

Le compilateur Basic  
\*\*\*\*\*

Ce compilateur référencé COMBASIC peut vous être fourni sous forme d'une disquette 40 ou 80 pistes, 5 pouces ou 3 pouces. La disquette supporte un seul fichier baptisé COMBASIC.COMD qui est le compilateur lui-même.

Rappelons qu'un compilateur, par opposition à un interpréteur, traduit lors de la phase dite de compilation, le langage évolué qui est ici le Basic en code machine directement exécutable par le micro-processeur. En d'autres termes, un programme Basic passé au compilateur est transformé en du code hexadécimal, que vous pouvez ensuite charger en mémoire et faire fonctionner alors que le compilateur n'est plus présent sur la machine. Les avantages majeurs du compilateur sur l'interpréteur sont une occupation mémoire beaucoup moins importante, puisque le compilateur n'a pas à être présent en mémoire lors de l'exécution du programme, et une rapidité d'exécution des programmes qui est de 3 à 10 fois supérieure à celle du même programme exploité avec un interpréteur.

L'utilisation du compilateur est très simple et peut globalement se décomposer comme suit :

- Edition de votre programme Basic au moyen de l'éditeur du DOS par exemple conformément à la syntaxe de COMBASIC présentée ci-après.
- Essai éventuel de ce programme avec le Basic disquette moyennant certaines restrictions dues aux différences entre ces deux programmes.
- Compilation proprement dite conduisant à la production d'un listing de compilation, d'un pseudo listing source d'assemblage, et du programme objet proprement dit qui est ensuite immédiatement chargeable en mémoire.

Le format de la ligne d'appel est du même type que ce que vous avez l'habitude de voir pour les commandes du DOS normales :

- COMBASIC SOURCE OBJET + OPTIONS OU COMBASIC, SOURCE, OBJET, + OPTIONS, puisque le séparateur peut être un espace ou une virgule. SOURCE est le nom du fichier contenant la source de votre programme, dest-à-dire le texte Basic de celui-ci. OBJET est facultatif. Si ce nom est spécifié le compilateur placera dans ce fichier le code obtenu après compilation; ce fichier aura donc généralement un suffixe BLN ou CMD. Aucune valeur par défaut n'est choisie pour ce suffixe et c'est à vous de le spécifier.

Si OBJET n'est pas spécifié, le compilateur ne produira qu'un listing (c'est très utile pour voir les éventuelles\_erreurs de compilation avant de faire réaliser le fichier OBJET).

OPTIONS sont des lettres qui permettent de spécifier diverses options de compilation :

- L interdit la sortie du listing de compilation.
- S génère la table des symboles, c'est-à-dire la liste de toutes

les variables et constantes symboliques utilisées dans le programme Basic avec leurs implantations en mémoire.  
– H fait imprimer sur le listing, après chaque ligne Basic, le code hexadécimal généré, ce qui vous permet de voir comment le compilateur a codé les diverses instructions.  
Si plusieurs options sont désirées, il suffit de mettre les lettres dans n'importe quel ordre les unes derrière les autres sans signe ou espace entre elles.  
Dans tous les cas, sauf SI vous ne voulez pas de listing bien sûr, une cartographie mémoire du programme compilé est affichée indiquant son adresse de fin, la taille mémoire des programmes, la taille mémoire des variables et les sous-programmes standards de la bibliothèque incluse dans le compilateur appelés avec leurs adresses d'implantation.

La figure 8 présente, à titre d'exemple, un programme Basic élémentaire tel que vous pouvez le frapper sous contrôle de l'éditeur. Ce même programme compilé sans aucune option est visible figure 9. Le listing comporte trois «champs»: le plus à gauche est l'adresse en hexadécimal de la première instruction en langage machine correspondant à la ligne Basic qui suit; le deuxième champ est le numéro de ligne Basic et le troisième champ est la ligne Basic que vous aviez frappée formatée de façon lisible.

En fin de listing, les sous-programmes appelés, ici OUT et MUL, sont indiqués avec l'adresse de leur première instruction. Viennent ensuite des indications relatives à la taille du programme et des variables associées.

La figure 10 présente ce même listing avec l'option S activée. Les informations supplémentaires visibles sont les adresses mémoire des variables utilisées. Les variables ont le même nom que celui qu'elles portent dans votre programme Basic. Toutes les variables marquées \* ou d'un nom ne figurant pas dans votre programme sont propres aux sous-programmes du compilateur.

```
-----  
-----  
10 PRINT  
20 FOR l = 1 TO 10  
30 PRINTl,l*l,l*l*l  
45 NEXTI  
50 STOP  
60 END
```

Fig. 8. – Exemple de programme à compiler.

```
-----  
-----  
PAGE 001                *** COMPILATEUR BASIC C.T. 6809 ***  
  
1000 0010 PRINT  
100F 0020 FOR I = 1 TO 10  
1019 0030 PRINT I, I* I, I* I* I
```

```
1049 0045 NEXT I
1057 0050 STOP
105A 0060 END
```

SOUS-PROGRAMMES

```
OUT 1062
MUL 1156
```

```
0 ERREUR DETECTEE
TAILLE VARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9
```

Fig. 9. - Listing de compilation sans aucune option.

---

```
PAGE 001                *** COMPILATEUR BASIC C.T. 6809 ***
```

```
1000 0010 PRINT
100F 0020 FOR I = 1 TO 10
1019 0030 PRINT I, I*I,I*I*I
1049 0045 NEXT I
1057 0050 STOP
105A 0060 END
```

SOUS-PROGRAMMES

```
OUT1062
MUL 1156
```

```
0 ERREUR DETECTEE
TAILLE VARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9
I 0030 00 00
* 0032 00 00
10 0034 00 00
```

Fig. 10. - L'option S fait afficher la' table des symboles.

---

```
PAGE 001                *** COMPILATEUR BASIC C.T. 6809 ***
```

```
1000 0010 PRINT
100C      17 00A3
100F 0020 FOR I = 1 TO 10
100F      00 0001
1012      DD 30
1014      CC 000A
1017      DD 32
```

```

1019 0030 PRINT I, I*I,I*I*I
1019      17 0061
101C      DC 30
101E      17 00D0
1021      17 0062
1024      DC 30
1026      30C9 0030
102A      17 0129
1020      17 00C1
1030      17 0053
1033      DC 30
1035      3009 0030
1039      17 011A
103C      30C9 0030
1040      17 0113
1043      17 00AB
1046      17 005B
1049 0045 NEXT I
1049      DC 30
104B      C3 0001
104E      DD 30
1050      93 32
1052      2E 03
1054      16 FFC2
1057 0050 STOP
1057      7E CD03
105A 0060 END
105A      7E CD03

```

#### SOUS-PROGRAMMES

```

OUT 1062
MUL 1156

```

```

0 ERREUR DETECTEE
TAILLEVARIABLES $0085
TAILLE PROGRAMME $01EA
FIN DU PROGRAMME $11E9

```

```

I 0030 00 00
* 0032 00 00
10 0034 00 00

```

Fig. 11. – L'option H fait afficher les codes hexadécimaux générés.

---

La figure 11 enfin vous montre ce que donne l'option H (et l'option S que nous avons laissée activée) ; chaque ligne Basic est suivie par les codes hexadécimaux qu'elle fait générer. Ainsi, par exemple, la ligne 10 PRINT est traduite en, un 17 00A3 qui n'est autre qu'un LBSR \$A3, c'est-à-dire un saut au sous-programme de sortie d'une ligne blanche se trouvant dans le sous-programme OUT du compilateur.

Cette analyse peut aller plus loin; ainsi, si vous regardez les codes qui suivent la ligne 0045, vous pourrez facilement les traduire en langage machine suivant : LDD \$30 qui charge D avec la variable l (l est en 0030 comme indiqué en fin de listing).  
ADDD #01 qui augmente d'une unité, puisque le pas de notre boucle FOR NEXT est de 1. STD \$30 qui place le contenu de D (donc l) en 30.  
CMPD \$0032 qui compare avec sa valeur maximale qui est stockée en 0032. BGT 03 qui fait sauter sur Stop si l est égal ou supérieur à sa valeur maximale. LBRA \$FFC2 qui fait remonter au début de la boucle FOR NEXT si l est inférieur à sa valeur maximale.

## Généralités

\*\*\*\*\*

Il faut avoir présent à l'esprit que le programme source à compiler se présente presque exactement comme un programme que vous passeriez à l'interpréteur ; vous ne devez donc pas être dépaysé.

Le programme à compiler comporte donc une suite de lignes qui commencent impérativement par un numéro de un à quatre chiffres maximum, ces chiffres étant décimaux et entiers.

Les commandes multiples sont autorisées sur une même ligne ; elles doivent être séparées par deux points (:). La longueur maximale d'une ligne ne doit en aucun cas excéder 80 caractères.

Les espaces sont ignorés, hormis dans les chaînes de caractères, comme ils n'utilisent pas de place mémoire après compilation, vous pouvez les employer largement pour améliorer la lisibilité du programme. Les REM peuvent aussi être très largement utilisés pour la même raison.

Même si vous ne mettez pas d'espace, le compilateur formate automatiquement votre listing à la compilation pour en améliorer la lisibilité.

Enfin, la dernière ligne d'un programme est celle contenant le END; la compilation s'arrête tant là et ignorant tout ce qui peut se trouver après.

## Nombres et variables

\*\*\*\*\*

Les nombres sont représentés après compilation sur 16 bits; cela signifie que le Basic ne travaille que sur des entiers compris entre + 32 767 et - 32 768, lorsque l'on emploie des entiers signés. La plage de variation va de 0 à + 65 535 lorsque l'on travaille sur des nombres non signés.

Les constantes pouvant apparaître dans un programme Basic, peuvent être exprimées en décimal ou en hexadécimal ; dans ce dernier cas, elles doivent être, comme en assembleur, précédées par un dollar (\$). Une valeur négative doit être précédée par le signe -; en revanche, le signe + devant une constante n'est pas autorisé, l'absence de moins, étant implicitement considérée comme un plus. Ce Basic se prêtant bien aux opérations logiques, il est possible de représenter le complément d'un nombre (complément à 1, bit à bit de ce nombre représenté en binaire) en faisant précéder celui-ci du signe dièse(#).

Ainsi, par exemple, les constantes suivantes sont valables

- 300, — 4 000, \$ 80, \$ 2 500, # 2, # \$ FF00.

En revanche, ne sont pas autorisées, par exemple :

-8.45, 100 000, + 15.

Les variables sont représentées, dans ce Basic, par une lettre de A à Z ou par une lettre suivie d'un chiffre de 0 à 9.

Comme dans tout Basic, les variables peuvent être indicées et former des tableaux à une ou deux dimensions. Les indices commencent à 1 et peuvent aller jusqu'à 255. Attention cependant à ce que vous faites; une directive du style

DIM A(255, 255) risque fort de ne pas passer à la compilation car cela génère un tableau de 65 025 éléments, soit toute l'étendue mémoire du 6809 !

Les indices peuvent être des constantes, des variables ou des expressions arithmétiques,

par exemple, sont valables: N(M), A(15). B5(C, D). U(N\*(A/B). 8).

Le Basic considère une variable non indicée comme étant le premier élément d'un tableau ; ainsi la variable A sera considérée comme étant A(1,1) ; attention donc au choix de vos noms de variables.

Sachez aussi que :

- Il faut 250 us pour accéder à un élément d'un tableau à deux dimensions contre 20 us pour un tableau à une seule dimension.

- Chaque variable ou élément d'un tableau se voit assigner deux octets de RAM.

#### Opérateurs arithmétiques

\*\*\*\*\*

Le Basic dispose des quatre opérateurs classiques : addition (+), soustraction (-), multiplication (X) et division (/); par ailleurs, il dispose également de quatre opérateurs logiques qui sont :

- Le ET logique représenté par le caractère «et commercial » (&).

- Le OU logique représenté par un point d'exclamation (!).

- Le OU EXCLUSIF logique représenté par le caractère « pour cent » (%).

- Le complément représenté par un dièse (#).

Tous ces opérateurs peuvent être mélangés dans des expressions. Les opérateurs logiques travaillent bit par bit sur les 16 bits des variables concernées.

La priorité entre les différents opérateurs s'établit comme suit, par ordre de prio-

rité décroissante :

- Fonctions

- Changement de signe (-) et complément logique

- ET logique, OU logique et OU EXCLUSIF logique

- Multiplication et division

- Addition et soustraction.

Des parenthèses peuvent, bien sûr, être utilisées pour modifier ces priorités.

#### Fonctions arithmétiques

\*\*\*\*\*

Ce Basic dispose de quatre fonctions dites arithmétiques, ce sont :

- ABS (EXPRESSION): qui donne la valeur absolue de l'expression concernée.
- RND ou RND (EXPRESSION) : produit un nombre pseudo-aléatoire compris entre 0 et + 32 767 si RND est utilisé seul. Si une expression suit RND, le nombre produit est le même chaque fois que l'expression se reproduit identique à elle-même.
- PEEK (EXPRESSION): donne la valeur se trouvant à l'adresse spécifiée par l'expression. Contrairement à beaucoup de Basic, l'expression peut être exprimée en hexadécimal si, elle est précédée par un dollar, ce qui est très pratique.
- POS : indique la position actuelle du curseur du terminal (ou de la tête d'impression de l'imprimante).
- SWAP (EXPRESSION): échange les deux octets de l'expression considérée.
- ERR: indique par un numéro de code le dernier type d'erreur rencontré (voir tableau de la figure 12 pour la signification de ce numéro de code).

#### Les chaînes de caractères \*\*\*\*\*

Une chaîne de caractères est constituée de tout ce qui se trouve compris entre deux guillemets à l'exception classique du retour chariot, du saut ligne et du caractère de contrôle dont la valeur en ASCII est \$ 1A, car celui-ci joue un rôle particulier. Si un guillemet doit être inclus dans une chaîne de caractères, il doit être frappé en double pour

ne pas le confondre avec ceux de début et de fin de chaîne.

Une chaîne de caractères peut avoir une longueur maximale telle qu'elle tienne dans une ligne standard du Basic (80 caractères maxi).

Les noms des variables chaînes de caractères ne peuvent être constitués que par une lettre de A à Z suivie par un dollar. Une chaîne de caractères peut être utilisée avec dimensionnement ou faire appel à un dimensionnement implicite ; dans ce cas, le compilateur réservera 32 octets de mémoire pour la chaîne et sa longueur ne pourra donc dépasser 32 caractères; il vous faudra donc déclarer toute chaîne de caractères plus longue que 32 caractères. Par ailleurs, si vous voulez économiser de la RAM sur le programme compilé, vous avez intérêt à déclarer systématiquement toutes vos chaînes avec leurs longueurs réelles.

L'opérateur d'addition (+) peut être utilisé sur les chaînes de caractères, par exemple :

- « HAUT » + « PARLEUR » est égal à la chaîne de caractères « HAUT-PARLEUR ».

Une chaîne de caractères peut être initialisée à «rien » en début de programme de la façon suivante :

- AS="".

Cette possibilité d'initialisation est importante, car il ne faut pas oublier que plus tard cette chaîne sera constituée par plusieurs octets de RAM qui peuvent contenir n'importe quoi à la mise sous tension.

---



---

NUMERO	TYPE D'ERREUR
0 à 28	Erreur DOS, codes indiqués p. 189 n° 1692
29	Sans signification
30	Sans signification
31	Sans signification
32	Débordement lors d'une multiplication
33	Division par zéro
34	Erreur de conversion ASCII — binaire

---

Fig. 12. – Tableau des erreurs exploitables par la fonction ERR.

---



---

Les fonctions relatives aux chaînes de caractères  
 \*\*\*\*\*

Dans les définitions qui vont suivre, N, M seront des constantes, des variables ou des expressions numériques tandis que X\$ sera une variable ou une expression chaîne de caractères. Les fonctions dont on dispose sont divisées en deux groupes, les fonctions produisant un résultat numérique et celles produisant un résultat chaîne de caractères. Nous allons commencer par les derniers.

- CHR\$(N) : produit un caractère dont N est le code ASCII.
- LEFT\$(X\$, N) : produit une chaîne de caractères constituée par les N caractères de gauche de la chaîne X\$.
- MID\$(X\$, N, M) : produit une chaîne de caractères qui est le sous-ensemble de X\$ commençant au nième caractère de X\$ en partant de la gauche et comprenant M caractères.
- RIGHT\$(X\$,N) : produit une chaîne de caractères constituée par les N caractères en partant de la droite de X\$.
- STR\$(N) : produit une chaîne de caractères dont le contenu est N, ainsi: STR\$(4567) = la chaîne de caractères « 4567 ».
- TRIM\$(X\$) : enlève les espaces se trouvant en fin d'une chaîne de caractères; ainsi: TRIM\$«BASIC » = «BASIC»

Les fonctions suivantes exploitent des chaînes de caractères pour produire des résultats numériques, ce sont :

- ASC(X\$) : qui donne le code ASCII du premier caractère de la chaîne.
- LEN(X\$) : qui donne la longueur de la chaîne de caractères concernée.
- INSTR(X\$, V\$) : qui cherche la chaîne V\$ dans la chaîne X\$ ; si la recherche s'avère positive, cette fonction fournit un nombre égal à la position du premier caractère de V\$ dans X\$; sinon, la valeur 0 est affectée au résultat de cette fonction.
- VAL(X\$) : a pour effet de convertir une chaîne de caractères en son équivalent numérique, ainsi VAL(«1234») = 1234.

De toutes ces fonctions, VAL et INSTR peuvent produire des erreurs si



les données fournies sont incompatibles avec la fonction, ces erreurs peuvent être traitées par un ON ERROR GOTO. Ce Basic, de par le fait qu'il est compilateur, permet d'utiliser en plus de ce qui a été vu, une chaîne de caractères particulière qui est son «buffer» d'entrées/ sorties ; celui-ci s'appelle tout simplement BUFS et il contient au maximum 129 octets. Les commandes suivantes peuvent, par exemple, être utilisées :

- Pour le lire : PRINT BUF\$ ou READ#N, BUF\$.
- Pour y écrire: INPUT BUF\$ ou WRITE#N, BUF\$.

#### Assignment boucles et sauts

\*\*\*\*\*

Comme dans tout Basic, il existe une directive d'assignation de valeur :

- LET; son utilisation est optionnelle et A = 2 aura le même effet que LET A = 2.

Il est également possible de placer une valeur en mémoire à une adresse spécifiée au moyen de la directive POKE, sous la forme POKE (X) = Y où Y est une expression qui est calculée puis tronquée à 8 bits, et qui est ensuite placée à l'adresse spécifiée par l'expression X.

Contrairement à beaucoup de Basic interprétés, il est possible de faire appel directement à des sous-programmes écrits en langage machine; cela se fait au moyen de la directive CALL sous la forme:

- CALL ADRESSE ou ADRESSE représente l'adresse de début du sous-programme ainsi appelé. L'adresse, comme toutes les variables, peut être spécifiée en décimal ou hexadécimal si elle est précédée d'un dollar. Le sous-programme ainsi appelé doit impérativement se terminer par un RTS et doit rendre la pile telle qu'elle se trouvait lors de l'appel.

Les boucles s'exécutent, comme dans tout Basic, par une commande FOR NEXT de la façon suivante :

- FOR VARIABLE = EXPRESSION TO EXPRESSION STEP EXPRESSION ou VARIABLE est une variable non indicée numérique et où EXPRESSION sont n'importe quelles expressions valides. Si STEP n'est pas indiqué, la valeur prise par défaut est 1.

De plus, en raison même du procédé de compilation qui produit un code machine très compact et rapide :

- La boucle sera toujours exécutée au moins une fois, quelle que soit la valeur précisée après TO.

- A la fin de la boucle, le compteur utilisé sera supérieur à la valeur finale spécifiée d'une quantité égale à STEP.

- Ces boucles peuvent être quittées et reprises quand vous le désirez.

- Lors de la compilation, vous pouvez avoir jusqu'à 16 boucles actives simultanément.

- Les valeurs initiales, finales et le pas (valeur indiquée après STEP) peuvent être positives ou négatives.

Les sauts peuvent faire appel aux directives GOSUB, IF THEN, ON

ERROR GOTO, ON GOTO et ON GOSUB que nous allons étudier dans cet ordre.

GOSUB est très classique et sert à appeler un sous-programme en Basic, sous-programme qui doit impérativement se terminer par un RETURN. La syntaxe de GOSUB est très simple:

- GOSUB NUMERO DE LIGNE. La compilation de ces deux directives (GOSUB et RETURN) est très courte puisque GOSUB donne un JSR, tandis que RETURN donne un RTS.

IF THEN est très classique, son utilisation est la suivante :

- IF EXPRESSION RELATION EXPRESSION THEN NUMERO DE LIGNE ou RELATION représente un des six opérateurs de comparaison qui sont : inférieur (<), supérieur (>), égal (=), différent (<>), inférieur' ou égal (<= ou <=) est supérieur ou égal (=> ou >=) Par ailleurs, le THEN peut être remplacé par un GOSUB. Si la comparaison est vraie, le programme se branche à la ligne indiquée, et si le THEN est un GOSUB, le RETURN a pour effet de faire revenir à la ligne suivant immédiatement celle du IF GOSUB. Compte tenu de la possibilité de mettre plusieurs commandes sur la même ligne, le IF THEN peut jouer le rôle d'un IF THEN ELSE, par exemple :

```
IF A> 2 THEN 1 000 ELSE 200 se traduira par :
IF A> 2 THEN '1000:
GOTO 200.
```

Le ON ERROR GOTO est un peu analogue à celui de l'interpréteur Basic étendu. Il s'utilise de la façon suivante :

- ON ERROR GOTO ou ON ERROR GOTO NUMERO DE LIGNE.

Dans le premier cas, la fonction ON ERROR GOTO est désactive, tandis que, dans le second cas, toute erreur détectée à la suite de cette directive causera un branchement au numéro de ligne précisé après le ON ERROR GOTO. Les erreurs qui peuvent être détectées par. cette directive sont les suivantes :

- Division par zéro dans une expression arithmétique.

- Erreur de conversion ASCII

- binaire dans un INPUT, READ, VAL (x\$).

- Dépassement de capacité en multiplication dans une expression arithmétique.

- Erreur d'entrée/sortie disque.

Il faut alors utiliser la fonction ERR vue ci-avant pour savoir à quel type d'erreur l'on a à faire.

ON GOTO et ON GOSUB travaillent de la façon suivante:

ON EXPRESSION GOT0 (GOSUB) NUMERO 1, NUMERO 2, etc., où EXPRESSION représente une expression valide et NUMERO 1, NUMERO 2, etc., des numéros de lignes. L'expression est calculée et le résultat indique à quel numéro de ligne faire le branchement; ainsi:

```
ON A GOTO 100, 200, 300, 400
```

fera brancher le programme en 100 si A = 1, en 200 si A = 2, etc. Si le calcul de l'expression donne une valeur nulle, le programme passe à la ligne suivante.

STOP est un branchement un peu particulier, en ce sens qu'il arrête l'exécution du programme et cause un saut inconditionnel au dos en son point d'entrée-CD03 à « chaud »

## Les directives machine

\*\*\*\*\*

Un compilateur étant très près du langage machine, il est possible d'agir sur certains paramètres au moyen des directives suivantes.

GEN permet de générer au sein d'un programme une suite de valeurs qui sont insérées telles quelles dans celui-ci lors de la compilation. Cette possibilité est utilisée pour ajouter quelques instructions lorsqu'elles sont plus faciles à écrire en langage machine qu'en Basic. L'utilisation se fait de la façon suivante :

- GEN NOMBRE 1, NOMBRE 2, NOMBRE 3, etc., ou NOMBRE 1, NOMBRE 2, etc., sont des valeurs décimales ou hexadécimales qui seront placées dans le programme généré. Si un des nombres est inférieur à 255, il occupera quand même un octet, même si le nombre est précédé de zéros. Pour faire un branchement à l'adresse \$ 1000 par exemple, vous pouvez écrire :

- GEN \$ 7E (code du JMP), \$ 10, \$00.

---

NUMERO	SIGNIFICATION
02	Numéro de ligne en double ou trop grand
03	Directive inconnue
04	Erreur de syntaxe
05	Nom de variable absent ou en erreur
06	Signe égal manquant
07	Référence à une ligne inexistante
08	Parenthèse droite absente ou en nombre incorrect
09	Opérande absent dans une expression
10	Numéro de ligne de destination absent ou en erreur
11	Nombre manquant
12	Mauvais enchevêtrement FOR-NEXT
13	Débordement de la table des symboles
14	Sans signification
15	Erreur sur un opérateur de relation
16	Délimiteur (, ou ;) absent
17	Guillemet absent en fin de chaîne
18	Compteur incorrect ou absent dans un FOR-TO
19	Tableau déjà défini
20	Erreur d'indice : indice manquant ou indices trop nombreux
21	Erreur d'indice : indice plus petit que 1 ou plus grand que 255
22	Débordement de la RAM affectée aux variables au-dessus de FFFF
23	Référence à un tableau non dimensionné
24	Erreur d'indice
25	Argument d'une fonction manquant ou incorrect (numérique)
26	Option illégale

- 27 Opérateur non reconnu dans une expression chaîne de caractères
- 28 Opérateur de concaténation (+) absent
- 29 Argument d'une fonction manquant ou incorrect (chaîne)
- 30 Nombre de boucles FOR-NEXT simultanées supérieur à 16
- 31 Débordement de la table des numéros de lignes
- 32 Débordement de la mémoire de programme au-dessus de FFFF
- 33 Sans signification
- 34 GOTO ou GOSUB manquant
- 35 Numéro de canal incorrect (non compris entre 0 et 9)
- 36 Erreur d'entrées/sorties disque

Fig. 13. – Liste des erreurs de compilation

-----  
 -----  
 Les entrées sorties  
 \*\*\*\*\*

Les entrées sorties se font par un « buffer » de 129 octets qui peut donc contenir 128 caractères. Celui-ci est automatiquement placé après toutes les autres affectations mémoire, sauf le « buffer » de chaîne de caractères.

Les entrées se font au moyen d'un INPUT sous la forme :  
 INPUT VARIABLE 1, VARIABLE 2, etc.

Un retour chariot, saut ligne et un point d'interrogation sont alors générés et le programme lit ensuite ce qui entre jusqu'à ce que son « buffer » soit plein ou qu'un retour chariot soit frappé. La frappe d'un CNTRL X fait imprimer « DEL » et efface complètement la ligne frappée, tandis qu'un CNTRL O renvoie le dernier caractère frappé et l'annule dans le « buffer » d'entrée. Lorsque plusieurs variables sont attendues, elles doivent être séparées par une virgule ou un espace si ce sont des variables numériques, et par une virgule si ce sont des variables chaînes de caractères (les espaces étant comptés comme éléments de la chaîne).

PRINT s'utilise d'une façon analogue sous la forme :

– PRINT EXPRESSION 1 DELIMITEUR EXPRESSION 2 DELIMITEUR, etc., où EXPRESSION 1, EXPRESSION 2, etc., peuvent être des expressions numériques ou des chaînes de caractères tandis que DELIMITEUR peut être une virgule ou un point virgule. Dans le premier cas, les expressions à imprimer le sont dans des zones de 8 caractères de large; la virgule faisant passer d'une zone à l'autre; dans le deuxième cas les expressions se suivront sans espacement. Parmi les expressions :

la fonction TAB peut être utilisée. Ce compilateur dispose en outre de possibilités d'entrées sorties sur disque avec création et accès aux fichiers.

Comme pour l'interpréteur BASIC étendu, les accès disque se font par des numéros de voies ou de canaux compris entre 0 et 9. Le

compilateur tolère donc un maximum de 10 canaux ouverts simultanément.

Les directives d'accès aux fichiers disque dont on dispose sont les suivantes :

– CLOSE FILES : doit être utilisé lorsque tous les accès disque sont terminés. Cette commande a pour effet de fermer tous les fichiers qui pourraient être restés ouverts. Il est également conseillé de placer cette directive dans les programmes de gestion des erreurs lors des accès disque; cela vous assurera d'avoir un disque parfaitement « propre », quoi qu'il arrive.

– OPEN: permet d'ouvrir un fichier disque pour une lecture ou une écriture. La syntaxe est: OPEN # NUMERO, NOM DE FICHIER où NUMERO est le numéro de voie (compris entre 0 et 9) qui sera affecté à ce fichier et où NOM DE FICHIER

est un nom conforme à la syntaxe du DOS. Un OPEN doit être effectué préalablement à tout accès à un fichier, de plus un numéro de voie ne doit être affecté qu'à un fichier et à un seul à un instant donné. Enfin, un fichier peut être ouvert en lecture ou en écriture. Lors de l'exécution de la directive OPEN, le BASIC cherche dans le répertoire des fichiers si celui-ci existe ou non. Dans le premier cas, le fichier est ouvert en lecture, dans le second cas, le fichier est ouvert en écriture. Pour savoir dans quel état se trouve l'ouverture d'un fichier donné, on dispose de la commande STATUS.

– CLOSE: permet de fermer un fichier et de libérer ainsi son numéro de voie pour un autre usage. La syntaxe en est : CLOSE # NUMERO où NUMERO est le numéro de voie à libérer. Cette commande agit que le fichier soit en lecture ou en écriture.

– WRITE : permet d'écrire dans un fichier ouvert en écriture. La syntaxe en est : WRITE # NUMERO, LISTE où NUMERO est le numéro de la voie où écrire et où liste représente une liste de constantes, de variables ou d'expressions numériques ou chaînes de caractères, séparées par des virgules. Chaque élément de la liste est considéré comme un article de l'enregistrement ainsi défini. La longueur totale d'un enregistrement donné ne peut excéder 128 octets. Il faudra donc faire autant d'enregistrements que nécessaire si vous dépassez cette longueur. Tous les articles sont convertis en ASCII sur le disque et créent donc ainsi un fichier « texte ». La virgule séparatrice se retrouve dans le fichier ainsi créé.

La figure 15 clarifie un peu tout cela en montrant ce qui se trouve réellement sur le disque après un WRITE donné.

– RWRITE réalise la même fonction que WRITE, mais permet en plus de spécifier un numéro d'enregistrement sous la forme RWRITE # NUMERO, ENREGISTREMENT, LISTE où ENREGISTREMENT est le numéro évoqué ci-avant. NUMERO et LISTE ont la même signification que pour la commande WRITE vue ci-avant.

– READ: permet de lire le contenu d'un enregistrement existant de la façon suivante : READ # NUMERO, LISTE où NUMERO est un numéro de voie ouverte en lecture et où LISTE est une liste de variables séparées par des virgules. Le fichier correspondant à la voie ainsi spécifiée doit être un fichier texte et le nombre d'articles qu'il contient doit être compatible au nombre de variables demandées. S'il y a plus d'articles dans l'enregistrement que de variables dans

la liste, ceux qui sont «en trop» seront ignorés. La liste de variables peut comporter tous les types admis par le Basic (numérique, chaîne de caractères, indices).

– RREAD est à READ, ce que RWRITE est à WRITE. L'utilisation et la syntaxe sont les mêmes que pour RWRITE vu ci-avant.

– CHAIN: permet au programme BASIC en cours d'exécution de charger et de lancer l'exécution d'un autre programme. La syntaxe est CHAIN NOM où NOM est le nom du programme à lancer.

Les contraintes suivantes sont à respecter: le programme ainsi appelé doit avoir, au préalable, été sauvegardé sur disque sous forme binaire: c'est-à-dire avoir été assemblé ou compilé au préalable; de plus, le programme doit être muni d'une adresse de transfert (voir mode d'emploi du DOS). Il faut être conscient du fait que, lors d'un CHAIN, le programme BASIC passe «la main» au DOS si bien que, si une erreur se produit (fichier non trouvé, disque pas prêt, etc.), le contrôle ne sera pas rendu au programme BASIC mais au DOS.

– RESTORE et SCRATCH:

RESTORE permet de fermer un fichier ouvert en lecture ou écriture et de l'ouvrir à nouveau en lecture. La syntaxe est RESTORE #1 NUMERO'1, # NUMERO 2, etc., où NUMERO 1, NUMERO 2, etc., sont les numéros de voie des fichiers concernés.

SCRATCH joue le même rôle mais ouvre à nouveau le ou les fichiers concernés en écriture. La syntaxe est identique à RESTORE. Attention, de par sa fonction SCRATCH détruit le contenu des fichiers concernés, puisque ceux-ci sont rendus à nouveau disponibles pour une écriture.

– KILL: est à manier avec beaucoup de précautions. KILL NOM détruit de façon définitive le fichier dont le nom est spécifié.

– EOF: permet de savoir si l'on a atteint la fin d'un fichier... La syntaxe en est EOF (# NUMERO) qui donne une valeur de 1 si l'on a atteint la fin du fichier dont le numéro de voie est spécifié, ou une valeur nulle dans les autres cas.

– FILSIZ : permet de connaître la longueur d'un fichier en nombre de secteurs sur le disque. La syntaxe est : FLSIZ (#: NUMERO) où NUMERO est le numéro de la voie du fichier concerné.

– STATUS : permet de savoir dans quel état se trouve un fichier. La syntaxe est : STATUS (# NUMERO) où NUMERO est le numéro de voie du fichier concerné. Cette commande donne une valeur nulle pour un fichier non ouvert, une valeur de 1 pour un fichier ouvert en lecture et une valeur de 2 pour un fichier ouvert en écriture.

Exemple d'utilisation : ON STATUS (# 2) GOTO 100,200, fera passer à la ligne suivante si la voie 2 n'est pas ouverte, fera aller en 100 si elle est ouverte en lecture et en 200 si elle est ouverte en écriture.

---



---

NOM	FONCTION
OUT	Sortie de caractères
INP	Entrée de caractères
DSK	Entrées/sorties sur disquettes
MUL	Multiplication arithmétique
DIV	Division
RND	Générateur de nombres aléatoires
ST1	Fonction relative aux chaînes de caractères
ST2	Fonction relative aux chaînes de caractères
ARR	Multiplication de deux tableaux
IND	Chargement indirect d'un tableau

Fig. 14. -- Noms et fonctions des sous-programmes de la bibliothèque du compilateur.

---



---

```

90 N = 10
100 WRTE # 2,25 - 400, « TOTO», N
Produit un fichier qui contient :
32 35 2C 2D 34 30 30 20 54 4F 54 4F 2C 31 30 OD
 2 5 - 4 0 0 T 0 T 0 . 1 0 EOF

```

Fig. 15. - Le contenu d'un fichier disque après un ordre WRITE

---



---

#### Les directives de compilation

\*\*\*\*\*

Ces directives n'ont aucun équivalent en BASIC interprété puisque l'implantation en mémoire du programme ne vous regarde pas. Ici, il n'en est pas de même et vous pouvez (et vous devez !) indiquer au compilateur où il devra implanter votre programme. Nous allons donc retrouver des directives analogues à celles utilisées en assembleur; en effet, nous allons commencer par ORG.

ORG sert à définir l'emplacement où va être implanté le code produit par le compilateur. La syntaxe est tout simplement ORG = XXXX où XXXX représente une adresse exprimée en décimal ou hexadécimal (précédée d'un dollar dans ce dernier cas). Cette directive peut être utilisée autant de fois que vous le désirez dans un programme, ce qui vous permet de le découper ainsi en autant de blocs que vous le souhaitez. '

Si vous ne précisez pas ORG, la valeur prise par défaut est \$ 1 000. Une deuxième directive, un peu analogue à ORG, est à votre disposition. Il s'agit de BASE qui sert à définir l'adresse de début d'implantation des RAM utilisées pour les variables du programme.

Les remarques faites ci avant pour ORG sont toutes valables, sauf la valeur prise par défaut qui est ici de \$ 0030.

Ces directives étant nouvelles dans un BASIC, nous allons vous donner quelques conseils à leur sujet afin que vous puissiez les utiliser au mieux.

Il est conseillé de faire attention à ce que les zones définies pour la RAM ne recouvrent pas celles définies pour le programme. Les informations données par le compilateur enfin de compilation sont très utiles dans ce cas.

Une bonne pratique consiste à commencer un programme par une directive BASE suivie par des DLM de toutes les variables, ce qui fera d'une pierre deux coups. Ensuite la directive ORG marquera le début du programme proprement dit.

La directive DLM peut, elle aussi, être considérée comme une directive de compilation; en effet, elle sert à déclarer les tableaux et, dans certains cas, les variables simples. Les tableaux numériques peuvent être déclarés comme étant à une ou deux dimensions sous la forme DLM A (X,Y) où X et Y représentent la taille du tableau A. Ne pas oublier que les indices commencent à 1. Les chaînes de caractères ne peuvent être déclarées que comme variables à une dimension. Cependant, DLM sert aussi, dans ce cas, à indiquer la longueur de la chaîne. Ainsi DLM A513 (50) signifie que la chaîne A\$ a une longueur de 50 caractères tandis que DLM A\$ (15,30) considère la chaîne A\$ comme étant un ensemble de 15 chaînes de 30 caractères. Attention ! dans le premier exemple ci-avant, A\$ sera à utiliser sans indice puisque A\$ sera une variable simple (DIM n'ayant servi qu'à définir sa longueur), tandis que dans le deuxième exemple, A\$5 sera à utiliser avec un indice puisque A\$ comportera 15 éléments. DLM peut aussi être utilisé pour déclarer des variables simples dans des cas particuliers. Ainsi, pour déclarer le registre d'état d'un ACIA en \$ 8008 comme étant la variable A, ferons-nous :

- BASE = \$ 8008
- DLM A (1)

Le registre d'état sera ainsi atteint lorsque l'on utilisera la variable A. Comme nous avons dit dans le paragraphe consacré aux variables que le fait d'écrire A était équivalent à A (1) ou à A (1,1), la définition de A réalisée ci-avant est très correcte.

La dernière directive de compilation doit impérativement être un END qui indique au compilateur qu'il doit s'arrêter. Ce END fait, par ailleurs, générer un saut au point d'entrée chaud du DOS par un JMP en CD03. Si l'endroit où se trouve le END ne correspond pas à la fin effective d'un programme (cas de sous-programmes écrits en fin de listing par exemple), la fin réelle du programme doit être matérialisée par un STOP qui fait aussi générer le même saut.

Les erreurs de compilation



\*\*\*\*\*

Lors de la phase de compilation, les erreurs détectées sont indiquées sur le listing par un numéro de code conforme à la liste visible figure 13. De plus, une flèche est positionnée, sous la ligne en erreur, à l'emplacement approximatif de celle-ci. Cette dernière fonction ne peut cependant pas être à 100 % exacte et ne doit être considérée que comme une indication. Ces erreurs de compilation ne doivent pas être confondues avec les erreurs détectées par la fonction ERR. Ces dernières sont des erreurs dues à l'exécution du programme alors que celles de compilation sont des erreurs de syntaxe lors de l'écriture du programme.

Enfin, pour information, la figure 14., vous indique la liste des sous-programmes du compilateur qui peuvent apparaître en fin de listing.

#### Informations diverses

\*\*\*\*\*

Les manettes de jeux dont nous avons parlé dans notre, dernier. article ne sont pas oubliées et seront décrites dès que possible, compte tenu de la place disponible et des projets en cours.

Nous avons commencé à recevoir et à ficher des réponses au sondage publié dans le numéro de mars, mais celles-ci sont encore trop peu nombreuses pour avoir une signification et permettre de constituer une liste valable de réalisateurs du système. Vous êtes, au 26.03.84, une centaine seulement à avoir répondu, ce qui fait moins de 10 % des réalisateurs du système...

#### Conclusion

\*\*\*\*\*

Nous en resterons là pour aujourd'hui en espérant que la diversité des sujets abordés aura intéressé nombre d'entre vous. Nous vous donnons rendez-vous le mois prochain pour un article dont le contenu n'est pas encore défini, tant nous avons de choses à dire à propos de ce mini-ordinateur au fur et à mesure de son évolution.

C. TAVERNIER

NOTE : Cet article a été publié dans le numéro 1704 de Mai 1984 de la Revue LE HAUT PARLEUR, dans la série d'articles intitulés : Réalisez votre ordinateur individuel par Christian Tavernier.