Le H-BASIC

Réalisé par Olivier Herz

Diffusé par Pom's - Editions MEV

TABLE DES MATIERES

Pom's - Editions MEV - 49, rue Lamartine - 78000 Versailles

INTRODUCTION

POURQUOI UN NOUVEAU LANGAGE?

En pratiquant un peu le BASIC, on s'aperçoit très vite des limites du langage et peut-être un peu moins vite des mauvaises habitudes qu'il fait prendre: les GOTOs rendent les programmes difficilement lisibles, l'absence de variables locales et de vrais sous-programmes empêche de traiter des problèmes récursifs et rend difficile la structuration du programme.

Pour pallier ces problèmes, on peut passer au Pascal, mais ce langage coute cher, est plus complexe, plus difficile à débugger, ne possède pas de graphisme basse-résolution et est très lourd dès qu'il s'agit d'utiliser des routines assembleur. On peut aussi faire du LISP ou du LOGO, mais leur spécificité en fait des langages un peu à part. On peut enfin faire du FORTH (nous reparlerons certainement de ce langage plein d'avenir).

Nous avons préféré créer un langage hybride possédant les structures de bloc du Pascal et "collant" de près à l'Applesoft. Deux possibilités s'offraient pour le mettre en oeuvre: l'amper-interpréteur (c'est-à-dire rajouter des commandes au BASIC par le biais de l'ampersand &), qui posait des problèmes de lisibilité du listing, et le compilateur (c'est-à-dire un programme traduisant un programme source écrit en H-BASIC en un programme objet écrit en Applesoft), solution retenue.

LA CREATION DU LANGAGE:

Le processus de création du langage s'est effectué en cinq étapes successives, à vrai dire pas très bien définies dans le temps, la finition du produit s'étant étalée sur une longue période.

1ère étape: il fallait d'abord définir le langage en pensant aux objectifs et aux contraintes (compilateur simple et pas trop lent à réaliser, compatibilité avec l'Applesoft).

2ème étape: il fallut ensuite écrire les routines en assembleur, c'est-à-dire d'une part les quelques routines destinées à simplifier et accélérer le compilateur, et d'autre part les routines de pile nécessaires pour faire tourner un programme une fois compilé en Applesoft.

3ème étape: écrire le compilateur. Plusieurs langages étaient possibles: le BASIC, mais la nature récursive du compilateur était une condition rédhibitoire; le Pascal, mais cela posait des problèmes pénibles de transfert d'un système d'exploitation de disquettes à l'autre. La solution retenue fut finalement la plus rationnelle: écrire le compilateur en H-BASIC.

4ème étape: il fallut alors compiler ce compilateur pour en obtenir une version Applesoft exécutable. Bien sur, puisque le langage n'était pas encore développé, il fallut réaliser l'opération à la main.

5ème étape: il ne restait alors plus qu'à rendre le compilateur fiable par approximations successives, la Nème version Applesoft compilant la N+1ème version H-BASIC. En fait, cette étape vit aussi une évolution du langage jusqu'à sa forme actuelle, et rien n'empêche l'utilisateur de la poursuivre pour modifier le langage à sa guise.

CHAPITRE I: LA SYNTAXE DU H-BASIC

Ce chapitre décrit le langage. Il est conseillé au lecteur de le lire en ayant sous les yeux les programmes H-BASIC fournis avec le package, à commencer par le compilateur.

LES INSTRUCTIONS ET LES COMMANDES:

On appelle 'commande' une instruction élémentaire du langage, qui peut être soit une déclaration, soit un mot-clef comme BEGIN ou END, soit encore une véritable instruction du programme. A l'instar du BASIC, les commandes sont séparées par le symbole deux-points ou par un retour-chariot. Les deux points constituant un séparateur de commandes, on ne peut en mettre à l'intérieur d'une commande, dans une remarque par exemple, qu'en les entourant de guillemets, comme avec REM "A:B".

Notons que l'on peut écrire un programme H-BASIC avec des minuscules, le compilateur rétablissant les majuscules (sauf bien sur dans le cas de minuscules entre guillemets). De même, les blancs qui ne sont pas entre guillemets n'ont aucune importance. NOTE: une commande ne comportant que des blancs, ou même une commande vide, sera ainsi ignorée par la routine de lecture de commande sur disque.

Toute commande commençant par deux tirets (--) est considérée comme une remarque H-BASIC et n'est pas compilée. A ne pas confondre avec l'instruction REM qui est traduite telle quelle (voir plus bas: les instructions simples).

Toute commande commençant par le pour-cent (%), y compris une commande qui ne comprend que ce signe, indique au compilateur qu'il faut "aller à la ligne" dans le programme compilé.

LA STRUCTURE PAR BLOCS:

Le H-BASIC possède une structure par blocs semblable à celle de Pascal ou d'ADA. Toutefois, on ne fait plus la différence entre PROGRAM, PROCEDURE et FUNCTION: il n'y plus que des PROCEDUREs. Cela permet à un programme de s'appeler lui-même comme

sous-programme. La notion de fonction a été abandonnée car le compilateur laisse inchangées les expressions numériques et alphanumériques. Toutefois, les fonctions FN du BASIC restent valables, à condition bien entendu qu'elles soient définies avant d'être appelées.

Un bloc commence donc par la commande PROCEDURE [nom de procédure] (où le nom de procédure peut être n'importe quelle chaine de caractères alphanumériques; les signes de ponctuation sont à éviter), suivie par d'éventuelles déclarations de paramètres et de variables, puis par zéro, un ou plusieurs blocs (d'où la nature récursive du compilateur), enfin par des instructions encadrées par les commandes BEGIN ou END.

NOTE: un programme est ainsi constitué par un bloc unique, toutefois toutes les commandes précédant la commande PROCEDURE de ce bloc sont traduites telles quelles par le compilateur. Il s'agit presque toujours de commandes mettant en place le vecteur de l'ampersand en direction des routines de pile, ce qui doit être fait avant le premier appel de procédure.

```
PROCEDURE PRINCIPALE

PROCEDURE INTERNE1

PROCEDURE INTERNE2
BEGIN
...
END

BEGIN -- INTERNE1
...
END

PROCEDURE INTERNE3
BEGIN
...
END

BEGIN -- PRINCIPALE
...
END
```

La procédure PRINCIPALE peut appeler comme sous-programmes les procédures INTERNE1 et INTERNE3; INTERNE1 et INTERNE2 peuvent s'appeler l'une l'autre, ou bien appeler PRINCIPALE, mais ne peuvent pas appeler INTERNE3 (déclarée plus tard); INTERNE3 peut appeler PRINCIPALE et INTERNE1, mais pas INTERNE2: INTERNE2 est "invisible" à l'extérieur de INTERNE1. On pourrait la rendre visible en changeant légèrement le langage: il suffit de supprimer dans le compilateur, à la fin de la procédure PROCEDURE, les commandes IF NOM()-1: FOR N=0 TO NOM: NOM$(NOM%(N))="": ENDFOR: ENDIF.

LES VARIABLES EN H-BASIC:

Les variables du H-BASIC sont les mêmes que celles de l'Applesoft: deux lettres significatives, un dollar ou un pour-cent, nom ne contenant pas de mot-clef de l'Applesoft, etc.

La seule différence concerne les tableaux: dans les expressions (alpha)numériques, ils sont utilisés comme en BASIC, mais les déclarations sont différentes et on peut les affecter entre eux. Un tableau référencé ainsi globalement prend un dièse devant son nom.

DECLARATIONS DE PARAMETRES:

Il y a trois sortes de paramètres passés aux procédures: les paramètres IN (passés à la procédure, qui ne les retourne pas), les OUT (retournés par la procédure, mais non passés) et les INOUT (passés et retournés).

La déclaration a alors l'une des formes suivantes IN A$,#B%(1,2),!C ou OUT #A(10),!N$ ou encore INOUT A,B,I$.

Les paramètres INOUT doivent être déclarés en premier (s'ils existent), puis les IN et les OUT, une seule commande de chaque type étant permise. Les tableaux doivent être dimensionnés comme ci-dessus et le point d'exclamation, devant une variable simple, indique qu'elle doit être empilée au moment de l'appel de la procédure (ceci afin de la rendre locale pour éviter un conflit avec une variable plus globale de même nom - pour des raisons techniques, les tableaux sont toujours empilés).

NOTE: nous ne considérons pas les éléments de tableaux comme des variables simples, car les routines de pile ne peuvent les empiler. Ainsi, si la commande IN A(1) est tolérée, la commande OUT !A$(0) est absolument interdite, bien que le compilateur ne la détecte pas, laissant l'erreur se déclencher à l'exécution.

ATTENTION: le débutant prendra bien soin de mettre un point d'exclamation pour toutes les variables simples, même ci cela doit ralentir le programme. Une fois le programme mis au point, il lui sera loisible de supprimer les points d'exclamation qui s'avèrent inutiles.

DECLARATION DE VARIABLES:

Les variables sont déclarées comme les paramètres, le mot-clef étant non plus IN, OUT ou INOUT, mais VAR. De plus, les variables peuvent être initialisées en mettant une affectation derrière le nom de variable: var #A(1,2)=#B, A$="HELLO", !V$, !I=3*WW, J=!I, K=I, A=!B, B=!A, C=D, D=C.

Le point d'exclamation derrière le signe '=' indique au compilateur qu'il faut stocker l'expression suivant ce signe avant les empilements et faire l'affectation après (sinon, l'affectation est faite directement après empilement et les précédentes affectations de cette commande VAR, et la valeur de certaines variables de l'expression suivant le signe '=' peut être modifiée): ainsi dans cet exemple, à J est affectée la valeur d'une variable globale I ou d'un paramètre s'il y en a un qui porte ce nom, tandis qu'à K est affectée celle de la variable locale I=3*WW; à A et B sont affectées les valeurs de B et A tandis qu'à C et D sont affectées deux fois la valeur antérieure de D. ATTENTION: ceci n'est pas valable pour les tableaux et des instructions telles que VAR #A(10)=#B,#B(10)=#A ou VAR #A$(10)=#A$ sont malheureusement à proscrire.

Une variable qui n'est ni empilée ni initialisée peut ne pas être déclarée, mais c'est là une chose dangereuse qu'il est préférable d'éviter. ATTENTION: il est par contre obligatoire de déclarer les tableaux, et l'instruction DIM est prohibée, sous peine d'obtenir des messages d'erreur de l'Applesoft à l'exécution du programme compilé pendant les routines de pile. Pour la même raison, le compilateur laissant telles quelles les expressions (alpha)numériques, il est absolument interdit d'utiliser dans une expression un élément d'un tableau qui n'a pas été déclaré, l'Applesoft le dimensionnant automatiquement dans ce cas avec les valeurs par défaut des dimensions indiquées dans son manuel de référence. De plus, nous conseillons fortement au débutant de mettre tous les points d'exclamation d'empilement des variables, et de n'initialiser les variables qu'avec des constantes (ce qui rend inutile l'emploi du point d'exclamation suivant le '='), l'affectation initiale des variables pouvant très bien être réalisée dans le corps de la procédure.

NOTE: là encore, VAR A(1)=10 est toléré, tandis que VAR !A$(0) est absolument interdit.

L'APPEL DE PROCEDURE:

Pour appeler une procédure (qui doit avoir été déclarée au-dessus ou dans la procédure appelante), on utilise l'arobase (@), suivi du nom de la procédure et d'une éventuelle liste de paramètres entre parenthèses. Le nombre et les types des paramètres doivent obligatoirement correspondre au nombre et aux types de ceux déclarés dans les commandes IN, OUT et INOUT de la procédure appelée. Les paramètres passés peuvent être des expressions dans le cas où les paramètres déclarés correspondants sont déclarés dans un IN, mais doivent bien sur être des variables (simples ou tableaux) dans le cas de OUT et INOUT. NOTE: comme les paramètres passés dans la liste ne sont pas empilés, il est toléré de passer un élément de tableau à une variable simple de type OUT ou INOUT.

Exemple: @ TOTO(3*A,#B,!C(1)). Notons que les tableaux ne doivent pas être dimensionnés dans l'appel, mais que la taille d'un tableau passé doit correspondre à celle du tableau receveur de la procédure appelée, sous peine d'erreurs à l'exécution dans les routines de pile. La présence du point d'exclamation avant l'expression indique que, à l'entrée (IN), à la sortie (OUT) ou aux deux (INOUT), la valeur passée est stockée avant le dépilement ou l'empilement et affectée après. Cela permet de faire des passages de paramètres du genre:

```
PROCEDURE TOTO: IN !A,!B: OUT !C
BEGIN ... END
PROCEDURE TITI: VAR A,B,C
BEGIN:..@ TOTO(!B,!A,!C)..:END
```

Par contre, le point d'exclamation est interdit avec les tableaux, et des passages de paramètres semblables avec des tableaux ne sont pas possibles: il faut donc être très prudent dans le passage des tableaux. De plus, nous répétons au débutant que rajouter de tels points d'exclamation ne coute que du temps d'exécution et permet d'éviter des déboires.

NOTE: étant donné que, à cause de la structure de l'Applesoft, le H-BASIC fait ses empilements de variables au moment de l'appel des procédures plutot que de gérer ses variables par zones locales et globales comme le fait Pascal, on en déduit une légère différence d'avec ce langage en ce qui concerne les zones de visibilité des variables (alors que le H-BASIC est semblable au Pascal en ce qui concerne la visibilité des procédures).

Le Pascal imprime 20:

```
PROGRAM XX;
VAR A:INTEGER;
  PROCEDURE YY;
  BEGIN WRITELN(A) END;
  PROCEDURE ZZ;
  VAR A:INTEGER;
  BEGIN A:=10; YY END
BEGIN (*XX*) A:=20; ZZ END.
```

là où le H-BASIC imprime 10:

```
PROCEDURE XX: VAR A
  PROCEDURE YY
  BEGIN: PRINT A: END
  PROCEDURE ZZ: VAR A
  BEGIN: A=10: @ YY: END
BEGIN -- XX: A=20: @ ZZ: END
```

Cette visibilité des variables, bien que moins "belle" qu'en Pascal, offre une plus grande souplesse (voir les procédures BOUCLE et POP du compilateur par exemple).

LES BOUCLES D'INSTRUCTIONS:

Il y a trois types de boucles, qui se comprennent aisément: les boucles FOR (dont la syntaxe suivant le FOR est celle du BASIC), WHILE ("tant que") et UNTIL ("jusqu'à ce que") dont voici des exemples.

```
FOR I=1 TO 10: PRINT I: ENDFOR

WHILE I<2000
  PRINT I: I=2*I
ENDWHILE

UNTIL I<.3: I=.3*I
ENDUNTIL
```

Le nombre de FOR emboités est limité comme en Applesoft à 10 niveaux, sous peine d'erreur Applesoft du programme compilé.
NOTE: les FOR suivants sont considérés comme emboités.

```
PROCEDURE EXTERNE: VAR I

PROCEDURE INTERNE: VAR !I
BEGIN
  FOR I=1 TO 3: @ TOTO: ENDFOR
END

BEGIN -- EXTERNE
  FOR I=1 TO 3: @ INTERNE: ENDFOR
END
```

ATTENTION: on remarque qu'on peut ainsi emboiter des boucles FOR avec le même nom de variable, mais il est indispensable que la variable de la boucle interne soit empilée. Sinon, cela revient à emboiter des boucles FOR en Applesoft avec le même nom de variable, ce qui ne marche pas bien.

NOTE: Une commande BEGIN, END ou ENDmachin peut comporter des caractères derrière le BEGIN... Ceux-ci seront considérés comme une remarque par le compilateur. Ainsi BEGIN HELLO et BEGIN: -- HELLO sont équivalents.

LES INSTRUCTIONS DE CONTROLE:

Il y en a encore trois types: le IF..THEN..ELSIF..ELSE.. ("si..alors..sinon si..sinon.."), le CASE..WHEN..WHENOTHERS ("cas..quand..quand les autres..") et le DO..ONERR.. ("faire..en cas d'erreur..")

```
IF A=1: THEN B=2          (le THEN est
ELSIF A=2: THEN B=4        facultatif)
ELSIF A=3: THEN B=8
ELSE B=0
ENDIF
```

```
IF X*X=1: A=1: ELSE: A=0: ENDIF

IF X: PRINT X: ELSIF Y: Y=0: ENDIF

CASE 3*A+2
  WHEN 1,2: B=0: A=2
  WHEN 3: B=1: A=2
  WHENOTHERS: B=-1: A=0
ENDCASE

CASE$ NOM$                (noter le $
  WHEN "THIRIEZ"          derrière CASE)
    PRINT "HELLO"
    PRINT "HOW DO YOU DO?"
  WHEN "HERZ"
    PRINT "SALUT"
    PRINT "COMMENT VAS-TU?"
ENDCASE

DO: PRINT D$"DELETE"N$
ONERR
  IF PEEK(222)<>6
    PRINT "ERREUR E/S"
  ENDIF
ENDDO
```

Les instructions entre ONERR et ENDDO sont exécutées dès qu'une erreur Applesoft ou DOS est rencontrée entre DO et ONERR et seulement dans ce cas. Notons que le compilateur met un POKE 216,0 (et donc que les erreurs ne sont plus récupérées) s'il rencontre une instruction POP, RETURN ou STOP. S'il rencontre un appel de procédure, les erreurs ne sont pas récupérées pendant l'exécution de la procédure appelée, mais le sont avant et après l'appel.

NOTE: le séparateur de commandes (les deux-points) après THEN, ELSE, WHENOTHERS, DO et ONERR est facultatif.

LES INSTRUCTIONS SIMPLES:

HIMEM= et LOMEM= remplacent HIMEM: et LOMEM: car les deux points sont compris par le compilateur comme un séparateur de commandes.

#A=#B est l'affectation de tableaux: ils doivent être de même type et de même taille sous peine d'erreur Applesoft à l'exécution.

READ fonctionne comme en BASIC, à la différence près qu'au premier READ rencontré dans une procédure, le pointeur de DATA est placé au début de la procédure. Ainsi le READ fonctionne-t-il de façon locale.

STOP provoque une sortie prématurée du programme, RETURN de la procédure courante et POP de la dernière boucle engagée. Cette sortie peut être conditionnelle avec un WHEN, comme par exemple: RETURN WHEN A=1.

La plupart des instructions Applesoft sont valables en H-BASIC et sont traduites telles quelles par le compilateur. Celles qui ne conviennent pas sont celles reconnues par le compilateur comme instructions H-BASIC (END, FOR, READ, POP, IF et STOP) et celles qui ne sont pas possibles (HIMEM:, LOMEM: et DIM). D'autres enfin ne déclenchent pas les foudres du compilateur, mais sont à éviter (NEXT, DEL, ONERR, RESUME, GOTO, RUN, RESTORE, GOSUB, RETURN, ON, CONT, LIST, CLEAR et NEW).

Enfin, dans les cas ambigus, on peut faire commencer la commande par le crochet fermant (]): tout ce qui suit sera traduit tel quel par le compilateur. Cela sert par exemple à faire un]STOP ou un]IF A=1 THEN PRINT. ATTENTION: dans ce dernier cas, il faudra bien sur que la prochaine commande H-BASIC commence par un pour-cent (%), de façon à passer à la ligne suivante dans le programme Applesoft compilé. De plus, il faudra se méfier de l'erreur de débutant en H-BASIC qui consiste à écrire par exemple:]IF A=1 THEN TOTO.

On remarque ainsi que le H-BASIC ne possède pas de GOTO: il est en effet suffisamment structuré et possède assez d'instructions de controle (POP, RETURN, etc.) pour pouvoir se passer du fléau que constituent les GOTOs dans un langage.

Il faut tout d'abord dire que l'assembleur utilisé est le LISA 2.5, que les fichiers sources de la disquette ont à la fin le suffixe ".LISA" et que les fichiers objets n'ont pas de suffixe.

LES ROUTINES INPUT:

INPUT constitue les routines dont le compilateur à besoin. Il s'agit d'un petit amper-interpréteur relogeable situé normalement entre $8000 et $8100. Le mot-clef INPUT doit suivre l'ampersand (&), sinon la main est donnée aux routines de pile (voir ci-dessous). La reprogrammation du vecteur de saut pour l'ampersand se fait dans ce cas par *3F5: 4C 00 80 ou]POKE 1013,76: POKE 1014,0: POKE 1015,128.

- & INPUT ;A$ permet de lire une commande à l'entrée de caractères (sur le disque dans le cas du compilateur). A$ contiendra tous les caractères lus jusqu'au premier retour chariot ou jusqu'aux premiers deux-points qui ne soient pas entre guillemets. De plus, les minuscules sont remises en majuscules, sauf celles entre guillemets; les caractères de controle et les blancs sont éliminés, sauf ceux entre guillemets. Enfin, les chiffres en tête de la commande sont aussi éliminés, ce qui permet d'utiliser comme programme H-BASIC un fichier TEXT constitué d'un pseudo-programme BASIC (voir le chapitre III: comment utiliser le H-BASIC?). Cette routine utilise un buffer situé entre $8100 et $8200 car le buffer normal situé entre $200 et $300 posait certains problèmes.
- & INPUT [caractère] A$, où le caractère peut être dans le cas du compilateur le signe '=', une virgule ou une parenthèse ouvrante retourne dans la mémoire $FF (lue par un PEEK(255)) la position du premier caractère en question trouvé dans A$ et qui ne soit pas entre guillemets, ni entre parenthèses dans le cas du signe '=' et de la virgule. Lorsque le caractère n'est pas trouvé, LEN(A$)+1 est retourné.

LES ROUTINES DE PILE:

Ces routines contiennent l'ensemble des sous-programmes assembleur nécessaires à l'exécution d'un programme H-BASIC compilé en Applesoft. Elles sont situées entre $7D00 (32000 décimal) et $8000 et utilisent une pile située entre $8000 et $9000, le haut étant en $9000 (c'est-à-dire les premiers octets empilés). Le compilateur utilise en fait ses propres routines de pile car le bas de sa pile est en $8200 à cause de la présence des routines d'input et de leur buffer. Cette pile fonctionne de manière LIFO (Last In First Out: dernier entré, premier sorti), règle qu'il faut absolument respecter si on utilise ces routines "à la main". De plus toute instruction qui amène à dépiler ou empiler hors de la pile amène un OUT OF MEMORY ERROR.

On peut les appeler, soit avec un CALL et une virgule comme séparateur (CALL 32000, etc.), soit avec l'ampersand (&) reprogrammé (*3F5: 4C 03 FD ou !POKE 1013,76: POKE 1014,3: POKE 1015,125, ce qui donne &etc.; ou bien *3F5:4C 00 FD ou]POKE 1013,76: POKE 1014,0: POKE 1015,125, ce qui donne &,etc.).

NOTE: le programme H-BASIC COMPILATEUR reprogramme l'ampersand en direction de la routine d'input qui le redirige vers les routines de pile si elle ne rencontre pas INPUT derrière l'ampersand.

- & = réinitialise la pile en remettant le pointeur de pile au sommet ($9000).
- &) [numéro de ligne] met le pointeur de DATA au début de la ligne en question (de la suivante, si elle n'existe pas; s'il n'y a pas de suivante on obtient un UNDEF'D STATEMENT ERROR): le prochain READ lira la première instruction DATA située à partir de cette ligne.
- & #[nom de tableau 1]=#[nom de tableau 2] réalise l'affectation du 2ème tableau au premier. Si un des tableaux n'existe pas, on obtient un OUT OF DATA ERROR. S'ils ne sont pas de même type ou si leurs dimensions diffèrent, on obtient un TYPE MISMATCH ERROR (en fait seule leur taille est comparée et il se peut que deux tableaux aient la même taille avec des dimensions réparties différemment: l'affectation dans ce cas peut produire des résultats étranges). Il se peut aussi que l'on obtienne un OVERFLOW ERROR: en fait, cette erreur ne doit jamais arriver; si elle se produit, cela signifie qu'il y a eu une catastrophe dans le stockage mémoire des variables et tableaux, et il vaut alors mieux tout effacer et recommencer.
- & GOSUB et & RETURN permettent un emboitement de sous-programmes Applesoft qui n'est limité que par la taille de la pile, alors que l'Applesoft limite, lui, le nombre de sous-programmes emboités à 24. Si le & RETURN a lieu au mauvais moment (c'est-à-dire si la règle LIFO n'est pas respectée), on obtient un RETURN WITHOUT GOSUB ERROR.

NOTE: les octets empilés sur la pile sont le pointeur de programme (TXTPTR=$B8,B9), le numéro de ligne courante (CURLIN=$75,76) et le token du GOSUB ($B0).

- &) [nom de variable] empile une variable. S'il elle n'existe pas, elle est créée et initialisée à 0 ou "" selon le cas.

NOTE: les octets empilés sur la pile sont les deux octets du nom et les deux de l'adresse de la variable; tandis qu'on "cache" la variable en lui donnant le nom AT, AT% ou AT$ selon le cas (AT ne peut être un nom légal de variable car c'est un mot-clef de l'Applesoft), ce qui permet de recréer une autre variable portant ce nom. La méthode est donc différente de celle de HAIFA (voir Pom's 5) où l'on empilait le descripteur de la variable (c'est-à-dire sa valeur si elle est numérique ou l'adresse de sa chaine si elle est alphanumérique), méthode qui possédait l'inconvénient de perdre parfois les chaines empilées, car elles n'étaient plus référencées par une variable.

ATTENTION: bien que la routine ne le signale pas, il est absolument interdit d'empiler un élément d'un tableau, pour la bonne raison que le nom AT irait abimer l'élément précédent dans le tableau.

- & ([nom de variable] dépile une variable: s'il existait une variable (locale donc) portant ce nom, elle sera perdue. Si

le nom qui avait été empilé ne correspond pas à celui suivant le signe ')' (c'est-à-dire si la règle LIFO n'est pas respectée), on obtient un TYPE MISMATCH ERROR.

- &) #[nom de tableau] empile un tableau. Une fois le tableau empilé, il est absolument nécessaire de dimensionner un nouveau tableau portant ce nom (le compilateur le fait bien entendu automatiquement), sous peine de "plantaison" au dépilement.

NOTE: les octets empilés sont les deux octets du nom du tableau, et les deux octets de son adresse relative à l'adresse (ARYTAB=$6B,6C) du début de la zone de stockage des tableaux (l'adresse absolue du tableau ne marche pas, car la création de variables simples déplace la zone des tableaux; si le tableau n'existe pas, on empilera $FFFF) et le code ASCII du dièse ($23). Là encore, on utilise le prête-nom AT (dans le cas où le tableau existe).

ATTENTION: on comprend le role indispensable joué par le dièse dans la déclaration de variables ou de paramètres dans un programme en H-BASIC. L'oubli de ce dièse est catastrophique, car le tableau n'est alors plus empilé ni dimensionné et au contraire un élément du tableau est éventuellement empilé.

- & (#[nom de tableau] dépile un tableau: comme indiqué plus haut, un tableau (local) portant ce nom doit avoir été dimensionné, sous peine de OUT OF DATA ERROR. De plus, ce tableau doit être le dernier à avoir été créé, sous peine de ILLEGAL QUANTITY ERROR. C'est pour cette raison que l'instruction DIM est interdite en H-BASIC: pour éviter des REDIM'D ARRAY ERROR, il est obligatoire de déclarer les tableaux, le compilateur se chargeant de générer les instructions d'empilement et de dimensionnement, puis de dépilement après usage. Si la variable à dépiler n'est pas un tableau (si l'octet du code ASCII du dièse n'est pas sur la pile) ou si le nom du tableau dépilé ne correspond pas (c'est-à-dire si la règle LIFO n'est pas respectée), on obtient un TYPE MISMATCH ERROR.

Notons qu'on peut réunir plusieurs instructions d'empilement ou de dépilement: CALL 32000,) A, #C, B%, #D$ ou CALL 32000, (#D$, B%, #C, A.

CHAPITRE III: COMMENT UTILISER LE H-BASIC?

EXECution d'un fichier TEXT: dans tout ce qui suit, nous appelons ainsi le fait de faire EXEC !nom de fichier!, le contenu du fichier étant alors pris comme des commandes au clavier. En particulier, lorsque le fichier contient des lignes de programme Applesoft, celles-ci seront entrées en mémoire. ATTENTION: si le fichier contient un programme entier, il faut effacer le programme en mémoire (avec un NEW ou un FP) avant de l'EXECuter.

Appelons TOTO le programme à réaliser.

Il faut d'abord créer un fichier TEXT contenant le programme H-BASIC, qu'on appellera par exemple "TOTO.H". Deux méthodes sont possibles: la première consiste à utiliser un système de traitement de texte écrivant dans un fichier TEXT comme APPLE WRITER II ou //e ou bien écrivant dans un fichier binaire comme APPLE WRITER I et en utilisant le programme "APPLEWRITER I TO TEXT" publié dans le numéro 7 de Pom's et fourni sur la disquette H-BASIC.

ATTENTION: une commande étant limitée par le symbole deux-points ou par un retour-chariot, il convient de faire suivre le END final du programme H-BASIC par un de ces délimiteurs, sous peine d'un OUT OF DATA ERROR pendant la compilation. Notons d'ailleurs que tout ce qui suit éventuellement ce délimiteur sera ignoré par le compilateur. De plus, il ne faut pas mettre dans le texte H-BASIC édité des lignes (intervalles entre deux retours-chariot consécutifs) de plus de 255 caratères, taille maximum d'une commande. Il est même conseillé d'utiliser des lignes de moins de 80 caratères afin d'avoir une belle disposition du programme (voir par exemple le compilateur).

La seconde méthode consiste à écrire le programme H-BASIC dans les REMs d'un pseudo-programme Applesoft à partir d'une ligne de numéro supérieur ou égal à 100 et de rajouter les lignes 10 à 90 suivantes. On pourra sauver le pseudo-programme Applesoft sous le nom "FAIT TOTO.H".

ATTENTION: il ne faut pas perdre de vue le fait que la "tokenization" de l'Applesoft entoure les mot-clefs de blancs, et en particulier rajoute un blanc derrière le mot-clef REM.

NOTE: on trouve ces lignes de programme 10 à 90 sur la disquette sous forme du programme Applesoft "PSEUDO-APPLESOFT TO H-BASIC" ou sous forme du fichier TEXT à EXECuter "PSEUDO.EXEC". On trouve aussi sur la disquette le programme inverse "H-BASIC TO PSEUDO-APPLESOFT" qui permet le listage d'un programme H-BASIC pour ceux qui n'ont pas de traitement de texte avec fichiers TEXT: il traduit un fichier H-BASIC en un fichier TEXT dont l'EXECution fournit un pseudo-programme Applesoft; il utilise pour cela une routine d'"INPUT ANYTHING", qui remet par ailleurs les minuscules en majuscules; de plus, puisqu'il rajoute un numéro au début de chaque ligne, le programme H-BASIC ne doit pas en comporter (autrement dit, il ne doit pas être le résultat d'un "PSEUDO-APPLESOFT TO H-BASIC").

```
10  TEXT : HOME : VTAB 10: HTAB 5:
    PRINT "FABRICATION DU FICHIER
    TOTO.H"
20 A$ = "300:A5 67 85 06 A5 68 85
    07 A0 02 B1 06 C9 64 B0 05 C8 B1
    06 F0 06 A0 04 A9 20 91 06 A0 00
    B1 06 AA C8 B1 06 86 06 85 07 D0
    DF 60 N D823G"
30  FOR I = 1 TO LEN (A$): POKE 511
    + I,ASC ( MID$ (A$,I 1)) + 128:
    NEXT : POKE 72,0: CALL -144
40 D$ = CHR$ (13) + CHR$ (4): PRINT
    D$"NOMONCIO"
50 N$="TOTO.H": PRINT D$"OPEN"N$D$
    "DELETE"N$D$"WRITE"N$
60  CALL 768: POKE 33,33: LIST 100:
    PRINT D$"CLOSE"
70  POKE 792,178: CALL 768: TEXT :
    END
80  REM
90  REM
```

Le petit programme en langage machine situé en $300 et appelé par CALL 768 remplace les REMs en début des lignes 100 et suivantes par des blancs. En fait, plus exactement, le blanc remplace le premier octet de la ligne - ici le token du REM. Puis le fichier TEXT "TOTO.H" est créé, qui contiendra les numéros des lignes 100 et suivantes et le contenu des REMs, à l'exclusion du mot REM. Le second CALL 768 rétablit les REMs du pseudo-progamme.

On lance alors le compilateur qui demande le nom du fichier source (ici "TOTO.H") et son numéro de drive, puis ceux du fichier objet qu'il va créer (qu'on pourra appeler "TOTO.B").

ATTENTION: il est conseillé de LOCKer avant la compilation le fichier "TOTO.H", car une malheureuse faute de frappe consistant à écrire "TOTO.H" à la place de "TOTO.B" pourrait le détruire.

La structure par blocs et la création d'un fichier TEXT contenant le programme BASIC permettent de réaliser une compilation en une passe. L'ordre DOS "MON O" permet de voir tout au long de la compilation les lignes qui sont écrites dans le fichier "TOTO.B". Il ne reste alors plus qu'à faire "FP" pour effacer le compilateur, puis "EXEC TOTO.B" et l'on obtiendra le programme Applesoft compilé qu'on pourra sauver sous le nom "TOTO".

NOTE: on peut à tout moment arrêter le compilateur en appuyant sur la touche ESC (il peut s'écouler quelques secondes entre l'enfoncement de la touche et l'arrêt du programme). Il ne faut surtout pas arrêter le programme par CTRL-C ou par RESET: si cela a quand même été fait, il faut taper un CLOSE depuis le clavier (il faut aussi le faire si le compilateur "plante" sur un message d'erreur Applesoft, ce qui en principe ne doit pas arriver). En effet, les fichiers ont été ouverts et il faut les fermer, en particulier le fichier source, dans lequel on écrit. Dans le cas contraire, le catalogue n'affiche qu'une taille de 001 secteurs pour ce fichier, qui est donc perdu, alors que la VTOC (Volume Table Of Contents) de la disquette considère comme occupés les secteurs qui avaient été écrits dans ce fichier, d'où une perte de place sur la disquette. En tout cas, il ne faut absolument pas arrêter le compilateur en éteignant l'Apple...et prions pour qu'il n'y ait pas de panne de courant.

LES MESSAGES DU COMPILATEUR:

Le compilateur s'arrête et émet un message en cas d'erreur au niveau du disque. En principe, il ne peut s'agir que des erreurs suivantes:

- DISK FULL: il n'y a plus de place sur la disquette pour le fichier objet.
- END OF DATA: le fichier source se termine avant le END final ou n'existe pas.
- FILE LOCKED: le fichier objet existe déja et est vérouillé.
- FILE TYPE MISMATCH: l'un des fichiers n'est pas de type TEXT.
- I/O ERROR: disquette abimée, porte du drive ouverte...
- NO BUFFERS AVAILABLE: MAXFILES a été mis à 1 alors que l'on ouvre deux fichiers simultanément.
- SYNTAX ERROR: le nom de l'un des fichiers n'est pas correct.
- WRITE PROTECTED: la disquette où se trouve le fichier objet est protégée contre l'écriture.

De plus, le compilateur émet ses propres messages d'erreur lorsqu'il rencontre une syntaxe H-BASIC incorrecte. Dans ce cas, il attend la frappe d'une touche pour continuer. A noter que la touche ESC arrête la compilation.

ATTENTION: étant donné le caractère profondément récursif du compilateur, et le fait que le H-BASIC ne possède pas de vrai mot-clef (les mots-clefs BEGIN, END et autres, ne sont analysés que quand on les rencontre), certaines erreurs peuvent avoir des causes éloignées de l'effet, c'est-à-dire du message émis, et rendre de plus la suite de la compilation caduque. Par exemple, nous demandons au lecteur de se pencher sur les conséquences de l'oubli d'un ENDIF.

NOTE: comme le H-BASIC est assez lent, le compilateur ne fait pas une analyse syntaxique très poussée du programme H-BASIC, mais n'analyse que ce qui lui est indispensable. Aussi est-il très fortement conseillé à l'utilisateur d'appliquer sur le programme compilé l'utilitaire d'analyse syntaxique "SNTX" publié dans le numéro 6 de Pom's et un utilitaire de documentation pour lister les variables (le "VARDOC" du package APPLE-DOC ou bien le "CONCORDANCE" publié par la revue NIBBLE).

LES PARAMETRES DU COMPILATEUR:

On peut avoir besoin, avant de compiler un programme, de changer la valeur de certaines constantes du compilateur. Ces constantes sont parmi les variables déclarées dans la commande VAR de la procédure COMPILATEUR et on peut les changer facilement dans la version compilée du compilateur, où on les trouve tout à la fin juste sous le REM -- CORPS DE COMPILATEUR.

- HIM représente la valeur de la HIMEM du programme compilé et l'adresse de chargement des routines de pile (voir chapitre V: l'occupation de la mémoire).
- CAL$ représente la commande d'appel des routines de pile, par exemple CALL 32000, ou &.
- MXL est la longueur maximum d'une ligne du programme compilé, LGN le numéro de la première ligne de ce programme et INC l'incrément entre deux lignes successives. INC doit être différent de 1 car le compilateur met un REM dans les lignes, dont le numéro est inférieur de 1 au numéro de la ligne du début d'une procédure.
- NPRO est le nombre maximum de procédures que possède le programme H-BASIC à compiler, NVAR le nombre maximum de variables,

PVAR le nombre maximum de variables par procédure.
- NST est la taille des tableaux de stockage: c'est le nombre maximum de RETURNs par procédure, de POPs par boucle, de ELSIFs par IF, de WHENs par CASE, d'appels de procédures entre un DO et un ONERR.

LES TABLEAUX =ZZ$, =ZZ, YY$ ET YY:

Ces 4 tableaux appartiennent à tout programme compilé, par le biais de la ligne écrite dans la procédure COMPILATEUR, qui les DIMensionne et permet aussi le chargement des routines de pile, la mise en place de la HIMEM et l'appel de la procédure principale du programme compilé. Ces noms de variables ne peuvent donc être utilisés en H-BASIC.
- =ZZ$ et =ZZ permettent de stocker pour la durée de l'empilement ou du dépilement les paramètres passés ou sortis d'une procédure et comportant un point d'exclamation, ou les variables d'une procédure possédant un point d'exclamation après le signe '=' d'affectation (voir le chapitre I: la syntaxe du H-BASIC). La dimension de ces tableaux indique le nombre maximum de tels paramètres autorisés par appel de procédure ou par commande VAR. Ces paramètres sont comptés par les variables ACH et ANU.
- =YY$ et =YY permettent de stocker les expressions suivant une commande CASE$ ou un CASE afin de les incorporer dans le IF qui traduit une commande WHEN. Leur dimension indique le nombre maximum d'instructions CASE$ ou CASE que peut comporter le programme à compiler. Ces instructions sont numérotées par les variables CCH et CNU.

CHAPITRE IV: COMMENT FONCTIONNE LE COMPILATEUR?

Ce chapitre explique sommairement le fonctionnement du compilateur. Pour en savoir plus, le lecteur pourra examiner le compilateur, le propre des programmes en H-BASIC étant d'être très facilement lisibles par un autre que leur auteur. De plus, chaque procédure est précédée d'une remarque résumant son role.

LES PROCEDURES DU COMPILATEUR:

Le compilateur commence par charger les routines d'input et par mettre en place le vecteur de l'ampersand (lignes précédant la première procédure). Puis il entre dans la procédure COMPILATEUR où il commence par appeler la procédure INITIALISATION qui demande les noms et drives des programmes source et objet; alors, il traduit telles quelles toutes les commandes précédant la première commande "PROCEDURE", puis écrit la première ligne du programme compilé et rentre dans la procédure PROCEDURE. Là, il analyse la déclaration de paramètres et de variables (procédure DECVAR) et retombe récursivement dans PROCEDURE tant qu'il rencontre la commande "PROCEDURE". Il entre alors dans la procédure BLOC, dans laquelle il attend "BEGIN", puis empile et initialise les variables (procédure EMPILE) et, avant de les dépiler (procédure DEPILE), entre dans SUITINST où il traduit toutes les instructions jusqu'au mot "END" grace à la procédure INSTRUCTION. Là, les instructions sont analysées et éventuellement traduites dans les procédures APPEL, IF, CASE, DO, RETURN et BOUCLE.

LES VARIABLES DU COMPILATEUR:

Les variables contenant des constantes utiles pouvant être modifiées ont été décrites dans le chapitre III (comment utiliser le H-BASIC?). Ce qui suit décrit les autres variables importantes.

- PROCEDURE COMPILATEUR:

  - SRC$ et OBJ$ sont les noms des fichiers source et objet.
  - WR=0 si on lit le source et WR=1 si on écrit dans l'objet.
  - LGN$ est la ligne courante du programme compilé.
  - CD$ est la commande courante du programme H-BASIC, recopiée dans CMD$ pour y être décortiquée.
  - PRO et VAR sont les numéros des variables et procédures rencontrées.
  - #NOM$ et #VAR$ contiennent les noms des procédures et des variables.
  - #VAR% contient les pointeurs dans #NOM$ des zones de paramètres et variables des différentes procédures.
  - #ADR% contient les numéros des lignes d'appel des procédures.

- PROCEDURE PROCEDURE:

  - #NOM% contient les numéros pour #NOM$ des procédures emboitées dans la procédure courante, procédures comptées par NOM.
  - NUM et OLD sont l'indice dans #NOM$ et l'adresse d'appel de la procédure courante.
  - TYPE est le type de paramètre ou variable (INOUT=1, IN=2, OUT=3, VAR=4).

- PROCEDURE BLOC:

  - ERR=1 entre un DO et un ONERR.
  - #DO$ stocke les "ONERR GOTO.." générés entre un DO et un ONERR, comptés par DO.
  - REA stocke l'adresse de début de la procédure pour préparer le RESTORE NNN d'un premier READ éventuel et vaut 0 après ce READ.
  - #RT stocke les "GOTO.." ou "IF..GOTO.." générés par les RETURNs de la procédure, comptés par RT.

- PROCEDURE INSTRUCTION:

  - #ST$ stocke des "IF..GOTO" ou "GOTO" associés à l'instruction courante (spécialement IF ou CASE), comptés par ST.
  - ST$ stocke un "IF..GOTO" ou "GOTO.."

- PROCEDURE APPEL:

  - #V$ contient les noms des paramètres passés à ou sortis de la procédure appelée, comptés par V.
  - PIL$ contient les empilements avant appel et les dépilements après appel.
  - AFF$ contient les affectations avant et après appel et AUX$ contient les affectations auxiliaires de stockage quand le paramètre passé ou sorti est précédé d'un point d'exclamation. Dans ce cas, ACH et ANU sont les indices des variables de stockage alphanumériques et numériques, variables qui sont les éléments des tableaux ZZ$ et ZZ.

- PROCEDURE BOUCLE:

  - #SP$ stocke les "GOTO.." ou "IF..GOTO.." générés par les POPs de la boucle, comptés par SP.
  - GT$ stocke un "IF..GOTO.." dû à WHILE ou à UNTIL.
  - BCL=1 à l'intérieur d'une boucle.

CHAPITRE V: L'OCCUPATION DE LA MEMOIRE

L'écriture d'un programme en H-BASIC revenant à écrire un fichier TEXT, nous supposons cela acquis et ne parlons ici que de l'exécution d'un programme compilé.

Voici l'occupation de la mémoire standard pour le compilateur:

$0000-$02FF: système
$0300-$03FF: l'utilisateur peut mettre ici des routines en langage machine
$0400-$07FF: page TEXT
$0800-$7CFF: programme, variables, pages graphiques haute résolution, le tout étant à la disposition du programmeur comme pour un programme Applesoft classique. La HIMEM est mise en $7D00 (décimal 32000).
$7D00-$7FFF: routines de pile
$8000-$80FF: routines d'input
$8100-$81FF: buffer de l'input
$8200-$8FFF: pile H-BASIC
$9000-$95FF: libre (pour PLE par exemple)
$9600-$FFFF: DOS, entrées-sorties et ROM ou carte langage.

Pour un autre programme H-BASIC, la seule différence est que la pile fait $8000-$8FFF, car on n'a pas besoin des routines d'input.

CHANGEMENT DE CES VALEURS:

Le programmeur peut avoir envie de changer la place mémoire de la pile et des routines de pile destinées à ses programmes H-BASIC (ce n'est pas la peine de les changer pour le compilateur, qui a été prévu pour fonctionner avec les valeurs par défaut), soit pour gagner de la place s'il n'utilise pas PLE, soit pour en libérer s'il veut utiliser un amper-interpréteur comme HAIFA, APA ou CRAE.

Si l'on dispose de LISA 2.5, il suffit de changer les valeurs de début et de fin de pile dans le fichier assembleur (BASPILE=$80, HAUTPILE=$90) en y mettant les valeur désirées, puis de changer la valeur de l'ORG du programme à la valeur désirée (ne pas toucher à l'OBJ) et d'assembler le tout.

Si l'on ne dispose pas de LISA 2.5, il suffit de BLOADer le fichier binaire PILE, d'y changer les valeurs de début et de fin de pile là où on les emploie (CPX #BASPILE -) E0 80 et CPX #HAUTPILE -) E0 90), puis enfin, avant de le BSAVEr, de reloger le programme à l'adresse désirée, soit "à la main" en changeant les JMP et les LDX, DEC ou INC concernant les adresses PILE1+1, PILE1+2, PILE2+1 et PILE2+2, soit comme indiqué par Jean-François Duvivier dans le numéro 1 de Pom's.

Enfin, il faut changer dans le programme COMPILATEUR (version compilée) la valeur par défaut de la variable HIM, qui représente la HIMEM et l'adresse de chargement des routines de pile pour le programme compilé. Si l'on veut dissocier la HIMEM de cette adresse de chargement, on peut le faire en mettant dans HIM l'adresse de chargement et en remplaçant le deuxième STR$(HIM) par STR$([valeur de la HIMEM]) dans l'instruction PR$ = STR$ (LGN) + "PRINT CHR$ (4)" + Q$ + "BLOAD PILE,A" + STR$ (HIM) + Q$ + ":HIMEM:" + STR$(HIM) + ":DIM ZZ$(19), ZZ(19), YY(9), YY$(9):" + CAL$ + "GOSUB" + STR$ (LGN+INC) + ":END" à la fin du compilateur.

CHAPITRE VI: AVANTAGES ET INCONVENIENTS DU H-BASIC

AVANTAGES PAR RAPPORT A L'APPLESOFT:

Le H-BASIC est beaucoup mieux structuré, possède de vrais sous-programmes et des variables locales. Il est donc récursif et de plus les programmes sont très faciles à écrire et très faciles à "éplucher" par un autre que par leur auteur. En outre, le H-BASIC possède une instruction puissante avec l'affectation de tableaux.

INCONVENIENTS PAR RAPPORT A L'APPLESOFT:

Le principal inconvénient du H-BASIC en est la compilation, assez longue et un peu lourde. De plus, les programmes H-BASIC compilés en BASIC sont plus encombrants en mémoire et un peu plus lents que des programmes Applesoft équivalents.

AVANTAGES PAR RAPPORT AU PASCAL:

Tout d'abord, le H-BASIC est beaucoup moins cher. Il est moins complexe et plus facile à débugger. En particulier, on peut faire une légère modification au programme directement sur le programme objet sans devoir recompiler le source H-BASIC. Faire du langage machine est beaucoup moins lourd (en Pascal, il faut faire une édition de liens; en H-BASIC, il suffit d'un CALL ou d'un ampersand). De même, le H-BASIC utilise le DOS, qui est beaucoup moins lourd que le Pascal Operating System. Enfin, le H-BASIC possède l'instruction DATA et le graphisme basse résolution.

INCONVENIENTS PAR RAPPORT AU PASCAL:

Le principal inconvénient du H-BASIC en est bien sur la lenteur, surtout dans les programmes récursifs où le Pascal excelle (car le Pascal est compilé en P-code, langage interprété très proche du langage machine). Enfin, le H-BASIC est pauvre en types de données, en noms de variables (2 lettres significatives au lieu de 8) et ne possède pas de fonctions au vrai sens du terme, ni de tortue haute-résolution.

Notons qu'on peut pallier beaucoup d'inconvénients du H-BASIC en utilisant des amper-interpréteurs tels que HAIFA (voir Pom's numéro 5).

CHAPITRE VII: PROGRAMMES H-BASIC JOINTS

Nous avons mis sur la disquette H-BASIC quelques programmes de démonstration. On trouve la version H-BASIC ("FAIT TOTO.H" ou bien "TOTO.H") et la version compilée en Applesoft ("TOTO").

LE COMPTE EST BON:

Ce programme se propose de résoudre le célèbre jeu télévisé et il y arrive dans 95% des cas. Il s'agit là d'un programme écrit primitivement en Pascal, la transposition en H-BASIC n'ayant guère posé de problèmes. Pour en savoir plus, on peut se reporter au numéro de Mars 1983 de "L'Ordinateur Individuel" qui publie et explique le programme.

C'est une bonne illustration de l'intérêt du H-BASIC, car ce programme aurait été beaucoup plus difficile à écrire en Applesoft, étant donné le caractère récursif du problème.

D'un autre coté, ce n'est pas une très bonne illustration, car c'est un programme de calculs récursifs, domaine où le Pascal excelle. Comparons les temps de compilation et d'exécution.

- Compilation Pascal: 1mn20
- Compilation H-BASIC: 5mn30
- EXECution du fichier COMPTE.B: 1mn30
- Temps de résolution en Pascal: (10s dans 90% des cas.
- Temps de résolution en H-BASIC: (150s dans 90% des cas, avec pointes de fréquence à 30s et 90s.

HILBERT:

Ce programme reprend un des programmes de démonstration fournis avec le Pascal Apple: il permet de matérialiser à des ordres divers le dessin d'un fractal, la courbe de Hilbert.

Là encore, le caractère récursif du problème privilégie le H-BASIC par rapport à l'Applesoft.

VON KOCH:

Alors que la courbe de Hilbert est un fractal de dimension 2, car elle 'tend à remplir' le plan lorsque l'ordre tend vers l'infini, la courbe de Von Koch, elle, est de dimension comprise entre 1 et 2.

Le lecteur pourra expérimenter à sa guise le tracé de courbes fractales, ce que le H-BASIC rend extrêmement facile. Pour

l'orienter dans cette direction, il y a d'excellents ouvrages sur les fractals, dont celui de Benoit Mandelbrot, le "père" des fractals.

POLYNOMES:

Ce programme est issu d'un programme original écrit en Pascal par Denis Attal. Il simule le comportement d'une calculatrice polynomiale très simple. Il y a 26 polynomes appelés 'A' à 'Z', initialisés à 0 sauf 'X' qui est bien sur le polynome formel utilisé dans la représentation d'un polynome (1 + X + 3*X^2 par exemple). Les opérations autorisées sont, par ordre de priorité croissante, l'addition, la soustraction, la multiplication, l'élévation à une puissance entière, la composition et la dérivation de polynomes.

On peut rentrer un nom de polynome, et sa valeur sera affichée; ou bien on fait suivre ce nombre par le signe "=" et une expression, qui sera calculée. On a ainsi le dialogue suivant:

```
?A
 VAUT 0
?A=X+1+X^2
A VAUT 1 + X + X^2
?B=A+A
B VAUT 2 + 2*X + 2*X^2
?C=X+1
C VAUT 1 + X
?D=C(B)-B
D VAUT 1
?E=3+B'
E VAUT 5 + 4*X
?E=E*E+D
E VAUT 26 + 40*X + 16*X^2
```

NOTES: taper "/" pour terminer. Ne pas changer le polynome X. Si "A=-1" est autorisé, "B=-A" ou "C=A(-1)" sont interdits, à cause de la présence du signe "-". De plus, il faut respecter scrupuleusement la syntaxe du programme. En effet, il ne sert que de démonstration et ne contient pas de tests de vraisemblance de la ligne entrée par l'utilisateur; ainsi, si cette ligne est incorrecte, elle peut provoquer une erreur du genre ILLEGAL QUANTITY ERROR. Mais le lecteur pourra facilement façonner le programme à son gout pour le rendre ergonomique et tout à fait opérationnel.

CHAPITRE VIII: QUE CONTIENT LE PACKAGE H-BASIC?

NOTATIONS: tous les programmes en H-BASIC sont des fichiers TEXT terminés par le suffixe ".H". On peut aussi les trouver dans les REMs d'un pseudo programme BASIC dont le nom commence par "FAIT". Les fichiers TEXT contenant les programmes compilés se terminent par le suffixe ".B" et les programmes Applesoft obtenus par l'EXECution des précédents n'ont pas de suffixe. Les routines écrites en assembleur (LISA 2.5) se terminent par le suffixe ".LISA" et leurs programmes binaires objets n'ont pas de suffixe.

On trouve tout d'abord bien sur le programme "COMPILATEUR", ainsi que les routines "INPUT" et "PILE". Notons que le compilateur a ses propres routines de pile: "PILE.COMPILATEUR".

On trouve aussi les programmes de démonstration écrits en H-BASIC et compilés en Applesoft, comme le jeu "LE COMPTE EST BON" (programme "COMPTE") ou les programmes de dessins de fractals "HILBERT" et "VON KOCH".

On trouve enfin quelques utilitaires: le programme "PSEUDO-APPLESOFT TO H-BASIC" qui sert aux pseudo-programmes du genre "FAIT TOTO.H", sa version EXEC "PSEUDO.EXEC", son inverse "H-BASIC TO PSEUDO-APPLESOFT" et le programme "APPLEWRITER I TO TEXT".

CATALOGUE DE LA DISQUETTE:

```
*A 004 APPLEWRITER I TO TEXT
*A 048 COMPILATEUR
*T 056 COMPILATEUR.H
*A 018 COMPTE
*T 022 COMPTE.H
*A 024 FAIT COMPTE.H
*A 009 FAIT HILBERT.H
*A 008 FAIT VON KOCH.H
*A 003 H-BASIC
*A 005 H-BASIC TO PSEUDO-APPLESOFT
*A 006 HILBERT
*T 007 HILBERT.H
*B 002 INPUT
*B 014 INPUT.LISA
*B 006 MASK 1
*B 006 MASK 2
*B 004 PILE
*B 004 PILE.COMPILATEUR
*B 033 PILE.COMPILATEUR.LISA
*B 032 PILE.LISA
*A 017 POLYNOMES
*T 017 POLYNOMES.H
*A 004 PSEUDO-APPLESOFT TO H-BASIC
*T 004 PSEUDO.EXEC
*B 033 TITRE
*A 005 VON KOCH
*T 006 VON KOCH.H
```

```
-- ****************************
-- *                          *
-- * COMPILATEUR DE H-BASIC   *
-- *                          *
-- * (C) O. HERZ POUR POM'S   *
-- *                          *
-- ****************************

- on charge les routines d'input et on place le vecteur de l'amperaand

if PEEK(32768)<>32 then print CHR$(4)"BLOAD INPUT,A$8000": x
oke 1013,76: poke 1014,0: poke 1015,128

-- cette procedure est le programme lui-meme

rocedure COMPILATEUR
var  D$=CHR$(4),  Q$=CHR$(34), SRC$, OBJ$, WR=1, HIM=32000, CAL$="CALL"+STR$(HIM
)+",", MXL=200, LGN=100, INC=10, LGN$="", CMD$, CD$,
CH=-1, CNU=-1, PRO=-1, VAR=-1, NPRO=50, NVAR=200, PVAR=50, NST=20, #NOM$(NPRO),
#VAR$(NVAR), #VAR%(NPRO,4), #ADR%(NPRO)

- sortie du programme

procedure SORTIE
egin: normal: print: print D$"CLOSE": atop: end

-- sort du programme en cas d'erreur disque

rocedure DISKERR
egin
 print: print D$"CLOSE"
 inverse: print " ERREUR DISQUE: ";
 case PEEK(222)
   when 9:  print "DISK FULL "
   when 5:  print "END OF DATA "
   when 10: print "FILE LOCKED "
   when 6:  print "FILE NOT FOUND "
   when 13: print "FILE TYPE MISMATCH "
   when 8:  print "I/O ERROR "
   when 12: print "NO BUFFERS AVAILABLE "
   when 11: print "SYNTAX ERROR "
   when 4:  print "WRITE PROTECTED "
   whenothers: print "ERREUR ANORMALE - NUMERO ";PEEK(222);" "
 endcase: @ SORTIE
end

- ecrit une ligne dans le fichier objet

rocedure PRINT: in PR$
egin
 do
   if not WR: print: print D$"WRITE"OBJ$: WR=1
   endif: print PR$
 onerr @ DISKERR
 enddo
nd
```

```
-- termine la ligne courante du programme compile

procedure FINLGN
begin
  return when LGN$=""
  @ PRINT(STR$(LGN)+LGN$): LGN=LGN+INC: LGN$=""
end

-- lit une commande dans le fichier source et elimine les remarques

procedure LITCMD
var KEY
begin
  do
    until LEFT$(CD$,2)<>"--"
      if WR: print D$"READ"SRC$: WR=0
      endif: & INPUT;CD$
      if LEFT$(CD$,1)="%": @ FINLGN
        ]if CD$="%" then CD$="--": %
      endif
    enduntil: CMD$=CD$
    KEY=PEEK(-16384): poke-16368,0
    if KEY=155: @ SORTIE: endif
  onerr @ DISKERR
  enddo
end

-- prend les noms et drives des fichiers et les ouvre

procedure INITIALISATION
var C1$, C2$
begin
  text: home: vtab 5: htab 9
  print "COMPILATEUR DE H-BASIC"
  print: print
  htab 6: print "(C)  OLIVIER HERZ POUR POM'S"
  print: print: print
  input "NOM DU PROGRAMME SOURCE: ";SRC$
  print "NUMERO DE DRIVE: ";
  until C1$="1" or C1$="2": GET C1$: enduntil
  print C1$: print
  input "NOM DU PROGRAMME OBJET : ";OBJ$
  print "NUMERO DE DRIVE: ";
  until C2$="1" or C2$="2": GET C2$: enduntil
  print C2$: print
  do print D$"NOMONCI": print D$"MONO"
    print D$"OPEN"SRC$",D"C1$
    print D$"OPEN"OBJ$",D"C2$
    print D$"DELETE"OBJ$: print D$"OPEN"OBJ$
  onerr @ DISKERR
  enddo
end

-- affiche un message d'erreur du H-BASIC

procedure ERREUR: in ER$
var KEY
begin
  @ PRINT(""): inverse: print " ";ER$;" ";chr$(7);chr$(7)
  print " INSTRUCTION: ";CD$;" "
  ]if NOM$(NUM)<>"" then print " PROCEDURE: ";NOM$(NUM);" ": %
   normal: poke-16368,0: wait-16384,128
   KEY=PEEK(-16384): poke-16368,0
   if KEY=155: @ SORTIE: endif
end
```

```
-- prend le corps de la commande apres un mot-clef

procedure CORPSCMD: in CO$
begin
  CMD$=MID$(CMD$,LEN(CO$)+1)
  if CMD$=""
    @ ERREUR("SUITE DE '"+CO$+"' ATTENDUE")
  endif
end

-- attend un mot clef au debut de la commande

procedure ATTEND: in AN$
begin
  if LEFT$(CMD$,LEN(AN$))<>AN$
    @ ERREUR("'"+AN$+"' ATTENDU")
  endif
end

-- lit,soit un nom de variable dans une declaration, soit un parametre passe

procedure LITVAR: out LI$
var LI
begin
  & INPUT,CMD$: LI=PEEK(255)
  if LI=1: @ ERREUR("VARIABLE OU PARAMETRE ATTENDU")
  else LI$=LEFT$(CMD$,LI-1): CMD$=MID$(CMD$,LI+1)
  endif
end

-- ajoute une instruction a la ligne courante

procedure ECRIT: in EC$
begin
  return when EC$=""
  if LEN(LGN$)+LEN(EC$)>MXL: @ FINLGN: endif
  if LGN$="": LGN$=EC$: else LGN$=LGN$+":"+EC$
  endif
end

-- stocke la ligne courante en attente d'un numero de ligne pour se brancher

procedure STOCKE: in A$: out B$
begin
  if LEN(LGN$)+LEN(A$)>MXL: @ FINLGN: endif
  if LGN$="": LGN$=A$: else LGN$=LGN$+":"+A$
  endif
  B$=STR$(LGN)+LGN$: LGN=LGN+INC: LGN$=""
end

-- stocke la ligne courante dans un tableau

procedure STOTAB: inout #T$(NST), T: in T$
begin: T=T+1: @ STOCKE(T$,T$(T)): end

-- ecrit la ligne stockee, connaissant la ligne ou se brancher

procedure FINSTOCK: in T$
begin: @ FINLGN: @ PRINT(T$+STR$(LGN)): end
```

```
-- ecrit le tableau stocke

procedure FINSTOTAB: in #T$(NST), T
var I
begin
  return when T=-1
  for I=0 to T: @ FINSTOCK(T$(I)): endfor
end

-- ecrit l'instruction d'affectation d'une variable passee

procedure AFFECT: in VV
var C$, V1$, V2$
begin
  ]if AFF$<>"" then AFF$=AFF$+":": %
  if LEFT$(EX$,1)="!": EX$=MID$(EX$,2)
    ]if AUX$<>"" then AUX$=AUX$+":": %
    if RIGHT$(VR$,1)="$"
      ACH=ACH+1: C$="ZZ$("+STR$(ACH)+")"
    else ANU=ANU+1: C$="ZZ("+STR$(ANU)+")"
    endif
    if VV: AUX$=AUX$+C$+"="+EX$: AFF$=AFF$+VR$+"="+C$
    else AFF$=AFF$+C$+"="+VR$: AUX$=AUX$+EX$+"="+C$
    endif
  else
    if VV: V1$=VR$: V2$=EX$: else V1$=EX$: V2$=VR$: endif
    if LEFT$(VR$,1)="#": AFF$=AFF$+CAL$+V1$+"="+V2$
    else AFF$=AFF$+V1$+"="+V2$
    endif
  endif
end

-- ecrit l'instruction d'empilement ou de depilement d'une variable

procedure PILE
var J
begin
  if LEFT$(VR$,1)="#": & INPUT(VR$: J=PEEK(255)
    ]if PIL$<>"" then PIL$=PIL$+",": %
    VR$=LEFT$(VR$,J-1): PIL$=PIL$+VR$
  elsif LEFT$(VR$,1)="!": VR$=MID$(VR$,2)
    ]if PIL$<>"" then PIL$=PIL$+",": %
    PIL$=PIL$+VR$
  endif
end

-- empile les variables a empiler et affecte les variables passees

procedure EMPILE: in TYPE, N
var VR, VR$, EX$, J
begin
  return when VAR%(N,TYPE-1)=VAR%(N,TYPE)
  for VR=VAR%(N,TYPE-1)+1 to VAR%(N,TYPE): VR$=VAR$(VR)
    if TYPE=4: & INPUT=VR$: J=PEEK(255)
      EX$=MID$(VR$,J+1): VR$=LEFT$(VR$,J-1)
    else @ LITVAR(EX$)
      if TYPE<>2: V=V+1: V$(V)=EX$: endif
      if TYPE=3: EX$="": endif
    endif
    if LEFT$(VR$,1)="#"
      ]if AFF$<>"" then AFF$=AFF$+":": %
      AFF$=AFF$+"DIM"+MID$(VR$,2)
    endif
    @ PILE: if EX$<>"": @ AFFECT(1): endif
  endfor
end
```

```
-- depile les variables a depiler et affecte les variables passees

procedure DEPILE: in TYPE, N
var VR, VR$, EX$, J
begin
  return when VAR%(N,TYPE-1)=VAR%(N,TYPE)
  for VR=VAR%(N,TYPE) to VAR%(N,TYPE-1)+1 step -1
    VR$=VAR$(VR)
    if TYPE=4: & INPUT=VR$: J=PEEK(255)
      VR$=LEFT$(VR$,J-1)
    endif: @ PILE
    if TYPE=1 or TYPE=3
      EX$=V$(V): V=V-1: @ AFFECT(0)
    endif
  endfor
end

-- analyse et traduit une suite d'instructions entre deux mots-clefs

procedure SUITEINST: in !DEB$, !FIN$

-- analyse et traduit une instruction H-BASIC

procedure INSTRUCTION
var #ST$(NST), !ST=-1, !ST$

-- instruction d'appel de procedure

procedure APPEL
var K, L, NB=-1, #V$(PVAR),  V=-1, BN=0, PIL$, AUX$, AFF$, ACH, ANU
begin
  if ERR: @ ECRIT("POKE 216,0"): endif
  @ CORPSCMD("@"): & INPUT(CMD$: L=PEEK(255)
  if L=1: @ ERREUR("NOM DE PROCEDURE ATTENDU")
  else
    while NB<=PRO and not BN: NB=NB+1
      BN=LEFT$(CMD$,L-1)=NOM$(NB)
    endwhile
  endif
  if not BN
    @ ERREUR("PROCEDURE NON TROUVEE"): return
  endif: CMD$=MID$(CMD$,L+1)
  if CMD$=""
    @ ECRIT(CAL$+"GOSUB"+STR$(ADR%(NB)))
  else
    if RIGHT$(CMD$,1)<>")": @ ERREUR("')' ATTENDUE")
    elsif CMD$=")": @ ERREUR("PARAMETRES ATTENDUS")
    else CMD$=LEFT$(CMD$,LEN(CMD$)-1)
    endif
    PIL$="": AUX$="": AFF$="": ACH=-1: ANU=-1
    for K=1 to 3: @ EMPILE(K,NB): endfor
    @ ECRIT(AUX$)
    if PIL$<>"": @ ECRIT(CAL$+">"+PIL$):endif
    @ ECRIT(AFF$): @ ECRIT(CAL$+"GOSUB"+STR$(ADR%(NB)))
    PIL$="": AUX$="": AFF$="": ACH=-1: ANU=-1
    for K=3 to 1 step-1: @ DEPILE(K,NB): endfor
    @ ECRIT(AFF$)
    if PIL$<>"": @ ECRIT(CAL$+"<"+PIL$): endif
    @ ECRIT(AUX$)
  endif
  if ERR: @ STOTAB(#DO$,DO,"ONERR GOTO"): endif
end
```

```
-- instruction IF..ELSIF..ELSE..ENDIF

procedure IF

-- stocke le IF et regarde s'il y a un THEN facultatif

procedure THEN
begin
  @ STOCKE("IF NOT("+CMD$+")THEN",ST$): @ LITCMD
  return when LEFT$(CMD$,4)<>"THEN"
  CMD$=MID$(CMD$,5): if CMD$="": @ LITCMD: endif
end

begin -- IF
  @ CORPSCMD("IF"): @ THEN
  while CMD$<>"ENDIF" and LEFT$(CMD$,4)<>"ELSE"
    if LEFT$(CMD$,5)="ELSIF"
      @ STOTAB(#ST$,ST,"GOTO"): @ FINSTOCK(ST$)
      @ CORPSCMD("ELSIF"): @ THEN
    else @ INSTRUCTION
    endif
  endwhile
  if LEFT$(CMD$,4)="ELSE"
    @ STOTAB(#ST$,ST,"GOTO"): @ FINSTOCK(ST$)
    @ SUITEINST("ELSE","ENDIF")
  else @ FINSTOCK(ST$)
  endif: @ FINSTOTAB(#ST$,ST)
end

-- instruction CASE..WHEN..WHEN..WHENOTHERS..ENDCASE

procedure CASE
var !CS$

-- traite la commande WHEN

procedure WHENCASE
var W$
begin
  @ CORPSCMD("WHEN"): @ LITVAR(W$)
  ST$="IF"+CS$+"<>"+W$
  while CMD$<>""
    @ LITVAR(W$): ST$=ST$+"AND"+CS$+"<>"+W$
  endwhile
  ST$=ST$+"THEN": @ STOCKE(ST$,ST$): @ LITCMD
end

begin -- CASE
  @ CORPSCMD("CASE")
  if LEFT$(CMD$,1)="$": CCH=CCH+1
    CS$="YY$("+STR$(CCH)+")"
    @ CORPSCMD("$")
  else CNU=CNU+1: CS$="YY("+STR$(CNU)+")"
  endif
  @ ECRIT(CS$+"="+CMD$): @ LITCMD
  @ ATTEND("WHEN"): @ WHENCASE
  while CMD$<>"ENDCASE" and CMD$<>"WHENOTHERS"
    if LEFT$(CMD$,4)="WHEN"
      @ STOTAB(#ST$,ST,"GOTO"): @ FINSTOCK(ST$)
      @ WHENCASE: else @ INSTRUCTION
    endif
  endwhile
```

```
  if LEFT$(CMD$,10)="WHENOTHERS"
    @ STOTAB(#ST$,ST,"GOTO"): @ FINSTOCK(ST$)
    @ SUITEINST("WHENOTHERS","ENDCASE")
  else @ FINSTOCK(ST$)
  endif: @ FINSTOTAB(#ST$,ST)
end

-- instruction DO

procedure DO
begin
  if ERR: @ ERREUR("'DO' EMBOITES"): endif
  @ STOTAB(#DO$,DO,"ONERR GOTO")
  ERR=1: @ SUITEINST("DO","ONERR")
  @ STOCKE("GOTO",ST$): @ FINSTOTAB(#DO$,DO)
  @ ECRIT("POKE 216,0"): ERR=0
  @ SUITEINST("ONERR","ENDDO"):@ FINSTOCK(ST$)
end

-- teste s'il y a un WHEN avec POP, RETURN ou STOP

procedure WHEN: in W$
begin
  CMD$=MID$(CMD$,LEN(W$)+1)
  return when CMD$="": @ ATTEND("WHEN")
  @ CORPSCMD("WHEN"): CMD$="IF"+CMD$+"THEN"
  ]if ERR then CMD$=CMD$+"POKE 216,0:": %
end

-- instruction RETURN

procedure RETURN
begin
  @ WHEN("RETURN"): @ STOTAB(#RT$,RT,CMD$+"GOTO")
end

-- instruction POP

procedure POP
begin
  if NOT BCL: @ ERREUR("'POP' ILLEGAL")
  else @ WHEN("POP"): @ STOTAB(#SP$,SP,CMD$+"GOTO")
  endif
end

-- les bouclea FOR, WHILE et UNTIL

procedure BOUCLE: in TYPE$
var #SP$(NST), !SP=-1, !GT$, !BCL=1
begin
  @ CORPSCMD(TYPE$)
  case$ TYPE$
  when "FOR"
    @ ECRIT("FOR"+CMD$): & INPUT=CMD$: I=PEEK(255)
    if I=1: @ ERREUR("VARIABLE ATTENDUE APRES 'FOR'")
    else GT$=LEFT$(CMD$,I-1)
    endif
    @ SUITEINST("","ENDFOR"): @ ECRIT("NEXT"+GT$)
  when "WHILE"
    @ FINLGN: GT$="GOTO"+STR$(LGN)
    @ STOCKE("IF NOT("+CMD$+")THEN",ST$)
    @ SUITEINST("","ENDWHILE")
    @ ECRIT(GT$): @ FINSTOCK(ST$)
```

```
  when "UNTIL"
    @ FINLGN: GT$="IF NOT("+CMD$+")THEN"+STR$(LGN)
    @ SUITEINST("","ENDUNTIL")
    @ ECRIT(GT$): @ FINLGN:
  endcase: @ FINSTOTAB(#SP$,SP)
end

begin -- INSTRUCTION
  if LEFT$(CMD$,1)="@": @ APPEL
  elsif LEFT$(CMD$,2)="IF":@ IF
  elsif LEFT$(CMD$,4)="CASE": @ CASE
  elsif LEFT$(CMD$,2)="DO": @ DO
  elsif LEFT$(CMD$,6)="RETURN": @ RETURN
  elsif LEFT$(CMD$,3)="POP": @ POP
  elsif LEFT$(CMD$,3)="FOR": @ BOUCLE("FOR")
  elsif LEFT$(CMD$,5)="WHILE": @ BOUCLE("WHILE")
  elsif LEFT$(CMD$,5)="UNTIL": @ BOUCLE("UNTIL")
  elsif LEFT$(CMD$,1)="]": @ CORPSCMD("]"): @ ECRIT(CMD$)
  elsif LEFT$(CMD$,1)="#": @ ECRIT(CAL$+CMD$)
  elsif LEFT$(CMD$,4)="READ":
       if REA
         @ ECRIT(CAL$+"@"+STR$(REA)): REA=0
       endif: @ ECRIT(CMD$)
  elsif LEFT$(CMD$,4)="STOP": @ WHEN("STOP")
    @ ECRIT(CMD$+"END"): @ FINLGN
  elsif LEFT$(CMD$,6)="LOMEM="
    @ CORPSCMD("LOMEM="): @ ECRIT("LOMEM:"+CMD$)
  elsif LEFT$(CMD$,6)="HIMEM="
    @ CORPSCMD("HIMEM="): @ ECRIT("HIMEM:"+CMD$)
  elsif LEFT$(CMD$,3)="DIM": @ ERREUR("'DIM' EST INTERDIT")
  elsif LEFT$(CMD$,3)="REM": @ FINLGN: @ PRINT(CMD$)
  else @ ECRIT(CMD$)
  endif: @ LITCMD
end

begin -- SUITEINST
  if DEB$="": @ LITCMD
  else CMD$=MID$(CMD$,LEN(DEB$)+1)
    if CMD$="": @ LITCMD: endif
  endif
  while LEFT$(CMD$,LEN(FIN$))<>FIN$: @ INSTRUCTION
  endwhile
end

-- analyse et traduit une procedure

procedure PROCEDURE
var #NOM%(NPRO), !NOM=-1, !NUM=PRO+1, !OLD, TYPE=0, N, R1$, R2$

-- analyse et traduit la declaration des parametres et des variables

procedure DECVAR: in TYPE$
var V=LEN(TYPE$), V$, I, J
begin
  TYPE=TYPE+1
  if LEFT$(CMD$,V)=TYPE$: CMD$=MID$(CMD$,V+1)
    while CMD$<>"": @ LITVAR(V$)
      VAR=VAR+1: VAR$(VAR)=V$
    endwhile: @ LITCMD
  endif: VAR%(NUM,TYPE)=VAR
end
```

```
-- empile et depile les variables locales et traduit un bloc BEGIN..END

procedure BLOC
var ERR=0, #DO$(NST), DO=-1, BCL=0, REA=LGN, #RT$(NST), RT=-1, PIL$="", AFF$=""
begin
  @ ATTEND("BEGIN"): @ EMPILE(4,NUM)
  if PIL$<>"": @ ECRIT(CAL$+">"+PIL$): endif
  @ ECRIT(AFF$): @ SUITEINST("","END")
  @ FINSTOTAB(#RT$,RT): PIL$="": @ DEPILE(4,NUM)
  if PIL$<>"": @ ECRIT(CAL$+"<"+PIL$): endif
  @ ECRIT(CAL$+"RETURN"): @ FINLGN
end

begin -- PROCEDURE
  @ CORPSCMD("PROCEDURE"): NOM$(NUM)=CMD$
  @ LITCMD: PRO=PRO+1: VAR%(NUM,TYPE)=VAR
  @ DECVAR("INOUT"): @ DECVAR("IN")
  @ DECVAR("OUT"): @ DECVAR("VAR")
  @ FINLGN: ADR%(NUM)=LGN: OLD=LGN: LGN=LGN+INC
  while LEFT$(CMD$,9)="PROCEDURE"
    NOM=NOM+1: NOM%(NOM)=PRO+1: @ PROCEDURE: @ LITCMD
  endwhile: R1$="REM"+CHR$(10)+"-- "
  R2$=" "+NOM$(NUM)+CHR$(10)
  if LGN-OLD<>INC
    @ PRINT(STR$(OLD)+"GOTO"+STR$(LGN))
    @ PRINT(STR$(OLD-1)+R1$+"DEBUT DE"+R2$)
    @ PRINT(STR$(LGN-1)+R1$+"CORPS DE"+R2$)
  else LGN=OLD
    @ PRINT(STR$(LGN-1)+R1$+"PROCEDURE"+R2$)
  endif: @ BLOC
  if NOM<>-1
    for N=0 to NOM: NOM$(NOM%(N))="": endfor
  endif
end

begin -- COMPILATEUR
   @ INITIALISATION: @ LITCMD
  while LEFT$(CMD$,9)<>"PROCEDURE": @ ECRIT(CMD$): @ LITCMD
  endwhile: @ FINLGN
  @ PRINT(STR$(LGN)+"IF PEEK("+STR$(HIM)+")<>32 THEN PRINT CHR$(4)"+Q$+"BLOAD PI
LE,A"+STR$(HIM)+Q$)
  LGN=LGN+INC
  @ PRINT(STR$(LGN)+"HIMEM:"+STR$(HIM)+":DIM ZZ$(19),ZZ(19),YY(9),YY$(9):"+CAL$+
"=:"+CAL$+"GOSUB"+STR$(LGN+INC)+":END")
  LGN=LGN+INC: @ PROCEDURE : @ SORTIE
end
```

```
5  REM

** PSEUDO-APPLESOFT TO H-BASIC **
10 N$ = "TOTO.H"
20  TEXT : HOME : VTAB 10: HTAB 5: PRINT "FABRICATION DU FICHIER ";N$
30  A$  =  "300:A5 67 85 06 A5 68 85 07 A0 02 B1 06 C9 64 B0 05 C8 B1 06 F0 06 A
0 04 A9 20 91 06 A0 00 B1 06 AA C8 B1 06 86 06 85 07
D0 DF 60 N D823G"
40  FOR I = 1 TO  LEN (A$): POKE 511 + I, ASC ( MID$ (A$,I,1)) + 128: NEXT : POK
E 72,0: CALL  - 144: ONERR  GOTO 60
50  D$  =  CHR$  (13)  +  CHR$ (4): PRINT D$"MONO": PRINT D$"OPEN"N$D$"DELETE"N
$D$"OPEN"N$D$"WRITE"N$: CALL 768: HOME : POKE 33,33:
LIST 100,63999: PRINT D$"CLOSE": POKE 792,178: CALL 768: TEXT : END
60  PRINT D$"CLOSE": IF  PEEK (222) = 5 OR  PEEK (222) = 255 THEN  END
70  PRINT : PRINT "ERREUR DISQUE NUMERO "; PEEK (222)
80  REM
90  REM
```

```
 1 ;*********************************
 2 ;*                               *
 3 ;* 'INPUT' CONTIENT LES ROUTINES *
 4 ;* UTILITAIRES DESTINEES A AUG-  *
 5 ;* MENTER LA VITESSE D'EXECUTION *
 6 ;* DU COMPILATEUR DE H-BASIC.    *
 7 ;*    (ROUTINE RELOGEABLES)      *
 8 ;*                               *
 9 ;*  (C) OLIVIER HERZ POUR POM'S  *
10 ;*                               *
11 ;*********************************
12 ;
13 ;   MEMOIRES UTILISEES
14 ;
15 RETURN   EPZ 13              ;RETOUR CHARIOT
16 FRETOP   EPZ $6F             ;PTR DE LA ZONE LIBRE
17 VARPNT   EPZ $83             ;POINTEUR D'ADRESSE
18 CAR      EPZ $FB             ;CARACTERE CHERCHE
19 ADR      EPZ $FC             ;VALEUR D'UNE ADRESSE
20 PAR      EPZ $FE             ;COMPTE DES PARENTHESES
21 LNG      EPZ $FF             ;LONGUEUR D'UNE CHAINE
22 CHRGET   EQU $B1             ;INC. ET LIT LE PTR DE PGM
23 CHRGOT   EQU $B7             ;LIT LE POINTEUR DE PGM
24 ;
25 ;   ADRESSES UTILISEES
26 ;
27 PILE     EQU $7D03           ;ROUTINES DE PILE
28 BUF      EQU $8100           ;BUFFER D'ENTREE
29 ;
30 ;   ROUTINES APPLESOFT
31 ;
32 INCHR    EQU $D553           ;LIT UN CHR EN ENTREE
33 SNTX     EQU $DEC9           ;SYNTAX ERROR
34 PTRGET   EQU $DFE3           ;CHERCHE UNE VARIABLE
35 GETSPA   EQU $E452           ;LIBERE DE LA PLACE
36 LONGERR  EQU $E5B2           ;STRING TOO LONG ERROR
37 MOVSTR   EQU $E5E2           ;DEPLACE UNE CHAINE
38 ;
39          ORG $8000
40          OBJ $800
41 ;
42 ;   ANALYSE DE LA COMMANDE
43 ;
44          JSR CHRGOT
45          CMP #132
46          BEQ )0
47          JMP PILE            ;ROUTINE DE PILE
48 ^0       JSR CHRGET
49          CMP #';'
50          BEQ GET2PTS         ;LECTURE COMMANDE
51          CMP #208            ;=
52          BNE )1
53          LDA #'='
54 ^1       STA CAR
55 ;
56 ;   RECHERCHE D'UN SIGNE DANS UNE CHAINE
57 ;
58          JSR CHRGET
59          JSR PTRGET          ;NOM DE LA VARIABLE CHAINE
60          LDY #0
61          STY PAR
62          LDA (VARPNT),Y      ;STOCKE LA LONGUEUR
63          STA LNG
64          INY
65          LDA (VARPNT),Y      ;STOCKE L'ADRESSE
66          STA ADR
67          INY
68          LDA (VARPNT),Y
69          STA ADR+1
70          LDY #$FF
71 PRENCAR  CPY LNG             ;FIN DE LA CHAINE?
72          BEQ FINI
73          INY
74          LDA (ADR),Y         ;ON CHERCHE LE SIGNE
75          CMP CAR
76          BNE )1
77          LDA PAR             ;SI TROUVE EST IL ENTRE () ?
78          BEQ FINI
79 ^1       CMP #'"'            ;EST-CE '"'?
80          BNE )3
81 ^2       CPY LNG             ;ON SAUTE L'ESPACE
82          BEQ FINI            ;SITUE ENTRE 2 "
83          INY
84          LDA (ADR),Y
85          CMP #'"'
86          BNE (2
87          BEQ PRENCAR
88 ^3       CMP #'('            ;EST-CE '('?
89          BNE )4
90          INC PAR             ;SI OUI, UNE DE PLUS
91          CLC
92          BCC PRENCAR         ;JMP RELOGEABLE
93 ^4       CMP #')'            ;EST-CE ')'?
94          BNE PRENCAR
95          DEC PAR             ;SI OUI UNE DE MOINS
96          BPL PRENCAR
97          LDA #0              ;UNE DE TROP!
98          STA PAR
99          BEQ PRENCAR
100 FINI    INY                 ;ON STOCKE LA PLACE
101         STY LNG             ;DU SIGNE DANS LNG
102         RTS
103 ;
104 ;   LECTURE D'UNE INSTRUCTION SUR LE DISQUE
105 ;
106 GET2PTS  JSR CHRGET
107         JSR PTRGET          ;CHAINE D'ACCUEIL
108         LDX #0
109 ^0      JSR INCHR           ;ON LIT UN CARACTERE
110         CMP #'!'            ;LES ESPACES ET CTRL
111         BLT (0              ;SONT ELIMINES
112         CMP #'0'            ;LES CHIFFRES ET : AUSSI
113         BLT )1
114         CMP #';'
115         BGE )1
116         BLT (0
117 ^2      JSR INCHR           ;ON LIT UN CARACTERE
118 ^1      CMP #RETURN         ;FIN DE LIGNE?
119         BEQ FIN
120         CMP #'!'            ;LES ESPACES ET CTRL
```

```
121         BLT (2              ;SONT ELIMINES
122         CMP #':'            ;EST-CE UNE FIN D'INSTRUCTION?
123         BEQ FIN
124         CMP #96             ;EST-CE UNE MINUSCULE?
125         BLT )4
126         AND #$5F            ;SI OUI, -) MAJUSCULE
127 ^4      STA BUF,X           ;CARACTERE ENTRE
128         INX                 ;INCREMENTE LE POINTEUR
129         CPX #$FF            ;COMMANDE TROP LONGUE?
130         BEQ TOOLONG
131         CMP #'"'            ;EST-CE '"'?
132         BNE (2
133 ^3      JSR INCHR           ;ON LIT UNE CHAINE
134         CMP #RETURN
135         BEQ FIN
136         STA BUF,X
137         INX
138         CPX #$FF            ;COMMANDE TROP LONGUE?
139         BEQ TOOLONG
140         CMP #'"'
141         BNE (3
142         BEQ (2
143 FIN     LDA #0              ;MISE EN PLACE DE LA CHAINE
144         STA BUF,X
145         TXA
146         STA LNG
147         JSR GETSPA          ;ON CHERCHE DE LA PLACE
148         LDX #BUF
149         LDY /BUF
150         JSR MOVSTR          ;ON LA TRANSFERE
151         LDA LNG             ;ON ECRIT SA LONGUEUR
152         STA (VARPNT),Y
153         INY
154         LDA FRETOP          ;ON ECRIT SON ADRESSE
155         STA (VARPNT),Y
156         INY
157         LDA FRETOP+1
158         STA (VARPNT),Y
159         RTS
160 TOOLONG  JMP LONGERR
161         DCM "BSAVE INPUT,A$800,L$E0"
162         END
```

```
1 ;*******************************
2 ;*                             *
3 ;* 'PILE' CONSTITUE L'ENSEMBLE *
4 ;* DES ROUTINES NECESSAIRES A  *
5 ;* UN PROGRAMME H-BASIC UNE    *
6 ;* FOIS COMPILE EN APPLESOFT.  *
7 ;*                             *
8 ;*   (C) O.HERZ POUR POM'S     *
9 ;*                             *
10 ;*******************************
11 ;
12 ;   MEMOIRES UTILISEES
13 ;
14 A1L      EPZ $3C             ;SERT POUR LE MOVE
15 A2L      EPZ $3E             ;  IDEM
16 A4L      EPZ $42             ;  IDEM
17 ARYTAB   EPZ $6B             ;DEBUT DES TABLEAUX
18 STREND   EPZ $6D             ;FIN DES TABLEAUX
19 FRETOP   EPZ $6F             ;FIN DE LA ZONE LIBRE
20 CURLIN   EPZ $75             ;LIGNE COURANTE
21 DATPTR   EPZ $7D             ;POINTEUR DE DATA
22 VARNAM   EPZ $81             ;NOM DE VARIABLE
23 VARPNT   EPZ $83             ;POINTEUR DE VARIABLE
24 LOWTR    EPZ $9B             ;POINTEUR DE TABLEAU
25 ;
26 CHRGET   EPZ $B1             ;LIT UN CHR. DU PGM.
27 CHRGOT   EPZ $B7             ; IDEM SANS INC. DE TXTPTR
28 TXTPTR   EPZ $B8             ;POINTEUR DANS LE PROGRAMME
29 MEM      EPZ $D7             ;MEMOIRE POUR PILE
30 ;
31 ;   EMPLACEMENT DE LA PILE
32 ;
33 BASPILE  EPZ $80             ;BAS EN $8000
34 HAUTPILE EPZ $90             ;HAUT EN $8FFF
35 ;
36 ;   ROUTINES APPELEES
37 ;
38 MEMERR   EQU $D410           ;OUT OF MEMORY ERROR
39 FNDLIN   EQU $D61A           ;CHERCHE UNE LIGNE
40 NEWSTT   EQU $D7D2           ;NOUVELLE INSTRUCTION
41 GOTO     EQU $D93E           ;GOTO APPLESOFT
42 RETERR   EQU $D979           ;RETURN WITHOUT GOSUB
43 DATA     EQU $D995           ;SAUT-)FIN D'INSTRUCTION
44 LINGET   EQU $DA0C           ;LIT UN NUMERO DE LIGNE
45 MMCH     EQU $DD76           ;TYPE MISMATCH ERROR
46 SYNCHR   EQU $DEC0           ;ATTEND UN CHR. SPECIAL
47 CHKCOM   EQU $DEBE           ;ATTEND UNE VIRGULE
48 SNTX     EQU $DEC9           ;SYNTAX ERROR
49 PTRGET   EQU $DFE3           ;CHERCHE UNE VARIABLE
50 ILLEGERR EQU $E199           ;ILLEGAL QUANTITY
51 GETSPA   EQU $E452           ;PLACE POUR UNE CHAINE
52 MOVSTR   EQU $E5E2           ;RECOPIE UNE CHAINE
53 OFLWERR  EQU $E8D5           ;OVERFLOW ERROR
54 GETARYPT EQU $F7D9           ;CHERCHE UN TABLEAU
55 MOVE     EQU $FE2C           ;CMD. MOVE DU MONITEUR
56 ;
57          ORG $7D00
58          OBJ $800
59 ;
60 ; ANALYSE SYNTAXIQUE
61 ;
62          JSR CHKCOM
63          JSR CHRGOT          ;ENTREE SANS LA VIRGULE
64          CMP #208            ;= INITIALISATION
65          BEQ INIPILE
66          CMP #'@'            ;RESTORE NNNN
67          BEQ RESTORE
68          CMP #176            ;SUPER-GOSUB
69          BEQ GOSUB
70          CMP #177            ;SUPER-RETURN
71          BEQ RETURN
72          CMP #'#'            ;AFFECTATION DE TABLEAUX
73          BNE )0
74          JMP AFFECT
75 ^0       CMP #207            ;)  EMPILE
76          BNE )1
77          JMP EMPVAR
78 ^1       CMP #209            ;(  DEPILE
79          BNE )2
80          JMP DEPVAR
81 ^2       JMP SNTX
82 ;
```

```
 83 ;   INITIALISATION DE LA PILE
 84 ;
 85 INIPILE  JSR CHRGET
 86         LDA #0
 87         STA PILE1+1
 88         STA PILE2+1
 89         LDA #HAUTPILE
 90         STA PILE1+2
 91         STA PILE2+2
 92         RTS
 93 ;
 94 ;   RESTORE AVEC NUMERO DE LIGNE
 95 ;
 96 RESTORE  JSR CHRGET
 97         JSR LINGET          ;LIT UN NUMERO
 98         JSR FNDLIN          ;CHERCHE LA LIGNE
 99         SEC
100         LDA LOWTR           ;MET LE POINTEUR
101         SBC #1              ;DE DATA
102         STA DATPTR          ;AU DEBUT DE
103         LDA LOWTR+1         ;LA LIGNE
104         SBC #0
105         STA DATPTR+1
106         RTS
107 ;
108 ;   GOSUB QUASI-ILLIMITE
109 ;
110 GOSUB    JSR CHRGET
111         LDA TXTPTR          ;ON EMPILE TXTPTR
112         JSR EMPILE
113         LDA TXTPTR+1
114         JSR EMPILE
115         LDA CURLIN          ;ON EMPILE CURLIN
116         JSR EMPILE
117         LDA CURLIN+1
118         JSR EMPILE
119         LDA #176            ;RETURN
120         JSR EMPILE          ;ON EMPILE LE TOKEN
121         JSR CHRGOT
122         JMP GOTO            ;ROUTINE APPLESOFT
123 ;
124 ;   RETURN DU SUPER-GOSUB
125 ;
126 RETURN   JSR CHRGET
127         JSR DEPILE
128         CMP #176            ;GOSUB
129         BEQ )0
130         JMP RETERR
131 ^0       JSR DEPILE          ;ON DEPILE CURLIN
132         STA CURLIN+1
133         JSR DEPILE
134         STA CURLIN
135         JSR DEPILE          ;ON DEPILE TXTPTR
136         STA TXTPTR+1
137         JSR DEPILE
138         STA TXTPTR
139         JMP DATA            ;FIN D'INSTRUCTION
140 ;
141 ;   AFFECTATION DE TABLEAUX
142 ;
143 AFFECT   JSR CHRGET
144         JSR GETARYPT        ;1ER TABLEAU
145         LDA LOWTR+1         ;ON SAUVE
146         PHA                 ;L'ADRESSE
147         LDA LOWTR           ;DE SON NOM
148         PHA
149         LDA VARNAM          ;ON STOCKE LE TYPE
150         EOR VARNAM+1        ;DE VARIABLE
151         AND #$80
152         STA MEM
153         LDA #208            ;=
154         JSR SYNCHR
155         LDA #'#'
156         JSR SYNCHR
157         JSR GETARYPT        ;2EME TABLEAU
158         LDA VARNAM          ;EST-CE LE BON TYPE?
159         EOR VARNAM+1
160         AND #$80
161         CMP MEM
162         BNE )1              ;SI NON, MISMATCH ERROR
163         CLC
164         PLA                 ;ON RECUPERE L'ADRESSE
165         STA VARPNT          ;DU NOM
166         ADC #2              ;DU 1ER TABLEAU
167         STA A4L             ;QU'ON STOCKE EN VARPNT
168         PLA                 ;ET ON STOCKE
169         STA VARPNT+1        ;L'ADRESSE
170         ADC #0              ;DE SA TAILLE EN A4L
171         STA A4L+1
172         CLC
173         LDA LOWTR           ;ET ON STOCKE CELLE
174         ADC #2              ;DE LA TAILLE
175         STA A1L             ;DU 2ND EN A1L
176         LDA LOWTR+1
177         ADC #0
178         STA A1L+1
179         LDY #0
180         LDA (A4L),Y         ;ON COMPARE
181         CMP (A1L),Y         ;LES TAILLES
182         BEQ )2              ;DES 2 TABLEAUX
183 ^1      JMP MMCH
184 ^2      INY                 ;Y=1
185         LDA (A4L),Y
186         CMP (A1L),Y
187         BNE (1
188         DEY                 ;Y=0
189         CLC
190         LDA LOWTR           ;ON STOCKE
191         ADC (A1L),Y         ;L'ADRESSE DE LA FIN
192         STA A2L             ;DU 2ND TABLEAU EN A2L
193         DEC A2L             ;QU'ON DECREMENTE
194         INY                 ;Y=1
195         LDA LOWTR+1
196         ADC (A1L),Y
197         STA A2L+1
198         LDX A2L
199         INX
200         BNE )3
201         DEC A2L+1
202 ^3      DEY                 ;Y=0
203         CLC                 ;ET CELLE DU 1ER
204         LDA VARPNT          ;EN VARNAM
205         ADC (A4L),Y
206         STA VARNAM
207         INY                 ;Y=1
208         LDA VARPNT+1
```

```
209          ADC (A4L),Y
210          STA VARNAM+1
211          DEY                ;Y=0
212          JSR MOVE           ;RECOPIE DU 2ND TABLEAU
213          LDA MEM            ;TABLEAU DE CHAINES?
214          BEQ )0             ;SI NON, C'EST FINI
215          CLC
216          LDY #4             ;SI OUI, RECOPIE DES CHAINES
217          LDA (VARPNT),Y     ;NOMBRE DE DIMENSIONS
218          ASL
219          CLC
220          ADC #5             ;ON CHERCHE LE DEBUT
221          ADC VARPNT         ;DU TABLEAU PROPREMENT DIT
222          STA VARPNT
223          LDA #0
224          ADC VARPNT+1
225          STA VARPNT+1
226 ^4       LDY #0
227          LDA (VARPNT),Y     ;LONGUEUR DE LA CHAINE
228          PHA
229          JSR GETSPA         ;ON CHERCHE LA PLACE
230          LDY #1             ;POUR LA COPIER
231          LDA (VARPNT),Y
232          TAX
233          INY
234          LDA (VARPNT),Y
235          TAY
236          PLA
237          JSR MOVSTR         ;ON LA RECOPIE
238          LDY #1
239          LDA FRETOP
240          STA (VARPNT),Y     ;ON MET EN PLACE L'ADRESSE
241          INY                ;DE LA CHAINE RECOPIEE
242          LDA FRETOP+1
243          STA (VARPNT),Y
244          CLC
245          LDA VARPNT         ;ON PASSE A LA
246          ADC #3             ;PROCHAINE CHAINE
247          STA VARPNT
248          LDA VARPNT+1
249          ADC #0
250          STA VARPNT+1
251          CMP VARNAM+1       ;A T-ON FINI LE TABLEAU?
252          BLT (4             ;SI NON, PROCHAINE CHAINE
253          BNE OVFL
254          LDA VARPNT
255          CMP VARNAM
256          BLT (4
257          BNE OVFL
258 ^0       RTS
259 ;
260 ;   ON EMPILE UNE VARIABLE
261 ;
262 EMPVAR   JSR CHRGET
263          CMP #'#'           ;TABLEAU?
264          BEQ EMPTAB
265          JSR EMPNOM         ;ON EMPILE SON NOM
266          SEC
267          LDA VARPNT         ;ON MET L'ADRESSE
268          SBC #2             ;DU NOM EN VARPNT
269          STA VARPNT
270          LDA VARPNT+1
271          SBC #0
272          STA VARPNT+1
273          JSR CACHNOM        ;ON CACHE SON NOM
274          LDA VARPNT
275          JSR EMPILE         ;ON EMPILE L'ADRESSE
276          LDA VARPNT+1       ;DE LA VARIABLE
277 FINEMP   JSR EMPILE
278          JSR CHRGOT
279          CMP #','           ;AUTRE VARIABLE?
280          BEQ EMPVAR
281          RTS
282 OVFL     JMP OFLWERR        ;NE DOIT PAS ARRIVER
283 ;
284 ;   ON EMPILE UN TABLEAU
285 ;
286 EMPTAB   JSR CHRGET
287          JSR EMPNOM         ;ON EMPILE SON NOM
288          LDA ARYTAB         ;ON CHERCHE SI
289          STA VARPNT         ;UN TABLEAU (EVENTUEL)
290          LDA ARYTAB+1       ;PORTE CE NOM
291          STA VARPNT+1       ;(GETARYPT NE MARCHE PAS)
292 BCLE     LDA VARPNT+1       ;EST ON RENDU A LA FIN
293          CMP STREND+1       ;DE LA ZONE DES TABLEAUX?
294          BLT )0
295          BNE OVFL
296          LDA VARPNT
297          CMP STREND
298          BLT )0
299          BNE OVFL
300          LDA #$FF           ;TABLEAU NON TROUVE
301          JSR EMPILE         ;ON EMPILE $FFFF
302          JSR EMPILE
303          LDA #'#'
304          JMP FINEMP
305 ^0       LDY #0
306          LDA (VARPNT),Y     ;EST-CE LE BON NOM?
307          CMP VARNAM
308          BNE )1
309          INY
310          LDA (VARPNT),Y
311          CMP VARNAM+1
312          BNE )1
313          JSR CACHNOM        ;ON CACHE SON NOM
314          SEC                ;ON EMPILE SON ADRESSE
315          LDA VARPNT
316          SBC ARYTAB         ;RELATIVE AU DEBUT
317          TAX                ;DE LA ZONE DES TABLEAUX
318          LDA VARPNT+1
319          SBC ARYTAB+1
320          TAY
321          TXA
322          JSR EMPILE
323          TYA
324          JSR EMPILE
325          LDA #'#'           ;ON EMPILE UN #
326          JMP FINEMP
327 ^1       CLC                ;NOM NON TROUVE
328          LDY #2             ;ON CHERCHE
329          LDA (VARPNT),Y     ;LE TABLEAU SUIVANT
330          ADC VARPNT
331          TAX
332          INY
333          LDA (VARPNT),Y
334          ADC VARPNT+1
```

```
335           STA VARPNT+1
336           STX VARPNT
337           JMP BCLE
338 ;
339 ;   ON DEPILE UNE VARIABLE
340 ;
341 DEPVAR    JSR CHRGET
342           CMP #'#'            ;TABLEAU?
343           BEQ DEPTAB
344           JSR PTRGET          ;ON CHERCHE LA VARIABLE
345           JSR DEPILE          ;ON DEPILE SON ADRESSE
346           STA VARPNT+1
347           JSR DEPILE
348           STA VARPNT
349 DEPNOM    JSR DEPILE          ;ON DEPILE SON NOM
350           CMP VARNAM+1
351           BNE )0              ;EST-CE LE BON NOM?
352           LDY #1
353           STA (VARPNT),Y
354           JSR DEPILE
355           CMP VARNAM
356           BNE )0
357           DEY
358           STA (VARPNT),Y
359 FINDEP    JSR CHRGOT
360           CMP #','            ;AUTRE VARIABLE?
361           BEQ DEPVAR
362           RTS
363 ^0        JMP MMCH            ;NE DOIT PAS ARRIVER
364 ^1        JMP ILLEGERR        ;NE DOIT PAS ARRIVER
365 ;
366 ;   ON DEPILE UN TABLEAU
367 ;
368 DEPTAB    JSR CHRGET
369           JSR GETARYPT        ;ON CHERCHE LE TABLEAU
370           CLC
371           LDY #2              ;EST-CE LE DERNIER?
372           LDA (LOWTR),Y
373           ADC LOWTR
374           TAX
375           INY
376           LDA (LOWTR),Y
377           ADC LOWTR+1
378           CMP STREND+1
379           BNE (1
380           CPX STREND
381           BNE (1
382           LDA LOWTR           ;SI OUI ON REMET
383           STA STREND          ;L'ANCIEN STREND
384           LDA LOWTR+1
385           STA STREND+1
386           JSR DEPILE
387           CMP #'#'            ;A-T'ON EMPILE UN TABLEAU?
388           BNE (0
389           JSR DEPILE          ;ON DEPILE SON ADRESSE
390           CMP #$FF            ;RELATIVE
391           BEQ )0              ;Y EN AVAIT-IL UN ANCIEN?
392           TAY
393           JSR DEPILE          ;SI OUI:
394           CLC                 ;ON REMET L'ADRESSE ABSOLUE
395           ADC ARYTAB          ;DU DEBUT DU TABLEAU
396           STA VARPNT
397           TYA
398           ADC ARYTAB+1
399           STA VARPNT+1
400           JMP DEPNOM          ;ET ON DEPILE SON NOM
401 ^0        JSR DEPILE          ;SI NON, ON DEPILE DU VENT
402           JSR DEPILE
403           JSR DEPILE
404           JMP FINDEP
405 ;
406 ;   CACHE UN NOM DE VARIABLE
407 ;
408 CACHNOM   LDY #0
409           LDA (VARPNT),Y      ;ON UTILISE LE
410           AND #$80            ;PRETE-NOM AT,AT% OU AT$
411           ORA #'A'
412           STA (VARPNT),Y
413           INY
414           LDA (VARPNT),Y
415           AND #$80
416           ORA #'T'
417           STA (VARPNT),Y
418           RTS
419 ;
420 ;   EMPILE UN NOM DE VARIABLE
421 ;
422 EMPNOM    JSR PTRGET          ;CHERCHE LA VARIABLE
423           LDA VARNAM
424           JSR EMPILE
425           LDA VARNAM+1
426 ;
427 ;   EMPILEMENT D'UN OCTET
428 ;
429 EMPILE    DEC PILE1+1         ;ON DECREMENTE
430           DEC PILE2+1         ;LES POINTEURS DE PILE
431           LDX PILE1+1
432           INX
433           BNE )0
434           DEC PILE1+2
435           DEC PILE2+2
436 ^0        LDX PILE1+2
437           CPX #BASPILE        ;A T-ON TROP EMPILE?
438           BGE PILE1
439 MEMORY    JMP MEMERR
440 PILE1     STA HAUTPILE*256    ;POINTEUR DE PILE
441           RTS
442 ;
443 ;   DEPILEMENT D'UN OCTET
444 ;
445 DEPILE    LDX PILE1+2
446           CPX #HAUTPILE       ;A T-ON TROP DEPILE?
447           BGE MEMORY
448 PILE2     LDA HAUTPILE*256    ;POINTEUR DE PILE
449           INC PILE1+1         ;ON INCREMENTE
450           INC PILE2+1         ;LES POINTEURS DE PILE
451           BNE )0
452           INC PILE1+2
453           INC PILE2+2
454 ^0        RTS
455           DCM "BSAVE PILE,A$800,L$2F0"
456           END
```