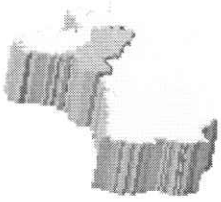
IREM

INSTITUT DE RECHERCHE SUR L'ENSEIGNEMENT DES MATHEMATIQUES
DES PAYS DE LA LOIRE



NANTA IREMICA N° 5

Le langage basic

M. Belhache

1977

INTRODUCTION

A LA PROGRAMMATION

PAR LE BASIC SOUS I.T.F.

M. BELHACHE

NANTA IREMICA

Les volumes de la collection Nanta Iremica sont écrits pour les Enseignants à partir d'un travail critiqué et testé par des équipes organisées dans le cadre de l'Institut de Recherche sur l'Enseignement des Mathématiques de l'Académie de Nantes.

Qu'on ne se méprenne surtout pas en imaginant que cette phase critique fasse des ouvrages de Nanta Iremica des textes de référence définitifs. La critique collective, si elle élimine des erreurs, si elle organise plus rationnellement le plan d'un texte, si elle l'adapte mieux aux besoins des lecteurs en gommant les lubies et manies de l'auteur, ne saurait fournir un brevet de valeur assurée. Que le lecteur garde tout son esprit critique, l'aiguise même. Les textes de Nanta Iremica n'ont aucun caractère contraignant et ne sont imposés par aucune hiérarchie. Tels qu'ils sont, ils représentent une étape, certes figée par le texte, de l'activité pédagogique et mathématique d'une Académie. Mais les modes, les goûts et les connaissances évoluent et les étapes se succèdent.

Dans cette collection, il y a place pour des textes d'information mathématique, place pour des recueils commentés de problèmes et place pour des progressions analysées de telle ou telle classe de l'Enseignement, en particulier l'enseignement dans les classes littéraires ou les classes professionnelles, défavorisées du point de vue mathématique.

Il y a aussi place pour des documents qui ne font pas partie de la panoplie usuelle des gadgets du mathématicien moyen. Par exemple des documents d'épistémologie mathématique, par exemple des documents de docimologie, par exemple des synthèses sur les différentes réformes des mathématiques dans le monde, par exemple puisque tout individu a échoué, échoue ou échouera sur un problème mathématique, des réflexions sur l'échec en mathématiques.

Le présent ouvrage déroge à la règle générale de la collection en ce sens qu'il a d'abord été écrit pour des étudiants de Faculté. Cependant, il a été utilisé par des stagiaires à l'I.R.E.M. de Nantes en 1975-1976. Le rôle de ce texte, dans le cadre de l'I.R.E.M., en dehors de la formation apportée aux lecteurs, est de susciter chez les Enseignants la volonté d'utiliser l'informatique dans leurs cours. Utilisation des techniques de calcul bien sûr, mais surtout imprégnation d'un certain état d'esprit. A plus lointaine échéance, l'I.R.E.M. de Nantes souhaite publier un texte produit par des Enseignants du secondaire. Ce sera l'héritier du présent ouvrage, pour lequel nous remercions vivement M. BELHACHE, animateur à l'I.R.E.M. de Nantes et maitre assistant à l'U.E.R. de Mathématiques de l'Université de Nantes.

Monsieur DHOMBRES
Directeur de l'I.R.E.M.
de NANTES

AUTRES PUBLICATIONS DE LA COLLECTION

NANTA - IREMICA

(Parues ou à paraître)

Volume N° 1	Introduction à la logique	MM. DURAND, VAN DEN BOSSCHE
Volume N° 2	Introduction à la Théorie des Ensembles	MM. DURAND, VAN DEN BOSSCHE
Volume N° 3	Etude épistémologique et historique de la notion de nombre réel et de mesure des grandeurs	Jean DHOMBRES
Volume N° 4	Documents relatifs au Volume N° 3	Jean DHOMBRES
Volume N° 5	Le langage BASIC	M. BELHACHE
Volume N° 6	Echec en Mathématiques	A. BIGARD
Volume N° 7	Eléments d'Analyse Fonctionnelle	Jean DHOMBRES
Volume N° 8	Algèbre linéaire et Géométrie vectorielle	R. SEROUX
Volume N° 9	Méthode Mathématiques Modernes utilisées en Théorie de l'approximation	Jean DHOMBRES
Volume N° 10	Analyse	Melle VENARD Jean DHOMBRES
Volume N° 11	Le langage PL/I	M. BELHACHE

Pour se procurer ces livres s'adresser à :

I.R.E.M de NANTES

Université de Nantes

38, boulevard Michelet

BP 1044 44037 NANTES-CEDEX

Ce cours polycopié est principalement destiné à faciliter l'initiation à la programmation par l'étude du BASIC sous I.T.F.

Le langage BASIC, dans sa version sous I.T.F., a été choisi comme outil d'introduction à la programmation en raison de sa grande simplicité. Il est conçu pour le traitement des données numériques. Ses entrées-sorties ne demandent qu'une programmation sommaire. Et malgré une insuffisance certaine dans la conception et l'utilisation des sous-programmes, le BASIC se révèle bien adapté à la résolution de calculs scientifiques ou techniques, en particulier en raison des facilités qu'offrent ses instructions matricielles. Enfin, le langage est disponible en mode conversationnel. Cette facilité de dialogue se révèle très intéressante car, à ce stade de leur formation, les étudiants ont besoin de rester en contact direct avec l'ordinateur.

Cet ouvrage n'est donc ni un manuel de référence ni même un cours au sens habituel du terme. Il doit plutôt être considéré comme un "support de cours" destiné à faciliter l'enseignement et l'étude du langage. Le plus souvent, on se contente d'indiquer sans explication approfondie les notions du langage et les règles principales qui s'y rattachent en les illustrant d'exemples et de contre-exemples extraits de programmes effectivement réalisés. Les détails sont exposés dans le cadre de l'enseignement traditionnel ou bien découverts par l'étudiant lui-même au cours des séances de travaux dirigés de programmation sur terminal.

La disposition matérielle reflète également le souci de faciliter le travail de l'étudiant. Le texte imprimé occupe à peine les deux tiers de la surface de la page. On dispose ainsi de la place nécessaire pour consigner ses notes personnelles, adaptées à sa propre compréhension.

Dans l'organisation de ce cours on a cherché à mettre immédiatement entre les mains de l'étudiant les premiers outils suffisants pour lui permettre d'écrire ses propres programmes, dès le début de l'enseignement. La programmation, en effet, s'apprend par l'exemple et la pratique.

On commence donc par un chapitre d'introduction à la programmation en BASIC, dans lequel on étudie sur un programme très simple les quelques instructions et commandes I.T.F. fondamentales. Il est alors possible d'écrire et de mettre au point des programmes élémentaires de quelques instructions. L'étude détaillée du traitement des données numériques, pratiquement les seules manipulables en BASIC, fait l'objet du chapitre 2. On y a inclus les fonctions incorporées considérées comme de simples extensions des opérateurs arithmétiques. La lecture par INPUT, l'empression par PRINT et les deux principales instructions de débranchement GOTO et IF font l'objet du 3ème chapitre. Au chapitre suivant, on complète l'ensemble des éléments de données traitées par BASIC en étudiant les tableaux numériques et les données littérales. Il devient alors nécessaire de disposer d'autres instructions telles que FOR/NEXT ; c'est le rôle du chapitre 5. Initialement, les instructions d'entrées-sorties avaient été limitées aux formes les plus simples. Le chapitre 6 traite des autres possibilités du BASIC : lecture par READ/DATA et impression avec image PRINT USING. Ces six premiers chapitres constituent donc ce que l'on pourrait appeler le "BASIC élémentaire".

Dans les trois chapitres suivants, on trouvera l'étude des autres possibilités du langage. Elles n'ont pas toutes le même intérêt selon la spécialisation de l'étudiant. Les fonctions-utilisateurs et les sous-programmes sont traités au chapitre 7 ; les instructions matricielles font l'objet du chapitre 8 et les quelques possibilités offertes par les fichiers du BASIC sont regroupées au chapitre 9.

Enfin, on a rassemblé dans un dernier chapitre l'étude complète des commandes de l'I.T.F. Elles ne font pas partie du langage de programmation, mais leur bonne connaissance et leur bon emploi sont néanmoins indispensables à la mise en oeuvre des programmes. Une part importante de ce chapitre a été consacrée aux problèmes de mise au point et à l'utilisation du sous-mode TEST qui est en effet un outil efficace de mise au point et il importe que l'étudiant apprenne très tôt à s'en servir.

Pour conclure, qu'il nous soit permis de suggérer aux étudiants de prendre la peine d'étudier les explications et les exemples de ce polycopié AVANT de prendre part à l'enseignement traditionnel ; ce dernier n'en sera que plus fructueux.

NANTES - janvier 1976

M. BELHACHE

SOMMAIRE

CHAPITRE 1
INTRODUCTION A LA PROGRAMMATION

Section 1.1
LA PROGRAMMATION

§ 1.1.1 I.T.F. et BASIC

1. Le traitement à faire exécuter sur les données de base constitue le programme : ensemble organisé d'instructions écrites en Basic.

2. De plus, on peut communiquer avec l'ordinateur par l'intermédiaire des commandes du système I.T.F.

3. La distinction est fondamentale : une instruction basic est toujours précédée d'un numéro qui l'identifie.

§ 1.1.2 Notion de programme

1. Le traitement à réaliser doit avoir été préalablement défini lors de l'Analyse du problème.

2. Il est ensuite décomposé en fonctions élémentaires qui sont alors traduites dans le langage de programmation.
   Les instructions sont construites à partir du vocabulaire du langage conformément aux règles de sa syntaxe.

§ 1.1.3 Entrées et sorties

1. Entrée ou lecture
   Opérations consistant à transmettre au programme lorsqu'il s'exécute en mémoire interne, des informations qui lui sont externes.

2. Sortie ou écriture

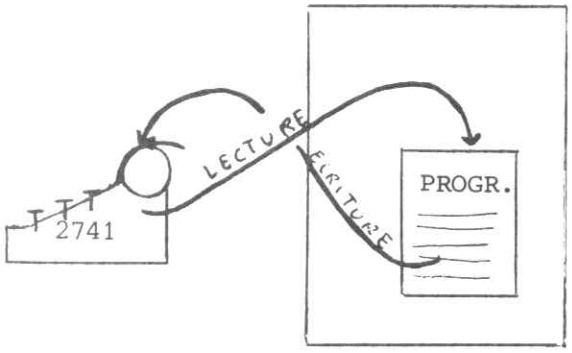
Opérations consistant à transmettre des résultats élaborés par le programme lorsqu'il s'exécute en mémoire interne à destination d'un utilisateur externe.

3. Conversions lors des transmissions

- Extérieurement, l'information s'exprime au moyen de caractères naturellement reconnus : lettres, chiffres...
- Intérieurement, dans la mémoire centrale, elle se résoud finalement en un assemblage de bits ou éléments d'information binaire dont l'organisation et l'interprétation dépendent du type de l'information.

4. Terminal 2741

Machine à écrire, connectée à l'ordinateur. Son clavier est l'intermédiaire obligé de toute transmission entre l'utilisateur et le programme qui s'exécute en mémoire.



§ 1.1.4 MISE EN OEUVRE D'UN PROGRAMME

1. Ecriture du programme

- Les commandes I.T.F. d'introduction exécutées, les instructions sont frappées une par une, et transmises à l'ordinateur.

- Dès que l'instruction est transmise, sa syntaxe est contrôlée :
  - si elle est convenable, l'instruction est stockée ;
  - sinon, un message est imprimé ; il faut réécrire l'instruction.

2. Compilation
   - Le programme écrit en Basic doit être traduit en langage machine.
   - Cette compilation exige au préalable un contrôle de la structure du programme ; de nouvelles erreurs peuvent donc être décelées et un message imprimé. Les instructions erronées devront être réécrites.

3. Exécution
   Une compilation sans erreur permettra l'exécution du programme. Néanmoins d'autres erreurs pourront être révélées :
   => La mise au point d'un programme pourra être délicate et exiger de nombreux essais.

Section 1.2
EXEMPLE DE PROGRAMME BASIC

§ 1.2.1 L'exemple

1. Le sujet du problème
   Calculer et imprimer la valeur, A, prise par un capital C, placé à intérêts composés au taux annuel I % pendant N années : $A = C(1 + I/100)^N$.

2. Ouverture de session

```
① logon blh1
  THANK YOU.          DATE 24/11/75       TIME 10.20.46
  READY edit v11 bqsic
  0170       INV SYN
  READY ?
  0170       ELEMENTS OF COMMAND INVALID OR IN IMPROPER SEQUENCE

② READY edit v11 basic
```

3. Frappe du programme

```
  EDIT  100 rem ** Calcul d'un Capital **
  EDIT  105 rem **        v.11          **
  EDIT  200    imput
                nput c
  EDIT  205      if c<=0 then 998
  EDIT  210    input i,n
  EDIT  220      a = c*(1+i/100)**n
  EDIT  230    print 'le capital deci
                                  vient', a ; "francs"
  EDIT  240    goto 300
  EDIT  998 stop v.11
  EDIT  999 end
③ EDIT  save
```

4. Résultats de compilation ; correction

```
④ EDIT  run

  0661       UNDEF STM NUM
  EDIT  ?
  0661       STATEMENT NUMBER 00000300 REFERENCED IN A STATEMENT
                                       NOT DEFINED IN PROGRAM
  EDIT  list 240
  00240  GOTO 300
  EDIT  240    goto 200
  EDIT  save
```

5. Liste du programme

```
00100 REM ** CALCUL D'UN CAPITAL **
00105 REM **         V.11          **
00200     INPUT C
00205       IF C<=0 THEN 998
00210     INPUT I,N
00220       A = C*(1+I/100)**N
00230     PRINT 'LE CAPITAL DEVIENT', A ; "FRANCS"
00240     GOTO 200
00998 STOP V.11
00999 END
```

6. Résultats d'exécution simple précision

```
EDIT  run

  ?   1000
  ?   5,10
LE CAPITAL DEVIENT                    1628.882    FRANCS
  ?   1000,5,15
EXCESS1000
  ?   5
TOOFEW5, 1  5
LE CAPITAL DEVIENT                    2078.905    FRANCS
  ?   -1
EDIT
```

7. Résultats d'exécution en double précision

```
EDIT  run lprec

  ?   1000
  ?   5, 10
LE CAPITAL DEVIENT                    1628.89462677744 FRANCS
  ?   1000
  ?   5,15
LE CAPITAL DEVIENT                    2078.92817941136 FRANCS
  ?   0
EDIT
```

8. Clôture de session

```
⑤ EDIT  end
⑥ READY logoff
        LOGGED OFF AT 10.28.10
```

§ 1.2.2 Analyse de commandes

1. Logon

   Ouverture de la session, mise en liaison avec l'ordinateur et identification de l'utilisateur.

2. Edit nom basic

   nom : 3 caractères au plus
   1 lettre (+ 1 ou 2 lettres ou chiffres)

   Précise le nom du programme et le langage utilisé.
   Permet de rappeler un programme stocké en bibliothèque.

3. Save

   Recopie et sauvegarde le programme sous le nom précisé à l'edit.

4. Run

   Demande la compilation suivie, si possible, de l'exécution.

5. End

   Marque la fin du mode EDIT ; le programme est effacé.
   Ne pas confondre avec l'instruction end :
   commande end : EDIT end
   instruction end : EDIT n° end

6. Logoff

   Fin de session ; rupture de la liaison.

§ 1.2.3 Analyse des instructions

1. N°s 100 et 105 : REM

   Introduisent des lignes d'explications dans le programme. Sans effet sur son déroulement.

2. N°s 200 et 210 : lectures de C, I, N

   La lecture se fait par le terminal. A ce moment un point d'interrogation apparaît au début de la ligne.

   => Frapper une valeur, dans l'ordre, pour chacune des variables de l'INPUT concerné.

3. N° 220 : Calcul de A

   Dans une instruction d'affectation :

   A = C * (1 + I/100) ** N

   variable = expression arithmétique.

4. N° 230 : écriture de la valeur de A

   - La valeur de A est imprimée au terminal.
   - Les constantes littérales qui l'entourent sont imprimées telles qu'elles apparaissent dans l'instruction.
   - Ces valeurs sont séparées
     - par une virgule
     - ou par un point-virgule.

5. N° 240 : débranchement

   Après calcul et impression de A , on revient aux lectures de C, I, N pour un nouveau calcul. Un test sur C permet d'interrompre le déroulement du programme et de se débrancher en 998.

6. N° 998 et 999 : arrêt et fin du programme

   Stop : arrêt de l'exécution du programme.

   End : limite du programme et arrêt de son exécution.

§ 1.2.4 Analyse des résultats

1. Premier run

   Erreur dans la destination du GOTO ; il faut réécrire la ligne 240 .

2. Deuxième run

   Les résultats numériques ont 7 chiffres.

3. Troisième run

   run lprec : longue précision

   15 chiffres et meilleure précision.

§ 1.2.5 Trace du programme

En mode TEST, la commande "trace" permet principalement de connaître :

- les changements de valeur des variables,
- les n°s des instructions de débranchement.

```
      run test
TEST  trace
TEST  go
          00200
  ?   1000
00200     C=+1.000000E+03
  ?   5,10
00210     I=+5.000000E+00
00210     N=+1.000000E+01
00220     A=+1.628882E+03
LE CAPITAL DEVIENT                    1628.882    FRANCS
          00200
  ?   1000
00200     C=+1.000000E+03
  ?   5,15
00210     I=+5.000000E+00
00210     N=+1.500000E+01
00220     A=+2.078905E+03
LE CAPITAL DEVIENT                    2078.905    FRANCS
          00200
  ?   0
00200     C=+0.000000E-01
          00998
EDIT
```

Section 1.3
ELEMENTS DE BASE DU LANGAGE BASIC

§ 1.3.1 Alphabet

1. 29 caractères alphabétiques

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

Majuscules et minuscules ont même valeur.

2. 10 chiffres

0 1 2 3 4 5 6 7 8 9

Ne pas confondre : O, o et 0

L, l, i, I et 1

3. 24 caractères spéciaux

- + * / & = ( ) ' " . , ; : ? ! | < > ¬ ♭ _ % ¢

♭ signifie espacement.
Certains caractères symbolisent un opérateur, d'autres ne possèdent que leur propre valeur.

§ 1.3.2 Vocabulaire

1. Vocabulaire réservé

Mots-clés du langage et opérateurs ayant une signification déterminée dans le langage :

⇒ leur orthographe et leur utilisation doivent être strictement respectées.

2. Vocabulaire de l'utilisateur

Mots construits par le programmeur afin d'exprimer (constante) ou de symboliser (variable) des éléments d'information.

3. Symboles de ponctuation

. sépare la partie entière de la partie fractionnaire d'une constante numérique.

, sépare les éléments d'une suite de constantes ou de variables.

( ) mise en facteur.

' apostrophe
" guillemet
} par paire, délimitent une constante littérale.

§ 1.3.3 Instruction

1. Rôle

Exprime - une action à entreprendre,
- ou bien des caractéristiques de données.

Construite autour de mots-clés ou du symbole d'affectation.

2. Forme et écriture

n° ␣ ␣ ... ␣ Instruction

n° est un numéro de 1 à 5 chiffres, obligatoire et séparé de l'instruction par un blanc au moins.
Instruction doit être écrite sur une seule ligne (120 positions). On ne peut avoir qu'une seule instruction par ligne.

§ 1.3.4 Rôle des blancs

En général, ils ne sont pas obligatoires.
Facilitent la lecture en séparant les mots de l'instruction.

§ 1.3.5 Instruction REM

1. Rôle et forme

Permet d'introduire des lignes d'explications dans le programme.

n° ␣␣ ... ␣ REM [explication]

2. Règles principales

- n° obligatoire
- explication : quelconque et facultative
- des instructions REM peuvent être placées n'importe où dans le programme.

§ 1.3.6 Le programme

1. Classes d'instructions

Non-exécutable comme REM, sans influence sur le déroulement.

Exécutable : traduite en langage machine.

2. Structure du programme

- Au moins une instruction exécutable.
- La dernière doit être n° end qui marque la limite et l'arrêt du programme.

3. Numérotation des instructions

Chaque numéro obligatoire a un double rôle :

- détermine la séquence d'exécution des instructions qui se fait selon la séquence croissante des numéros, sauf cas de débranchement programmé.
- identifie des instructions auxquelles renvoient des instructions de débranchement.

4. Nom du programme

Doit être donné à l'initialisation du mode edit :

READY edit nom basic

Nom : 3 caractères au plus :

- l'initiale : 1 lettre obligatoire,
- plus 1 ou 2 autres caractères alphanumériques facultatifs.

CHAPITRE 2
TRAITEMENT DES DONNEES NUMERIQUES

Section 2.1
ELEMENTS DE DONNEES NUMERIQUES

§ 2.1.1 Classes de données numériques

1. Constante numérique

   Elément d'information de valeur numérique fixe ; son écriture exprime immédiatement ses caractéristiques et sa valeur.

2. Variable numérique

   - Nom symbolique représentant une information numérique dont la valeur, en principe, varie au long du programme.

   - Au nom est associé un emplacement mémoire dont le contenu est la valeur de la variable .

     ⇒ le nom identifie le contenu de cet emplacement.

§ 2.1.2 Constantes numériques

1. Forme générale

   - Des chiffres 0 à 9 .
   - Eventuellement
     - un signe (+ ou -)
     - un point (.) entre partie entière et fractionnaire
     - une puissance de 10 à la suite de E.

2. Constante entière

   Une valeur de ℤ, avec 7 chiffres au plus

   12 -345 +6732

3. Constante en format F

Sous forme décimale, avec 7 chiffres au plus et un point décimal.

12. -34.5 +Ø.Ø67

4. Constante en format E

Sous forme mantisse-exposant.

mantisse : constante entière ou en format F formée des chiffres significatifs.

exposant : E±××

×× exprime la puissance de 10

12E+Ø -Ø.Ø345E+3 +67ØØØE-6

5. Constantes figuratives

π est représenté par &PI et vaut 3.141593

e est représenté par &E et vaut 2.718282

$\sqrt{2}$ est représenté par &SQR2 et vaut 1.414214

§ 2.1.3 Variables numériques

1. Identificateur, nom d'une variable

Soit une lettre : A, B, ..., #

Soit une lettre et un chiffre : AØ, ..., # 9

2. Valorisation d'une variable

- Par une instruction
  - d'affectation
  - ou de lecture
- Une fois la valorisation réalisée, la valeur est disponible par l'intermédiaire du nom jusqu'à ce qu'une nouvelle valorisation remplace l'ancienne valeur qui est alors perdue.
- Tant qu'une variable n'a pas été valorisée, elle est inutilisable.

§ 2.1.4 Forme interne des valeurs numériques

1. Représentation interne

   En virgule flottante binaire

   => la partie fractionnaire d'un nombre réel sera rarement représentée avec exactitude.

2. Simple, double précision

   Pour faire les calculs avec plus de 7 chiffres, utiliser la commande run lprec .

   Toutes les variables numériques du programme ont alors 16 chiffres.

3. Exemples

```
00010 REM ** PRECISION DES CALCULS **
00020 REM **         V.23           **
00030      A = 2/7
00040      B = 3/17
00050      C = &PI
00060      D = 22/7
00070   PRINT A;B
00080   PRINT C;D;D-C
00090 STOP
00100 END
EDIT
```

```
EDIT  run

 .2857143     .1764706
 3.141593     3.142857     1.263618E-03

EDIT  run lprec

 .285714285714286   .176470588235294
 3.14159265358979   3.14285714285714   1.2644892673E-03
```

```
00010 REM **     APPROXIMATION  INTERNE    **
00020 REM **  DE LE PARTIE FRACTIONNAIRE **
00030 REM **              V.24             **
00040       H = 0.2
00050       K = 0
00060       PRINT K;H
00070 REM ---------------------------------
00080   I=1
00090   K=K+H
00100   PRINT K
00110 IFI>=5THEN150
00120   I=I+1
00130 GOTO90
00140 REM
00150 IFK=1THEN190
00160      PRINT K;'N''EST PAS EGAL A 1'
00170 GOTO200
00180 REM
00190      PRINT K;'VAUT 1'
00200 STOP
00210 END
```

```
EDIT  run
 0      .2000000
 .2000000
 .4000000
 .6000000
 .8000000
 .9999999
 .9999999    N'EST PAS EGAL A 1

EDIT  run lprec
 0      .200000000000000
 .200000000000000
 .400000000000000
 .600000000000000
 .800000000000000
 1
 1     N'EST PAS EGAL A 1

EDIT
```

4. Limitations

$$0.54\ E-78 \leqslant \begin{matrix}\text{valeur}\\ \text{absolue}\end{matrix} \leqslant 0.72\ E+76$$

Underflow si $<$ 0.54 E - 78 , remise à zéro et poursuite du calcul.

Overflow si $>$ 0.72 E + 76 , arrêt du programme.

```
00010 REM ** LIMITE PAR OVERFLOW **
00020 REM **          V.29          **
00030     INPUT A
00040     I=1
00050      PRINT A*I
00060      I = I+1
00070 IFI>=8THEN90
00080 GOTO50
00090 END
EDIT

      run lprec

  ?   1.2e+75

 1.2000000000E+75
 2.4000000000E+75
 3.6000000000E+75
 4.8000000000E+75
 6.0000000000E+75
 7.2000000000E+75
→00050        0545        EXP OVERFLOW
EDIT  ?
0545        THE RESULT EXPONENT EXCEEDS 75 IN FLOATING
EDIT        POINT ADD, SUBTRACT, MULTIPLY OR DIVIDE
```

```
00010 REM ** LIMITE PAR UNDERFLOW **
00020 REM **          V.2A          **
00030     INPUT A
00040      I=1
00050      PRINT A/I
00060        IF I>=7 THEN 90
00070       I=I+1
00080       GOTO 50
00090 END
EDIT
      run lprec

  ?   2.4e-78

 2.4000000000E-78
 1.2000000000E-78
 8.0000000000E-79
 6.0000000000E-79
→0
 0
 0
EDIT
```

Section 2.2
EXPRESSION ARITHMETIQUE
INSTRUCTION D'AFFECTATION

§ 2.2.1 Opérations arithmétiques

1. Opérateurs

   + addition - soustraction * multiplication
   / division ** élévation à la puissance.

2. Opérandes

   Constantes, variables numériques, expressions arithmatiques, fonctions numériques.

3. Remarques

   + et - peuvent être opérateurs unaires.
   Dans A ** B , A = 0 et B $\leq$ 0 ou A < 0 et B n'étant pas une constante entière, provoquent une erreur.

§ 2.2.2 Expressions arithmétiques

1. Définition

   Une expression arithmétique est l'association dans une même instruction d'un nombre fini d'opérateurs et d'opérandes en vue de représenter une valeur numérique.

   +1 , (A + B + C) * (X + Y) / 3 .

2. Règles principales

   - Deux opérandes doivent être réunis par un opérateur.
   - Deux opérateurs ne doivent pas être consécutifs.

```
      10 rem ** expressions numeriques **
EDIT  20 rem **       erronees          **
EDIT  30 rem **         v.26            **
EDIT  40 rem **
EDIT  50       input x,y,z
```

```
EDIT  60       a = (x+y) (x-y)
0603       SYN ERR EXPR
EDIT  ?
0603       SYNTAX ERROR IN AN EXPRESSION
```

```
EDIT  60       a = (x+y) * (x-y)
EDIT
```

```
      70       b = x** -2
0603       SYN ERR EXPR
EDIT  ?
0603       SYNTAX ERROR IN AN EXPRESSION
```

```
EDIT  70       b = x**(-2)

EDIT  80  print a;b;d
EDIT  90 stop
EDIT  100 end
EDIT
```

§ 2.2.3 Evaluation d'une expression sans parenthèse.

1. Hiérarchie des opérateurs

   1er niveau ** (plus haute priorité)

   2è niveau + et - unaires

   3è niveau * et /

   4è niveau + et - binaires (plus basse priorité).

2. Règle d'évaluation

   Les opérateurs sont réalisés :

   - de gauche à droite,
   - avec les opérandes associés,
   - en respectant la hiérarchie.

   => Si 2 opérateurs consécutifs sont :

   - de même niveau : celui de gauche est d'abord réalisé,
   - de niveau différent : celui de premier niveau est d'abord réalisé.

Exemple :

A ** B * 2 + 3.14 * Y

$$\begin{cases} (1)\ A ** B \longrightarrow R_1 \\ (2)\ R_1 * 2 \longrightarrow R_2 \\ (3)\ 3.14 * Y \longrightarrow R_3 \\ (4)\ R_2 + R_3 \longrightarrow \text{résultat final.} \end{cases}$$

3. Remarque

A **B **C s'évalue $\begin{cases} (1)\ A ** B \longrightarrow \mathcal{R} \\ (2)\ \mathcal{R} ** C \longrightarrow \text{résultat final} \end{cases}$

représente donc $(a^b)^c$ ou a^{bc} .

§ 2.2.4 Emploi des parenthèses

1. Règle fondamentale

Toute partie d'une expression, placée entre parenthèses, constitue une sous-expression.
Cette sous-expression est réalisée en premier, puis remplacée par sa valeur dans la poursuite de l'évaluation de l'expression globale.

2. Sous-expressions imbriquées

Si une sous-expression contient d'autres sous-expressions, on évalue :

- d'abord la sous-expression la plus interne,
- puis les autres, en remontant les niveaux vers le plus externe.

Chaque sous-expression élémentaire est évaluée selon les règles habituelles.

3. Rôle des parenthèses

- Modification de l'ordre naturel d'évaluation de l'expression.
- Mise en facteurs au sens algébrique.

a(b + c) s'écrira A *(B + C)

a^{bc} s'écrira A ** (B ** C)

§ 2.2.5 Instruction d'affectation

1. Forme générale

N° variable = expression-arithm.
(cible) (source)

Variable : nom d'une variable numérique.

Expression-arithm. : plus ou moins complexe. Représente une valeur numérique.

2. Réalisation

- D'abord évaluation de l'expression-arithm. avec les valeurs actuelles des variables.
- Puis, ensuite, affectation de la valeur à la variable-cible.

I = I + 1 signifie que la nouvelle valeur de I est faite de son ancienne, augmentée de 1.

3. Règles principales

- La cible doit être un nom de variable et non pas expression.

  L'équation x + 2 = y + 5 doit être transformée
  en x = y + 3 avant d'être traduite
  en X = Y + 3 .

- Après affectation, l'ancienne valeur de la cible est perdue.

4. Affectation multiple

Si plusieurs variables, v_1, v_2, ..., v_k doivent recevoir la même valeur :

n° v_1, v_2, ..., v_k = expression-arithm.

§ 2.2.6 Exemple

```
00010 REM **    EXPRESSIONS  NUMERIQUES  **
00020 REM **    PRIORITE DES OPERATEURS  **
00030 REM **              V.27           **
00040 REM **
00050       INPUT X,Y,Z
00060       C = X*Y+1.2
00070       D = X*(Y+1.2)
00080       PRINT '1';C;D
00090 REM **************
00100  E = X/Y/Z
00110  F = X/(Y*Z)
00120  G = X/Y*Z
00130  H = (X/Y)*Z
00140  PRINT '2';E;F;G;H
00150 REM **************
00160       L = X**Y**Z
00170       M = X**(Y**Z)
00180       N = (X**Y)**Z
00190       P = X**(Y * Z)
00200       PRINT '3';L;M;N;P
00210 STOP
00220 END
EDIT
```

```
EDIT  run

  ?   5,3,2

1 16.20000     20.99998

2 .8333330     .8333333     3.333332     3.333332

3 15625     1953097      15625     15625


EDIT  run

  ?   1,2,3

1 3.200000     3.200000

2 .1666666     .1666666     1.500000     1.500000

3 1      1      1      1
```

Section 2.3
FONCTIONS INCORPOREES AU BASIC

§ 2.3.1 Généralités

1. Rôle

   23 fonctions sont incorporées au langage afin d'en étendre les possibilités de calcul.

2. Utilisation

   - Dans une expression arithmétique ou bien seule, toujours en position de source, sous forme de "référence de fonction" :

     nom-fonction (argument)

   - L'argument peut être une expression arithmétique

ou une autre référence de fonction.

- Dans une expression, une référence de fonction est réalisée avant tout autre opérateur puis remplacée par sa valeur, sauf si l'argument est une expression. Dans ce dernier cas, l'argument est d'abord évalué, puis la référence de fonction.

§ 2.3.2 Fonctions mathématiques

1. Fonctions circulaires

SIN , COS , TAN , COT : argument en radians.

SEC (sécante) , CSC (cosécante)

ACS : Arccos
ASN : Arcsin
ATN : Arctg

$|arg^t| \leqslant 1$

Détermination principale en radians.

2. Fonctions hyperboliques

EXP : e^x
HCS : cosinus hyperbolique
HSN : sinus hyperbolique
HTN : tangente hyperbolique

argument < 174.6

3. Fonctions logarithmes

LOG : à base e
LTW : à base 2
LGT : à base 10

argument > 0

```
00010 REM ** FONCTIONS  "HTN" ET "LOG"  **
00020 REM **           V.22               **
00030      X = 0.25
00040      Y = HTN(X)
00050      Z = LOG((1+Y)/(1-Y)) / 2
00060    PRINT X;Y;Z
00070 REM
00080 IFX>1THEN110
00090      X = X+.25
00100 GOTO40
00110  STOP
00120 END
EDIT
```

```
EDIT  run

 .2500000     .2449186     .2499995
 .5000000     .4621171     .4999996
 .7500000     .6351488     .7499995
 1      .7615942     .9999995
 1.250000     .8482836     1.249999
```

4. Autres fonctions mathématiques

SQR : racine carrée d'un argument positif ou nul,
détermination positive.

RND : restitue un nombre pseudo-aléatoire dans
l'intervalle]0,1[.

```
00010 REM ** FONCTION  "RND"  SANS ARGUMENT  **
00020 REM **              V.21               **
00030    I=1
00040 PRINTUSING50,I,RND
00050 : ##   #.#####
00060 IFI>=10THEN90
00070    I=I+1
00080 GOTO40
00090 STOP
00100 END

EDIT
```

```
EDIT  run

   1  0.91841
   2  0.22514
   3  0.26625
   4  0.31762
   5  0.61603
   6  0.39840
   7  0.97156
   8  0.43263
   9  0.49610
  10  0.00088
```

§ 2.3.3 Fonctions arithmétiques

ABS : valeur absolue

SGN : +1 si l'argument est positif
0 si l'argument est nul
-1 si l'argument est négatif

§ 2.3.4 Fonctions de conversion

INT : partie entière par troncature

DEG : de radians en degrés et fractions décimales

RAD : de degrés (et fractions décimales) en radians

```
00010 REM **     LIMITES D'ARGUMENT     **
00020 REM ** POUR QUELQUES FONCTIONS    **
00030 REM **          V.2C              **
00040 REM **
00050       INPUT X
00060       PRINT SQR(X);
00070       PRINT ASN(X);
00080 PRINT
00090       PRINT EXP(X);
00100       PRINT LTW(X);
00110 PRINT
00120       PRINT COS(X);SIN(X);
00130 END
EDIT
```

```
run

  ?   .67
 .8185353     .7342088
 1.954237    -.5777675
 .7838218     .6209859
EDIT
```

```
run

  ?   1.01

 1.004987
00070        0667        INV ASN / ACS ARG
EDIT  ?
0667         ARGUMENTS TO ARCSINE AND ARCOSINE MUST LIE
EDIT                              BETWEEN -1 AND +1
```

```
run

  .?   -0.45

00060        0497        INV ARG SQUARE ROOT
EDIT  ?
0497         SQUARE ROOT BUILT-IN FUNCTION DOES NOT ACCEPT
EDIT                              NEGATIVE ARGUMENTS
```

```
run
  ?   0
 0     0
 1
00100        0533        LOG ARG <= 0
EDIT  ?
0533         ARGUMENT TO LOGARITHM (BASE E, 2 , OR10 )BUILT-
EDIT        IN FUNCTIONCAN NOT BE LESS THAN OR EQUAL TO ZERO

      delete 70
EDIT run

  ?   180

 13.41641
00090        0532        EXP ARG TOO BIG
EDIT  ?
0532         BUILT-IN FUNCTION EXP RECEIVED ARGUMENT GREATER
EDIT
```

```
00010 REM ** QUELQUES FONCTIONS   **
00020 REM **         V.2B         **
00030      INPUT X
00040      PRINT '1';INT(X);ABS(X)
00050      PRINT '2';SGN(X)
00060      PRINT '3';RAD(X);DEG(X); RAD(DEG(X))
00070 END
EDIT
      run
  ?   3.14159

1 3      3.141589
2 1
3 5.483107E-02    179.9998      3.141588

EDIT  run

  ?   -0.7854

1 0      .7854000
2-1
3-1.370781E-02   -45.00009     -.7853996
```

CHAPITRE 3
PREMIERES INSTRUCTIONS

Section 3.1
LECTURE ET ECRITURE

§ 3.1.1 Généralités

1. Lecture

En cours d'exécution, le programme demande à l'utilisateur de lui transmettre des données par l'intermédiaire de l'instruction INPUT.
L'utilisateur doit frapper au clavier et transmettre des valeurs numériques.

2. Ecriture

En cours d'exécution, l'utilisateur désire connaître des résultats élaborés par le programme.
L'instruction PRINT imprime ces valeurs.

§ 3.1.2 Instruction INPUT

1. Forme et rôle

n° INPUT liste-variable
Lit des valeurs numériques frappées au terminal et les affecte aux variables de liste-variable.

2. Fonctionnement

Au moment où le déroulement du programme atteint un INPUT :
- le terminal frappe un point d'interrogation (?) et quelques blancs ;
- le programme se met en attente ;

- le clavier étant débloqué, l'utilisateur frappe des constantes numériques,
  - . une par variable,
  - . dans l'ordre,
  - . en les séparant par une virgule ;
- les valeurs sont transmises au programme par un retour-chariot ;
- le programme les affecte aux variables et poursuit son exécution.

3. Règles principales

- Si les constantes sont incorrectes, un message apparaît ; aucune valeur n'est transmise au programme.
- La frappe doit être reprise depuis le début.
- Dans le programme, les valeurs numériques sont ajustées à la précision.

4. Messages d'erreur

TOOFEW : les constantes ne sont pas en nombre suffisant.

EXCESS : constantes trop nombreuses.

NG CON : constante invalide, trop grande ou trop petite.

```
00100 REM **  MESSAGES A L'INPUT   **
00105 REM **          V.31          **
00110 REM
00120    INPUT A,B,C
00130    PRINT A;
00140    PRINT B;
00150    PRINT C
00160    PRINT
00170 END
EDIT  run
  ?   1.2, 3.4,
TOOFEW1.2, 3.4, 678, 890
EXCESS  1.2,    ,  , 3.4, 678
NG CON  1.2 . 3.4, 678
NG CON  1.2,  3.4, 67890
 1.200000     3.400000     67890

EDIT  end
```

§ 3.1.3 Instruction PRINT pour une valeur

1. Forme et rôle

n° PRINT v

Imprime la valeur de v au début d'une nouvelle ligne.

2. Règles principales

v : constante, variable, expression arithmétique.

Une expression est d'abord évaluée puis imprimée ; sa valeur n'est pas conservée.

3. Forme externe de la valeur numérique

Les valeurs imprimées sont arrondies sur le dernier chiffre. Trois formes sont possibles ; le choix se fait en fonction de l'ordre de grandeur en valeur absolue et de la précision.

	Valeur entière	Valeur non-entière	
	Format I	Format E	Format F
Simple précision	0 à 10^7-1	$< 10^{-1}$ ou $\geqslant 10^7$	autres cas
Double précision	0 à $10^{15}-1$	$< 10^{-1}$ ou $\geqslant 10^{15}$	autres cas

4. Cas particulier : ligne blanche

n° PRINT

Provoque le saut d'une ligne sans aucune impression.

§ 3.1.4 Instruction PRINT pour plusieurs valeurs

1. Forme et rôle

n° PRINT ...$v_i s_i$ ...

Les v_i sont séparés par des séparateurs s_i qui servent à contrôler la disposition des formes externes sur la ligne d'impression.

2. Séparateurs

- soit la virgule (,)
- soit le point-virgule (;)

D'une manière générale, la virgule permet d'espacer les valeurs sur la ligne alors qu'au contraire le point-virgule resserre la disposition.

3. Règles principales

- Les v_i ont les mêmes formes que précédemment.
- Les valeurs sont imprimées l'une à la suite de l'autre sur la ligne ; éventuellement sur la ligne suivante.

4. Exemple

```
00010 REM ** FORME DE EXTERIEURE DE CONSTANTES **
00020 REM **            NUMERIQUES                **
00030 REM **              V.30                    **
00040   A=12
00050   B=12.
00060   C=12.0
00070   D=1.200E+01
00080   E=.0000012E+7
00090 PRINT
00100 PRINT'....5....0....5....0....5....0....5....0....5'
00110 PRINT A
00120 PRINT B,C
00130 PRINT D;E
00140 REM ***************************************
00150     A=12.34
00160     B=1234000E-4
00170     D=-123456789
00180     E=-0.0000076
00190 PRINT
00200     PRINT A
00210     PRINT B;C
00220     PRINT D;E
00230 REM ***************************************
00240   A=&PI
00250   B=&E
00260   C=&SQR2
00270   PRINT
00280   PRINT A;B;C
00290 PRINT '....5....0....5....0....5....0....5....0....5'
00300 END
```

```
EDIT  run
....5....0....5....0....5....0....5....0....5
|12
|12                  12
|12     12
|
|12.34000
|123.4000     12
-1.234568E+08   -7.600000E-06
|
|3.141593     2.718282      1.414214
|....5....0....5....0....5....0....5....0....5
```

```
EDIT  run lprec
....5....0....5....0....5....0....5....0....5
|12
|12                  12
|12     12
|
|12.3400000000000
|123.400000000000  12
-123456789   -7.60000000000E-06
|
|3.14159265358979   2.71828182845904   1.41421356237309
|....5....0....5....0....5....0....5....0....5
EDIT
```

5. Cas particulier

n° PRINT ... v_i s_i ... v_n [,] [;]

Si la dernière valeur est suivie d'un point-virgule ou bien d'une virgule, après impression de v_n, la boule ne revient pas au début de la ligne suivante.

=> La prochaine impression se fera à la suite de v_n.

Pour revenir à la ligne, il faut un PRINT blanc.

```
00010    INPUT A,B,C
00020      PRINT A;B;
00030      PRINT C
00040    PRINT '***'
00050    PRINT A;B,C,
00060    PRINT
00070    PRINT '***'
00080 END

EDIT  run
  ?   &pi,&e,&sqr2
 3.141593     2.718282     1.414214
***
 3.141593     2.718282          1.414214
***
 EDIT
```

Section 3.2
INSTRUCTIONS DE CONTROLE DE SEQUENCE

§ 3.2.1 Déroulement du programme

1. Déroulement séquentiel

Par principe, les instructions d'un programme s'exécutent l'une après l'autre selon la séquence croissante de leurs numéros.
Mais, s'il est nécessaire
- de répéter plusieurs fois une même séquence ;
- de ne réaliser une séquence que si certaines conditions se trouvent réalisées,

2. Débranchement

Dans ce but, il faudra pouvoir
- se débrancher : c'est-à-dire interrompre le déroulement séquentiel ;
- se rebrancher : reprendre le déroulement du programme en une autre instruction.

3. Identification d'une instruction

Le numéro de ligne identifie l'instruction qu'elle contient.

§ 3.2.2 Branchement impératif

1. Forme et rôle

nº GOTO nºdestination
Réalise le débranchement obligatoire à l'instruction identifiée par "nºdestination".

2. Règle

Si "nºdestination" correspond à une instruction non-exécutable (REM, DIM), le contrôle passe, en séquence, à la première instruction exécutable.

§ 3.2.3 Comparaison de valeurs numériques

1. Opérateurs de comparaison

   = égal à

   < > différent de

   > supérieur à

   >= supérieur ou égal à

   < inférieur à

   <= inférieur ou égal à

2. Expression de comparaison

   - De la forme exp_1 ⊗ exp_2

     ⊗ représente l'un des 6 opérateurs de comparaison.

     exp_1 et exp_2 : expressions arithmétiques, dont les valeurs sont comparées algébriquement.

   - Le résultat d'une telle comparaison ne peut être que VRAI ou bien FAUX .

   - Une expression de comparaison ne peut apparaître que dans un IF.

§ 3.2.4 Débranchement conditionnel: IF

1. Forme

   nº IF exp-comp | THEN / GOTO | nºdestination

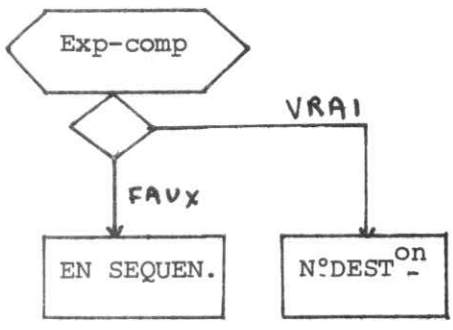
   exp-comp : expression de comparaison.

   THEN ou GOTO : 1 seul doit apparaître.

   nºdestination : nº de l'instruction de reprise du déroulement du programme.

2. Fonctionnement

   - Exp-comp est évaluée
   - Si sa valeur est VRAI, le débranchement est réalisé et le programme se poursuit avec "nºdestination"
   - Si sa valeur est FAUX,



il n'y a pas de débranchement, le déroulement se poursuit en séquence.

```
00010 REM **  EXEMPLES DE  "IF" ERRONES  **
00020 REM **             V.32             **
00030 REM
00040    INPUT A
00050  IF A=0 THEN 75
00060    PRINT A+1
00070    PRINT A+10
00080 REM ---------------------------------
00090 IFA=0GOTO100
00100    PRINT A+2
00110 END
EDIT  run
0661       UNDEF STM NUM
EDIT  ?
0661       STATEMENT NUMBER 00000075 REFERENCED
EDIT       IN A STATEMENT NOT DEFINED IN PROGRAM
```

```
00010 REM **  EXEMPLES DE  "IF" ERRONES  **
00020 REM **             V.32             **
00030 REM
00040    INPUT A
00050 IFA=0THEN70
00060    PRINT A+1
00070    PRINT A+10
00080 REM ---------------------------------
00090 IFA=0GOTO100
00100    PRINT A+2
00110 END
```

```
EDIT  run
  ?   22
 23
 32
 24
EDIT  run
  ?   0

 10
 2
EDIT
```

§ 3.2.5 Fin de programme: END

1. Forme

   n° END [commentaire] .

2. Règles

   - A la compilation, marque la fin du programme.
     ⇒ toute instruction placée au-delà du END est ignorée du compilateur.
   - A l'exécution, marque la fin de l'exécution. On doit donc s'y débrancher pour interrompre le déroulement du programme.

§ 3.2.6 Exemples de programme

1. Valeur d'un polynôme

```
00100 REM ** CALCUL DE LA VALEUR D'UN POLYNOME **
00110 REM **              W.13                  **
00120 REM ** ------------------------------------ **
00200       INPUT A,B,C,D
00210       INPUT X
00220         IF X=999 THEN 999
00230       INPUT H, X9
00240 REM **
00300         IF X>X9 THEN 210
00310       Y = ((A*X+B)*X+C)*X+D
00320       PRINT X;Y
00330       X = X+H
00340       GOTO 300
00350 REM **
00999 END
```

```
EDIT  run
  ?   1,3,3,1
  ?   0
  ?   0.2,1
 0       1
 .2000000      1.728000
 .4000000      2.743999
 .6000000      4.095999
 .8000000      5.831999
 .9999999      7.999997
  ?   -1
  ?   .3,0
-1       0
-.7000000      2.699996E-02
-.6000001      .2159998
-.5000001      .7289997
  ?   999
EDIT
```

2. Moyenne de nombres positifs

```
00100 REM ** CALCUL DE LA MOYENNE DE **
00110 REM **   NOMBRES POSITIFS       **
00120 REM **          W.14            **
00130 REM ** ------------------------ **
00200       S, N = 0
00210       INPUT X
00220         IF X>0 THEN 300
00230         IF X<0 THEN 900
00240 REM **
00250     M = S/N
00260     PRINT N;S,M
00270     GO TO 200
00290 REM **
00300     S = S + X
00310     N = N + 1
00320     GOTO 210
00390 REM **
00900 PRINT
00910 STOP W.14
00999 END
EDIT
```

```
EDIT  run
  ?   1
  ?   2
  ?   3
  ?   0
 3        6                    2
  ?   1.2
  ?   2.3
  ?   3.4
  ?   0
 3        6.899999             2.299999
  ?   -1

EDIT
```

CHAPITRE 4
AUTRES ELEMENTS DE DONNEES

Section 4.1
TABLEAUX DE VALEURS NUMERIQUES

§ 4.1.1 Notion de tableau

1. Ensemble d'éléments de données
   - organisé selon une ou plusieurs directions (dimensions)
   - dont chaque élément sera repéré par ses coordonnées dans chaque dimension.

2. Exemples
   - liste de valeurs : 1 dimension.
   - matrice : 2 dimensions.

§ 4.1.2 Tableau de valeurs numériques

1. Nom du tableau
   - Seul le tableau reçoit un nom qui servira également à identifier les éléments.
   - Formé par un seul caractère alphabétique

     A, B, ..., #

2. Organisation du tableau

   En fonction du traitement programmé, l'organisation du tableau se fera selon 1 ou 2 dimensions ; le choix est souvent dicté par le problème lui-même.

§ 4.1.3 Déclaration du dimensionnement

1. Forme et rôle

n° DIM ... , nom-tab. (dimens$^{t}_{-}$), ...

Pour chaque tableau, on précise :

- le nombre des dimensions ;
- dans chacune, la borne supérieure de variation de la coordonnée ;

La borne inférieure est toujours égale à 1.

2. Règles

- DIM est une instruction non exécutable.
- Avant d'être manipulé, un tableau doit être déclaré dans une instruction DIM.
- (dimens$^{t}_{-}$) peut prendre 2 formes :

  (S) pour une liste

  (S_1,S_2) pour un tableau

  Les S étant des constantes numériques comprises entre 1 et 255.

3. Remarque

Si une variable, dont le nom est formé d'une lettre seule, apparaît comme si elle représentait un élément de tableau, le compilateur suppose qu'il s'agit d'un tableau qu'il dimensionne implicitement à

(10) s'il s'agit d'une dimension,

(10,10) s'il s'agit de 2 dimensions.

```
EDIT   10 rem ** declaration et dimensionnement  **
EDIT   20 rem **        d'un tableau             **
EDIT   30 rem **             v.4b                **
EDIT   40 rem **
EDIT   50          input p,q
EDIT   60          dim a(p,q)

0619         NOT POS INT
EDIT   ?
0619         BOUNDS IN A DIM STATEMENT MUST BE POSITIVE INTEGERS
EDIT
       60          dim a(50,50)

EDIT   65     dim a(-1,0)
0619         NOT POS INT
EDIT   65     dim a(256,240)
0648         SUBSC > LIM
EDIT   ?
0648         MAXIMUM ARRAY BOUND IS 255
EDIT
```

§ 4.1.4 Identification d'un élément

1. Principe de l'identification

Un élément de tableau est identifié par

- le nom du tableau
- suivi, entre parenthèses, d'une liste d'indices dont les valeurs entières sont ses coordonnées dans chacune des dimensions du tableau.

2. Nombre d'indices

- Cas d'une liste : 1 indice, le rang de l'élément.
- Cas d'un tableau :

  1er indice : rang de la "ligne"

  2ème indice : rang de la "colonne"

3. Expression indicielle

- La valeur d'indice doit être
  - numérique entière
  - comprise entre 1 et la borne supérieure déclarée ou implicite.
- Elle peut s'exprimer par une expression dont seule la partie entière est utilisée.
- La valeur Ø est interdite.

4. Exemple

```
00010 REM ** ------------------------------ **
00020 REM **  EXEMPLES D'EXPRESSIONS        **
00030 REM **           INDICIELLES           **
00040 REM ** ---------  W.15 ------------- **
00050 REM
00060   DIM A(10)
00070 REM
00080   PRINT '....5....0....5....0....5....0'
00090       X=0
00100        R = RAD(X)
00110        A(1+X/10) = SIN(R) + COS(2*R)
00120 IFX>=90THEN160
00130       X=X+10
00140 GOTO100
00150 REM **
00160       I=1
00170 IFI>=10THEN210
00180   PRINT ' X = '; 10*(I-1) ; A(I)
00190       I=I+1
00200 GOTO170
00210 PRINT 'F I N'
00220 END
```

```
EDIT  run
....5....0....5....0....5....0
 X =  0     1
 X =  10    1.113340
 X =  20    1.108064
 X =  30    1
 X =  40    .8164361
 X =  50    .5923967
 X =  60    .3660257
 X =  70    .1736485
 X =  80    4.511529E-02
F I N
EDIT
```

5. Manipulation d'un élément de tableau

Elle se fait comme pour n'importe quelle variable élémentaire à condition que :

- le tableau soit convenablement dimensionné ;
- les indices (ou l'indice) soient convenables.

```
00010 REM ** TABLEAUX NUMERIQUES A 2 DIMENSIONS  **
00020 REM **                V .41                **
00030 REM **
00040   PRINT 'P, Q'
00050   INPUT P,Q
00060       DIM A(5,5)
00070 REM **
00080 REM REMPLISSAGE PAR LIGNES
00090 REM **
00100  S = 1
00110    FOR I=1 TO P
00120    FOR J=1 TO Q
00130       A(I,J) = S
00140       S = S+1
00150    NEXT J
00160   NEXT I
00170 PRINT
00180 REM **
00190 REM IMPRESSION DU TABLEAU
00200 REM **
00210     FOR I=1 TO P
00220       FOR J=1 TO Q
00230       PRINT A(I,J);
00240       NEXT J
00250     PRINT
00260     NEXT I
00270   PRINT
00280 REM **
00290 REM  RECHERCHE DANS LE TABLEAU
00300 REM **
00310    INPUT U,V,W
00320      I = 5*U/V+COS(W)
00330      J = EXP(U+V)
00340     PRINT I;J;A(I,J)
00350 PRINT
00360 GOTO310
00370 END
EDIT
```

```
run

P, Q
  ?   3,4

 1      2      3      4
 5      6      7      8
 9      10     11     12
```

```
  ?   0.2, 0.4, 0.6

3.685465    1.822119    9
```

```
  ?   1,2,3

12.56767    20.08554

00340        0402        INV ARRY SUPSC
EDIT  ?
0402        ARRAY SUBSCRIPT IS INCOMPATIBLE
EDIT        WITH DECLARED ARRAY BOUNDS
```

6. Remarque

La notion de variable indicée, au sens algébrique, ne doit pas être confondue avec celle d'élément de tableau. Une telle variable ne se sera représentée par un tel élément que si le traitement exige que ses valeurs successives soient stockées en mémoire. Dans l'exemple ci-dessous, seule la 3ème méthode nécessite cette mémorisation des termes successifs de la série.

```
00010 REM ** ---------------------------- **
00020 REM ** SUITE ET SERIE                **
00030 REM **   (-1)**N * X**(N+1)          **
00040 REM **   ------------------          **
00050 REM **        (N + 1)                **
00060 REM **                               **
00070 REM **----------------- V.4U ------- **
00080 REM
00090     INPUT X,E
00100     N = 0
00110     U = (-1)**N * X**(N+1) / (N+1)
00120     PRINT U
00130 IFABS(U)<ETHEN160
00140     N = N+1
00150 GOTO110
00160 REM ----------------------------------
00170 PRINT '***'
00180  N,S = 0
00190  U = (-1)**N * X**(N+1) / (N+1)
00200  S = S + U
00210  PRINT U ; S
00220 IFABS(U)<ETHEN250
00230  N = N+1
00240 GOTO190
00250 REM ----------------------------------
00260 PRINT '***'
00270   DIM V(30)
00280     N = 0
00290     V(N+1) = (-1)**N*X**(N+1)/(N+1)
00300 IFABS(V(N+1))<ETHEN330
00310     N = N+1
00320 GOTO290
00330 REM
00340   S = 0
00350   S = S + V(N+1)
00360   PRINT V(N+1) ; S
00370 IFN<=0THEN400
00380   N = N-1
00390 GOTO350
00400 REM
00410 PRINT
00420 PRINT LOG(1+X)
00430 STOP V4U
00440 END
EDIT
```

```
 ?   0.2 , 1e-10
 .2000000
-2.000000E-02
 2.666667E-03
-3.999998E-04
 6.399996E-05
-1.066666E-05
 1.828571E-06
-3.200000E-07
 5.688888E-08
-1.024004E-08
 1.861828E-09
-3.413358E-10
 6.301597E-11
***
 .2000000     .2000000
-2.000000E-02   .1799999
 2.666667E-03   .1826666
-3.999998E-04   .1822666
 6.399996E-05   .1823305
-1.066666E-05   .1823199
 1.828571E-06   .1823217
-3.200000E-07   .1823213
 5.688888E-08   .1823213
-1.024004E-08   .1823213
 1.861828E-09   .1823213
-3.413358E-10   .1823213
 6.301597E-11   .1823213
***
 6.301597E-11   6.301597E-11
-3.413358E-10  -2.783198E-10
 1.861828E-09   1.583509E-09
-1.024004E-08  -8.656528E-09
 5.688888E-08   4.823235E-08
-3.200000E-07  -2.717676E-07
 1.828571E-06   1.556803E-06
-1.066666E-05  -9.109862E-06
 6.399996E-05   5.489010E-05
-3.999998E-04  -3.451005E-04
 2.666667E-03   2.321558E-03
-2.000000E-02  -1.767844E-02
 .2000000     .1823215
 .182321556793955
EDIT
```

§ 4.1.5 Calcul matriciel

Les tableaux numériques à 1 ou 2 dimensions pourront être manipulés globalement et non plus seulement par élément en utilisant les possibilités des instructions matricielles. Cf. chapitre 8.

Section 4.2
DONNEES LITTERALES

§ 4.2.1 Constantes littérales

1. Définition

   Suite ou chaîne de caractères quelconques du clavier du 2741, délimitée à gauche et à droite
   - soit par une apostrophe (')
   - soit par un guillemet (")

   'CHAINE' "DE 16 CARACTERES"

2. Longueur

   Nombre des caractères, blancs internes inclus, mais délimiteurs exclus.
   Dans l'exemple ci-dessus : 6 et 16

3. Remarque

   Si l'apostrophe sert de délimiteur et qu'il doit aussi apparaître dans la chaîne avec sa valeur caractère, par convention, on doit le doubler à l'intérieur de la chaîne.

'	C	'	'	E	S	T	'

§ 4.2.2 Variable littérale

1. Nom d'une variable littérale

   2 caractères :

   1): 1 lettre A, B, ..., Z, ... #

   2): $ obligatoirement.

2. Valeur d'une variable littérale

   Sa valeur peut provenir d'une constante littérale ou d'une autre variable, uniquement.

3. Forme interne

   En mémoire, chaque variable dispose de 18 octets, chaque octet, pouvant recevoir un caractère.

§ 4.2.3 Manipulation des valeurs littérales

1. Opérations de comparaison

   Seuls les opérateurs de comparaison peuvent s'appliquer à des valeurs littérales.

   De telles expressions ne peuvent apparaître que dans un IF.

2. Réalisation de la comparaison

   - De gauche à droite.
   - Caractère par caractère.
   - La valeur la plus courte étant complétée de blancs à droite.
   - Le résultat sera VRAI ou FAUX.
   - L'ordre lexicographique des caractères est :

     ∅ < caractères spéciaux < A < B ... < Z < Ø < 1 ... < 9

3. Exemple de comparaison

```
00010 REM ** COMPARAISONS ENTRE LITTERAUX **
00020 REM **              V.46              **
00030 REM **
00040 IF'AB'='AB  'THEN80
00050 REM
00060     PRINT '1 / DIFFERENTS'
00070 GOTO90
00080     PRINT '1 / EGAUX'
00090 REM ************************
00100 IF'AB'=' AB'THEN190
00110  PRINT '2 / DIFFERENTS'
00120 REM
00130 IF'AB'>' AB'THEN160
00140     PRINT '2 / INFERIEUR'
00150 GOTO180
00160     PRINT '2 /   SUPERIEUR'
00170 GOTO180
00180 GOTO200
00190  PRINT '2 / EGAUX'
00200 END
EDIT
```

```
1 / EGAUX
2 / DIFFERENTS
2 /   SUPERIEUR
EDIT
```

4. Instruction d'affectation

n° var-littérale = valeur littérale

- Le nom de la variable est reconnaissable au $ en 2ème caractère.
- Valeur littérale : constante littérale ou bien variable littérale.

En mémoire, la valeur est affectée à la cible. Elle est cadrée à gauche. L'ajustement se fait à droite :

- remplissage de # jusqu'à la 18ème position si la source est trop courte ;
- troncature à partir du 19ème caractère si la source est trop longue.

```
EDIT  10 rem ** constantes litterales **
EDIT  20 rem **          v.45           **
EDIT  30 rem **
EDIT  40        a$ = 'c'est l'homme'
0603      SYN ERR EXPR
EDIT  ?
0603      SYNTAX ERROR IN AN EXPRESSION
EDIT
      40        a$ = "c'est l'homme"
EDIT  50        print '1',a$
```

§ 4.2.4 Transmission des données littérales

1. Lecture par INPUT

   nº INPUT, ..., V$,...

   Sur le terminal, les valeurs littérales doivent se présenter sous forme de constantes littérales convenablement délimitées. Au-delà du 18ème caractère, elles seront tronquées.

2. Ecriture par PRINT

   nº PRINT ...vs ...

   v représente une valeur littérale, variable ou constante.

   s un séparateur (,) ou (;).

   Dans le cas d'une variable littérale, la longueur de la zone d'impression est de 18, moins les blancs de tête. S'il s'agit d'une constante, la longueur est celle de la constante.

```
00010 REM ** CONSTANTES LITTERALES **
00020 REM **          V.45          **
00030 REM **
00040         A$ = "C'EST L'HOMME"
00050         PRINT '1',A$
00060 REM
00070         B$ = 'C''EST L''HOMME'
00080         PRINT '2',B$
00090 REM
00100    INPUT C
00110    PRINT '3';C
00120 END
EDIT  run

1                    C'EST L'HOMME

2                    C'EST L'HOMME
```

```
  ?   c'est l'homme
NG CON'c'est l'homme'
NG DEL"c'est l'homme"
00100        0660        INV TYPE DATA
EDIT  ?
0660        THE DATA BEING ASSIGNED MUST BE THE SAME
EDIT         TYPE AS THE VARIABLE RECEIVING THE DATA
```

§ 4.2.5 Liste de valeurs littérales

Le nom est de la même forme que pour une variable. Le tableau ne peut avoir qu'une seule dimension. La déclaration se fait explicitement par DIM ou bien implicitement.

```
00010 REM ** LECTURE ET IMPRESSION **
00020 REM **       DE LITTERAUX     **
00030 REM **         V.47           **
00040 REM **
00050  PRINT '....5....0....5....0....5....0....5....0'
00060     INPUT A$
00070     PRINT '1';A$
00080 REM
00090     INPUT A$(1),A$(2)
00100     PRINT '2';A$(1);A$(2)
00110     PRINT '3';A$(1) , A$(2)
00120 PRINT '....5....0....5....0....5....0....5....0'
00130 END
EDIT
```

```
|....5....0....5....0....5....0....5....0
   ?   'ce texte doit etre lu, puis imprime'
1CE TEXTE DOIT ETRE
   ?   'ce texte doit etre' , " lu, puis imprime"
2CE TEXTE DOIT ETRE LU, PUIS IMPRIME
3CE TEXTE DOIT ETRE<--------------->LU, PUIS IMPRIME
|....5....0....5....0....5....0....5....0
```

Exemple de liste de littéraux

```
00010 REM ** TABLEAUX DE LITTERAUX   **
00020 REM **          V.49           **
00030 REM **
00040      INPUT M,A$(M)
00050        IF M <> 99 THEN 40
00060 REM **
00070 FOR I=1 TO 20
00080 PRINT I;A$(I)
00090 NEXT I
00100 END
EDIT

     run
  ?  1, 'ain'
  ?  2, 'aisne'
  ?  3, 'aude'
  ?  3, 'aube'
  ?  44, 'loire-atlantique'
00040        0402        INV ARRY SUBSC
EDIT  ?
0402        ARRAY SUBSCRIPT IS INCOMPATIBLE WITH DECLARED ARRAY BOUNDS
EDIT
```

```
00010 REM ** TABLEAUX DE LITTERAUX   **
00020 REM **          V.49           **
00030 REM **
00040   INPUT M
00045    IF M = 99 THEN 70
00050    INPUT A$(M)
00055    GOTO 40
00060 REM **
00070 FOR I=1 TO 20
00080 PRINT I;A$(I)
00090 NEXT I
00100 END
EDIT
```

```
EDIT  run
  ?   1
  ?   'ain'
  ?   2
  ?   'aisne'
  ?   3
  ?   'aube'
  ?   3
  ?   'pyrenees atlantiques'
  ?   8
  ?   'alpes de haute provence'
  ?   99

 1    AIN
 2    AISNE
 3    PYRENEES ATLANTIQU
 4
 5
 6
 7
 8    ALPES DE HAUTE PRO
 9
 10
 11
00080        0402        INV ARRY SUBSC
```

§ 4.2.6 Emploi des valeurs littérales

Le Basic n'autorise qu'une utilisation limitée et accessoire des valeurs littérales principalement dans "le mode conversationnel" pour établir le dialogue entre le programme et l'utilisateur, via le terminal.

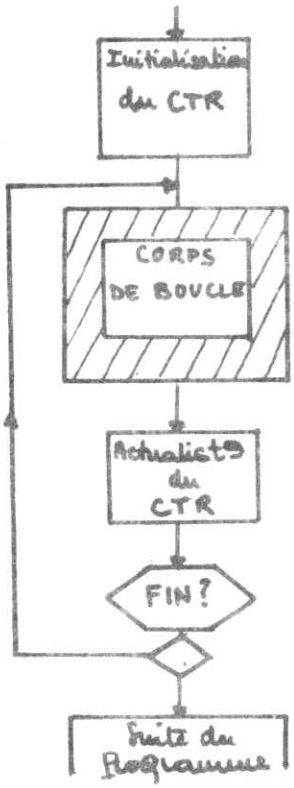
```
00010 REM **  CALCUL DE MOYENNE   **
00020 REM **       EN  MODE       **
00030 REM **   CONVERSATIONNEL    **
00040 REM **         V.44         **
00050 PRINT 'DES VALEURS A LIRE'
00060     INPUT R$
00070 IFR$='OUI'THEN120
00080 IFR$="NON"THEN300
00090 PRINT '"OUI" OU "NON"'
00100 GOTO60
00110 REM **
00120     PRINT 'COMBIEN'
00130     INPUT N
00140     S, M = 0
00150 REM **
00160 REM ** LECTURE & CUMUL DES  X
00170 REM **
00180     INPUT X
00190     M = M+1
00200     S = S+X
00210 IFM<NTHEN180
00220 REM **
00230 REM **  IMPRESSIONS
00240 REM **
00250     PRINT
00260     PRINT '  TOTAL    = '; S
00270     PRINT ' MOYENNE   = ';S/M
00280    PRINT '***'
00290 GOTO50
00300 PRINT 'F I N'
00310 STOP
00320 END
EDIT
```

```
EDIT  run

DES VALEURS A LIRE
  ?   'oui'
COMBIEN
  ?   3
  ?   1
  ?   2
  ?   3

  TOTAL    =  6
 MOYENNE   =  2
***
DES VALEURS A LIRE
  ?   ' oui'
"OUI" OU "NON"
  ?   'oui  '
COMBIEN
  ?   2
  ?   1.2
  ?   2.3

  TOTAL    =  3.499999
 MOYENNE   =  1.749999
***
DES VALEURS A LIRE
  ?   "non"
F I N
EDIT
```

CHAPITRE 5
INSTRUCTIONS DE CONTROLE DE SEQUENCE

Section 5.1
BOUCLES FOR/NEXT

§ 5.1.1 Généralités



Les éléments caractéristiques d'une boucle sont :

- Initialisation du compteur d'itérations.
- Corps de boucle à répéter.
- Actualisation du compteur.
- Test de sortie de boucle : par comparaison à une valeur limite, on décide de réaliser à nouveau le corps de boucle ou bien de l'abandonner pour reprendre le déroulement séquentiel.

§ 5.1.2 Instruction FOR

1. Forme et rôle

   nº FOR v = exp1 TO exp2 [STEP exp3]

   - Marque la limite antérieure du corps de boucle.
   - Initialise le compteur des itérations (v) et contrôle le nombre de leurs réalisations.

   v : variable de contrôle ou compteur des itérations : doit être numérique.

   exp1 : valeur initiale de v,

   exp2 : valeur de référence, limite à ne pas dépasser,

   exp3 : pas de variation de v.

2. Règles principales

   - Les "expi" sont des constantes, variables ou expressions numériques.

- Leurs valeurs sont évaluées et mémorisées avant toute exécution de la boucle
  => restent invariantes
- exp3 omis => exp3 = 1
- A chaque FOR doit être associée une instruction NEXT spécifiant la même variable de contrôle v.

§ 5.1.3 Instruction NEXT

1. Forme et rôle

nº NEXT v

Marque la limite postérieure du corps de boucle.

2. Règle principale

v doit être le nom d'une variable de contrôle d'un FOR antérieur.

3. Exemples d'erreur

```
00010 REM **       ERREURS      **
00020 REM **   POUR  FOR/NEXT   **
00030 REM **         V.5B       **
00040 REM
00050      FOR I=1 TO 3
00060      A(I)=I
00070    NEXT K
00080  END
EDIT  run
```

```
00070          0658          NEXT VAR NOT = FOR VAR VAR
EDIT  ?
0658          NEXT STATEMENT VARIABLE MUST MATCH THE
EDIT           PREVIOUS FOR STATEMENT VARIABLE
```

```
00010 REM **       ERREURS      **
00020 REM **   POUR  FOR/NEXT   **
00030 REM **         V.5B       **
00040 REM
00050 ┌──FOR I=1 TO 3
00060 │  ┌─FOR J=1 TO 3
00070 ?  │ A(I,J)=I+J
00080 *  NEXT J
00090  END
EDIT
       run
```

```
0633          NUM FOR/NEXT NOT =
EDIT  ?
0633          THE PROGRAM MUST CONTAIN THE SAME
EDIT           NUMBER OF FORS AND NEXTS
```

§ 5.1.4 Boucle FOR/NEXT

1. Schéma d'une boucle contrôlée par FOR/NEXT

n°- FOR v = exp1 TO exp2 [STEP exp3]

Corps de boucle

n°' NEXT v

n°" instruction suivante

FOR doit précéder immédiatement la première instruction du corps de boucle ; NEXT doit suivre immédiatement la dernière.

2. Réalisation du corps de boucle

(1) Initialisation : exp1, exp2, exp3 sont évaluées puis stockées.

(2) v prend la valeur de exp1.

(3) Test :

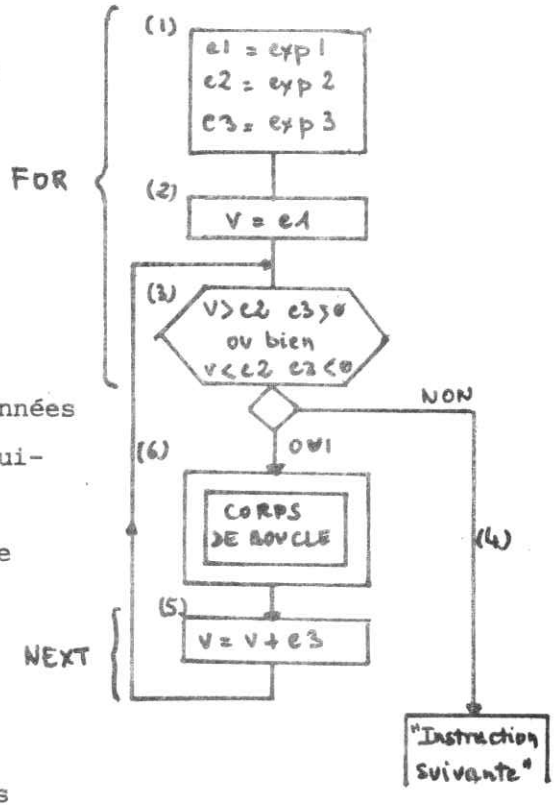
v est comparé à exp2 :

- si exp3 > 0 on vérifie si v > exp2.
- si exp3 < 0 on vérifie si v < exp2.

(4) Si exp2 est dépassé, les itérations sont abandonnées et le programme se poursuit avec "Instruction suivante", placée derrière le NEXT.

(5) Actualisation de v : une fois le corps de boucle réalisé, v est augmenté algébriquement de e3.

(6) Retour au test.



3. Sortie de boucle

Normale : lorsque toutes les possibilités spécifiées dans le FOR ont été réalisées.

Anormale : toutes les itérations n'ont pas été réalisées, en raison de tests supplémentaires ou d'erreurs à l'exécution.

4. Règles principales

- Si la condition qui met fin aux itérations est réalisée d'emblée, la boucle n'est pas réalisée du tout.
- La partie entière de v peut servir d'indice.
- Un pas négatif doit toujours être précisé.

5. Exemple

```
00010 REM **  VALEURS PRISES PAR LE VARIABLE  **
00020 REM **     DE CONTROLE  DU   FOR        **
00030 REM ** ------------ V.51 ------------ **
00040 REM **
00050 REM **
00060      FOR I=14 TO 17
00070      PRINT I;
00080      NEXT I
00090 PRINT
00100 REM **
00110      FOR I=14 TO 17
00120      PRINT I;
00130       I = I+1
00140      NEXT I
00150 PRINT
00160 REM **
00170      FOR J=1 TO 8 STEP 2
00180       PRINT J;
00190       NEXT J
00200 PRINT
00210 REM **
00220      FOR K=5 TO 3 STEP -1.5
00230      PRINT K;
00240      NEXT K
00250 PRINT
00260 REM **
00270      A = 1
00280      B = 2
00290      C = .4
00300         FOR U = A TO B STEP C
00310          PRINT U;
00320          B = B+1
00330          C = C/2
00340         NEXT U
00350 PRINT
00360 REM **
00370 GOTO320
00380 PRINT 'F I N'
00390 STOP
00400 END
```

```
run
 14     15     16     17

 14     16

 1      3      5      7

 5      3.500000

 1      1.400000     1.799999

00340          0624          INV FOR/NEXT BRANCH
EDIT  ?
0624          INVALID BRANCH INTO FOR/NEXT LOOP
EDIT
```

§ 5.1.5 Boucles multiples

1. Boucles successives

Leurs portées sont disjointes. La 1ère se termine avant que la seconde se réalise.

```
FOR  v = ——
NEXT v
FOR  v = ——
NEXT v
```

2. Boucles imbriquées

La portée de l'une ("boucle interne") est comprise en totalité (FOR et NEXT inclus) à l'intérieur de la portée de l'autre dite "boucle externe".
A chaque réalisation de chaque boucle externe, la boucle interne est totalement exécutée. Les deux variables de contrôle doivent avoir des noms différents.

```
FOR  v = ——
  FOR  w = ——
  NEXT w
NEXT w
```

3. Exemples

```
00010 REM **    BOUCLES SUCCESSIVES   **
00020 REM **           V.5A           **
00030 REM **
00040   FOR I=1 TO 3
00050     B(I)=I*I
00060   NEXT I
00070 REM
00080   FOR I=1 TO 3
00090     PRINT B(I);
00100 NEXT I
00110   PRINT
00120   PRINT '***'
00130 REM ****************************
00140 REM **
00150 REM **  BOUCLES IMBRIQUEES    **
00160 REM **
00170   FOR I=1 TO 3
00180     FOR J= I TO 3
00190     A(I,J),A(J,I)=10*I+J
00200     NEXT J
00210   NEXT I
00220 REM
00230   FOR J=1 TO 3
00240     FOR K=1 TO 3
00250       PRINT A(J,K);
00260     NEXT K
00270     PRINT
00280   NEXT J
00290 STOP
00300 END
```

```
EDIT  run

 1     4     9

***
 11    12    13
 12    22    23
 13    23    33
```

4. Remarque

Deux boucles ne doivent pas se chevaucher.

```
00010 REM **       ERREURS      **
00020 REM **   POUR  FOR/NEXT   **
00030 REM **         V.5B       **
00040 REM
00050 ┌─FOR I=1 TO 3
00060 │ ┌─FOR J=1 TO 3
00070 │ │ A(I,J)=I+J
00075 └─┼─ NEXT I
00080   └NEXT J
00090 END
EDIT
```

```
     run
00075        0658        NEXT VAR NOT = FOR VAR VAR
```

§ 5.1.6 Transfert du contrôle

1. Sortie du corps de boucle

Elle est toujours possible.

2. Retour dans le corps de boucle

- On ne peut rentrer dans un corps de boucle dont les itérations ont été initialisées, c'est-à-dire par le FOR.
- Si des boucles sont imbriquées, il n'est pas possible de sauter des niveaux d'imbrication.
- Pour passer à l'itération suivante, se rebrancher au NEXT.

3. Exemple

```
00010 REM ** SORTIE ANORMALE DE BOUCLE **
00020 REM **            V.5H            **
00030 REM
00040    FOR I=1 TO 3
00050     FOR J=1 TO 3
00060     A(I,J)=I+J
00070     NEXT J
00080    NEXT I
00090 REM **
00100  FOR I=1 TO 3
00110   FOR J=1 TO 3
00120      IF A(I,J)>2 THEN 200
00130   NEXT J
00140 →NEXT I
00150 STOP
00200      PRINT I;J;A(I,J)
00310    GOTO 140
00320 END
EDIT
```

```
EDIT  run

 1      2      3
 2      1      3
 3      1      4
EDIT
```

Section 5.2
AUTRES INSTRUCTIONS

§ 5.2.1 Branchement calculé

1. Forme et rôle

   n° GOTO n1, n2, ..., nk ON exp .

   Réalise le débranchement aux instructions identifiées par n1, n2, ..., nk selon que la partie entière de "exp" prend les valeurs 1, 2, ..., k.

2. Règles

   - exp : est une expression arithmétique dont seule la partie entière est utilisée.
   - Si cette valeur n'appartient pas à [1,k] , le déroulement se poursuit en séquence.
   - Les ni peuvent ne pas être différents entre eux, mais doivent être différents du n° de l'instruction.

3. Exemple

4. Remarque

Deux boucles ne doivent pas se chevaucher.

```
00010 REM **       ERREURS      **
00020 REM **  POUR  FOR/NEXT    **
00030 REM **        V.5B        **
00040 REM
00050 ┌─FOR I=1 TO 3
00060 │ ┌─FOR J=1 TO 3
00070 │ │ A(I,J)=I+J
00075 └─│─ NEXT I
00080   └NEXT J
00090 END
EDIT
```

```
run
00075        0658        NEXT VAR NOT = FOR VAR VAR
```

§ 5.1.6 Transfert du contrôle

1. Sortie du corps de boucle

Elle est toujours possible.

2. Retour dans le corps de boucle

- On ne peut rentrer dans un corps de boucle dont les itérations ont été initialisées, c'est-à-dire par le FOR.
- Si des boucles sont imbriquées, il n'est pas possible de sauter des niveaux d'imbrication.
- Pour passer à l'itération suivante, se rebrancher au NEXT.

3. Exemple

```
00010 REM ** SORTIE ANORMALE DE BOUCLE **
00020 REM **              V.5H            **
00030 REM
00040     FOR I=1 TO 3
00050      FOR J=1 TO 3
00060      A(I,J)=I+J
00070      NEXT J
00080     NEXT I
00090 REM **
00100  FOR I=1 TO 3
00110   FOR J=1 TO 3
00120      IF A(I,J)>2 THEN 200
00130   NEXT J
00140 →NEXT I
00150 STOP
00200      PRINT I;J;A(I,J)
00310    GOTO 140
00320  END
EDIT
```

```
EDIT  run

 1      2      3
 2      1      3
 3      1      4
EDIT
```

Section 5.2
AUTRES INSTRUCTIONS

§ 5.2.1 Branchement calculé

1. Forme et rôle

   n° GOTO n1, n2, ..., nk ON exp .

   Réalise le débranchement aux instructions identifiées par n1, n2, ..., nk selon que la partie entière de "exp" prend les valeurs 1, 2, ..., k.

2. Règles

   - exp : est une expression arithmétique dont seule la partie entière est utilisée.
   - Si cette valeur n'appartient pas à [1,k] , le déroulement se poursuit en séquence.
   - Les ni peuvent ne pas être différents entre eux, mais doivent être différents du n° de l'instruction.

3. Exemple

```
00010 REM **          GOTO CALCULE
00020 REM **  PARTIE ENTIERE D'UNE
00030 REM **  VALEUR  NUMERIQUE
00040 REM ** -------- V.56 ----------
00050 REM **
00060   INPUT E1, E2, E3
00070   PRINT
00080   PRINT E1;E2;E3
00090   PRINT
00100 REM **
00110       FOR X = E1 TO E2 STEP E3
00120       PRINT X;
00130  GOTO 200, 220 ON INT(X)
00140       PRINT 'HORS-INTERVALLE'
00150     NEXT X
00160     PRINT
00170     PRINT 'F I N'
00180 GOTO260
00190 REM **--------------------------
00200       PRINT 'INT(X)=1'
00210 GOTO150
00220       PRINT 'INT(X)=2'
00230 GOTO150
00260 STOP
00270 END
EDIT
```

```
EDIT  run
  ?  00.8,2.3, 0.2

 .8000000     2.299999     .2000000

 .8000000    HORS-INTERVALLE
 .9999999    HORS-INTERVALLE
 1.200000    INT(X)=1
 1.400000    INT(X)=1
 1.599999    INT(X)=1
 1.799999    INT(X)=1
 1.999999    INT(X)=1
 2.199999    INT(X)=2

F I N
```

```
EDIT  run

  ?   0.75, 2.3, 0.25

 .7500000     2.299999     .2500000

 .7500000    HORS-INTERVALLE
 1     INT(X)=1
 1.250000    INT(X)=1
 1.500000    INT(X)=1
 1.750000    INT(X)=1
 2     INT(X)=2
 2.250000    INT(X)=2

F I N
```

```
00120       PRINT X;
00130  GOTO 200, 220 ON INT(X+2E-6)
00140       PRINT 'HORS-INTERVALLE'
```

```
EDIT  run

  ?   0.8, 2.3, 0.2

 .8000000     2.299999     .2000000

 .8000000    HORS-INTERVALLE
 .9999999    INT(X)=1
 1.200000    INT(X)=1
 1.400000    INT(X)=1
 1.599999    INT(X)=1
 1.799999    INT(X)=1
 1.999999    INT(X)=2
 2.199999    INT(X)=2

F I N
```

§ 5.2.2 Arrêt programmé

1. Rôle

La fin de l'exécution d'un programme est obtenue lorsque le END du programme est exécuté.
Pour interrompre le programme temporairement ou définitivement, on peut utiliser un PAUSE ou un STOP.

2. PAUSE

n° PAUSE [commentaire]

- Arrêt et mise en attente du programme avec le message.

  PAUSE AT LINE n°

- Pour que le programme se poursuive, faire "retour-chariot", éventuellement après avoir frappé un commentaire.

3. STOP

n° STOP [commentaire]

- Arrêt définitif du programme et retour au mode EDIT

4. Exemple

```
00010  REM **  PAUSE ET STOP **
00020  REM **      V.5K       **
00030 REM
00040     FOR I=1 TO 5
00050     A(I)=COS(I)
00060       PRINT A(I)
00070 IFA(I)<0THEN100
00080     NEXT I
00090 GOTO120
00100      PAUSE  V.5K
00110 GOTO80
00120 REM
00130 STOP V.5K
00140  END
EDIT
```

```
      run
 .5403023
-.4161468
 PAUSE AT LINE 00100
      negatif
-.9899926
 PAUSE AT LINE 00100
      negatif
-.6536437
 PAUSE AT LINE 00100

 .2836621
EDIT
```

CHAPITRE 6
NOUVELLES FORMES D'ENTREES-SORTIES

Section 6.1
LECTURE : READ-DATA

§ 6.1.1 Fichier des données incorporé DATA

1. But

La lecture par INPUT nécessite la frappe des données à chaque exécution du programme. Il est parfois préférable de fournir en bloc, une fois pour toutes, l'ensemble des données nécessaires au programme. Un tel ensemble de données constituera "un fichier incorporé de données". Il est inséré dans le programme au moyen d'instructions DATA.

2. Instruction DATA

n° DATA liste-constantes.

- C'est une instruction non exécutable.
- Peut être placée n'importe où dans le programme.
- Liste-constantes : suite de constantes valides ; numériques et/ou littérales. Elles sont séparées entre elles par une virgule.
- La dernière constante d'une DATA ne doit pas être suivie d'une virgule.

n° DATA C1, C2, ..., CK

3. Cas de plusieurs DATA

- Plusieurs DATA, où qu'ils se trouvent dans le programme, ont un effet cumulatif

n_1 DATA C_1, C_2

n_2 DATA C_3

est équivalent à DATA C_1, C_2, C_3

- Ces DATA constituent une seule table interne de

constantes construite selon la séquence de leurs numéros.

- Dans un programme, il est préférable de fractionner les DATA : cela facilite leur modification.

§ 6.1.2 Lecture du fichier incorporé : READ

1. Forme et rôle

n° READ liste-variables.

Réalise la lecture des constantes du fichier de données incorporé pour les affecter aux variables spécifiées.

Dans la suite des lectures successives, le fichier est exploré séquentiellement de la 1ère à la dernière valeur.

Un indicateur se déplaçant le long de la table indique la 1ère constante disponible.

2. Règles

- Il doit y avoir concordance entre les variables et les constantes.
- Les valeurs numériques sont ajustées à la précision du programme.
- Si le fichier est trop court, le programme est abandonné avec le message MSNG DATA.

3. Remarque : contrôle des lectures

Afin d'accorder la longueur du fichier incorporé au nombre des lectures, on devra pouvoir interrompre ces lectures lorsque sera atteinte la fin du fichier,

- soit en connaissant à priori le nombre des lectures,
- soit en plaçant à la fin du fichier une donnée jouant le rôle de "marque de fin de fichier".

4. Exemple

```
00010 REM **  UTILISATION DE DATA   **
00020 REM **         V.68            **
00030 REM **
00040      Z = 0
00050      READ A$
00060 IFA$='FIN'THEN400
00070      READ N
00080 IFA$='ARITH.'THEN220
00090 IFA$='QUADR.'THEN290
00100  PRINT 'ERREUR';A$
00110   STOP V.68
00120 REM ------------
00130 DATA 'ARITH.'
00140 DATA 6
00150 DATA 1,2,3,4,5,6
00160 DATA 'QUADR.', 6
00170 DATA 1,2,3,4,5,6
00180 DATA 'ARITH.'
00190 DATA 2,2,2
00200 DATA 'FIN'
00210 REM ------------
00220 REM
00230     FOR I=1 TO N
00240     READ X
00250     Z = Z+X
00260     NEXT I
00270      M = Z/N
00280 GOTO360
00290 REM
00300     FOR I=1 TO N
00310     READ X
00320     Z = Z + X*X
00330     NEXT I
00340       M = SQR(Z/N)
00350 GOTO360
00360 REM
00370 PRINT
00380   PRINT 'MOYENNE ';A$;M
00390 GOTO40
00400 PRINT
00410 PRINT A$
00420 END
```

```
EDIT  run

MOYENNE ARITH. 3.500000

MOYENNE QUADR. 3.894440

MOYENNE ARITH. 2

FIN

EDIT
```

§ 6.1.3 Reprise de lecture : RESTORE

1. Forme et rôle

n° RESTORE

Repositionne l'indicateur de lecture au début du fichier incorporé.

=> le READ suivant lit la 1ère valeur du fichier.

Un RESTORE sans objet est ignoré.

2. Exemple

```
00010 REM ** RECHERCHE DES FACTEURS DANS UN NOMBRE **
00020 REM **         UTILISATION DE  RESTORE         **
00030 REM **               V.03                      **
00040      PRINT
00050      DATA  2,3,4
00060      INPUT N
00070 IFN=-1THEN250
00080      S = 0
00090      READ D
00100 IFD=-1THEN180
00110 IFN/D<>INT(N/D)THEN90
00120      PRINT D;
00130      S = 1
00140    DATA    5
00150 GOTO90
00160 REM **
00170       DATA 7,11,13,17
00180    RESTORE
00190 REM **
00200 IFS=1THEN40
00210 PRINTUSING220
00220 : PAS DE FACTEUR CONNU POUR N
00230 GOTO40
00240 REM **
00250 REM **
00260 STOP V.03
00270 DATA -1
00280    END
EDIT
```

```
      run

 ?   120
2     3     4     5
 ?   77
7     11
 ?   1001
7     11    13
 ?   1949
PAS DE FACTEUR CONNU POUR N

 ?   11011
7     11    13
 ?   17017
7     11    13    17
 ?   -1
EDIT
```

Section 6.2
IMPRESSION AVEC IMAGE

§ 6.2.1 Généralités

- Les seuls séparateurs utilisables dans un PRINT n'offrent que de faibles possibilités d'organiser la disposition des valeurs imprimées.
- Si l'on veut programmer cette disposition, on utilisera un PRINT USING auquel on associe une "instruction-image" dont le rôle est de décrire la disposition des données sur la ligne d'impression.

§ 6.2.2 Instruction PRINT USING

1. Forme et rôle

   nº PRINT USING nº image [, liste-valeurs]

   - Imprime les valeurs conformément aux spécifications de l'instruction-image identifiée par "nº image".
   - nº image doit être le nº d'une instruction-image qui peut être placée n'importe où dans le programme.
   - liste-valeurs : si elle existe, elle a la même forme que pour un PRINT, la virgule étant le seul séparateur autorisé.

2. Règles fondamentales

   - A chaque valeur numérique à transmettre, on doit pouvoir associer une image.
   - Chaque valeur numérique est alors convertie en forme externe conformément aux spécifications de l'image.
   - Si liste-valeurs n'existe pas, seuls les littéraux internes de "instruction-image" sont imprimés.

§ 6.2.3 Instruction image

1. Forme générale

n°-image : liste-images.

Caractérisée par le ":" qui suit le n° image, après un blanc au moins.

n° image doit se retrouver dans un PRINT USING.

liste-images : donne le modèle d'une ligne d'impression. La description de la ligne commence immédiatement après le ":".

2. Littéraux internes

Ce sont des littéraux inclus directement dans l'image. Ils n'ont pas besoin d'être délimités par ' ou " et ne doivent pas contenir le caractère # .

Ils sont imprimés exactement comme ils se présentent dans l'image.

3. Images pour valeurs numériques

Ce sont + - . # !

Ils permettent d'indiquer le format et la précision de la forme externe de la valeur numérique.

+ ou - : position de signe,

. : position du point fractionnaire,

: position pour un chiffre décimal (ou un blanc),

! : position d'exposant. (4)

4. Différentes formats possibles

Format I [±] # # # ... # exclus : . et !

Format F [±] # # ... . # obligatoire : .

exclus : !

Format E [±] # # ... . # !!!! symbolise l'exposant.

5. Apparition du signe

Image	Valeur	
	positive	négative
+	+	-
-	b	-
b	b	- ou ** ... *

6. Image pour valeurs littérales

Ce sont les valeurs littérales du PRINT. On peut leur associer une image quelconque, chaque symbole représentant une position de caractère.
Le cadrage se fait à gauche avec ajustement à droite.

§ 6.2.4 Réalisation du PRINT USING

1. Principe de fonctionnement

PRINT USING n°-image, v_1, ..., v_k
n°-image : l_1 s_1 ... l_k s_k

En admettant qu'il y a autant de valeur que d'images. L'impression se fait caractère par caractère selon les spécifications d'image. Un littéral interne l_i est imprimé exactement comme il apparaît dans l'image. La valeur v_i sera imprimée avec la spécification s_i et apparaîtra avec un nombre de caractères égal au nombre des symboles de l'image s_i .
L'impression débute toujours au commencement de l'image.

2. Cas particuliers

- Si les v_i sont plus nombreux que les s_i , l'image est réutilisée depuis le début, littéraux internes inclus, une nouvelle ligne étant alors imprimée.
- Si les v_i sont moins nombreux que les s_i , l'impression s'arrête avec le premier s_i sans v_i correspondant.

```
00010 REM **  PRINT USING POUR LITTERAUX  **
00020 REM **          V.66                **
00030 REM **
00040     INPUT A$, B$, C$, D$
00050      FOR I=1 TO 5
00060     PRINT '....5....0';
00070      NEXT I
00080     PRINT
00090 REM **
00100 PRINTUSING110,A$,B$,C$,D$
00110 :  VALEURS LUES..########    ###.##!!!!   +###.#####
00120  STOP V.66
00130 END
```

```
run

   ?   'bonjour','aujourd''hui','1234567890','a**b**c+d-x'
..|..5....0....5....0....5...0....5...0....5...0
  VALEURS LUES.BONJOUR    AUJOURD'HU  123456789
  VALEURS LUES.A**B**C+D
EDIT
```

3. Détails pour l'impression des valeurs numériques

- Si une image est réservée au signe, il apparaît dans cette position.
- les # inutiles sont représentés par des blancs.
- Le cadrage se fait sur la position (présumée) du point.
- La partie fractionnaire est arrondie.
- Si l'image est trop courte, compte tenu d'un éventuel signe - , des astérisques apparaissent à la place du nombre.

```
00010 REM ** UTILISATION D'IMAGES **
00020 REM **       V.69             **
00030 REM **
00040 DATA 1,12, 5, 125, 4, 51
00050 DATA 0, 12345, 8,-678, 2,-35
00060 DATA -1
00070       DIM A(12)
00080       FOR I=1 TO 12
00090         READ J
00100 IFJ<0THEN130
00110         READ A(J)
00120       NEXT I
00130 REM **
00140     FOR I=1 TO 9 STEP 3
00150 PRINTUSING160,A(I),A(I+1),A(I+2)
00160 : TAB "A"  +###.## ; - ##.#!!!!
00170      NEXT I
00180 END
EDIT
```

```
      run

 TAB "A"   + 12.00 ; - -3.5E+01
 TAB "A"   +  0.00 ; -
 TAB "A"   + 51.00 ; - 12.5E+01
 TAB "A"   +******* ; -
 TAB "A"   +  0.00 ; - -6.8E+02
 TAB "A"   +  0.00 ; -
EDIT
```

§ 6.2.5 Autre exemple

```
00010 REM **  TABLE DE FONCTIONS   **
00020 REM **         V.05           **
00030 REM **
00040     INPUT X0,H,N
00050 PRINTUSING190
00060 PRINTUSING200
00070 PRINTUSING210
00080 PRINTUSING200
00090 PRINTUSING190
00100 REM **
00110   FOR I=1 TO N
00120     X = X0 + H*(I-1)
00130 PRINTUSING220,X,SIN(X),COS(X),EXP(X),HSN(X),HCS(X)
00140     IF I/5 <> INT(I/5) THEN 100
00150 PRINTUSING190
00160 NEXT I
00170 REM **
00180 PRINTUSING190
00190 :  *-------++------------+------------++------------+------------+------------+*
00200 :  |       ||            |            ||            |            |            ||
00210 :  |   X   ||    SIN(X)  |   COS(X)   ||   EXP(X)   |   SH(X)    |   CH(X)    ||
00220 :  |  #.## ||  ##.###### |  ##.###### ||  ##.###### |  ##.###### |  ##.###### ||
00230 STOP V.05
00240 END
```

```
EDIT  run

   ?   0, 0.02, 7
   *-------++------------+------------++------------+------------+------------+*
   |       ||            |            ||            |            |            ||
   |   X   ||    SIN(X)  |   COS(X)   ||   EXP(X)   |   SH(X)    |   CH(X)    ||
   |       ||            |            ||            |            |            ||
   *-------++------------+------------++------------+------------+------------+*
   |  0.00 ||   0.000000 |  1.000000  ||  1.000000  |  0.000000  |  1.000000  ||
   |  0.02 ||   0.019999 |  0.999800  ||  1.020202  |  0.020001  |  1.000199  ||
   |  0.04 ||   0.039989 |  0.999200  ||  1.040811  |  0.040011  |  1.000799  ||
   |  0.06 ||   0.059964 |  0.998201  ||  1.061836  |  0.060036  |  1.001800  ||
   |  0.08 ||   0.079915 |  0.996802  ||  1.083287  |  0.080085  |  1.003201  ||
   *-------++------------+------------++------------+------------+------------+*
   |  0.10 ||   0.099833 |  0.995004  ||  1.105171  |  0.100167  |  1.005004  ||
   |  0.12 ||   0.119712 |  0.992809  ||  1.127497  |  0.120288  |  1.007208  ||
   *-------++------------+------------++------------+------------+------------+*
EDIT
```

CHAPITRE 7
SOUS-PROGRAMMES DU BASIC

Section 7.1
FONCTION UTILISATEUR

§ 7.1.1 Introduction

- La notion de sous-programme est fondamentale. Elle offre la facilité de découper de gros programmes en unités plus petites et relativement indépendantes que l'on appelle : "sous-programmes".
- Un sous-programme n'est écrit qu'une seule fois quelque soit le nombre de fois qu'on l'utilise.
- Un sous-programme peut être mis au point indépendamment.

Restriction BASIC

Les transmissions de données, à l'aller comme au retour, entre le programme et le sous-programme, sont entièrement à la charge du programmeur.

§ 7.1.2 Classes de sous-programmes

On distingue :

1. La fonction "utilisateur"
   - Définie par une instruction.
   - Utilisée comme une fonction incorporée.
   - Pour laquelle la transmission de donnée est prise en charge par le système.

2. Le sous-programme proprement dit
   - Activé par une instruction spécialisée : GOSUB
   - Pour lequel les transmissions sont à la charge du

programmeur.

§ 7.1.3 Définition d'une fonction-utilisateur

1. Rôle et forme de DEF

   n° DEF ϕ FNα (v) = exp-arithm.

   Définit, sous le nom FNα , une fonction utilisateur dont v est le paramètre.

2. Règles principales

   - α est l'une des 29 lettres : A, B, ..., ✱
   - v est le paramètre de la fonction. C'est un nom qui n'a signification qu'à l'intérieur de l'instruction où il figure la donnée qui sera transmise et avec laquelle l'exp-arithm. sera effectivement réalisée. Il ne doit y avoir qu'un seul paramètre.
   - exp-arithm. : doit suffire à exprimer la totalité de la fonction. Tous les opérandes doivent avoir été définis. Ils peuvent être des références à d'autres fonctions utilisateurs définies antérieurement.

§ 7.1.4 Utilisation des fonctions-utilisateurs

1. Place de l'instruction DEF

   - L'instruction DEF doit apparaître avant toute utilisation de la fonction.
   - En particulier, si une fonction fait appel à une autre fonction, la définition de la fonction appelée doit précéder celle de la fonction appelante.

2. Emploi d'une fonction-utilisateur

   - Comme pour une fonction incorporée : par une référence de fonction, avec un argument.
   - Cet argument peut être une expression dont la valeur se substitue automatiquement au paramètre

au moment de la réalisation de la fonction.

- Au retour, la valeur de la fonction remplace automatiquement la référence de fonction.

3. Exemple

```
 0100 REM **  FONCTIONS UTILISATEUR  **
00110 REM **          V.72            **
00120 REM **
00130 REM ** DEFINITION DES FONCTIONS
00140     DEF FNS(V) = SQR(V*V+1)
00150     DEF FNT(V) = SQR(V*V-1)
00160     DEF FNQ(V) = (1+V)/(1-V)
00170 REM **
00175     DEF FND(X) = LOG(X + FNS(X))
00180     DEF FNC(X) = LOG(X + FNT(X))
00190     DEF FNF(X) = LOG(SQR(FNQ(X)))
00200 REM ** IDENTITES FONCTIONNELLES **
00210 REM **
00220       FOR X=2 TO 4
00230          (U = FNC(X)
00240          V = FND(X)
00250          W = FNF(1/X)
00255       PRINT X;1/X
00260    PRINT USING 265,U,HCS(U),V,HSN(V),W,HTN(W)
00265 : ###.######    =====>    ##.######
00270  NEXT X
00300 PRINT 'F I N'
00310 STOP
00320 END
EDIT
```

```
EDIT  run
 2      .5000000
   1.316957    =====>     1.999999
   1.443635    =====>     1.999998
   0.549305    =====>     0.500000
 3      .3333333
   1.762747    =====>     2.999998
   1.818446    =====>     2.999999
   0.346573    =====>     0.333333
 4      .2500000
   2.063437    =====>     3.999997
   2.094712    =====>     3.999998
   0.255412    =====>     0.249999
F I N
EDIT
```

Section 7.2
SOUS-PROGRAMME

§ 7.2.1 Organisation générale d'un sous-programme

1. Définition

Chaque fois qu'un ensemble d'instructions doit être utilisé plusieurs fois, dans un même programme ou dans des programmes différents, on a intérêt à organiser cet ensemble en un sous-programme de façon à l'écrire et le mettre au point une fois pour toutes.

2. Structure générale

On distingue :

- les instructions du sous-programme,
- la procédure d'appel, y compris la transmission de données,
- la procédure de retour, y compris la transmission de données.

La procédure d'appel appartient à l'unité appelante alors que celle de retour appartient à l'unité appelée.

La première instruction d'un sous-programme peut être quelconque alors que la fin logique doit se faire par RETURN.

3. Remarque fondamentale

En BASIC, ces unités ne sont pas indépendantes. Il faudra donc veiller :

- soit à différencier convenablement les noms des variables propres à des unités différentes,
- soit à définir un ensemble de variables communes à plusieurs (ou toutes les) unités.

4. Exemple de sous-programme

```
00010 REM ** SIMULATION DE   " SGN "   *
00020 REM **         V.78              *
00030 REM **
00040   INPUT X
00050 Y = X
00060 GOSUB100
00070       X0 = Y0
00080    PRINT X;X0
00090 GOTO40
00100 REM ** SOUS-PROGRAMME
00110 IFY>0THEN150
00120 IFY<0THEN170
00130  Y0 = 0
00140  RETURN
00150    Y0 = +1
00160    RETURN
00170  Y0 = -1
00180  RETURN
00190 REM **
00200 END
EDIT
```

```
      run
  ?    1.3
 1.299999     1
  ?   -2.6
-2.599999    -1
  ?    0
 0       0
  ?    "
  ?
EDIT
```

§ 7.2.2 Instruction GOSUB

1. Forme et rôle

nº GOSUB nº-ss-prog

Transfert le contrôle au sous-programme identifié par "nº-ss-prog"

Ce sous-programme devient alors actif.

2. Règles

- "nº-ss-prog" doit être le nº de la 1ère instruction du sous-programme.
- Au moment du GOSUB, les transmissions de données doivent avoir été réalisées.
- Le sous-programme reste actif jusqu'à exécution d'un RETURN.

§ 7.2.3 Instruction RETURN

1. Forme et rôle

nº RETURN [commentaire]

En association avec un GOSUB, provoque le retour du contrôle à l'instruction suivant le GOSUB qui a activé le sous-programme auquel ce RETURN appartient.

2. Règles

- Avant exécution d'un RETURN, les résultats du sous-programme doivent avoir été transmis à l'unité appelante.
- Le sous-programme doit être actif, c'est-à-dire réalisé lors d'un appel par GOSUB.
- Dans un même sous-programme, il peut y avoir plusieurs RETURN, chacun mettant fin à une branche du traitement. Un seul sera exécuté.
- Si un sous-programme est exécuté, sans avoir été explicitement activé par un GOSUB, le premier RETURN rencontré provoque l'arrêt de l'exécution avec le message : NO ACT GOSUB .

3. Exemple

```
00010 REM ** SOUS-PROGRAMME  **
00020 REM **      V.71       **
00030 REM **
00040       INPUT A,B
00050 GOSUB60
00060 REM -------------------
00070       PRINT A*B; A/B
00080       RETURN
00090 REM -------------------
00100 GOTO40
00110 END
EDIT
      run
  ?   240,48
 11520     5
 11520     5
00080         0670        NO ACT GOSUB
EDIT  ?
0670          RETURN STATEMENT FOUND BEFORE A GOSUB
EDIT
```

```
      delete 100
EDIT  55  goto 110

EDIT  list
```

```
00010 REM ** SOUS-PROGRAMME  **
00020 REM **      V.71       **
00030 REM **
00040       INPUT A,B
00050 GOSUB60
00055  GOTO 110
00060 REM -------------------
00070       PRINT A*B; A/B
00080       RETURN
00090 REM -------------------
00110 END
EDIT
      run
  ?   240, 48
 11520     5
EDIT
```

```
00010 REM **    GRAPHE D'UNE FONCTION POSITIVE     **
00020 REM **                V.77                   **
00030 REM **
00040   DEF FNA(X) = (2*X*X + 4*X +5)/(X*X+1)
00050 GOSUB120
00060   FOR X=-2 TO 5 STEP .3
00070       U = INT(5*FNA(X))
00080 GOSUB190
00090       PRINT '#'
00100    NEXT X
00110 GOTO260
00120 REM **
00130 REM ** SOUS-PROG.  N. 1
00140     FOR I=1 TO 5
00150     PRINT '1234567890';
00160     NEXT I
00170   PRINT
00180 RETURN
00190 REM **
00200 REM **  SOUS-PROG.  N.2
00210     PRINT 'I';
00220       FOR S=2 TO U
00230       PRINT ' ';
00240     NEXT S
00250  RETURN
00260 PRINT
00270 PRINT 'F I N'
00280 END
EDIT

     run
1234567890123456789012345678901234567890123456789012345678901234567890
I     #
I     #
I     #
I      #
I         #
I              #
I                   #
I                        #
I                            #
I                            #
I                          #
I                        #
I                      #
I                    #
I                  #
I                #
I               #
I               #
I              #
I             #
I             #
I            #
I            #
I           #
I           #

F I N

EDIT
```

§ 7.2.5 Imbrication de sous-programmes

1. Définition

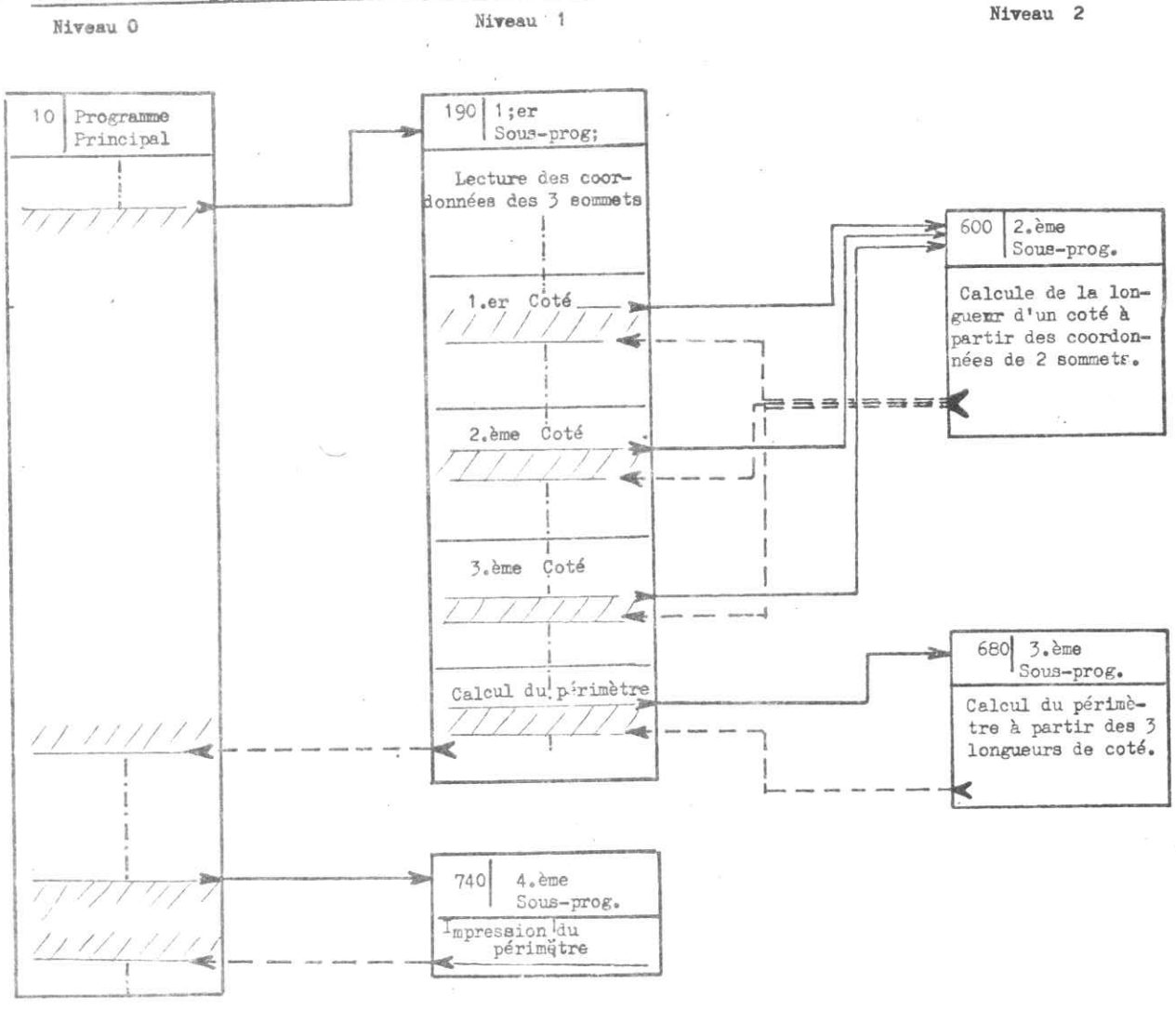
Un sous-programme déjà actif, peut au cours de son exécution, faire appel à un autre sous-programme qui devient alors actif à son tour.

2. Pile des GOSUB

Lorsqu'un RETURN est exécuté, le programme doit pouvoir déterminer le point de retour. Pour cela, le système gère une pile des GOSUB actifs, à chacun étant associé le numéro de l'instruction de retour.
A chaque RETURN le contrôle est transmis au dessus de la pile qui est ensuite effacé, libérant le retour correspondant au GOSUB antérieur.

§ 7.2.6 Exemple d'enchaînement de sous-programmes

1. Schéma des appels et des transmissions



2. Variables du programme et des sous-programmes

Eléments de données	Programme principal	1er ss-prog	2ème ss-prog	3ème ss-prog	4ème ss-prog
Coordonnées des 3 sommets.....		M,N,P			
Coordonnées de 2 sommets.....		(M,N) (N,P) (P,M)	U,V		
Longueur de côté..		K1,H1,G1	K2	A3,B3,C3	
Périmètre.....	PØ	P1		P3	P4

3. Programme

```
00010 REM ** EXEMPLE DE SOUS-PROGRAMMES      **
00020 REM ** AVEC ENCHAINEMENT  DES APPELS   **
00030 REM ** -------- V.75 ---------------- **
00040 REM **
00050 REM ******* PROG. PRINCIPAL *************
00060 REM **
00070       PRINT 'LECTURE DES COORDONNEES DES SOMMETS'
00080 GOSUB190
00090       P0 = P1
00100 REM **  CALCUL DU PERIMETRE
00110       P4 = P0
00120 REM **  IMPRESSIONS
00130 GOSUB740
00140  PRINT '---------------------------------'
00150 GOTO70
00160       PRINT 'F I N'
00170       STOP V.75
00180 REM ***************************************
00190 REM **  1.ER SOUS-PROGRAMME                ***
00200 REM ** LECTURE DES SOMMETS
00210 REM **
00220      DIM M(2), N(2),P(2)
00230      DIM U(2),V(2)
00240     INPUT M(1),M(2)
00250     INPUT N(1),N(2)
00260     INPUT P(1),P(2)
00270 REM **
00280 REM ** APPEL POUR CALCUL DU 1ER COTE     **
00290     U(1)=M(1)
00300     U(2)=M(2)
00310     V(1)=N(1)
00320     V(2)=N(2)
00330 GOSUB600
00340     K1 = K2
00350 REM **
00360 REM ** APPEL POUR CALCUL DU 2.EME COTE   **
00370    FOR I=1 TO 2
00380     U(I)=N(I)
00390     V(I)=P(I)
00400    NEXT I
00410 GOSUB600
00420     H1 = K2
00430 REM **
00440 REM ** APPEL POUR CALCUL DU 3.EME COTE   **
00450    FOR I=1 TO 2
00460     U(I)=P(I)
00470     V(I)=M(I)
00480    NEXT I
00490 GOSUB600
00500     G1 = K2
00510 REM **
00520 REM ** APPEL POUR CALCUL DU PERIMETRE    **
00530         A3=K1
00540         B3=H1
00550         C3=G1
00560 GOSUB680
00570     P1=P3
00580 REM ***
00590       RETURN
00600 REM ***************************************
00610 REM **  2.ME SOUS-PROGRAMME                **
00620 REM **     CALCUL DE LA LONGUEUR DU COTE   **
00630 REM **
00640       R2=V(1)-U(1)
00650       T2=V(2)-U(2)
00660       K2 = SQR(R2*R2 + T2*T2)
00670     RETURN
00680 REM ***************************************
00690 REM **  3.EME  SOUS-PROGRAMME              **
00700 REM **       CALCUL DU PERIMETRE           **
00710 REM **
00720          P3 = A3+B3+C3
00730          RETURN
00740 REM ***************************************
00750 REM **  4.EME SOUS-PROGRAMME               **
00760 REM **    IMPRESSION DU PERIMETRE          **
00770 REM **
00780       PRINT
00790 PRINTUSING800,P4
00800 : LE PERIMETRE VAUT : ###.###
00810       PRINT
00820       RETURN
00830 REM ###########################################
00840       END
EDIT
      run

LECTURE DES COORDONNEES DES SOMMETS
  ?   0,0
  ?   0,1
  ?   1,0

 LE PERIMETRE VAUT :   3.414

---------------------------------
LECTURE DES COORDONNEES DES SOMMETS
  ?   -1,0
  ?   1,0
  ?   0, 1.73205

 LE PERIMETRE VAUT :   6.000

---------------------------------
LECTURE DES COORDONNEES DES SOMMETS
  ?   1,2
  ?   2,3
  ?   3, 5.6

 LE PERIMETRE VAUT :   8.318

---------------------------------

EDIT
```

CHAPITRE 8
CALCUL MATRICIEL

Section 8.1
PRINCIPES ET GENERALITES

§ 8.1.1 Introduction

- Pour Basic, une matrice est un tableau numérique à 1 ou 2 dimensions.
- Dans le cas de 2 dimensions,
  - le tableau se présente sous la forme de n lignes de p valeurs correspondant à autant de colonnes ;
  - si $n = p$ la matrice est carrée,
    si $n \neq p$ la matrice est rectangulaire.
- Dans le cas de 1 dimension, la matrice se trouve réduite à un vecteur.
- L'élément d'une matrice sera identifié par 1 ou 2 indices selon le cas.

§ 8.1.2 Les matrices du Basic

1. Nom d'une matrice

   C'est un nom de tableau, formé d'une seule lettre :

   A, B,... , #

2. Dimensionnement

   Le dimensionnement initial sera :

   - soit implicite : 10 ou 10 x 10 ,
   - soit explicite dans une instruction DIM précédant toute instruction matricielle faisant référence à la matrice.

   Ce dimensionnement initial est aussi maximal.

3. Redimensionnement

- On pourra modifier le dimensionnement d'une matrice pour l'adapter à des valeurs définies au cours de l'exécution du programme, mais ces nouvelles bornes ne devront pas être supérieures aux bornes du dimensionnement initial.
- Un redimensionnement reste valable tant qu'il n°est pas modifié par un nouveau redimensionnement.

Section 8.2
TRANSMISSION DES MATRICES

§ 8.2.1 Lecture au terminal MAT INPUT

1. Forme et rôle

n° MAT INPUT ..., M $[(x_1 [,x_2])]$, ...

M : une matrice à 1 ou 2 dimensions.

x_1, x_2 : expressions arithm.

Transmet aux éléments de la matrice des valeurs lues au terminal, après avoir, éventuellement, réalisé un redimensionnement aux bornes x_1 et x_2 .

2. Fonctionnement

- La lecture de la 1ère ligne est demandée par un point d'interrogation.
  Frapper alors toutes les valeurs de la 1ère ligne avant de faire "retour-chariot".
- La lecture des autres lignes, sera demandée par un double point d'interrogation. La frappe se fait de la même façon.
- La 1ère ligne de la matrice suivante sera requise par un simple point d'interrogation.

3. Exemple

```
00010 REM **  LECTURE DE MATRICES PAR  INPUT **
00020 REM **              V.81                **
00030 DIM A(3,5), B(2,2), C(3)
00040   MAT INPUT B
00050 PRINT '**********'
00060   PRINT 'U & R'
00070   INPUT U,R
00080 PRINT '**********'
00090   MAT INPUT A(U-1,R+1), C,B
00100 REM **
00110 REM **
00120 PRINT '**********'
00130
00140 END
EDIT
```

```
          run
B ?   101,102,103
  EXCESS101,102
  ??  103,104
 **********
 U & R
   ?   3,3
 **********
A ?   11,12,13,14
  ??  21,22,23,24
C ?   -1,-2,-3
B ?   101,102
  ??  103,104
 **********
```

4. Règles principales

- Si les valeurs d'une ligne de matrice sont trop nombreuses pour être frappées sur une seule ligne d'impression, faire suivre la dernière constante frappée d'une virgule avant le "retour-chariot" et poursuivre la frappe sur la ligne suivante.
- En cas d'erreur, frapper à nouveau la totalité de la ligne de la matrice.
- Les expressions x_1 et x_2 doivent être telles que leur partie entière soit comprise entre 1 et la borne supérieure maximale déclarée implicitement ou explicitement.

§ 8.2.2 Lecture d'une matrice par MAT READ

1. Forme et rôle

nº MAT READ ..., M $[(x_1 [,x_2])]$, ...

M, x_1 et x_2 ont les mêmes significations que précédemment et doivent obéir aux mêmes règles.

Cette fois, les valeurs sont prises dans le fichier incorporé du programme.

2. Fonctionnement

La lecture se fait à partir de la position du pointeur et la valorisation, par lignes consécutives de rang croissant.

3. Exemple

```
      00010 REM ** LECTURE DE MATRICE PAR READ **
      00020 REM **              V.82              **
      00030 REM **
      00040       DIM A(3,5),B(2,2),C(3)
      00050       MAT READ B
      00060     READ U,R
      00070       MAT READ A(U-1,R+1), C,B(1,2)
      00080 REM **------------------------------
      00090       MAT PRINT A,B,C
      00100       PRINT '**********'
      00110       MAT PRINT A;B;C
      00120 REM **------------------------------
B  -->00130 DATA 101,102,103,104
      00140 DATA 3,2
A  -->00150 DATA 11,12,13,21,22,23
C  -->00160 DATA 5,6,7
B  -->00170 DATA 555,666
      00180 END
      EDIT
```

§ 8.2.3 Impression de matrice par MAT PRINT

1. Forme et rôle

nº MAT PRINT ... Ms ...

M est une matrice.

s un séparateur (,) ou (;).

L'impression de M se fait ligne par ligne.

2. Fonctionnement

- Chaque matrice est imprimée ligne par ligne.
- La 1ère ligne débute une nouvelle ligne d'impression ; elle est séparée de la ligne d'impression précédente par 2 lignes blanches.
- Chacune des autres lignes de la matrice est imprimée

à partir du début de la ligne et est séparée de la précédente par une ligne blanche.

3. Exemple

```
00010 REM ** LECTURE DE MATRICE PAR READ **
00020 REM **              V.82              **
00030 REM **
00040     DIM A(3,5),B(2,2),C(3)
00050     MAT READ B
00060   READ U,R
00070     MAT READ A(U-1,R+1), C,B(1,2)
00080 REM **--------------------------------
00090     MAT PRINT A,B,C
00100     PRINT '**********'
00110     MAT PRINT A;B;C
00120 REM **--------------------------------
00130 DATA 101,102,103,104
00140 DATA 3,2
00150 DATA 11,12,13,21,22,23
00160 DATA 5,6,7
00170 DATA 555,666
00180 END
EDIT
```

```
A { 11          12
  { 21          22

B { 555         666

C { 5           6
  { **********
```

```
A { 11    12    13
  { 21    22    23

B { 555   666

C { 5             6
  EDIT
```

4. Règles principales

- Chaque valeur est convertie en format caractère, sous forme étendue ou condensée selon le séparateur utilisé.
- Ces valeurs seront plus ou moins espacées selon que le séparateur est une virgule ou un point-virgule.
- Si une ligne d'impression est trop courte, l'impression se prolonge sur la ligne suivante.

§ 8.2.4 Impression de matrice par MAT PRINT USING

1. Forme et rôle

nº MAT PRINT USING nºimage, ..., M, ...

nº image : # # # ...

M est une matrice.

nºimage doit identifier une instruction-image.

2. Fonctionnement

La matrice est imprimée, ligne par ligne comme précédemment, si ce n'est que la disposition des valeurs

sur la ligne d'impression est commandée par l'image d'impression.

3. Exemple

```
                                                    EDIT  run
00010 REM ** LECTURE DE MATRICE PAR READ **
00020 REM **              V.82                  **        11        12        13
00030 REM **                                                21        22        23
00040     DIM A(3,5),B(2,2),C(3)
00050     MAT READ B
00060   READ U,R
00070     MAT READ A(U-1,R+1), C,B(1,2)                     555       666
00080 REM **------------------------------
00090 MATPRINTUSING100,A,B,C
00100 : ######  ######  ######                              5         6         7
00110     PRINT '**********'                          **********
00120 MATPRINTUSING130,A,B,C
00130 : *** ####### #######
00140 REM **------------------------------              ***       11        12
00150 DATA 101,102,103,104                              ***       13
00160 DATA 3,2
00170 DATA 11,12,13,21,22,23                            ***       21        22
00180 DATA 5,6,7                                        ***       23
00190 DATA 555,666
00200 END                                               ***       555       666

                                                        ***       5         6
                                                        ***       7
                                                    EDIT
```

4. Règles principales

- Si l'image est trop longue, le superflu est ignoré. L'impression de chaque ligne débute avec le commencement de l'image de ligne.
- Si l'image est trop courte, un retour de boule se produit à la fin de l'image et l'image est réutilisée depuis le début pour l'impression de la fin de ligne de la matrice.

Section 8.3
ARITHMETIQUE MATRICIELLE

§ 8.3.1 Affectation matricielle

1. Expression matricielle

Des matrices peuvent être valorisées à partir de 3 formes d'expressions matricielles :

- Expression de valorisation à l'aide des fonctions CON, IDN ou ZER.
- Expression arithmétique à 2 opérandes : addition, soustraction, multiplication matricielles, multiplication par un scalaire.
- Expression de transformation d'une matrice à l'aide des fonctions TRN ou INV.

2. Instruction d'affectation matricielle

n° MAT cible = source

source : l'une des 3 formes d'expressions matricielles définies précédemment.

cible : une matrice.

3. Règles principales

- Dans le cas des 2 dernières formes d'expression, la matrice cible doit être différente des matrices opérandes. Ainsi :

  MAT A = A + B

  est interdit.
- Lors de la réalisation de l'instruction, les matrices concernées doivent avoir un dimensionnement identique ou convenable.

§ 8.3.2 Valorisation par fonction

1. Forme générale

n° MAT cible = fonction $\left[(x_1\left[,x_2\right])\right]$

fonction : CON, IDN ou ZER

$(x_1[,x_2])$: précise le dimensionnement de la cible.

2. Fonctionnement

CON : tous les éléments prennent la valeur 1.

IDN : la matrice doit être carrée ; restitue une matrice unité.

ZER : tous les éléments prennent la valeur Ø.

3. Exemple

```
00010 REM ** VALORISATION DE MATRICE  **
00020 REM **           V.83           **
00030 REM **
00040     DIM A(5,5), B(5,5)
00050 PRINT '***'
00060 PRINT 'N.1'
00070 MATPRINTUSING80,A
00080 : ##   ##   ##   ##   ##
00090     MAT A = CON(3,4)
00100 MATPRINTUSING80,A
00110 REM **-----------------------------
00120 PRINT '***'
00130 PRINT 'N.2'
00140     MAT A = ZER
00150 MATPRINTUSING80,A
```

run

```
***
N.1
```

A(5,5)

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

CON(3,4)

1	1	1	1
1	1	1	1
1	1	1	1

```
***
N.2
```

ZER(3,4)

0	0	0	0
0	0	0	0
0	0	0	0

§ 8.3.3 Arithmétique matricielle

1. Addition-soustraction

nº MAT $M_1 = M_2 \pm M_3$

La somme (ou la différence) de M_2 et M_3, au sens matriciel, est affectée à la matrice M_1 .

2. Produit par un scalaire

n° MAT M_1 = (expres.) * M_2

expres. : - est d'abord évalué ;

- puis sa valeur sert à multiplier chacun des éléments de M_2 , le résultat étant affecté à M_1 ;
- doit être placé entre parenthèses.

3. Produit matriciel

n° MAT M_1 = M_2 * M_3

Le produit matriciel de M_3 par M_2 est affecté à M_1 .

Les tableaux doivent être à 2 dimensions et leurs dimensionnements doivent être compatibles.

4. Remarque importante

Dans tous les cas, la matrice cible M_1 doit être distincte de M_2 et M_3 .

5. Exemples

```
00160      MAT A = IDN(2,2)
00170 PRINT '***'
00180 PRINT 'N.3'
00190      MAT B = ZER(2,2)
00200        FOR I=1 TO 3
00210        MAT B = (I)*A
00220 MATPRINTUSING80,B
00230        NEXT I
00240 REM **--------------------
00250 PRINT '***'
00260 PRINT 'N.4'
00270      MAT B = (B(2,2))*A
00280 MATPRINTUSING80,B
00290 END
EDIT
```

```
***
N.3

 1   0
 0   1

 2   0
 0   2

 3   0
 0   3
***
N.4

 3   0
 0   3
EDIT
```

```
00010 REM ** ARITHMETIQUE MATRICIELLE **
00020 REM **          V.84              **
00030 REM **
00040      DIM A(5,5), B(5,5), C(5,5)
00050 REM PRODUIT MATRICIEL
00060 REM **
00070      MAT INPUT A(4,3)
00080      MAT INPUT B(3,2)
00090      MAT C = ZER(4,2)
00100      MAT C = A*B
00110 PRINT
00120 PRINT 'PRODUIT MATRICIEL'
00130 PRINT
00140 MATPRINTUSING150,A,B,C
00150 :  ###.####  ###.####  ###.####
00160 REM **----------------------------
```

```
EDIT  run
A ?   0.1, 0 , 0.1
  ??  0.3, 0.2, 0.1
  ??  0 , 0.1, 0.2
  ??  0.1, 0.1, 0
B ?   8,6
  ??  0,3
  ??  5,3
```

```
PRODUIT MATRICIEL

A   0.1000    0.0000    0.1000
    0.3000    0.2000    0.1000
    0.0000    0.1000    0.2000
    0.1000    0.1000    0.0000

B   8.0000    6.0000
    0.0000    3.0000
    5.0000    3.0000

C   1.3000    0.9000
    2.9000    2.7000
    1.0000    0.9000
    0.8000    0.0000
```

§ 8.3.4 Transformation d'une matrice

1. Transposition d'une matrice

   n° MAT M_1 = TRN (M_2)

   La transposée de M_2 est affectée à M_1 . Les dimensionnements de M_1 et M_2 doivent être compatibles.

2. Inversion d'une matrice

   n° MAT M_1 = INV (M_2)

   L'inverse de M_2 est affecté à M_1 ; M_1 et M_2 doivent être carrées et M_2 régulière.

   La précision des résultats n'est pas garantie ; elle dépend de la précision et de la stabilité de la matrice.

3. Fonction DET

   Son argument doit être une matrice carrée dont elle restitue le déterminant.

4. Exemple

```
00170 REM **  DETERMINANT ET INVERSION
00180       MAT INPUT A(3,3)
00190       D = DET(A)
00200       MAT C = ZER(3,3)
00210       MAT C = INV(A)
00220 PRINT
00230 PRINT 'DETERMINANT DE A ='; D
00240 PRINT
00250 PRINT 'INVERSE DE A '
00260 MAT PRINT C
00270 REM **---------------------------
00280 REM ** RESOLUTION DE SYSTEME
00290    DIM X(3,1), S(3,1)
00300       MAT INPUT A(3,3)
00310       MAT INPUT S
00320    MAT C = ZER(3,3)
00330       MAT C = INV(A)
00340       MAT X = C*S
00350 PRINT
00360 PRINT 'RESOLUTION DE SYSTEME'
00370 PRINT
00380 PRINT 'VECTEUR SOLUTION  "X"'
00390 MATPRINTUSING150,X
00400 END
EDIT
 ?
```

A
```
     2,3,-1
??   3,5,2
??   1,-2,-3
```

```
DETERMINANT DE A =  21.99998

INVERSE DE A

 - 0.5000     0.5000     0.5000

   0.5000   - 0.2273   - 0.3182

 - 0.5000     0.3182     0.0455
 ?
```

A
```
     2,3,-1
??   3,5,2
??   1,-2,-3
```

S
```
?    1
??   8
??   -1
```

```
RESOLUTION DE SYSTEME

VECTEUR SOLUTION  "X"
```

X
```
    3.0000

 - 1.0000

    2.0000
```

```
EDIT
```

CHAPITRE 9
LES FICHIERS DU BASIC

Section 9.1
PRINCIPES ET GENERALITES

§ 9.1.1 Définitions

1. Enregistrement

Ensemble d'éléments de données numériques et/ou littérales, réunis par un lien logique d'appartenance à une même entité .
En général, cet ensemble est transmis en totalité dans une seule instruction de transmission.

2. Fichier

- Les enregistrements placés consécutivement sur le support externe (disque) constituent le fichier.
- Ce fichier est placé dans la bibliothèque de l'utilisateur.
- Il n'est accessible que par programme.

§ 9.1.2 Utilisation des fichiers

1. Cas principaux d'utilisation

Pour des ensembles des données volumineux, utilisés par plusieurs programmes ou plusieurs fois par un même programme.
Pour transmettre des résultats entre des programmes exécutés consécutivement.

2. Gestion de fichier

- Au départ : ouverture.
- A la fin : fermeture.
- En cours de programme :

la création par PUT

la lecture par GET

§ 9.1.3 Exemple d'utilisation de fichier

1. Création

On calcule les racines carrées, troisième et quatrième des entiers consécutifs tant que la somme des racines reste inférieure à 7.
Une fois les calculs et écritures terminés, on place un enregistrement supplémentaire destiné à marquer la fin du fichier.

```
00010 REM ** EXEMPLE D'UTILISATION DE FICHIER **
00020 REM **         V.90                      **
00030 REM **
00040 REM **    1) CREATION, OUVERTURE IMPLICITE
00050 REM **
00060       N=1
00070       X=SQR(N)
00080       Y=N**(1/3)
00090       Z=N**(0.25)
00100 IFX+Y+Z>7THEN140
00110         PUT 'RAC',N,X,Y,Z
00120       N=N+1
00130 GOTO70
00140 REM **
00150 REM **    2) FERMETURE EXPLICITE
00160 REM **
00170       {PUT 'RAC',999,999,999,999
00180       {CLOSE 'RAC'
00190 REM **
```

2. Relecture

Après avoir fermé puis "rebobiné" le fichier, on boucle sur la lecture afin d'imprimer le contenu de RAC jusqu'à rencontrer la "marque de fin de fichier" qui indique la fin des lectures.

```
00200 REM **    3) RELECTURE
00210 REM **
00220       DIM A(4)
00230       RESET 'RAC'
00240        MAT GET 'RAC', A
00250 IFA(1)=999THEN290
00260 MATPRINTUSING270,A
00270 : N= ### , RACINES : ##.####    ##.####    ##.####
00280 GOTO240
00290 REM **
00300 REM **    4)  FIN DES LECTURES
00310 REM **
00320        PRINT 'FIN'
00330 STOP
00340 END
EDIT
```

```
 N=    1 , RACINES :  1.0000     1.0000     1.0000

 N=    2 , RACINES :  1.4142     1.2599     1.1892
      ---         ---          ---          ---
 N=    8 , RACINES :  2.8284     2.0000     1.6818

 N=    9 , RACINES :  3.0000     2.0801     1.7320
FIN
EDIT
```

Section 9.2
GESTION D'UN FICHIER

§ 9.2.1 Nom d'un fichier

- Formé de 3 caractères au plus :
  - l'initiale est une lettre,
  - les 2 autres alphanumériques et facultatifs.
- Le tout placé entre apostrophes

§ 9.2.2 Ouverture

1. Rôle de l'ouverture

   Associer le nom du fichier à un espace-disque et à

ses caractéristiques, en particulier le sens de la transmission : lecture ou écriture.

2. Réalisation

Elle est implicite lors du 1er GET ou PUT concernant le fichier, selon qu'il est utilisé en lecture ou en écriture.

§ 9.2.3 Fermeture

1. Rôle de la fermeture

Rompre l'association en cours et par le fait même autoriser une nouvelle association.

2. Instruction CLOSE

n° CLOSE nom-fichier [, nom fichier ...]

3. Règles

- Si le fichier n'était pas ouvert, l'instruction est ignorée.
- Si dans un programme on veut modifier le sens de transmission d'un fichier, il faut d'abord le fermer par CLOSE.

§ 9.2.4 Rebobinage

1. Rôle

Replacer le fichier sur le 1er enregistrement, tout en le laissant ouvert.

2. Instruction RESET

n° RESET nom-fichier, ...

Section 9.3
TRANSMISSIONS DE DONNEES

§ 9.3.1 Généralités

1. Création ou écriture
   - Les données sont transmises du programme sur l'espace-disque,
     - soit comme résultats du traitement,
     - soit à partir du terminal, via le programme.
   - La 1ère écriture réalise l'ouverture.
   - La création se fera depuis le début du fichier et les enregistrements sont placés consécutivement, l'un après l'autre, sans aucun repère.

2. Lecture
   - Les données, en provenance du fichier, sont mises à la disposition du programme.
   - La lecture commence toujours au 1er enregistrement et se poursuit séquentiellement selon la chronologie des lectures. Il est impossible d'accéder à un enregistrement sans avoir exploré tous ceux qui le précèdent.

§ 9.3.2 Instruction d'écriture: PUT

1. Forme

   nº PUT nom-fichier, liste-valeurs

2. Règles
   - Liste-valeurs représente des constantes, variables ou expressions.
   - Le 1er PUT réalise l'ouverture en écriture.

§ 9.3.3 Instruction de lecture : GET

1. Forme

   n° GET nom-fichier, liste-variables

2. Règles

   - Liste-variables représente les variables concernées par la lecture.
   - Le fichier doit avoir été préalablement créé.
   - Un indice lu peut immédiatement servir pour la lecture suivante :

     GET 'F12' , I, A(I)

   - Une valeur numérique affectée à une variable simple précision pourra être tronquée.

§ 9.3.4 Cas particulier des matrices

1. Rôle

   Transmission globale des matrices

2. Forme

   n° MAT {GET / PUT} nom-fichier, ..., nom-matrice, ...

3. Règles

   - En lecture, le nom-matrice peut être accompagné d'un redimensionnement.
   - La transmission se fait par lignes de matrice.

§ 9.3.5 Exemple

```
00010 REM **    RECHERCHE  SEQUENTIELLE    **
00020 REM **    DANS LE FICHIER  'RAC'     **
00030 REM **              V.94             **
00040 REM **
00050  DIM S(4)
00060       INPUT N
00070 IFN<=0THEN250
00080       INPUT R
00090 IFR<2THEN220
00100 IFR>4THEN220
00110       RESET 'RAC'
00120          FOR I=1 TO N
00130          MAT GET 'RAC',S
00140 IFS(1)=999THEN190
00150          NEXT I
00160 PRINTUSING170,R,N,S(R)
00170 : RACINE ##.EME  DE ###   =   ##.####
00180 GOTO60
00190  PRINT
00200  PRINT 'N EST TROP GRAND'
00210 GOTO60
00220 PRINT
00230 PRINT 'R EST HORS LIMITES'
00240 GOTO80
00250 PRINT
00260 PRINT 'FIN'
00270 END

EDIT  run
  ?   2
  ?   2
 RACINE  2.EME  DE   2  =   1.4142
  ?   10
  ?   10

R EST HORS LIMITES
  ?   3

N EST TROP GRAND
  ?   5
  ?   3
 RACINE  3.EME  DE   5  =   1.7100
  ?   5
  ?   4
 RACINE  4.EME  DE   5  =   1.4953
  ?   8
  ?   4
 RACINE  4.EME  DE   8  =   1.6818
  ?   -1

FIN
```

CHAPITRE 10
LE SYSTEME I.T.F.

Section 10.1
GENERALITES

§ 10.1.1 Le système I.T.F.

1. Rôle du système

Le système I.T.F. (Interactive Terminal Facility) est destiné à gérer un ensemble de terminaux en assurant un échange d'informations entre

- un utilisateur au terminal
- et l'ordinateur.

A un instant donné, un seul terminal dispose des ressources de l'ordinateur pour réaliser son programme stocké dans la mémoire de l'ordinateur.

2. Modes de fonctionnement de I.T.F.

Il existe 3 modes :

- CONTROL : supervise les 2 autres modes et sert principalement à l'initialisation, à la clôture de session et à la gestion de la bibliothèque de l'utilisateur.
- CALCULATEUR : permet d'utiliser le terminal comme calculateur programmable en PL/I.
- EDIT et TEST : pour la création, la mise au point, l'exécution et la maintenance des programmes et fichiers de l'utilisateur.

3. Remarque fondamentale

On doit donc distinguer :

- d'une part : les commandes du système I.T.F.
- d'autre part : les instructions des langages de programmation, BASIC ou PL/I.

Leur confusion provoquera un message d'erreur et même

l'effacement du programme.

4. Commande I.T.F.

Elle ne comporte pas de numéro de ligne, mais débute par l'indicatif du mode de fonctionnement :

READY en mode CONTROL

CALC en mode CALCULATEUR

EDIT en mode EDIT

TEST en sous-mode TEST .

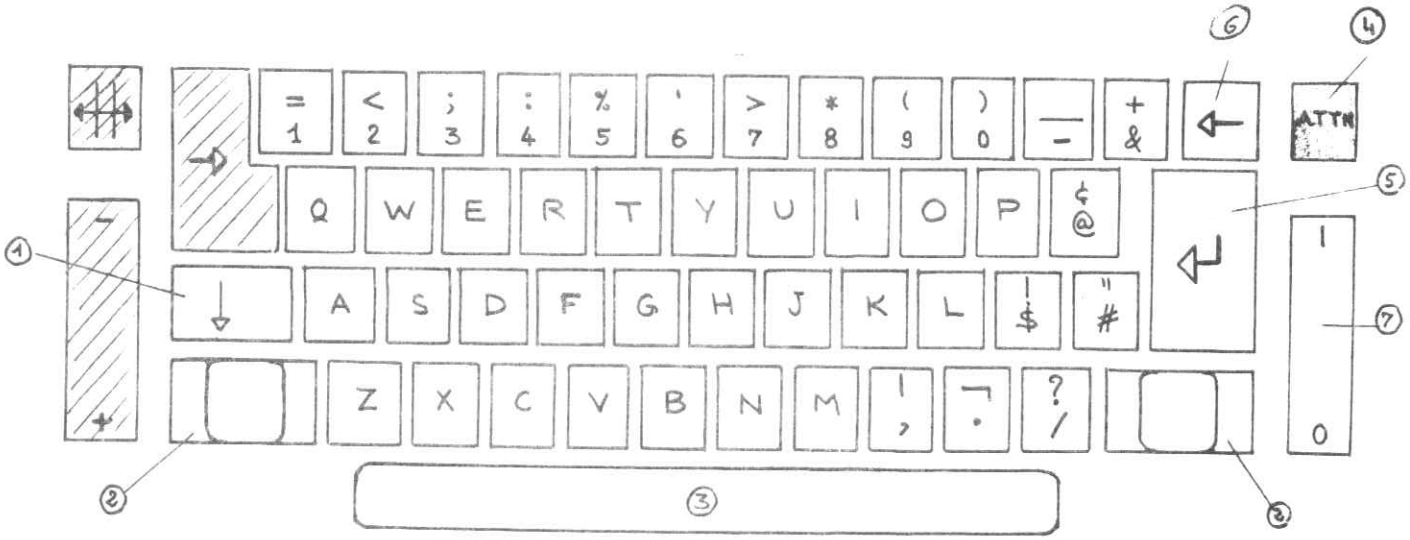
5. Instruction

Elle doit commencer par un numéro de ligne, éventuellement à la suite de l'indicatif EDIT.

§ 10.1.2 Clavier du terminal 2741

1. Caractères du clavier

- On retrouve l'alphabet des langages de programmation : lettres, chiffres et caractères spéciaux.
- Certaines touches comportent 2 caractères. Le caractère supérieur sera sélecté en plaçant le clavier en majuscules, touches ① ou ② .
- Les lettres majuscules et minuscules sont équivalentes.



2. Mise sous tension

Touche à bascule ⑦

0 : hors tension

1 : sous tension

3. Touches de liaison

- Retour chariot ⑤
  - Transmet à l'ordinateur la ligne qui vient d'être frappée et avance le papier d'une ligne.
  - Toute information frappée au terminal doit être transmise par R.C.
- ATTN ④

  Permet d'interrompre immédiatement toute exécution d'une commande ou d'un programme.

4. Autres touches

- Retour-arrière ⑥

  Déplace la boule vers la gauche en "effaçant" les caractères de la ligne.
- Touches de tabulation : interdites.

5. Remarque

- Les réponses de l'ordinateur sont frappées en majuscules.
- L'utilisateur utilisera en général les lettres minuscules.

Section 10.2
CONSTRUCTION D'UN PROGRAMME

§ 10.2.1 Définition de la session

1. Contrôles préliminaires

   - Vérifier l'alimentation du papier et la position des butées définissant la longueur de ligne (120 positions).
   - Mettre sous tension.
   - Renvoyer la boule en début de ligne par un R.C.

2. Ouverture de la session

   - Identification de l'utilisateur par la commande "logon"

     logon ⌀ clé [R.C.]

   - clé identifie l'utilisateur qui doit être connu d'I.T.F.
   - I.T.F. répond par READY et un message. On se trouve alors en mode CONTROL.
   - En cas d'erreur, donner la clé convenable.

```
1
logon abc
0187        INV USERID
LOGON ?
0187        USERID ABC  NOT RECOGNIZED BY SYSTEM
LOGON smm
THANK YOU.          DATE 22/10/75       TIME 10.14.07
READY
```

3. Fin de session

   - Avant de quitter le terminal, l'utilisateur rompt la liaison avec l'ordinateur.
   - Revenir au mode CONTROL.
   - Frapper la commande "logoff".
   - Mettre le terminal hors-tension.

```
  ?
EDIT  end
READY logoff
LOGGED OFF AT  11.35.36  22/10/75
#SESSION DURATION  01.21.29  CPU TIME USED  101870/300THS SEC.
```

§ 10.2.2 Le mode CONTROL

1. Rôle

   - Passage obligatoire entre les autres modes, il est caractérisé par l'indicatif READY.
   - Il est également le mode de la gestion de la bibliothèque.

2. Passage en mode CONTROL

   - Le mode CONTROL est automatique à l'ouverture de la session.
   - En cours de session, pour revenir au mode CONTROL, il suffit de mettre fin au mode en cours par une commande END.

   Par exemple :
   ```
   TEST end
   EDIT end
   READY
   ```

§ 10.2.3 Création du programme

1. Passage au mode EDIT

   - L'écriture, l'exécution et la maintenance d'un programme se font en mode EDIT.
   - L'appel du mode EDIT se fait en indiquant le nom du programme et le langage utilisé.

   ```
   READY ∅ edit ∅ nom ∅ basic
   EDIT ∅ ∅
   ```

   - Le nom du programme doit comporter de 1 à 3 caractères :
     - le 1er : lettre ,
     - les 2e et 3e : lettres ou chiffres.

```
READY edit bas basic
0172        COLL NM TOO LONG
READY ?
0172        COLLECTION NAMES MUST BE 3 CHARACTERS OR LESS
READY bas basic
0103        INV CMT
READY ?
0103        COMMAND INVALID IN CNTL MODE
READY edit bas basic
```

2. Ecriture du programme

- A partir du moment où I.T.F. est passé en mode EDIT, on frappe les instructions, ligne par ligne, à la suite de l'indicatif EDIT.
- Chaque ligne doit avoir un numéro et être transmise par R.C.
- Chaque instruction transmise est immédiatement contrôlée avant d'être stockée dans la zone de travail réservée à l'utilisateur.
- Si l'instruction comporte une erreur de syntaxe :
  - elle n'est pas stockée,
  - un message "précise" l'erreur détectée,
  - l'utilisateur doit réécrire convenablement l'instruction.

```
EDIT  210    impur x
0615        MSNG = OR UNID STM
EDIT  ?
0615        MISSING EQUAL SIGN IN ASSIGNMENT STATEMENT OR UNIDENTIFIABLE STATEMENT TYPE
EDIT  210    input x
EDIT  220      if x=999 then 999
EDIT  230    input h,x9
EDIT  240 rem ****************************************
EDIT  300    if x<x9 then 150
EDIT  310    y = ((a*x h)*x+c)*x+d
0603        SYN ERR EXPR
EDIT  ?
0603        SYNTAX ERROR IN AN EXPRESSION
EDIT  310    y = ((a*x+b)*x+c)*x+d
EDIT  330    x = x+h
EDIT  340    goto 300
```

3. Numérotation des lignes

- Elle est obligatoire pour les instructions du programme et interdite pour les commandes.
- Le numéro peut comporter jusqu'à 5 chiffres.
- Il faut prévoir un pas suffisant afin de permettre l'insertion de nouvelles instructions.

§ 10.2.4 Listage du programme

En mode EDIT, le programme contenu dans la zone de travail peut être listé totalement ou partiellement.

EDIT ß ß list : totalité du programme

list ß n1 : la ligne de numéro n1

list ß n1 ß n2 : la partie du programme entre les lignes n1 et n2 incluses.

```
EDIT  list
00100 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00110 REM **
00200   INPUT A,B, C
00210   INPUT X
00220     IF X=999 THEN 999
00230   INPUT H,X9
00240 REM ****************************************
00300   IF X<X9 THEN 150
00310   Y = ((A*X+B)*X+C)*X+D
00330   X = X+H
00340   GOTO 300
00999 STOP
EDIT
```

§ 10.2.5 Sauvegarde en bibliothèque

1. La bibliothèque

   A chaque utilisateur identifié par une clé est associée une bibliothèque dans laquelle il peut sauvegarder indéfiniment ses programmes et fichiers.

2. Commande save

   - En mode EDIT, copie dans la bibliothèque de l'utilisateur, le programme qui se trouve en zone de travail, dans l'état où il se trouve, sous le nom donné à la création.
   - L'ancienne version, de même nom, est détruite.

3. Commande save nom'

   Si nom' est différent du nom donné à la création, on sauvegarde une autre version du programme tout en conservant l'ancienne.

```
EDIT  save
EDIT  save bbb
EDIT
      end
READY listcat
FREE TRACKS      017
BAS    BASIC              BBB     BASIC
READY
```

4. Remarque

Il faut prendre la précaution de sauvegarder un programme :

- immédiatement après l'avoir écrit ou même en cours d'écriture s'il est long,
- après correction du programme.

§ 10.2.6 Exécution du programme

1. Programme en zone de travail

Si le programme vient d'être écrit, la commande "run" à la suite d'EDIT provoque la compilation puis l'exécution.

2. Programme en bibliothèque

Il faut d'abord introduire le programme dans la zone de travail :

```
READY edit nom basic
EDIT run
```

3. Arrêt de l'exécution

Il peut être programmé :

- définitif : instructions STOP ou END,
- temporaire : instruction PAUSE,

ou impromptu : appuyer sur la touche ATTN.

§ 10.2.7 Modification du programme

1. Principe

Il doit être en zone de travail et I.T.F. doit fonc-

tionner en mode EDIT.

2. En cours de frappe d'une instruction

- Avant transmission par R.C., la touche retour-arrière permet de revenir à gauche pour "effacer" des caractères erronés.
- Ensuite, on doit reprendre totalement la frappe à partir du dernier caractère effacé.

```
EDIT  100 rem ** calcul de valeur
                          la valeur d'un polynome  **
EDIT  110 rem **
EDIT  200   input a b,c
                  b,c
EDIT  210   impur x
```

3. Effacement de ligne

La commande "delete" permet d'effacer des lignes du programme stocké en zone de travail.

EDIT delete ↵ n1 : efface la ligne de nº n1

delete ↵ n1 ↵ n2 : efface toutes les lignes depuis n1 à n2 inclus.

```
EDIT

      delete 240
EDIT  list 240

0140        IN NOT FIN
EDIT  ?
0140        LINE NUMBER 00240  NOT FOUND IN COLLECTION
```

```
EDIT  list 230 300

00230   INPUT H,X0
00300     IF X>X0 THEN 210

EDIT
```

4. Insertion de ligne

En EDIT, frapper la ligne en lui donnant un numéro tel que sa place dans la suite des numéros corresponde à la place de l'instruction dans le déroulement du programme.

```
→EDIT  300      if x>x9 then 210
 EDIT  save
 EDIT  list 300 340
 00300      IF X>X9 THEN 210
 00310    Y = ((A*X+B)*X+C)*X+D
 00330    X = Y+H
 00340    GOTO 300
 EDIT
```

5. Substitution de ligne

Il suffit de frapper la nouvelle forme de l'instruction avec le n° de l'instruction que l'on veut remplacer.

```
 EDIT

       list 200
→00200    INPUT A,B,C

→EDIT  200    input a,b,c,d

 EDIT  save
```

6. Remarque

Après modification, sauvegarder la nouvelle version.

§ 10.2.8 Renumérotation

1. Principe

Lorsqu'un programme a subi de nombreuses modifications, il peut être souhaitable de renuméroter ses lignes totalement ou partiellement.
Les n°s de débranchement sont automatiquement ajustés.

2. Formes de la commande "renum"

Commande	Portée	1er n°	Pas
renum	totalité du programme	10	10
renum ⊬ n1	"	n1	10
renum ⊬ n1 ⊬ n2	"	n1	n2
renum ⊬ n1 ⊬ n2 ⊬ n3	à partir de l'ancien n° n3 qui devient →	n1	n2

3. Exemples

```
      renum
EDIT  list
00010 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00020 REM **
00030   INPUT A,B,C,D
00040   INPUT X
00050 IFX=999THEN130
00060   INPUT H,X9
00070 IFX>X9THEN40
00080   Y = ((A*X+B)*X+C)*X+D
00090 PRINTUSING100,X,Y
00100 : X= ##.##   Y= ###.####
00110   X = X+H
00120 GOTO70
00130 STOP
EDIT
```

```
      renum 50
EDIT
      list
00050 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00060 REM **
00070   INPUT A,B,C,D
00080    ← ATTN
EDIT
```

```
      renum 10 20
EDIT  list

00010 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00030 REM **
00050   INPUT A,B,C,D
00070   INPUT X    ← ATTN
```

```
EDIT  renum 100 5 30
EDIT  list

00010 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00100 REM **
00105   INPUT A,B,C,D
00110   INPUT X

EDIT
```

Section 10.3
GESTION DE LA BIBLIOTHEQUE

§ 10.3.1 La bibliothèque

C'est un espace disque mesuré en cylindres de 20 pistes identifié par la clé de l'utilisateur.
Il est utilisé pour stocker de façon permanente les programmes et fichiers de l'utilisateur.
Les programmes y sont sauvegardés par une commande "save" alors que les fichiers y sont placés lors de leur création.

§ 10.3.2 Gestion de la bibliothèque

1. Effacement

   En mode CONTROL, la commande "delete"

   READY ␣ delete ␣ nom

   permet d'effacer le programme ou le fichier identifié par "nom".
   La commande ne peut comporter qu'un seul nom.

2. Répertoire de la bibliothèque

   En mode CONTROL, la commande "listcat" permet de connaître :

   - le nombre de pistes disponibles (FREE TRACKS), un programme de 80 instructions environ occupant une piste de l'espace disque.
   - les noms des programmes et fichiers déjà sauvegardés.

3. Modification d'un nom

   En mode CONTROL,

   READY ␣ rename ␣ ancien ␣ nouveau

   Le nom "ancien" est remplacé par le nom "nouveau".

Ils doivent satisfaire aux règles habituelles et avoir même longueur.

```
      end
READY listcat
FREE TRACKS      013
BAS     BASIC              BC      BASIC              BAL     BASIC
BPL     BASIC              BOL     BASIC              FIL     FILE
READY delete bd

0114        COLL NOT FND
READY ?
0114        COLLECTION NOT FOUND
READY delete bc
READY listcat

FREE TRACKS      014
BAS     BASIC              BAL     BASIC              BPL     BASIC
BOL     BASIC              FIL     FILE

READY
```

§ 10.3.3 La commande "merge"

1. Rôle

   Réalise l'insertion de tout ou partie d'un sous-programme à l'intérieur d'un autre programme.
   Le programme contenant doit être en zone de travail et le sous-programme à insérer, en bibliothèque.

2. Forme

   EDIT merge nom [n1 n2][n3]
   Nom : nom du (sous-)programme à insérer.
   n1,n2,n3 : n°s de ligne.

3. Règles

   1. Le programme contenant doit être en zone de travail.
   2. n1 et n2 sont des numéros de lignes à l'intérieur du (sous-)programme à insérer.
      Si n1 et n2 sont précisés, toutes les lignes de n1 à n2 inclus sont insérées.
      S'ils sont omis, la totalité du (sous-)programme est insérée.

3. n3 représente le n° de la ligne du programme contenant après laquelle l'insertion doit se faire. Si n3 est omis, le sous-programme vient à la fin du programme contenant.
4. Les instructions du (sous-)programme sont renumérotées pour prendre la suite du programme.
5. Par contre, les débranchements ne sont pas ajustés.

4. Pratique de merge

1. Les (sous-)programmes à insérer doivent être complets, mais il faut effacer les END qu'ils pourraient contenir.
2. Veiller à ce que leurs n°s de ligne soient supérieurs au dernier numéro du programme contenant.
3. Placer le (sous-)programme à la fin du programme contenant.
4. Réajuster les GOSUB ou autres GOTO.
5. Placer un END final.

5. Exemple

```
      edit ppp basic
EDIT  list
00100 REM ** PROGRAMME PRINCIPAL **
00110   PRINT 'APPEL DE SP.1'
00120   GOSUB 700
00130   PRINT S
00200 REM ** ----------------------
00210   PRINT 'APPEL DE SP.2'
00220   GOSUB 800
00230   PRINT U
EDIT
      merge sp1

0661        UNDEF STM NUM
EDIT  ?
0661        STATEMENT NUMBER 00800    REFERENCED IN A
EDIT                  STATEMENT NOT DEFINED IN PROGRAM
      delete 120
EDIT  delete 220
EDIT
      merge sp1
EDIT  merge sp2

EDIT  list
00010 REM ** PROGRAMME PRINCIPAL **
00020   PRINT 'APPEL DE SP.1'
00030   PRINT S
00040 REM ** ----------------------
00050   PRINT 'APPEL DE SP.2'
00060   PRINT U
00070 REM ** SOUS-PROGRAMME N.1 **
00080  S = 0
00090  FOR I=1 TO 3
00100  S = S+I**2
00110  NEXT I
00120 RETURN
00130 REM ** SOUS-PROGRAMME N.  2  **
00140   U = 0
00150   FOR J=1 TO 3
00160    U = U+I/J
00170   NEXT J
00180 RETURN
EDIT
      25 gosub 70
EDIT  55 gosub 130
EDIT  65 goto 200
EDIT  200 end
EDIT
      run

 APPEL DE SP.1
  13.99999
 APPEL DE SP.2
  5.500000
```

```
      list
00010 REM ** PROGRAMME PRINCIPAL **
00020   PRINT 'APPEL DE SP.1'
00025 GOSUB 70
00030   PRINT S
00040 REM ** ----------------------
00050   PRINT 'APPEL DE SP.2'
00055 GOSUB 130
00060   PRINT U
00065 GOTO 200
00070 REM ** SOUS-PROGRAMME N.1 **
00080  S = 0
00090  FOR I=1 TO 3
00100  S = S+I**2
00110  NEXT I
00120 RETURN
00130 REM ** SOUS-PROGRAMME N.  2  **
00140   U = 0
00150   FOR J=1 TO 3
00160    U = U+I/J
00170   NEXT J
00180 RETURN
00200 END
```

Section 10.4
MISE AU POINT DE PROGRAMME

§ 10.4.1 Généralités

1. Types d'erreurs

Des erreurs détectées par I.T.F. au cours de l'écriture, la compilation ou l'exécution du programme provoqueront l'impression de messages et l'interruption de la commande en cours.

2. Outils de mise au point

- Pour mettre au point son programme l'utilisateur dispose immédiatement des diagnostics d'erreurs et des résultats d'exécution.
- Néanmoins il sera parfois nécessaire de réaliser une analyse plus fine du déroulement du programme ; pour cela, on utilisera le sous-mode TEST du mode EDIT.

§ 10.4.2 Erreurs d'écriture

1. Cas d'erreur

- Orthographe incorrecte d'un "mot" ou syntaxe erronée d'une instruction ou commande.
- Elle est immédiatement détectée, l'instruction n'est pas enregistrée ni la commande exécutée.

2. Message

- Apparaît à la suite de la ligne erronée :

  n° message - message condensé
- En frappant un point d'interrogation (?) à la suite de l'indicatif de mode, on obtient la forme explicite du message.

3. Correction

Elle se fait immédiatement à la suite du message.

4. Exemples

```
EDIT  210   input x
0615        MSNG = OR UNID STM
EDIT  ?
0615        MISSING EQUAL SIGN IN ASSIGNMENT STATEMENT OR UNIDENTIFIABLE STATEMENT TYPE
EDIT  210   input x
EDIT  220     if x=999 then 999
EDIT  230   input h,x9
EDIT  240 rem ****************************************
EDIT  300   if x<x9 then 150
EDIT  310   y = ((a*x_b)*x+c)*x+d
0603        SYN ERR EXPR
EDIT  ?
0603        SYNTAX ERROR IN AN EXPRESSION
EDIT  310   y = ((a*x+b)*x+c)*x+d
EDIT  330   x = x+h

      320   print using 325, x,y
EDIT  325:  x= ##.##   y= ###.####
0103        INV CMD
EDIT  ?
0103        COMMAND INVALID IN EDIT MODE
EDIT
      325 : x= ##.##   y= ###.####
EDIT  save
```

```
READY edit basic
0172        COLL NM TOO LONG
READY ?
0172        COLLECTION NAMES MUST BE 3 CHARACTERS OR LESS
READY bas basic
0103        INV CMD
READY ?
0103        COMMAND INVALID IN CNTL MODE
READY edit bas basic
EDIT
```

§ 10.4.3 Erreurs à la compilation

1. Cas d'erreur

Il s'agit principalement d'une mauvaise architecture du programme ou bien de relations erronées entre des instructions ou des instructions et des données.

2. Message

- Apparaît au moment de la tentative de compilation, lors de l'exécution de la commande "run".

n° de la ligne erronée	n° du message	message condensé

- La forme explicite du message peut s'obtenir comme précédemment.

- Le compilateur ne détecte qu'une seule erreur à la fois.

3. Correction

- Il faut modifier le programme en supprimant, ajoutant ou réécrivant des instructions.
- Veiller à choisir convenablement le nº de ligne des corrections.
- Faire un listage de la partie corrigée et ne pas oublier de sauvegarder le programme dans sa nouvelle version.

4. Exemples

```
EDIT  list
00100 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00110 REM **
00200   INPUT A,B, C
00210   INPUT X
00220     IF X=999 THEN 999
00230   INPUT H,X9
00240 REM ****************************************
00300   IF X<X9 THEN 150
00310   Y = ((A*X+B)*X+C)*X+D
00330   X = X+H
00340   GOTO 300
00999 STOP
EDIT
```

```
      run

0661        UNDEF STM NUM
EDIT  ?
0661        STATEMENT NUMBER 00000150 REFERENCED IN A STATEMENT NOT DEFINED IN PROGRAM
EDIT

      300   if x<x9 then 210

EDIT  save
```

```
00100 REM **  FICHIERS  **
00110  FOR I=1 TO 3
00120   PUT 'FIL', I**5
00130  NEXT I
00140  CLOSE 'FIL'
00145  GOSUB 160
00150  FOR J=1 TO 3
00160   GET 'FIL',U
00170   PRINT U;
00180 NEXT J
00190 END
```

```
EDIT  run
 1
00180        0624        INV FOR/NEXT BRANCH
EDIT  ?
0624         INVALID BRANCH INTO FOR/NEXT LOOP
EDIT
      145 gosub 150
```

```
EDIT  run
 1     32.00002     242.9994
```

```
00100 REM **  FICHIERS  **
00110  FOR I=1 TO 3
00120   PUT 'FIL', I**5
00130   NEXT K
00140  CLOSE 'FIL'
00150   FOR J=1 TO 3
00160   GET 'FIL',U
00170   PRINT U;
00190 END
```

```
EDIT  run

00130        0658        NEXT VAR NOT = FOR VAR
EDIT  ?
0658        NEXT STATEMENT VARIABLE MUST MATCH THE PREVIOUS FOR STATEMENT VARIABLE
EDIT
     130  next i
```

```
EDIT  run

0633        NUM FOR/NEXT NOT =
EDIT  ?
0633        THE PROGRAM MUST CONTAIN THE SAME NUMBER OF FORS AND NEXTS
EDIT   180 next j
```

```
EDIT  run
 1     32.00002     242.9994
```

§ 10.4.4 Erreurs à l'exécution

1. Cas d'erreur

   - Erreurs provoquant l'arrêt du programme et qui sont dues à une anomalie qui interdit l'exécution d'une instruction. Après impression du message le programme s'arrête.
   - Erreurs dans les résultats de l'exécution. Aucun message ne permet d'identifier la cause de l'erreur ; il faudra analyser le déroulement du programme.

2. Message d'interruption

   Il a la même forme qu'un message d'erreur de compilation.

3. Exemple d'interruption

```
00100 B=16
00110 C = B/D
00120 D = 4
00130 PRINT C
```

```
EDIT  run

00110        0547        DIV BY ZERO
EDIT  ?
0547        DIVISION BY A FLOATING POINT NUMBER WITH ZERO FRACTION WAS ATTEMPTED
EDIT  end
READY
```

4. Exemple de résultats erronés

```
00010 REM **  ERREUR A L'EXECUTION  **
00020 REM **          V.A1           **
00030 REM
00040      INPUT A,B,C
00050      D = B*B - 4*A*C
00060      X1 = (-B+SQR(D))/2*A
00070      X2 = (-B-SQR(D))/2*A
00080   PRINT USING 90,X1,X2
00090   : X1 = ###.###     X2 = ###.###
00100 GOTO 40
00110 END
RDY
      run
  ?   1,-2,1
 X1 =    1.000     X2 =    1.000   <- CORRECT
  ?   4,-8,4
 X1 =   16.000     X2 =   16.000   <- ERRONE
  ?
RDY
```

5. Correction

Elle est souvent plus délicate car il faut d'abord localiser l'origine de l'erreur. Pour cela, on peut :

- Suivre, pas à pas, le déroulement du programme, sur un listing à l'aide des données fournies à l'exécution.
- Imprimer des résultats intermédiaires pour les comparer à ceux du jeu d'essai afin de déterminer à partir de quelle instruction ils deviennent erronés.
- Exécuter le programme dans le sous-mode TEST.

§ 10.4.5 Jeux d'essais

1. Définition

Ensemble de données de base pour lesquelles les résultats du traitement programmé (résultats intermédiaires et résultat final) ont été déterminés indépendamment du programme.

2. Caractéristiques

Ces jeux, en principe, doivent permettre de contrôler

le bon déroulement du programme. En particulier, ils doivent permettre de tester toutes les possibilités du programme.
Ils ne doivent pas être constitués uniquement de données particulières ou simples qui risqueraient alors de masquer des erreurs.
Les jeux d'essais doivent être constitués en même temps que l'on écrit le programme.

Section 10.5
SOUS-MODE TEST

§ 10.5.1 Fonctionnement

1. Buts

L'exécution d'un programme en sous-mode TEST facilite la mise au point en permettant :

- de suivre, pas à pas, le déroulement du programme au cours de son exécution en donnant
  - . la trace des débranchements,
  - . les valeurs des variables du programme ;
- d'interrompre provisoirement le déroulement du programme en un point déterminé ;
- de modifier, en cours d'exécution, les valeurs des variables numériques du programme.

2. Initialisation du sous-mode "test"

- Elle se fait au niveau de la commande "run"

```
EDIT ↵ ↵ run ↵ test
TEST ↵ ↵
```

- A la suite de la réponse TEST, l'utilisateur précise les commandes nécessaires au contrôle de l'exécution de son programme.

3. Commande GO

- Une fois toutes les commandes de contrôle précisées, l'exécution effective du programme peut commencer. C'est le rôle de la commande "go". GO est indispensable à l'exécution effective du programme.
- De même, à chaque point d'interruption programmée, d'autres commandes peuvent être spécifiées et l'exécution du programme se poursuivra à la suite de la commande GO.

4. Clôture du mode TEST

Fin normale : fin du programme.

Arrêt provisoire : en frappant la touche ATTN. I.T.F. répond TEST. On peut alors introduire de nouvelles commandes.

Arrêt définitif : si à la suite d'une réponse TEST, on frappe "end", l'exécution du programme s'interrompt définitivement et l'on se retrouve en mode EDIT.

5. Exemple

```
00010 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00100 REM **
00105   INPUT A,B,C,D
00110   INPUT X
00115 IFX=999THEN160
00120   INPUT H,X9
00125 REM ****************************************
00130 IFX>X9THEN110
00135   Y = ((A*X+B)*X+C)*X+D
00140 PRINTUSING145,X,Y
00145 : X= ##.##   Y= ###.####
00150   X = X+H
00155 GOTO130
00160 STOP
EDIT
```

```
      run test
TEST  at 115,150
TEST  go
   ?  1,3,3,1
   ?  0
A00115 list(a,b,c,d)
A=+1.000000E+00
B=+3.000000E+00
C=+3.000000E+00
D=+1.000000E+00
A00115off 115
A00115go
   ?  0.5,1
 X=  0.00  Y=   1.0000
```

```
A00150list x
***102 INVALID TMODE COMMAND
A00150list (x)
X=+0.000000E-01
A00150off
A00150go
 X=  0.50  Y=   3.3750
 X=  1.00  Y=   8.0000
  ?  -1
  ?  0.5,0
 X= -1.00  Y=   0.0000
 X= -0.50  Y=   0.1250
 X=  0.00  Y=   1.0000
  ?  ATTN
TEST  end
EDIT
```

§ 10.5.2 Interruption programmée de l'exécution

1. Principe

Avant ou en cours d'exécution, on peut définir des points d'arrêts dans le programme et en ces points il sera alors possible d'accéder aux valeurs des variables, soit pour en prendre connaissance, soit pour les modifier ou bien d'introduire d'autres commandes de TEST.

2. Commande "at"

- La commande "at" permet de préciser les n°s des lignes où l'on désire interrompre l'exécution.

  TEST ␣ ␣ at ␣ n1 [,n2, ..., n1Ø]

- L'arrêt se fait juste avant réalisation de l'instruction et I.T.F. écrit :

  A n° ligne

- On peut alors introduire des commandes de test puis relancer le programme par GO, ou bien interrompre le mode TEST.

3. Annulation d'interruption

- En faisant apparaître le n° de ligne dans une commande "off", on annule l'interruption qui avait été programmé pour cette instruction.

  TEST ␣ ␣ off ␣ [n1,n2, ..., n1Ø]

- Si l'on frappe simplement "off" sans préciser de n° de ligne, tous les points d'interruption sont annulés.

4. Exemple

```
EDIT  run test

TEST  at 140, 190,210
TEST  go
```

```
A00140list(i)
I=+1.000000E+00
A00140  i = 5
A00140go
```

```
A00190 list(i)
I=+1.000000E+00
A00190 i=4
A00190list i
***102 INVALID TMODE COMMAND
A00190go
A00190list(i)
I=+5.000000E+00
A00190list(i)
I=+5.000000E+00
A00190go
```

```
A00190off 140,190
A00190go
A00210list(a$)
A$='                    '
A00210at 220
A00210go
A00220list(a$)
A$='F I N                '
A00220 a$='enfin fini'
***102 INVALID TMODE COMMAND
A00220go
 0      125    216
F I N

EDIT
```

§ 10.5.3 Accès aux valeurs du programme

1. Listage de valeurs

Au point d'interruption, on peut demander l'impression des valeurs de variables par la commande "list".

list : toutes les variables du programme.

list (x,...,z) : les valeurs des variables x,...,z qui doivent être des noms de variables élémentaires ou de tableaux.

2. Exemple de listage

```
        run test
TEST    at 150
TEST    go
  ?     1,3,3,1
  ?     0
  ?     0.5,1
 X=   0.00   Y=     1.0000
→A00150list
 A=+1.000000E+00
 B=+3.000000E+00
 C=+3.000000E+00
 D=+1.000000E+00
 X=+0.000000E-01
 H=+5.000000E-01
 X9=+1.000000E+00
 Y=+1.000000E+00
 A00150go
```

```
   X=   0.50   Y=     3.3750
→A00150list(x,y)
 X=+5.000000E-01
 Y=+3.375000E+00
 A00150off
 A00150go
   X=   1.00   Y=     8.0000
    ?   999
 EDIT
```

3. Modification de valeurs

- Au point d'interruption on écrit une "pseudo-instruction" d'affectation de la forme :

  var-élémentaire arithm. non-indicée = constante numérique.

- Dans le programme, la variable prend cette nouvelle valeur.

4. Exemple de modification de valeur

```
00100 REM ** ACCES AUX VALEURS  **
00120 DIM A(6)
00130   FOR I=1 TO 6
00140   A(I) = I**3
00150   NEXT I
00160 REM ************************
00180 FOR I=1 TO 6
00190 PRINT A(I);
00200 NEXT I
00210 A$='F I N'
00220 PRINT
00230 PRINT A$
00240 END
EDIT
```

```
EDIT  run test

TEST  at 140, 190,210
TEST  go

 A00140list(i)
 I=+1.000000E+00
→A00140  i = 5
 A00140go
 A00140go
→A00190 list(i)
 I=+1.000000E+00
→A00190 i=4


 A00190go
 A00190list(i)
→I=+5.000000E+00
```

§ 10.5.4 Contrôle du flot du programme

1. Principe

Au moyen de la commande "trace", il est possible de connaître automatiquement :

- la nouvelle valeur d'une variable,
- le numéro d'une instruction à laquelle le programme se débranche,
- le nom d'une fonction incorporée ou d'un fichier dès qu'ils sont utilisés.

2. Commande "trace"

TEST ♯ ♯ trace — Tous les noms de variables, numéros de débranchement, fichiers et fonctions incorporées.

TEST ♯ ♯ trace (*) — Uniquement les numéros de débranchement.

TEST ♯ ♯ trace (x,...z) — Uniquement les variables non indicées, tableaux, fichiers ou fonctions incorporées x,...,z.

TEST ♯ ♯ trace (*,x,...z) — En plus, les numéros de débranchement.

3. Résultats

- Pour un nom de variable, après changement de valeur

  n° inst^on nom-var = valeur-variable

- Pour un débranchement, juste avant sa réalisation

  n° de la ligne

- Pour un fichier ou une fonction incorporée

  n° ligne FILE / B.I.F. BEING REFERENCED

4. Commande "notrace"

Elle a les mêmes formes que "trace", avec l'effet contraire.

5. Exemple n° 1

```
00010 REM ** CALCUL DE LA VALEUR D'UN POLYNOME  **
00100 REM **
00105   INPUT A,B,C,D
00110   INPUT X
00115 IFX=999THEN160
00120   INPUT H,X9
00125 REM ****************************************
00130 IFX>X9THEN110
00135   Y = ((A*X+B)*X+C)*X+D
00140 PRINTUSING145,X,Y
00145 : X= ##.##  Y= ###.####
00150   X = X+H
00155 GOTO130
00160 STOP
EDIT
```

```
      run test
TEST  trace
TEST  go
  ?   1,3,3,1
00105       A=+1.000000E+00
00105       B=+3.000000E+00
00105       C=+3.000000E+00
00105       D=+1.000000E+00
            00110
  ?   0
00110       X=+0.000000E-01
  ?   0.5,1
00120       H=+5.000000E-01
00120       X9=+1.000000E+00
            00130
00135       Y=+1.000000E+00
 X=  0.00   Y=   1.0000
            00145
00150       X=+5.000000E-01
            00130
00135       Y=+3.375000E+00
 X=  0.50   Y=   3.3750
            00145
00150       X=+1.000000E+00
            00130
00135       Y=+8.000000E+00
 X=  1.00   Y=   8.0000
            00145
00150       X=+1.500000E+00
            00130
            00110
  ?   999
00110       X=+9.990000E+02
            00160
EDIT
```

```
      run test
TEST  trace(*)
TEST  go
  ?   1,3,3,1
            00110 ←
  ?   0
  ?   0.5,1
            00130 ←
 X=  0.00   Y=   1.0000
            00145 ←
            00130 ←
 X=  0.50   Y=   3.3750
            00145 ←
            00130 ←
 X=  1.00   Y=   8.0000
            00145 ←
            00130 ←
            00110 ←
  ?   999
            00160 ←
EDIT
```

```
      run test
TEST  trace(x,y)
TEST  go
  ?   1,3,3,1
  ?   0
→ 00110       X=+0.000000E-01
  ?   0.5, 1
→ 00135       Y=+1.000000E+00
  X=  0.00    Y=   1.0000
→ 00150       X=+5.000000E-01
→ 00135       Y=+3.375000E+00
  X=  0.50    Y=   3.3750
→ 00150       X=+1.000000E+00
→ 00135       Y=+8.000000E+00
  X=  1.00    Y=   8.0000
→ 00150       X=+1.500000E+00
  ?   999
→ 00110       X=+9.990000E+02
  EDIT
```

6. Exemple n° 2

```
00100 REM ** FONCTIONS INCORPOREES **
00110 DIM A(3)
00120  FOR I=1 TO 3
00130    A(I) = SIN(RAD(10*I))
00140    PRINT ABS(A(I));
00150  NEXT I
00160 END
EDIT  run test
TEST  trace
TEST  go
 00120       I=+1.000000E+00
 00130       RAD B.I.F. BEING REFERENCED
 00130       SIN B.I.F. BEING REFERENCED
 00130       A(1)=+1.736481E-01
 00140       ABS B.I.F. BEING REFERENCED
 00150       I=+2.000000E+00
 00130       RAD B.I.F. BEING REFERENCED
 00130       SIN B.I.F. BEING REFERENCED ←  ATTN
```

```
TEST  end
 .1736481
EDIT
      run test
TEST  trace(sin)

TEST  go
 00130      SIN B.I.F. BEING REFERENCED
 00130      SIN B.I.F. BEING REFERENCED
 00130      SIN B.I.F. BEING REFERENCED
  .1736481     .3420200     .4999999
```

7. Exemple n° 7

```
U
 0100 REM **  FICHIERS  **
00110  FOR I=1 TO 3
00120   PUT 'FIL', I**5
00130  NEXT I
00140  CLOSE 'FIL'
00150   FOR J=1 TO 3
00160   GET 'FIL',U
00170   PRINT U;
00180  NEXT J
00190 END

EDIT  run

 1      32.00002     242.9994

EDIT  run test

TEST  trace
TEST  go
 00110      I=+1.000000E+00
 00120      FIL FILE BEING REFERENCED
 00130      I=+2.000000E+00
 00120      FIL FILE BEING REFERENCED
 00130      I=+3.000000E+00
 00120      FIL FILE BEING REFERENCED
 00140      FIL FILE BEING REFERENCED
 00150      J=+1.000000E+00
 00160      FIL FILE BEING REFERENCED

 00160      U=+1.000000E+00 <-- ATTN
TEST  notrace
TEST  go
  1      32.00002     242.9994
```

```
EDIT  run test
TEST  trace(fil)
TEST  go
 00120      FIL FILE BEING REFERENCED
 00120      FIL FILE BEING REFERENCED
 00120      FIL FILE BEING REFERENCED
 00140      FIL FILE BEING REFERENCED
 00160      FIL FILE BEING REFERENCED  <-- ATTN
TEST  end
EDIT
```