2 - INTRODUCTION A LA COMPILATION

La principale différence entre un interpréteur et un compilateur est dans la manière d'exécuter un programme.

- L'interpréteur traduit et exécute au fur et à mesure chaque instruction de chaque ligne de programme. Il compare chaque mot et chaque signe avec des tables en mémoire pour s'assurer que leur syntaxe est correcte et pour en déduire la marche à suivre, c'est à dire à quels sous-programmes se brancher pour exécuter les tâches demandées. Cette double recherche s'effectue à chaque opération.

- Un compilateur compile, c'est à dire traduit, en une seule fois tout le programme basic (ou programme source) créé à l'aide de l'éditeur en une suite d'appels à des sous-programmes en langage machine qui, lorsqu'ils seront exécutés, donneront un résultat identique à celui de l'interpréteur.

Le gain de temps est obtenu en supprimant durant la phase d'exécution :

1) l'analyse syntaxique,
2) la recherche des erreurs.

Il faut savoir qu'une fois compilé, le programme basic est entièrement en codes machine.
On peut modifier ou même supprimer le programme basic, cela n'aura aucune influence sur l'exécution du programme compilé. En contrepartie, le code machine ne pourra plus être remodifié directement.
La séquence, introduction d'un programme source, tests puis compilation doit être de nouveau exécuté.
Nous allons faire, à la suite de ce chapitre, une introduction au vocabulaire utilisé avec les compilateurs et décrire la procédure à suivre pour développer correctement un programme.

2-1) VOCABULAIRE :

Bien que ce manuel évite d'utiliser un langage trop technique, les termes suivants doivent être bien compris :

- programme source (source file) : le programme créé avec l'éditeur basic est appelé "source" parce que c'est à partir de lui qu'un programme machine équivalent sera créé.

- code objet (objet file) : le compilateur traduit des programmes source en codes machine.
Le code objet est un programme binaire du même type que les routines appelées par les instructions basic : EXEC et USR.

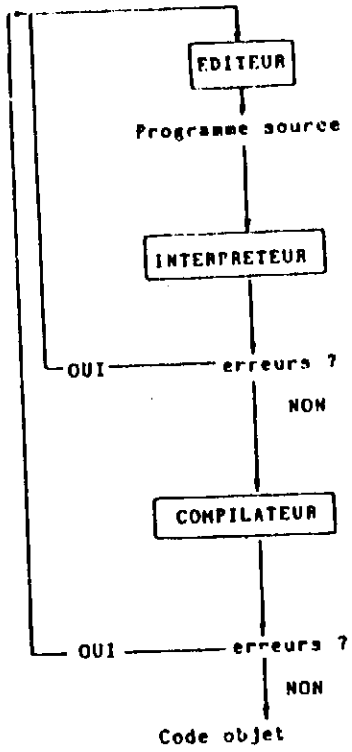
- phase de compilation (compiletime) : c'est la période pendant laquelle le compilateur traduit le programme source en code objet.

- phase d'exécution (Runtime) : c'est la période durant laquelle le programme compilé est exécuté.

- bibliothèque des sous programmes (runtime library) : la bibliothèque comporte un certain nombre de sous-programmes en langage machine qui sont utilisés par le programme compilé. Ces sous-programmes permettent la réduction de la place requise pour le code objet.

2-2) DEVELOPPEMENT D'UN PROGRAMME :

Voici le schéma général que vous devez suivre pour la création d'un programme avec compilateur :



1- Création du source

2-Tests et détection des erreurs avec l'interpréteur

3-Compilation

4-Exécution du programme compilé

Cette prodédure est à suivre scrupuleusement, surtout pour les programmes longs et complexes car le code issu de la compilation étant en langage machine, obéira aveuglément à tous vos ordres même suicidaires !

La mise au point des programmes avec l'interpréteur vous permet d'une part, de découvrir les erreurs de logique et, d'autre part, de tester ses différentes parties et de contrôler la valeur des variables après ou pendant l'exécution.

3 - INTRODUCTION A SPEEDY WONDER

Pour mieux comprendre le fonctionnement d'un compilateur et apprécier la rapidité de SPEEDY WONDER, tapez le petit programme qui suit :

```
10 CLS:FOR A=7999 TO 0 STEP-2
20 POKE A,255
30 NEXT
40 PLAY"DORE":END
```

Tapez COMP [ENTREE].

L'écran est effacé et vous avez l'affichage :

```
END OF COMPILATION          (MO5)          (TO7 à TO9)

OBJECT FILE > END ADDRESS:19068        34071 — 40071 (avec DOS)
BYTES FREE : ---------->08323                  9736
OK
```

END ADDRESS est l'adresse de fin du code objet qui résulte de la compilation du programme basic d'origine.

BYTES FREE correspond aux nombres d'octets restant libres pour le code objet.

Pour avoir la mémoire restant disponible pour le programme source, vous pouvez utiliser la fonction basic FRE ().

L'adresse de début du programme compilé n'est pas figée, elle dépend de l'adresse limite utilisée par l'interpréteur.
Cette limite, déterminée par l'instruction CLEAR, ADRESSE , est fixée au chargement à :

- (MO5) 19000 - ou 23999 (version DRIVE ou QDD et DOS)

- (TO7 à TO9) 34000 - ou 40000 (version DRIVE ou QDD et DOS)

Cela signifie que l'interpréteur basic ne peut pas utiliser les adresses supérieures à 19000 (MO5) et 34000 (TO7 à TO9).

Voici le schéma général de la répartition mémoire entre l'interpréteur et compilateur :

DEBUT DE LA MEMOIRE UTILISATEUR :

- (MO5) 8704 - (TO7 à TO9) 24832

FIN DU BASIC INTERPRETE : ADRESSE, fixée à :

- (MO5) 19000 ou 23999 (version DRIVE ou QDD et DOS)

- (TO7 à TO9) 34000 ou 40000 (version DRIVE ou QDD et DOS)

et modifiable avec CLEAR, adresse.

- (MO5) ADRESSE + 1 - (TO7 à TO9) ADRE

FIN DU BASIC COMPILE :

- (MO5) 27392 - (TO7 à TO9)

L'utilisateur en utilisant CLEAR, ADRESSE, avec adresse < (MO5) ou 43775 (TO7 à TO9), peut modifier l'implantation du objet ce qui permet d'avoir simultanément en mémoire plusie programmes compilés.
La mémoire restant libre après la fin du code objet jus 27392 (MO5) ou 43775 (TO7 à TO9), peut éventuellement utilisée pour l'implantation des programmes en langage mach créés par l'utilisateur.

Revenons maintenant à l'exécution du programme.

Tapez : BRUN [ENTREE] . Temps d'exécution : 1"2

[RAZ] puis RUN [ENTREE] . Temps d'exécution : 1"1.6

La différence de rapidité en faveur du compilateur est impressionante.

Si vous désirez voir comment le programme basic (source) traduit en langage machine, utilisez la commande BLIST.

Tapez BLIST [ENTREE] . Notez que vous pouvez à tout mome arrêter le listing en appuyant sur [STOP] ou le quitter en utilisant CNT-C.

BLIST [ENTREE] .

Vous avez à l'écran :

```
VERSION MO5 :

10 CLS : FOR A=7999 TO 0 STEP-2  (basic source)
**
4A3C JSR   CLS   (routine CLS de SPEEDY WONDER)
4A3F TFR PC,X
4A41 BRA 0C
4A43 FCB 00
4A44 FCB 8B
4A45 FCB 01


CNT-C
Break
OK
```

3 - INTRODUCTION A SPEEDY WONDER

Pour mieux comprendre le fonctionnement d'un compilateur et apprécier la rapidité de SPEEDY WONDER, tapez le petit programme qui suit :

```
10 CLS:FOR A=7999 TO 0 STEP-2
20 POKE A,255
30 NEXT
40 PLAY"DORE":END
```

Tapez COMP [ENTREE].

L'écran est effacé et vous avez l'affichage :

```
END OF COMPILATION          (MO5)          (TO7 à TO9)

OBJECT FILE > END ADDRESS:19068       34071 — 40071 (avec DOS)
BYTES FREE : ---------->08323              9736
OK
```

END ADDRESS est l'adresse de fin du code objet qui résulte de la compilation du programme basic d'origine.

BYTES FREE correspond aux nombres d'octets restant libres pour le code objet.

Pour avoir la mémoire restant disponible pour le programme source, vous pouvez utiliser la fonction basic FRE ().

L'adresse de début du programme compilé n'est pas figée, elle dépend de l'adresse limite utilisée par l'interpréteur. Cette limite, déterminée par l'instruction CLEAR, ADRESSE , est fixée au chargement à :

- (MO5) 19000 - ou 23999 (version DRIVE ou QDD et DOS)

- (TO7 à TO9) 34000 - ou 40000 (version DRIVE ou QUD et DOS)

Cela signifie que l'interpréteur basic ne peut pas utiliser les adresses supérieures à 19000 (MO5) et 34000 (TO7 à TO9).

Voici le schéma général de la répartition mémoire entre l'interpréteur et compilateur :

DEBUT DE LA MEMOIRE UTILISATEUR :

- (MO5) 8704 - (TO7 à TO9) 24832

FIN DU BASIC INTERPRETE : ADRESSE, fixée à :

- (MO5) 19000 ou 23999 (version DRIVE ou QDD et DOS)

- (TO7 à TO9) 34000 ou 40000 (version DRIVE ou QDD et DOS)

et modifiable avec CLEAR, adresse.

- (MO5) ADRESSE + 1 - (TO7 à TO9) ADRESSE + 1

FIN DU BASIC COMPILE :

- (MO5) 27392 - (TO7 à TO9) 43775

L'utilisateur en utilisant CLEAR, ADRESSE, avec adresse < 27392 (MO5) ou 43775 (TO7 à TO9), peut modifier l'implantation du code objet ce qui permet d'avoir simultanément en mémoire plusieurs programmes compilés.
La mémoire restant libre après la fin du code objet jusqu'à 27392 (MO5) ou 43775 (TO7 à TO9), peut éventuellement être utilisée pour l'implantation des programmes en langage machine créés par l'utilisateur.

Revenons maintenant à l'exécution du programme.

Tapez : BRUN [ENTREE] . Temps d'exécution : 1"2

[RAZ] puis RUN [ENTREE] . Temps d'exécution : 11".6

La différence de rapidité en faveur du compilateur est impressionante.

Si vous désirez voir comment le programme basic (source) a été traduit en langage machine, utilisez la commande BLIST.

Tapez BLIST [ENTREE] . Notez que vous pouvez à tout moment arrêter le listing en appuyant sur [STOP] ou le quitter en utilisant CNT-C.

BLIST [ENTREE].

Vous avez à l'écran :

```
VERSION MO5 :

10 CLS : FOR A=7999 TO 0 STEP-2  (basic source)
**
4A3C JSR   CLS   (routine CLS de SPEEDY WONDER)
4A3F TFR PC,X
4A41 DRA 0C
4A43 FCB 00
4A44 FCB 8B
4A45 FCB 81
      .
      .
      .
CNT-C
Break
OK
```

VERSIONS TO7 à TO9 :

```
10 CLS : FOR A=7999 TO 0 STEP-2   (basic source)
&&
84D7 JSR   CLS    (routine CLS de SPEEDY WONDER)
84DA TFR PC,X
84DC BRA OC
84DE FCB 00
84DF FCB 88
84E0 FCB B1
```

```
CNT-C
Break
OK
```

Tapez maintenant NEW [ENTREE], puis BRUN [ENTREE]. Vous constatez que le programme compilé est toujours là et s'exécute toujours aussi rapidement.

Ecrivons maintenant un petit programme qui affiche 100 fois les caractères ASCII de 32 à 127.

Tapez [RAZ] puis introduisez le programme suivant :

```
10 FOR A=1 TO 100
20 FOR B=32 TO 127
30 IF B=127 THEN PRINT:PRINT
40 PRINT CHR$(B);
50 NEXT:NEXT
60 PLAY"SI":END
```

COMP [ENTREE], [RAZ] puis BRUN [ENTREE].
Temps d'exécution du programme : 53".8
Lancez maintenant le programme basic en introduisant :

CLS:RUN [ENTREE]

Temps d'exécution : 2'10"

En résumé de ce qui a été dit jusqu'ici, il y a trois nouvelles commandes qui vous permettent de compiler, lister ou exécuter un programme :

COMP : compilation du programme basic en mémoire avec implantation du code objet à partir de la première adresse libre après le basic interprété.

BLIST : listage de chaque ligne du programme basic en mémoire, suivie du code équivalent en langage machine (l'utilisation de cette commande sans programme source en mémoire est déconseillée).

BRUN : lance l'exécution du dernier programme compilé.

Notez que la commande BRUN (non programmable) n'est pas équivalente à RUN car elle n'efface pas les variables antérieurement utilisées. Pour avoir une remise à zero des variables du compilateur, vous devez utiliser à l'intérieur du programme compilé, l'instruction CLEAR.

L'instruction RUN peut, par contre, être utilisée dans un programme compilé. Elle fonctionne de façon identique à l'instruction RUN de l'interpréteur.

Pour sauvegarder le programme source courant sur cassette ou disquette, vous utilisez, comme avec le basic interprété, SAVE "nom du programme. BAS".

La sauvegarde du code objet (programme compilé) se fera par :

SAVEM"nom.COM", ADRESSE1,ADRESSE2,0.

ADRESSE 1 : adresse du dernier CLEAR, adresse +1.
ADRESSE 2 : adresse de fin du programme compilé courant.

Vous pouvez avoir le début du programme objet courant par :

- (MO5) -
```
PRINT PEEK(&H75B6)*256+PEEK(&H75B7)
```

- (TO7 à TO9) -
```
PRINT PEEK(&HB55D)*256+PEEK(&HB55E)
```

Vous obtiendrez la fin du programme compilé courant par :

- (MO5) -
```
PRINT PEEK(&H79DB)*256+PEEK(&H79DC)
```

- (TO7 à TO9) -
```
PRINT PEEK(&HAB20)*256+PEEK(&HAB21)
```

Vous avez la possibilité de pouvoir utiliser le programme objet sans que le compilateur soit en mémoire.
Pour ce faire, vous devez sauvegarder avec votre code objet la bibliothèque des sous-programmes par :

- (MO5) -
```
SAVEM"LIBRARY.BIN",&H8220,&H8B6F,0
```

- (TO7 à TO9) -
```
SAVEM"LIBRARY.BIN",&HC220,&HCB80,0
```

La bibliothèque (RUN "THE LIBRARY) est indispensable au code objet car c'est elle qui contient les routines nécessaires à l'exécution correcte du programme compilé.

IMPORTANT :

1) les routines de la bibliothèque de SPEEDY WONDER ne peuvent fonctionner correctement qu'avec certains registres du 6809, des adresses de la page 0 du basic et les paramètres fournis par le code objet dans un état déterminé. Leur utilisation donc, au clavier ou à partir du basic interprété, est de fait fortement déconseillée.

2) Si dans votre programme compilé vous utilisez l'instruction RUN, la présence du compilateur est INDISPENSABLE.

3) Avant chargement pour exécution de votre code objet par LOADM, vous devez repartir manuellement ou par programme de la mémoire entre interpréteur et langage machine en utilisant l'instruction CLEAR, adresse.

4 - COMPARAISON INTERPRETEUR-COMPILATEUR SPEEDY WONDER

SPEEDY WONDER A ET CONCU POUR ETRE COMPATIBLE AVEC LE BASIC 1.0 DE MICROSOFT.

CELA SIGNIFIE QUE TOUT PROGRAMME ECRIT SOUS SPEEDY WONDER SERA ACCEPTE ET EXECUTE PAR L'INTERPRETEUR RESIDENT.

A CAUSE DES LIMITATIONS IMPOSEES PAR LA CAPACITE DE LA MEMOIRE CENTRALE ET LA NATURE DES COMPILATEURS, L'INVERSE N'EST PAS TOUJOURS VRAI.
CELA SIGNIFIE QU'UN CERTAIN NOMBRE DE PROGRAMMES ECRITS POUR LE BASIC INTERPRETE DOIVENT ETRE REMANIES AVANT DE POUVOIR ETRE COMPILES ET CORRECTEMENT EXECUTES PAR SPEEDY WONDER.

Vous allez trouver, à la suite, les principales différences entre l'interpréteur et le compilateur.

4-1) COMMANDES DE L'INTERPRETEUR ET DU COMPILATEUR

- Commandes du basic interprété :

CONT, DELETE, LIST, MOTOROFF, MOTORON, NEW, TRON, TROFF

Ces commandes ne sont pas compilables et provoquent le message d'erreur : "IMMEDIAT COMMAND IN...". Sont également considérées comme commandes directes, les instructions suivantes :

SAVE, SAVEM, FRE, MERGE

- Commandes de SPEEDY WONDER :

BRUN, COMP, BLIST, DOS (avec QDD uniquement)

Les commandes BRUN, COMP et BLIST ne sont pas compilables et provoquent le message d'erreur : "SYNTAX ERROR IN...". La commande DOS n'est pas compilable et provoque le message "ILLEGAL STATEMENT IN...".

4-2) INSTRUCTIONS ET FONCTIONS NON IMPLANTEES

CINT, CLOSE, COS, DEFINT, DEFSNG, DEFSTR, DIM, ERL, ERR, ERROR, EOF, EQV, EXP, FIX, IMP, INSTR, INT, LOG, NOT, OPEN, RESUME, SIN, SQR, STR$, TAN, VARPTR, TAB, USING, SPC()

Ces instructions ou fonctions provoquent le message d'erreur : "ILLEGAL STATEMENT IN...".

Sont également non implantées, toutes les fonctions ou instructions du DOS. Ces instructions ou fonctions provoquent le message "SYNTAX ERROR IN...".

END

Cette instruction arrête l'exécution du programme compilé et rend la main à l'interpréteur.
ELLE DOIT ETRE UTILISEE OBLIGATOIREMENT POUR SIGNALER LA FIN DU CODE OBJET.

Pour les programmes compilés que vous voulez inclure dans un programme basic, utilisez RETURN au lieu de END. (Voir les exemples dans l'annexe 1)

FOR...TO...STEP:NEXT

La syntaxe générale de FOR est la suivante. Notez que les mots entre crochets sont optionnels.

FOR [variable numérique non indicée] = [variable numérique non indicée]

TO

[variable numérique non indicée]

STEP

[variable numérique non indicée ou constante]

NEXT

L'imbrication des boucles avec SPEEDY WONDER n'est pas indéterminée comme avec l'interpréteur. L'imbrication maximum doit s'arrêter au 6e niveau.
L'instruction NEXT doit être utilisée sans variable mais, si ce n'est pas le cas, SPEEDY WONDER arrêtera la compilation en affichant : "FOR/NEXT ERROR IN...".

FOR

La boucle FOR...NEXT compilée se termine avec une valeur de variable égale à la limite.

ex. : FOR I=1 TO 9 : PRINT I : NEXT
Imprime les valeurs de 1 à 9, mais la variable I = 9

RND

La version compilée génère un nombre aléatoire entier de 0 à 255.
Pour différencier la situation basic de la situation compilée, vous pouvez tester si vous êtes sous basic ou non.

Si B est < 9636, vous êtes en compile.
Si B est > 9636, vous êtes sous basic.

Exemple (T07 à T09) : B=PEEK(25017)*256+PEEK(25018)

Si B est < 26145, vous êtes en compilé.
26101 pour T07/T0

Si B est > 26145, vous êtes sous basic.
26101 pour T07/T0

DATA

Chaque instruction DATA peut être suivie d'un nombre quelconque de constantes numériques ou de constantes chaine.
TOUTEFOIS, ELLE NE DOIT ETRE SUIVIE, SUR LA LIGNE COURANTE, D'AUCUNE AUTRE INSTRUCTION OU FONCTION.

READ

L'instruction READ n'admet qu'une seule variable numérique ou alphanumérique à la fois (variables obligatoirement non indicées).

RUN

L'instruction RUN du compilateur est non paramétrable. Elle fonctionne de manière identique à l'instruction RUN de l'interpréteur, mise à zero des variables, remise à jour de la pile des retours de sous-programmes, lancement de l'exécution du programme compilé.

IF... THEN

La syntaxe générale de IF est la suivante (les mots entre crochets sont optionnels) :

IF variable ou expression dyadique logique [AND / OR / XOR] variable ou expression dyadique logique

THEN GOTO nO de ligne
ou THEN GOSUB nO de ligne
ou THEN suite d'instructions

Le paramètre ELSE n'est pas compilé.

IF. .THEN GOTO...

En cas de branchement, il n'est pas permis d'indiquer THEN 520 ; le GOTO nO de ligne est obligatoire.

ON VARIABLE {GOSUB / GOTO

- admet les valeurs de 0 à 255
- on se débranche au 1er numéro de la ligne avec la valeur 0, au 5e avec la valeur 6 (alors qu'en basic 5 débranche au 5e numéro indiqué).

Pour pouvoir être compatible, vous pouvez utiliser la technique suivante :

```
- (MO5) -
INPUT A
B=PEEK(8627)*256+PEEK(8628)
IF B < 9636 THEN A=A-1
ON A GOTO 30,40,50
END
30 PRINT "30":GOTO 10
40 PRINT "40":GOTO 10
50 PRINT "50":GOTO 10

- (TO7 à TO9) -
INPUT A
B=PEEK(25017)*256+PEEK(25018)
IF B < 26145 THEN A=A-1   ou   26101 (TO7/70)
ON A GOTO 30,40,50
END
30 PRINT "30":GOTO 10
40 PRINT "40":GOTO 10
50 PRINT "50":GOTO 10
```

LINE
BOX
BOXF
PSET

n'acceptent pas le paramètre couleur, avant, il faut utiliser COLOR x, y

COLOR

Il faut préciser les deux paramètres Forme et Fond.

ABS
VAL
COLOR

n'acceptent pas une variable indicée en paramètre.

RESTORE

RESTORE [nO de ligne obligatoire] doit être utilisée, comme avec l'interpréteur, pour pointer sur la première donnée d'une suite des DATA. RESTORE doit précéder READ dans le Basic.NOTEZ QU'AVEC LE BASIC COMPILE, IL N'Y A PAS DE RESTORE PAR DEFAUT.

STOP

Cette instruction provoque l'arrêt de l'exécution du programme compilé et l'affichage de "BREAK".
CONTRAIREMENT A L'INTERPRETEUR, L'EXECUTION NE PEUT PAS ETRE REPRISE.

LOAD, LOADM, SKIPF (uniquement avec la cassette)

Ces instructions sont utilisables comme dans le basic interprété mais obligatoirement sans descripteur de fichier. Elle s'applique donc au fichier suivant sur la cassette.

DEFGR$

Dans le cas du MO5, cette fonction est à utiliser de la même façon que DEFGR$ du basic interprété. (Avec les versions 107 à TO9, ne pas utiliser cette fonction sous peine de destruction du code objet courant).

Elle attend un argument obligatoirement numérique (0 à 9), DEFGR$ (0) à DEFGR$ (9), puis une suite de 8 constantes numériques (0 à 255) qui détermineront le caractère graphique à créer.

NOTEZ QUE LES CARACTERES GRAPHIQUES , CREES AVEC LA FONCTION DEFGR$ COMPILEE, N'ONT PAS A ETRE RESERVES COMME AVEC L'INTERPRETEUR.
D'AUTRE PART, DEFGR$ COMPILEE NE DEFINIT PLUS DE SIMPLES CARACTERES GRAPHIQUES, MAIS DES SPRITES OU LUTINS.

GR$

Cette fonction, utilisée de la même façon que GR$ du basic, n'affiche pas un caractère graphique mais un sprite préalablement défini avec DEFGR$.

CLEAR

Cette instruction doit être utilisée sans paramètres.

Son but est d'effacer toutes les variables du compilateur. Notez que les variables du compilateur n'ont aucun rapport avec les variables du même nom utilisées avec le basic interprété.

PLAY

Arguments (notes : DO, RE, MI, FA, SOL, LA, SI) de type constante chaine uniquement. Le tempo, l'octave, la longueur, sont ceux définis par défaut.

INPUT

L'instruction INPUT n'admet qu'une seule variable numérique ou alphanumérique (variables non indicées) à la fois. Sa syntaxe est la même qu'avec l'interpréteur.

INPUT variable ou INPUT "constante chaine" ; variable

INPUT "texte";variable

Seul le ";" est autorisé, pas la virgule qui n'affiche pas le ?

INPUT est la seule instruction qui permet d'arrêter l'exécution du programme compilé avec CNT-C. A l'exécution, INPUT variable numérique attend obligatoirement des chiffres allant de 0 à 9.

La fonction INPUT$ attend l'appui de 1 à 9 touches. (argument numérique 1 à 9).

Seulement 2 paramètres - le 3e, servant à l'effacement du curseur, n'est pas autorisé.
Utiliser pour cela PRINT CHR$( ) 20 ou 17

PRINT

La syntaxe générale de PRINT est la suivante :

```
PRINT   expression dyadique,
        numérique,
        logique,
        alphanumérique  ; expression ; expression ;...
```

- Les variables numériques ne sont pas comme en BASIC MO5, imprimées précédées d'une position à Blanc.

```
A=1:B=2
PRINT A;B  donne  12 (le 1 est en colonne 0)
```

- Les constantes alphanumériques doivent être obligatoirement terminées par ".

TOUTES LES FONCTIONS OU INSTRUCTIONS PARAMETRABLES ACCEPTENT COMME ARGUMENT, SOIT DES CONSTANTES, SOIT DES VARIABLES ADEQUATES NON INDICEES.

Les expressions avec les affectations des variables ont la forme générale suivante :

Variable num. = expression polyadique
(logique,arithmétique)

Variable alphanum. = expression dyadique alphanumérique

DANS LES EXPRESSIONS ARITHMETIQUES, LA PRIORITE DE CALCUL EST DE LA GAUCHE VERS LA DROITE.

4-4) VARIABLES . TABLEAUX

SPEEDY WONDER traite uniquement des variables entières et des variables chaine (capacité max. 50 caractères).

La répartition mémoire des variables est statistique et 3300 octets ont été réservés d'office.

Les variables A à W sont des variables entière non indicées. Les tableaux entiers X (), Y(), Z() n'ont pas à être dimensionnés et leur indice peut varier de 0 à 99.
Les variables A$ à W$ sont des variables chaine. Les tableaux chaine X$(), Y$(), Z$() n'ont pas à être dimensionnés et leur indice peut varier de 0 à 9.

DIM n'est pas une instruction compilée mais est considérée comme commentaire par SPEEDY WONDER pour des questions de

4-5) OPERATEURS LOGIQUES

Les opérateurs logiques permis avec les chaines, alphanumériques (variables ou constantes) sont = , < et > .

Les opérateurs logiques acceptés pour les variables numériques entières sont les mêmes que ceux du basic interprété à l'exception de IMP, EQV, NOT. (voir 4-2)

4-6) ARITHMETIQUE ENTIERE

Le basic interprété, même avec DEFINT et %, utilise toujours, pour calculer, la virgule flottante, puis, procède aux conversions nécessaires.

SPEEDY WONDER utilise une véritable arithmétique entière ce qui explique sa très grande rapidité dans l'exécution des opérations numériques ou logiques.
Tous les calculs avec SPEEDY WONDER se font modulo 65535 en mode signé.

Exemples :

a) 10 A=34*3/5/5+1-31000:PRINT"A=";A:END
   avec l'interpréteur vous aurez : - 30994.5
   avec SPEEDY WONDER vous aurez : - 30995

b) 10 A=65535:PRINT A : END
   avec l'interpréteur vous aurez : 65535
   avec SPEEDY WONDER vous aurez : -1

c) 10 A=34*3/5+1:PRINT A:END
   avec l'interpréteur vous aurez : 21.4
   avec SPEEDY WONDER vous aurez : 21

d) 5 A=0
   10 A=A XOR 255+11*5
   20 PRINT A
   30 END
   avec l'interpréteur vous aurez : 310
   avec SPEEDY WONDER vous aurez : 1330

Le basic effectue les calculs de la façon suivante (AOS) :
   11*5=55
   255+55=310
   0 XOR 310=310

Avec SPEEDY WONDER les calculs sont effectués de gauche à droite

   0 XOR 255=255
   255+11=266
   266*5=1330

Pour avoir le même résultat que l'interpréteur :

```
5 A=0
10 A=(1^5+255 XOR A
20 PRINT A
30 END
```

a) 10 PRINT 65535/-1
20 END

avec l'interpréteur vous aurez : - 65535
avec SPEEDY WONDER vous aurez : 1

5 - DIFFERENTES ROUTINES - COMPARAISON DES VITESSES D'EXECUTION

a) Tri à bulles (Bubble sort)

```
10  N=10:M=N:RESTORE  110:FOR  A=1  TO  10:READ  B:X(A)=B:NEXT
20 F=0
30 FOR I=2 TO M
40 J=I-1:IF X(J) <=X(I) THEN GOTO 60
50  X(J)=X(I)XOR  X(J):X(I)=X(I) XOR X(J):X(J)=X(I) XOR X(J):F=1
60 NEXT
70 IF M=2 THEN GOTO 90
80 IF F=1 THEN M=M-1:GOTO 20
90 FOR I=1 TO N:PRINT "X(";I;")=";X(I):NEXT
100 END
110 DATA 5,66,2,1,0,3,9,20,33,1988
```

Basic : temps d'exécution = 2".3
SPEEDY WONDER " : 0".4
Rapport : 1/5.75

b) POKE'S

```
10 CLS:FOR A=0 TO 7999
20 POKE A,255
30 NEXT
40 PLAY"S1":END
```

Basic : temps d'exécution = 34".5
SPEEDY WONDER " = 2".8
Rapport : 1/12.32

c) Boucle vide

```
10 FOR A=0 TO 20000
20 NEXT:PLAY"S1D0":END
```

Basic : temps d'exécution = 26".1
SPEEDY WONDER " = 3".3
Rapport : 1/7.9

d) Appel des sous-programmes

```
10 FOR A=1 TO 500:GOSUB 20:NEXT:PLAY"S1D0":END
20 GOSUB 30:RETURN
30 GOSUB 40
40 GOSUB 50
50 GOSUB 60
60 GOSUB 70
70 GOSUB 90
90 RETURN
```

Basic : temps d'exécution = 22"
SPEEDY WONDER " = 0".9
Rapport : 1/24.44

e) traitement des chaines

```
10 FOR B=1 TO 100:A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20  L=LEN(A$):FOR  A=1  TO  L:B$=MID$(A$,A,1):C$=B$+RIGHT$(A$,3)
30 NEXT:NEXT
40 END
```

Basic : temps d'exécution = 37".1
SPEEDY WONDER " = 9".8
Rapport : 1/7.72

f) Affichage et PEEK'S

```
10 CLS:FOR A=7999 TO 0 STEP-1
20 PRINT PEEK(A);"/";:NEXT
30 PLAY"S1":END
```

Basic : temps d'exécution = 9'30".9
SPEEDY WONDER " = 2'56".5
Rapport : 1/1.68

g) EXEC'S

```
10 FOR A=1 TO 1000
20 EXEC &B29:NEXT
30 PLAY"S1":END
```

Basic : temps d'exécution = 9".1
SPEEDY WONDER " = 0".3
Rapport : 1/13.66

h) Graphismes

```
10 CLS
20 FOR A=0 TO 199
30 LINE (0,A)-(319,A)
40 NEXT
50 END
```

Basic : temps d'exécution = 6".3
SPEEDY WONDER " = 2".0
Rapport : 1/2.15

i) Opérations arithmétiques

```
10 B=0:FOR A=1 TO 1000
20 B=B+A-1
30 NEXT
40 END
```

Basic : temps d'exécution = 8".0
SPEEDY WONDER " = 0".8
Rapport : 1/10

j) Opérations logiques

```
10 B=0:FOR A=1 TO 1000
20 B=B XOR A AND B OR A
30 NEXT
40 END
```

Basic : temps d'exécution = 9".1
SPEEDY WONDER " = 1".0
Rapport : 1/9.10

6 - RUN TIME LIBRARY

SPEEDY WONDER est essentiellement composé de deux parties :

- le compilateur proprement dit qui génère le code machine exécutable à partir du source basic,

- la bibliothèque des sous-programmes (RUN TIME LIBRARY) qui est activée pendant l'exécution du code objet. En fait, le code objet est constitué d'une suite de transferts de paramètres, de constantes et d'appels à la bibliothèque.

Quand vous utilisez la commande BLIST, vous voyez le listing du code objet avec :

- des JSR (instruction machine équivalente au GOSUB du basic)

- des TFR PC,X (transfert du compteur programme dans le registre X)

- des BRA, LBRA, JMP (branchements inconditionnels)

- des FCB (constantes)

Dans la suite de ce chapitre, nous allons vous présenter les différentes routines qui composent la bibliothèque.

AND : AND logique entre 2 entiers
ADD : addition signée entre 2 entiers
ATTR : ATTRB
BEEP : BEEP
BOX : BOX
BOXF : BOXF
CLEA : CLEAR. Sous-programme de remise à 0 des variables
CLS : CLS
COLO : COLOR
CONS : CONSOLE
DIF : routine d'évaluation de l'opérateur "<>" (entiers)

DIFS : routine d'évaluation de l'opérateur "<>" (chaines)

DIV : division signée entre 2 entiers
END : END retour sous contrôle de l'interpréteur
EQU : routine d'évaluation de l'opérateur "=" (entiers)

EQUS : routine d'évaluation de l'opérateur "=" (chaines)

EXE : EXEC
EXP : routine auxiliaire d'évaluation d'une expression

EXPV : routine d'évaluation d'une expression dyadique
FOR : FOR
GSB : GOSUB
GTO : GOTO
GRS : routine qui gère l'affichage des sprites
GE : routine d'évaluation de ">="
GT : routine d'évaluation de ">"
IF : IF

INPE : INPEN
INPF : INPUTPEN
INPU :INPUT
INPV :INPUT$()
JEXP : routine d'évaluation d'une expression polyadique
KEY : INKEY$
LD : LOAD
LDM :LOADM
LE : routine d'évaluation de "<="
LINE : LINE
LOC : LOCATE
LT : routine d'évaluation de "<"
MOD : MOD
MUL : multiplication signée entre 2 entiers
NEXT : NEXT
ODDA : routine auxiliaire de PRINT
ONG : ON....GOTO, GOSUB
OR : OR
PLAY : PLAY
POKE : POKE
PRI : routine générale PRINT
PRIT : routine d'affichage
PSET : PSET
PUNI : PUNIC
READ : READ
REST : RESTORE
ROUT : routine aux. d'évaluation d'une expression polyadique
RUN : RUN - Il met à 0 les variables, restaure la pile, relance le programme.

SCR : SCREEN
SCPR : SCREENPRINT
SKPF : SKIPF
STOP : STOP
STOV : routine aux. de FOR
SUB : soustraction signée entre 2 entiers
SYNE : routine aux.de IF
SYVA : routine aux. d'affectation d'une variable
TFVA : routine d'affectation d'une variable
TRT1 : 1re routine de traitement général
TRT2 : 2e routine de traitement général
TUNE : TUNE (MO5)
XOR : XOR

7 - COMMENT TRAVAILLE LE COMPILATEUR

Ce chapitre explique comment SPEEDY WONDER travaille et deve
rendre les opérations internes du compilateur moins
mystérieuses.
Toutefois, une compréhension de la façon dont SPEEDY WO
travaille n'est pas indispensable à son utilisation.

La principale tâche d'un compilateur est de traduire le programme source en langage machine.

Cette procédure peut être divisée en deux étapes :

1) Analyse syntaxique : reconnaissance des instruct
basic

2) Génération du code : production d'un code mac
équivalent au programme source d'origine

SPEEDY WONDER est un compilateur à deux passes.

La première passe opère une analyse syntaxique et génère la grande partie du code. En examinant le programme source première passe collecte l'information nécessaire pour l'affectation des variables et la construction de la table branchements.

La passe 2 utilise toutes les informations collectées pa passe 1 pour déterminer les adresses effectives des variable des branchements.

7-1) ANALYSE SYNTAXIQUE

La routine d'analyse syntaxique examine les différents générés par l'interpréteur ainsi que les variables e constantes.
La première analyse syntaxique opérée par l'interpréteur tr en codes (TOKENS) les instructions, fonctions et opéra basic.

Par exemple :

```
FOR I=A TO C

sera transformé en
129:token de FOR
I : variable I
212 : token de =
A : variable A
187 : token de TO
C : variable C
```

SPEEDY WONDER analyse donc ces "tokens" ainsi que les diff codes ASCII, fait les conversions nécessaires et stocke d code objet les codes 6809 appropriés.

7-2) GENERATION DU CODE

La génération du code se fait de deux façons différentes :

- s'il s'agit d'une instruction simple telle END, le compilateur génère directement le code objet approprié ; les instructions qui impliquent l'utilisation de paramètres sont traduites par stockage dans le code objet des différentes constantes ou adresses et des appels aux routines adéquates de la bibliothèque.

Pour ceux qui connaissent la programmation en langage machine 6809, nous donnons à la suite, les listings assembleur des routines qui génèrent les codes machines pour les constructions sans paramètre et pour GOTO, GOSUB et PLAT.

```
1)    INNER   CMPB    #$9A  LOAD
              BNE     SUIV
              JSR     <$AC  avance pointeur CHRGET (MO5)
              JSR     <$B2  avance pointeur CHRGET (TO7 à
                                                    TO9)
              CMPA    #'M  LOADM
              BEQ     LODM
              JSR     DECBJ  décrémente pointeur CHRGET
      SUIV    LDY     #CODE1
              STY     ADCOD+1
              BSR     STOCK  stockage codes
              BRA     TFIN
      LODM    LDY     POB  pointeur code objet
              LDA     #$BD
              STA     ,Y+
              LDD     #XLOADM  codes LOADM
              LBRA    COM2-2  vers stockage

2)            CMPA    #$BC  code de SUB
              BEQ     GSUB  vers traitement GOSUB
              CMPA    #$BB  code TO
              LBNE    ER2  erreur de syntaxe
              BSR     COM1  vers décodage n0 de ligne
              LDA     #$BD
              STA     ,Y+
              LDD     #IGOT  codes GOTO
              STD     ,Y++
      COM2    JSR     TMEM  test mémoire disponible
              JMP     TFIN  fin de stockage
      GSUB    BSR     COM1
              LDA     #$BD
              STA     ,Y+
              LDD     #IGOS
              STD     ,Y++
              BRA     COM2

3)    PLAT    CLR     ADRX
              CLR     ADRX+1
              LDX     #TF2  stockage codes
              JSR     STO2
              PSHS    Y  sauvegarde pointeur code objet
              LEAY    2,Y
              JSR     <$AC  avance CHRGET (MO5)
              JSR     <$B2  avance CHRGET (TO7 à TO9)
              CMPA    #'"
              LBNE    ER2  vers erreur de syntaxe
      PLAT1   JSR     <$AC décodage paramètres PLAT (MO5)
      PLAT1   JSR     <$B2 décodage paramètres PLAT (TO7
                                                    à TO9)
              PSHS    A
              JSR     <$AC (MO5)
              JSR     <$B2 (TO7 à TO9)
              TFR     A,B
              PULS    A
              CLR     ,-S  compteur
              INC     ,S
              INC     ,S
              LDX     #NOTE  tableaux notes
      LDXN    CMPD    ,X++
              BEQ     FOUND
              CMPX    #NOTEF  test fin tableau
              LBEQ    ER2
              INC     ,S
              INC     ,S
              BRA     LDXN
      FOUND   INC     ADRX+1
              LDA     ,S+
              STA     ,Y+
              JSR     TMEM  test mémoire disponible
              JSR     <$AC (MO5)
              JSR     <$B2 (TO7 à TO9)
              JSR     DECBJ
              CMPA    #'"  test fin chaine
              BNE     PLAT1
              LDA     #':RD
              STA     ,Y+
              LDD     #PLAT  stockage codes PLAT
              STD     ,Y++
              JSR     TMEM  test mémoire disponible
              LDD     ADRX
              STD     [,S++]
              JSR     <$AC (MO5)
              JSR     <$B2 (TO7 à TO9)
              JMP     TFIN
```

En résumé, tandis que l'interpréteur doit chercher, pendant l'exécution dans des listes de variables et des numéros de lignes pour s'orienter vers les routines appropriées de la ROM, le compilateur génère pendant la phase de compilation, des adresses effectives pour les numéros de lignes et les variables.

C'est en grande partie cette utilisation de l'adressage absolu qui explique la rapidité du compilateur par rapport à l'interpréteur.

8 - MESSAGES D'ERREUR

Pour expliquer le fonctionnement des messages d'erreur reprenons le programme de tri du chapitre 5 et introduisons quelques erreurs :

```
ligne 20 : F=0
supprimons la ligne 60
ligne 80 : M=M-1:CHR$(M)
```

Tapez COMP [ENTREE]

Vous avez à l'écran :

```
Compilation : Passe1
VARIABLE ERROR IN 00020
```

Tapez 20 F=0 puis recompilez.

Vous avez à l'écran :

```
Compilation : Passe2
LINE ERROR IN 00080
```

Tapez 60 NEXT, puis COMP [ENTREE]

Vous avez à l'écran :

```
Compilation : Passe1
BAD EXPRESSION IN 00080
```

Vous remarquez donc que, pour chaque erreur détectée durant phase de compilation, le compilateur affiche un message d'err et arrête la compilation.
La plupart des erreurs sont détectées durant la "Passe 1", p les branchements à des lignes inexistantes qui ne sont détect qu'à la "Passe 2".

IMPORTANT :

APRES UN QUELCONQUE MESSAGE D'ERREUR, NE FAITES JAMAIS UN

Passez en mode éditeur, corrigez, testez le déroulement programme avec l'interpréteur, corrigez les autres erre éventuelles, compilez puis , utilisez la commande RUN p exécuter le code objet.

Les messages d'erreur sont, en général, générés durant la p de compilation.

Toutefois, certaines erreurs indétectables pendant la compilation sont générées pendant l'exécution du code co

8-1) MESSAGES D'ERREUR PENDANT LA COMPILATION :

SYNTAX ERROR :	erreur de syntaxe ou de paramètre (introduction ou fonction non implantée, D.O.S.)
OUT OF MEMORY :	la mémoire disponible pour le stockage du code objet est insuffisante
VARIABLE ERROR :	erreur de variable
BAD EXPRESSION :	expression incorrecte ou non compilable
RESTORE ERROR :	le paramètre de RESTORE ne correspond pas à une ligne de DATA
LINE ERROR :	numéro de ligne inexistant
IMMEDIAT COMMAND :	commande non compilable, utilisable uniquement au clavier
ILLEGAL STATEMENT :	instruction ou fonction non implantée
FOR/NEXT ERROR :	-erreur de boucle (plus de 6 imbrications) -FOR sans NEXT correspondant -NEXT sans FOR correspondant
ARRAY ERROR :	erreur de variable indicée
INDEX TOO LARGE :	indice en dehors des limites

8-2) MESSAGES D'ERREUR PENDANT L'EXECUTION :

(MO5)	(TO7 à TO9)	
ERROR 6	OV ERROR	nombre trop grand dans un calcul
ERROR 11	/0 ERROR	division par zero
ERROR 15	LS ERROR	formule de chaine trop complexe
ERROR 51	FM ERROR	mode d'accès au fichier incorrect
ERROR 53	IO ERROR	erreur sur entrée/sortie
ERROR 56	DS ERROR	commande directe dans un fichier en cours de chargement
ERROR 59	LV ERROR	périphérique occupé
ERROR 60	DV ERROR	périphérique absent ou indisponible

9 - COMMANDES, INSTRUCTIONS ET FONCTIONS DE SPEEDY WONDER

9-1) COMMANDES :

COMP : compilation
RRUN : lancement du code objet
RLIST : listage du programme compilé

Dans le cas du MO5, uniquement :
DOS : avec le QDD, avec le DRIVE, initialisation et chargement de SPEEDY WONDER

9-2) INSTRUCTIONS ET FONCTIONS :

";$+-*<=>/?

ABS	NEXT
AND	ON...GOSUB
ASC ()	ON...GOTO
ATTRB	OR
BEEP	PEEK
BOX	PLAY
BOXF	POINT()
CHR$	POKE
CLEAR	POS
CLS	PRINT
COLOR	PSET
CONSOLE	PTRIG
CSRLIN	READ
DATA	REM
DEFGR$ (MO5)	RESTORE
END	RETURN
EXEC	RIGHT$
FOR...TO...STEP	RND
GOSUB	RUN
GOTO	SCREEN
GR$	SCREEN()
IF...THEN	SCREENPRINT
INKEY$	SGN()
INPUT	SKIPF
INPUT$()	STOP
INPUTPEN	TUNE (MO5)
INPEN	VAL()
LEFT$()	XOR
LEN()	
LINE	
LOAD	
LOADM	
LOCATE	
MID$	
MOD	

10 - CONFIGURATION MEMOIRE SOUS SPEEDY WONDER

Version MO5 :

DECIMAL	HEXA	
0 - 8191	$0 - $1FFF	ECRAN
8192 - 8447	$2000 - $20FF	Page 0 moniteur
8448 - 8703	$2100 - $21FFF	Page 0 Basic + compil

Selon configuration et jusqu'à $6R00 DOS, BASIC etc.

27393 - 40959 $6R01 - $9FFF Compilateur, Bibliothè et adresses réservées

Version TO7 à TO9 :

DECIMAL	HEXA	
24832 - 25087	$6100 - $61FF	Page Basic + compilate

Selon configuration et jusqu'à $AAFF DOS, BASIC etc

41808 - 57343 $AR20 - $DFFF Compilateur, Bibliothè et adresses réservées

A N N E X E 1

1 - COMPILATION DES LONGS PROGRAMMES

Grâce à l'instruction "CLEAR, adresse", vous pouvez scinder votre code objet en plusieurs parties qui seront implantées à des endroits différents.

Ces codes objet seront sauvegardés successivement sur la cassette ou la disquette.

Un programme basic interprété chargera l'exécution par les instructions : LOADM"nom." (ou LOADM"CAS:nom":):EXEC adresse.

CHAQUE CODE OBJET PARTICULIER DEVRA SE TERMINER NON PAS PAR L'INSTRUCTION END MAIS PAR RETURN.

Vous trouverez à la suite, des exemples de cette procédure :

a) CLEAR,26000 (MO5)
   CLEAR, 41000 (TO7 à TO9)

```
10 CLS:A=99:B=2:C=A:D=A+B+C:SCREEN 0,7,8
20 PRINT "1ER PROGRAMME :D=";D
30 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
CLEAR,25000 (MO5)
CLEAR,40000 (TO7 à TO9)

10 D=D-2.SCREEN 1,7,8
20 PRINT"2EME PROGRAMME:D=";D
30 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
CLEAR, 24000 (MO5)
CLEAR, 39000 (TO7 à TO9)

10 D=D-1:SCREEN 2,7,8
20 PRINT"3EME PROGRAMME:D=";D
30 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
CLEAR, 23000 (MO5)
CLEAR, 38000 (TO7 à TO9)

10 D=D+ASC("A"):SCREEN 4,7,8
20 PRINT"4E PROGRAMME:D=";D
30 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
      (MO5)      (TO7 à TO9)
10 EXEC 26000      41000
20 EXEC 25001      40000
30 EXEC 24001      39000
40 EXEC 23001      38000
50 SCREEN 3,0,0
```

RUN puis vous aurez à l'écran :

```
1ER PROGRAMME : D=200
2EME PROGRAMME : D=397
3EME PROGRAMME : D=401
4EME PROGRAMME : D=465
```

OR

b) CLEAR,25000 (MO5)
   CLEAR,40000 (TO7 àTO9)

```
10 CLS
20 FOR A=32 TO 127
30 PRINT CHR$(A);
40 NEXT:PRINT
50 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
CLEAR,24000 (MO5)
CLEAR,39000 (TO7 à TO9)

10 FOR B=A TO 32 STEP-1
20 PRINT CHR$(B);
30 NEXT
40 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
CLEAR,23000 (MO5)
CLEAR,38000 (TO7 à TO9)

10 FOR C=A TO 129
20 PLAY"DO":PRINT C
30 RETURN
```

COMP [ENTREE]
NEW [ENTREE]

```
      (MO5)          (TO7 à TO9)
10 EXEC 25001      10 EXEC 40004
20 EXEC 24001      20 EXEC 39004
30 EXEC 23001      30 EXEC 38004
40 END             40 END
```

RUN et vous aurez à l'écran les caractères ASCII 32 à 127 affichés dans l'ordre croissant puis décroissant, puis l'affichage de :

```
127
128
129

OK
```

2 - APPEL D'UN PROGRAMME COMPILE A PARTIR D'UN PROGRAMME INTERPRETE :

Vous avez, également, la possibilité de compiler un programme ou une partie de programme nécessitant une vitesse d'exécution élevée.
Ce programme, comme dans les exemples précédents, sera lu comme un sous-programme en langage machine par l'interpréteur et sera appelé par " EXEC, adresse " comme un programme machine ordinaire.

N'OUBLIEZ PAS, DANS LE CAS OU VOUS EXECUTEZ UN PROGRAMME COMPILE SANS LE COMPILATEUR, VOUS DEVEZ AU PREALABLE AVOIR SAUVEGARDE LA BIBLIOTHEQUE DE SOUS-PROGRAMMES ET VOUS DEVEZ RECHARGER AVEC VOTRE PROGRAMME (voir 3).

A N N E X E 2

POUVEZ RENCONTRER AVEC LES PROGRAMMES

tés qui s'automodifient

en basic interprété obtiennent des effets difiant durant l'exécution.

URNERONT PAS CORRECTEMENT UNE FOIS COMPILES.

enir une exécution correcte de ces iliser des méthodes plus classiques pour les

cte de l'instruction CLEAR, adresse

à des adresses différentes d'un ou plusieurs , la détermination de la limite Basic/Code ction CLEAR, doit commencer par l'adresse la 'éviter la destruction d'une partie du code stème.

implantation d'un 1er programme
TO7 à TO9)

implantation d'un 2e programme
TO7 à TO9)

implantation d'un 3e programme
TO7 à TO9)

mpilateur ou de la bibliothèque

commande BRUN avec un programme compilé non é avec l'interpréteur ou comportant des bles pendant la compilation, peut, parfois, ction d'une ou plusieurs parties du a bibliothèque.

it, sauvegardez le programme source, éteignez e et rechargez le compilateur.

A N N E X E 3

PRECOMPILATEUR

Comme indiqué au chapitre 4-4, SPEEDY WONDER ne traite que des variables dont le nom ne comprend qu'un caractère.

Pour vos programmes existants, il serait fastidieux de modifier manuellement chacune des variables.

Le Précompilateur est un outil qui le fera à votre place.

Version cassette :

Le Précompilateur se trouve à la suite du programme "DEMO1.BAS" sur la cassette.

Tapez RUN"PRECOMP.BAS" pour le charger.

Version disquette ou Q.D.D. :

Le Précompilateur se trouve sur la disquette.

Tapez RUN"PRECOMP.BAS" pour le charger.

Vous pouvez alors charger un programme avec LOAD"...Pour modifier les variables, tapez PRECOMP, le précompilateur vous indique TERMINE si tout s'est bien passé.
Vous pourrez vérifier le résultat en tapant LIST.

Sauvegardez votre programme modifié par SAVE.

CODES ERREUR PENDANT L'UTILISATION DE "PRECOMP" :

Affichés sous la forme ERREUR x LIGNE 00000, x peut prendre les valeurs suivantes :

2 : rencontre de plus de 3 tableaux numériques ou type chaine ou de plus de 26 variables numériques ou type chaine.

4 : un tableau à deux niveaux a été rencontré.

6 : indice d'un tableau plus grand que permis (9 pour variables chaine, 99 pour variables entières).

8 : formule arithmétique trouvée en indice de tableau.

Aucun contrôle de syntaxe n'est effectué, pas plus que d'autres contrôles ni substitution.

Nota : La longueur des noms de variables prise en compte est limitée à 15 caractères.

SPEEDY WONDER
THOMSON - BASIC VERSION 1.0
TABLE DES MATIERES

1-1) CHARGEMENT :

A) Q.D.D. ou lecteur de disquettes :

Insérez la disquette, puis,

- (MO5) tapez DOS
- (TO7) choisissez l'option 2
- (TO9) choisissez l'option E

Vous pouvez choisir soit de charger le compilateur plus le DOS (ou le QDOS) basic, soit le compilateur seul.
Dans le premier cas, les instructions BACKUP et COPY ne sont pas disponibles. Leur utilisation provoque les messages d'erreur suivants :

- (MO5) respectivement "ERROR 30" et "ERROR 32"
- (TO7 à TO9) SN ERROR

L'instruction DIR est également non disponible et son utilisation peut provoquer une perturbation du système.

B) Lecteur de cassettes :

Introduisez la cassette, puis tapez RUN"

SI VOTRE COMPILATEUR EST SUR CASSETTE, OU SI ETANT SUR DISQUETTE (OU Q.D.D.), VOUS AVEZ CHOISI DE LE CHARGER SANS LE DOS OU QDOS), LES PROGRAMMES BASIC A COMPILER SERONT LUS SUR CASSETTES.

1-2) PROGRAMME DE DEMONSTRATION :

VOUS AVEZ, AVEC LE COMPILATEUR, UN PROGRAMME BASIC : DEMO1.BAS.

SI vous désirez essayer tout de suite votre compilateur, tapez LOAD"DEMO1". Si vous avez la version disquette et que vous avez chargé le compilateur sans le DOS (ou le QDOS), le programme "DEMO1" ne pourra pas être chargé. Ce programme de jeu vous permettra d'aprécier la rapidité de SPEEDY WONDER. Une fois le programme chargé, tapez :

COMP [ENTREE] , permet de compiler le programme basic.

BRUN [ENTREE] , lance l'exécution du programme compilé.

Il s'agit d'un petit programme de casse briques.

Déplacez votre raquette avec les touches [←] et [→]. Pour refaire une nouvelle partie, appuyez sur n'importe quelle touche. Pour quitter le jeu, utilisez CTN-C. Pour ralentir la vitesse d'exécution du programme, insérez des instructions GOSUB vers le sous-programme de saisie C$=INKEY$.

Vous pouvez constater l'énorme différence de rapidité entre compilateur et basic interprété en tapant :

RUN ENTREE . Vous exécutez alors le programme basic