

OMIKRON.

BASIC

3

COMPILATEUR

Le Basic de l'Atari ST

Vous devez impérativement renvoyer la carte d'enregistrement pour recevoir gratuitement le manuel de programmation de l'OMIKRON. De plus, aucune mise à jour, gratuite ou non, ne sera transmise aux acheteurs de l'OMIKRON n'ayant pas retournés leur carte d'enregistrement.

NOUS AVONS BESOIN DE CES CARTES, RENVOYEZ NOUS LES !

L'OMIKRON n'est pas protégé contre la copie, et vous pouvez le copier autant de fois que vous le désirez, sous réserve que ces copies ne servent qu'à des fins de sauvegarde pour vous prémunir contre la perte des disquettes originales.

Comme un livre ne peut être lu par deux personnes à la fois dans deux lieux différents, ce logiciel ne peut être utilisé à la fois sur deux micro ordinateurs par deux personnes.

En aucun cas vous ne devez modifier, adapter ou intégrer ce logiciel dans d'autres produits destinés à la revente sans autorisation préalable de OMIKRON. Software & Hardware.

TABLE DES MATIERES

Table des matières	Page 1
Introduction	Page 2
Mise en route	Page 3
Messages d'erreur	Page 4
Différences avec l'interpréteur	Page 7
Types des fonctions	Page 11
Organisation de la mémoire	Page 12
Segment Pointer	Page 13
Mots clés du compilateur	Page 14
La variable COMPILER	Page 16
Optimisation	Page 17
Programmes d'aide	Page 19

Veuillez s'il vous plait lire ce manuel avant de nous téléphoner pour nous dire que le compilateur ne fonctionne pas. Dans le chapitre concernant les différences entre le compilateur et l'interpréteur, vous trouverez les erreurs qui reviennent le plus souvent.

INTRODUCTION

Tout d'abord, nos félicitations pour avoir acheté ce programme qui représente l'aboutissement de plusieurs mois de développement et de mise au point. Nous sommes certains que, comme des milliers d'utilisateurs de ce compilateur de par le monde, vous serez très satisfait des prestations qu'il vous apportera.

Pourquoi les programmes compilés sont-ils plus rapides ?

Dans un langage interprété, chaque instruction doit être décodée, puis l'instruction correspondante doit être exécutée. Le décodage prend énormément de temps. De plus, les instructions se doivent d'être générales et ne sont donc pas forcément optimisées en ce qui concerne les temps d'exécution.

Dans un compilateur, un programme source est disséqué et optimisé. Une fois la compilation terminée, il n'y a plus de décodage à l'exécution de chaque instruction. Le compilateur effectue ce travail une fois pour toute à la compilation. Le résultat est un programme directement compréhensible par la machine.

Le compilateur accepte les sources en Basic OMIKRON au format FIT (Fast Interpreting Technic, méthode de codage de l'OMIKRON). Il produit un programme au format TOS, en langage machine 68000.

La vitesse d'exécution est multipliée par un facteur compris entre 1 et 30. Il n'y a pas de facteur moyen, car cela dépend étroitement des instructions utilisées. Par exemple, les appels au système et les instructions graphiques ne sont que peu améliorées, ce qui est normal puisque le programme continue de faire appel à des fonctions de la ROM.

Le compilateur OMIKRON que vous possédez a été entièrement écrit en OMIKRON Basic. Pour vous donner une idée de la rapidité des programmes compilés, la compilation du compilateur par lui-même en interprété a pris 20 minutes. Le compilateur compilé a mis cette fois 4 minutes pour se recompiler. Soit un facteur 5, ce qui est tout à fait correct, vous en conviendrez !

MISE EN ROUTE

- Allumez l'ordinateur et insérez la disquette originale.
- Double-cliquez sur le programme "COMPILER.PRG"
- Le compilateur se charge et un sélecteur de fichiers apparaît.

Comme dans le super-éditeur, le sélecteur de fichiers dispose d'améliorations qui facilitent la vie :

Si, à la place du nom, vous tapez "A:" ou "C:", vous obtiendrez le directory du lecteur A ou C. L'OMIKRON réaffichera alors le nom par défaut. Il vous suffira donc de cliquer "CONFIRMER". De plus, le compilateur comprendra que vous voulez changer de chemin d'accès par défaut si vous continuez après avoir effacé le nom du fichier.

IMPORTANT :

- Le programme doit avoir été sauvegardé par un SAVE "NOM" standard et non pas SAVE "NOM",A (le compilateur n'accepte pas l'ASCII).
- La compilation s'effectue en trois passes.
- Vous voyez défiler les numéros de ligne pour chacune des trois passes.
- Le compilateur travaille à 48 Koctets par minute au maximum.
- Le compilateur sauvegarde le programme.
- Assurez-vous qu'il y a assez de place libre sur la disquette. En règle générale, un programme compilé est plus long que le source en Basic (et jusqu'à deux fois plus long).
- Le programme compilé contient l'extension .PRG – un programme du nom "LISTE.BAS" est automatiquement sauvé sous le nom "LISTE.PRG".
- De nouveau, le sélecteur de fichiers apparaît. Vous pouvez recommencer l'opération, et compiler autant de programmes que vous le désirez.
- Sur chaque disquette contenant des programmes compilés, vous devez recopier le fichier BASLIB_3 ou bien utilisez le programme CUTLIB. Le fichier BASLIB_3 contient toutes les routines dont votre programme compilé a besoin. Si vous disposez d'un disque dur, il n'est pas nécessaire de recopier BASLIB_3 sur chaque partition. Le programme compilé recherchera directement ce fichier sur la partition C.

MESSAGES D'ERREUR

Certains programmes, qui fonctionnent parfaitement avec l'interpréteur, doivent être modifiés pour fonctionner avec le compilateur (voir le chapitre 'Différences avec l'interpréteur').

Le compilateur n'accepte pas les tableaux non dimensionnés alors que l'interpréteur les tolère si leur dimension n'excède pas 10.

En fait, l'interpréteur se montre plus 'compréhensif' vis à vis des petites erreurs et des oublis.

Le compilateur est plus strict et il affiche un ou des messages d'erreurs. Après avoir trouvé une première erreur, le compilateur continue pour vous montrer toutes les erreurs qui se trouvent dans le programme. Mais il ne sauvegarde pas le programme résultant (c'est impossible: il y a une erreur !).

En ce qui concerne les erreurs secondaires, (celles qui n'empêchent pas la compilation), le compilateur affiche un 'WARNING' quand il en trouve une.

En fait, vous avez tout intérêt à trouver et à supprimer le pourquoi d'un WARNING. Bien souvent, en effet, un programme comportant des WARNING donne des résultats... inattendus !!!

Too many variables (Trop de variables).

L'emplacement réservé aux variables ne peut pas dépasser 64 Koctets.

Out of memory (Plus de mémoire).

La mémoire disponible ne suffit pas pour compiler le programme.

Solutions :

- ❑ Supprimer ou réduire la taille du RAM disque ;
- ❑ Raccourcir le programme ou le séparer en plusieurs morceaux.
Pour passer d'une partie à l'autre, utilisez CHAIN. Le compilateur est assez gourmand en mémoire, aussi, il est préférable d'utiliser un appareil avec 1 Mo de RAM.

Type mismatch (Type de variable incompatible).

Le type de variable indiquée n'est pas correct. Par exemple, vous avez indiqué un nombre au lieu d'une variable alphanumérique.

Warning : RETURN type mismatch

(Type de la valeur retournée incompatible).

Une fonction à plusieurs lignes, définie par vous, doit obligatoirement recevoir et retourner le même type de variable.

Faux : FN addition#(X,Y):RETURN X+Y

Quand une fonction est définie en réel double, elle doit aussi retourner un réel double.

Vrai : DEF FN addition#(X#,Y#):RETURN X#+Y#

Vrai : DEF FN addition#(X,Y):RETURN CDBL(X+Y)

Bad EXIT (Mauvais EXIT).

EXIT fonctionne avec le compilateur sous la forme EXIT simple ou EXIT -1. Donc, EXIT 3 est interdit.

Quand à EXIT TO, il n'est autorisé que pour quitter une boucle. EXIT TO n'est pas utilisable pour quitter une procédure ou une fonction.

Structure too long (Structure trop grande).

Une structure (FOR...NEXT, REPEAT...UNTIL, WHILE...WEND, IF...THEN..ELSE) ne peut contenir plus de 32 Ko une fois compilée.

L'interpréteur est lui aussi incapable de travailler avec de telles structures, mais il ne faut pas perdre de vue qu'un programme compilé réclame plus de mémoire qu'un programme interprété.

Il suffit d'utiliser des procédures pour diminuer la taille de la structure.

Warning: Unused statement(s) : (Elément(s) inutilisé(s)).

Les procédures et les étiquettes mentionnées ne sont pas utilisées dans le programme. Vous pouvez les éliminer, afin de gagner de la place.

Undefined statement(s) or DIMs

(Etiquette non définie ou pas de DIM)

Les GOTO, GOSUB etc... sautent à une ligne ou une étiquette qui n'existe pas.

OU

Un tableau n'est pas dimensionné. L'interpréteur, par défaut, dimensionne les tableaux jusqu'au dixième élément. Pour supprimer le problème, il suffit de dimensionner suffisamment les tableaux en début de programme et après chaque CLEAR.

DIFFERENCES AVEC L'INTERPRETEUR

- Les variables alphanumériques utilisées avec FIELD ne doivent pas être utilisées comme variables locales. Elles sont réservées et il est interdit de les employer dans les fonctions et les procédures.
- GOTO <Etiquette> tient compte des huit premiers caractères. Les caractères suivants sont ignorés.
- Lors de la division de deux nombres entiers, l'interpréteur tente de conserver un résultat entier quand c'est possible. Par contre, le compilateur passe automatiquement en réels. Afin d'éviter la perte de temps consécutive à cette transformation, vous pouvez utiliser la division entière '\'. Donc, la ligne :

A = A/2

s'écrira :

A = A\2 ou encore A = A SHL 1

- Les tableaux ne sont pas automatiquement dimensionnés. Vous devez vous-même dimensionner les tableaux au début du programme ou après chaque CLEAR. Si vous n'avez pas dimensionné un tableau, il est possible que vous obteniez 'BUS ERROR' à l'écran.
- ATTENTION : les routines de correction d'erreur standards peuvent parfois détruire :
 - La valeur finale des indices FOR...NEXT
 - Les variables incluses dans les registres (non implémenté pour l'instant).

Voici une routine de traitement d'erreur qui fonctionne correctement :

```

100 ON ERROR GOTO Traite_Erreur
110 FOR I = 0 TO Nombre-1
120 OPEN "I",I+1,Fnam$(I)
130 NEXT I

...
50000 -Traite Erreur
50010 IF ERL=110 THEN FOR I=I TO Nombre-1 : RESUME NEXT : NEXT I
50020 RESUME NEXT
  
```

Ce n'est pas joli, mais cela fonctionne.

- A l'intérieur de procédures ou de fonctions à plusieurs lignes, les éléments d'un tableau utilisés en tant que variables locales ou en tant que paramètres locaux, ne doivent ni ouvrir, ni fermer des fichiers, ni dimensionner ou re-dimensionner des tableaux.
- Les variables locales ne sont pas contenues dans le Segment Garbage, mais dans le Segment Chaîne et dans la pile. Donc, il sera peut être nécessaire d'augmenter la taille de la pile avec un CLEAR (CLEAR ,X).
- EXIT ne fonctionne avec le compilateur que sans argument ou sous la forme EXIT -1.

Par exemple, EXIT 3 ne fonctionne pas.

EXIT TO permet uniquement de quitter une boucle. EXIT TO ne peut être utilisé pour quitter une procédure ou une fonction.

EXIT-1 détruit toutes les valeurs des variables locales (ne retourne pas les variables locales, mais n'efface que la pile du micro-processeur).

- INPUT\$ et INPUT USING sont interrompus par ON TIMER GOSUB, comme avec 'Multitasking_between_Statements'.
- CALL exige "Multitasking_Between_Statements".
- Les programmes qui contiennent les ordres ON ERROR, ON TIMER, ON MOUSEBUT, ON KEY ou ON HELP doivent, en règle générale, comporter les deux mots clés Multitasking_Between_Statements" et "Trace_On" (ces deux instructions seront expliqués par la suite).
- Les fonctions doivent porter dans leur nom le type du résultat de la fonction. Si une fonction définie par vous doit retourner un réel simple précision, n'oublier pas le "!" (suffixe pour un réel simple). Pour les réels doubles, vous devez ajouter le "#" correspondant:

Correct :

```
DEF FN Total!(Montant!) : RETURN Montant!*1.14
```

Faux :

```
DEF FN Total(Montant!)
```

car il manque le suffixe pour les réels simples (le '!').

Vrai :

```
DEF FN Circonference*(Rayon) : RETURN Rayon*2* PI
```

la variable entière Rayon n'est pas gênante, car PI oblige le calcul en réel double !!! (voir le chapitre "types des fonctions").

Faux :

DEF FNA(X)=1/X : REM 1/X est un réel.

Il sera arrondi (par erreur) en entier.

DEF FNA*(X): RETURN X*2+17

une fonction réelle ne peut aucune façon retourner une valeur entière.

Correct :

DEF FNA(X)=X*2+17

Vous pouvez éviter toutes ces difficultés en passant par une fonction à plusieurs lignes :

DEF FN Circonference!(Rayon) : RETURN CSBG(Rayon*2*Pi)

En règle générale, le compilateur OMIKRON indique les erreurs de ce type sous la forme :

Warning : RETURN type mismatch

- Le type des données attendu par READ doit concorder avec les données lues.

Faux :

READ A!, B!, C! : DATA 1.5, 3, 4.5 :REM 3 est un entier.

Vrai :

READ A!, B!, C! : DATA 1.5, 3., 4.5 :REM 3. est un réel.

- Les variables du type %F (Flag) ne doivent pas être utilisées dans des procédures par LOCAL, DEF PROC ou DEF FN.
- Quand on utilise une variable du type %F dans une procédure, la variable de retour est vide, même quand "R Variable" est indiqué dans la définition de la procédure.
- Les caractères de calcul sont exploités dans n'importe quel ordre. Une expression de la forme :

FNA(X) + FNB(Y) avec :

**DEF FNA(X) : Y=X^2 : RETURN X^3
DEF FNB(Y) : RETURN Y**

indique soit $X^3 + X^2$ soit X^3+Y . Utilisez des parenthèses.

- La valeur de `ASC(“”)` est nulle ; il n'indique pas ? Illegal function call.
- Le compilateur n'effectue pas de vérification sur les dépassements de capacité. Par exemple:

`2000000000+1000000000` donne `-1294967296`.

Les deux nombres sont des entiers, le compilateur estime qu'il s'agit d'une addition d'entiers.

Pour l'interpréteur, il y a dépassement de la capacité des nombres entiers, et il passe automatiquement le résultat (`3000000000`) en réel. Pour le compilateur, un type de variable a été choisi au départ et ne peut pas être changé par la suite.

- Comme nous venons de le voir, un dépassement (?Overflow) n'est pris en compte qu'avec les nombres réels et non avec les nombres entiers. Vous devez vous assurer que `1000000000*10` comme `2000000000+1000000000` ne dépassent pas le domaine des nombres entiers.

Il est facile de dépasser les limites avec les entiers (`%` et `%W`, limites s'étendant de `-32768` à `32767`) et les octets (`%B`, domaine de calcul de `0` à `255`).

TYPES DES FONCTIONS

Le résultat est toujours du type :

Chaîne :	@ UPPER\$ LOWER\$ ERR\$ TIME\$ DATE\$ HEX\$ STR\$ OCT\$ BIN\$ INPUT\$ INKEY\$ CHR\$ MID\$ LEFT\$ RIGHT\$ STRING\$ MIRROR\$ MKD\$ MKS\$ MKIL\$ MKI\$ SPC SPACE\$
Réel simple :	RND CSGN CVS
Réel :	DET EXP LN LOG SQR FACT ^ SIN COS
(Simple ou Double) :	TAN COT ATN ARCSIN ARCCOS ARCTAN ARCCOT SINH COSH TANH COTH SECH COSECH ARSINH ARTANH ARCOTH SEC COSEC
Réel double :	PI VAL CDBL CVD
Entier :	CRSLIN ERR ERL MEMORY HIGH LOW SEGPTR TIMER MOUSEX MOUSEY MOUSEBUT EOF LOF LOC FRE LPOS POS USR PEEK WPEEK LPEEK INSTR LEN ASC VARPTR POINT CINTL CINT CVIL CVI BIT SGN NOT SHL SHR = < > <> >= <= AND OR IMP XOR EQV NAND NOR
Du type des arguments :	MIN MAX INT ABS FIX FRAC - * MOD + - () +1 -1 *2

Avec ces fonctions, il n'est pas toujours simple de déterminer le type de variable résultant, voici un exemple pour vous en convaincre :

INT(3333333333.33), soit 3333333333, est calculé en réel double et non pas en entier, le format "Entier-Long" ne suffisant pas.

ORGANISATION DE LA MEMOIRE

Système, Desktop

Basepage Lib.

Programm-Header	.L	Longueur du programme
	.L	Longueur du tableau des numéros de lignes
	.L	Longueur du tableau des labels
	.L	Longueur Relocation Info
	.L	Longueur du domaine (variables #)
	.L	Longueur du domaine (variables !)
	.L	Longueur du domaine (variables %)
	.L	Longueur du domaine (variables \$)
	.L	Longueur du tableau des pointeurs
	.L	Longueur du tableau Library (40/0)
	.L	Pointeur du premier DATA
	.L	Utilisation Interne
Programmes		Pêle-mêle de code optimisé
Numéros de lignes	.W	Numéro de lignes
	.L	Pointeur relatif
Tableau des Labels		8 Octets par Label
	.L	Pointeur relatif
Variables		A6 montre le départ +\$8000
Données pour V_Opnvwrk		
Pile		CLEAR,X
Tableaux	.L	Longueur des tableaux
	.W	Type :
		- 1=\$, 0=#, 1=!, 2=%L, 3=%W, 4=%B, 5=%F
	.W	Nombre de bits par éléments
	.W	Varptr (relatif à A6)
	.W	Dimensionne le nombre
	.W	Dimension (suite: 1,2, ..., n)
	.L	Pour Dim
Chaînes	.W	Longueur + 4 (+1)
	.L	Pointeur retour sur le pointeur
	.W	Numéro de fichier pointeur retour
	.L	Offset
Mémoire libre		Fre(O) et Fre(****)
LIBREMEMORY(-1)		CLEAR X
ECRAN		

SEGMENT POINTER

La variable système SEGPTR adresse le tableau de pointeurs suivant (comparez avec SEGPTR dans le manuel de programmation de l'interpréteur).

Décalage	Domaine de la mémoire
0	Tableau des numéros de lignes
4	Départ du code compilé
8	Fin du tableau des étiquettes
12	Départ des variables
16	Réservé
20	0 (voir page suivante)
24	Départ du buffer fichier
28	0 (voir page suivante)
32	Domaine libre
36	Pointeur de programme (uniquement utilisé par Trace_On")
40	Garbage-top (pointeur actuel)
44	Garbage-bottom
48	Garbage-high (fin de Garbage ; le plus souvent, la plus grande adresse de la mémoire)
52	Valeur maximale de la pile
56 - 60	La plus grande adresse de la mémoire de la pile système du micro-processeur (SSP)

Les nouveaux venus sont : (inexistants dans l'interpréteur !)

64	Adresse de basepage
68(.B)	Multitasking_Between_Statements", 'Multitasking_Always'

Flag doit uniquement être placé pour ces deux mots clés du compilateur.

70(inter : 20)	Début des tableaux
74(inter : 28)	Début des chaînes

Les deux derniers pointeurs ne servent qu'à assurer la compatibilité avec l'interpréteur. Il faut les utiliser comme suit avec le compilateur :

LPEEK(VARPTR(Texte\$))+LPEEK(SEGPTR+28)

et

LPEEK(VARPTR(Tableau(0,0)))+LPEEK(SEGPTR+20)

L'expression LPEEK(SEGPTR+X) est nulle dans les deux cas.

MOTS CLES DU COMPILATEUR

Le compilateur comprend plusieurs mots-clés qui ne servent pas dans l'interpréteur. Pour se servir d'un mot-clé, il suffit de taper :

COMPILER "Mot_Cle"

Le mot-clé peut être en majuscule ou en minuscule, cela n'a pas d'importance.

Trace On :

[Ctrl]-[C], les interruptions & le clavier sont vérifiés avant l'exécution de chaque instruction. Grâce à cela, les fonctions multitâches sont possible entre deux ordres. Les pointeurs pour ON ERROR et RESUME sont mis à jour.

Trace Off :

[Ctrl]-[C], les interruptions & le clavier ne sont pas vérifiés avant l'exécution de chaque instruction. Les fonctions multi-tâches ne fonctionnent pas ; ERL est toujours nul ; RESUME est impossible.

Multitasking Always :

[Ctrl]-[C] et les fonctions multitâches sont vérifiés même si il manque un Trace On. Les interruptions ont lieu n'importe quand, en plein milieu de l'exécution d'une fonction si il le faut (par exemple dans un SORT). Mais il n'est pas possible d'interrompre un accès disque (par exemple, [Ctrl]-[C] avec un BLOAD ne fonctionne pas).

Les routines multitâches, qui sont appelés par ON TIMER, ON KEY, ON HELP et ON MOUSEBUT ne doivent contenir aucune assignation de chaîne.

Multitasking Between Statements (MBS) :

Forme compatible avec l'interpréteur : les interruptions ont uniquement lieu entre les instructions (Trace On) ou pas du tout (Trace Off). Ces deux derniers mots-clés (MA et MBS) doivent à nouveau être remis après chaque CLEAR.

Voyons ce qui se passe quand on utilise deux mots-clés du compilateur en même temps :

MA & Trace Off :

Mise au point standard: aucun ERL, aucun RESUME. Lors de l'exécution d'une routine multi-tâche, aucune opération sur les chaînes, aucun DIM, aucun OPEN. L'exécution du sous-programme multi-tâche est possible pendant un SORT.

MBS & Trace Off :

Aucun ERL, aucun RESUME, pas de multitâche.

MA & Trace On :

Même chose que MA & Trace Off, jusqu'à 40 pour-cents plus lent. Aucune opération sur les chaînes, aucun DIM, aucun OPEN à l'intérieur du sous-programme exécuté par la fonction mutli-tâche.

MBS & Trace On :

Même chose, jusqu'à 40 pour-cents plus lent. Entièrement compatible avec l'interpréteur.

Les recouvrements de zone mémoire ne fonctionne pas compilé sans ces mots cles du compilateur

COMPILER "MEMORY MOVE BACK"

* Avant : MEMORY-MOVE Adresse, Quantité TO Adresse + Décalage

COMPILER "MEMORY MOVE FORWARD"

* Avant: MEMORY-MOVE Adresse + Décalage, Quantité TO Adresse

* Identique avec MEMORY-MOVB

LA VARIABLE COMPILER

COMPILER

Syntaxe : IF COMPILER THEN...

COMPILER est une variable dont la valeur est vraie quand le programme est compilé (-1) et fausse quand le programme est interprété (0).

COMPILATION CONDITIONNELLE

Il est maintenant possible d'interdire la compilation d'une partie d'un programme. Pour ce faire, on utilise le mot-clé :

COMPILER

Syntaxe : COMPILER [ON/OFF]

COMPILER OFF interdit la compilation des lignes qui suivent, tant que le mot-clé COMPILER ON n'est pas rencontré. A quoi cela peut-il servir?

Voici un très joli exemple, écrit par Dietrich RAISIN (le développeur de DRAW et de bien d'autres programmes en OMIKRON).

```
COMPILER OFF
DEF PROC E : Appl_Exit : EDIT
DEF PROC C
EXEC " c:\compiler.prg "," where_is.BAS"
EXEC " c:\cutlib.prg","where_is.PRG -NJJ"
END
COMPILER ON
```

En tapant cette petite routine et en l'insérant dans votre programme (en l'adaptant, naturellement!), vous pourrez compiler et cutliber le programme en mémoire. Il suffit de taper 'C' sous éditeur standard, et ça part. Cette routine ne sera pas compilée, elle sera ignorée.

COMMENT INTERDIRE LE [CTRL]-[C] DANS UN PROGRAMME ?

Il suffit de taper ce qui suit :

```
IPL 3+1 SHL 31
```

pour revenir au mode normal, tapez :

```
IPL 3
```

OPTIMISATION

Ce chapitre a été conçu pour les (nombreux) adeptes de la vitesse.

Types des variables

Avant d'utiliser des nombres réels, réfléchissez. Dans beaucoup de cas, les entiers suffisent largement. Les entiers ont plusieurs avantages:

- ❑ Aucune perte de précision (fort heureusement!)
- ❑ Vitesse (au minimum 1,5 fois plus vite en entier)
- ❑ Taille réduite (4 octets pour les entiers longs, contre 10 octets pour les réels double précision).

Types de calculs

Certaines fonctions calculent automatiquement en réels.

Exemple : $A = \text{SQR}(I) + J - I * 3$

La racine carrée donne un résultat réel, c'est pourquoi le reste de la ligne est aussi calculé en réel. Dans cette ligne, le fait de calculer en réel ne sert vraiment à rien, puisque la variable receptrice est de type entier. Voici le même calcul, avec la même précision, en plus rapide :

$A = \text{SQR}(I) + (J - I * 3)$

Maintenant, la suite de calcul situé après la deuxième parenthèse est exécutée en entier. Le résultat est converti en réel pour être ajouté à la racine.

Voyons une solution encore plus rapide : $A = \text{CINT}(\text{SQR}(I)) + J - I * 3$ (sur cet exemple, on gagne 24 % par rapport au premier programme)

Avec CINT (Convert to Integer), le résultat réel obtenu par la racine carrée est converti en entier.

Attention: la division normale ("/") donne un résultat réel !!! Des lignes comme : $A = A / 5$ doivent absolument utiliser la division entière ("\ \backslash ")

$A = A / 5$ (on gagne au moins 20 % du temps)

Parfois, cela vaut la peine de remplacer une constante réelle par une fraction entière :

$A = B * 2.5$ est calculé en réel (98,7 msec)

$A = B * 5 \backslash 2$ est calculé en entier (39,8 msec).

Constantes

Compactez au maximum les constantes entre parenthèses. En effet, les constantes entre parenthèses sont évaluées à la compilation et non pas à l'exécution.

Exemple : $A=B*(3*4)$ est automatiquement compacté en $A=B*12$.

Alors que $A=B*3*4$ n'est pas compacté.

Pensez que le désordre : $A=J*2*K*5$ peut être ordonné en $A=J*K*(2*5)$... et le compilateur peut optimiser!

Chaînes

Concentrez autant que possible les fonctions portant sur les chaînes. La réorganisation de la mémoire et les sauvegardes intermédiaires demandent beaucoup de temps. Au lieu de :

```
A$ = "Compilateur OMIKRON"
B$ = LEFT$(A$,7) C$= RIGHT$(A$,8)
D$ = C$+B$
REM Ecrivez :
A$ = "Compilateur OMIKRON"
D$= RIGHT$(A$,8) + LEFT$(A$,7)
```

(Naturellement, si vous avez besoin des résultats intermédiaires, cette astuce ne vous convient pas !!!)

IF..THEN

Le compilateur peut aussi optimiser les comparaisons quand elles sont sous la forme suivante :

IF $A>B$ THEN... A et B peuvent être n'importe quelle expression de calcul au lieu de ">", ce peut être "<","=",">=",etc..., par exemple:IF $3*4+J = \text{nombre}(I)/2$ THEN... est aussi optimisé. Le compilateur peut moins bien optimiser les expressions suivantes : IF(A>B) AND (B<C) THEN...pour accélérer, il vaut mieux écrire : IF $A>B$ THEN IF $B<C$ THEN...

Les comparaisons entre parenthèses sont elles aussi difficilement optimisables : IF (A>B) THEN...est plus long que IF $A>B$ THEN...

N.B.: Dans le premier cas, il s'agit d'une fonction "()", dont l'argument est une comparaison; dans le deuxième cas, il s'agit d'une comparaison.

Boucles imbriquées

Si possible, utilisez le plus souvent la boucle parcourue en interne.

PROGRAMMES D'AIDE

GETADDR

Avec GETADDR, vous pouvez obtenir l'emplacement (Adresse) d'une ligne du code source au sein du programme compilé.

Fonctionnement : Vous indiquez le nom du programme (GETADDR charge le programme). Vous indiquez le numéro de ligne (GETADDR affiche l'adresse à l'écran). L'adresse découverte ne peut être utilisée directement, c'est un décalage par rapport au pointeur '.TEXT'. Pour réellement débiter, vous devez déjà lier votre programme à la bibliothèque BASLIB_3. Il faut donc d'abord utiliser CUTLIB avec les paramètres 9,9,9 avant d'utiliser GETADDR et de débiter.

CUTLIB

CUTLIB est le programme qui permet d'intégrer dans le .PRG la bibliothèque BASLIB_3. CUTLIB ne met que les routines indispensables, ce qui permet de gagner de la place. Pour que CUTLIB fonctionne, vous devez avoir sur la disquette les trois fichiers suivants : BASLIB_3, BASLIB.REQ, BASLIB.REL

Démarrez CUTLIB. Un sélecteur de fichier s'affiche. Choisissez le programme que vous désirez cutliver. CUTLIB vous pose une, deux ou trois question.

Interruption du programme par [CTRL]-[C] (O/N). CUTLIB vous demande si le programme doit-être ou non interruptible par [CTRL]-[C]. Répondez par O ou N.

Fonctions étendues d'affichages en OMIKRON (O/N). CUTLIB vous demande si le programme contient ou non des fonctions étendues d'affichage. Dans les fonctions étendues d'affichage, on trouve par exemple: les fenestres texte (VT-52), les lignes liées avec SCREEN 0, CSRLIN, POS(0), HCOPY TEXT...

Fonctions étendues de saisie en OMIKRON (O/N). CUTLIB vous demande si le programme contient ou non des fonctions étendues de saisie. Dans les fonctions étendues de saisie, on trouve par exemple :

Annulation du buffer clavier par [ALT]-[CTRL], KEY n=, ON HELP GOSUB, OPEN 'K', [CTRL]-[C]...

A chaque réponse positive, la bibliothèque correspondante est ajoutée au programme. Bien entendu, CUTLIB enlève les fonctions inutilisées.

Il est possible d'intégrer la bibliothèque complète dans le programme. Pour ce faire, il faut utiliser le programme SHELL ou bien demander dans le bureau à ce que CUTLIB prenne des paramètres au démarrage ou bien encore inspirez-vous de la routine indiquée dans ce manuel (COMPILER ON|OFF). La deuxième série de paramètres doit contenir -999, ce qui indique qu'on veut introduire la totalité de la bibliothèque. Voici la syntaxe des paramètres de CUTLIB :

CUTLIB <Nom_Programme>[,<Nom_Programme>...][-OOO][-999]

Comme vous pouvez vous en apercevoir, il est possible de cutliber plusieurs programmes à la fois. Mais attention : Tous les programmes seront cutlibés avec les mêmes paramètres. Le premier groupe de paramètres indique les trois réponses aux trois questions (O pour OUI, N pour NON). Le deuxième groupe de paramètres est obligatoirement -999, et il indique qu'on veut que la totalité de la bibliothèque soit contenue dans le programme.

SHELL

SHELL est un interpréteur de commande. Les commandes autorisées sont les suivantes :

CHDIR <Chemin> (CD <Chemin>) : Le chemin d'accès par défaut est le chemin indiqué.

MKDIR <Chemin> (MD <Chemin>) : Création d'un répertoire.

RMDIR <Chemin> (RD <Chemin>) : Destruction d'un répertoire.

CLS : Effacer l'écran.

COPY <Fichier_Départ> <Fichier_Arrivée> : Copier le fichier indiqué sur le chemin indiqué.

DATE [<Date>] : Mise en place de la date.

TIME [<Heure>] : Mise en place de l'heure.

DEL <Nom_Fichier> : Détruire le fichier spécifié.

DIR [<Sélection>] : Afficher le directory du disque.

ECHO ON et ECHO OFF : Eteindre les messages à l'écran ou les allumer (très utile pour les fichiers BATCH).

EXIT : Quitter le SHELL.

REN <Ancien_Nom> <Nouveau_Nom> : Renommer le premier fichier sous le deuxième nom.

PAUSE : Attendre une touche.

REM : Remarque

PROMPT <Caractere_Prompt> : PROMPT permet de choisir ce que vous voyez au début de chaque ligne. Les commandes du PROMPT sont toujours précédées d'un \$. Il y a 5 commandes:

\$d affiche la date. \$t affiche l'heure. \$n: affiche le lecteur actif (A: ou C:...), \$p affiche le chemin actif (OMIKRON\COMPI). \$g affiche le signe >

TYPE <Nom_Fichier> : Lister un fichier. Interruption par [CTRL]-[S] et reprise par [CTRL]-[Q].

VER : Retourne la version de GEMDOS.

<Nom_Programme> [<Paramètre>] : Démarre le programme indiqué avec les paramètres qui suivent.

<Fichier_Batch> [<Paramètre>] : Démarre un fichier batch utilisable sous SHELL. Toutes les entrées que l'utilisateur doit taper sont indiquées sous la forme: %n. Le premier paramètre est %1, le deuxième %2 etc. Vous trouverez plusieurs fichiers batch sur la disquette compilateur.

OMIKRON.

BASIC

Le Basic de l'Atari ST

OMIKRON. FRANCE

11, rue Dérodé - 51100 REIMS - TEL. 26.02.60.44 - R.C. Reims B 343 750 972