GRAMMAIRE

DU

LANGAGE

BASIC

GRAMMAIRE DU LANGAGE BASIC

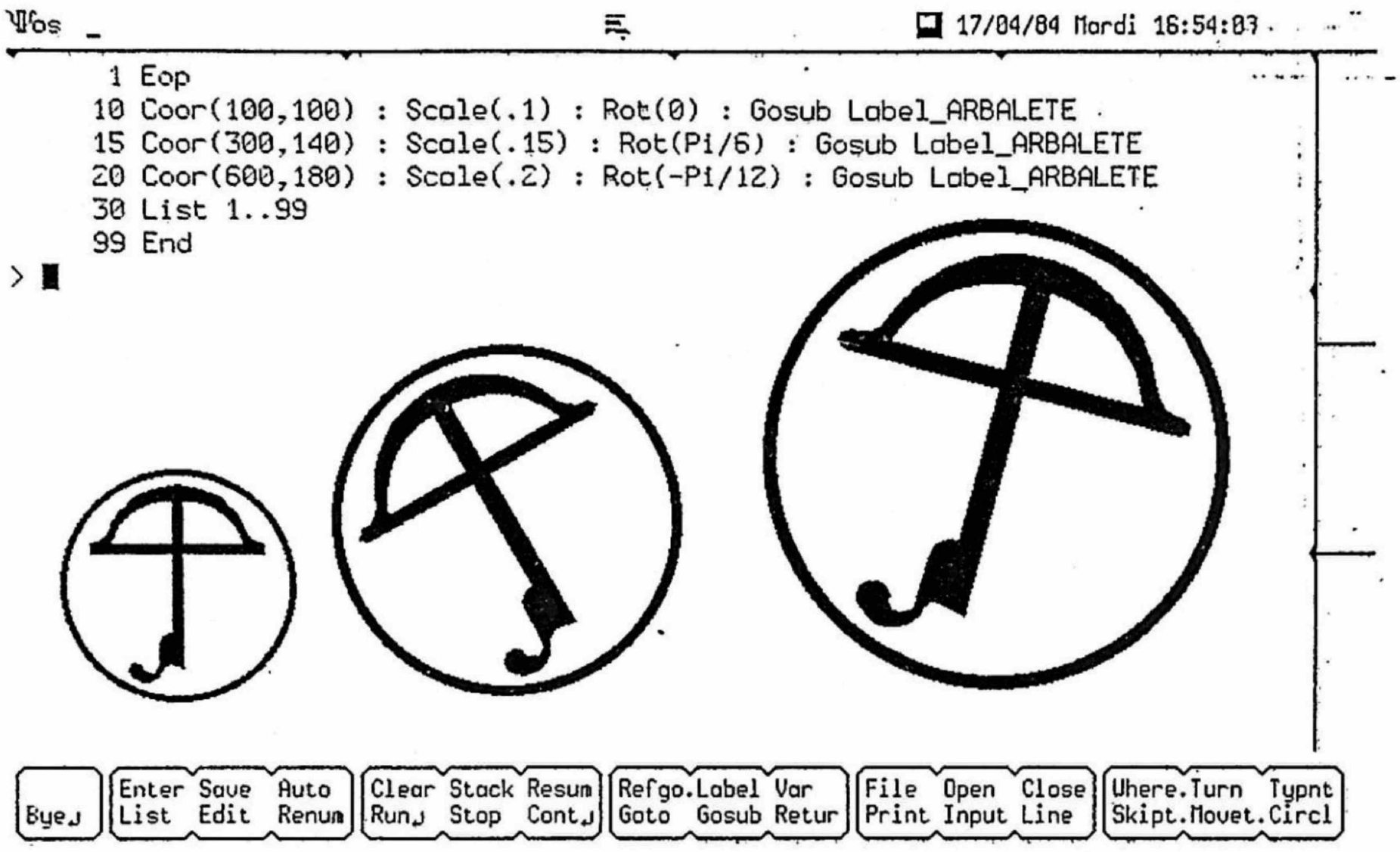


Table des matières

BASIC 6.65, St-Aubin le 19.4.84, J.M.Paratte

1 GRAMMAIRE BASIC

Cette grammaire définit la construction d'une commande ou d'un programme BASIC en décrivant chaque ordre, chaque type de donnée, chaque élément nécessaire à la construction rigoureuse et seule acceptée d'un programme.
La présentation qui suit est un résumé complet qui s'utilise avec une bonne dose de réflexion logique. Examinez les exemples détaillés et expliqués ci-dessous.

1.1 Définitions et abréviations syntaxiques

|	métasymbole utilisé pour séparer des choix exclusifs.
[]	métasymboles utilisés pour isoler un élément ou un groupe d'éléments qui est optionnel.
{}	métasymboles utilisés pour isoler un élément ou un groupe d'éléments qui peut se répéter.
--	commentaire ou remarque dans une définition (jusqu'à la fin de la ligne et les lignes suivantes).
ELEMENT ELEMENT_COMPOSE GRAND_ELEMENT_COMPOSE	désigne un élément avec son schéma de construction comme par exemple une expression numérique. Un élément est toujours écrit en lettres majuscules. Le symbole _ permet de lier plusieurs mots (comme COMMANDE_DIRECTE) afin de donner un sens plus explicite du fonctionnement de l'élément.
ELEMENT%	désigne un élément de donnée du type entier. Des ELEMENT% sont par exemple: - EXPR% une expression quelconque du type entier. - VAR% une variable du type entier. - CAR% une expression du type entier dont le nom explique le sens de l'expression à calculer (dans ce cas, un caractère ascii).
ELEMENT! ELEMENT$ ELEMENT%!	comme ci-dessus mais pour le type réel. comme ci-dessus mais pour le type "string". comme ci-dessus mais acccepte les 2 types entier et réel.
Mot	désigne un mot réservé BASIC, par exemple: Print, Sin, Mod. Un mot réservé est toujours écrit avec la première lettre en majuscule et les suivantes en minuscules.
+ (=	désigne un symbole qui doit être écrit selon la présentation demandée.

1.2 Exemples accompagnés d'explications

1.2.1 L'élément CHIFFRE

CHIFFRE 0|1|2|3|4|5|6|7|8|9

Définition: l'élément CHIFFRE est formé au choix de l'un des 10 chiffres de la base décimale. Lorsque l'élément CHIFFRE est demandé, il faut "répondre" avec un des caractères numériques (0,1,2,3,...).

1.2.2 L'élément NOMBRE

NOMBRE CHIFFRE {CHIFFRE}

Définition: l'élément NOMBRE est formé d'un élément CHIFFRE suivi optionnellement d'autres éléments CHIFFRE. L'élément NOMBRE appelle la définition de l'élément CHIFFRE. Ainsi, avec cette construction, il est possible de générer un nombre formé d'un seul CHIFFRE (ex: le NOMBRE 3) ou de plusieurs (exemple: le NOMBRE 257).

1.2.3 L'élément CARACTERE

CARACTERE LETTRE | CHIFFRE | SPECIAL

Définition: l'élément CARACTERE est formé au choix par l'un des 3 éléments LETTRE, CHIFFRE ou SPECIAL. Chacun de ces 3 éléments appellent eux-même une nouvelle définition d'élément. L'élément CARACTERE englobe presque tous les caractères du clavier (exemples: la LETTRE A, le CHIFFRE 5, le caractère SPECIAL /).

1.2.4 La commande Auto

Auto START_LIGNE%,INC_LIGNE% -- pour le fonctionnement de Auto, voir plus loin.

Définition: il y a 4 éléments qui sont de gauche à droite:

- Auto — un mot réservé qui doit être écrit.
- START_LIGNE% — une expression du type entier donnant le 1er numéro de ligne.
- , — un symbole qui doit être écrit.
- INC_LIGNE% — une expression du type entier donnant l'incrément de chaque nouvelle ligne.

Les éléments START_LIGNE% et INC_LIGNE% ont une construction identique à l'élément EXPR% (expression du type entier) mais avec des limites (<= 0 interdit).

2 JEU de CARACTERES

CARACTERE LETTRE | CHIFFRE | SPECIAL

LETTRE A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

CHIFFRE 0|1|2|3|4|5|6|7|8|9

SPECIAL %|!|#|$|@|'|"|+|-|*|/|(|)|<|>|=|&|?|,|;|_|.| |:

RETURN -- la touche (RETURN), qui termine l'introduction de chaque ligne de commande. Ce caractère n'est pas représenté bien qu'il soit nécessaire.

CAR_ASCII -- une touche du clavier, n'importe quel code.

3 ATOMES

ATOME
| DELIMITEUR -- arithm. ou autre
| MOT_RESERVE -- ex: Print, Next, Mod, Sin, ...
| VARIABLE -- ex: VITESSE
| LABEL -- ex: Label_CALCUL
| CONSTANTE -- numérique ou "string"
| NUM_LIGNE -- numéro de ligne

```
NOM                     LETTRE {LETTRE|CHIFFRE} -- suite de lettres et de chiffres.

NOMBRE                  CHIFFRE {CHIFFRE} -- suite de chiffres.
```

3.1 Délimiteurs

```
DELIMITEUR              DELIMITEUR_SIMPLE | DELIMITEUR_COMPOSE

DELIMITEUR_SIMPLE       +|-|*|/|(|)|=|&|:|;|,|.|(|)

DELIMITEUR_COMPOSE      **|(=|)=|<>|..|--
```

3.2 Mots réservés

```
MOT_RESERVE             NOM
                        -- mot réservé du language BASIC. ex: Print, Sin, Mod.
                           Ces mots ne sont pas utilisables pour des noms de
                           variables mais sont utilisables pour des noms de LABEL.
```

3.3 Variables

```
VARIABLE                | VAR%          -- entière
                        | VAR!          -- réelle (flottante)
                        | VAR$          -- "string"
```

3.4 Labels (étiquettes)

```
LABEL                   Label_NOM       -- le mot Label_ et un NOM
                                           ex: Goto Label_BOUCLE
```

3.5 Constantes

```
CONSTANTE               | CONST%        -- entière
                        | CONST!        -- réelle (flottante)
                        | CONST$        -- "string"
```

3.6 Numéros de ligne

```
NUM_LIGNE               NOMBRE          -- 1 <= NUM_LIGNE <= 32767
```

4 LIGNES BASIC, FORMAT, BLOC

```
COMMANDE                | COMMANDE_DIRECTE      -- exécution immédiate
                        | COMMANDE_INDIRECTE    -- exécution différée

COMMANDE_DIRECTE        ORDRES                  -- au maximum 254 caractères

COMMANDE_INDIRECTE      | COMM_IND_EFFACE       -- efface une ligne
                        | COMM_IND_INSERE_CHANGE -- insère ou change une ligne

COMM_IND_EFFACE         NUM_LIGNE

COMM_IND_INSERE_CHANGE  NUM_LIGNE | LABEL_DEF_IND [: ORDRES]
                                  |     -- définit une ligne et un label,
                                  |        indentation impossible.
                                  | [INDENT] ORDRES
                                       -- définit une ligne,
                                          indentation optionnelle.
```

LABEL_DEF_IND — LABEL [(PARAM_LABEL_DEF {, PARAM_LABEL_DEF})]
-- définit un label et déclare optionnellement des paramètres pour une routine.

BLOC — NUM_LIGNE [INDENT] ORDRES
-- un bloc est formé d'une ou plusieures lignes de commandes indirectes.

INDENT — -- une suite de caractères (ESPACE)

ORDRES — ORDRE {: ORDRE} -- une suite d'ordres BASIC.

5 QUELQUES PARAMETRES PARTICULIERS

RANGE_TEXTE — [NUM_LIGNE|LABEL] [.. [NUM_LIGNE|LABEL]]
-- désigne une partition du texte du programme BASIC.
RANGE_TEXTE permet de construire:

10	la ligne 10 uniquement.
10..20	les lignes 10 à 20 y compris.
10..	la ligne 10 et les suivantes jusqu'à la fin du texte.
..99	les lignes du début jusqu'à la ligne 99 comprise.

FICHIER$ — -- une expression "string" désignant un fichier quelconque (exemples: #PRINTER, STOCK.GESDO).

FICHIER_BAS$ — -- une expression "string" désignant un fichier BASIC. L'extension '.BAS' est accolée automatiquement si aucune extension n'est précisée (exemples: FORMULES ==) FORMULES.BAS, ESSAI.BAS0 inchangé).

FICHIER_CBAS$ — -- comme ci-dessus mais désignant un fichier BASIC "compilé" avec l'extension '.CBAS'.

5.1 Quelques expressions du type entier

COL% — -- un numéro de colonne sur l'écran, de gauche à droite. La colonne de gauche a toujours la valeur 0. La dernière colonne de droite a une valeur qui dépend du générateur de caractères utilisé.

LIGNE% — -- un numéro de ligne sur l'écran, de haut en bas. La ligne du haut a toujours la valeur 0. La dernière ligne a une valeur qui dépend du générateur utilisé.

LEN% — -- exprime une longueur.

CAR% — -- exprime la valeur d'un caractère dans le code ascii.

INDEX% — -- exprime la valeur d'un index dans un "string". Le 1er caractère est pointé lorsque l'INDEX% vaut 1.

CANAL% — -- exprime le numéro de canal d'un fichier BASIC. La valeur est comprise entre 0 et 7, 0 désignant toujours l'écran et le clavier.

6. ORDRES et FONCTIONS, GENERALITES

ORDRE -- un ordre avec sa construction sémantique.
ex: Print...

FONCTION -- une fonction ou une "varsys" avec sa construction.
ex: Sin(EXPR!)

6.1 Symboles spéciaux de programmation

: séparateur d'ordre dans un ligne.
? remplace l'ordre Print.
-- remplace l'ordre Rem ou : Rem.

6.2 Commandes de manipulations de programmes

```
Audit | FICHIER$          -- copie l'écran depuis cette commande dans un fichier.
      | End               -- termine la copie d'écran.
Auto [ START_LIGNE%,INC_LIGNE% ]
                          -- génère automatiquement des numéros de ligne. Lorsque
                             aucun paramètre n'est précisé, Auto démarre à 100 avec
                             un pas de 10.
Bye                       -- quitte l'interpréteur BASIC.
Clear                     -- efface le "stack".
Copy FICHIER_CBAS$        -- sauve le programme "compilé" dans un fichier.
Del [RANGE_TEXTE]         -- efface une portion du programme.
Delete FICHIER$           -- efface un fichier.
Edit [RANGE_TEXTE]        -- met en édition une portion du programme.
Enter | FICHIER_BAS$      -- connecte un fichier (remplace le clavier).
      | End               -- déconnecte et ferme le "fichier Enter".
      | Off               -- suspend le "fichier Enter".
      | On                -- réactive le "fichier Enter".
Load FICHIER_CBAS$        -- charge un programme "compilé".
New                       -- efface tout et remet à jour l'interpréteur BASIC.
Program EXPR$             -- démarre un programme depuis le BASIC (voir l'appel
                             ?EXECUTE dans le manuel FOS). Le déroulement du
                             programme BASIC est interrompu tant que le programme
                             démarré est actif. Ensuite le clavier est redonné au
                             programme BASIC dont l'exécution reprendra.
                             Exemple: Program "FILER LIST *BAS".
Rename FICHIER$, NOUVEAU_NOM_DE_FICHIER$
                          -- change le nom d'un fichier.
Renum [ [RANGE_TEXTE] [=[START_LIGNE%][,[INC_LIGNE%]]] ]
                          -- renumérote une portion du programme. Lorsque aucun
                             paramètre n'est précisé, Renum démarre à 100 avec un pas
                             de 10.
Save FICHIER_BAS$ [,Upper][,_Off][,New][,Run]
                          -- sauve le programme dans un fichier de texte (code ascii).
                             L'option Upper sauve le programme en lettres majuscules.
                             L'option _Off supprime le symbole "_" dans le programme
                             pendant le sauvetage.
                             L'option New envoie une commande New dans le fichier de
                             sauvetage devant le programme.
                             L'option Run envoie une commande Run dans le fichier de
                             sauvetage après le programme.
```

6.3 Commandes de Listages

```
Data                    -- liste l'état des ordres Data,Read,Restore.
Function                -- liste les fonctions BASIC.
Label                   -- liste les "labels" du programme.
List [RANGE_TEXTE]      -- liste une portion du programme.
Oper                    -- liste les opérateurs BASIC.
Order                   -- liste les ordres et mots réservés particuliers du BASIC.
Refgoto                 -- liste les lignes du texte référencées par Goto,...
Stack                   -- liste le contenu du "stack".
Var                     -- liste les variables avec leur type et leur contenu.
Varsys                  -- liste les variables systèmes du BASIC.
```

6.4 Exécution, dépannage de programme

```
Cont                    -- continue l'exécution après l'ordre Stop.
On Error Then | Stop    -- arrête l'exécution du programme à la prochaine erreur.
              | [Goto] NUM_LIGNE|LABEL -- provoque un débranchement dans l'exécution
                                       à la prochaine erreur.
Pop                     -- enlève l'élément sur le sommet du "stack".
Program EXPR$           -- exécute un programme Smaky8. Le programme BASIC est
                           suspendu pendant l'exécution de cet ordre. L'ordre est
                           terminé dès que le programme Smaky8 est terminé.
                           Ex: Program 'EPRO 5000/M TRAPEZE.BAS' -- démarre EPRO
                               avec l'édition du fichier TRAPEZE.BAS.
Resume |                -- réexécute l'ordre qui a provoqué une erreur et continue.
       | Next           -- saute l'ordre qui a provoqué une erreur et continue.
       | NUM_LIGNE|LABEL -- oublie l'ordre et la routine qui ont provoqué une
                            erreur et continue.
Run                     -- démarre le programme (terminé par les ordres End ou Bye).
Stop                    -- suspend l'exécution d'un programme (voir l'ordre Cont).
```

6.5 Ordres généraux de programmation

```
Bye                     -- termine l'exécution d'un programme et quitte le BASIC.
Def NOM_FNI(PARAM_FN_DEF{,PARAM_FN_DEF})]=EXPR -- définit un fonction.
Dim NOM_DIM(EXPR%{,EXPR%}) -- définit les dimensions d'un tableau.
Edit Line LIGNE$, INDEX%, MAX%, CAR%
                        -- édite le "string" LIGNE$, place le curseur
                           sur le INDEX%ième caractère. La longueur du "string" est
                           limitée par MAX%. Le caractère qui termine l'édition est
                           rendu dans CAR% (avec la touche fonction).
End                     -- termine l'exécution d'un programme.
Error EXPR%             -- provoque une erreur.
Exec EXPR$              -- analyse et interprète un "string".
For VAR%!=EXPR%! To EXPR%! [Step EXPR%!]
                        -- utilisé en conjonction avec l'ordre Next pour répéter
                           une suite d'instructions dans une ou plusieures lignes
                           entre ces deux commandes.
Gosub ! NUM_LIGNE       -- appelle une routine en NUM_LIGNE, terminée par Return.
      | LABEL [(PARAM_LABEL_CALL{,PARAM_LABEL_CALL})] -- appelle la routine LABEL
                        -- avec des paramètres (passage par valeur ou adresse).
Goto NUM_LIGNE|LABEL    -- branche l'exécution sur NUM_LIGNE ou LABEL.
If EXPR% Then [IF_CLAUSE] [Else [IF_CLAUSE]]
                        -- choix booléen entre 2 exécutions:
                           si EXPR% <> 0 alors Then [IF_CLAUSE] est exécuté
                           sinon [Else [IF_CLAUSE]] est exécuté.
      IF_CLAUSE         -- membre de l'énoncé de IF
              | ORDRES  -- une suite d'ordres (If acceoté).
              | [Goto] NUM_LIGNE|LABEL -- branchement sur NUM_LIGNE ou LABEL.
```

```
Label_NOM[(PARAM_LABEL_DEF{,PARAM_LABEL_DEF})]
                        -- définit un label avec des paramètres pour une routine
                           terminée par Return.
Modwda( XX%,YY%,DX%,DY% )
                        -- change les dimensions de la fenêtre alphanumérique
                           du texte. Il n'est plus possible d'écrire en dehors des
                           nouvelles dimensions. Définitions: voir le manuel LIB.
        XX%             -- position du cadre à gauche.
        YY%             -- position du cadre compté depuis le haut.
        DX%             -- largeur du cadre.
        DY%             -- hauteur du cadre.
Next [VAR%!{,VAR%!}]    -- termine la boucle For.
On EXPR% Gosub NUM_LIGNE!LABEL {,NUM_LIGNE!LABEL} [Else [If_CLAUSE]]
                        -- exécute un Gosub à l'adresse déterminée par EXPR%, de 1
                           à n adresses. Si EXPR% ne détermine aucune adresse,
                           [Else [If_CLAUSE]] est exécuté.
On EXPR% Goto NUM_LIGNE!LABEL {,NUM_LIGNE!LABEL} [Else [If_CLAUSE]]
                        -- branche à l'adresse déterminée par EXPR%, de 1 à n
                           adresses. Si EXPR% ne détermine aucune adresse,
                           [Else [If_CLAUSE]] est exécuté.
Print { [PRINT_EXPR] [PRINT_DELIMITEUR] }
                        -- affiche une suite d'expression. Un délimiteur est
                           optionnel. Dans le cas ou aucun délimiteur n'est fourni,
                           un caractère (SPACE) est affiché. Lorsque la dernière
                           expression est affichée, un caractère (RETURN) est envoyé
                           sur l'écran si aucun délimiteur ne suit l'exoression.
        PRINT_EXPR      -- type d'expression de l'ordre Printt.
                | EXPR                      -- une expression quelconque.
                | Format(E%,F%) EXPR%!      -- une expression numérique formatée,
                |                              E% digit devant la virgule,
                |                              F% digit après la virgule.
                | Tab(COL%)                  -- tabule horizontalement dans la ligne
                |                              mais ne recule jamais le curseur.
                | Carxy(COL%,LIGNE%)         -- positionne le curseur alphanumériaue
                |                              dans l'écran.
                | Caryx(LIGNE%,COL%)         -- comme ci-dessus mais COL% et LIGNE%
                                               croisés.
        PRINT_DELIMITEUR -- délimiteur d'expression de l'ordre Print.
                |,                           -- affiche des (SPACE), tabule module 8.
                |;                           -- n'affiche rien, accolle l'expression
                                               suivante sans (SPACE).
Rem STRING_ASCII        -- commentaire(s)
Return                  -- fin d'un routine, reprend l'exécution après Gosub.
```

6.6 Ordres d'assignations

```
Data EXPR {,EXPR}       -- établit la liste des données lues par un ordre Read.
Dec VAR%!               -- décrémente une variable (-1).
Inc VAR%!               -- incrémente une variable (+1).
Input INPUT_PROMPT VAR {,VAR} [;]
                        -- assignation de variables par le clavier.
                           [;] supprime l'écho de (RETURN) sur l'écran.
Input Line INPUT_PROMPT VAR$ [;]
                        -- assignation particulière par le clavier: une ligne
                           entière est lue et copiée dans VAR$.
        INPUT_PROMPT    -- "caractères d'annonce" de l'ordre Input.
                |               | -- affiche "? ".
                | ;             | -- supprime l'affichage de "? ".
                | CONST$  |,     | -- affiche un "string" et "? ".
                | (EXPR$) |;     | -- affiche un "string".
[Let] VAR=EXPR {,VAR=EXPR} -- assigne une variable ou plusieurs.
```

```
Read VAR {,VAR}          -- assigne une suite de variables selon des expressions
                            programmées dans des ordres Data.
Restore [NUM_LIGNE]      -- repositionne l'index des Data pour les relire.
Swap VAR,VAR             -- échange les contenus de 2 variables de même type.
```

6.7 Ordres divers

```
Accord(NOTE1%,NOTE2%,NOTE3%,ENVELOPPE%,DUREE%)
                         -- joue un accord de 3 notes:
                            NOTE1%,NOTE2%,NOTE3%=programmation par demi-ton,
                            ENVELOPPE%,DUREE%=programmation par unité de 20 ms.
Beep(BEEP%)              -- son particulier, 1 <= BEEP% <= 7.
Carxy(COL%,LIGNE%)       -- positionne le curseur sur la colonne COL% (0=gauche) et
                            sur la ligne LIGNE% (0=haut).
Caryx(LIGNE%,COL%)       -- positionne le curseur comme Carxy mais avec les
                            paramètres croisés.
Eop                      -- efface l'écran et met le curseur en haut à gauche.
Random( EXPR% )          -- détermine la valeur de départ du générateur aléatoire.
Tab( COL% )              -- positionne le curseur dans la ligne sur la colonne COL%.
```

6.8 Ordres de programmation structurée

```
Do IF EXPR% [: ORDRES]   -- test booléen et exécution d'un groupe d'instructions.
[ BLOC ]                    Si "vrai", exécute jusqu'à [ Do Else ... ] ou Do End.
[ Do Else [ORDRES] ]        Si "faux", exécute [Do Else [ORDRES] ] jusqu'à Do End.
[ BLOC ]
Do End

Loop [: ORDRES]          -- boucle avec plusieurs tests de fin de boucle
[ BLOC ]                    placés en différents endroits de la boucle.
{ Exit If EXPR% }
{ BLOC }
End Loop

Repeat [: ORDRES]        -- boucle jusqu'à ce que EXPR% soit <> 0.
[ BLOC ]
Until EXPR%

Select EXPR                          -- cherche une expression EXPR parmi plusieurs
{ Case TEST_SELECT [: ORDRES] }         alternatives. Si aucune alternative n'est
{ BLOC }                                déterminée, alors [ Case Else [ORDRES] ]
[ Case Else [ORDRES] ]                  et [ BLOC ] sont exécutés.
[ BLOC ]
End Select

        TEST_SELECT                                  -- expression de test
                TEST1_SELECT {, TEST1_SELECT}           trouvée dans une suite
        TEST1_SELECT
                | EXPR                                  de valeur
                | [EXPR] .. [EXPR]                      ou de limites.

While EXPR% [: ORDRES]           -- boucle tant que EXPR% <> 0.
[ BLOC ]
End While
```

6.9 Fonctions mathématiques

```
Abs( EXPR%! )              -- valeur absolue
Atan( EXPR%! )             -- arctangante
Cos( EXPR%! )              -- cosinus
Cosh( EXPR%! )             -- cosinus hyperbolique
Exp( EXPR%! )              -- élévation à la puissance de la constante e
Int( EXPR%! )              -- partie entière
Ln( EXPR%! )               -- logarithme naturel
Log( EXPR%! )              -- logarithme décimal
Pi                         -- approximation du nombre pi
Real( EXPR% )              -- conversion au type réel
Sgn( EXPR%! )              -- valeur du signe: 1 ou -1, 0 si EXPR%!=0
Sin( EXPR%! )              -- sinus
Sinh( EXPR%! )             -- sinus hyperbolique
Sqrt( EXPR%! )             -- racine carrée
Tan( EXPR%! )              -- tangante
Tanh( EXPR%! )             -- tangante hyperbolique
```

6.10 Fonctions caractères ("string")

```
Asc( EXPR$ )               -- donne la valeur ascii du ier caractère.
Butleft$( EXPR$,LEN% )     -- retranche LEN% caractères de la partie gauche de EXPR$.
Butright$( EXPR$,LEN% )    -- retranche LEN% caractères de la partie droite de EXPR$.
Chr$( EXPR% )              -- donne le "string" de 1 caractère dont la valeur ascii
                              est déterminée par EXPR%.
Hex$( EXPR%! )             -- donne le "string" obtenu par la conversion
                              numérique ==) hexadécimale-ascii.
Left$( EXPR$,LEN% )        -- donne LEN% caractères de la partie gauche de EXPR$.
Input$(LEN%)               -- lit LEN% caractères du clavier et donne le "string".
Instr( A$,B$[,INDEX%] )    -- cherche le "string" B$ dans A$, depuis le INDEX%ième
                              caractère, sinon depuis le début et donne l'index
                              de l'occurence, 0 dans le cas contraire.
Line$( LIGNE_BASIC% )      -- donne le "string" de la ligne LIGNE% du programme.
Lower$( EXPR$ )            -- effectue la conversion majuscule ==) minuscule de chacune
                              des lettres de EXPR$ et donne le résultat.
Mid$(EXPR$,INDEX%,LEN%)    -- extrait une portion de EXPR$ (LEN% caractères depuis
                              -- le INDEX%ième caractère) et donne le résultat.
Num$([Format(E%,F%)] EXPR%!) -- donne le "string" obtenu par la conversion
                              -- numérique ==) décimale-ascii. Le "string" est
                              optionnellement formaté (voir l'ordre Print).
Right$( EXPR$,LEN% )       -- donne LEN% caractères de la partie droite de EXPR$.
Space$( LEN% )             -- donne un "string" de LEN% caractères (SPACE) (H'20).
String$( LEN%,CAR% )       -- donne un "string" de LEN% caractères (CAR%).
Upper$( EXPR$ )            -- effectue la conversion minuscule ==) majuscule de chacune
                              des lettres de EXPR$ et donne le résultat.
Val( EXPR$ )               -- donne la valeur décimale de EXPR$.
Valhex( EXPR$ )            -- donne la valeur hexadécimale de EXPR$.
```

6.11 Fonctions et "varsys" spéciales

```
Clock$                      -- donne l'heure du système.
Dim( VAR(*), M_DIM% )       -- donne des renseignements sur un tableau:
        M_DIM% = 0          -- le nombre de dimensions du tableau
                 1          -- le nombre d'éléments de la 1ère dimension.
                 ...        -- le nombre d'éléments des dimensions suivantes.
Enter                       -- indique par "vrai" ou "faux" si le "fichier ENTER" est
                               ouvert.
Error                       -- donne la valeur de la dernière erreur BASIC.
Error$( ERREUR% )           -- donne le message d'erreur PSos.
False                       -- donne la valeur 0, résultat "faux" d'un test booléen.
Free                        -- donne le nombre de bytes libres pour le programme.
High( EXPR% )               -- donne le "high byte" d'un "integer".
Key                         -- attend un touche du clavier et retourne le code de
                               la touche ("low byte") et de la touche fonction
                               ("high byte").
Low( EXPR% )                -- isole le "low byte" d'un "integer".
Pos                         -- donne la position courante du tabulateur dans la ligne.
Random( M_RAMDOM% )         -- donne un nombre aléatoire:
        M_RAMDOM% = 0       -- redonne le nombre précédent.
                    1       -- donne un nouveau nombre.
Testbit( EXPR%,BIT% )       -- indique par "vrai" (-1) ou "faux" (0) l'état du bit BIT%
                               d'un "integer".
True                        -- donne la valeur -1, résultat "vrai" d'un test booléen.
Version                     -- donne la valeur décimale de la version du BASIC.
```

6.12 Ordres et fonctions souris (MOUSE)

Ordres

```
Mouse | Off                 -- retire la souris de PSos pour le BASIC.
      | On                  -- redonne la souris à PSos.
Mouse( X%,Y% )              -- positionne la souris sur l'écran.
```

Fonction

```
Mouse( M_MOUSE% )           -- lit l'état de la souris:
        M_MOUSE% = 0        -- détermine un nouvel état de la souris et donne la coor. X.
                   1        -- donne la coor. Y.
                   2        -- donne l'état du bouton de gauche.
                   3        -- donne l'état du bouton de droite.
                   4        -- donne l'état du bouton du centre.
                   5        -- indique si le nouvel état est vraiment un nouvel état.
```

6.13 Touches spéciales et softkeys

Ordres

```
Setbar EXPR%                -- allume les barres sous les softkeys. A chaque bit de
                               EXPR% corresoond une barre, de gauche à droite pour les
                               bits 0 à 15.
Setmenu H0$,L0$,H1$,L1$,H2$,L2$,H3$,L3$,H4$,L4$,H5$,L5$
                            -- affiche des nouveaux textes dans les softkeys.
                               Les "string" Hx$ sont affichés sur la ligne du haut, les
                               Lx$ sur la ligne du bas. Les "string" x0$ sont affichés
                               dans la softkey isolée à gauche, les autres de gauche à
                               droite dans les touches suivantes.
        -- Exemple: Setmenu '','de','','gauche','','à','','droite,','','en','','bas'
Setskey S'KEY$              -- programme les touches soéciales.
```

Exemple: Setskey Chr$(0)&'DFRC' -- dévie les 4 touches DFRC du traitement normal. Chr$(0)& conserve l'action habituel de la touche F0 (break).

Fonction

```
Skey( M_SKEY% )          -- donne l'état des touches spéciales:
        M_SKEY% = 0      -- donne les actions sur les touches spéciales.
                  1      -- donne l'état des touches spéciales.
```

Exemple: If Testbit(Skey(1),2) Then -- test si la 2ième touche spéciale est enfoncée (selon l'exemple ci-dessus dans Setkey).

7 NTREL

Ordres

```
Delms EXPR%              -- attend EXPR% * 20 ms.
Priority EXPR%           -- change la priorité d'exécution du BASIC.
Timeout EXPR%            -- change le temps d'attente des processus du BASIC.
```

Fonctions

```
Priority                 -- donne la priorité actuelle du BASIC.
Timeout                  -- donne le temps d'attente des processus du BASIC.
```

8 FICHIERS

Ordres

```
Close [File(CANAL%)]     -- ferme un fichier et libère le canal BASIC ou ferme tous
                            les fichiers.
Command File(CANAL%), TYPE_CANAL%, COMMAND$ -- envoie une commande au "driver".
        TYPE_CANAL% =
                TYPMM% = 1       -- mémoire de masse
                TYPKEY% = 2      -- clavier (ré-assignable)
                TYPDIS% = 3      -- écran (bit map complet)
                TYPWDO% = 4      -- fenêtre dans un écran
                TYPIO% = 5       -- périphérique genre "streamer"
                TYPPRI% = 6      -- imprimante
                TYPNETW% = 7     -- réseau local type SWAN
                TYPSTAT% = 8     -- lecture d'un statut uniquement
        COMMAND$                 -- "string" de commande, ex: '*('&Chr$(0)
                                    équivalent à *([0] dans le FILER.
Get File(CANAL%) ,VAR {,VAR}
                            -- assigne une suite de variables. Le fichier contient une
                               suite de données en représentation binaire des variables.
                               Une VAR% occupe 2 bytes, une VAR! occupe 4 bytes, une
                               VAR$ occupe un nombre de bytes égal à sa longueur.
                               Il est nécessaire de prévoir le nombre de caractères
                               d'une VAR$ avant de la lire par Get. Cette opération est
                               effectuée en donnant une valeur initiale à VAR$, le
                               contenu étant sans importance, c'est la longueur (le
                               nombre de caractères) qui est testée.
Input File(CANAL%) ,VAR {,VAR} -- assigne une suite de variables.
Input Line File(CANAL%) [,VAR$]
                            -- lit une ligne entière et assigne une variable
```

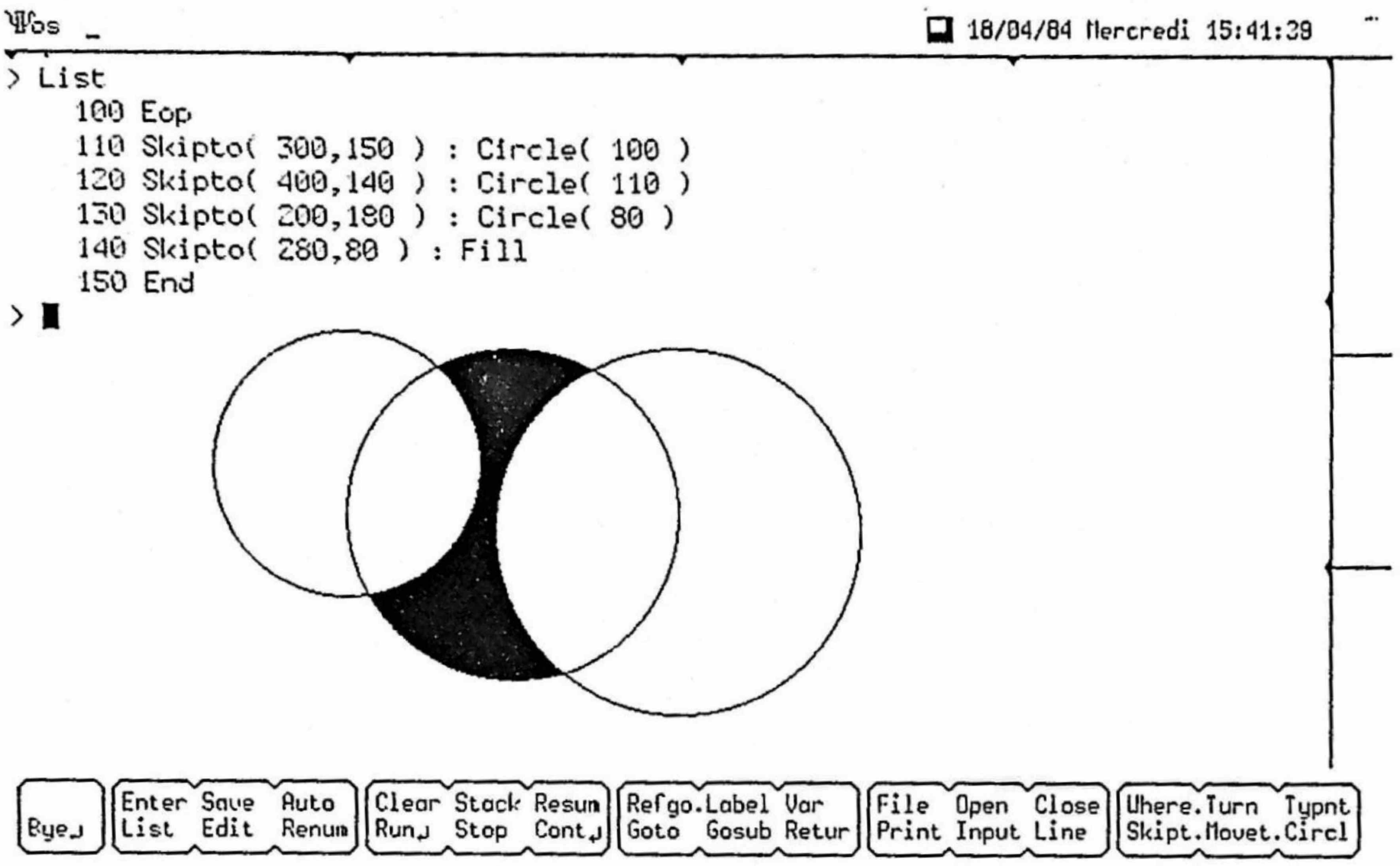
"string" ou absorbe simplement la fin d'une ligne.

```
Open File(CANAL%),FICHIER$,M_OPEN%{,LEN_BUFFER%}
                        -- ouvre un fichier en lecture ou écriture, en mode
                           bufferisé (par défaut:1024 bytes) ou interractif si
                           LEN_BUFFER%=0.
      M_OPEN%           -- mode d'ouverture:
              0 [Or OPEXCL%] [Or OPRD%] [Or OPWR%]
              OPEXCL% = 1      -- ouverture exclusive
              OPRD% = 2        -- ouverture en lecture
              OPWR% = 4        -- ouverture en écriture
   LEN_BUFFER% = 0 -- buffer longueur nulle: fichier interactif.
                 ) 0 -- fichier avec mémoire tampon.
Pos Eof(CANAL%)         -- positionne l'index de lecture&écriture à la fin du
                           fichier.
Pos File(CANAL%), POSITION!
                        -- positionne l'index de lecture&écriture dans le
                           fichier, saute POSITION! bytes depuis le début du
                           fichier.
Print File(CANAL%) [,]  -- écrit un caractère (RETURN) dans le fichier.
Print File(CANAL%), { [PRINT_EXPR] [PRINT_DELIMITEUR] }
                        -- écrit une suite d'expressions dans un fichier.
                           PRINT_EXPR et PRINT_DELIMITEUR sont décrits dans
                           l'ordre Print.
                           Carxy() et Caryx() sont retranchés des choix de
                           PRINT_EXPR.
Put File(CANAL%), EXPR {, EXPR}
                        -- écrit une suite d'expressions dans un fichier mais en
                           représentation binaire pour chaque expression.
```

Fonctions et "varsys"

```
Eof(CANAL%)             -- indique par "vrai" ou "faux" si l'index de lecture&
                           écriture est à la fin du fichier.
Eoln(CANAL%)            -- indique par "vrai" ou faux" si l'index de lecture
                           est devant un caractère (RETURN).
Input$(File(CANAL%), LEN%) -- lit LEN% caractères et donne ces caractères.
Pos Eof(CANAL%)         -- donne la longueur actuelle du fichier.
Pos File(CANAL%)        -- donne la position de l'index de lecture&écriture.
Rdstatus File(CANAL%), TYPE_CANAL%, LEN%, VAR$ -- lit le statut d'un "driver".
```

9 GRAPHISME



Ordres

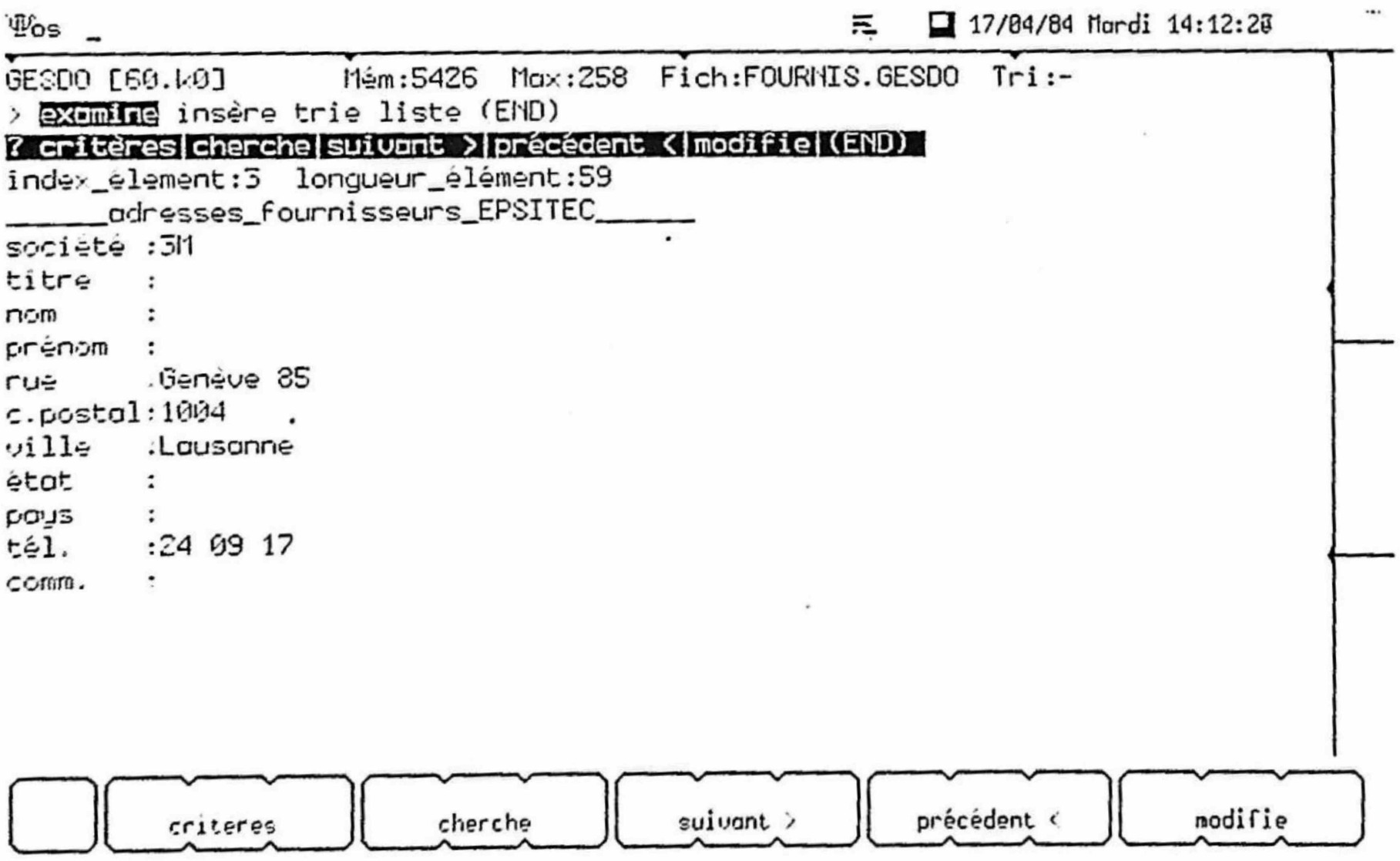
```
Cgraph                  -- copie les données graphiques courantes dans les données
                           graphiques auxiliaires.
Circle( RAYON% )        -- dessine un cercle.
Coor( X%,Y% )           -- positionne les coordonnées des axes, le point (0|0)
                           sur l'écran.
Cturn                   -- calcule la pente du dernier tracé en coor. cartésienne.
Cwindow                 -- efface la fenêtre graphique.
Eomove                  -- termine un tracé interrompu.
Fill                    -- remplit une surface délimitée par un contour fermé.
Igraph                  -- initialise tous les paramètres graphiques.
Move( LONGUEUR% )       -- dessine un tracé en coor. polaire.
Movesm6(X%,Y%,M_SM6%)   -- dessine un tracé compatible Smaky 6.
        M_SM6% = 0 à 7  -- trait compatible Smaky 6.
Moveto( X%,Y% )         -- trace en coor. cartésiennes absolues.
Movetor( DX%,DY% )      -- trace en coor. cartésiennes relatives.
Point( X%,Y% )          -- dessine un point en coor. cartésiennes absolues.
Pointr( DX%,DY% )       -- dessine un point en coor. cartésiennes relatives.
Rot( ALPHA! )           -- tourne les axes de dessin.
Scale( ECHELLE! ) .     -- détermine l'échelle des axes de dessin.
Sgraph                  -- permute les données graphiques courantes avec les
                           données graphiques auxiliaires.
Skip( LONGUEUR% )       -- saute en coor. polaires.
Skipto( X%,Y% )         -- saute en coor. cartésiennes absolues.
Skiptor( DX%,DY% )      -- saute en coor. cartésiennes relatives.
Turn( ALPHA! )          -- détermine la direction en coor. polaires.
Turnr( DALPHA! )        -- détermine la direction en coor. polaires relativement
                           à la précédente direction.
Typdash( T_DASH% )      -- détermine le type de trait d'un tracé.
        T_DASH% = 0     -- trait fin, sans aucune fuite, mode spécial.
                  1..4  -- trait fin, mode courant, épaisseur de 1 à 4 points.
                  5...  -- traits ......, - - -, -.-.-., + + + +, -..-..-,
                           programmables en 4 épaisseurs et 4 longueurs.
Typnt( T_PNT% )         -- détermine le type d'opération d'un point d'un tracé.
        T_PNT% = 0      -- pas d'opération sur le point.
                 1      -- allume le point.
                 2      -- éteint le point.
```

```
                    3       -- inverse le point.
                    4..     -- modes spéciaux
Window( XMIN%,YMIN%,XMAX%,YMAX% ) -- détermine les dimensions de la fenêtre du
                            -- graphisme dans l'écran du BASIC.
```

Fonctions et Varsys

```
Coor( M_COOR% )             -- donne les coordonnées de l'origine sur l'écran.
        M_COOR% = 0         -- donne l'abscisse.
                  1         -- donne l'ordonnée.
Cpoint                      -- donne le nombre de points affectés par le dernier tracé.
Eomove                      -- indique par "vrai" ou faux" si le tracé est complet
                               (mode de trait spécial).
Rot                         -- donne la pente des axes.
Scale                       -- donne l'échelle aopliquée sur les axes.
Testp( X%,Y% )              -- indicue par "vrai" (allumé) ou "faux" (éteint) l'état
                               d'un point de l'écran.
Testpr( DX%,DY% )           -- comme Testp mais en coordonnées cartésiennes relatives.
Turn                        -- donne la direction polaire.
Typdash                     -- donne la valeur du type de trait courant.
Typnt                       -- donne la valeur du type de point courant.
Wherepen( M_WHERE% )        -- donne des informations sur le dernier tracé.
        M_WHERE% = 0        -- donne la coordonnée courante graphique programmée x.
                   1        --    "   "     "          "        "          "    y.
                   2        --    "   "     "          "        "     sur l'écran x.
                   3        --    "   "     "          "        "      "      "   y.
Window( M_WINDOW% )         -- donne les dimensions de la fenêtre graphique.
        M_WINDOW% = 0       -- donne XMIN% courant
                    1       -- donne YMIN% courant
                    2       -- donne XMAX% courant
                    3       -- donne YMAX% courant
```

10 MODULE GESDO



Lorscu'on utilise les commandes GESDO, le module "GESDO.BIN" est d'abord chargé en mémoire central puis exécuté. Les appels GESDO permettent la gestion d'une base de donnée complexe. L'utilisation de ces appels depuis le BASIC est présenté dans le manuel GESTION.

Paramètres

```
CANAL_GESDO%            EXPR%    -- 1 (= CANAL_GESDO% (= 7
ORDRE_GESDO%            EXPR%    -- un numéro de commande GESDO, voir GESTION.
FONCTION_GSTION%        EXPR%    -- un numéro de fonction GESDO, voir GESTION.
PARAMETRE_GESDO         EXPR     -- selon la commande GESDO, voir GESTION.
```

Ordre

```
Gesdo(ORDRE_GESDO%,CANAL_GESDO%) {,PARAMETRE_GESDO}
                        -- exécute une routine GESDO.
```

Fonction

```
Gesdo(FONC_GESDO%,CANAL_GESDO%)
                        -- exécute une fonction GESDO.
```

11 EXPRESSIONS

```
EXPR                    [OPERATEUR_UNAIRE] FACTEUR [OPERATEUR FACTEUR]

FACTEUR                 | CONSTANTE              -- ex: 4 ou 12 ou "résultat="
                        | VARIABLE               -- ex: MACHINE ou TABLEAU()
                        | FONCTION|VARSYS        -- ex: Sin(EXPR%!) ou Coor(M_COOR%)
```

12 OPERATEURS

12.1 Opérateurs unaires

```
Not EXPR%               -- complément à 1 pour chaque bit
+|- EXPR%!              -- +|-, ex: Print - 4
```

12.2 Opérateurs logiques

```
EXPR% And EXPR%         -- et bit à bit
EXPR% Or EXPR%          -- ou bit à bit
EXPR% Xor EXPR%         -- ou-exclusif bit à bit
```

12.3 Opérateurs relationnels

```
EXPR ( EXPR             -- plus petit que
EXPR ) EXPR             -- plus grand que
EXPR = EXPR             -- égal à
EXPR (= EXPR            -- plus petit ou égal à
EXPR )= EXPR            -- plus grand ou égal à
EXPR () EXPR            -- différent de
```

12.4 Opérateurs arithmétiques

```
EXPR%! + EXPR%!         -- addition
EXPR%! - EXPR%!         -- soustraction
EXPR%! * EXPR%!         -- multiplication
EXPR%! / EXPR%!         -- division réelle
EXPR% Div EXPR%         -- division entière
EXPR% Mod EXPR%         -- reste de la division entière
EXPR%! **|^ EXPR%!      -- élévation à la puissance
```

12.5 Opérateur "string"

```
EXPR$ +|& EXPR$     .      -- concaténation de "string"
```

12.6 Parenthèses

```
( EXPR )                   -- évalue l'expression entre paranthèses.
```

12.7 Schéma de précédence

```
        (),+|- unaire              -- exécuté en premier
        **|^
        *,/,Div,Mod
        +,-
        <,>,=,<=,>=,<>
        Not
        And
        Or,Xor                     -- exécuté en dernier
```

13 CONSTANTES

```
CONST%                  | NB_ENTIER      -- ex: 123
                        | H'NB_ENTIER_HEXA -- ex: H'3E8 (vaut 1000)

CONST!                  | NB_REEL        -- ex: 10E10
                        | H'NB_REEL_HEXA -- ex: H'1H8 (vaut 256)

CONST$                  | 'STRING_ASCII' -- ex: 'une chaîne de car.'
                        | "STRING_ASCII" -- ex: "#mm1:DEMO"
        -- si le délimiteur apparait dans le "string", il faut le doubler comme dans
        -- le mot 'aujourd''hui'.

NB_ENTIER               [+|-] NOMBRE              -- -32768 <= NB_ENTIER <= 32767

NOMBRE                  CHIFFRE { CHIFFRE }

NB_ENTIER_HEXA          [+|-] NOMBRE_HEXA         -- -8000 <= NB_ENTIER_HEXA <= 7FFF

NOMBRE_HEXA             DIGIT_HEXA { DIGIT_HEXA }

DIGIT_HEXA              0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F -- base hexadécimale
                                  |a|b|c|d|e|f

NB_REEL                 [+|-] NOMBRE[.[NOMBRE]][E[+|-][NOMBRE]] -- env. 1E-20 .. 1E+20

NB_REEL_HEXA            [+|-] NOMBRE_HEXA[.[NOMBRE_HEXA]][H[+|-][NOMBRE_HEXA]]

STRING_ASCII            CAR_ASCII { CAR_ASCII }
```

14 VARIABLES

Les variables sont définies de 2 façons, pour 2 contextes totalement différents:
 - l'utilisation lors d'un calcul d'expression.
 - l'assignation.
Les variables supportent actuellement 3 types de données:
 - le nombre entier codé sur 16 bits, identifié par le signe %.
 - le nombre réel codé sur 32 bits, sans identificateur (! pour la description).
 - le "string" (long. max env. 32000 car.), identifié par le signe $.
Les variables sont construites pour 4 modes d'utilisation:
 - VAR_NOR: la "variable normale" à usage général.
 - VAR_DIM: la "variable tableau" pour des vecteurs, des matrices ou éq.
 - VAR_FN: la "variable fonction" pour la définition de fonctions particulières.
 - VAR_IND: la "variable indirecte" qui permet le passage de paramètres par adresse
       dans les routines du programme BASIC.

```
VAR_EXPR                VAR_NOR | VAR_DIM | VAR_FN | VAR_IND -- calcul d'expression
VAR                     VAR_NOR | VAR_DIM | VAR_IND -- assignation (Let,Input,...)

VAR_NOR                 VAR_NOR% | VAR_NOR! | VAR_NOR$  -- "variable normale"
VAR_NOR%                NOM%                            -- ex: VITESSE, ALTITUDE, I
VAR_NOR!                NOM
VAR_NOR$                NOM$

VAR_DIM                 VAR_DIM% | VAR_DIM! | VAR_DIM$  -- "variable tableau"
VAR_DIM%                NOM_DIM%(EXPR%{,EXPR%})         -- ex: MATRICE(1,2)
VAR_DIM!                NOM_DIM!(EXPR%{,EXPR%})
VAR_DIM$                NOM_DIM$(EXPR%{,EXPR%})
NOM_DIM%                NOM%
NOM_DIM!                NOM
NOM_DIM$                NOM$

VAR_FN                  VAR_FN% | VAR_FN! | VAR_FN$     -- "variable fonction"
VAR_FN%                 NOM_FN%[(PARAM_FN_CALL{,PARAM_FN_CALL})] -- ex: Fn_XX(4)
VAR_FN!                 NOM_FN![(PARAM_FN_CALL{,PARAM_FN_CALL})]
VAR_FN$                 NOM_FN$[(PARAM_FN_CALL{,PARAM_FN_CALL})]
NOM_FN%                 Fn_NOM%
NOM_FN!                 Fn_NOM
NOM_FN$                 Fn_NOM$

VAR_IND                 VAR_IND% | VAR_IND! | VAR_IND$  -- "variable indirecte"
VAR_IND%                @NOM%                           -- ex: @RESULTAT
VAR_IND!                @NOM
VAR_IND$                @NOM$
```

14.1 Paramètres d'une VAR_FN

```
PARAM_FN_DEF            VAR_NOR
        -- le corps de la définition d'une VAR_FN dans l'ordre Def Fn_... utilise
        -- une suite de VAR_NOR pour le passage des paramètres (par valeur).

PARAM_FN_CALL           EXPR
        -- utilisation de la VAR_FN avec ses paramètres.
```

14.2 Paramètres d'un LABEL

```
PARAM_LABEL_DEF         PARAM_LABEL_DEF_NOR | PARAM_LABEL_DEF_IND
        -- les paramètres utilisés dans le corps de la définition d'un LABEL sont
           de 2 types: normal (direct) ou indirect, pour les 2 types de passage
           de paramètres: par valeur ou par adresse.
```

```
PARAM_LABEL_DEF_NOR     VAR_NOR
        -- passage par valeur
PARAM_LABEL_DEF_IND     VAR_IND
        -- passage par adresse

PARAM_LABEL_CALL        PARAM_LABEL_CALL_NOR | PARAM_LABEL_CALL_IND
        -- les paramètres utilisés dans l'appel de la routine sont de 2 types:
           des valeurs ou des adresses.
PARAM_LABEL_CALL_NOR    EXPR
        -- passage par valeur.
PARAM_LABEL_CALL_IND    VAR_NOR|NOM_DIM(*)|NOM_FN|VAR_IND
        -- passage par adresse d'une variable normale, d'un tableau, d'une fonction ou
           d'une autre variable indirecte.
```

15 EXEMPLES BASIC

15.1 Conjugaison d'un verbe

L'exercice consiste à conjuguer un verbe régulier à terminaison -er. Nous nous contenterons de conjuguer la 1ère personne du singulier et la 1ère personne du pluriel du temps présent.

Voici le listing:

```
100 Input 'verbe',V$
110 Print 'je ' + Left$( V$,Len(V$)-1 )
120 Print 'nous ' + Left$( V$,Len(V$)-2 ) + 'ons'
130 End
```

Les opérations effectuées dans chaque ligne sont:

100) demande un verbe (la forme infinitive) à l'opérateur.

110) affiche 'je ' puis la conjugaison qui est construite en retirant la dernière lettre de la forme infinitive du verbe, soit la lettre r.

120) affiche 'nous ' puis la conjugaison qui est construite en retirant les deux dernières lettres de la forme infinitive du verbe et en accolant la terminaison 'ons'.

130) termine le programme.

Ce programme fonctionne correctement mais il a un défaut: il est "linéaire", c'est-à-dire qu'il commence en ligne 100 et se termine en ligne 130. Cela n'est pas critique pour un petit programme mais devient inadmissible pour des grands programmes (programme "spaghetti"). Une meilleure approche est la décomposition du problème en plusieurs petits problèmes.

Voici le nouveau listing:

```
100 Input 'verbe',V$
110 Gosub Label_CONJUGUE( V$ )
120 End
130 Def Fn_SD$( I$ )=Butright$( I$,1 )
140 Label_CONJUGUE( VERBE$ )
150 Print 'je ' & Fn_SD$( VERBE$ )
160 Print 'nous ' & Fn_SD$( Fn_SD$( VERBE$ ) ) & 'ons'
170 Return
```

Les points intéressants sont:

a) le programme "principal" tient dans les lignes 100 à 120. Il demande à l'opérateur un verbe puis appelle une routine en donnant le verbe comme paramètre.

b) la définition d'une fonction Fn_SD$ qui retourne les caractères de son

propre paramètre d'entrée sauf le dernier caractère.

c) la définition d'une routine de conjugaison dans les lignes 140 à 170. La routine reçoit un paramètre, le verbe à conjuguer. Ce type de passage de paramètre est dit "par valeur". La variable VERBE$ qui reçoit le verbe à conjuguer, est d'abord souvé dans le "stack". Le contenu original de VERBE$ avant l'appel de la routine de conjugaison, est retrouvé lorsque la routine est terminée par l'instruction Return.

d) il est maintenant possible d'utiliser la routine de conjugaison en mode directe, c'est-à-dire en tapant directement au clavier par exemple:

```
Gosub Label_CONJUGUE( 'cirer' )
```

e) le symbole & est identique au symbole + dans les expressions de chaînes de caractères ("string"), tous les deux représentent la concaténation de "string".

15.2 Dessin d'un carré et d'une maison

Il y a 2 modes distincts de dessins:
- par des déplacements polaires (1 angle et 1 distance).
- par des déplacements cartésiens (1 abscisse et 1 ordonnée).

Le dessin d'un carré est un motif régulier et répétitif. Il est avantageux d'utiliser le mode polaire. Les angles sont donnés en radian (180 degrés corresoond à PI radian, PI=3.14159...).

Voici le listing:

```
100 Skipto( 100,100 )
110 Turn( Pi/2 )
120 For I=1 To 4
130   Move( 100 ) : Turnr( -Pi/2 )
140 Next
150 End
```

Les opérations effectuées dans chaque ligne sont:

100) positionne l'angle inférieure gauche du carré.
110) détermine la direction polaire origine.
120) répète 4 x la ligne 130 (et la ligne 140 également).
130) motif élémentaire d'un côté du carré. Noter l'emploi de l'ordre Turnr qui permet de tourner relativement à la dernière direction connue et qui deviendra la nouvelle direction.
140) fin de la boucle de répétion.
150) fin du programme.

Le dessin stylisé d'une maison est irrégulier. Il est plus commode d'utiliser le mode cartésien.

Voici le listing:

```
100 Skipto( 100,100 )
110 Moveto( 200,100 )
120 Moveto( 200,200 )
130 Moveto( 150,250 )
140 Moveto( 100,200 )
150 Moveto( 100,100 )
160 End
```

L'ordre Skipto (traduction: saute vers) déplace la plume sans trace.
L'ordre Moveto (traduction: avance vers) déplace la plume avec une trace.

15.3 Dessin d'un polygone

Ce dessin utilise le mode de déplacement polaire comme le dessin du carré mais une petite fantaisie est ajoutée: on incrémente (ou décrémente) la longueur du côté après chaque dessin d'un côté.

Voici le listing:

```
100 Skipto( 350,200 )
110 Turn( Pi/2 )
120 C=10,I=1,A=45*Pi/180,N=100
130 For J=1 To N
140    Move( C )
150    C=C+I
160    Turnr( A )
170 Next
180 End
```

Les variables déclarées dans la ligne 120 sont:

C	la longueur du côté initial.
I	l'incrément du côté.
A	l'angle de rotation du prochain côté.
N	le nombre de côté à dessiner.

Des dessins agréables sont produits en variant les paramètres d'assignation de la ligne 120 (utiliser l'ordre Edit):

C=	I=	A=	N=	commentaire
100	0	90*PI/180	4	dessine un carré
100	0	72*PI/180	5	dessine un pentagone
10	5	144*PI/180	100	joli dessin
10	5	120*PI/180	100	spirale triangulaire
10	5	124*pi/180	100	joli dessin

15.4 Lecture d'un fichier

Ce programme lit un fichier BASIC et l'affiche sur l'écran.

Voici le listing:

```
100 Inout 'quel fichier BASIC voulez-vous voir', F$
110 F$=F$+'.BAS'
120 Open File(1),F$,2
130 If Eof(1) Then 170
140 Input Line File(1), A$
150 Print A$
160 Goto 130
170 Close File(1)
180 End
```

Les opérations effectuées dans chaque ligne sont:

100) le programme demande à l'opérateur le nom du fichier BASIC à afficher.
110) l'extension '.BAS' est accolée au nom du fichier BASIC.
120) le fichier est ouvert en lecture. Une erreur sera produite si quelque chose ne fonctionne pas (fichier inconnu,...).
130) l'indication "fin de fichier atteinte" (la fonction Eof) est valide dès l'ouverture du fichier. Il est impératif de tester que le fichier contient encore quelque chose avant ce commander une lecture dans le fichier (contrairement à l'usage des fichiers BASIC du Smaky6). Cette facon d'utiliser les fichiers est compatible avec le language PASCAL.
140) une ligne entière est lue.

150) la ligne est affichée.
160) boucle pour la lecture de la ligne suivante.
170) le fichier est fermé.
180) le programme est terminé.

Ce programme fonctionne correctement. Il comporte un boucle, la boucle de lecture de chaque ligne du fichier et de son affichage. Cette boucle pourrait être définie de la manière suivante:
tant que le fichier n'est pas vide, lit une ligne et l'affiche puis recommence.

La traduction en BASIC est: While Not Eof() ... End While.

La boucle While EXPR% ... End While exprime parfaitement ce que les lignes 130, 160 et 170 (sortie de la boucle) réalisent.

Voici le nouveau listing:

```
100 Input 'quel fichier BASIC voulez-vous voir', F$
110 F$=F$+'.BAS'
120 Open File(1),F$,2
130 While Not Eof(1)
140   Input Line File(1), A$
150   Print A$
160 End While
170 Close File(1)
180 End
```

15.5 Impression d'un fichier

L'imprimante a un nom logique: #PRINTER. On utilise ce nom logique comme un fichier.

Voici le listing:

```
100 Open File(1),'#PRINTER',4+1
110 Input 'quel fichier voulez-vous imprimer',F$
120 Open File(2),F$,2
130 If Eof(2) Then 170
140 Input Line File(2),L$
150 Print File(1),L$
160 Goto 130
170 Close File(2)
180 Close File(1)
190 End
```

Les opérations effectuées dans chaque ligne sont:
100) l'imprimante est "ouverte" en mode écriture exclusive.
110) le programme demande à l'opérateur le nom du fichier à imprimer.
120) le fichier à imprimer est "ouvert" en mode lecture non-exclusive.
130) Eof teste si le fichier à imprimer est vide ou entièrement lu.
140) une ligne entière est lue.
150) la ligne est imprimée.
160) boucle pour la ligne suivante à imprimer.
170) ferme le fichier à imprimer.
180) ferme l'imprimante.
190) termine le programme.

La boucle de lecture et d'écriture peut également être construite avec la structure While EXPR% ... End While, comme pour l'exemple précédent.