COMPILATEUR BASIC

MEM / DOS

Réalisé par Georges MALLET
MEMSOFT S.A.
NICE - FRANCE

pour APPLE II et IIe

COMPILATEUR BASIC

MEM/DOS

I - INTRODUCTION

II - COMPILATEUR BASIC

III - ENVIRONNEMENT LOGICIEL

RECOMMANDATIONS IMPORTANTES

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

De même, il doit vérifier auprès de son revendeur, la compatibilité du produit avec le matériel et les logiciels qu'il possède déjà ou qu'il compte acquérir.

Le client ou l'utilisateur doit satisfaire au respect absolu des conditions et précautions d'utilisation, notamment :

- Usage normal.

- Locaux présentant les garanties techniques, pour éviter toute détérioration des programmes.

- Absence totale de toute intervention logique ou physique, de toute modification ou tentative de modification des programmes.

- Absence de variation ou défaillance du courant électrique.

- Copie-sauvegarde journalière de l'ensemble des fichiers dès qu'une modification leur est apportée et contrôle, afin de s'assurer de la conformité de l'opération de copie.

- Conservation hors des locaux d'une copie du logiciel et des fichiers de données ainsi que leur renouvellement périodique afin de pouvoir redémarrer le système en cas de destruction de l'ensemble des copies (incendies, innondation, etc...).

- Conservation des logiciels et copies dans un emplacement sec, à température normale (16 à 25 degrés Celsius), hors de tout champ magnétique et électrostatique.

COMPILATEUR BASIC MEM / DOS

I - INTRODUCTION

I - 1 - Contenu du package

Le package fourni comprend, outre le compilateur lui-même, des outils de documentation et de contrôle offrant au professionnel les moyens de fonctionnement de son atelier logiciel.

Ces outils, qui vous sont proposés dans le menu de départ, vous permettent :

- l'édition de ''listings'' indentés de vos programmes,

- un contrôle syntaxique,

- une référence croisée des variables utilisées et des lignes où elles apparaissent,

- des statistiques d'utilisation des mots clés du langage BASIC et des opérateurs arithmétiques vous permettant d'évaluer la structuration de vos logiciels,

- un comparateur de programmes donnant les différences entre deux versions successives.

Commencez par faire une copie de votre disquette compilateur, afin de parer à toute erreur de manipulation.

La présente documentation vous indiquera comment utiliser le compilateur et les outils additionnels. Elle vous aidera également à optimiser ces programmes. Enfin, elle vous expliquera les différences de possibilités et de performances entre interpréteur et compilateur.

Un glossaire placé en fin de manuel vous aidera (nous l'espérons) à mieux comprendre le texte.

I - 2 - But du compilateur

Le role d'un compilateur est double :

1 - Accélérer la vitesse d'exécution.

2 - ''Cacher'' le source du programme donc éviter les modifications non contrôlées de vos logiciels.

Notez que l'accélération de vitesse obtenue dépend de la façon dont vous programmerez.

Pour celà, vous trouverez plus loin (chapitre II - 10 -) des conseils de programmation vous permettant de tirer le meilleur parti possible de votre compilateur MEM/DOS.

Si vous avez des problèmes de place mémoire, consultez le chapitre II - 10 - qui vous aidera à les résoudre.

I - 3 - Mise en place

Votre compilateur MEM/DOS est prévu pour fonctionner sur un APPLE II ou un APPLE IIe équipé de deux lecteurs de disquettes APPLE et d'une carte MEM/DOS 20 K version programmation.

Il comprend :

- Une disquette ''compilateur''.
- Une disquette ''outils''.
- Une carte électronique.

La mise en place est très simple :

1/ Mettez votre APPLE hors tension.

2/ Ouvrez le capot.

3/ Vérifiez la présence de la carte MEM/DOS 20 K.

4/ Insérez votre carte compilateur dans l'un des slots libres 1 à 5.

5/ Vérifiez que les deux lecteurs de disquettes sont connectés à la même carte contrôleur en slot 6.

6/ Refermez votre APPLE.

Votre compilateur est prêt à fonctionner.

IMPORTANT :

! LA CARTE NE SERT QUE LORS DE LA COMPILATION, LES PROGRAMMES OBTENUS PEUVENT FONCTIONNER AVEC UN ENVIRONNEMENT MEM/DOS NORMAL.

La compilation ne peut se faire que dans la version de base du MEM/DOS (c'est-à-dire : version avec deux lecteurs de disquettes en slot 6 et carte MEM/DOS).

Par contre, les programmes obtenus peuvent fonctionner dans toute autre configuration :

- disquette grande capacité,
- disquette 8 pouces,
- disque dur,
- multipostes.

Le périphérique utilisé doit être compatible MEM/DOS (c'est-à-dire : être équipé d'un ''handler'' permettant son utilisation avec MEM/DOS).

I - 4 - Principe de fonctionnement du compilateur

Expliquons en quelques mots, le principe de la compilation :

Le but est d'obtenir un programme directement exécutable à partir d'un source MEM/DOS interprété.

Le résultat exécutable sera composé de deux parties :

- une partie fixe commune à tous les programmes compilés appellée BIBLIOTHEQUE,

- une partie variable générée à partir de votre programme.

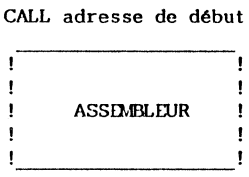
Des détails sur le contenu et l'implantation mémoire de ces modules seront donnés ultérieurement.

Les différentes étapes sont les suivantes :

1/ Transformation du programme source interprétable en fichier MEM/DOS.

2/ Compilation de ce programme en deux passes.

3/ Transformation du fichier résultant en faux programme BASIC de la forme :



Le tout est sauvé sur la disquette sous forme de programme BASIC.

Pour l'utiliser, lisez le chapître II - 5 - concernant l'exécution d'un programme compilé, le seul problème étant de s'assurer de la présence de la bibliothèque.

I - 5 - Les points forts du compilateur MEM / DOS

Le principal point fort du compilateur MEM/DOS est sa structure de données, entièrement compatible avec celle de l'interpréte (détaillée en chapître II - 7 -).

Cela permet :

- de faire fonctionner les masques et fichiers MEM/DOS,
- de compiler SANS MODIFICATION des fonctions & utilisant des variables BASIC,
- d'interfacer facilement des programmes assembleurs.

Les calculs avec des variables ou constantes entières sont faits en binaires sur 16 bits, ce qui accroît de beaucoup la vitesse d'exécution.

La gestion des chaînes de caractères permet un rangement vingt fois plus rapide qu'avec l'interpréteur.

Le code généré est du code cousu (appels nombreux à une bibliothèque) ce qui réduit de beaucoup le foisonnement (rapport de taille de l'objet et du source).

On peut, dans un logiciel, compiler seulement les programmes clés, les programmes interprétés pouvant appeller des programmes compilés et vice versa. L'exécution se fait par ''RUN'' dans tous les cas, donc les appels entre programmes sont inchangés.

I - 6 - Que peut-on attendre d'un compilateur ?

L'accélération de vitesse qui est l'un des avantages d'un compilateur, doit être mesurée avec circonspection.

En effet, le compilateur n'accélérera vos programmes que si l'emploi d'instructions BASIC non MEM/DOS est important.

Si par exemple, votre programme réalise la lecture séquentielle d'un fichier et son affichage à l'écran par masque :

```
100 LET''N,1''
110 IF WS THEN END
120 LET''O,M''
130 GOTO 100
```

il est peu probable que le compilé soit plus rapide car pour les quatre instructions BASIC de la boucle, des lectures disques sont nécessaires, considérablement plus lentes que votre programme, même interprété.

Par contre, si vous effectuez la remise à 0 d'un tableau par:

```
100 FOR I = 0 TO 1000
110 A(I) = 0
120 NEXT
```

vous améliorerez de beaucoup la vitesse en utilisant l'option de compilation en entier (voir chapitre II - 11 -).

Les programmes donnés en exemple dans MEMJEU.MENU vous montrerons les cas qui tirent le plus grand parti de l'accélération de vitesse.

De toute façon, un avantage vous restera toujours acquis en compilant vos programmes : leur protection.

II - LE COMPILATEUR

II - COMPILATEUR BASIC

II - 1 - Géographie de la mémoire

Un programme compilé a besoin d'un ensemble de sous-programmes systèmes.

Ces derniers ont été réunis dans l'objet binaire BIBLIO.OBJ qui se trouve sur la disquette du compilateur.

Ce binaire peut être mis à n'importe quel endroit de l'espace RAM normalement réservé à votre programme BASIC.

Toutefois, il est fortement conseillé de le mettre au début de cet espace.

Le programme que vous allez compiler peut aussi être mis n'importe où dans l'espace RAM réservé à votre programme BASIC. Toutefois, il est recommandé de prendre comme adresse de début du programme compilé celle de fin de BIBLIO.OBJ. Pour vous aider à bien implanter en mémoire vos programmes, sachez que :

- la bibliothèque demande environ 5000 octets,

- il faut prévoir outre la place de la bibliothèque, la place pour :

    - le programme compilé,

    - les chaînes de caractères, qui étant de tailles variables, ne peuvent être intégrées au programme lui-même,

    - les buffers pour les fichiers MEM/DOS.

Une place judicieusement choisie de la bibliothèque (comme celle proposée par défaut) vous évitera de la recharger à l'exécution de chaque programme compilé.

II - 2 - Implantation par défaut

Lorsque vous voudrez compiler un programme, des adresses vous seront proposées pour la bibliothèque et le programme compilé.

Acceptez les telles quelles dans un premier temps. Elles correspondent au chargement de la bibliothèque et du programme compilé au dessus de la place réservée aux handlers 80 colonnes.

Ne modifiez ces valeurs que:

- si vous n'utilisez pas de carte 80 colonnes et avez des problèmes en place mémoire centrale,

- si vous utilisez des programmes systèmes en RAM qui rentrent en conflit avec la bibliothèque ou le code généré,

- si vous utilisez une adaptation 80 colonnes pour MEM/DOS différente de la version MEMSOFT pour la carte 80 colonnes APPLE IIe et implantée ailleurs ou dans un espace mémoire plus important.

II - 3 - Lancement du compilateur

Il est nécessaire pour lancer le compilateur que la carte soit installée. Nous le supposerons. Dans le cas contraire, reportez vous au chapitre I - 3 - ''Mise en place''.

Pour fonctionner, le compilateur nécessitera une disquette de travail initialisée.

Vous pouvez initialiser au préalable une disquette vierge sous MEM/DOS en exécutant la commande :

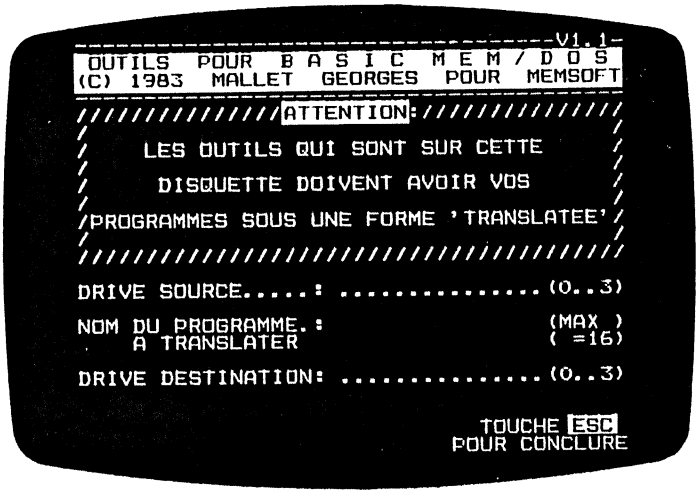
LET'' F, 1''

Cette disquette sera mise dans le second lecteur de disquette.

Le premier devra contenir la disquette compilateur que vous ne devez jamais protéger en écriture car le compilateur y inscrit des informations utiles durant le temps de la compilation. Nous vous conseillons donc de faire une copie de votre disquette compilateur avant de l'utiliser.

Mettez l'APPLE sous tension. Vous verrez alors apparaître un logo (vous pouvez éviter sa visualisation en appuyant sur la barre d'espacement après la mise sous tension).

L'écran suivant apparaît ensuite :



Le premier choix qui vous est offert permet de transférer un programme source en fichier MIM/DOS pour le compiler ou l'analyser.

(Si l'opération a déjà été faite, le programme transformé est déjà présent sur la seconde disquette, faites CTRL A pour passer à la suite).

Pour transférer un programme, répondez simplement aux questions posées :

- drive où se trouve le programme source (défaut : 0),

- nom du programme à translater,

- drive où l'on doit écrire le fichier translaté (défaut: 1)

NOTES :

- Vous pouvez initialiser la deuxième disquette, si ce n'était fait, en tapant ' F,1' au lieu du nom du programme.

- Vous pouvez voir le catalogue des programmes d'une disquette en tapant '*,0' au lieu du nom programme.

Mettez alors dans le drive source (défaut : 0) la disquette où se trouve le programme source.

La disquette initialisée se trouve déjà dans le drive 1, donc si vous avez choisi l'option par défaut, tout est prêt.

Appuyez alors sur ESC pour que l'opération se réalise.

Un message d'erreur apparait si le programme à translater n'existe pas sur la disquette.

Si tout se passe bien, vous voyez le message ''NOMBRE DE BLOCS/DISQUE'' avec un nombre qui s'incrémente. Votre programme est en cours de translation.

Après quelques instants (dépendant de la longueur de votre programme), vous voyez un écran vous demandant si vous voulez :

1/ Comparer deux versions d'un programme.

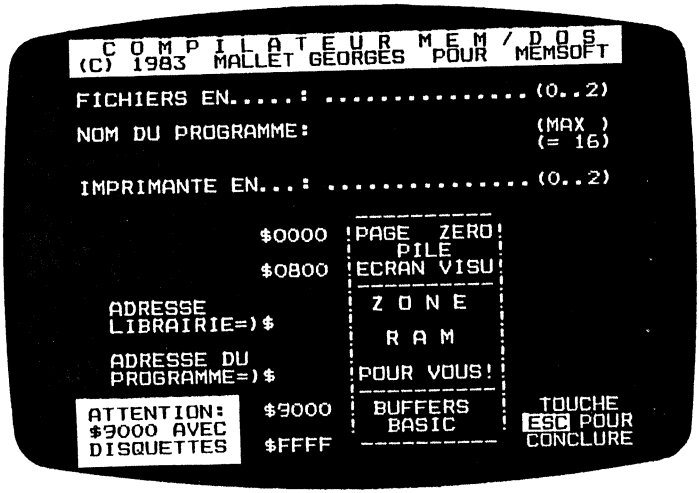
2/ Faire un dossier de votre programme.

3/ Compiler un programme.

Enlevez votre disquette source du premier lecteur et mettez à sa place la disquette compilateur.

Vous pouvez translater un autre programme (dans le cas du comparateur) par exemple en appuyant sur les touches CTRL A.

Tapez sur la touche 3. Le premier programme du compilateur se charge.



Il faut maintenant indiquer les paramètres de compilation.

Le seul qui n'a pas d'option par défaut est le nom du programme à compiler :

drive source : (défaut 1)

nom du programme : ( )

drive objet : (défaut 1)

adresse bibliothèque : (défaut $0D00)

adresse programme objet : (défaut $2001)

Validez par ESC vos choix.

RAPPEL : seul le nom du programme est nécessaire si vous n'avez pris jusqu'ici que les options par défaut

A partir de maintenant, tout est automatique.

Suivant la taille de votre programme, vous pouvez patienter, boire un café ou aller vous restaurer en attendant la fin de la compilation.

Le message ''PROGRAMME ENCAPSULE'' indique que la compilation s'est bien passée. Consultez alors le chapitre II - 5 - pour l'exécution de l'objet. Sinon, reportez-vous au chapitre II - 4 - pour analyser l'erreur survenue.

Les différentes étapes de la compilation

Si vous regardez le compilateur fonctionner, vous pourrez voir successivement :

- Constitution d'un pavé de points matérialisant l'initialisation du compilateur.

- Génération du code objet : le programme est listé au cours de la génération.

- Edition des liens, visualisée par une liste de points. Elle consiste à résoudre les références non résolues lors de la génération du code objet. Prenons comme exemple l'instruction 10 GOTO 1000. Il est impossible de connaître l'adresse de la ligne 1000 lors de la génération du code objet de la ligne 10.

- Encapsulage : l'objet est transformé en un programme BASIC exécutable. Si vous listez un programme compilé, vous verrez un CALL à la première ligne du programme compilé.

II - 4 - Les erreurs de compilation

Lors de la compilation du programme, des erreurs peuvent être détectées. Elles correspondent pour la plupart à des parties de votre programme qui n'ont pas été testées.

En pareil cas, votre ligne est affichée avec le message '' ERREUR '' inséré à l'endroit où le compilateur ne sait plus où donner de la tête. Cet endroit peut être quelques caractères à droite de l'erreur réelle.

En général, l'erreur s'apparente à celle que vous connaissez lors de vos développements avec l'interpréteur BASIC.

SYNTAXE BASIC :

Cette erreur correspond à une partie de votre programme qui donnera certainement ''SYNTAX ERROR'' avec l'interpréteur BASIC.

SYNTAXE MEM/DOS :

Après un mot clé MEM/DOS, l'expression qui suit doit être de type chaîne de caractères construite par concaténation. Vous avez peut être :

- mis un ';' à la place d'un '+'

- oublié une conversion nombre --- chaîne (STR$( ))

MANQUE VARIABLE :

Vous avez oublié de spécifier le nom de la variable dans :

- un DEF FN
  exemple : DEF FN (X) = X + 2
  au lieu de : DEF FN AD (X) = X + 2

- un FN
  exemple : Y = FN (X)
  au lieu de : Y = FN AD (X)

- un FOR
  exemple : FOR = 1 TO 10
  au lieu de : FOR I = 1 TO 10

- le nom d'un tableau
  exemple : DIM (10)
  au lieu de : DIM T (10)

- une variable à gauche dans une affectation
  exemple : 6 = 5 + 1
  au lieu de : Y = 5 + 1

- un INPUT
  exemple : INPUT''X='';
  au lieu de : INPUT''X='';X

- un READ
  exemple : READ
  au lieu de : READ X

- un GET
  exemple : GET
  au lieu de : GET A$

MANQUE OPERANDE :

Vous avez oublié une variable ou doublé un opérateur.

MAUVAIS TYPE :

Celà correspond à l'erreur ''TYPE MISMATCH'' avec l'interpréteur BASIC.

Exemple : IF X = A$
au lieu de : IF X = A
ou : IF X$ = A$

INSTRUCTION INCOMPLETE :

Votre ligne finit par un opérateur sans variable ou le nombre de parenthèses ouvrantes-fermantes n'est pas balancé.

Exemple : X = X +
au lieu de : X = X + 1

Exemple : X = X * (X + 3
au lieu de : X = X * (X + 3)

MANQUE PARAMETRE :

Le nombre de paramètres d'une fonction BASIC n'est pas respecté.

```
Exemple     : A$ = LEFT$ (A$)
au lieu de : A$ = LEFT$ (A$,3)
```

REDEFINITION :

Vous définissez plusieurs fois :

- un DEF FN
  exemple :
```
10 DEF FN AD (X) = X + 2
..
..
1240 DEF FN AD (X) = X - 2
```

- un tableau
  exemple :
```
10 DIM T (10)
..
..
1240 DIM T (30)
```

INDICE :

Dans la définition d'un tableau, vous avez voulu calculer une taille de tableau :

```
exemple : N = 20 : DIMF(N)

au lieu de : DIM T(20)
```

- omis un nombre
```
  exemple     : DIM T (10,)
  au lieu de : DIM T (10,2)
```

- mis un nombre négatif ou plus grand que 32767
```
  exemple     : DIM T (-10)
  au lieu de : DIM T (10)
```

EXPRESSION TROP COMPLEXE :

Votre expression est trop longue. Coupez-la en plusieurs parties.

OBJET TROP LONG :

Vous avez défini un tableau dont les dimensions dépassent la capacité de l'espace RAM.

```
Exemple : DIM T (32000)
```

PROGRAMME TROP LONG :

Votre programme compilé dépasse la capacité de l'espace RAM.

TABLE DES SYMBOLES PLEINE :

Cette erreur est rarissime. Votre programme contient trop de variables, constantes et branchements. Pour y remédier, utilisez seulement quelques variables intermédiaires au lieu de leur donner à chaque fois un nom différent.

SYNTAXE DIRECTIVE :

Une ligne REM commence par un $ et ne continue pas par une directive de compilation. Si ce n'est pas une directive, rajoutez un caractère devant le $.

Exemple : REM $0D00 = LIB
devient : REM'$0D00 = LIB

Si c'est une directive, reportez-vous au chapitre II - 11 - pour vérifier la syntaxe.

DIRECTIVE TARDIVE :

Vous voulez compiler une instruction REM$ alors que du code a déjà été généré. Mettez ces lignes REM$ au début du programme avant toute instruction autre que REM.

N'oubliez pas que l'ordre est :

REM $INTEGER...
REM $COMMON...

Pour plus de détails, reportez-vous au chapitre II - 11 - sur les directives de compilation.

INSTRUCTIONS NON COMPILABLES :

Certaines instructions ne sont pas supportées par ce compilateur. Ce sont :

- SHLOAD, RECALL, STORE : elles n'ont plus de raison d'être puisque vous travaillez avec des disquettes,

- CONT, LIST, DEL : ne sont plus opérants car la notion de lignes et d'interpréteur disparait avec un programme compilé.

- RESUME : cette instruction demanderait une génération de code trop grande. Elle n'a donc pas été implémentée.

- CALL FN : n'est pas compilable actuellement (voir chapitre II- 7 -).

II - 5 - Lancement d'un programme compilé

Si vous n'utilisez pas de carte 80 colonnes, faites un petit programme de démarrage qui est le suivant :

```
10 REM HELLO.40
20 BI$ = ''0D00''
30 PR = 8193 : REM ''$2001''
40 LOAD ''$'' + BI$ + '',$FFFF : BIBLIO.OBJ''
50 POKE 103, PR-INT (PR/256) * 256 : POKE 104, INT (PR/256)
   : POKE PR-1,0 : RUN ''MENU''
```

POKE 103,1
POKE 104,32
POKE 8192,0

MENU est un programme qui peut être indifféremment compilé ou interprété. Vous pouvez bien sur changer ce nom. ATTENTION : ne coupez pas la ligne 50 en plusieurs parties sous peine d'effets imprévisibles.

Si vous utilisez une carte 80 colonnes, recourrez aux deux programmes qui suivent. Ils sont aussi viables dans le cas sans carte 80 colonnes, mais demandent la présence de deux programmes supplémentaires sur votre disquette.

```
10 REM HELLO.80
30 PR = 8193 : REM ''$2001''
40 POKE 103, PR-INT (PR/256) * 256 : POKE 104, INT (PR/256)
   : POKE PR-1,0 : RUN ''HELLO.LIB''
```

```
10 REM HELLO.LIB
20 BI$ = ''0D00''
40 LOAD ''$'' + BI$'',$FFFF : BIBLIO.OBJ''
50 RUN ''MENU
```

Si vous désirez changer les adresses de la bibliothèque ou du programme, utilisez la version 2 en changeant PR (adresse du programme) ligne 30 de HELLO.80 et BI$ (adresse bibliothèque en hexadécimal) ligne 20 de HELLO.LIB.

Note : L'adresse du programme est (en décimal), celle que vous avez indiquée à la compilation.

II - 6 - Les erreurs à l'exécution

Lorsque vous exécutez un programme, une erreur peut se produire.

II - 6 - 1 - Si vous n'avez pas mis de ONERR GOTO :

Le message de l'erreur est affiché et le contrôle vous est rendu. ATTENTION : vous êtes à ce moment dans un état qui vous empêche tout ordre MEM/DOS.

Pour avoir de nouveau le contrôle de MEM/DOS, faites un CALL au début de la bibliothèque :

```
CALL 13 * 256 : REM'$D00
LET ''*''
```

Tout est revenu en ordre.

Si vous voulez connaître la valeur de vos variables, vous pouvez le faire de façon traditionnelle en mode direct par l'instruction PRINT.

Si vous avez besoin de plus d'aide lors de la mise au point de votre programme compilé, n'oubliez pas que les instructions TRACE et NOTRACE sont compilables tout en ayant une action légèrement différente de celle de l'interpréteur. Voyez le chapître II - 7 pour plus de détails.

ATTENTION : Ne relancez pas le programme par un RUN. Les conséquences sont imprévisibles. Rechargez le programme compilé depuis la disquette avant de faire un RUN.

II - 6 - 2 - Si vous avez mis un ONERR GOTO :

Le numéro de l'erreur se trouve à l'adresse 222 comme avec l'interpréteur.

Le ONERR GOTO peut être désactivé en mettant dans le programme POKE 216,0. Ce comportement est identique à celui de l'interpréteur.

II - 6 - 3 - Différence entre programme interprété et compilé en cas d'erreur :

Le compilateur optimise la vitesse en faisant des calculs en entier partout où celà lui semble possible.

Cette optimisation peut entrainer un dépassement de capacité lors de certains calculs.

Exemple : PRINT ''SOMME =';PR% * QT%

peut entrainer l'erreur 53 (ILLEGAL QUANTITY) si le résultat dépasse 32767.

Pour remédier à ce problème, forcez un calcul réel en faisant:

```
VL = PR% : PRINT ''PRIX=';VL * QT%
```

Ce comportement différent se produit surtout si vous voulez connaitre une adresse mémoire à partir de deux octets en mémoire, calcul qui se fait à partir d'entiers. Le résultat devrait donc être un entier, mais peut demander un réel s'il dépasse 32767.

Au lieu de : AD = PEEK (128) + PEEK (129) * 256

faites : AD = 256 : AD = PEEK (128) + AD * PEEK (129)

La règle générale est :

- une opération entre entier et entier donne un entier
- ..... entier .. réel ..... réel
- ..... réel .... réel ..... réel

Un entier est une variable entière ou une constante entière, c'està-dire comprise entre -32767 et +32767.

Toutefois, 10.5 devient une constante réelle mais 10.0 est une constante entière.

II - 7 - Compatibilité avec l'interpréteur

&

Le '&' est une facilité ajoutée au BASIC de l'APPLE II pour étendre le nombre d'instructions du BASIC.

Ce compilateur est le seul à notre connaissance à compiler le & de tel façon qu'il soit utilisable.

Vous pouvez ainsi faire ou utiliser des sous-programmes systèmes normalement prévus pour l'interpréteur.

Toutefois, certains sous-programmes & ne fonctionnent pas avec le compilateur. C'est le cas des sous-programmes & qui modifient le texte source du programme basic (ici inexistant car compilé !).

structure des données

La compilation du & est possible car la structure des variables de votre programme est la même aussi bien en compilé qu'en interprété.

En cas d'erreur en exécution, vous pouvez interroger la valeur de vos variables par l'instruction PRINT (reportez-vous au chapitre II - 6- pour avoir plus de précisions).

La structure commune des variables entre programmes compilés et interprétés vient du fait que les ordres MEM/DOS utilisent des dictionnaires de variables BASIC pour lire/écrire des données sur écran ou disque.

les Tableaux dans les Fichiers

Vous pouvez en MEM/DOS changer un octet qui dimensionne automatiquement les tableaux. Le compilateur ne reconnait pas cet octet.

En MEM/DOS, la valeur par défaut du nombre d'éléments est 10. Celle du compilateur aussi.

Pour prévenir au lieu de guérir, il vous est donc fortement recommandé de dimensionner vos tableaux par l'instruction BASIC DIM qui elle, est reconnue par le compilateur.

DIM : Le DIM fonctionne comme d'habitude, mais n'accepte pas de taille calculée ni de multiples dimensionnement du même tableau qui en interprété serait sélectionné par des tests.

TRACE, NOTRACE

Ces deux instructions d'aide à la mise au point sont compilables.

Toutes les instructions comprises entre un TRACE et un NOTRACE dans l'ordre du programme seront tracées.

Notez la différence avec l'interpréteur, qui lui TRACE jusqu'à l'EXECUTION de NOTRACE.

Les sous-programmes appelés et externes à cette zone ne seront donc pas tracés.

Exemple : le programme :

```
 10 TRACE
 20 GOSUB 100
 30 NOTRACE
 40 END
100 PRINT ''S/P'';
110 RETURN
```

une fois compilé et exécuté affichera :

```
 20   S/P      30
```

La ligne 100 n'a pas été tracée bien que compilée.

ATTENTION : un programme compilé avec option TRACE tient plus de place.

PEEK, POKE, CALL

Ces trois instructions utilisent des paramètres de type entier, c'est-à-dire compris entre -32767 et +32767.

Par exemple, pour effacer l'écran, faîtes CALL 936 au lieu de CALL 64600 (dans ce cas, l'instruction HOME efface aussi bien l'écran et est plus lisible !!!).

Pour convertir une adresse réelle en adresse entière, faîtes:

```
IF AD 32767 THEN AD = AD -65536
```

Pour mettre le résultat dans une variable entière, vous pouvez utiliser une fonction :

```
 10 DEF FN EN(X) = X - 65536 * (X   32767)
 ..
 ..
100 AD% = FN EN(64600)
110 CALL AD%
```

ATTENTION: CALL FN EN (64600) rentre en conflit avec l'instruction CALL FN, et n'est donc pas possible.

ATTENTION !

Cas particulier : l'interpréteur APPLESOFT ne permet pas d'utiliser la valeur binaire 1000000000000000 qui devrait être représentée par +32768 ou -32768, le compilateur reproduit cette particularité.

Il est donc impossible d'utiliser sous forme d'adresse entière dans un PEEK, POKE ou CALL l'adresse 32768, il faut dans ce cas utiliser une adresse dans une variable flottante.

CALL FN

L'accès du programme compilé aux valeurs des variables est fait actuellement à travers des adresses absolues pour des raisons de gain de vitesse.

Ceci nous empêche donc pour l'instant la compilation du CALL FN qui demande une pile des adresses des variables.

EXECUTE : LET"!"

Le LET ''!'' est compilable s'il ne comprend pas d'instruction de retour (GOTO) dans le programme compilé. Il peut toujours être utilisé comme boîte aux lettres pour passer des informations entre deux programmes.

Exemple :

```
20 LET''!'' : GET A$
```

est compilable mais

```
20 LET'' 10 GOTO 30'' : LET ''!'' : END
30 ...
```

n'est pas compilable puisque la notion de numéro de ligne disparaît dans un programme compilé.

SHLOAD, RECALL, STORE

Ces trois instructions sont utiles pour des échanges de données avec un mini-cassette. L'utilisation des disques et disquettes rend obsolète ces trois instructions.

CONT

La notion d'interpréteur ayant disparu, vous ne pouvez plus arrêter et continuer l'exécution d'un programme une fois qu'il est compilé.

LIST, DEL

La notion de lignes d'instruction disparait avec un programme compilé. Il n'est donc plus possible de visualiser ou supprimer des lignes.

RESUME

La compilation du RESUME aurait demandé une génération de code trop grande eu égard à la place mémoire d'un micro ordinateur.

N'oubliez pas que le ONERR GOTO est compilable et que le numéro de l'erreur se trouve en PEEK (222) (comme avec l'interpréteur BASIC).

RUN

Il est interdit de relancer le programme par RUN sous risque de catastrophe.

Le RUN ''programme'' a un fonctionnement très voisin de sa version interprétée (voir II-8-).

Le RUN '' '' n'est plus utilisable, puisqu'il sert à réunir ensemble des programmes interprétés.

Le RUN ''/'' a un comportement différent (voir II-11-).

LOAD

Le seul LOAD utilisable est le LOAD binaire mais attention, la place où était chargée le binaire en mémoire en version interprétée n'est peut-être plus libre !

Contrôle C

Les programmes compilés ne sont plus interruptibles par CTRL/C.

En effet, le test après chaque instruction de l'enfoncement de cette touche demanderait 3 octets supplémentaires par instruction et allongerait le temps d'exécution.

Par contre, l'arrêt du catalogue (LET '' '') par CTRL/C continue à fonctionner.

II - 8 - Enchaînement de programmes compilés et interprétés

II - 8 - 1 - Interprétés --> compilés

La méthode a déjà été explicitée au chapitre II - 5 -. Notons néanmoins que si la bibliothèque est déjà chargée à la bonne adresse et si l'adresse de début de programme est correcte, un simple RUN ''nom programme'' suffit.

II - 8 - 2 - Compilés --> compilés

Tous les programmes compilés finissent par une phase d'encapsulage qui consiste à faire croire au MEM/DOS qu'il s'agit d'un simple programme à interpréter.

Vos commandes RUN''PROGRAMME'' continuent donc de fonctionner sans changer votre programme si tous les programmes de la chaîne ont été compilés à la même adresse et avec la même adresse bibliothèque.

Sinon, repasser par le programme de lancement donné au chapitre II - 5 - adapté à la nouvelle configuration après avoir éventuellement récupéré la place de l'ancienne bibliothèque.

II - 8 - 3 - Compilés --> interprétés

Le lancement d'un programme interprété à partir d'un programme compilé nécessite de remettre en place certains pointeurs nécessaires à MEM/DOS qui ont été modifiés par le programme compilé.

Pour celà, il vous faudra ajouter les lignes qui sont dans le programme END.COMPILE au début du programme interprété.

```
1 REM END.COMPILE
20 CALL 13 * 256 : REM'$0D00 --> DEBUT BIBLIOTHEQUE.RESTAURE
   CHRGOT MEM/DOS
110 POKE 103,1 : POKE 104,13 : POKE 3328,0
    : RUN ''PRG.INTERPRETE''
    : REM DEBUT PROGRAMME EN $0D00 SI 80 COLONNES
```

> ATTENTION: si vous voulez un jour compiler ce programme, n'oubliez pas d'enlever ces lignes.

II - 9 - Le problème du foisonnement

Le foisonnement est un rapport entre le nombre d'octets du programme compilé et le nombre d'octets du programme source. Par exemple, un foisonnement de 2 veut dire qu'un octet de votre programme source génère 2 octets dans votre programme compilé.

L'interpréteur BASIC sur l'APPLE II utilise une codification des mots-clés du langage appellés TOKEN.

Par exemple, PRINT est représenté en mémoire par l'octet 186, donc un gain de 4 octets par rapport au texte source. Une fois compilé, un PRINT correspond à un appel d'un sous-programme système qui fait un retour à la ligne. Cet appel prend trois octets. Le foisonnement peut donc être supérieur à 1 dans notre cas.

Les compilateurs d'autres langages (ou du même langage dans une autre version) demandent au contraire un texte source sous une forme standard où PRINT représente 5 octets. Dans ce cas, le foisonnement est un nombre inférieur à 1.

Nous arrivons donc au point crucial : on peut écrire de gros programmes avec l'interpréteur BASIC sur APPLE II, mais parfois on ne peut pas les exécuter une fois compilés car ils deviennent trop gourmands en place mémoire.

Pour résoudre ce problème, vous pouvez couper votre programme en deux parties qui se passent les données à travers un COMMON. Voyez le chapitre II - 11 - pour plus de détails.

Vous pouvez aussi changer légèrement votre programme afin de diminuer sa taille. Voyez le chapitre II - 10 - pour connaître les optimisations possibles.

II - 10 - Conseils de programmation

Il existe deux façons d'optimiser un programme :

- en place mémoire occupée,
- en vitesse d'exécution.

L'optimisation en place mémoire n'est pas obligatoirement contradictoire avec l'optimisation de la vitesse d'exécution. Aussi, essayez de suivre ces grandes lignes si vous ne le faisiez déjà.

Toutefois, certaines des méthodes que nous exposons ont même tendance à ralentir la vitesse d'exécution du programme interprété. Ne vous en formalisez pas si vous êtes sur de le compiler.

II - 10 - 1 - Optimisation en place mémoire

GOSUB :

Si une suite d'instructions ou une expression apparaît plusieurs fois dans votre programme, faites en un sous-programme ; ce sera plus rentable.

Par exemple, un sous-programme qui insère un caractère A$ dans la chaîne CH$ à la position I s'écrit :

```
CH$ = MID$ (CH$, 1, I - 1) + A$ + MID$ (CH$,I + 1)
```

Cette expression est très gourmande en génération de code car elle travaille sur des chaînes de caractères. La mettre en sous-programme vous fait donc gagner autant de place qu'il existe d'appels à ce sous-programme (-1 pour le sous-programme lui-même).

Au niveau théorique, on peut aller jusqu'à mettre I = I + 1 dans un sous-programme qui prend 9 octets en génération de code, et seulement 3 lors d'un appel de sous-programme. Au niveau pratique, la modification et la relecture d'un programme ainsi modifié devient très difficile.

A vous de trouver le niveau à partir duquel vous passez trop de temps à tourner les pages de votre programme pour savoir ce que font les sous-programmes.

Sous-expression commune

L'idée est la même que celle des GOSUB mais laisse le source de votre programme lisible. Toutefois, elle consomme souvent une variable que l'on utilise pour y mettre la valeur temporaire.

Exemple 1 : regarder si un caractère est dans A...Z

```
100 IF MID$ (CH$,I,1)  = ''A'' AND MID$ (CH$,I,1)  = ''Z''
    THEN A$ = MID$ (CH$,I,1) : GOTO 120
110 PRINT''ERREUR'':GOTO...
120 ........
```

devient :

```
100 EC$ = MID$ (CH$,I,1) : IF EC$  = ''A'' AND EC$  = ''Z''
    THEN A$ = EC$ : GOTO 120
110 PRINT''ERREUR'':GOTO...
120 ........
```

Exemple 2 : Calculer un taux de remise

```
100 IF PR * QT * TX    = 10000 AND PR * QT * TX    50000
    THEN RM = .1
```

devient :

```
100 EC = PR * QT * TX : IF EC    10000 AND EC    50000
    THEN RM = .1
```

Pour éviter un grand nombre de variables temporaires, utilisez toujours les mêmes (par exemple EC et EC$ sont réutilisables dans notre exemple) dans votre programme si vous faites cette phase d'optimisation.

Cette méthode a un second avantage : elle optimise la vitesse d'exécution car on ne fait qu'une fois un calcul là où on le faisait deux ou trois fois.

ON GOTO :

Suivant la valeur d'un élément, on fait un branchement en des points différents du programme. Utilisez plutôt un ON... GOTO pour aller à ces différentes parties de votre programme.

Exemple : branchement suivant le code client :

```
100 IF CC = 1 THEN 200
110 IF CC = 2 THEN 300
120 REM CC = 3 N'EXISTE PAS
130 IF CC = 4 THEN 400
140 IF CC = 5 THEN 500
150 PRINT''ERREUR'':GOTO...
```

devient :

```
100 ON CC GOTO 200, 300, 150, 400, 500
150 PRINT''ERREUR'':GOTO...
```

INDICATEURS BOOLEENS :

On peut avoir des indicateurs dans un programme pour diverses raisons : nombre positif ou négatif, solde débiteur ou créditeur... Ces indicateurs sont des booléens puisqu'ils n'ont que deux états.

Lors des tests sur les booléens, il est inutile de les comparer à une des constantes 0 ou 1. Ne rien mettre correspond à tester si la valeur est différente de zéro.

Exemple : indicateur de nombre positif ou négatif

```
100 BL% = 0 : IF NB    = 0 THEN BL% = 1
110 IF BL% = 1 THEN...
```

devient :

```
100 BL% = (NB    = 0)
110 IF BL% THEN...
```

Cette façon de travailler permet en plus de gagner en vitesse d'exécution. Ce point est particulièrement important si on applique cette méthode à la variable status MEM/DOS WS dont on teste souvent la non nullité.

NOTE : Cette remarque est aussi valable pour l'interpréteur.

Variables de travail

Cette optimisation demande généralement beaucoup de travail pour un gain de place très faible. Il faut donc que vous soyez à quelques octets près pour y recourrir.

Le principe consiste simplement à bien étudier la logique de votre programme pour éviter l'utilisation de trop de variables de travail.

Exemple : soit T1 et T7, 2 variables de stockage temporaire

```
100 T1 = PR * QT * TX
110 IF QT    10000 THEN PT = T1 * .90
..
..
200 T7 = Q1 + Q2 + Q3
210 IF T7    100 THEN RM = .1 : GOTO 230
220 RM = 0
..
..
```

devient :

```
100 T1 = PR * QT * TX
110 IF QT    10000 THEN PT = T1 * .90
..
..
200 T1 = Q1 + Q2 + Q3
210 IF T1    100 THEN RM = .1 : GOTO 230
220 RM = 0
```

II - 10 - 2 - Gain en vitesse

IF...THEN... : GOTO

Lors d'un choix multiple, on oublie souvent de mettre un GOTO qui irait à la fin de ce choix. En en mettant, on gagne le temps de calcul des autres tests qui seront faits mais pas validés.

Cette méthode peut ne pas demander plus de code dans votre programme si vos tests sont jointifs.

Exemple : calcul du taux de remise

```
100 IF QT    1000 THEN RM = 0
110 IF QT    = 1000 AND QT    10000 THEN RM = .1
120 IF QT    = 10000 THEN RM = .2
130 .......
```

devient :

```
100 IF QT    1000 THEN RM = 0 : GOTO 130
110 IF QT    10000 THEN RM = .1 : GOTO 130
120 RM = .2
130 .....
```

Invariant :

On appelle invariant une expression qui se trouve dans une boucle et dont le résultat ne varie pas. Pour optimiser les temps de calculs, il suffit alors de sortir cette expression de la boucle.

Exemple : somme d'une facture avec taxe et remise

```
100 SM = 0
110 FOR I = 1 TO N
120 SM = SM + PR (I) * TX * RM
130 NEXT I
```

devient :

```
100 SM = 0
110 FOR I = 1 TO N
120 SM = SM + PR (I)
130 NEXT I
140 SM = SM * TX * RM
```

Un invariant peut être sorti au début ou à la fin de la boucle suivant les cas.

Variables indicées :

A chaque fois que votre programme accède à un élément du tableau, il y a une perte de temps pour calculer l'adresse de cet élément en mémoire centrale.

Si dans une boucle FOR..NEXT vous accédez plusieurs fois au même élément d'un tableau, utilisez plutôt une variable temporaire pour mettre la valeur de l'élément du tableau.

Exemple : calcul du minimum et du maximum d'un tableau

```
100 MI = 1 E 38 : MA = -1 E 38
110 FOR I = 1 TO N
120 IF T(I)    MI THEN MI = T(I)
130 IF T(I)    MA THEN MA = T(I)
140 NEXT I
```

devient :

```
100 MI = 1 E 38 : MA = -1 E 38
110 FOR I = 1 TO N : T1 = T(I)
120 IF T1    MI THEN MI = T1
130 IF T1    MA THEN MA = T1
140 NEXT I
```

On a gagné le temps de trois calculs d'adresse.

Multiplication et puissance :

Dans certains cas, une multiplication peut se réduire à une addition, et une puissance à une multiplication.

Cette méthode est surtout viable quand le deuxième opérande est la constante 2.

Exemple : Y = X * 2 : Z = X 2

devient : Y = X + X : Z = X * X

Entiers :

Un des avantages du compilateur MEM/DOS est de vous offrir une bibliothèque de calculs sur les nombre entiers.

En effet, la compilation d'une expression permet de déterminer si tous les éléments intervenant dans un calcul sont entiers donc de prévoir si le résultat le sera.

Si vous ne voulez pas modifier tout votre programme afin de transformer vos I = I + 1 en I% = I% + 1, utilisez la directive de compilation REM$INTEGER I. Voyez le chapitre II - 11 - pour plus de détails.

Booléens :

Nous avons déjà parlé des booléens dans les gains en place mémoire. Il advient que la même approche permet aussi un gain en vitesse d'exécution : le test sur la constante n'étant plus fait, le programme va plus vite.

Exemple :

regardez celui du chapitre II - 10 - 1 sur les booléens.

II - 11 - Directives de compilation

Ces dernières permettent au compilateur de générer du code plus performant dont les fonctionnalités se rapprochent le plus de celles du programme compilé.

Une directive se remarque dans un programme source par une instruction REM suivie d'un $. Elle n'a donc aucun effet sur un programme interprété, mais en a un sur le compilateur.

ATTENTION :

- Hormis la directive $LOMEM, les directives doivent se trouver avant toute ligne exécutable.

- Ne mettez pas de blanc superflu entre le REM et le $. Lors de la compilation un message DIRECTIVE apparaîtra sur la ligne REM$ pour indiquer que cette ligne a bien été comprise comme directive de compilation.

INTEGER :

La directive INTEGER permet de définir des variables du programme comme étant entières, alors qu'elles apparaissent dans le programme source comme étant réelles. Ces variables peuvent être aussi bien des variables simples que des tableaux.

Exemple :

```
10 REM$INTEGER N,T (10), S (5,2)
```

définit N,T ( ) et S ( ) comme étant des variables entières.

La variable de parcours d'une boucle FOR..NEXT peut aussi être déclarée comme étant de type entier. Tous vos calculs sur tableaux auront aussi un gain de temps très appréciable car l'indice sera entier.

Exemple : somme d'un tableau :

```
  10 REM$INTEGER I,N
  ..
  ..
 100 SM = 0
 110 FOR I = 1 TO N
 120 SM = SM + T(I)
 130 NEXT I
```

RAPPEL : L'interpréteur n'accepte pas de variables de boucles entières dans un FOR ... NEXT. Pour l'obtenir à la compilation, la seule méthode est celle préconisée ci-dessus de la directive $INTEGER.

COMMON, USECOM, LOMEM

Ces trois directives vous permettent de compiler le RUN''/''. Si vous n'avez pas ce besoin, sautez cette partie allégrement.

Vous voulez vraiment compiler une chaîne de programmes avec des variables communes ? OK, alors allons-y. Pour des raisons liées à l'accès rapide aux données en mode compilé, le RUN ''/'' ne peut pas fonctionner de la même manière.

En mode compilé, il faut déclarer dans tous les programmes de votre chaîne de programmes, la liste des variables qui apparaissent dans tous les programmes.

Pour vous aider dans ce travail, un utilitaire de références croisées est fourni avec le compilateur. Utilisez le en demandant les références croisées des variables programme par programme.

Vous n'aurez plus qu'à fusionner ces listes en une seule que vous inclurez dans chaque programme.

Exemple :

créons la liste des variables communes à deux programmes

Programme 1 :

```
 8 LOMEM : 16384
10 DIM T(20)
20 N = 20 : K = 2
30 FOR I = 1 TO N
40 T(I) = I * K
50 NEXT I
60 LOAD ''/PROGRAMME 2''
```

Programme 2 :

```
10 FOR I = 1 TO N
20 SM = SM + T(I)
30 NEXT I
40 PRINT SM
```

Liste des variables pour le programme 1 : I,K,N,T(20)
. . . . . 2 : I,N,SM,T(20)
FUSION DES LISTES : I,K,N,SM,T(20)

Il faut donc insérer la ligne :

```
5 REM$COMMON I,K,N,SM,T(20)
```

dans chacun des deux programmes.

Notre travail est presque fini. Il faut dire dans le deuxième programme que c'est un programme esclave, c'est-à-dire que les valeurs des variables viennent d'un autre programme. Pour celà, il suffit d'insérer un $USECOM dans le deuxième programme.

Troisième et dernière étape : dire à quelle adresse doivent être les variables. Vous le faîtes avec l'interpréteur par un LOMEM. Le travail est le même avec un programme compilé, sauf que le $LOMEM doit être dans tous les programmes, et que l'adresse doit être une constante.

Nous pouvons maintenant modifier nos deux programmes :

Programme 1 :

```
  5 REM$COMMON I,K,N,SM,T(20)
  7 REM$LOMEM 16384
  8 LOMEM : 16384
 10 DIM T(20)
 20 N = 20 : K = 2
 30 FOR I = 1 TO N
 40 T(I) = I * K
 50 NEXT I
 60 RUN ''/PROGRAMME 2''
```

Programme 2 :

```
  5 REM$COMMON I,K,N,SM,T(20)
  6 REM$USECOM
  7 REM$LOMEM 16384
 10 FOR I = 1 TO N
 20 SM = SM + T(I)
 30 NEXT I
 40 PRINT SM
```

NOTES :

1/ Aucune variable supplémentaire non définie par $COMMON ne peut ûtre utilisée dans l'un ou l'autre des programmes.

2/ Les variables qui ne sont utilisées QUE par des masques ou des fichiers sont automatiquement transférées sans avoir à ûtre déclarées.

3/ Si la liste complète des variables communes ne tient pas en une ligne REM$COMMON, vous pouvez en utilisez plusieurs à la suite.

4/ Il est impératif de commencer l'exécution par un programme n'ayant pas de $USECOM et définissant le LOMEM basic de l'ensemble.

5/ ATTENTION : LE $LOMEM doit définir l'endroit oç commencent les variables sans écraser aucun des programmes de la chaine, mais le plus gros programme compilé n'est pas forcément le plus gros programme à interpréter ! Utilisez les indications données par le compilateur pour connaitre la taille de chaque module.

Nous pouvons maintenant modifier nos deux programmes :

Programme 1 :

```
  5 REM$COMMON I,K,N,SM,T(20)
  7 REM$LOMEM 16384
  8 LOMEM : 16384
 10 DIM T(20)
 20 N = 20 : K = 2
 30 FOR I = 1 TO N
 40 T(I) = I * K
 50 NEXT I
 60 RUN ''/PROGRAMME 2''
```

Programme 2 :

```
  5 REM$COMMON I,K,N,SM,T(20)
  6 REM$USECOM
  7 REM$LOMEM 16384
 10 FOR I = 1 TO N
 20 SM = SM + T(I)
 30 NEXT I
 40 PRINT SM
```

NOTES :

1/ Aucune variable supplémentaire non définie par $COMMON ne peut être utilisée dans l'un ou l'autre des programmes.

2/ Les variables qui ne sont utilisées QUE par des masques ou des fichiers sont automatiquement transférées sans avoir à être déclarées.

3/ Si la liste complète des variables communes ne tient pas en une ligne REM$COMMON, vous pouvez en utilisez plusieurs à la suite.

4/ Il est impératif de commencer l'exécution par un programme n'ayant pas de $USECOM et définissant le LOMEM basic de l'ensemble.

5/ ATTENTION : LE $LOMEM doit définir l'endroit où commencent les variables sans écraser aucun des programmes de la chaîne, mais le plus gros programme compilé n'est pas forcément le plus gros programme à interpréter ! Utilisez les indications données par le compilateur pour connaître la taille de chaque module.

ATTENTION

L'ordre des directives est important. Il faut mettre dans l'ordre les lignes :

- REM$INTEGER
- REM$COMMON
- REM$USECOM
- REM$LOMEM

- Vous pouvez mettre la liste de vos variables en INTEGER ou en COMMON sur plusieurs lignes si vous avez beaucoup de noms de variables.

- Les directives doivent être avant tout autre ligne exécutable de votre programme.

Vous pouvez constater que le premier programme de notre exemple ne contient pas de $USECOM. Celà veut dire qu'il fait l'équivalent d'un CLEAR en BASIC au début de son exécution. Vous devez donc mettre un $USECOM au programme de menu général de votre chaîne dont le rôle est de passer les variables aux différents programmes.

Il suffit alors de faire un premier programme qui ne sera exécuté qu'une seule fois. Son rôle sera de faire le CLEAR puis, d'appeller le programme MENU.

```
 1 REM HELLO.MENU
 5 REM$COMMON I,N,T(20)
 7 REM$LOMEM 16384
10 RUN ''/MENU''
```

```
 1 REM MENU
 5 REM$COMMON I,N,T(20)
 6 REM$USECOM
 7 REM$LOMEM 16384
10 PRINT ''N='';N
20 INPUT ''CHOIX:'';I
30 IF I = 1 THEN : RUN ''/PROG 1''
40 IF I = 2 THEN : RUN ''/PROG 2''
50 GOTO 20
```

```
 1 REM PROG1
 5 REM$COMMON I,N,T(20)
 6 REM$USECOM
 7 REM$LOMEM 16384
10 N = 1
20 RUN ''/MENU''
30 PRINT ''ERREUR''
```

```
 1 REM PROG2
 5 REM$COMMON I,N,T(20)
 6 REM$USECOM
 7 REM$LOMEM 16384
10 N = 2
20 RUN ''/MENU''
30 PRINT ''ERREUR''
```

RAPPEL : LOAD''/NOM'' et RUN''/NOM'' donnent le même résultat, c'est-à-dire charge et exécute le prochain programme sans remettre les valeurs des variables à zéro.

COMPILATEUR BASIC

MEM/DOS

III - ENVIRONNEMENT LOGICIEL

III - ENVIRONNEMENT LOGICIEL

Un langage informatique se définit par le langage dans lequel on doit écrire, mais aussi par les outils qui lui sont associés.

Le compilateur MEM/DOS vous apporte deux briques pour vous aider dans la construction de vos programmes :

- un outil de documentation,

- un outil de comparaison.

III - 1 - Outil de documentation

Celui-ci vous permet d'avoir un listing indenté de votre programme afin de mieux voir la structure générale du programme dans le cas des boucles FOR..NEXT et des IF imbriqués. Il teste en même temps si la syntaxe de votre programme est conforme à celle du BASIC dans l'optique d'une compilation.

Il vous donne aussi, à la demande, les références croisées de vos variables, constantes et branchements.

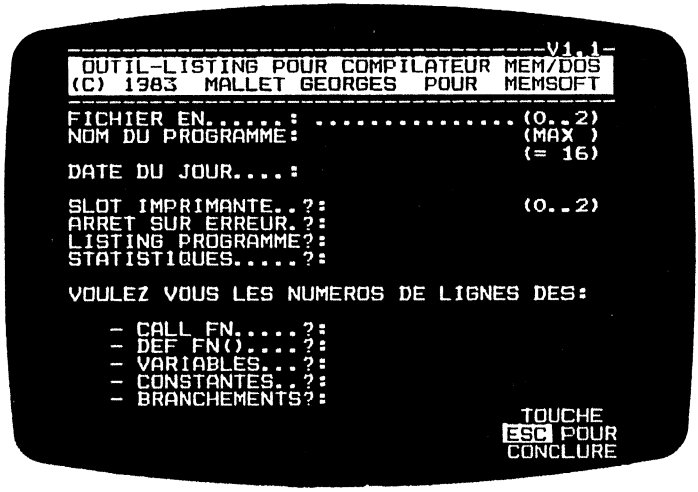
Lancement de l'outil

Mettez la disquette OUTILS POUR COMPILATEUR MEM/DOS dans le premier lecteur. Mettez votre APPLE sous tension. Vous verrez un logo puis, un écran vous demandant s'il faut translater un programme. Faîtes le si ce n'est déjà fait comme explicité en II - 3 -.

Appuyez sur les touches CTRL et A si c'était déjà fait.

Dans tous les cas, appuyez sur la touche 2 pour avoir le dossier de votre programme.

Après quelques instants, vous voyez un écran qui vous demande les différents paramètres. Regardons les ensemble.



Fichier en : (défaut 1)

C'est le lecteur suivant la numérotation MEM/DOS dans lequel vous devez insérer la disquette contenant votre programme translaté.

Nom du Programme : ( )

Tapez ici le nom de votre programme.

Date du Jour : ( )

Si vous voulez qu'elle apparaisse sur votre listing, mettez la.

Slot Imprimante : (défaut 1)

Cet outil est prévu pour une imprimante. Si vous mettez 0, le résultat apparaîtra sur l'écran. Cette option peut vous permettre de tester un programme du point de vue syntaxique avant de le compiler.

Arrêt sur erreur : (défaut 0)

Mettez un N (pour NON) si vous voulez que ce programme fasse son travail hors de votre présence. Il vous suffira alors de regarder le nombre d'erreurs à la fin du listing pour savoir si votre programme contient ou non des erreurs syntaxiques.

Listing Programme : (défaut 0)

Si votre programme est très long et que vous avez un listing récent, mettez un N.

Statistiques : (défaut 0)

Cette option est très intéressante car elle imprime à la fin de l'analyse de votre programme, les noms des mots-clés BASIC que vous utilisez et le nombre de fois où ils apparaissent dans votre programme. Cette option vous permet de mieux appréhender votre façon de programmer.

Numéros de lignes des :

CALL FN..... : (défaut N)
DEF FN ( )..... : (.....)
variables..... : (.....)
constantes..... : (.....)
branchements..... : (.....)

Mettez O (pour OUI) à une ou plusieurs des options dont vous voulez les références croisées. Signalons au passage que la liste des constantes est rarissime dans cette gamme d'outils pour BASIC, bien que très utile.

ATTENTION:

- Si vous demandez l'étude d'un très gros programme avec toutes les options, vous pouvez avoir le message d'erreur TABLE DES SYMBOLES PLEINE. Pour y remédier, faites les différentes références croisées en redémarrant deux fois (ou plus) le programme.

- Demandez par exemple la première fois le listing avec les CALL FN, DEF FN ( ) et variables, puis redémarrez le programme en mettant N au listing, N aux statistiques, O aux constantes et branchements.

Il ne vous reste plus qu'à appuyer sur la touche ESC et à admirer votre dossier sur l'imprimante. Une fois fini, séparez les feuilles, mettez les dans un classeur, et dormez tranquillement.

Vous apprécierez cette méthode de travail si vous devez vous replonger dans votre programme après quelques mois.

III - 2 - Outil de comparaison

Ce programme est celui que vous allez le moins utiliser, mais vous remercierez votre (ou vos) dieu (x) qu'il existe lorsque vous en aurez besoin.

Son but ? Vous dire les lignes que vous avez modifiées, ajoutées, supprimées entre la version actuelle et celle d'il y a une semaine (c'est-à-dire d'une de vos sauvegardes de sécurité).

Vous pouvez enfin trouver avec plus de facilité le ''bug'' qui fait que votre programme ne marche plus alors qu'il fonctionnait très bien il y a quelques jours.

Lancement de l'outil

Mettez la disquette OUTILS POUR COMPILATEUR MEM/DOS dans le premier lecteur. Mettez votre APPLE sous tension. Vous verrez un logo puis, un écran vous demandant le nom du programme à translater. Faites le comme explicité en II - 3 - avec votre version du jour.

Remettez la disquette OUTILS et appuyer sur les touches CTRL et A lors du choix de l'outil. Vous verrez l'écran vous redemander le nom du programme à translater.

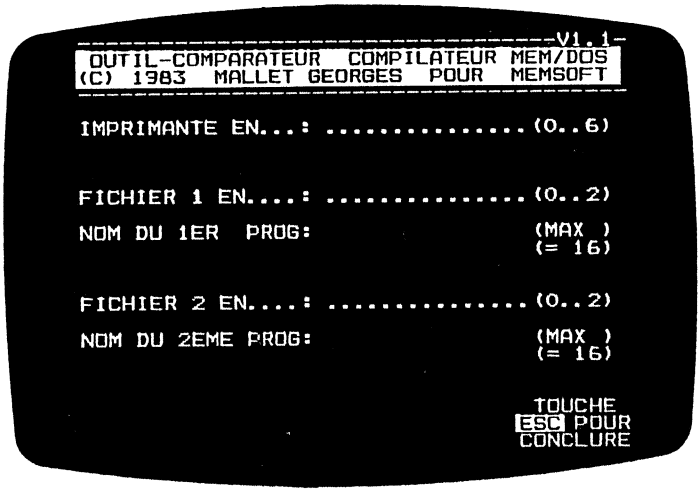
Courrez à la banque pour prendre dans votre coffre la disquette contenant l'ancienne version. Enlevez la disquette OUTILS, mettez à la place votre copie de sauvegarde. Laissez en place la disquette dans le deuxième lecteur.

Donnez le nom de votre programme à translater puis, appuyez sur la touche ESC. Ainsi, la deuxième disquette contient les deux versions de votre programme.

ATTENTION : cette méthode est valable si les noms sont différents. Si ce n'est le cas, utilisez deux disquettes que vous insérerez dans chacun des deux lecteurs lors de la comparaison.

Lorsque vous voyez l'écran du menu des outils, remettez la disquette OUTILS puis, appuyez sur la touche 1 pour faire la comparaison.

Après quelques instants, vous voyez un écran qui vous demande les différents paramètres.



Regardons les ensemble.

Imprimante en : (défaut 0)

Si votre programme a subi très peu de modifications, le résultat sur écran suffira. Sinon, tapez 1 si votre imprimante est reliée à l'APPLE par le slot 1.

Fichier 1 en : (défaut 1)

Si vous avez deux disquettes contenant chacune une version différente de votre programme translaté, tapez 0, enlevez la disquette OUTILS, et mettez une de vos disquettes à la place.

Nom du premier programme : ( )

Tapez ici le nom d'une des versions.

Fichier 2 en : (défaut 1)

Cette valeur n'a pratiquemment jamais à être modifiée.

Nom du deuxième programme : ( )

Tapez ici le nom de l'autre version.

Il ne vous reste plus qu'à appuyer sur la touche ESC et à voir les numéros des lignes différentes s'afficher ou s'imprimer.

Bonne chance dans votre chasse au BUG

COMPILATEUR BASIC

MEM/DOS

IV - EXEMPLES

IV - EXEMPLES

Des programmes de démonstration sont sur la disquette OUTILS POUR COMPILATEUR MEM/DOS.

Il s'agit de deux jeux avec un menu d'appel en BASIC interprété qui s'appelle MEMJEU.INIT.

Tapez RUN''MEMJEU.INIT'' et choisissez un chiffre de 1 à 4.

Appréciez la différence de vitesse entre un programme compilé et interprété dés que l'on utilise des calculs sur des nombres entiers.

Pour retourner au menu, tapez sur les touches CTRL et A.

Profitez en aussi pour regarder dans MEMJEU.MENU comment se fait l'appel des deux programmes compilés.

V - GLOSSAIRE

Algorithme:
---

ensemble d'instructions concourrant à un but commun donné.

Bibliothèque (ou Librairie):
---

ensemble de sous-programmes en langage machine utiles à l'exécution d'un programme compilé.

Catalogue:
---

liste des noms des différents objets qui sont sur une disquette.

Chaîne (de caractères):
---

elle est représentée en BASIC par une constante (''VOTRE NOM:''), une variable (NM$). Elle peut être le résultat d'une concaténation (LET ''*,''+STR$(D)).

Chaîne (de programmes):
---

une mémoire d'ordinateur est parfois trop petite pour un problème donné. Aussi, on scinde le problème en plusieurs parties en mettant chaque partie dans un programme. Exemple : chaîne de comptabilité, gestion de stocks,...

Compilateur:
---

programme qui traduit votre programme écrit dans un langage de niveau donné (ici BASIC) en un langage de plus bas niveau (ici langage machine 6502).

CTRL A:
---

enfoncez la touche à gauche du clavier marqué CTRL puis, sans relâcher cette dernière, appuyez sur la touche A.

Directive (de compilation):
---

elle est constituée d'une instruction REM suivie d'un '$' et d'un des mots-clé : INTEGER, COMMON, USECOM, LOMEM.

Disquette:
---

support magnétique pouvant contenir approximativement 140000 caractères dans le cas de l'APPLE II.

Drive:
---
lecteur de disquette.

Formatter:
---

initialisation d'une disquette en vue de mettre des informations dessus. Si vous voulez le faire en MEM/DOS, tapez LET '' F,1'' avec une disquette vierge dans le deuxième lecteur.

Handler:
---

programme en langage machine qui permet de connecter des périphériques de même nature mais de constructeurs différents sans que votre programme n'ait à en tenir compte. Exemple : lecteurs de disquette/disque, cartes 80 colonnes, tables traçantes, imprimantes.

Langage machine:
---

langage directement compréhensible par le processeur d'un ordinateur. C'est le langage le plus rapide sur tous les ordinateurs.

Numérotation (des lecteurs):
---

le premier lecteur de disquette est le numéro 0 pour MEM/DOS, le deuxième lecteur de disquette est le numéro 1 pour MEM/DOS.

Opérande:
---

constante ou variable dans une expression.

Opérateur:
---

opération à faire sur les opérandes qui se trouvent à droite et/ou à gauche.

Programme esclave:
---

programme compilé qui utilise un COMMON, c'est-à-dire qui contient une directive REM$USECOM.

Programme maître:
---

programme compilé qui initialise un COMMON, c'est-à-dire qui ne contient pas de directive REM$USECOM. Il est utilisé une seule fois dans l'exécution de la chaîne de programmes.

Programme objet:
---

appelé aussi programme compilé. Equivalent en langage machine de votre programme BASIC. Cette traduction a été faite par le compilateur.

Programme source:
---

programme écrit dans un langage compréhensible pour l'être humain. Dans notre cas, un programme source est écrit en BASIC.

Références croisées:
---

liste triée des objets d'un programme avec les numéros de lignes où ils apparaissent. Les objets peuvent être des variables, constantes ou branchements.

Slots:
---

connecteurs situés au fond de la grande plaque qui est dans l'APPLE. Vous devez mettre la carte du compilateur dans un de ceux numérotés de 1 à 6.

Translation:
---

opération faite dans le compilateur MEM/DOS pour mettre votre programme source dans un état plus facilement lisible pour le compilateur.

Vierge:
---

se dit des disquettes non encore utilisées. Il faut les formatter avant de pouvoir en faire un bon usage.

EXCLUSION DE GARANTIE

MEMSOFT S.A. ne confère aucune garantie sur le logiciel décrit dans le présent manuel, à la seule exception de l'intégrité physique du support et ce, pendant trois (3) mois à compter de sa livraison au revendeur.

En tout état de cause, la responsabilité de MEMSOFT S.A., éventuellement admise, est limitée au maximum au montant du prix de vente hors taxe du produit par MEMSOFT S.A. au revendeur, conformément aux tarifs appliqués par MEMSOFT S.A..

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

MEMSOFT S.A. n'a aucun lien de droit avec le client final ou l'utilisateur.

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