1/2): ";I$
40  IF I$="1" THEN RU$="POLITIQUE":AU$="A"
50  IF I$="2" THEN DA$="01/01/77"
60  LET "X-1,1,"+I$
70  LET "R-1,"+I$: IF PEEK(S)=10 THEN 90
80  PRINT RU$;" ";AU$;" - ";DA$;" ** ";JO$
90  LET "N-1,"+I$: IF PEEK(S) <>255 THEN 80
100 LET "£CL,$":END
```

Remarque: dans ce programme,quel que soit le numéro du moyen d'accès, le tri se fera sur la première variable. La syntaxe de XTRACT est en effet:

LET "XTRACT-<u.l>,<No de variable>,<No de clé>"

Si l'on introduit un faux numéro de variable ou de moyen d'accès, il y a impression d'un message erreur et arr&t du programme. On peut ommettre le No de clé (valeur par défaut=1) mais pas celui de la variable (risque de confusion avec XINDEX).

L'instruction BORNE a le m&me effet qu'avec les fichiers à une clé, à cette différence près que, dans le cas de fichiers multiclés, la limite inférieure est lue par NEXT. En voici un exemple: rajoutez dans le programme de tri les 2 lignes suivantes:

```
32  IF I$="1" THEN RU$="POLITIQUE": AU$="J. LEMAIRE":LET "B-1,1"
```

```
33  IF I$="2" THEN DA$="31/05/78":LET "B-1,2"
!np
```

Vous constaterez que dans les 2 cas, la limite de la borne, qui correspond à un enregistrement réel, appara&tra à l'écran.

Ce programme est bien s&r sommaire. Dans la réalité, il faudrait conditionner le PRINT par le numéro du moyen d'accès, paramétrer le numéro de la variable du XTRACT, etc... Le tout étant de ne pas s'embrouiller avec les différents moyens d'accès!

```
!cj
--------------
---------
-----
-
```

!1j
3. STRUCTURE DES FICHIERS

Un fichier M/DOS est découpé en un certain nombre d'unités de base que nous appelerons "PISTES LOGIQUES". Ces dernières peuvent ou non correspondre à des pistes physiques.

3.1. CREATION DU FICHIER

Lors de la création du fichier, le système réserve 2 pistes logiques: la première pour les clés ou les pointeurs, la seconde pour les enregistrements. Au fur et à mesure que la taille du fichier s'accro&t, de nouvelles pistes logiques sont allouées dynamiquement au fichier.

Chaque fichier M/DOS a une structure qui lui est propre: c'est la raison pour laquelle lui est attaché un descriptif appelé DCB (Data Control Bloc) qui vous est donnée en annexe. Les principaux éléments en sont:

- une description de l'enregistrement:
  Ces paramètres définissent les différentes clés, leur type, etc... 7 octets sont réservés pour la description de chaque variable de l'enregistrement.

- 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 et classés dans un ordre fictivement croissant pour permettre les recherches par clés.

- Les différents pointeurs permettant dans ce fichier de conna&tre les blocs libérés à la suite de destruction ou non encore utilisés.

3.2. LES MOYENS D'ACCES: POINTEURS ET CLES

On distingue les 2 types de fichiers: fichiers relatifs et fichiers séquentiels indexés.

3.2.1. FICHIERS RELATIFS

La structure est la suivante:

Les pistes des pointeurs contiennent un cha&nage sur l'enregistrement. Ce cha&nage est une suite de 3 octets, ce qui permet de prévoir des extensions importantes. On peut mettre 85 pointeurs dans chaque bloc de 256 octets qui leur est réservé. Chaque pointeur renvoie sur l'enregistrement par 2 paramètres:
!np
- le numéro de la piste logique o& se trouve le début de l'enregistrement.

- le numéro du bloc dans cette piste logique.

Lors de la lecture d'un enregistrement, le système calcule d'abord la piste logique o& se trouve le pointeur et lit le bloc correspondant. Le pointeur permet alors de trouver l'enregistrement.

Cette méthode implique de lire au moins 2 blocs pour accéder à un enregistrement, mais elle n'impose en revanche aucune contrainte sur la longueur de l'enregistrement. C'est ce qui permet au M/DOS de gérer des fichiers à taille variable.

3.2.2. LES FICHIERS SEQUENTIELS INDEXES

Les pointeurs vers les enregistrements sont alors complétés par des clés, se divisant éventuellement elles-m&mes en sous-clés.

Le coefficient de blocage des clés d'un fichier séquentiel indexé est calculé ainsi (voir le paragraphe C pour une explication détaillée des concepts de "coefficient de blocage" et de "bloc de base"):

Soit L le nombre d'octets de la clé: on enregistrera pour chaque article L octets pour la clé, 3 octets pour pointer sur l'enregistrement et 2 octets pour pointer vers la clé suivante, soit un total de L+5 octets. On ne pourra donc mettre que Partie entière de (256/(L+5)) clés dans un bloc de 256 octets. Dans le cas de fichiers multiclés, chaque clé demandera un ensemble de pistes logiques.

3.2.3. LES PRINCIPES DE RECHERCHE PAR CLE

Les clés d'un fichier séquentiel indexé se répartissent en 2 catégories:

- les clés triées
- les clés non triées, appelées "TAS"

Les clés triées sont rangées par ordre croissant sur les pistes logiques, tandis que les clés non triées sont ajoutées en fin de liste. A défaut d'&tre classées, elles sont cha&nées aux autres clés. Bien entendu, les temps d'accès et de recherche sont beaucoup plus performants avec les clés triées qu'avec les clés du tas. Une recherche sur les premières se fait par DICHOTOMIE, dont nous rappelons le principe:

Il s'agit, lorsqu'on recherche une clé dans une liste ORDONNEE, de comparer la clé recherchée avec la clé "médiane" (située au milieu de la liste). Quel que soit le résultat de la comparaison, il permettra toujours d'éliminer la moitié de la liste à chaque lecture.
!np
Prenons comme exemple le cas suivant:

liste: A1,A4,B5,C0,F6,G1,G2,K9,L5,O5,U2

- 1er cas: clé recherchée: K9

CLE MEDIANE	COMPARAISON	NOUVELLE ZONE DE REHERCHE
G1	<	G2 - U2
L5	>	G2 - K9
K9	=	recherche terminée

- 2ème cas: clé rechercheé: B9

CLE MEDIANE	COMPARAISON	NOUVELLE ZONE DE RECHERCHE

```
-------------------------------------------------------
     G1              >        A1 - F6
     B5              <        C0 - F6
     F6              >        C0
     C0              /        la clé n'existe pas
```

L'intér&t de la méthode est l'élimination de la moitié de la liste à chaque essai (qui demande au plus une lecture disque). Ceci signifie que, s'il faut 10 lectures pour rechercher une clé dans un fichier de 10000 enregistrements, il n'en faudra que 11 si le fichier comporte 20000 enregistrements. Sur un disque dur, par exemple, à raison d'un temps d'accès moyen de 50 ms et d'un coefficient de blocage de 25 (par exemple 25 clés de 5 octets, ce qui donne, compte tenu des 5 octest supplémentaires, 250 octets, soit presque un bloc), la recherche d'une clé parmi 10000 prendra environ 1/2 seconde.

Ce principe de recherche dichotomique n'est plus valable dans le cas des clés non triées.

Lors d'une création d'articles, le système n'insère pas la nouvelle clé dans la liste triée: cette opération impliquerait en effet un décalage de toutes les clés supérieures à la nouvelle. Dans l'exemple précédent, l'adjonction d'une clé en début de fichier provoquerait le décalage de 10000 clés, ce qui prendrait 400 (blocs) x 50 ms (temps d'accès) = 20 secondes! Ce temps n'étant pas admissible en utilisation normale, on préfère placer la nouvelle clé dans le "tas".

La nouvelle clé du tas est cha&née à la clé immédiatement inférieure: on se sert pour cela des 2 octets (cf plus haut) réservés pour pointer sur la clé suivante. Bien entendu, le processus de recherche est modifié. Pour accéder à une clé du tas, il faut dans un premier temps déterminer les deux clés consécutives de la liste triée entre lesquelles se trouve la clé recherchée: on "remonte" ensuite le réseau de cha&nages qui conduit à la clé désirée (dans le cas o& celle-ci existe, bien s&r). Dans le cas d'un fichier dont la moitié des clés ne serait pas triée, en supposant que la répartition soit homogène (une clé du tas entre deux clés triées), la dégradation moyenne du temps d'accès serait d'un accès supplémentaire par clé du tas. Autre exemple: à supposer que l'on fasse une recherche sur un fichier que l'on vient de remplir et qui était initialement vide, le temps d'accès serait alors directement
!np
proportionnel à la taille du fichier et serait à peine deux fois plus court qu'une recherche séquentielle.

Pour reclasser toutes les clés, le système comporte une instruction que nous avons déjà vue au paragraphe II:

```
LET "£REORGANIZE-<u.1>"
```

Le temps nécessaire pour l'exécution de cet ordre est proportionnel à la taille du fichier. Dans le cas d'un fichier de 10000 articles stocké sur disque dur (temps d'accès = 50 ms), dont les clés tiendraient en 400 blocs, il faudrait 400 x 2 (lecture + écriture) x 50 = 40000 ms, soit 40 secondes pour trier le fichier.

Dans le cas de fichiers multiclés, l'opération est répétée sur chacune des clés. Bien entendu, les fichiers relatifs ne peuvent &tre réorganisés, les pointeurs étant fixés une fois pour toutes, m&me si des enregistrements ont été détruits.

Pour que la réorganisation soit possible, il est nécessaire que le nombre de pistes libres sur le support choisi soit suffisant pour recopier la liste des clés. Dans le cas contraire, il faudrait "faire de la place" en détruisant quelques fichiers. Ce système a en outre un avantage: comme les pistes des clés changent à chaque réorganisation, l'usure de la surface est répartie sur toute la surface du disque (dans le cas de disques souples, bien s&r!).

3.3. LA STRUCTURE DES ENREGISTREMENTS

Quel que soit le type du fichier considéré, la structure des enregistrements est toujours la m&me. Deux aspects importants sont à retenir: la détermination de la taille des blocs de base et le codage interne des informations.

3.3.1. LES BLOCS DE BASE

Il s'agit de la quantité d'informations "utiles" saisie en une seule opération de lecture ou d'écriture sur disque. Bien entendu, l'idéal serait que la taille du bloc de base correspond&t toujours avec celle des enregistrements. C'est malheureusement impossible, puisque les enregistrements sont de longueur variable. De plus les blocs de base ne peuvent contenir que 32, 64, 128 ou 256 octets. La taille (exprimée en octets) du bloc de base est appelée COEFFICIENT DE BLOCAGE. Quel que soit le nombre d'octets, 3 sont toujours réservés pour les cha&nages internes. La taille optimum est déterminée par le sytème lors de la création du fichier par l'algorithme suivant:

En ne tenant compte que des variables de l'enregistrement contenues dans le dictionnaire:

chaque flottant vaut 5 octets
chaque binaire vaut 1 octet
chaque entier vaut 2 octets
chaque date vaut 2 octets
!np
Voilà pour ce qui est certain. Il reste le cas des cha&nes de caractères qui sont approximées à 12 octets et celui des tableaux dont on suppose qu'ils contiennent en moyenne 3 éléments non nuls. Ajoutons à ceci les 3 octets de cha&nage et nous aurons la taille "adéquate" du bloc de base en prenant la valeur possible (32,64,128 ou 256) immédiatement supérieure à la valeur trouvée (si cette dernière excède 256, le système prendra pourtant 256 comme taille du bloc de base, car cette valeur représente un maximum). L'utilisateur a cependant la possibilité d'imposer une valeur inférieure à la valeur théorique (s'il sait, par exemple, que toutes ses cha&nes de caractères ne contiendront qu'un seul caractère alors que le système prend 12 caractères comme valeur moyenne pour son calcul), par l'instruction:

LET "£>n", o& n peut prendre les valeurs 32, 64 ou 128 . En aucun cas l'utilisateur ne peut choisir une valeur supérieure à celle qui a été calculée par le système.

La taille du bloc de base est une constante du fichier qui est déterminée lors de la création. Elle ne peut plus &tre modifiée par la suite.

Le principe de fonctionnement des blocs de base est aisé à comprendre: si un enregistrement est plus petit que le bloc de base, il ne demandera qu'un seul bloc et le cha&nage interne indiquera une fin d'article. Dans

le cas contraire, il faudra plusieurs blocs pour le contenir et ceux-ci seront cha&nés à l'aide des pointeurs internes. Seul le dernier bloc ne contient pas de cha&nage significatif, comme dans le cas d'un bloc unique.

L'intér&t de l'optimisation de la taille du bloc de base est évident:

- plus le bloc de base est petit, et plus la place sur disque sera optimisée (moins de place perdue).

- en revanche, plus le bloc de base est grand, et plus le temps d'accès sera petit (les différents blocs de base ne se trouvant pas forcément sur le m&me bloc physique, il faudra plusieurs lectures disque pour les enregistrements dont la taille est supérieure à celle des blocs de base. De plus, en cas d'écriture, il faut lire les blocs avant de les réécrire).

3.3.2. LE CODAGE DES INFORMATIONS

Pour lire ou écrire les informations, le système ne se réfère qu'au dictionnaire. Les variables simples sont donc toutes écrites directement sans séparateurs. Chaque variable a une longueur fixe déterminée par son type (un flottant = 5 octets, un binaire = 1 octet, un entier = 2 octets et une date = 2 octets) sauf les cha&nes de caractères, pour lesquelles un octet supplémentaire est réservé en début de zone afin d'en indiquer la longueur, sachant que les blancs des extrémités ont été supprimés pour gagner de la place.
!np
Dans le cas des tableaux, les variables sont lues dans l'ordre comme une suite de variables simples, sauf dans le cas d'une suite de n variables nulles ou vides. Le codage prend alors la forme suivante:

si n est < 128, l'ensemble des variables est codé sur 2 octets de la forme: 0,n

si n est > 128, l'ensemble des variables est codé sur 3 octets, de la forme 0,poids fort de n+128, poids faible de n (on prend le poids fort de N+128 et non de N, car il y aurait confusion avec le cas o& N<128: rappelez-vous qu'il n'existe pas de séparateurs pour isoler les variables).

La fin de tableau est repérée par une configuration normalement impossible, à savoir deux 0 consécutifs.

Il nous reste à voir le cas un peu particulier des dates: celles-ci sont codées sur deux octets dont la disposition est la suivante:

```
AAAAAAAM  MMMJJJJJ
12345678  12345678
```

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)

Les dates inscrites ont été contr&lées par le système avant l'écriture. En cas de non validation, les deux octets sont vides. Dans le cas des clés, les dates sont bien s&r classées par ordre croissant en tenant compte du fait que les années 50-99 (1950-1999) viennent avant les années 00-49 (2000-2049). A cet effet, le système ajoute au moment de l'écriture 50 aux années 00-49 et retranche 50 aux années 50-99.

** exemple concret de codage d'un enregistrement **

L'enregistrement est défini par le dictionnaire suivant:

```
A (clé)
=
A$,B,C%,A;,B$;
```

Ecrivons dans le fichier un nouvel enregistrement avec:

```
B=1
A$="TEST"
C%=10
A(0)=1,A(6)=2, les autres sont nuls
B$(0)="TUTU" , les autre éléments sont vides
```

!np
Ce qui donne, en codage hexadécimal:

```
A$ -> 04 54 45 53 54   (04 est la longueur)

B  -> 81 00 00 00 00

C% -> 00 0A             (entier sur 2 octets)

A; -> 82 00 00 00 00  00 05  82 00 00 00 00  00 00
      <--    1    -->  <5x0>  <--   2    -->  <fin>

B$ -> 04 54 55 54 55  00 00
```

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 04 54 55 54 55 00
: 00
```

Si les tableaux étaient remplis, cet enregistrement pourrait très bien comporter plusieurs milliers d'octets.

!lm5
!lj
1. NOTION DE MASQUE

Le masque est avant tout une définition d'écran: il permet d'introduire (entrée) ou de visualiser (sortie) des données selon un format très strict et à des endroits sur l'écran définis au préalable lors de la création du masque. Cette présentation d'écran pourra &tre sauvegardée sur disque et &tre chargée en mémoire centrale lorsqu'on voudra l'utiliser.

Un exemple qui permettra de mieux saisir le concept de masque est celui du formulaire administratif. pour &tre accepté, ce dernier doit &tre rempli dans des zones prédéfinies selon une syntaxe rigoureuse: nous allons ainsi définir les 4 éléments du masque à partir de l'exemple suivant.
!cj

SITUATION MATRIMONIALE (*):..

!lj

(*) 1: célibataire
2: marié
3: veuf
4: divorcé

On distingue:
!lm10

le TEXTE du masque, qui permet de renseigner l'utilisateur sur ce qu'il doit introduire (ici, situation matrimoniale).Ce texte n'a aucune fonction active et peut &tre réduit à un blanc.

la VARIABLE, qui est la réponse, et qui peut prendre plusieurs valeurs (ici,1,2,3 ou 4). Qu'elle soit entrée par l'utilisateur ou affichée en sortie, elle est extérieure au masque lui-m&me.

la FENETRE qui est la zone réservée (ici, une case) pour la réponse. A chaque fen&tre correspond une variable (ou un tableau de variables). Cette correspondance est établie une fois pour toutes.

le FORMAT, qui est la syntaxe de la réponse: ici, la réponse ne peut &tre que 1,2,3 ou 4. Dans le cas d'un masque M/DOS, le format porte sur la nature des variables (entière, positive, alphanumérique...), et non pas sur les valeurs qu'elles peuvent prendre. De plus, le masque exerce un contr&le qui emp&che la validation d'une variable dont le format serait erronné.
!lm5

Nous voyons donc que le masque, tel qu'il est défini, n'est ni un simple fichier de données, puisqu'il exerce un controle actif (taille et nature des données), ni un programme. Il a sa propre nature, caractérisée dans les ordres M/DOS par la lettre M. Il se manipule comme un fichier normal (ouverture, fermeture), mais les commandes d'utilisation ne sont plus lecture et écriture, mais SAISIE, VISUALISATION, etc...
!np
Les masques peuvent &tre manipulés aussi bien en mode direct qu'en mode programme: dans ce dernier cas, l'ordre M/DOS deviendra une simple

instruction BASIC.

Il est important de bien distinguer la phase de création du masque, o& l'utilisateur définit le TEXTE, les FENETRES et les FORMATS, et la phase d'utilisation o& le masque est appelé (généralement au cours d'un programme) pour la saisie ou la sortie de variables.

!cj

-
!1j
!np

2. CREATION DE MASQUE

Comme les fichiers, les masques doivent &tre créés avant d'&tre utilisés. La différence avec les fichiers tient cependant dans le fait que la définition du masque ne porte pas sur sa nature (puisqu'il n'y à qu'un type de masque "simple" - nous verrons plus loin le cas des masques globaux), mais directement sur son contenu (TEXTES, FENETRES, FORMATS). l'instruction de création du masque est:

LET "£NEW,<u.l>,MASQUE,<d>:<nom du masque>

Ainsi, l'instruction LET"£N,1,M,0:TRYMASK définit sur le drive 0 un masque de nom "TRYMASK" et lui affecte l'unité logique 1 pour la création.

A partir de ce moment, l'ordinateur laisse la main à l'utilisateur (le curseur est alors positionné tout en haut à gauche de l'écran): sur un espace de 24 lignes de 40 colonnes chacune, l'utilisateur va pouvoir entrer tout ce qu'il veut. Quand il a terminé, il valide le masque en tapant sur la touche ESC. Si aucune erreur de syntaxe dans la définition des fen&tres et des formats n'est détectée, le masque est globalement validé et l'opération est terminée (ne pas oublier toutefois de fermer l'unité logique ouverte lors de l'instruction LET "£NEW..). Dans le cas contraire, le curseur revient en haut à gauche et une sonnerie indique que le masque n'est pas correct. Il faut alors corriger l'erreur et retaper ESC.

2.1. Déplacement sur l'écran et validation

Pour se déplacer rapidement sur l'écran, l'utilisateur dispose de plusieurs touches: <-- et --> pour les déplacements latéraux, RETURN ou CTRL/Z pour un déplacement vers le bas , CTRL/R ou CTRL/Q pour un déplacement vers le haut et CTRL/T pour un retour du curseur en haut à gauche. La touche CRTL/§ permet d'effacer tout ce qu'il y a d'écrit sur l'écran.

ATTENTION: la touche RETURN n'a ici qu'un effet de déplacement et non de validation de ligne comme en mode BASIC.

Un point important est le fait que la validation finale par ESC se fait sur l'ensemble de l'écran: tout ce qui est INSCRIT sur l'écran au moment o& l'on tape ESC est validé: si on désire par exemple corriger une lettre au milieu d'une ligne, ligne elle-m&me placée au milieu de l'écran, il suffit de déplacer le curseur jusqu'a cette lettre, de taper la nouvelle

lettre et de faire ESC pour que tout soit validé. Si l'on désire détruire un caractère en décalant le reste du texte pour ne pas laisser un blanc, on utilisera la touche CTRL/D (D pour Delete), le curseur étant positionné sur le caractère suivant celui à détruire. De m&me, pour inserer un caractère, taper CTRL/I à l'endroit voulu, puis taper le caractère désiré. Si vous souhaitez abandonner en cours de création de masque, taper CTRL/A.
!np
Pour vous familiariser avec ceci, essayez de créer un masque "fixe" (ne contenant que du texte, lequel n'obéit à aucune syntaxe), avec, par exemple, quelques lignes de texte et une figure géométrique.

ATTENTION: ne pas utiliser les caractères <,> et " qui sont réservés.

2.2. Syntaxe des différents éléments du masque

2.2.1. LE TEXTE

Le texte n'est soumis à aucun contr&le: il peut se trouver n'importe o& sur l'écran et comprendre tous les caractères sauf les caractères reservés (CTRL/R,§ sur fond blanc,<,>,",etc..)
Aucune différence n'est faite entre un texte purement décoratif (ligne d'étoiles) et un texte explicatif se rapportant à une variable (référence du produit: ...)

3 fonctions sont disponibles pour modifier la présentation des caractères du texte:

CTRL/B: imprime des caractères noirs sur fond blanc.
CTRL/F: imprime des caractères clignotants. Cumulable avec CTRL/B
CTRL/N: retour à la normale.

Une fonction permet à l'utilisateur la reproduction dans le sens vertical d'un caractère: il faut positionner le curseur SUR le caractère à reproduire et taper CTRL/V autant de fois que l'on désire. CTRL/V sert surtout pour l'impression de colonnes.

Le caractère § sur fond blanc a pour effet de rendre transparente la zone du masque qu'il occupe: en mode utilisation (voir chapitre suivant), lorsqu'un masque est chargé, il occupe toute la place de l'écran et efface ce qui s'y trouvait affiché, sauf précisement aux endroits o& l'on a tapé § sur fond blanc dans le masque.

2.2.2. LES FENETRES

Elles permettent de définir une zone physique pour l'entrée et la sortie de variables. En mode de création, la fen&tre est définie par les signes "<" et ">", indiquant respectivement le début et la fin de la fen&tre.

Exemple:

col.19

(ligne 10) donnez votre nom: <   >

Il y a 6 blancs entre les 2 signes, ce qui signifie que le nom aura 8 caractères de longueur maximum, car les extremités sont comprises. Lors

!np
de la saisie de la variable (voir chapitre suivant), le curseur va se positionner automatiquement en ligne 10, colonne 19 pour recevoir la première lettre du nom. Si ce dernier fait plus de 8 caractères, le curseur restera bloqué sur la 8ème lettre et n'ira pas plus loin.

Cas particulier: si la variable ne fait qu'un caractère, seul le signe ">" sera retenu: exemple;

REPONDEZ 1 POUR OUI, 0 POUR NON: >

N.B: La syntaxe de ces 2 derniers exemples n'est cependant pas complète, puisqu'il manque le format et la variable, dont la syntaxe fait l'objet du 3ème paragraphe. En effet, à chaque fen&tre, doit correspondre impérativement un et un seul FORMAT.

2.2.3. LES FORMATS

Le format est un contr&le qu'exerce la machine sur la nature et le positionnement sur l'écran des variables à saisir ou à imprimer, en m&me temps qu'il établit une correspondance directe entre les noms des variables dans le masque et les noms des variables dans le programme. Toutes les fois o& une variable ne correspondra pas à son format (alphanumériques au lieu d'entiers, date impossible, etc...), elle sera refusée (retour du curseur au début de la fen&tre). Le format s'écrit toujours entre guillemets: les oublier ou n'en mettre qu'un constitue une erreur de syntaxe qui entra&nera la non-validation du masque. Ce format peut se placer n'importe o& sur l'écran; il y aura correspondance automatique entre la Nème fen&tre et le Nème format.

Tous les cas suivants sont possibles et aboutissent au m&me résultat.

NOM: <"A$" > AGE: <"N%">

NOM: < > "A$" AGE: <"N%">

NOM: < > AGE: < > "A$" "N%"

Le fait que le format soit ou non à l'intérieur des fen&tres ne modifie en rien ces règles.

Dans les formats de masques, les types de variables suivants sont acceptés.

XX	flottant (nombre réel)
XX%	entier (nombre entier dont la valeur absolue est <32767)
XX%/	binaire (en fait nombre entier compris entre 0 et 255 inclus, appelé ainsi car sa représentation binaire tient dans un seul octet)
XX%+	entier positif
XX+	flottant positif
XX$	flottant entier (nombre entier pouvant aller jusqu'a 1. E38 -1)
XX$	chaine de caractères (jusqu'a 255 caractères)
XX$*	date (format jj/mm/aa)
XX$+	numerique stocké dans une chaine de caractères avec affichage gestion (cf. fonctions de calcul à 48 chiffres)

!np
Précisions au sujet de la date: lorsqu'une variable de type date est saisie, le traitement se fait de la fa&on suivante. La machine ne prend en compte que les chiffres et regarde s'ils forment tels quels une date cohérente. Si oui, la date est réaffichée avec le format JJ/MM/AA. Sinon,

la machine essaie d'insérer des 0 pour former une date possible. En cas d'échec, la date est refusée en saisie. Voici quelques exemples destinés à clarifier les esprits. Toutefois, ceux-ci ne sont vérifiables qu'en mode utilisation et non création (voir chapitre suivant)

saisie	affichage
1249	12/04/09
1/2/49	01/02/49
12/49	12/04/09 (en fait, 2009)
32/5/78	refusée (32 jours en mai)
29/02/61	refusée (61 n'est pas bissextile)
31560789	31/05/60
56789	05/06/78

L'année n'ayant que 2 chiffres, les nombres de 50 à 99 correspondent à 1950-1999, les nombres 00 à 49 aux années 2000-2049.

Le masque accepte aussi des précisions de formattage. Il ne s'agit plus d'un contr&le sur une variable saisie, mais d'un reformattage à l'écran au moment de la saisie ou en sortie. Les cas suivants sont possibles pour éviter le cadrage à gauche par défaut:

XX§*: flottant entier cadré à droite (<121..> -> <..121>)
XX* : flottant format gestion (<7.5 > -> < 7.50>)
XX%: entier cadré à droite

Note sur le format gestion: lorsque le formattage gestion est impossible (variable trop élevée ou fen&tre trop petite), le cadrage se fait à gauche et une "*" appara&t à l'extr&me droite, ce qui permet de repérer facilement les anomalies.

Par défaut, le nombre de décimales est 2. On peut le changer par l'ordre LET "§-n", o& n est le nombre de décimales (1 à 9). En valeur par défaut (2), toutes les décimales sont imprimées, m&me s'il y en a plus de 2.

ATTENTION: il y a correspondance entre les noms des variables du masque et de celles du programme BASIC o& le masque est utilisé. Cependant, le BASIC n'a pas autant de types de variables que les masques. Ainsi, D$* ou A%/ n'existent pas. Voici quelles sont les correspondances:
!np

Variables Masque:	traitées en BASIC sous le nom:
X%	
X%+	X% (entier)
X%/	
X%*	
YY	
YY+	YY (flottant)
YY§	
YY*	
Z1$	
Z1$*	Z1$ (chaine de caractères)
Z1$+	

Enfin, il est possible d'ajouter 3 options au format:

: exemple: <"Z%:"> Enchainement automatique. Lors de la saisie des données, lorsque le curseur arrive à la dernière case de la fen&tre, il passe directement à la fen&tre suivante sans qu'il soit besoin de faire RETURN. S'il est dans la DERNIERE fen&tre (que celle-ci soit ou non réservée pour les variables en sortie), il provoque une sortie de masque.

; exemple: <"Z$;"> Utilisé pour les tableaux en entrée aussi bien qu'en sortie. 2 syntaxes possibles:
!1m10

1/<"A;5"> se rapporte à A(5) dans le programme BASIC qui utilisera le masque.

2/<"A;99"> se rapporte à l'ensemble du tableau A. Cela signifie qu'il faudra introduire (ou afficher si l'on est en sortie) non pas UNE, mais toutes les valeurs du tableau, séparées par des ";". L'indice maximum théorique est 9, mais on peut modifier cette valeur en changeant le contenu le l'adresse 770 par l'instruction POKE 770,<nouvelle valeur>. Cette dernière valeur ne peut toutefois excéder 98. Les variables non entrées sont considérées comme nulles. Exemple: on veut saisir les données de A(0) à A(4), sachant que la valeur de l'adresse 770 est 9.

texte du masque: VALEUR DU TABLEAU A:<"A;99"   >

à l'exécution : VALEUR DU TABLEAU A:1;;18;4

affectation des valeurs: A(0)=1 A(1)=0 A(2)=18 A(3)=4 A(4) à A(9)=0

Si, en sortie, la fen&tre n'est pas assez grande, une étoile (*) est imprimée.
!1m5
!np
? exemple <"B%/?"> Ne concerne que les variables de sortie. En mode saisie, le programme les ignore et passe directement à la fen&tre suivante.

Tous les effets de ces formats seront montrés plus en détail dans le prochain chapitre. Pour le moment, nous vous proposons la création d'un masque sous le nom de "PRODMASK". Ce masque nous resservira dans les exemples d'utilisation.

LET "£NEW-1,M,0:PRODMASK"

```
   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  <-- sur fond blanc
   * CALCUL DE LA VALEUR D'UNE PRODUCTION *

   NOM DU PRODUIT: <"P$:"        >

   DATE DE FABRICATION: <"DF$*:"    >

   ENTRER LA PRODUCTION TRIMESTRIELLE

        !  1   !  2   !  3   !  4   !
```

Les masques créés lorsque cette carte est utilisée seront toujours du type 80 colonnes. Ils ne pourront donc &tre relus avec une configuration ordinaire à écran 40 colonnes. En revanche, les masques créés en 40 colonnes, avec un système standard ou en 80 colonnes simulées, pourront &tre utilisés avec cette carte.

Les fonctions HOME, PR£0, ainsi que les routines de limitations de la fen&tre d'écran gr&ce qui se servent des pointeurs stockés aux adresses décimales 32,33,34,35 ont été réécrites pour permettre leur utilisation avec la carte SUP'R'TERMINAL. La fonction HTAB ne fonctionne pas au delà de la quarantième colonne.

6.3. VERSION APPLE III:

Sous cette nouvelle version de l'APPLE, qui comprend un écran 80 colonnes, M/DOS 6502 détermine automatiquement si un masque est en version 40 ou 80 colonnes, et fait automatiquement la conversion d'écran.

De plus les fonctions de déplacement vertical du curseur dans les masques (CTRL/Q et CTRL/Z) sont remplacées par des touches spécifiques de déplacement du curseur vers le haut et vers le bas, indiquées par des flèches.
!cj

-
!lj

7. STRUCTURE DES MASQUES

L'objet de cette partie est de montrer comment sont codés les masques du M/DOS 6502. Pour cela, vous verrons successivement le cas du texte et celui des zones de saisie.

7.1. TEXTE DU MASQUE

Il s'agit du texte qui appara&t à l'écran lors de l'exécution de LET "C-<u.l>". Le traitement de ce texte est tout à fait indépendant de celui des zones de saisie.

La particularité du codage du texte provient du "compactage" de ce dernier. Pour gagner de la place, on code sur 4 octets une zone o& un m&me caractère est répété plus de 4 fois. Si un caractère "X" est répété n fois, la zone sera codée:

```
X,code("<"),n si n<256
X,code(">"),poids faible de n, poids fort de n si n>256
```

On utilise comme caractères de contr&le les signes "<" et ">", puisque ceux-ci sont interdits dans les zones de texte (ils servent à délimiter les fen&tres).

La fin de l'écran est repérée par un nombre nul de caractères identiques (configuration normalement impossible).

Exemple de codage du masque:

```
TEST DE MASQUE
--------------




212 - 148 T         160 (blanc)
197 - 133 E         188 <         26 blancs consécutifs
211 - 147 S         26  26
212 - 148 T
160       (blanc)
196 - 132 D         173 -
197 - 133 E         188 <         14 tirets consécutifs
160       (blanc)   14  14
205 - 141 M
193 - 129 A
211 - 147 S         160 (blanc)
209 - 145 Q         190 >         blancs jusqu'à la fin de l'écran
213 - 149 U         0   0
197 - 133 E         0   0
!np
```

7.2. ZONES DE SAISIE

En ce qui concerne les zones de saisie, on enregistrera 7 octets par variable (variable simple ou tableau). Le découpage se fait comme suit:

```
Adresse écran du début de la zone  --> 2 octets
Longueur de la zone                --> 1 octet
Nom de la variable (nom BASIC)     --> 2 octets
Contr&les à effectuer              --> 1 octet
Incices dans le cas d'un tableau   --> 1 octet
(99 dans le cas d'une liste)
```

Le dernier octet n'est pas utilisé dans le cas d'une variable simple.
L'octet de contr&le se décompose quant à lui de la fa&on suivante:

```
2 bits pour le type:  00 : flottant
                      01 : binaire
                      10 : entier
                      11 : alphanumérique

1 bit pour le          0 : date ou numérique gestion
contr&le (*)           1 : autres cas

1 bit pour entrée/     0 : cas général
sortie (?)             1 : sortie seulement

1 bit pour positif     1 : numérique (ou alphanumérique
                           considéré comme numérique) positif
                       0 : dans le cas contraire

1 bit pour autoskip    0 : cas général
                       1 : autoskip (:)

1 bit pour entier (§)  0 : cas général
                       1 : entier, binaire ou flottant entier

1 bit pour tableaux    0 : variable simple
                       1 : tableau
```

Exemple: à la fen&tre ("NG§*:") va correspondre l'octet de contr&le suivant:

```
           00000110
```

!np
Enfin, l'ensemble du masque est structuré de la fa&on suivante (NV est le nombre de variables):

CONTENU		POSITION
numéro logique		0
type ("M")		1
longueur totale:	poids faible	2
	poids fort	3
longueur de la table: (7*NV)	poids faible	4
	poids fort	5
table (7*NV octets)		6
-		
texte du masque		6+(7*NV)

```
!      !      !      !      !
! <  > ! <  > ! <  > ! <  > !
!"A%;1"!"A%;2"!"A%;3"!"A%;4"!
```

ENTRER LES 4 PRIX SOUS FORME DE TABLEAU

<"P*;99" >

VALEUR TOTALE DE LA PROD.: <"PT*?">

2.3. Mode automatique

Il est possible de créer un masque en mode programme: la syntaxe est identique à celle du mode manuel; au moment o& le programme exécute l'instruction LET "£NEW-1,M,..., il donne la main à l'utilisateur pour créer son masque. le processus est alors exactement le m&me qu 'en mode manuel. La touche ESC a pour effet de valider le masque, puis de redonner la main au programme.

Il existe une autre possibilité: dans le programme, remplacer l'instruction NEW habituelle par l'instruction:

LET "£NEW-<u.l>,M,/<d>:<nom du masque>.

Le "/" placé devant le No du disque a pour effet de faire valider comme masque tout ce qui est inscrit sur l'écran, et ce, sans donner la main à l'utilisateur.

Il est à noter dans ce cas que l'on ne peut pas utiliser l'option § dans l'ordre NEW pour sauver un masque sous un nom qui existe déjà.

2.4. CREATION DE MASQUES EN MEMOIRE:

M/DOS offre la possibilité de créer des masques en mémoire sans que ceux-ci ne soient sauvés sur disque.

L'instruction de création d'un masque en mémoire est la suivante:

LET"£IMAGE-<u.l>,M:(NOM)

On pourra notamment utiliser cette option pour:
--> créer des masques temporaires, dont on n'aura plus besoin par la suite.
--> fabriquer directement des masques qui seront ensuite regroupés dans un masque global (rappelons que la création d'un global n'entraine pas la destruction sur le disque des masques simples qu'il contient).
--> utiliser la gestion de masques en DOS 3.2 ou DOS 3.3. Ces systèmes permettent en effet l'utilisation de certaines instructions M/DOS, & condition toutefois que les contenus de certaines adresses de la page 3 aient été préalablement modifiés (cf chapitre sur les adresses du M/DOS).

3. UTILISATION DES MASQUES:

Cette partie présente les différentes instructions d'utilisation d'un masque déjà existant. L'exemple utilisé dans la partie précédente sera repris ici. A la différence des fichiers, toutes les instructions concernant les masques sont utilisables aussi bien en mode direct qu'à l'intérieur de programmes.

3.1. OUVERTURE ET FERMETURE DES MASQUES

Comme les fichiers, les masques doivent &tre ouverts avant d'&tre utilisés. L'instruction d'ouverture est la suivante:

LET"£OPEN-<u.l>,MASQUE,<d>:<nom>

Exemple:LET"£O-1,M,0:PRODMASK

On pourra, en fin d'utilisation, pour récupérer de la place dans les buffers du DOS, fermer le masque par l'instruction:

LET"£CLEAR,<u.l>

qui ferme l'objet correspondant à l'unité logique utilisée, ou en se servant de l'ordre:

LET"£CLEAR,$

qui ferme tous les masques et fichiers ouverts, et efface tous les pointeurs du M/DOS.

En cas d'erreur, l'utilisateur pourra se référer à l'annexe x qui décrit les messages d'erreur.

3.2. LES INSTRUCTIONS D'UTILISATION DES MASQUES

3.2.1. AFFICHAGE DU TEXTE:

LET"CHARGE-<u.l>

provoque l'affichage sur l'écran des textes composant le masque. Les fen&tres et les variables ne sont pas affichées. Cette instruction entraine l'effacement du contenu de l'écran sauf là o& le masque a été défini comme transparent par le caractère § sur fond blanc.
!np
Par exemple, pour le masque PRODMASK créé au chapitre précédent, les instructions:

```
LET"£O-1,M,0:PRODMASK
PRINT"BONJOUR"
LET"C-1"
```

provoqueront l'affichage suivant:

```
BONJOUR
* CALCUL DE LA VALEUR D'UNE PRODUCTION *

NOM DU PRODUIT:

DATE DE FABRICATION:

ENTRER LA PRODUCTION TRIMESTRIELLE

    !  1   !  2   !  3   !  4   !
    !      !      !      !      !
    !      !      !      !      !
    !      !      !      !      !

ENTRER LES 4 PRIX SOUS FORME DE TABLEAU


VALEUR TOTALE DE LA PROD.:
```

3.2.2. SAISIE DES DONNEES:

LET"INPUT-<u.l>,<No de zone>

donne la main à l'utilisateur pour qu'il rentre les données correspondant aux fen&tres de saisie. Le numéro de zone indique au programme à quelle fen&tre doit commencer la saisie, La première portant le numéro 1, qui est aussi la valeur par défaut lorsque le numéro de zone n'est pas précisé. L'ordre des zones ne tient compte ici que des fen&tres de saisie; les zones de sortie ne seront donc pas prises en compte dans cette numérotation. En outre, il est toujours possible de remonter sur des zones de saisie d'ordre inférieur à celui de la zone de début d'entrée des données. Si le numéro de la zone d'entrée indiqué dans l'instruction est supérieur au nombre de fen&tres de saisie, le programme commencera à balayer le masque autant de fois qu'il est nécessaire pour atteindre ce numéro, un compteur étant incrémenté de 1 à chaque changement de fen&tre. Si la dernière zone du masque est en enchainement automatique, le programme ressortira sans donner la main.

Il existe une stricte concordance entre les noms des variables telles qu'elles ont été définies dans les fen&tres et les variables du programme qui utilise le masque. Dans l'exemple de PRODMASK, la saisie du nom de produit affectera une chaine de caractères à P$ sans m&me qu'il soit nécessaire que cette variable ait été définie ou utilisée au préalable dans le programme.
!np
L'instruction LET"INPUT ne provoque ni l'affichage des textes du masque, ni celui des anciennes valeurs des variables qui vont &tre modifiées par la nouvelle saisie. Il sera donc souvent utile de la faire précéder d'un affichage des textes par l'ordre LET"CHARGE, ou d'un affichage des textes et des variables par LET"PRINT (cf infra) si la saisie à effectuer ressemble à la précédente.

Le curseur ne se positionnera, en entrée, que sur les zones de saisie. Pour passer à la fen&tre suivante, il suffit de taper RETURN. Cependant, si la zone est en enchainement automatique, le passage se fait directement lorsque le curseur atteint la fin de la fen&tre.

De plus, en saisie, s'exerce un contr&le du format des données entrées par l'utilisateur, lequel ne pourra pas quitter une fen&tre tant qu'il n'aura pas tapé des données sous le bon format. On ne pourra pas, par exemple, rentrer un mot dans la deuxième fen&tre de PRODMASK, puisque la machine attend une date. Une absence de données dans une fen&tre de saisie (provoquée par un RETURN tapé en début de fen&tre) laissera son ancienne valeur à la variable correspondante.

La validation de la saisie se fait en appuyant sur la touche ESC, laquelle, quel que soit l'endroit o& elle est tapée, valide comme entrée tout ce qui est écrit dans les fen&tres de saisie. Cependant, si la dernière variable est assortie d'un ":" (enchainement automatique), le dernier RETURN a la valeur d'un ESC. Dans le cas contraire, un RETURN tapé sur la dernière fen&tre du masque provoque un retour à la première fen&tre. Un ESC tapé à cet endroit validera d'ailleurs l'ensemble des entrées sans qu'il soit besoin de tout retaper.

ATTENTION: L'ordre INPUT ne provoque pas l'affichage des anciennes valeurs des variables. Une fen&tre vide ne signifie pas que la variable correspondante soit vide. Pour &tre bien s&r de mettre à zéro(ou à blanc) une valeur, il faut taper un blanc, puis RETURN. Si l'on tape un RETURN seul, l'ancienne valeur, m&me si cette dernière n'appara&t pas à l'écran, est conservée. On peut pallier cet inconvénient en utilisant l'ordre TAKE (voir plus loin).

Les fonctions suivantes sont disponibles en mode saisie:

RETURN : déplace le curseur sur la fen&tre suivante.
CTRL/R : remonte le curseur sur la fen&tre de saisie précédente.
---> : déplace, à l'intérieur d'une fen&tre, le curseur vers la droite.
<--- : déplace, à l'intérieur d'une fen&tre, le curseur vers la gauche.
CTRL/D : efface, dans une fen&tre, le caractère précédant le curseur.
CTRL/I : insère un caractère à l'endroit du curseur en décalant la suite des caractères figurant dans la fen&tre.
CTRL/A : ordonne l'abandon de la saisie. Les données figurant sur l'écran ne sont pas transmises et les variables gardent leurs anciennes valeurs. La variable STATUS prend la valeur 1.
ESC : valide la saisie.
!np

Application au masque PRODMASK:

```
LET"£0-1,M,0:PRODMASK
PRINT"BONJOUR"
LET"C-1
LET"I-1,2"
```

provoque l'affichage des textes du masque et attend que l'utilisateur rentre ses données, la saisie commen&ant à partir de la deuxième fen&tre, c'est à dire la date.

```
BONJOUR
* CALCUL DE LA VALEUR D'UNE PRODUCTION *
NOM DU PRODUIT: BOULONS

DATE DE FABRICATION: 06/08/81
```

```
ENTRER LA PRODUCTION TRIMESTRIELLE

     !  1   !  2   !  3   !  4   !
     !      !      !      !      !
     ! 100  ! 150  ! 120  ! 160  !
     !      !      !      !      !

ENTRER LES 4 PRIX SOUS FORME DE TABLEAU

0.80;0.85;0.90;1.00

VALEUR TOTALE DE LA PROD.:
```

Dans cet exemple, la dernière fen&tre, associée au texte "valeur totale de la prod.", est en sortie. Le curseur, quoiqu'on fasse, ne se positionnera jamais sur cette fen&tre. Un RETURN tapé pour sortir de la fen&tre de saisie des prix le placera donc au début de la zone de saisie du nom du produit.

Après que l'utilisateur a tapé ESC, le programme reprend la main et les variables ont les valeurs suivantes:

```
P$="BOULONS"   DF$="06/08/81"
A%(1)=100  A%(2)=150  A%(3)=120  A%(4)=160
P(1)=0.80  P(2)=0.85  P(3)=0.90  P(4)=1.00
```

ATTENTION: On notera que, lorsqu'une variable prend la valeur zéro, celle-ci est remplacée à l'affichage, aussi bien en saisie qu'en sortie, par un blanc. L'utilisateur ne s'étonnera donc pas de voir disparaitre, après avoir changé de fen&tre, le 0 qu'il vient de taper. Une seule exception: lorsque la valeur associée à l'indice zéro d'un tableau est nulle, elle n'est pas remplacée à l'affichage par un caractère blanc.
!np
3.2.3. AFFICHAGE DES VARIABLES:

LET"OUTPUT-<u.l>,(0/0)

affiche sur l'écran les valeurs prises par les variables du masque. Le dernier paramètre de l'instruction est facultatif. En cas d'omission, toutes les variables du masque sont affichées. S'il prend la valeur 0, seules les variables définies en entrée (celles qui sont saisies par l'ordre LET"INPUT) sont affichées, et s'il prend la valeur 0, seules les variables définies en sortie le seront.

ATTENTION: Au cas o& une variable (ou un ensemble de variables dans le cas d'un tableau) serait trop longue pour entrer dans la fen&tre, elle serait tronquée et suivie du caractère "*" en fin de fen&tre.

exemple d'utilisation de OUTPUT:

LET"0-1"

donnera, dans l'exemple de PRODMASK, l'affichage suivant:

BOULONS

06/08/81

```
          100     150     120     160


     0.80;0.85;0.90;1.00

                                    475.50
```

Nous avons évidemment supposé que le cacul de la production totale (475.5) avait été fait au préalable dans le programme.

ATTENTION: lors de la sortie des variables, aucun contr&le n'est effectué par le masque au niveau de la nature des variables. Par exemple, si on affecte dans un programme la valeur 1000 à N%, et que cette valeur est sortie dans un masque sous le format N%/ (binaire: en fait, entier <256), il n'y aura aucun message d'erreur. Seules, les instructions de cadrage dans les formats seront prises en compte (cadrage à droite, format gestion)
!np
Voici un exemple qui vous permettra de vous familiariser avec ceci. Créez le masque "VMASK" et le programme "VP.SOR.MASK" reproduits ci-dessous. Essayez à l'exécution d'introduire des binaires > 255, des entiers négatifs, des "flottants entiers" avec des virgules, etc...

MASQUE "VMASK"

```
     NOMBRE BINAIRE: <"B%/" >

     DATE: <"D$*"    >

     FLOTTANT FORMAT GESTION: <"F*"       >

     CHAINE-NUMERIQUE    :<"CN$+"      >
     FORMAT GESTION

     FLOTTANT ENTIER: <"FE$"       >

     ENTIER POSITIF: <"EP%+" >
```

PROGRAMME VP.SOR.MASK

```
10  LET "£0-1,M,0:VM" : HOME
15  PRINT "INTRODUISEZ UNE VALEUR POUR:"
20  INPUT "B%/: ";B%
30  INPUT "D$*: ";D$
40  INPUT "F* : ";F
50  INPUT "CN$+:";CN$
60  INPUT "FE&: ";FE
70  INPUT "EP%+:";EP%
75  LET"C-1":LET"0-1"
80  VTAB 23 : HTAB 1 : INPUT "OK POUR CONTINUER?";R$
90  IF R$="OK" THEN 15
95  LET"£C,$" : END
```

3.2.4. AFFICHAGE DU TEXTE ET DES VARIABLES:

LET"PRINT-<u.l>,(O/O)

Cette instruction réalise à la fois un "CHARGE" (affichage du texte) et un "OUTPUT" (affichage des variables). Le paramètre (O/O) est optionnel et son utilisation est la m&me que pour l'ordre OUTPUT.
!np
exemple; LET"P-1 donnera sur l'écran:

```
* CALCUL DE LA VALEUR D'UNE PRODUCTION *
NOM DU PRODUIT: BOULONS

DATE DE FABRICATION: 06/08/81

ENTRER LA PRODUCTION TRIMESTRIELLE

   !  1   !  2   !  3   !  4   !
   !      !      !      !      !
   ! 100  ! 150  ! 120  ! 160  !
   !      !      !      !      !

ENTRER LES 4 PRIX SOUS FORME DE TABLEAU

0.80;0.85;0.90;1.00

VALEUR TOTALE DE LA PROD.:   475.50
```

3.2.5. AFFICHAGE DU TEXTE, DES FENETRES ET DES FORMATS:

LET"VISUALISE-<u.l>

imprime le masque tel qu'il était au moment o& sa création a été validée. Le texte, les fen&tres et leur contenu seront donc affichés. Cette instruction est surtout utilisée pour la modification d'un masque existant déjà (cf infra le paragraphe sur la modification des masques).

exemple:

LET"V-1 entra&ne l'affichage suivant:

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
* CALCUL DE LA VALEUR D'UNE PRODUCTION *
NOM DU PRODUIT: <"P$:"        >

DATE DE FABRICATION: <"DF$*:"    >

ENTRER LA PRODUCTION TRIMESTRIELLE

   !  1   !  2   !  3   !  4   !
   !      !      !      !      !
   ! <  > ! <  > ! <  > ! <  > !
   !"A%;1"!"A%;2"!"A%;3"!"A%;4"!
   !      !      !      !      !

ENTRER LES 4 PRIX SOUS FORME DE TABLEAU
```

<("P*;99"   >

VALEUR TOTALE DE LA PROD.: <PT*?">
!np
3.2.6. SAISIE DIRECTE DES VARIABLES FIGURANT SUR L'ECRAN:

LET"TAKE-<u.1>

Cette instruction affecte aux variables du masque, définies en entrée, les valeurs qui figurent sur l'écran au moment de son exécution.

On pourra par exemple utiliser cette instruction pour remettre à zéro ou à blanc les variables d'un masque qui aura été préalablement chargé sur l'écran.

LET"C-1" :LET"T-1" réalise cette initialisation.

3.3. MASQUE ISSU D'UN PRECEDENT ET MODIFICATION

3.3.1. MASQUE ISSU D'UN PRECEDENT:

Il est possible, à partir d'un masque existant déjà, d'en créer un nouveau. Il suffit d'ouvrir et de visualiser le masque initial, puis de donner l'ordre de création d'un masque, lequel donnera la main à l'utilisateur pour qu'il effectue les modifications voulues.
La séquence d'instructions pour cette opération sera donc:

LET"£O-1,M,<d>:MASKINITIAL
LET"V-1" : LET"£NEW-2,M,<d>:MASKFINAL

Après avoir tapé ESC, pour valider la création du nouveau masque (MASKFINAL), l'utilisateur disposera des deux masques sur le disque (MASKINITIAL et MASKFINAL dans cet exemple).

3.3.2. MODIFICATION D'UN MASQUE:

Le procédé est très peu différent de celui de la création d'un masque à partir d'un précédent. Au lieu d'utiliser l'instruction de création d'un nouveau masque, il faudra se servir de l'ordre de remplacement d'une ancienne version d'un masque par une nouvelle, à savoir:

LET"£NEW-(n),M,$<d>:(NOM)

exemple: LET"£O-1,M,0:MASQUE
LET"V-1" : LET"£N-2,M,$0:MASQUE"

Ces instructions ont pour effet de donner la main à l'utilisateur pour qu'il modifie le masque. Les modifications sont validées par la touche ESC, laquelle provoque la destruction sur le disque de l'ancienne version, qui reste cependant chargée en mémoire centrale, et son remplacement par la nouvelle.
!np
4. UTILISATION DE M/DOS POUR LES SORTIES SUR IMPRIMANTE:

4.1. COPIE D'UNE OU PLUSIEURS LIGNES

M/DOS offre la possibilité de faire sortir sur l'imprimante une ou

plusieurs lignes d'un masque figurant sur l'écran sans modifier la présentation de l'écran lui-m&me.

Chaque ligne à imprimer doit commencer et finir par une lettre sur fond blanc, qui sert à la repérer. Ces caractères de repérage s'afficheront sur l'écran, mais ne seront pas copiés sur l'imprimante.

exemple: Considérons le masque suivant, qui aura été affiché grâce, par exemple, à l'instruction LET"CHARGE.

<O> IMPRESSION <O>

METTRE L'IMPRIMANTE EN MARCHE!

<L> VOUS ETES SOUS M/DOS <L>

NB:<O> représente la lettre O sur fond blanc.

la première et la troisième ligne de ce masque pourront &tre copiées sur l'imprimante, mais pas la deuxième.

Toutes les lettres de l'alphabet, et seulement elles, sont utilisables comme caractère de repérage.
En outre, des lignes différentes doivent &tre identifiées par des lettres différentes.

Avant de lancer l'impression, il faudra tout d'abord:

--> définir l'imprimante qu'on désire utiliser par l'instruction:
PR£(No du slot sur lequel est connecté l'interface de l'imprimante)

--> protéger l'écran si la carte interface réalise l'affichage à l'écran en m&me temps que l'impression. L'instruction PRINT CHR$(9)"80n", par exemple, permet, pour les interfaces en parallèle, de le garder en l'état malgré la copie sur l'imprimante.

On peut maintenant lancer l'impression par l'instruction:

LET"?-(caractère)

qui provoque la copie de la ligne entouré de (caractère) sur fond blanc.

PR£0 permet ensuite de revenir à l'affichage sur l'écran.
!np
exemples: le masque précédent est toujours affiché.

```
PR£1
PRINT CHR$(9)"80N"
LET"?-L"
PR£0
```

donnera la copie du texte suivant:

VOUS ETES SOUS M/DOS

```
PR£1
PRINT CHR$(9)"80N"
LET"?-O" : LET"?-L"
PR£0
```

fera imprimer:

```
          IMPRESSION:
        VOUS ETES SOUS M/DOS
```

```
PR£1
PRINT CHR$(9)"80n"
LET"?-L" : LET"?-O"
```

inversera l'ordre des lignes par rapport à l'exemple précédent:

```
        VOUS ETES SOUS M/DOS
          IMPRESSION:
```

Dans aucun de ces exemples la présentation de l'écran n'a changé. Il contient toujours le masque tel qu'il a été défini.

4.2. COPIE COMPLETE DE L'ECRAN

L'instruction LET"?-*" réalise la copie complète de l'écran sur l'imprimante.
Là aussi, il faudra au préalable avoir défini l'imprimante et protégé la présentation de l'écran.

exemple: l'écran contient toujours le masque utilisé comme exemple au cours du paragraphe précédent.

```
!np
PR£1
PRINT CHR$(9)"80n"
LET"?-*"
```

réalise la copie suivante:

```
       O IMPRESSION O

 METTRE L'IMPRIMANTE EN MARCHE!

     L VOUS ETES SOUS M/DOS L
```

On notera que, dans ce cas, les caractères de repérage sont imprimés.

4.3. COMPLEMENTS

La ligne de copie peut représenter plusieurs lignes de l'écran. On pourra donc réaliser des impression sur toute la largeur de l'imprimante (80 colonnes, 132 colonnes...).

exemple: soit le masque suivant:

```
 <0> CE TEXTE NE TIENT PAS SUR UNE LIGNE
     D'ECRAN <0>
```

LET"?-0" donnera sur l'imprimante:

```
     CE TEXTE NE TIENT PAS SUR UNE LIGNE D'ECRAN
```

Il faudra toutefois veiller à ce que la longueur du texte entouré de caractères de repérage sur fond blanc identiques ne dépasse pas celle de l'imprimante, car les caractères excédentaires ne seraient pas imprimés.

ATTENTION: LET"?-*" reproduit l'écran strictement et donnera:

```
  0 CE TEXTE NE TIENT PAS SUR UNE LIGNE
    D'ECRAN 0
```

!np

On notera enfin que M/DOS gère le nombre de lignes imprimées. Celui-ci figure à l'adresse 947, et chaque fois qu'une ligne est imprimée son contenu est incrémenté de 1. lorsque la valeur 66 est atteinte, il repasse à zéro. L'ordre LET"?" génère un FORM FEED (haut de page) et remet le compteur à zéro.

L'utilisation des masques pour les impressions permet de gagner de la place mémoire. En effet, les définitions de masques figurent dans une partie de la mémoire à laquelle l'utilisateur n'a pas accès (les buffers du DOS). Il gagne donc, dans la partie de la RAM o& réside le programme, la place qu'il aurait consacrée à des instructions de sortie souvent très volumineuses à cause du texte. En outre les masques facilitent la définition des formats d'impression, notamment lorsqu'il s'agit de sortir des variables indicées.

5. LES MASQUES GLOBAUX:

5.1. PRESENTATION

Un masque global regroupe plusieurs masques simples tels qu'ils viennent d'&tre présentés. Cet ensemble de masques figurera sur le disque sous un seul nom (celui du masque global), et sera précédé dans le catalogue de la lettre G (rappelons que M représente un masque simple, P un programme, B un module binaire).

Les masques globaux présentent les avantages suivants:

--> gain de place sur le disque: en effet un masque isolé occupe sur disque la place d'un nom dans le catalogue et un nombre

entier de secteurs. L'utilisation d'un masque global permettra donc de réduire le nombre de noms et d'optimiser l'occupation du disque.

--> gain de vitesse lors de l'exécution des programmes, car le chargement de l'ensemble des masques constituant un masque global se fera en un seul accès disque et avec, en outre, un chargement minimum de blocs en mémoire.

--> gain de place en mémoire pour les programmes, le masque global étant chargé dans les buffers du DOS.

--> raccourcissement des programmes: une seule instruction d'ouverture permet de disposer de plusieurs masques simples pr&ts à &tre utilisés dans le programme.

Cependant, la modification des masques ainsi réunis est beaucoup plus difficile à réaliser que dans le cas de masques simples. Il sera donc préférable de ne constituer les masques globaux qu'une fois la mise au point des programmes terminée.
!np
5.2. UTILISATION

5.2.1. CREATION D'UN MASQUE GLOBAL:

Il faut tout d'abord ouvrir les masques que l'on veut regrouper.

```
exemple:     LET"£0-1,M,0:MASK1
             LET"£0-2,M,0:MASK2
             LET"£0-3,M,0:MASK3
             LET"£0-4,M,0:MASK4
```

On crée ensuite le masque global grâce à l'instruction:

LET"£NEW,<u.l>,G,<d>:<NOM>

exemple: LET"£N,2,G,<d>:MASKGLOBAL"

Les masques simples figureront toujours sur la disquette, mais ils pourront &tre détruits pour gagner de la place, car l'ensemble des informations est contenu dans le masque global.

ATTENTION: La création d'un masque global a pour effet de fermer tous les fichiers ouverts ainsi que les drives autres que celui sur lequel se fait l'opération.

5.2.2. UTILISATION DES MASQUES GLOBAUX:

Il faut, avant d'utiliser un masque global, l'ouvrir (sauf s'il vient d'&tre créé et qu'il n'a pas été fermé) par l'instruction:

LET"£OPEN,<u.l>,G,<d>:<NOM>

exemple: LET"£0,A,G,0:MASKGLOBAL

Tous les masques composant MASKGLOBAL sont maintenant accessibles par le numéro d'unité logique sous lequel il ont été ouverts pour sa création. Dans l'exemple de MASKGLOBAL, il faudra donc établir la correspondance suivante:

unité logique	nom du masque
1	MASK1

```
2                    MASK2
3                    MASK3
4                    MASK4
```

On voit donc que le numéro d'unité logique, sous lequel un masque a été ouvert pour la création d'un global, est fondamental, car il servira par la suite à le différencier des autres éléments du masque global.

En revanche, le numéro d'unité logique indiqué lors de la création ou de l'ouverture de globaux n'a pas d'importance. Toutefois le système n'acceptera pas un numéro d'unité logique déjà utilisé, sauf s'il correspond à un autre masque global.
!np

```
exemple:     LET"£0,1,F,0:FICHIER
             LET"£0,1,G,0:MASKGLOBAL  ne marchera pas!

             LET"£0,1,G,0:MASKGLOBAL
             LET"£0,1,G,0:MASKGLOBALBIS   marchera!
```

Il est possible de charger un masque global dont un des éléments est identifié par une unité logique qui est déjà associée à un autre objet (masque simple, élément d'un global ,fichier). Dans ce cas l'unité logique identifie le premier objet qui a été chargé.

exemple 1:

```
LET"£0-1,M,0:MASQUESIMPLE
LET"£0,G,G,0:MASKGLOBAL
LET"C-1"                chargera, non pas MASK1 (élément de MASKGLOBAL),
                        mais MASQUESIMPLE qui a été ouvert le premier
                        sous l'unité logique 1.
```

Il faudra donc, pour utiliser MASK1, fermer MASQUESIMPLE, l'unité logique 1 étant alors automatiquement affectée a MASK1. On utilisera donc la séquence suivante:

```
LET"£C,1    ferme MASQUESIMPLE et l'unité logique 1 est attribuée à MASK1
LET"C-1"    affiche les textes de MASK1
```

exemple 2: MASKGLOBALBIS regroupe MASKBIS1 identifié par l'unité logique 1 et MASKBIS2 par la 2.

```
LET"£0,G,G,0:MASKGLOBALBIS
LET"£0,G,G,0:MASKGLOBAL
LET"C-1"                    affiche les textes de MASKBIS1
LET"C-3"                     "           "         MASK3
LET"£C,1                    ferme MASKBIS1
LET"C-1"                    affiche les textes de MASK1
LET"£C,1                    ferme MASK1
LET"£0,G,G,0:MASKGLOBALBIS
LET"C-1"                    affiche les textes de MASKBIS1
```

6. LA GESTION DE MASQUES EN 80 COLONNES:

Ce paragraphe décrit essentiellement quelques principes concernant la compatibilité entre les masques 40 et 80 colonnes.

6.2. LA CARTE SUP'R' TERMINAL:

CHAPITRE VII: LES EXTENSIONS DU BASIC

1. FONCTIONS DE CALCUL SUR 48 CHIFFRES:

La précision des calculs, en utilisation normale, est parfois insuffisante pour certaines applications, notamment en gestion. M/DOS contient des fonctions de calcul (addition, soustraction, arrondi) qui opèrent sur 48 chiffres, avec un nombre de décimales fixé par l'utilisateur entre 1 et 9.

Ces fonctions manipulent des cha&nes de caractères. Les nombres à traiter devront donc &tre stockés dans des variables de type alpha-numérique. Le résultat de l'opération est toujours dans la variable WW$.

La syntaxe des différentes opérations est la suivante:

--> addition: LET"=opérande 1+opérande 2"
ou LET"="+V1$+"+"+V2$ o& V1$ et V2$ sont des variables.

exemples: LET"=345.3+6.25":PRINT WW$ fera afficher le nombre 351.55

LET"="+V1$"+"+V2$:PRINT WW$ avec V1$=345.3 et V2$=6.25 donnera le m&me résultat

--> soustraction: LET"=opérande 1-opérande 2"
ou LET"="+V1$+"-"+V2$

--> arrondi: LET"=opérande" ou LET"="+V1$

exemple: LET"=325.1274":PRINT WW$ donnera à l'affichage, si le nombre de décimales a été fixé & 2, 325.12

L'arrondi se fait toujours au nombre de décimales fixé par l'utilisateur, et en prenant la valeur par défaut (les décimales excédentaires sont tronquées).

La précision des calculs est fixée par l'instruction:

LET"§-(n)" avec (n)=nombre de décimales (entre 1 et 9)

exemples: LET"§-2":LET"=3.124+2.243":PRINT WW$
fera afficher le nombre 5.36 . Les troisièmes décimales sont ignorées par le système qui calcule en fait 3.12+2.24 .

LET"§-4":LET"=3.124+2.243":PRINT WW$
donnera cette fois ci 5.367 .

Si le nombre de décimales n'est pas précisé par l'utilisateur, le système prendra la valeur par défaut qui est 2.

On notera que cette instruction fixe, en m&me temps que la précision des calculs, le nombre de décimales pour l'affichage en format gestion dans les masques (cf le chapitre sur les masques pour plus de détails).
!np

2. FONCTION EXECUTE:

2.1. PRESENTATION:

Inspirée du langage APL, la fonction EXECUTE permet de construire et de faire exécuter par programme des entrées à partir d'un buffer. Il pourra aussi bien s'agir de données qui seront lues ensuite par un INPUT (ou un GET) que d'instructions qui seront affichées puis exécutées.

Le texte des données ou des instructions doit d'abord &tre enregistré dans un buffer, ce qui se fait de la fa&on suivante:

LET"> " pour entrer le texte

LET"+ " pour en rajouter à la fin du buffer

Lorsque la syntaxe exige de faire se succéder deux guillemets, on en remplacera un par CHR$(34).
Les lignes de texte, à l'exception de la dernière, devront se terminer par CHR$(13).

exemple: LET">HOME:VTAB 10"+CHR$(13)

LET"+PRINT"+CHR$(34)+"BONJOUR"

enregistre dans le buffer la séquence suivante:
HOME:VTAB 10:PRINT"BONJOUR"

Le contenu du buffer est exécuté grace à l'ordre:

LET"!"

qui devra &tre suivi d'un END si le buffer contenait des instructions. Le END ne sera pas nécessaire s'il ne contenait que des données. Le buffer est vidé après l'exécution.

L'instruction LET"V" permet de relire le contenu du buffer.

Les ordres sont exécutés comme s'ils étaient tapés en mode direct. Ils seront donc affichés avant d'&tre exécutés. Les boucles, par exemple, devront tenir sur une seule ligne du buffer. De plus les messages erreur s'afficheront m&me si un ONERR GOTO se trouve dans le programme.

exemple:Le programme suivant calcule la somme de 5 nombres entiers successifs, à partir d'une valeur entrée par l'utilisateur.

```
10 HOME:INPUT"DONNER UN NOMBRE ENTIER:";N
20 A$="PRINT"+CHR$(34)+"SOMME="+CHR$(34)
30 LET">FOR I=N TO N+5:S=S+I:NEXT"+CHR$(13)
40 LET"+"+A$+";S"
60 LET"!":END

 RUN
DONNER UN NOMBRE ENTIER:"1
 FOR I=N TO N+5:S=S+I:NEXT

 PRINT"SOMME=";S
SOMME=21
!np
```

2.2. APPLICATIONS DE L'EXECUTE:

Ce paragraphe présente quelques utilisations possibles de la fonction EXECUTE.

2.2.1. MODIFICATION DE L'ENTREE CLAVIER:

Les données ne sont plus lues au clavier, mais dans un buffer du DOS.

```
exemple: 10 LET">XYZ"
         20 LET"+ABC"
         30 LET"!"
         40 INPUT A$
         50 PRINT A$
         RUN
         XYZABC
```

Les caractères contenus dans le buffer peuvent &tre lus par les ordres INPUT et GET.
On note que la ligne 10 ne se termine pas par CHR$(13). C'est pourquoi "ABC" a &t& concaténé à XYZ.

```
         10 LET">XYZ"+CHR$(13)
         20 LET"+ABC"
         30 LET"!"
         40 INPUT A$,B$
         50 PRINT A$,B$
         RUN
         XYZ              ABC
```

Le CHR$(13) permet d'isoler les deux cha&nes

2.2.2. PASSAGE DE PARAMETRES D'UN PROGRAMME A UN AUTRE:

Vous désirez faire passer le contenu des variables A$ et B$ d'un programme A à un programme B exécuté à la suite.

```
programme A: 60 LET">"+A$+CHR$(13)+B$
             70 RUN"PROGRAMME B"
programme B: 10 LET"!"
             20 INPUT A$,B$
                :
                :
               etc...
```

Les valeurs des variables sont stockées dans le buffer au cours de l'exécution du premier programme et lues par le second. Lorsque tous le buffer a &t& lu, la saisie se refera normalement, à partir du clavier.

2.2.3. LECTURE DES DICTIONNAIRES:

L'exemple suivant stocke dans le buffer le dictionnaire d'un fichier et teste le premier caractère pour savoir si le fichier est relatif.

```
10 LET"£0-1,F,0:FILE"
20 LET"ENTER-1"
30 LET"!"
40 GET A$:IF A$="§" THEN PRINT"FICHIER RELATIF"
!np
```

2.2.4. EXECUTION D'ORDRES BASIC ENTRES PAR L'UTILISATEUR:

L'exemple suivant fait exécuter une formule de calcul saisie au clavier.

```
 5 X=5 : Z=3
10 INPUT "Y=";A$
20 LET"Y="+A$+":GOTO 100"+CHR$(13)
30 LET"+GOTO 200"
50 LET"!
60 END
100 IN£0:PRINT"Y=";Y:GOTO 10
```

```
200 PRINT"ERREUR!": GOTO 10
```

Le END rend la main au Basic qui exécute les ordres contenus dans le buffer. En cas d'erreur dans la formule saisie au clavier, la première ligne d'instructions du buffer ne s'exécutera pas entièrement. Le GOTO 100 ne se fera donc pas, et c'est le GOTO 200 qui s'exécute alors.
Le IN£0 en ligne 100 permet d'interrompre l'entrée à partir des buffers et d'éviter ainsi l'exécution du GOTO 200 lorsqu'il n'y a pas d'erreur.

exemple d'exécution du programme précédent:

```
 RUN
Y=?2*X+Z-4


 Y=2*X+Z-4:GOTO 100
9
Y=?
```

2.2.5. AFFICHAGE CONTINU DU CATALOGUE D'UN DISQUE:

L'affichage du catalogue d'un disque (ordre LET"*") s'interromp toutes les 20 lignes et attend que l'utilisateur tape sur une touche pour continuer. Pour obtenir un affichage continu, il suffit de faire:

```
10 LET">AAAAAAA
20 LET"!"
30 LET"*"
40 IN£0
```

Le IN£0 permet de rétablir l'entrée à partir du clavier, car on ne sait pas si le buffer est vide à la fin du catalogue.
!np
3. SOUS-PROGRAMMES AVEC ARGUMENTS:

3.1. PRESENTATION:

Inspirés des "subroutines" du langage FORTRAN, ces sous-programmes, prévus dans la nouvelle norme BASIC, présentent avec les sous-programmes standards appelés par GOSUB les différences suivantes:

-->Le sous-programme possède un nom qui lui est propre, et c'est par ce nom qu'il appelé gr&ce à l'instruction CALL FN "Nom"+arguments.

-->On peut établir des correpondances entre des variables du programme appelant et des variables de la subroutine portant un nom différent:

exemple 1:
```
10 A=5
20 CALL FN "SP" (A,B)
30 PRINT"B=";B
40 END
50 DEF FN "SP"(C,D)
60 D=C+1
70 END FN
```

A,B,C,D sont ce que l'on qualifie d'arguments.

Le PRINT "B=";B de la ligne 30 fera afficher le nombre 6.
En effet C et D sont des variables locales au sous-programme "SP"; mais

il y a correspondance entre les variables A et C, et B et D. C'est à dire que A représente la m&me adresse mémoire que C, et il en va de m&me pour B et D.
A l'entrée dans la subroutine, la valeur de C est donc la m&me que celle de A, c'est à dire 5. On a de m&me: D=B=0. Dans le sous-programme, la valeur 6 est affectée à D, et cette modification se retrouve en sortie pour la variable B qui vaut aussi 6 après l'appel de la subroutine.

-->Enfin, on peut utiliser dans les subroutines des variables purement internes qui sont totalement ignorées par le programme appelant. Pour les mettre en évidence, nous avons choisi, par convention, de les séparer des autres arguments par un "/" (l'utilisation des séparateurs est décrite dans le paragraphe suivant).

exemple 2:
```
10 A=5 : Z=99
20 CALL FN "SP"(A,B)
30 PRINT "B=";B
40 PRINT "Z (PROG. PRINCIPAL)=";Z
50 END
60 DEF FN "SP"(C,D/Z)
70 D=C+1
80 Z=25+Z
90 PRINT "Z (SOUS-PROG.)=";Z
95 END FN
```

Ce programme donnera à l'exécution l'affichage suivant:

```
Z (SOUS-PROG.)=25
B=6
Z (PROG. PRINCIPAL)=99
```

Les deux variables Z sont totalement distinctes: leurs contenus sont stockés dans des adresses mémoire différentes.
!np
A l'entrée dans la subroutine, Z vaut 0, d'o& la valeur 25 à l'affichage. Dans le programme principal, Z vaut toujours 99 puisqu'il s'agit d'une variable différente de celle du sous-programme, et qu'elle n'a pas été modifiée. En revanche, C et D sont, comme dans l'exemple 1, en communication avec A et B.

Vous vous demandez peut-&tre quels peuvent &tre les avantages de ce type de sous-programmes par rapport à ceux du Basic standard. L'exemple suivant en montre quelques uns.

exemple 3: Supposons que vous vouliez, à partir de variables contenant des prix hors taxe et des taux de TVA, calculer dans un sous-programme les prix toutes taxes comprises. Comparez les deux programmes suivants (A,B,C étant des prix; D,E,F des taux de TVA):

avec un sous-programme standard:

```
100 X=A:Y=D
110 GOSUB 200
120 X=B:Y=E
130 GOSUB 200
140 X=C:Y=F
150 GOSUB 200
160 END
200 Z=X*(1+Y)
210 PRINT "PRIX TTC=";Z
220 RETURN
230 END
```

avec un sous-programme CALL FN:

```
100 CALL FN "TTC" (A,D)
110 CALL FN "TTC" (B,E)
120 CALL FN "TTC" (C,F)
130 END
140 DEF FN "TTC" (X,Y)
150 Z=X*(1+Y)
160 PRINT "PRIX TTC=";Z
170 END FN
```

Vous voyez que les subroutines avec arguments permettent de gagner des instructions, et ce d'autant plus que les variables à transférer du programme principal vers le sous-programme sont nombreuses, ce qui est notamment le cas lorsqu'on manipule des tableaux. Les subroutines avec argument seront donc beaucoup plus facilement adaptable à un programme appelant quelconque que les sous-programmes standard qui sont en général très fortement dépendants d'un programme particulier pour lequel ils ont été con&us.

3.2. SYNTAXE:

Les exemples du paragraphe précédent vous ont déjà permis de vous familiariser avec les principes généraux de la syntaxe des fonctions avec arguments qui va maintenant &tre étudiée plus en détail.

Les sous-programmes avec arguments doivent commencer par:

DEF FN "Nom du sous-programme" (liste de variables)

La liste comprend toutes les variables locales. Viendont d'abord celles qui correpondent à des variables du programmes appelant (C et D dans l'exemple 2), puis celles qui sont purement internes à la subroutine et n'ont de ce fait aucun équivalent, du point de vue des adresses mémoire, dans le programme appelant (Z dans l'exemple 2).

Ils doivent se terminer par l'ordre:

END FN qui indique la fin du sous-programme et a valeur de RETURN.
!np
L'appel des subroutines se fait par l'instruction:

CALL FN "Nom de la subroutine" (liste de variables)

Il s'établit, dans l'ordre, une correspondance entre les variables de cette liste et celles de la liste de l'ordre FN. Le premier argument de la liste de l'instruction CALL FN partagera la m&me zone mémoire que le premier de la liste de l'ordre DEF FN. Et de m&me pour les suivants s'il y en a....

Lorsque la liste de l'ordre DEF FN comprend plus de variables que celle d'un ordre d'appel, les dernières variables (celles dont le rang dans la liste est supérieur à celui de la dernière variable de l'ordre CALL) sont considérées comme strictement locales pour cet appel.

Les différentes variables des listes doivent &tre isolées par des séparateurs qui peuvent &tre indifféremment: "," "/" "=" .

Les variables dimensionnées communiquant entre le programme appelant et la subroutine devront &tre suivies, dans les listes, de ";" pour indiquer qu'il s'agit de tableaux. Le dimensionnement donné dans le programme appelant s'appliquera aussi à la variable correspondante dans la subroutine.

Toutes les variables qui ne sont pas précisées dans les listes sont communes au programme appelant et au sous-programme.
On notera enfin qu'il est impossible de passer des constante en argument.

L'exemple suivant va vous permettre de vous familiariser avec toutes ces notions. Ce programme réalise le rangement dans un ordre croissant des éléments d'un tableau compris entre deux bornes fixées par l'utilisateur.

Le tri est effectué dans une subroutine:

```
10 DIM T(10)
20 C=99
30 INPUT "BORNE INF, BORNE SUP:";A,B
40 FOR I=A TO B:T(I)=RND(5):PRINT T(I):NEXT
50 CALL FN "TRI.TAB" (T;=A,B)
60 PRINT "I=";I
70 FOR I=A TO B:PRINT T(I):NEXT
80 END
90 REM
100 DEF FN "TRI.TAB" (V;=X,Y/I,M)
110 DIM M(2)
120 REM M(1):VARIABLE TEST DE FIN DE TRI, M(2):VARIABLE DE TRANSIT
130 M(1)=0
140 FOR I=X TO Y-1
150 IF V(I+1) > V(I) THEN 180
160 M(2)=V(I+1): V(I+1)=V(I):V(I)=M(2)
170 M(1)=1
180 NEXT I
190 IF M(1) <>0 THEN 130
200 I=0 :PRINT "C=";C
210 END FN
```

Regardez bien ce programme, tapez le et essayez de l'exécuter.
!np
V et T correspondent aux m&mes adresses mémoire. A l'entrée dans le sous-programme, V contient les valeurs initialement affectées à T. A la sortie, T contiendra, pour les indices compris entre les bornes fixées, les valeurs triées. Les variables A et X, B et Y sont aussi identique du point de vue adresse mémoire.

En revanche, la variable I est purement locale. Elle est remise à zéro à la fin du sous-programme. Cependant, l'instruction de la ligne 70 fera afficher pour I la valeur de B+1. C'est normal puisqu'il y a en fait deux variables I correpondant à deux adresses différentes. c'est la valeur de la variable I du programme principal qui est affichée.

C ne figure dans aucune des listes. Elle est donc connue et identifiée sous le m&me nom aussi bien par le programme appelant que par la subroutine.

Vous avez peut-&tre remarqué que M n'est pas suivi de ";" bien qu'il s'agisse d'un tableau. Cela provient tout simplement du fait que M est une variable strictement interne qui n'a pas encore été dimensionnée.

Un sous-programme peut en appeler d'autres. Il peut m&me s'appeler lui-m&me (récursivement) jusqu'à un maximum de 255 niveaux de récursivité.

L'exemple suivant qui calcule la factorielle d'un nombre illustre ce principe de récursivité:

```
1000 INPUT"FACTORIEL (TAPEZ 0 POUR SORTIR):";N%
1010 IF N%<=0 THEN 1050
1020 CALL FN "FACTORIEL" (N%,P%)
1030 PRINT"P!=";P%
1040 GOTO 1010
1050 END
1100 DEF FN "FACTORIEL" (X%,Y%/I%)
1110 IF X%=1 THEN Y%=1:GOTO 1150
1130 I%=X%-1: CALL FN "FACTORIEL" (I%,Y%)
```

```
1140 Y%=Y%*X%
1150 END FN
```

On notera en outre, à propos des sous-programmes, les deux extensions suivantes de l'ordre LIST:

--> LIST FN liste les noms de tous les sous-programmes contenus dans un programme donné

--> LIST FN "nom de subroutine" liste le sous-programme correspondant au nom indiqué.

!cj
LES UTILITAIRES
!lj

AZERTY (M): ce masque vous montre les touches à inverser pour passer du clavier américain au clavier fran&ais (QWERTY --> AZERTY). Voir la commande LET"Z" pour plus de détails.

BINARY (M): ce masque indique la longueur des modules binaires contenus sur la disquette Master. Cela vous permettra de les recopier sans problèmes. Exemple:

```
HIMEM : 16 * 256
LOAD"$1000,$1300,1:BOOT1"
SAVE"$1000,$1300,0:BOOT2"
```

AUTO COPIE: programme qui recopie d'un drive sur un autre tous les masques, les globaux et les programmes. Le drive d'arrivée n'est pas détruit. S'il s'agit d'une disquette vierge, il faut la formatter auparavant par l'ordre LET"£F,<No drive>". IMPORTANT: les drives peuvent &tre de types différents (5', 8', disque dur)

AUTO LIST: programme qui édite sur imprimante tous les masques et les programmes contenus sur une disquette.

AUTOSTART: programme qui simule les fonctions clavier de la ROM autostart dans le cas o& votre micro-ordinateur n'en est pas équippé.

```
CTRL/S:  bloque un list
 ESC/I:  déplacement du curseur vers le haut
 ESC/J:  déplacement du curseur vers la gauche
 ESC/K:  déplacement du curseur vers la droite
 ESC/L:  déplacement du curseur vers le bas
```

BOOT: programme qui formatte un disque en laissant la piste 0 pour le BOOT. Si celui-ci est sur la disquette, il sera recopié en piste 0. L'utilisation classique est la suivante: on place la disquette MASTER en drive 1 et on lance l'exécution du programme en faisant RUN"BOOT". Si on a un deuxième lecteur, on place la disquette à formatter dans le drive 2: sinon, on retire la MASTER pour mettre l'autre à sa place. Tout ceci, bien s&r, dès que le chargement en mémoire centrale du programme est terminé.

CHECK ROM: ce programme affiche les valeurs trouvées et les valeurs réelles des 8 ROM du DOS pour contr&ler leur validité.

COPIE: programme qui copie intégralement un drive sur un autre de m&me type. Si le disque d'arrivée a déjà été formatté (soit directement par

LET"£F,n", soit parce que c'est une copie dont on ne se sert plus), il faut répondre "N" à la question sur le formattage. Ce programme sert notamment pour les sécurités.

COPIE SI: copie un fichier séquentiel indexé d'un disque sur un autre, avec possibilité de changer le nom. Il est possible également de "dupliquer" un fichier, c'est à dire de le recopier sur la disquette de départ, à condition que la copie n'ait pas le m&me nom que l'original. Si le fichier départ comporte des clés homonymes, remplacer dans le programme le WRITE de la ligne 210 par ADD.

DEMO: Exemple de programme gérant un fichier en accès par clé.

DEBUG.MENU: Cet utilitaire permet d'appeler différent programmes de contr&le de l'état d'un disque, qui figurent tous sur la disquette Master sous des noms commencant par DEBUG. Ces programmes sont utilisables avec tous les types de disques ou de disquettes. Nous vous conseillons de les utiliser le plus souvent possible et de les intégrer à vos logiciels d'application. Cela vous évitera bien des désagréments au cas o& une erreur figurerait sur le disque. Les options disponibles sont les suivantes:

1/ ANAL PISTE: Cette option permet de s'assurer que plusieurs fichiers n'occupent pas la m&me place sur le disque. Il donne la liste des pistes détruites, puis le catalogue des pistes logiques allouées à chacun des fichiers (rappelons au passage que le catalogue du disque est lui-m&me un fichier; la place occupée par les masques et les programmes est donc comprise dans celle du catalogue). Enfin le programme donne la liste des fichiers par piste et signale les erreurs qui ont éventuellement été détectées.

2/ SUPER CAT: Ce programme donne un catalogue des fichiers et exerce un contr&le des pistes. Il indique, pour chaque fichier, la valeur de son coefficient de blocage, son nombre de clés et le nombre de piste qu'il occupe. Les erreurs détectées sont signalées.

3/ VERIFY: Cette option permet de vérifier la cohérence de la DCB (data control block) d'un fichier. Lorsqu'un fichier est mis à jour, il faut réecrire sur le disque sa DCB modifiée. Cet utilitaire permet de s'assurer que l'endroit o& vont &tre réecrites les informations est bien celui réservé au fichier. Il pourrait y avoir, en cas d'erreur, des répercutions sur l'ensemble du disque. N'utilisez plus un fichier dont la DCB comporterait une incohérence!

4/ INFORMATION: Ce module donne des informations concernant les options du programme DEBUG.MENU.

5/ DESTRUCTION: Ce programme permet de détruire un fichier endommagé. Nous attirons votre attention sur le fait qu'il ne faut JAMAIS DETRUIRE UN OBJET ENDOMMAGE PAR L'ORDRE LET"£D...

FILE COPY: ce programme copie un fichier d'un disque sur un autre, après affichage du catalogue des fichiers. Si le fichier de départ comporte des clés homonymes, remplacer le WRITE de la ligne 210 par ADD. Si le fichier est multiclés, le programme ne fonctionne que si toutes les clés sont répétées dans l'enregistrement.

HELLO: ce programme est celui qui est automatiquement exécuté lors de la mise en route. Vous pouvez le remplacer par un autre de votre choix. Si vous désirez exécuter un de vos logiciels dès la mise en route du système, il suffit de rajouter au programme HELLO l'instruction RUN"nom du programme à exécuter"

INTERRO: programme de mise à jour d'un fichier quelconque. Ce programme utilise deux masques provisoires (non enregistrés sur disque) qui ont pour fonctions respectives:

- saisie de la clé de recherche et de l'opération désirée (lecture, modification, écriture, lecture du suivant, destruction)

- description de l'article en affichage aussi bien qu'en saisie.

Dans le cas de tableaux, ceux-ci apparaissent en une ligne: les différents indices sont séparés par ";"

ATTENTION: si le fichier est multiclé, le programme ne fonctionne que si les clés sont reprises dans l'enregistrement.

si le fichier comporte des clés homonymes, il faut remplacer le WRITE de la ligne 210 par ADD.

N.B.: le programme exécute les ordres selon la syntaxe normale: si on veut, par exemple, lire un fichier séquentiellement, il n'y a pas besoin de donner une valeur à la clé: il suffit d'indiquer S (Suivant) pour l'ordre à exécuter.

MAJ PAGE3: programme de mise à jour des paramètres de la page 3 (voir le chapitre sur le paramétrage du M/DOS 6502). Le programme BOOT est modifié en conséquence.

RENUMEROTE: programme qui renumérote les lignes d'un programme BASIC en fonction de certaines valeurs: il s'utilise comme suit.

faire RUN"RENUMEROTE" -- taper RETURN à l'endroit demandé. Quand l'écran est redevenu blanc, chargez le programme à modifier (LOAD"nom du programme"). Tapez ensuite &-RETURN si vous gardez les valeurs par défaut (renumérotation de 10 en 10 à partir de la ligne 10). Sinon, écrivez après le signe "&" les paramètres que vous voulez modifier, lesquels sont au nombre de 4:

F: point de départ de la renumérotation (défaut=10)

I: pas de l'incrémentation (défaut=10)

S: ligne du programme à partir de laquelle commence la renumérotation (défaut=10)

E: ligne du programme jusqu'o& doit aller la renumérotation (défaut=63999)

Exemple: &F300,I5,S200,E700 renumérote la partie de programme comprise entre 200 et 700 de 5 en 5 à partir de 300 (300,305,310,etc...)

Bien entendu, la renumérotation agit aussi sur les GOTO et les GOSUB.

RWTS 3.2 et RWTS 3.3: programmes qui permettent de faire résider simultanément en mémoire les deux versions 110 K et 140 K pour effectuer des conversions entre les deux.

Exemple: vous avez démarré sur une version 140 K (3.3). Vous désirez lire et recopier un programme TEST écrit en M/DOS 6502 version 110 K. Faites RUN"RWTS 3.2". Le programme affiche l'état des lecteurs actuels: si vous avez deux lecteurs en slot 6 avec interface 140 K (drives 0 et 1) et deux en slot 5 avec interface 110 K (drives 2 et 3), le programme indique les valeurs par défaut:

```
0.....140 K
1.....140 K
2.....140 K
3.....140 K
4.....non connecté
5.....non connecté
```

Le programme demande le drive à passer en 110 K: répondre 2. Il suffit ensuite, après avoir repris la main, de faire:

```
LOAD "2:TEST"
SAVE "0:TEST"
```

SCROLL: ce programme affiche les adresses du SCROLL UP et du SCROLL DOWN. L'adresse du SCROLL DOWN est calculée par:

```
PEEK(116)*256 + PEEK(115)
```

L'adresse du SCROLL UP est fixe:

```
CALL 64624
```

SUPER CONTROLE: ce programme permet de lister des programmes BASIC en rempla&ant tous les caractères de contr&le par des caractères sur fond blanc, ceci pour permettre des corrections eventuelles. L'exécution des programmes se déroule normalement.

Utilisation: taper RUN"SUPER CONTROLE".
Que vous demandiez ou non l'exemple de démonstration, tous les programmes BASIC que vous listerez par la suite verront leurs caractères de contr&le édités sur fond blanc.

UTIL: programme de gestion des masques.

Cet utilitaire permet une mise à jour rapide et simple de tout votre catalogue de masques. Il permet les ordres suivants:

C: création d'un masque. La syntaxe est rigoureusement la m&me qu'en mode direct(cf. chapitre sur les masque, paragraphe II)

M: modification. Le contenu du masque appara&t sur l'écran. vous pouvez le modifier comme en mode création.

D: le masque est affiché pour un dernier contr&le: répondez "0" si vous &tes d'accord pour le détruire.

*: affiche le catalogue des masques

L: ouverture et chargement en mémoire d'un masque. Le masque ainsi chargé pourra servir de base pour la création d'un second masque.

S: crée un masque à partir d'un autre. La commande Save sauve sous le nom choisi par l'utilisateur le masque chargé en mémoire par l'instruction Load.

V: simple affichage d'un masque

F: sortie du programme UTIL

I: imprime un masque sur une imprimante placée en slot 1.

!cj
LES ADRESSES DU M/DOS
!1j

La carte M/DOS est théoriquement compatible avec:

--> Toute carte 80 colonnes

--> Tout terminal connecté en lieu et place de la vidéo et du clavier d'origine ayant l'écran projeté en mémoire

--> Toute mémoire de masse qu'elle peut gérer par tranches allant jusqu'à 16 méga octets.

Il faudra toutefois, pour utiliser ces extensions, implanter des programmes d'interface leur permettant de communiquer avec M/DOS. Ce chapitre décrit certains principes de fonctionnement de l'APPLE sous M/DOS, ainsi que les principaux pointeurs à mettre en place pour utiliser des périphériques particuliers.

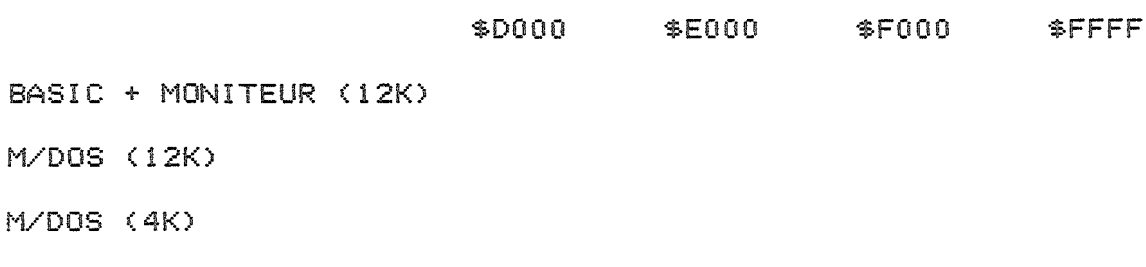
1. UTILISATION DE LA MEMOIRE:

La mémoire de votre APPLE se décompose, sous M/DOS, de la fa&on suivante:

!np

2. COMMUTATION ENTRE BASIC+MONITEUR ET M/DOS:

Le tableau d'utilisation de la mémoire montre que Basic+Moniteur et M/DOS occupent la m&me zone d'une taille de 12K. En effet l'APPLE offre la possibilité de déconnecter les ROM Basic et Moniteur pour les remplacer par d'autres mémoires externes.
La carte M/DOS utilise cette possibilité, et les programmes assembleurs du DOS fonctionnent donc en parallèle avec le Basic et le Moniteur. Seulement, alors que le Basic et le Moniteur occupent ensemble une place mémoire de 12K, le M/DOS en prend 16. Il a donc fallu décomposer ces 16K en deux parties qui se superposent.



Les commutations entre le Basic et les différentes pages du M/DOS 6502 se font par un logiciel intégré dans la carte. Cette opération est transparente pour l'utilisateur qui ne se rendra compte de rien, puisque Basic et M/DOS seront toujours disponibles.

On utilisera la notation suivante:

PAGE 0 : 1ère page du M/DOS, 4K de $D000 à $DFFF

PAGE 1 : 2ème page du M/DOS, 4K de $D000 à $DFFF

RACINE : reste du M/DOS (8K) de $E000 à $FFFF

BASIC : 12K de $D000 à $FFFF (comprend aussi le Moniteur)

Les commutations se font en écrivant à l'adresse $CXFF, o& X est le numéro du slot sur lequel est connectée la carte M/DOS:

```
LDA £ $80
STA $CXFF  ........ BASIC

LDA £ $00
STA $CXFF  ........ PAGE 0 + RACINE

LDA £ $01
STA $CXFF  ........ PAGE 1 + RACINE
```

!np

3. RESET ET INTERRUPTION:

La mise en route ou le RESET se font dans les ROM de la carte M/DOS, qui contiennent un programme de gestion des interruptions.

RESET: Le programme ramène au RESET de la ROM Moniteur de l'APPLE. La présence de la carte est donc transparente.

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.

ATTENTION: Lorsque la carte est sélectée, ce sont les vecteurs d'interruption du M/DOS qui servent. En revanche, quand elle est désélectée (le Basic est en mémoire), ce sont les vecteurs de la ROM Moniteur qui sont utilisés. Cependant, au moment d'une interruption autre que RESET, l'état de la mémoire reste inchangé et vous pouvez aussi bien &tre en mode Basic qu'en mode M/DOS. Si vous désirez, dans vos programmes d'interruptions, utiliser des fonctions Moniteur, vous devez:

--> chercher sous quel mode (M/DOS ou Basic) vous &tes. Pour cela, lisez un octet dans la page $DXXX à un endroit o& les trois occupations possibles sont dférentes.

--> commuter vers Basic par:

```
LDA $80
STA $CXFF
```

--> avant de terminer l'interruption, reconnectez, si besoin est, la page précédente par:

```
LDA $00  ou  $01
STA $CXFF
```

4. L'OCCUPATION RAM DU M/DOS:

Pour fonctionner, le M/DOS utilise diverses zones RAM:

1/ $0300-$03CF

Cette zone en page 3 ne doit pas &tre modifiée en cours de travail. Elle contient notamment les pointeurs définissant la position des buffers du DOS.
!np
2/ $0250-$02FF

Cette zone en page 2 est composée de variables de travail du M/DOS. Elle peut &tre détruite entre deux ordres. Elle sert d'ailleurs de buffer pour INPUT.

3/ LES BUFFERS DU DOS:

Ils contiennent des descriptifs de disques, de fichiers, les masques... Ils commencent là o& finit le Basic. Normalement le pointeur de fin de programme Basic et celui de début des buffers du DOS sont identiques. Ils est toutefois possible de laisser une zone vide entre la fin du Basic et le début des buffers. Les valeurs de ces pointeurs peuvent &tre modifiées. Elles sont stockés dans la zone $0300-$03CF en page 3. L'utilisateur consultera, pour plus de détails, le paragraphe sur la position des buffers du DOS.

4/ LES PROGRAMMES D'ACCES AUX PERIPHERIQUES:

Cette zone mémoire contient les programmes de communication avec les disques (lecture, écriture, formattage) baptisés d'HANDLERS. Il s'agit, dans le cas des lecteurs 5 pouces, de l'utilitaire RWTS figurant sur la disquette MASTER.
Le placement en mémoire des HANDLERS peut &tre défini par l'utilisateur (cf paragraphe sur les programmes d'accès aux périphériques).

5. LA STRUCTURE DES OBJETS EN MEMOIRE CENTRALE

Il existe 3 types d'objets contenus en mémoire:

- Les fichiers: type "F"
- Les masques : type "M"
- Les descriptions de disque: type "$"

Chaque objet est rangé sous la forme:

numéro logique

type

longueur poids faible

longueur poids fort

objet par lui-m&me

!np
Les objets sont placés de fa&on consécutive en mémoire. Pour retrouver un objet donné, on utilise la méthode suivante:

- accès au premier élément

- comparaison avec le numéro recherché

Si c'est le m&me | sinon

-- fin -- | On ajoute à l'adresse de départ la longueur totale qui permet d'accéder à l'élément suivant. Ensuite, retour au stade 2.

Deux pointeurs permettent de conna&tre la limite des objets:

- Les pointeurs de début des modules (\$300,\$301) + 292 (\$124)

- Les pointeurs du premier libre

Début des modules

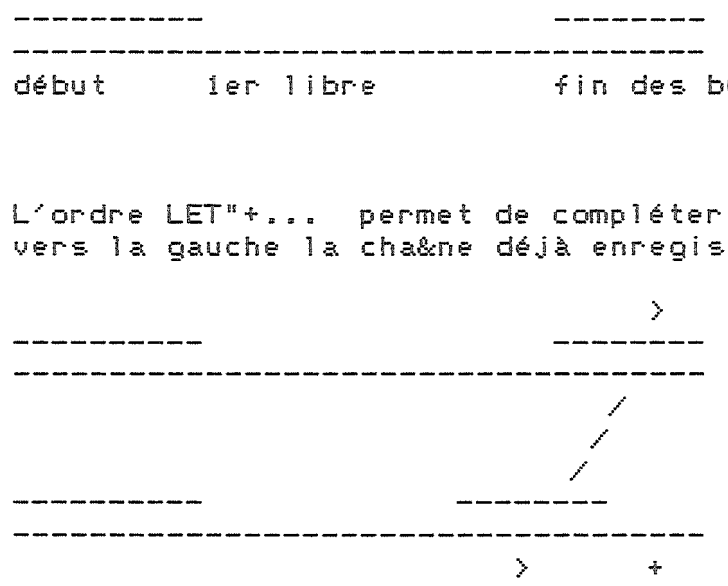
Le premier module commence à l'adresse:
PEEK(768)x256 + PEEK(769) + 1028

Destruction des objets: l'instruction LET"£C-<ul>" permet de récupérer la place d'un objet. Les objets suivants sont alors décalés de la longueur de l'objet détruit et le premier libre est diminué de la m&me valeur.
!np
Double utilisation des buffers: la place disponible dans les buffers entre le pointeur du premier libre et la fin des buffers est également utilisée. Elle sert:

- à enregistrer les descriptions de fichiers

- à enregistrer des ordres pour la fonction EXECUTE

Par exemple, l'ordre LET ">... va transférer la cha&ne de caractères suivant le ">" dans cet espace. Le cadrage se fait à droite.



L'ordre LET"+... permet de compléter cette cha&ne. Pour cela, on décale vers la gauche la cha&ne déjà enregistrée, puis on insère la nouvelle.

Trois ordres permettent d'inscrire des caractères dans ce buffer:

```
LET">...
LET"+...
LET"ENTER-<u.l>"
```

Ce dernier décode le dictionnaire d'un fichier déjà enregistré et le place dans le buffer.

Si la place disponible entre le premier libre et la fin des buffers est insuffisante, le message ?OUT OF MEMORY ERROR est affiché à l'écran.
!np
6. LA POSITION DES BUFFERS DU DOS:

Les pointeurs de positionnement des buffers du DOS sont les suivants (ils sont toujours dans l'ordre poids fort, poids faible):

```
$300-$301 =début des buffers du DOS
$303-$304 =fin de la première partie
$305-$306 =pointeur courant, qui indique le début de la place non
           encore occupée dans les buffers.
$307-$308 =fin des buffers du DOS
```

Pour positionner les buffers du DOS, choisissez les adresses mini et maxi que vous leur réservez et placez les en:

```
          $300-$301
          $307-$308
Placez en $303-$304 les m&mes valeurs qu'en $307-$308
Placez en $305-$306 l'adresse correspondant à celle indiquée en
          $300-$301 plus $124
```

```
exemple: $300: 99
         $301: 00    $9000
         $302:
         $303: B6
         $304: 00    $B600
         $305: 91
         $306: 24    $9124=$9000+$124
```

$307: B6
$308: 00 $B600

Dans cet exemple, la zone mémoire réservée aux buffers du DOS s'étend de $9000 à $B600 inclus.

Il ne faudra pas oublier de modifier aussi, dans le cas d'une augmentation de l'espace réservé aux buffers, le pointeur de fin du BASIC situé en:

115 (décimal):fin du Basic poids faible
116 (décimal):fin du Basic poids fort

Pour que les modifications soient prises en compte, il faut faire CLEAR après avoir changé la valeur du pointeur de fin du Basic et vider les buffers du DOS (LET"£C,$) après la modification des pointeurs les positionnant.
!np
7. LA DEFINITION DES PERIPHERIQUES:

Le M/DOS 6502 permet de gérer simultanément des périphériques de types différents. Des tables permmettent de décrire ces périphériques. Pour chacun d'entre eux, elles indiquent:

--> Le nombre de pistes
--> Le nombre de t&tes
--> Le nombre de secteurs
--> L'adresse du programme permettant d'y accéder (HANDLER)

Il est possible d'utiliser dans une m&me configuration sous M/DOS jusqu'à 6 périphériques.

Pour chaque périphérique (numéroté de 0 à 5) dont nous appellerons n le numéro,indiquez:

en $310+n	le nombre de pistes - 1
en $316+n	le nombre de t&tes
en $31c+n	le nombre de secteurs
en $322+2n-$323+2n	l'adresse (poids fort, poids faible) du HANDLER

Si les adresses correspondant au HANDLER sont remplacée par $00,$00, le périphérique sera considéré comme inexistant.

8. LES PROGRAMMES D'ACCES AUX PERIPHERIQUES:

Le handler doit &tre capable de communiquer (lecture, écriture, formattage) avec un disque d'après les paramètres suivants:

$309 :numéro du périphérique. il permet de différencier des périphériques pour lesquels l'adresse du HANDLER est la m&me

$30A :piste choisie

$30B :t&te choisie

$30C :secteur choisi

$30D :commande: 1 pour lecture
2 pour écriture
3 pour formattage

$30E :poids fort de l'adresse du bloc

$30F :poids faible de l'adresse du bloc
!np
Il n'est pas necessaire d'effectuer des contr&les de validité, ceux-ci sont faits par le M/DOS avant de faire appel & 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.

ATTENTION: Il faudra veiller à ce qu'après le formattage, tous les secteurs puissent &tre lus, m&me s'ils n'ont pas été remplis. En version 110K la lecture est impossible s'il n'y a pas eu une écriture préalable. C'est pourquoi, comme vous pourrez le constster, le formattage, dans cette version, sera suivi d'une écriture sur tous les secteurs de la disquette. C'est également le cas de certains autres types de disques tels que le CII D140 par exemple.
!np
!np
9. DESCRIPTION DE LA GESTION D'ECRAN:

La carte M/DOS est prévue pour pouvoir s'adapter à n'importe quel type de carte 80 colonnes ou de terminal intelligent. Pour cela, un certain nombre de paramètres décrivant l'écran sont à mettre en place.

$368-$369 adresse de l'écran (poids fort, poids faible)

$36A-$36B longueur de l'écran (poids fort, poids faible)

$36C nombre de caractères par ligne

Il faudra aussi écrire des programmes assembleurs réalisant un certain nombre de fonctions de gestion d'écran. Ces fonctions utilisent des pointeurs en page 0 notés:

VV ($DB,$DC)
WW ($DD,$DE)

VV indique la position du curseur (poids faible, poids fort)
WW indique le nombre de caractères (poids faible, poids fort) séparant le curseur du premier caractère de l'écran. WW évolue donc entre 0 et la longueur de l'écran -1.

La position du curseur peut &tre 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, M/DOS initialisera VV à cette valeur et WW à zéro.

M/DOS doit pouvoir se déplacer sur cet écran qui lui est inconnu, lire ou écrire dessus, convertir des caractères ASCII en code écran et inversement (on appelle CODE ECRAN la valeur qui permet d'écrire sur l'écran un caractère dont on conna&t le code ASCII). Il faudra donc implanter les programmes suivants:

AVANCE: calcule les nouvelles valeurs de VV et WW lorsqu'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 o& WW=0 en entrant dans le programme RECULE).

INCAR : renvoie dans l'accumulateur le code écran du caractère situé en VV.

OUTCAR: écrit, à l'endroit du curseur pointé par VV, le caractère dont le code écran est dans l'accumulateur.

!np

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, on pourra, par exemple, la remplacer par INVERS.

Il faudra indiquer au M/DOS les adresses des programmes réalisant ces fonctions. Pour cela, remplissez l'ensemble des JMP prévues à cet effet:

```
$36F    JMP AVANCE
$372    JMP RECULE
$384    JMP INCAR
$387    JMP OUTCAR
$378    JMP E A
$375    JMP AE
$37E    JMP INVERS
$381    JMP FLASH
```

Pour simplifier votre programmation, sachez que, lorsque le M/DOS se branche à l'INCAR ou à l'OUTCAR, le registre y est nul. Si VV est l'adresse réelle du caractère, on pourra donc choisir:

```
INCAR:  LDA (VV),y
        RTS

OUTCAR: STA (VV),y
        RTS
```

que l'on pourra placer directement à la place des JMP correspondant pour accélérer car, dans les deux cas, la longueur est de 3 octets.

Pour gagner du temps, le M/DOS ne reconnecte pas le BASIC lors de l'appel de ces fonctions. Vous ne pouvez donc pas utiliser des fonctions Moniteurs, ni tester vos programmes en utilisant l'instruction BRK (code $00).

Il faudra essayer 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:

--> 2000 appels à AVANCE
--> 2000 appels à RECULE
--> 2000 appels à INCAR

--> 2000 appels à OUTCAR

Vous trouverez, ci joint, la version 40 colonnes de base contenue dans la ROM:

```
!np
!np
!np
!np
```

10. MODIFICATION 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):

```
      $37B   JMP GET
```

Le programme doit renvoyer dans l'accumulateur ou le code ASCII, ou celui-ci + 128.

La fonction AZERTY pourra &tre utilisée quelque soit la fonction GE$T que vous avez créée.

11. AJOUT DE NOUVELLES COMMANDES AU DOS:

Ils est possible d'ajouter de nouvelles fonctions au M/DOS. Il existe, pour cela, une indirection sur deux octets ($38A poids faible, $380 poids fort) indiquant l'adresse du programme analysant les commandes. Pour ajouter une commande, il faudra donc procéder de la fa&on suivante:

1/ sauvez l'adresse précédemment placée dans cette indirection.

2/ mettez y l'adresse de votre programme.

3/ à la fin de l'exécution de votre programme d'analyse de commande, dans le cas o& la commande n'est pas pour lui, retournez à l'indirection précédente si celle-ci existait, sinon faites RTS. Si, au contraire, la commande concerne votre programme, exécutez la et faites:

```
-pla
-pla
-RTS
```

Cette méthode permet de faire fonctionner ensemble plusieurs programmes ajoutant de nouvelles commandes.

Dans la version de base, l'indirection contient 0,0 . Dans ce cas, le M/DOS ignore l'indirection.

Lorsque votre programme d'analyse prend la main:

--> l'accumulateur contient le premier caractère de la commande.

--> AA ($272) contient la longueur de l'instruction.

--> yy ($E1,$E2) contient l'adresse de l'instruction.

--> le registre y pointe vers le caractère en cours dans l'instruction par rapport à yy.

Les fonctions suivantes peuvent vous &tre utiles:

!np
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.

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 saisies sous la
forme $XXXX. En sortie X=poids faible, A=poids fort.

exemple: soit la commande: LET"H,valeur,une lettre,hexadécimal"
LET"H,12,A,$25F0"
!np
!np

RECAPITULATIF SYNTAXIQUE

Ce récapitulatif donne de fa&on synthétique la syntaxe des ordres M/DOS et des fonctions qui s'y rapportent. Les ordres sont classés suivant l'ordre des chapitres de la note technique, et, à l'intérieur des chapitres, par rubrique.

Pour mieux condenser les informations et en rendre la lecture plus aisée, nous avons adopté les conventions suivantes:

- Tout ce qui n'est pas entre crochets ou entre parenthèses doit &tre reproduit tel quel, sauf les ordres, pour lesquels la première lettre suffit.

- Les paramètres obligatoires sont placés entre < >.

- Les paramètres facultatifs sont placés entre ( ).

- Les différentes options sont séparées par un /.

<u> : No d'unité logique
<d> : No de drive: par défaut, le système prend le No du dernier drive manipulé.
<c> : caractère alphabétique quelconque.
<np>: nom de programme BASIC.
<nb>: nom de module binaire.
<nf>: nom de fichier.
<nm>: nom de masque.
<ng>: nom de masque global.
<lv>: liste de noms de variables.
<cl>: liste des noms des variables qui composent la clé.
<lp>: liste de paramètres

Quelques exemples:

LET"OUTPUT,<u>,(O/0)" --> 3 options possibles: o, 0 ou rien (option facultative)

LET"£DELETE,<P/B/F/M/G>,(d),<nn> --> l'option est obligatoire et doit &tre une des 5 lettres proposées. Le nom dépend du type de l'objet à détruire.

CHAPITRE IV: INSTRUCTIONS GENERALES

LET"£FORMAT,(d)"	formatte le disque du drive No d.
LET"*,(d)"	catalogue de tous les objets du drive No d.
LET"*,(d),(B/F/M/G/P)	catalogue des: Binaires/Fichiers/Masques/ Globaux/Programmes du drive No d.

LET"£REORG-$(d)"	réorganise le catalogue du drive No d.
LET"%(d)"	stocke dans l'adresse 189 le nombre de pistes libres du drive no d (cf. 3.2).
SAVE"(d),<np>"	sauve le programme "np" sur le drive d.
SAVE"§(d),<np>"	sauve une nouvelle version de "np".
SAVE"$XXXX,$YYYY,(d),<nb>"	sauve sous le nom de "nb" la partie de la mémoire comprise entre les adresses hexadécimales XXXX et YYYY.
LOAD"(d),<np>"	charge un programme stocké sur le drive d.
LOAD"£(d),<np>"	charge "np" à la suite d'un programme.
LOAD"$XXXX,$YYYY,(d),<nb>"	charge un module binaire à partir de $XXXX, $YYYY étant une adresse à ne pas dépasser.
LOAD"/(d),<np>"	charge et exécute "np" sans réinitialiser les variables.
RUN	exécute le programme chargé en M.C.
RUN"(d),<np>"	charge et exécute le programme "np"
RUN"£(d),<np>"	charge "np" à la suite d'un programme et exécute l'ensemble.
LET"£DELETE,<t>,(d),<nn>	détruit sur le drive d l'objet de type t (B/M/G/F/P) et de nom "nn"
LET"£C,"+CHR$(d)	ferme le drive d.
LET"£C,$$"	ferme tous les drives.
LET">M"	emp&che la fermeture automatique des drives en mode direct.
LET")"	rétablit la fermeture automatique des drives
LET"Z"	passage du clavier QWERTY au clavier AZERTY
LET"G"	émission d'un signal sonore.
LET"Y"	inversion vidéo de tout l'écran.
LET"CLEAR,$"	fermeture de toutes les unités logiques.
LET"CLEAR,<u>"	fermeture de l'unité logique u.

CHAPITRE V: LES FICHIERS

ORDRES COMMUNS A TOUS LES TYPES DE FICHIERS

LET"£NEW,<u>,F,(d),<nf>	création du fichier "nf" sur le drive d.
LET"ENTER,<u>"	édition en clair du dictionnaire
LET"VISUALISE"	affichage du dictionnaire édité par "ENTER".
LET"£OPEN,<u>,F,(d),<nf>	ouverture d'un fichier en unité logique u.
LET"£DELETE,<u>,F,(d),<nf>	destruction d'un fichier sur le drive d.
LET"£>n"	changement du coeff. de blocage (cf. 3.3.1)

FICHIERS SEQUENTIELS RELATIFS

LET">§<p>% = <lv>"	définition du dictionnaire: p est le pointeur. <lv> est une liste de variables.
LET"NEXT,<u>"	lecture séquentielle.
LET"READ,<u>"	lecture directe.
LET"WRITE,<u>"	écriture d'un nouvel enregistrement.
LET"DELETE,<u>"	destruction d'un enregistrement.
LET"UPDATE,<u>"	modification (ou réécriture).
LET"XINDEX",<u>"	calcul du No du pointeur du prochain enregistrement qui sera créé par "WRITE"

FICHIERS SEQUENTIELS INDEXES MONO- ET MULTICLES

LET"> <cl> = <lv>"	création du dictionnaire d'un fichier mono-

	clé. <cl> peut comporter plusieurs variables
LET"> <cl1>&..<cln> = <lv>"	création du dictionnaire d'un fichier multi-clés. Le nombre maximum de clés est 10.
LET"£REORGANIZE,<u>"	reclassement physique de la table des clés.

Dans les ordres qui vont suivre, <ma> représente le No du moyen d'accès. Il ne sert que pour les fichiers multiclés. Sa valeur par défaut est 1.

LET"NEXT,<u>,(ma)"	lecture séquentielle
LET"READ,<u>,(ma)"	lecture directe
LET"WRITE,<u>"	écriture sans clés homonymes.
LET"ADD,<u>"	écriture avec clés homonymes.
LET"DELETE,<u>,(ma)"	destruction d'un enregistrement.
LET"UPDATE,<u>,(ma)"	modification d(un enregistrement.
LET"XTRACT,<u>,<n>,(ma)"	extraction avec tri sur la nème variable du moyen d'accès choisi.
LET"BORNE,<u>,(ma)"	fixe un butoir pour la lecture séquentielle.

CHAPITRE VI: LES MASQUES

CREATION DE MASQUES

LET"£NEW,<u>,M,(d),<nm>	création du masque "nm" sur le drive d.
LET"£NEW,<u>,M,/(d),<nm>	création directe en mode programme.
LET"£NEW,<u>,M,§(d),<nm>	création sous un nom de masque déjà utilisé.
LET"£IMAGE,<u>,M,<nm>"	création d'un masque provisoire.
LET"£NEW,<u>,G,(d),<ng>	création d'un masque global.

FONCTIONS D'ECRAN EN MODE CREATION DE MASQUE

-->	déplacement vers la droite
<--	déplacement vers la gauche
RETURN ou CRTL/Z	déplacement vers le bas
CRTL/R ou CRTL/Q	déplacement vers le haut
CRTL/T	retour en haut à gauche de l'écran
CRTL/§	effacement de l'écran
CRTL/D	détruit le caractère qui précède le curseur
CRTL/I	insère un caractère après le curseur
CRTL/V	reproduit verticalement un caractère.
CRTL/A	abandon en cours de saisie
ESC	validation de la création
CTRL/B	caractères sur fond blanc
CRTL/F	caractères clignotants
CRTL/N	retour aux caractères normaux

TYPES DE VARIABLES AUTORISES

XX	flottant (réel)
XX+	flottant positif
XX§	flottant entier
XX*	flottant format gestion
XX§*	flottant entier cadré à droite
XX%	entier (-32768 < XX% < +32768)
XX%/	binaire (0 <= XX%/ < 256)
XX%+	entier positif
XX%*	entier cadré à droite
XX$	cha&ne de caractères

!cj
****
**
** INDEX RECAPITULATIF
**
****
!lj

!cj
LES MESSAGES ERREUR DU M/DOS

!1j

I/ Les erreurs recouvrables:

Elles ne provoquent pas d'interruption des programmes et sont indiquées par une valeur non nulle de la variable STATUS (PEEK(189)).

Valeurs possibles des codes erreur:

1 :Erreur sur un masque. Ce code traduit notamment un abandon: la sortie du masque, en création ou en saisie, s'est faite, non pas en appuyant sur la touche ESC, mais en tapant CTRL/A.

10 :Inexistant. L'objet ou l'enregistrement recherché n'a pas été trouvé.

20 :Priorité trop faible. Votre carte ne vous permet pas de lire l'information demandée.

30 :Existant déjà. L'écriture demandée ne peut &tre effectuée (homonymie de clés avec LET"WRITE, de nom avec SAVE....).

255 :Fin de fichier.

II/ Les erreurs non recouvrables:

Elles provoquent un arr&t du programme. Les différents messages suivants peuvent s'afficher sur l'écran:

SYNTAX ERROR: La syntaxe d'un ordre est erronée.

ILLEGAL QUANTITY ERROR: erreur sur un numéro logique (affectation d'un numéro déj& utilisé, pas d'objet ouvert sous cette unité logique.....)

BAD SUBSCRIPT ERROR: Tentative de lecture d'un tableau dans un enregistrement avec un dimensionnement inférieur & celui utilisé lors de l'écriture.

OUT OF MEMORY ERROR: Plus de place dans les buffers du DOS. Fermez les fichiers et les masques inutiles. Si la place ainsi libérée est insuffisante, il faut augmenter l'espace mémoire réservé aux buffers du DOS (voir pour cela le chapitre sur les adresses du M/DOS).

TA ERROR (Track Allocation Error): Le disque est plein.

DATA ERROR: Erreur de lecture ou d'écriture sur le disque. Ce message peut signaler un mauvais fonctionnement du support magnétique (le disque est peut-&tre endommagé, ou simplement mal positionné dans le lecteur). Il peut aussi provenir d'une protection d'écriture du support.

DIRECT ERROR: Ce message n'est affiché qu'en mode direct. Il signale une valeur non nulle de la variable STATUS.

FUNCTION ERROR: Erreur, détectée dans un contr&le, d&e à une incohérence des paramètres dans un accès disque.

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