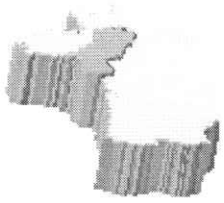
IREM

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



NANTA IREMICA N° 13

Le PL/I optimiseur sous DOS-370

Premier module
Le PL/I de base monobloc

M. Belhache

1978

UNIVERSITE DE NANTES

I. R. E. M.

LE PL/I OPTIMISEUR

SOUS DOS - 370

PREMIER MODULE

LE PL/I DE BASE MONOBLOC

par

M. BELHACHE

Maitre Assistant à l'U.E.R. de Mathématiques
Animateur à l'I.R.E.M. de Nantes

N A N T A I R E M I C A

Volume n° 13

LE PL/I OPTIMISEUR

sous DOS-370

PREMIER MODULE

LE PL/I DE BASE MONOBLOC

M. BELHACHE

I N T R O D U C T I O N

Ce cours polycopié est principalement destiné à des étudiants ayant déjà acquis l'expérience de la programmation et désireux d'acquérir la maîtrise d'un langage de programmation évolué et quasi-universel.

Le langage PL/I dans sa version optimiseur sous D.O.S.-370, dont il est question ici, répond à cette quasi-universalité d'emploi tout en laissant la possibilité de le subdiviser en sous-ensembles cohérents satisfaisant des besoins spécifiques à des domaines d'emploi plus limités : calculs scientifiques et techniques, traitement de fichiers, de données non-numériques... Cette modularité du langage permet donc de l'employer utilement sans s'astreindre à en connaître la totalité. Cependant, il faut noter que sa richesse et sa souplesse le destinent à un public averti et relativement restreint. Pour l'utiliser totalement et efficacement, il importe d'avoir déjà acquis une expérience suffisante de la programmation.

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'étude du langage et à en alléger l'enseignement.

Parfois, les explications seront abondantes et détaillées. Ce sera le cas des chapitres relatifs aux aspects originaux du PL/I comme par exemple la structuration par blocs, la gestion programmée des interruptions... Mais le plus souvent, on se contente d'indiquer les notions de base et les règles principales qui s'y rattachent en les illustrant d'exemples et de contre-exemples extraits de programme effectivement réalisés. Les autres détails sont exposés dans le cadre de l'enseignement traditionnel à moins qu'on ne laisse à l'étudiant le soin de les découvrir lui-même à travers ses propres programmes.

L'étude du langage a été divisée en 17 chapitres, eux-mêmes regroupés en 4 sous-ensembles correspondant à autant de modules.

- 1er module : PL/I de base monobloc
  chapitres 1, 2 et 3.
- 2ème module : PL/I scientifique multibloc,
  chapitres 4 à 8.

- 3ème module : PL/I - Fichiers D.O.S.
  chapitres 9 à 12.
- 4ème module : Outils avancés de PL/I
  chapitres 13 à 17.

Ce découpage n'est pas impératif. Certaines parties peuvent être étudiées succinctement ou même passées sous silence au cours d'une première approche du langage. De plus, il entraîne certaines "redites". Des notions comme la gestion programmée des interruptions sont étudiées de façon relativement sommaire dans le 2ème et le 3ème modules, alors qu'elles font l'objet d'un chapitre particulier du 4ème module où elles sont étudiées en totalité et en détail. Malgré cela, l'expérience montre que ce découpage correspond aux besoins des utilisateurs et aux nécessités pédagogiques de l'enseignement.

PREMIER MODULE
PL/I DE BASE MONOBLOC

Dans ce premier module, on se borne à l'étude des éléments de données de type numérique et chaînes et des principales instructions et fonctions permettant leur traitement. Les entrées/sorties sont restreintes au mode STREAM entre un lecteur de cartes et une imprimante. Le programme est limité à un seul bloc procédure.

L'étude de ce premier module étant un préliminaire obligatoire à celle des autres modules, elle constitue par le fait même une introduction au PL/I Optimiseur. Aussi avons-nous pensé préférable d'introduire immédiatement et de façon concrète les notions élémentaires du langage de sorte que l'étudiant puisse disposer le plus rapidement possible d'outils suffisants pour écrire ses propres programmes. La programmation est, en effet, une technique qui s'acquiert par l'exemple et la pratique. Plutôt que de commencer par l'étude abstraite des éléments de données, des instructions..., nous pensons préférable de débuter un tel enseignement par l'explication approfondie d'exemples élémentaires. Muni des premiers outils qu'il aura pu découvrir dans ces exemples et, s'en inspirant au besoin, l'étudiant pourra écrire et mettre au point tout de suite des petits programmes. C'est alors que l'étude du langage proprement dit sera fructueuse, car elle apparaîtra comme une synthèse et une extension des notions initialement acquise "sur le tas".

C'est la raison pour laquelle nous avons joint en annexe à ce premier module un ensemble de 13 programmes complets dont les dix premiers peuvent jouer le rôle d'introduction.

Au premier chapitre, on trouvera la définition et les caractéristiques des éléments de données de type numérique (réel et complexe), chaîne de caractères et chaîne de bits. Leur organisation en agrégats se limite au cas des tableaux, l'étude des structures étant renvoyée au chapitre 9. La déclaration des données comporte l'attribut INITIAL mais aussi les attributs de redéfinition : DEFINED, POSITION et ISUB. Enfin, pour clore ce chapitre, après le rappel des attributs par défaut standards du Système, un paragraphe sur l'instruction DEFAULT montre qu'il existe d'autres possibilités.

Le chapitre 2 est centré sur l'instruction d'affectation. On y étudie d'abord tous les opérateurs que l'on peut associer aux éléments de données définis précédemment en introduisant également la notion de conversion et et en l'illustrant par quelques possibilités liées à ces opérateurs. Il est alors possible de construire des expressions entre scalaires et/ou tableaux, de réaliser des instructions d'affectation sans oublier les conversions qui peuvent les accompagner.

Au dernier chapitre de ce module, on commence par étudier le GET et le PUT dans les formes LIST et DATA, renvoyant l'étude de la forme EDIT à la dernière section du chapitre. Les instructions de contrôle de séquence, GOTO, IF et DO font l'objet de la 2ème section du chapitre. Toutes les formes du DO y sont étudiées en détail. Enfin, on trouvera les fonctions et pseudo-variables incorporées à PLIOPT. On s'est limité aux principales permettant le traitement des éléments de données définis au chapitre 1. L'étude des autres fonctions est renvoyée aux chapitres spécialisés.

Plus de 40 programmes ont été inclus tout au long de ces 3 chapitres afin d'illustrer les notions introduites.

M. BELHACHE

Nantes, mai 1976.

TABLE DES MATIERES

ANNEXE

SOMMAIRE DES MODULES 2, 3 ET 4

INTRODUCTION

SECTION 1

LES CARACTERISTIQUES GENERALES

DU PL/I

Conçu en 1964 par un groupe d'utilisateurs auquel se joint IBM, PL/I (Programming Language n° I) contient d'une manière générale ce qu'il y avait de meilleur dans les langages existants et offre de nouvelles possibilités. Dès le départ, il est prévu pour exploiter les avantages de la troisième génération d'ordinateurs.

PL/I n'est donc pas un langage spécialisé mais, au contraire, un langage d'utilisation universelle et dont l'expression est suffisamment naturelle pour être facilement comprise. C'est également un langage modulaire qu'il n'est nul besoin de connaître à fond dans sa totalité pour utiliser efficacement. A la plupart des questions, il offre plusieurs réponses, plus ou moins sophistiquées et il est toujours possible de choisir l'outil le mieux adapté au traitement envisagé. Mais c'est aussi un langage extensible ; grâce à son macrolangage, il se prête simplement à l'incorporation de nouvelles fonctions.

Tous les types de données, numériques et non-numériques, peuvent être traitées par PL/I de façon au moins aussi simple qu'en FORTRAN ou en

...

COBOL... Et plus l'on avance dans l'étude du PL/I, plus l'on dispose d'outils permettant un traitement en mémoire de plus en plus fin, jusqu'à retrouver la souplesse de l'Assembleur. Enfin, n'oublions pas que le PL/I contient ses propres outils de mise au point. Quiconque a mis au point un programme COBOL un peu volumineux et sait combien cette tâche est délicate et fastidieuse appréciera cet avantage à sa juste valeur.

On doit donc reconnaître que malgré certains inconvénients liés à la richesse du langage et à la relative lenteur des compilateurs, le bilan global de la programmation en PL/I, depuis l'écriture du programme jusqu'à sa maintenance, est largement positif.

SECTION 2

ELEMENTS DE BASE DU PL/I

§ . 2 . 1 ALPHABETS DU PL/I

1. Alphabet standard : 60 caractères

26 lettres : A, ..., Z
plus : $, @, #
} alphabétiques

10 chiffres : 0, ..., 9 } numériques

} alphanumériques

21 caractères spéciaux :

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

2. Alphabet réduit : 48 caractères

26 lettres, $, 10 chiffres

11 caractères spéciaux :

. , ( ) + - * / = ¢ '

⟹ modifications : devient ..
; devient ,.

12 mots-clés du langage sont réservés pour remplacer certains opérateurs.

...

§ . 2 . 2 VOCABULAIRE PL/I

1. Généralités

- Tous les éléments du langage sont formés à partir des caractères de l'alphabet.

- Deux classes :

  . vocabulaire propre à PL/I { opérateurs ; mots-clés du langage

  . vocabulaire de l'utilisateur { identificateurs : constantes et variables ; commentaires

2. Opérateurs

- Formés à partir de caractères spéciaux,

- Pour exprimer une opération simple entre deux opérandes (en général) :

  . arithmétiques : + - * / **

  . chaînage : ||

  . booléens : ¬ & |

  . comparaison : > ¬> >= ¬= = <= ¬< <

3. Mots-clés

- extension de la notion d'opérateur : ils ont une orthographe stricte, ils doivent être employés conformément aux règles de la syntaxe et servent à exprimer une action, une fonction à réaliser ou un état des données.

- environ 300 :

  GET, PUT, SUBSTR, DECLARE, ...

- ces mots ne sont pas réservés ; ils sont reconnus et interprétés selon leur place dans le contexte de la phrase.

4. Identificateurs

- mots construits par le programmeur pour identifier des éléments d'information dans le programme : constantes ou variables.

...

- doivent avoir une unique signification à l'intérieur d'un bloc.

- règles pour un identificateur de variable :

  . initiale : caractère alphabétique

  . puis, jusqu'à 30 caractères alphanumériques

  . caractères spéciaux interdits sauf _

- en général, l'identificateur d'une variable est associé de façon biunivoque à l'adresse d'un emplacement de mémoire.

- exemples : 123.45 'ALPHA'
  NOM13 TAUX_HORAIRE

3. Remarque

Les identificateurs de fichiers et points d'entrée doivent comporter sept caractères au plus, sans _ .

§ . 2 . 3 SIGNES DE PONCTUATIONS

,	sépare les éléments d'une liste.
.	sépare la partie entière de la partie fractionnaire d'une constante numérique. relie des qualificateurs à une sous-structure.
;	marque la fin de l'instruction.
:	relie un préfixe à une instruction.
( )	mise en facteur et définit une sous-expression. délimite la portée d'une spécification répétitive dans une liste-donnée.
'	délimite à gauche et à droite une constante chaîne.
%	caractérise une instruction de précompilation.

```
123.45      'PLIOPT '
ALPHA: GET LIST(( B(I) DO I=1 TO N));
%DEACTIVATE  Z ;
```

...

§ . 2 . 4 UTILISATION DES BLANCS

- Interdits à l'intérieur d'un mot-clé, d'un opérateur composé, d'un identificateur de variable, d'une constante numérique.

- Obligatoires pour séparer les mots de l'instruction (en l'absence d'autre séparateur)

  GOTOB ≠ GOTO B

  GET EDIT(A) équivalent à GET EDIT (A)

- Autorisés partout ailleurs en nombre quelconque.

SECTION 3

SYNTAXE PL/I

§ . 3 . 1 L'INSTRUCTION PL/I

1. Rôle et forme

- analogue à la phrase, elle exprime soit une action à faire réaliser par l'ordinateur, soit un état de donnée du programme.

- en général construite autour d'opérateurs et/ou de mots-clés associés à des opérandes.

2. Classes d'instruction

- instruction d'affectation

```
cible = source ;
```

- instructions à mot-clé

```
GOTO LAB ;
GET LIST (X) ;
```

- instructions composées

```
IF  A = B  THEN... ;
           ELSE... ;
```

- instruction nulle

```
sans action
```

...

3. Ecriture pour le compilateur PLIOPT

- format quasiment libre entre les colonnes 2 à 72, sur une ou plusieurs cartes.

- une carte peut contenir plusieurs instructions ; une instruction peut s'étendre sur plusieurs cartes.

- seul impératif

  chaque instruction doit se terminer par un point-virgule (;)

§ . 3 . 2 PREFIXES D'INSTRUCTION

1. Etiquette ou label

- identifie l'instruction.

- doit être un identificateur valide.

- il peut y en avoir plusieurs étiquettes qui sont alors équivalentes.

- reliée(s) par :

```
ETIQ: LAB(I) :  GO TO X ;
```

2. Préfixe-condition(s)

- liste entre parenthèses, de noms de conditions, à propos desquelles les interruptions associées se réaliseront (ou non) si les conditions précises se présentent.

- relié à l'instruction par :
  précède les étiquettes.

```
(SIZE , NOFOFL ):
        LABEL   :  C =  A + B ;
```

§ . 3 . 3 STRUCTURE DE BASE DU PROGRAMME PL/I

1. Procédure

- les instructions doivent être rassemblées en une ou plusieurs procédures.

...

- chaque procédure

  commence par une instruction PROCEDURE, préfixée d'une étiquette appelée point d'entrée principal et qui donne son nom à la procédure

  se termine par une instruction END éventuellement suffixée du nom de la procédure .

- à l'intérieur, sauf débranchement programmé, les instructions se déroulent séquentiellement .

- exemple

```
POINT : PROCEDURE ;



        END POINT ;
```

2. Programme

- tout programme doit comporter au moins une procédure .

- la première (ou unique) à être exécutée à la suite du / / EXEC doit recevoir l'option MAIN ;

```
POINT : PROC OPTIONS (MAIN) ;
```

3. Blocs

- un programme peut être découpé en blocs .

- deux types : bloc Begin
  bloc Procédure

- ils réalisent un découpage du programme permettant, entre autre, de limiter la portée des identificateurs et de contrôler l'allocation mémoire .

4. Groupes

- à l'intérieur d'une procédure, c'est une séquence d'instructions considérée logiquement comme un tout .

- commence par DO ; se termine par END .

...

§ . 3 . 4 COMMENTAIRE

```
/*  CECI  EST  UN
          COMMENTAIRE    */
```

- suite de caractères délimitée
  - . à gauche par /*
  - . à droite par */
- pas de blanc entre / et *
- autorisé partout où l'on peut placer un blanc.

SECTION 4

TROIS SOUS-ENSEMBLES DANS PL/I

Suivant le domaine d'application envisagé, on peut découper trois sous-ensembles dans PL/I.

1. PL/I de BASE

Il comporte la plupart des instructions et fonctions pour le traitement des données numériques et des données de type chaîne en limitant les entrées/sorties, au mode Stream entre un lecteur de cartes et une imprimante et en restreignant la gestion mémoire et la gestion des interruptions aux cas élémentaires ou indispensables.

Ce premier sous-ensemble recouvre largement les possibilités du FORTRAN IV de niveau F. Il s'adresse particulièrement aux utilisateurs scientifiques et dans ce cas l'étude du traitement des chaînes peut être très réduite. Il se subdivise en deux modules.

1er module : PL/I de BASE monobloc

- chapitre 1 : Les éléments de données
- chapitre 2 : Expressions ; instruction d'affectation
- chapitre 3 : Instructions et fonctions de base

2ème module : PL/I de BASE multibloc

- chapitre 4 : Structuration par blocs et reconnaissance des noms

...

. chapitre 5 : Gestion de quelques interruptions ; outils de mise au point

. chapitre 6 : Gestion des classes de mémoire AUTO et CTL

. chapitre 7 : Les sous-programmes

. chapitre 8 : Les conversions supportées par PLIOPT

2. PL/I FICHIERS

Ce deuxième sous-ensemble s'adresse aux gestionnaires, utilisateurs de COBOL. Il traite de la manipulation des structures, des chaînes de caractères et des fichiers. Les entrées/sorties s'étendent à tous les supports et tous les modes de transmissions.

. chapitre 9 : Compléments au PL/I de BASE

. chapitre 10 : Les transmissions en mode Stream

. chapitre 11 : Déclaration des fichiers du PL/I sous DOS

. chapitre 12 : Les transmissions en mode Record.

3. PL/I AVANCE

Il intéresse les amateurs d'Assembleur séduits par sa souplesse et sa finesse de traitement. Ce dernier sous-ensemble regroupe donc tous les outils du PL/I qui permettront de retrouver ces avantages.

. chapitre 13 : La gestion de la mémoire

. chapitre 14 : La gestion des interruptions

. chapitre 15 : Le langage de précompilation

. chapitre 16 : Autres outils du PL/I

. chapitre 17 : Communications interlangages.

CHAPITRE 1

ELEMENTS DE DONNEES

SECTION 1.1

TYPES DE DONNEES

§ . 1 . 1 . 1 CLASSES DE DONNEES

1. Constante, variable

- Constante : élément d'information de valeur déterminée et invariante

  identifiée par son écriture qui exprime à la fois sa valeur et ses caractéristiques ou attributs

- Variable : nom symbolique (identificateur)

  en général associé de façon biunivoque à l'adresse d'un emplacement mémoire

  dont la valeur (contenu de l'emplacement mémoire) varie au cours du programme

  mais dont les attributs sont en général inconnus et devront être déclarés.

...

2. Donnée-problème, donnée contrôle de programme

- donnée-problème : élément d'information, d'origine souvent externe, sur lequel s'applique l'algorithme programmé

  2 types : numérique
  chaîne de caractères ou de bits

- donnée-contrôle de programme : élément d'information nécessaire à l'organisation et au contrôle du déroulement du programme : étiquette, point d'entrée, fichier, localisateur...

§ . 1 . 1 . 2 CONSTANTES NUMERIQUES

1. Formes autorisées

Le PL/I permet le traitement de valeurs numériques

- complexes ou réelles
- exprimées en base décimale ou binaire
  en virgule fixe ou flottante
  avec diverses précisions

2. Constantes numériques réelles

- décimale virgule fixe : jusqu'à 15 chiffres décimaux
  [, signe , point fractionnaire] :
  -123.45 + 12

- décimale virgule flottante : mantisse-exposant
  mantisse : constante virgule fixe jusqu'à 16 chiffres
  exposant : E±ee $-78 \leq ee \leq +75$
  -123.45E+15 12E-5

- binaire virgule fixe : jusqu'à 31 chiffres binaires (Ø ou 1)
  [, signe , point fractionnaire] , B
  -1ØØØ11.1B 1Ø1B

- binaire virgule flottante : mantisse-exposant
  mantisse : constante binaire virgule fixe jusqu'à 53 chiffres binaires (pas de B)

...

exposant : E±nnnB nnn décimal, -26Ø à + 252

1Ø.1E+3B vaut 1,5 x 2^3 = 12

3. Constantes numériques complexes

Sous la forme X + YI

X et Y étant des constantes réelles, I étant le caractère I :

1.E+1+5.E-1I

4. Remarque

Une constante numérique ne doit contenir aucun blanc intercalaire.

§.1 . 1 . 3 VARIABLES NUMERIQUES

1. Formes autorisées

- les mêmes formes sont possibles
- le nom doit être un identificateur valide
- en général les attributs de la forme devront être associés au nom dans la déclaration. La valeur numérique devra être conforme à ces attributs ou bien convertible dans ces attributs.

2. Attributs d'une variable numérique réelle

- Mode réel : REAL ; option par défaut qui peut donc être omise
- Base de numération : BINARY ou BIN
  DECIMAL ou DEC
- Virgule fixe ou flottante : FIXED
  FLOAT
- Précision en virgule fixe FIXED [p [,q])]

  p et q : constantes entières décimales
  p : nbr total des chiffres ≤ 15 (ou 31 en binaire)
  q : nbr des chiffres fractionnaires
  précise la position du point fractionnaire par rapport à la droite du nombre
  q = Ø peut être omis
  q < Ø ⇒ x 10^{-q} (ou x 2^{-q} en binaire)

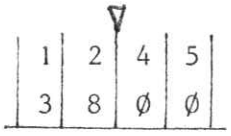
  cadrage de la valeur sur la position figurée du point
  troncature et/ou padding de zéros à droite et/ou à gauche.

...

FIXED (4,2)

12.456

338



Si l'attribut de précision est omis, par défaut le système prend

(5,Ø) ou (5) en décimal

(15,Ø) ou (15) en binaire

- Précision en virgule flottante FLOAT (p)

  p : constante entière décimale non signée $\leq$ 16 (ou 53 en binaire) nombre minimal des chiffres significatifs de la mantisse

  Si l'attribut est omis, le système prend (6) en décimal et (21) en binaire

3. Stockage en mémoire ; arithmétiques utilisées

DEC FIXED (p,q) : décimal condensé, 1 chiffre par demi-octet. Arithmétique décimale

BIN FIXED (p,q) : un demi-mot si $p \leq 15$
un mot si $16 \leq p \leq 31$.
Arithmétique binaire

DEC FLOAT (p) : un mot si $p \leq 6$
un double mot si $7 \leq p \leq 16$.
Arithmétique Virgule Flottante

BIN FLOAT (p) : un mot si $p \leq 21$
un double mot si $22 \leq p \leq 53$.
Arithmétique Virgule Flottante.

4. Variables numériques complexes

- l'attribut de mode est obligatoire : COMPLEX ou CPLX
- les autres attributs sont ceux d'une variable réelle et s'appliquent aux deux parties

CPLX DEC FIXED (7,3)

§.1.1.4 CONSTANTES-CHAINES

1. Définition

Une chaîne est formée d'une suite continue de caractères, délimitée à gauche et à droite par un apostrophe ('). Sa valeur est celle exprimée par l'ensemble des caractères qui la composent.

...

2. Propriétés fondamentales

- la longueur est égale au nombre des caractères
- une chaîne est en général manipulée globalement. Elle est cadrée à gauche (1er caractère) et éventuellement ajustée à droite
- elle peut s'exprimer par la répétition d'une chaîne de base qui est alors précédée de ce facteur de répétition placé entre parenthèses

  (2)'DO' => 'DODO'

  Le facteur de répétition doit être une constante.

3. Chaîne de caractères

- formée de caractères du 370

  l'apostrophe, utilisée pour sa valeur de caractère à l'intérieur de la chaîne, par convention, doit être doublé

  c'est écrit devient la constante 'C''EST ECRIT'

- un blanc interne est compté pour un caractère
- une chaîne de caractère stockée à raison de 1 caractère par octet.

4. Chaîne de bits

- formée uniquement de symboles (bits) Ø ou 1 ; l'apostrophe délimiteur de droite doit être suivie de B

  '11Ø'B

- le blanc est interdit
- stockée à raison de 8 bits par octet.

5. Chaîne nulle

Sans caractère ni bit, elle est de longueur nulle et formée de deux apostrophes consécutifs

'' ou ''B

§ . 1 . 1 . 5 VARIABLES CHAINES

Leur déclaration est obligatoire

...

1. Chaîne de caractères

- attribut CHARACTER ou CHAR suivi de l'attribut de longueur entre parenthèses (nombre (maximum) d'octets réservés en mémoire)
- dans une variable CHAR, une constante chaîne de caractères est cadr à gauche et ajustée à droite par padding de blanc ou troncature des caractères de droite

2. Chaîne de bits

- attribut BIT suivi de l'attribut de longueur (nombre (maximum) de bits réservés en mémoire)
- dans une variable BIT, une constante chaîne de bits est cadrée à gauche et ajustée à droite par padding de Ø ou troncature des bits de droite.

3. Attribut de longueur

Sous la forme CHAR (long.) ou BIT (long)

long.: constante décimale entrées de Ø à 32767
dans certains cas, une expression ou * .

(long) omis ⟹ CHAR (1) ou BIT (1)

4. Longueur variable

Utiliser l'attribut VARYING ou VAR

Signifier que la longueur de la variable peut

- . varier de Ø à la longueur déclarée qui est alors un maximum
- . et qu'elle est actualisée à la longueur de la constante que contient la variable

La troncature à droite n'intervient que si l'on dépasse la longueur maximum

Le padding n'a plus de raison d'être.

A CHAR (4) , B CHAR (4) VAR

- 'UN' dans A ⟹ | U | N | b | b |
- 'UN' dans B ⟹ | U | N |
- 'DEUX' dans A et B | D | E | U | X |
- 'TROIS' dans A et B | T | R | O | I |

En mémoire, la chaîne est précédée d'un préfixe qui contient la longueur actuelle de sa valeur.

...

§ . I.1.6 DONNEES CONTROLE DE PROGRAMME

1. Etiquette ou label

- constante : identificateur valide préfixant une instruction
  Il peut y avoir plusieurs étiquettes
- variable : doit être déclarée avec l'attribut LABEL et
  prend pour valeur des constantes-étiquettes du programme

2. Point d'entrée

- constante : étiquette d'instruction PROC ou ENTRY
  Dans certains cas devra être déclarée avec l'attribut ENTRY
- variable : attributs ENTRY VARIABLE
  prend pour valeur des contantes point-d'entrée du programme

SECTION 1.2

ORGANISATION DES DONNEES

§ . 1 . 2 . 1 AGREGATS

1. Variable élémentaire

ou élément ou scalaire

A un instant donné, le nom est associé à une valeur unique.

2. Agrégats

A un instant donné, au nom de l'agrégat est associé un ensemble de valeurs.

Cet ensemble est organisé de façon à établir des relations d'ordre entre les éléments et permettre l'identification des éléments ou de sous-ensembles.

Ces agrégats pourront donc être manipulés

- . par éléments
- . par ensemble
- . par sous-ensemble

On distingue

- . les structures (cf. ch. IX)
- . et les tableaux

3. Remarque fondamentale

Tous les types de données manipulables en PL/I peuvent être organisés en agrégat.

...

§ . 1 . 2 . 2 TABLEAU

1. Définition

Ensemble ordonné

selon une ou plusieurs directions (dimensions)

d'éléments de données

d'attributs identiques

jouant un rôle équivalent dans le traitement

Le tableau seul reçoit un nom

2. Attribut de dimensionnement

- forme : nom-tableau (D1, ..., Di, ..., D15)
- règles :
  - . jusqu'à 15 dimensions
  - . dans chaque dimension, Di, on précise les bornes inférieures et supérieures de variation sous la forme

    $A_i : B_i$ $\quad (A_i \leq B_i)$
  - . A_i et B_i peuvent être des constantes entières de -32768 à +32767 ou dans certains cas des expressions ou * .
  - . si $A_i = 1$, on peut se contenter d'écrire B_i

    $B_i \Leftrightarrow 1 : B_i$
- Exemples

A (3,3,3)	DEC FIXED (5,3)	27 éléments
B (-20:19)	BIT (1)	40 éléments
LAB (Ø:9)	LABEL	10 éléments

§ . 1 . 2 . 3 MANIPULATION D'UN ELEMENT

1. Identification d'un élément

Un élément est identifié par

- . le nom du tableau
- . suivi, entre parenthèse, d'une liste de valeurs, une dans chacune des dimensions.

  Ces valeurs ou indices représentent les coordonnées de l'élément.

...

2. Indice ; expression indicielle

- un indice doit avoir une valeur entière comprise entre les bornes déclarées de la dimension
- cette valeur peut s'exprimer au moyen d'une expression quelconque pourvue qu'elle soit calculable ; seule la partie entière de sa valeur est utilisée.

3. Exemple

A (1,2,1)

B (I,3*J+Z)

LAB (3*COS(X)*(U=1))

4. Disposition en mémoire

- les éléments sont disposés dans des positions de mémoire contigües, selon l'ordre naturel de lecture ou d'écriture, c'est-à-dire par lignes et, à l'intérieur, par numéro de colonne croissant

A (3,3) $a_{11}\ a_{12}\ a_{13}\ a_{21}\ a_{22}\ a_{23}\ a_{31}\ a_{32}\ a_{33}$

- d'une manière plus générale, la disposition se fait de sorte que la fréquence de variation d'indice croit de gauche à droite

A (I,J,...,N) $\quad i = 1,\ldots,I \left\{ j = 1,\ldots,J \right\{ \ldots \left\{ n = 1,\ldots,N$

§ . 1 . 2 . 4 SOUS-TABLEAUX

1. Définition

Dans un tableau à plusieurs dimensions, un sous-tableau est obtenu en fixant une ou plusieurs dimensions et en accordant la totalité du champ de variation aux autres dimensions

On donne une valeur aux dimensions que l'on veut fixer ; celles qui restent libres dans leurs variations sont notées par un * .

2. Exemple

Soit le tableau TAB (3,3)

Les lignes seront TAB (1,*) TAB (2,*) et TAB (3,*)

Les colonnes seront TAB (*,1) TAB (*,2) et TAB (*,3)

TAB (*,*) représente le tableau et est équivalent à TAB

...

```
 1      PO: PROC OPTIONS(MAIN);
        /*
        *    SOUS-TABLEAUX
        */
 2          DCL TAB(3,3);
 3          L:  DO I=1 TO 3; DO J=1 TO 3;
 5              TAB(I,J)=10*I+J;
 6              END L;
 7          PUT PAGE;
 8          DO I=1 TO 3;
 9          PUT SKIP EDIT(TAB(I,*))(R(FOR));
10          END;
11          PUT SKIP(2);
12          DO I=1 TO 3;
13          PUT SKIP EDIT(TAB(*,I))(R(FOR));
14          END;
15          PUT SKIP(2) EDIT(TAB(*,*))(R(FOR));
16      FOR: FORMAT(SKIP,3 F(4));
17      END PO;
```

11	12	13
21	22	23
31	32	33

11	21	31
12	22	32
13	23	33

11	12	13
21	22	23
31	32	33

3. Remarque

Du fait de leur disposition en mémoire, les éléments d'un sous-tableau peuvent ne pas être contigus. Ce sera le cas des colonnes de TAB.

SECTION 1.3

DECLARATION DES VARIABLES

§ . 1 . 3 . 1 INSTRUCTIONS DECLARE

1. Principe

A tout identificateur de variables doivent être attachés des attributs. S'il apparaît dans une instruction DECLARE, suivi de la liste des attributs, ceux-ci lui seront attachés par le compilateur. Si, au contraire, le nom n'apparaît pas dans une instruction DECLARE, le compilateur lui associera automatiquement des attributs par défaut (Cf. § 1 . 3 . 8).

La déclaration devient donc obligatoire si les attributs par défaut ne conviennent pas à la variable.

2. Forme générale

DECLARE (ou DCL) nom liste-attributs, ..., nom liste-attributs ;

déclaration

3. Règles principales

- une instruction DCL peut contenir un nombre quelconque de déclarations. Une déclaration doit se faire à l'intérieur d'une seule DCL.
- une instruction DCL peut être placée n'importe où dans un bloc.
- les déclarations successives doivent être séparées par une virgule.
- le nom doit être séparé de sa liste-attributs.
- les attributs de cette liste doivent être séparés.

  Cette séparation se fait par ⊬ ou un autre séparateur tel que (ou

...

- les attributs peuvent être placés dans n'importe quel ordre, sauf,
  - . dimensionnement : à la suite du nom de tableau
  - . longueur : à la suite de CHAR ou BIT
  - . précision : à la suite de l'un quelconque des attributs numériques

4. Factorisation

Des attributs communs à plusieurs identificateurs peuvent être mis en facteur pour ces identificateurs.

```
DCL  ((A,B)(1Ø),C) FIXED(3)
     ((F,G)CHAR(1Ø), U BIT(3)) VAR ;
```

§ . 1 . 3 . 2 INITIALISATION DE VARIABLE

1. Rôle

Dans une déclaration, afin de donner une valeur initiale à une variable au moment où cette variable reçoit son emplacement mémoire.

2. Forme

En incorporant l'attribut INITIAL (ou INIT) à la déclaration du nom de la variable.

```
DCL ..., nom ... INIT (valorisation), ... ;
         \_____ déclaration _____/
```

3. Règles principales

- la valorisation peut se faire
  - . par une constante convenable
  - . par un * qui signifie l'absence de valorisation
  - . dans certains cas, par une expression ou un appel de sous-programme.
- cette valorisation sera unique si la variable est élémentaire. Pour un tableau, on pourra préciser plusieurs valeurs, une par élément.
- l'initialisation est interdite pour une variable redéfinie par DEFINED, mais autorisée pour la variable de base.

4. Initialisation d'un tableau

- on donne une valeur par élément. Ces valeurs doivent apparaître selon la disposition des éléments en mémoire.

...

- s'il manque des valeurs, les derniers éléments ne sont pas valorisés ; si elles sont trop nombreuses, le superflu est abandonné.

- si une même valeur doit être affectée à plusieurs éléments consécutifs, on la préfixe d'un facteur d'itération.

- forme de l'initialisation

INIT (..., (fact-itération) valorisation, ...)

valorisation ayant les 3 trois formes

. constante convenable

. * (pas de valorisation)

. expression ou appel de sous-programme.

5. Cas des tableaux de chaîne

- une constante chaîne pouvant s'exprimer avec un facteur de répétition, la confusion est possible avec le facteur d'itération de l'initialisation.

Règle

un seul niveau de parenthèse définit un facteur de répétition de la chaîne de base. Deux niveaux consécutifs définissent, de gauche à droite, le facteur d'itération d'initialisation, puis, le facteur de répétition de chaîne.

(2)'A' signifie 1 fois 'AA'

(2)('A') ou (2)(1)'A' signifient 2 fois 'A'.

6. Cas des variables étiquettes

- on peut

. initialiser une variable étiquette ou un tableau d'étiquette

DCL EX LABEL INIT (L1),

EY (3) LABEL INIT (L1, L2, L3) ;

. ou bien sans initialiser, préciser les valeurs (constante-étiquette) qu'une variable pourra prendre au cours de l'exécution

DCL EZ LABEL (L1, L2, L3) ;

7. Exemples

...

```
1     P1: PROC OPTIONS(MAIN);
      /*
      *    INITIALISATION DE TABLEAUX                    1 0 0 0
      */                                                 2 0 0 1
                                                         0 0 3 2
2           DCL A(4,4)  FIXED(3) INIT ( (2)(1,(3)0,2,    0 0   0
                        (2)0 ),3,0 );
3           PUT PAGE EDIT (A)( 4 F(5),SKIP);
-------------------------------------------------------------------
4           DCL B(10) CHAR(4) INIT((2) ((2)'AX',          AXAX
                              (2)('CZ'),(2)(6)'T') );     CZ
                                                          CZ
5           PUT SKIP(5) EDIT (B) (  A(4), SKIP);          TTTT
                                                          TTTT
6     END P1;                                             AXAX
                                                          CZ
                                                          CZ
                                                          TTTT
                                                          TTTT
COMPILER DIAGNOSTIC MESSAGES

ERROR ID L    STMT     MESSAGE DESCRIPTION

WARNING DIAGNOSTIC MESSAGES

IELC892I W    4        TARGET STRING SHORTER THAN SOURCE.
                       RESULT TRUNCATED ON ASSIGNMENT.
```

§ . 1 . 3 . 3 REDEFINITION DE VARIABLE

1. Principe

Etant donné une variable de base occupant un emplacement mémoire, redéfinir sur cette variable de base une nouvelle variable occupant tout ou partie du même emplacement.

2. Forme de la redéfinition

DCL variable-base liste-attributs, ... ;

DCL variable-redéfinie liste-attributs DEFINED variable-base
(DEF)

...

3. Règles principales

- il doit y avoir concordance, de type en particulier, entre la variable de base et la variable redéfinie.

- les attributs de la variable de base ne sont pas implicites ; ils doivent être déclarés.

- la variable redéfinie ne doit pas déborder de la variable de base.

- une variable redéfinie exclut les attributs INIT, de portée, de classe mémoire ... qui sont ceux de la variable de base.

- les deux déclarations peuvent se faire dans la même DCL.

§ . 1 . 3 . 4 REDEFINITION PAR CORRESPONDANCE

1. Règles

- les deux variables ont même origine.

- s'il s'agit de tableau numérique, ils ont même nombre de dimensions

- s'il s'agit de chaînes, elles sont toutes deux de même type et toutes les deux VARYING ou non-VARYING.

- un tableau de chaînes et une chaîne peuvent être en redéfinition.

2. Exemples

voir ci-après les exemples

...

```
1     P2: PROC OPTIONS(MAIN);

      /*
      *    REDEFINITION PAR
      *            CORRESPONDANCE
      */

      /*  VARIABLES DE BASE    ....*/

2         DCL A CHAR(10) VAR,
              B CHAR(10) INIT ('CARACTERES'),
              C (3,3);

      /*  VARIABLES  REDEFINIES   */

3         DCL AR CHAR(5) VAR DEF A,
              BR(4) CHAR(2)   DEF B,
              CR (2,2)        DEF  C ;

4         DO I=1 TO 3; DO J=1 TO 3;
6         C(I,J)=10*I+J;
7         END; END;

9         AR = 'DEFINED';

10        PUT PAGE;
11        PUT SKIP EDIT(LENGTH(A),LENGTH(AR))(2 F(5));
12        PUT SKIP EDIT (A,AR)(A,X(3),A);
13        PUT SKIP(3) EDIT(B,BR)(A,SKIP,5 A(3));


14        PUT SKIP(3) EDIT(C,CR)(3(3 F(4),SKIP),
                                 SKIP(2),2(2 F(4),SKIP));
```

A	DEFIN (5)	DEFIN (5)	AR
B	CARACTERES		
BR	CA	RA	CT ER
C	11	12	13
	21	22	23
	31	32	33
CR	11	12	
	21	22	

```
SEVERE AND ERROR DIAGNOSTIC MESSAGES

IEL0495I E   3      AREA SIZE OR MAXIMUM LENGTH OF VARYING STRING IN SIMPLE DEFINED ITEM 'AR'
                    DIFFERS FROM THAT OF THE CORRESPONDING BASE.    RESULTS OF EXECUTION UNDEFINED.
```

§ . 1 . 3 . 5 REDEFINITION PAR RECOUVREMENT

1. Règles

- s'applique entre des variables de type chaîne de même nature.
- l'origine de la variable redéfinie est précisée à l'intérieur de la variable de base par l'attribut POSITION (origine).
- l'attribut VARYING est exclu.
- l'origine est exprimée par une constante ou une expression dans certains cas.

...

2. Exemples

```
        /*
        *    ATTRIBUTS
        *       DEFINED    ET    POSITION
        */
13              DCL     R    CHAR(11),
                        S(11)    CHAR(1) DEFINED R    ,
                        RA  CHAR(3) DEF R    POSITION(1),
                        RB  CHAR(5) DEF R    POS(7) ;

14              R = 'PROPOSITION';
15              PUT SKIP(2) EDIT(S) (11 A(2));
16              PUT SKIP DATA(RA,RB);

17              RA = ' AP';
18              PUT SKIP(2) EDIT(S) (11 A(2));
19              PUT SKIP DATA(R,RB);

20              RB = 'ER';
21              PUT SKIP(2) DATA(R);

22              DO I = 1 TO 11 BY 2 ;
23              S(I) = '*' ;
24              END;
25              PUT SKIP(2) DATA(R,RA,RB);
```

```
P R O P O S I T I O N
RA='PRO'                    RB='ITICN';
 A P P O S I T I O N
R=' APPOSITION'             RB='ITICN';
R=' APPOSER   ';
R='*A*P*S*R* *'             RA='*A*'                    RB='*R* *';
```

§ 1 . 3 . 6 REDEFINITION PAR ISUB

1. Règles

- sur un tableau de base, on redéfinit un autre tableau
  - . de même type
  - . mais dont le nombre de dimensions et les bornes peuvent être différents.

...

- pour cela, la partie du tableau de base recouverte est définie par une expression générique de ses éléments. Ces expressions font intervenir des variables spéciales : ISUB (I = 1,2, ...).

ISUB figure toutes les valeurs possibles, entre les bornes inférieure et supérieure de la dimension de rang i du tableau redéfinie.

2. Exemple n° 1

DCL A(3,3), B(3) DEF A(1SUB, 1SUB)

B est à une dimension ; les valeurs d'indice représentées par 1SUB seront 1,2 et 3, de sorte que le recouvrement des éléments de A par B s'effectue de la façon suivante :

. B (1) ⇒ 1SUB = 1 ⇒ A (1,1)

. B (2) ⇒ 1SUB = 2 ⇒ A (2,2)

. B (3) ⇒ 1SUB = 3 ⇒ A (3,3)

B redéfinit donc le tableau à 1 dimension formé des éléments de la diagonale principale de A.

```
1     P3: PROC OPTIONS(MAIN);
2         DCL A(3,3) CHAR(5),
              B(3) CHAR(2) DEF  A(1SUB,1SUB);
3         GET LIST(A);
4         PUT SKIP(2) EDIT(A)(3 A(6),SKIP(2));
5         PUT SKIP(2) EDIT(B) (3 A(6));
6     END P3;
```

```
JANVI FEVRI MARS
AVRIL MAI   JUIN
JUILL AOUT  SEPTE

JA    MA    SE
```

...

3. Exemple n° 2

```
1     BISUB: PROC OPTIONS(MAIN);

      /*
      *   CONSTRUCTION DES TABLEAUX
      *
      *   100000     111000     110000
      *   010000     111000     110000
      *   001000     111000     001100
      *   000100     000111     001100
      *   000010     000111     000011
      *   000001     000111     000011
      *
      *      CHAINES  DE  BITS
      *      TABLEAUX  PAR  ISUB
      */


2         DCL    A(6,6)  BIT(1) ;

      /*  REDEFINITION DES TABLEAUX PARTIELS DANS  A  */

3     DCL DIAGOP(6)  BIT(1)  DEF A(1SUB,1SUB),
          CARSUP(3,3) BIT(1)  DEF A(1SUB,2SUB),
          CARINF(3,3) BIT 1)  DEF A(3+1SUB, 3+2SUB),
          TROIS(3,2,2)    BIT(1)  DEF A((1SUB-1)*2+2SUB,
                                        (1SUB-1)*2 + 3SUB) ,
          B(6)     BIT(6)  DEF A;


4         PUT SKIP LIST('TABLEAU NO 1');
5         A = '0'B ;
6         DIAGOP = '1'B ;
7             DO I = 1 TO 6;
8             PUT SKIP LIST(B(I));
9             END;

10        PUT SKIP(3) LIST('TABLEAU NO 2');
11        B = '000000'B ;
12        CARSUP, CARINF = '1'B ;
13            DO I = 1 TO 6;
14            PUT SKIP LIST(B(I));
15            END;

16        PUT SKIP(3) LIST('TABLEAU NO 3');
17        B = (2)'0'B || (4)'0'B ;
18        TROIS = '1'B ;

19            DO I = 1 TO 6 ;
20            PUT SKIP LIST(B(I));
21            END;

22        PUT SKIP(3) LIST('FIN');
23    END BISUB;
```

```
TABLEAU NO 1
'100000'B
'010000'B
'001000'B
'000100'B
'000010'B
'000001'B


TABLEAU NO 2
'111000'B
'111000'B
'111000'B
'000111'B
'000111'B
'000111'B


TABLEAU NO 3
'110000'B
'110000'B
'001100'B
'001100'B
'000011'B
'000011'B


FIN
```

...

4. Remarques

- en général, les éléments d'un tableau redéfini par ISUB ne sont pas contigus.
- l'emploi des ISUB est coûteux en temps de compilation.

§ 1 . 3 . 7 ATTRIBUTS PAR DEFAUT

1. Attributs standards du système

En l'absence de toute déclaration, si l'initiale est

I,J,K,L,M ou N	REAL BIN FIXED (15,Ø)
A à H, O à Z, $, @, #	REAL DEC FLOAT (6)

2. Implication d'attribut

BIN ou DEC seuls entraînent FLOAT

FIXED ou FLOAT seuls entraînent DEC.

§ . 1 . 3 . 8 INSTRUCTION DEFAULT, OPTION RANGE

1. Rôle

Elle permet de définir, dans sa portée, des attributs par défaut et les classes d'identificateurs auxquels ces attributs seront associés. Une classe d'identificateurs est composée des identificateurs dont le ou les premiers caractères sont identiques en valeur et en rang.

2. Forme

DEFAULT RANGE (classe) attributs ;
DFT

classe peut prendre deux formes :

L : L'	comprend tous les identificateurs qui ont pour initiale les lettres de L à L'
XY...Z	comprend tous les identificateurs commençant par la "syllabe" XY...Z.

...

attributs — sous la forme d'une liste d'attributs qui s'appliqueront aux identificateurs de la classe, en l'absence de déclarations par DCL.

si tous les attributs ne sont pas prévus, ceux impliqués en standard par le système complèteront ces attributs.

des longueurs ou précisions apparaissent dans l'option VALUE (attribut)

3. Exemple

```
1    P4: PROC OPTIONS(MAIN);
     /*
     *   D E F A U L T
     */
2        DEFAULT  RANGE (TAB) (6) CHAR  VALUE(CHAR(3));
3        DFT      RANGE  (A:B, U:V) FIXED;
4        DCL AB(3) FLOAT(12), UV BIT(3);
5            PUT LIST(TABA, BAT);
6            PUT LIST(AU,VA);
7    END P4;
```

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
4	AB	(3) AUTOMATIC ALIGNED DECIMAL /* DOUBLE */ FLOAT (12)
********	AU	AUTOMATIC ALIGNED DECIMAL FIXED (5,0) 6
********	BAT	AUTOMATIC ALIGNED DECIMAL FIXED (5,0) 5
1	P4	EXTERNAL ENTRY RETURNS(DECIMAL /* SINGLE */ FLOAT (6))
********	SYSPRINT	EXTERNAL FILE PRINT 5,6
********	TABA	(6) AUTOMATIC UNALIGNED CHARACTER (3) 5
4	UV	AUTOMATIC UNALIGNED BIT (3)
********	VA	AUTOMATIC ALIGNED DECIMAL FIXED (5,0) 6

CHAPITRE 2

EXPRESSIONS
INSTRUCTION D'AFFECTATION

SECTION 2.1

OPERATIONS

§ . 2 . 1 . 1 OPERATIONS ARITHMETIQUES

1. Opérateurs

+ et - addition et soustraction comme opérateurs infixe ou préfixe

* et / multiplication et division

** élévation à la puissance

2. Opérandes

- les opérandes doivent être de type numérique ou convertibles en type numérique. Ainsi :

'12' * '3.4' est calculable

3. Règles principales

- les opérateurs + et - utilisés comme préfixes peuvent être précédés d'une autre opérateur

A * - B est valide

...

- la règle des signes s'applique

- l'opérateur * doit toujours être explicité

4. Remarque

Les fonctions ADD, MULTIPLY et DIVIDE permettent de réaliser l'opération en contrôlant la précision (cf. ch. 8).

5. Exemple

```
1     PB1:  PROC OPTIONS(MAIN);
          /*
          *   CONVERSIONS EN ''CHAR''
          */
2         DCL (Z0,Z1,Z2,Z3,Z4) DEC FIXED(5);
3         Z0= '123' + '249';
4         Z1= '011'B + '1000'B;
5         Z2= '1.E+2' + '11.E+2B';
6         Z3= '1.E+2' + '110'B;
7         PUT SKIP LIST ('ARITHMETIQUE SUR CHAINES');
8         PUT SKIP(2) DATA (Z0,Z1);
9         PUT SKIP DATA(Z2,Z3);
```

```
ARITHMETIQUE SUR CHAINES
Z0=      372                  Z1=       11;
Z2=      112                  Z3=      106;
```

§ . 2 . 1 . 2 OPERATIONS BOOLEENNES

1. Opérateurs

- ¬ opérateur de négation : NON
- & opérateur d'intersection : ET
- | opérateur de réunion : OU inclusif

...

2. Opérande(s)

Chaînes de bits ou convertibles en chaînes de bits ; en particulier, le résultat d'une comparaison qui est '0'B ou '1'B.

3. Règles

¬ opérateur préfixe qui inverse la valeur de chaque bit

¬ '01'B ⇒ '10'B

& opérateur infixe : le bit résultat est '1'B si les 2 bits opérandes sont '1'B ; il est '0'B dans les autres cas

'0011'B & '0101'B ⇒ '0001'B

| opérateur infixe : le but résultat est '0'B si les deux bits opérandes sont '0'B ; il est '1'B dans les autres cas

'0011'B | '0101'B ⇒ '0111'B

- l'opérande le plus court d'une opération infixe est complèté à droite de '0'B.

§ . 2 . 1 . 3 OPERATION DE CHAINAGE

1. Opérateur

II réalise la juxtaposition des deux chaînes opérandes.

2. Opérandes

Doivent être de type chaîne ou convertibles en chaîne. Si l'un des opérandes est CHAR, le deuxième sera converti en chaîne CHAR. Si un seul des opérandes est BIT, la conversion se fera en BIT.

3. Règles

- le résultat est la nouvelle chaîne obtenue par la juxtaposition des opérandes ou des résultats de conversion

- la longueur du résultat est la somme des longueurs des opérandes (après conversion)

- en général, l'opération n'est pas commutative.

...

```
10          PUT PAGE LIST((3)'123456789 ');


11          DCL (C1,C2,C3,C4) CHAR(15) VAR,
                IU FIXED(3), U FIXED(3,1);
12              U = 12.5;
13              IU = U;

14              C1 = 'CHAR'|| U;
15              C2 = 'CHAR'|| IU;
16              C3 = 'CHAR'|| 1.25E+1 ;
17              X = 12.5;
18              C4 = 'CHAR'|| X ;

19          PUT SKIP(2)LIST('CHAR || NUMERIQUE');
20          PUT SKIP(2)DATA(C1);
21          PUT SKIP DATA(C2);
22          PUT SKIP (2) DATA (C3);
23          PUT SKIP DATA(C4);
```

```
123456789 123456789 12345
CHAR || NUMERIQUE
C1='CHAR 12.5';
C2='CHAR  12';
C3='CHAR 1.25E+01';
C4='CHAR 1.25000E+0';
```

```
        /*
        *   CONVERSIONS  DE  CHAINAGE
        *           AVEC     BIT
        */
24          PUT PAGE LIST((3)'123456789 ');


25          DCL (C5 CHAR(15), B5 BIT(25) )VAR;
26          PUT SKIP(2)LIST('CHAR || BIT');

27          C5 = 'CHAR' || '1011'B;
28          PUT SKIP(2) DATA (C5);
29          PUT SKIP(2)LIST('NUMERIQUE || BIT');
30          DCL B4 BIN FIXED;
31          B4 = 101;
32          B5 = B4 || '1011'B;
33          PUT SKIP(2) DATA(B5);
34          B5 = 101 || '1011'B;
35          PUT SKIP(2)DATA(B5);
36          B5 = 12.5 || '1011'B;
37          PUT SKIP(2) DATA (B5);
```

```
IEL0906I I   34, 36    DATA CONVERSION WILL BE DONE BY SUBRCUTINE CALL.
```

```
123456789 123456789 123456789
CHAR || BIT
C5='CHAR1011';
NUMERIQUE || BIT
B5='C00|0000|0110|0101|1011'B;

IBM0351I  'ONCODE'=0615  'CONVERSION' CONDITION RAISED
   CONVERSION FROM CHARACTER TO BIT
   IN STATEMENT 34 AT OFFSET +0004E8 IN PROCEDURE WITH ENTRY PB1
```

§ . 2 . 1 . 4 OPERATIONS DE COMPARAISON

1. Opérateurs

=	égal à
¬=	non égal à (différent de)
<	inférieur à
¬<	non inférieur à
<=	inférieur ou égal à
>	supérieur à
¬>	non supérieur à
>=	supérieur ou égal à

2. Opérandes

On peut comparer entre elles

- des données-problèmes
- des données-contrôle-de-programme uniquement par = ou ¬=

3. Classes de comparaison

Pour les données-problèmes, il existe trois classes :

- classe supérieure : comparaison algébrique; valeur algébrique signée (cas des données numériques réelles).

- classe intermédiaire : comparaison caractère; en principe entre chaînes CHAR. Elle se fait selon l'ordre lexicographique du 370, octet par octet, de gauche à droite. La chaîne la plus courte est allongée de blancs à droite.

- classe inférieure : comparaison binaire; en principe entre chaînes BIT, de gauche à droite, bit par bit. L'allongement, à droite de la chaîne la plus courte, se fait par 'Ø'B.

4. Conversion, choix de la classe

Si les opérandes ne sont pas de même type, l'opérande correspond à la classe inférieure est converti, si possible, de façon à permettre la comparaison au niveau supérieur.

numérique et CHAR : CHAR ⇒ numérique

numérique et BIT : BIT ⇒ numérique

CHAR et BIT : BIT ⇒ CHAR.

...

5. Résultat d'une opération de comparaison

Le résultat, VRAI ou FAUX, est représenté conventionnellement par une chaîne de bits, de longueur 1 :

VRAI ⟺ '1'B et FAUX ⟺ 'Ø'B

On peut donc utiliser une expression de comparaison comme opérande d'une opération booléenne.

6. Remarque

Des valeurs numériques complexes ne peuvent être comparés que par $=$ ou $\neg=$.

7. Exemple

```
1     PB2: PROC OPTIONS(MAIN);
      /*
      *    CONVERSIONS DANS
      *            LES COMPARAISONS
      */
2          DFT RANGE (B) BIT  VALUE(BIT(1));
3          B1 = 'A' = '11000001'B;
4          PUT PAGE DATA (B1);
5          B2 = 193 = '11000001'B;
6          PUT SKIP DATA(B2);
```

```
B1='0'B;
B2='1'B;
```

§ . 2 . 1 . 5 HIERARCHIE DES OPERATEURS

1. Les sept niveaux d'opérateurs

Les opérateurs PL/I sont classés en sept niveaux qui comprennent, du niveau de plus haute priorité vers celui de plus basse priorité :

- 1er niveau , plus haute priorité : ** + préfixe - préfixe
- 2e niveau : * /
- 3e niveau : + infixe - infixe
- 4e niveau : ||

...

- 5e niveau : opérateurs de comparaison
- 6e niveau : &
- 7e niveau, plus basse priorité : |

2. Réalisation d'opérateurs consécutifs

En l'absence de parenthèses, si les deux opérateurs consécutifs sont :

- de niveaux différents, celui de plus haut niveau est d'abord réalisé avec les opérandes qui l'entourent,

- de même niveau autre que le premier niveau, le 1er rencontré c'est-à-dire celui de gauche est d'abord réalisé avec les opérandes qui l'entourent,

- de 1er niveau, le second, celui de droite, est d'abord réalisé avec les opérandes qui l'entourent.

Conséquence

Le résultat final dépend de l'opérateur de plus bas niveau, donc réalisé en dernier, et des attributs de ses opérandes.

3. Niveaux de parenthèses

- une paire de parenthèses modifie la séquence de réalisation des opérations et par le fait même peut modifier le résultat final.

4. Cas d'une référence de fonction

Elle est réalisée (par appel de la fonction) avant tout autre opérateur et la valeur retournée prend sa place dans la séquence des opérandes.

SECTION 2.2

EXPRESSIONS

§ . 2 . 2 . 1 TYPES D'EXPRESSIONS

1. Définition

Elle représente une ou plusieurs valeurs (cas d'agrégats) et s'exprime au moyen d'un nombre fini d'opérateurs, de paires de parenthèses et d'opérandes qui peuvent être

- des constantes, des variables

- des références de fonctions.

2. Expression scalaire ou élémentaire

- son évaluation restitue une valeur unique

- ses opérandes seront des scalaires mais des agrégats pourront apparaître dans l'expression, comme arguments de fonctions restituant une valeur élémentaire.

3. Expression-tableau

- le résultat est un tableau de valeurs

- l'un des opérandes au moins est un tableau ; les autres peuvent être des scalaires.

4. Expression-structure

(cf. ch. 9)

...

§ . 2 . 2 . 2 REALISATION D'UNE EXPRESSION

1. Cas général

- les opérations qui définissent l'expression sont réalisées en général de gauche à droite, en respectant la hiérarchie des opérateurs

- à la réalisation de chaque opération, les opérandes sont éventuellement convertis et un résultat intermédiaire est construit qui sera repris dans la poursuite de l'évaluation.

2. Cas des sous-expressions

- une paire de parenthèses définit une sous-expression qui, elle-même, peut contenir d'autres sous-expressions

- la sous-expression la plus interne est d'abord évaluée conformément aux règles habituelles ; elle est alors remplacée par sa valeur
- puis l'évaluation se poursuit en remontant la hiérarchie des sous-expressions jusqu'au résultat final.

§ . 2 . 2 . 3 EXPRESSION-TABLEAU

1. Règles fondamentales

- l'un des opérandes doit être un (sous-)tableau

- alors, tous les tableaux opérandes doivent avoir des attributs de dimensionnement identiques, c'est-à-dire

  - même nombre de dimensions
  - dans chaque dimension
    - même borne inférieure
    - même borne supérieure

- le compilateur décompose la manipulation des tableaux en une séquence de manipulations sur les éléments en correspondance. Cette séquence est organisée selon le schéma de disposition des éléments en mémoire.

2. Scalaire et tableau

L'opérateur applique le scalaire à chaque élément du tableau, selon le processus défini ci-dessus. Le scalaire peut être l'un des éléments du tableau.

...

DCL A(5,3)

$$A + 2 \iff \{A(I,J)+2\} \quad \forall I = 1,..,5 \quad \{ \forall J = 1,..,3$$

3. Tableau et tableau

L'opérateur s'applique entre les éléments correspondants des tableaux-opérandes selon le processus défini ci-dessus.

DCL (A,B)(5,3)

$$A + B \iff \{A(I,J)+B(I,J)\} \quad \forall I = 1,..,5 \quad \{\forall J = 1,..,3$$

4. Résultat d'une expression tableau

C'est un tableau de valeurs, organisées selon les attributs de dimensionnement communs aux tableaux-opérandes.

5. Exemple

```
      /*
      *    EXPRESSIONS-TABLEAUX
      */

3           DCL (A(2,3) INIT(1,2,3,4,5,6),
                 B(2,3) INIT(16,2,555,4,11,6) )FIXED(3);

4           PUT PAGE EDIT ( A*(B+1))(3 F(5),SKIP);

5           PUT SKIP(3)LIST((10)'123456789 ');
6           PUT SKIP(2) EDIT(A || B)(3 (A,X(1)),SKIP);
7           PUT SKIP(2)LIST((10)'123456789 ');

8           PUT SKIP(3) EDIT (A = B)(3 B(2),SKIP);
```

```
 17     6 1668
 20    60   42

123456789 123456789 123456789 123456789 123456789
    1    16     2     2     3   555
    4     4     5    11     6     6
123456789 123456789 123456789 123456789 123456789

0 1 0
1 0 1
```

SECTION 2.3

INSTRUCTION D'AFFECTATION

INTRODUCTION AUX CONVERSIONS

---

§ . 2 . 3 . 1 INSTRUCTION D'AFFECTATION

1. Rôle et forme

[préfixes :] cible = source ;

Construite sur le symbole d'affectation " = ", cette instruction a pour but d'affecter la valeur de la source à la cible, c'est-à-dire de placer cette valeur dans les positions de mémoire de la cible.

2. Règles générales

préfixes : représentent un ensemble de préfixes conditions et/ou d'étiquettes

cible : une variable scalaire ou agrégat, ou une pseudo-variable

source : en général, une expression qui peut se réduire à une simple constante

- l'expression de la source est d'abord évaluée, puis sa valeur est affectée à la cible

- si la source restitue une valeur d'agrégat, la cible doit être un agrégat convenable

- dans le cas de données-problème, les attributs de la cible peuvent différer de ceux de la valeur de la source ; il y aura alors tentative de conversion de cette valeur dans les attributs de la cible.

...

3. Affectation multiple

Si plusieurs cibles doivent recevoir la même valeur, elles peuvent apparaître dans une seule instruction d'affectation multiple

$cible_1$, ..., $cible_n$ = source ;

§ . 2 . 3 . 2 QUELQUES CAS PARTICULIERS

1. Cas de variables FIXED

La valeur est cadrée sur la position de virgule ; il peut donc y avoir perte de chiffres à droite et à gauche.

2. Cas de variables chaînes

La valeur affectée est cadrée à gauche et l'ajustement se fait à droite sur la longueur de la variable cible.

3. Données contrôle de programme

La cible et la source doivent être de même type. Ce sera le cas d'étiquettes, de points d'entrée, de fichiers... Seules des variables localisatrices peuvent être converties entre elles.

4. Remarque

L'instruction X = A = B ;

n'est pas une instruction d'affectation multiple. Le symbole = de droite représente l'opérateur de comparaison par égalité alors que celui de gauche est le symbole d'affectation.

X reçoit donc le résultat de la comparaison entre A et B, c'est-à-dire '0'B ou '1'B.

5. Exemple

```
        /*
        *   AFFECTATIONS  PARTICULIERES
        */
12      DCL   (A,B,C) FIXED(3) , X BIT(3),
              (U,V) CHAR(5) VAR INIT ('MAI'),
               L  LABEL;

13      L1: B,C = 543;
14          A = B+C;
15          PUT PAGE EDIT (A,B,C) (3 F(6));     86    543    543
16          L = L2;
17          GOTO TR;

18      L2: V = U||V;
19          PUT SKIP(5)EDIT(U,V)(2 A(6));       MAI    MAIMA
20          L = L3 ;
21          GOTO TR;

22      L3: X = A = B;
23          PUT SKIP(5)LIST(X);                 '000'B
24          L = SUITE;
25          GOTO TR;

26      TR:  GOTO L;
```

§ . 2 . 3 . 3 AFFECTATION D'AGREGAT

1. Cas des tableaux

La cible étant un tableau

- si la source est une expression-tableau, les attributs de dimensionnement de la cible et de la source doivent être identiques. L'affectation se fait alors entre éléments mis en correspondance

- si la source est une expression-scalaire, sa valeur est affectée à chaque élément de la cible.

Le compilateur décompose l'instruction d'affectation en une suite d'instructions d'affectation portant sur les éléments en correspondance et organisée selon la séquence des éléments en mémoire.

2. Cas des structures

(cf. ch. 9)

§ . 2 . 3 . 4 INTRODUCTION AUX CONVERSIONS

1. Principe

D'une manière générale, le compilateur PLIOPT autorise un grand nombre de conversions, conformément aux deux principes suivants.

- attributs d'opération

Dans une opération, les opérandes doivent avoir des attributs conformes et le résultat intermédiaire généré aura des attributs déduits de ceux des opérandes.

- attributs d'affectation

La valeur de la source doit avoir des attributs conformes à ceux de la cible

Au cas où la conversion tentée ne peut aboutir, une interruption apparaît ; elle provoque l'arrêt du programme sauf si elle est gérée dans une unité ON CONVERSION.

2. Cas principaux

- évaluation d'une expression quel qu'en soit le type

...

- affectation de valeur dans
  - . une instruction d'affectation
  - . une lecture, une écriture
  - . une initialisation par INIT
  - . des instructions IF, DO...

3. Les conversions de type

Les conversions peuvent porter sur la plupart des attributs.
Les principales conversions de type possibles sont

BIT ⇒ CHAR : les bits 'Ø'B et '1'B sont dans les caractères 'Ø' et '1'

CHAR ⇒ BIT : conversion réciproque. Seuls les caractères 'Ø' et '1' sont convertibles

CHAR ⇒ numérique : si la chaîne de caractères représente une constante numérique valide

numérique ⇒ CHAR : toujours possible

BIT ⇒ numérique : la chaîne de bits est d'abord convertie en en BIN FIXED qui est convertie en une chaîne de bits.

§ . 2 . 3 . 5 EXEMPLE D'EVALUATION AVEC CONVERSION

1. Exemple

```
DCL   R BIT(3), A FIXED(1), B BIN FIXED(3)
      C CHAR(2), D BIT(4) ;

R  =  A + B < C & D
```

2. Chaîne des opérations et des conversions

a) addition A + B

B est BIN FIXED(3) donc A est converti en BIN FIXED(3)
Le résultat R_1 est BIN FIXED (15)

b) comparaison R_1 à C

R_1 est de type numérique donc C va être converti en BIN FIXED
Le résultat de la comparaison R_2 est BIT (1)

c) opération R_2 & D

Puisque D est BIT(4), R_2 est étendu à une chaîne de 4 bits par 3 zéros à droite

R_2 & D ⟶ R_3 BIT(4)

...

d) affectation de R_3 à R

L'affectation se fait avec troncature du 4ème BIT de R_3

3. Résultats

La fonction UNSPEC utilisé dans ce programme permet d'obtenir la configuration binaire interne de l'argument.

```
      /*
      *   CONVERSIONS DANS L'EVALUATION
      *       D'UNE   EXPRESSION
      */
30    DCL   R BIT(3), A FIXED(1), B BIN FIXED(3),
            C CHAR(2), D BIT(4);
31          GET LIST (A,B,C,D) COPY;                  5, 3, '9', '101'B

32          PUT PAGE LIST(A, UNSPEC(A));              5          '01C111C0'B
33          PUT SKIP LIST(B, UNSPEC(B));              3          '0000000000000011'B
34          PUT SKIP LIST(C, UNSPEC(C));              9          '1111100101000000'B
35          PUT SKIP LIST(D, UNSPEC(D));              '1010'B    '1010'B

36          PUT SKIP(3) LIST('1');                    1
37          PUT SKIP LIST(A+B,UNSPEC(A+B));             8        '00C000CCC0001CC0'B

38          PUT SKIP(3) LIST('2');                    2
39          PUT SKIP LIST(A+B<C,UNSPEC(A+B<C));       '1'B       '1'B

40          PUT SKIP(3) LIST('3');                    3
41          PUT SKIP LIST(A+B<C&D,UNSPEC(A+B<C&D));   '1000'B    '1000'B

42          R = A+B<C&D;                              4
43          PUT SKIP(3) LIST('4');                    '100'B     '100'B
44          PUT SKIP LIST( R, UNSPEC(R));
```

WARNING DIAGNOSTIC MESSAGES

IEL0892I W 42, 42 TARGET STRING SHORTER THAN SOURCE.
RESULT TRUNCATED ON ASSIGNMENT.

COMPILER INFORMATORY MESSAGES

IEL0906I I 39, 39, 41, 41, 42 DATA CONVERSION WILL BE DONE
BY SUBROUTINE CALL.

CHAPITRE 3

INSTRUCTIONS ET FONCTIONS DU PL/I DE BASE

SECTION 3.1

LES TRANSMISSIONS EN MODE STREAM

§ . 3 . 1 . 1 LES TRANSMISSIONS EN PL/I

1. Les transmissions

- elles constituent l'ensemble des échanges d'information entre la mémoire interne et des supports externes. Ce sont les lectures ou entrées, à destination du programme stocké en mémoire interne et les écritures ou sorties sur support externe,

- le PL/I supporte deux modes de transmissions : le mode RECORD et le mode STREAM,

- dans ce premier sous-ensemble PL/I, les transmissions seront limitées au mode STREAM, entre un lecteur de cartes, en entrée, et une imprimante, en sortie.

...

2. Caractéristiques du mode STREAM

- sur le support externe, les données se présentent
  - . sous la forme d'un flot continu et séquentiel de valeurs (c'est-à-dire de constantes)
  - . exprimées sous forme caractère, indépendamment de la forme interne
  - . et apparamment sans organisation si ce n'est consécutive
- les transmissions se font
  - . valeur par valeur, selon la chronologie des instructions de transmission et la séquence des variables concernées,
  - . et en ignorant apparamment les frontières physiques du support externe. En particulier, la notion de carte de 80 colonnes disparaît en apparence mais, néanmoins, pourra être récupérée
- chaque transmission de valeur s'accompagne de conversion entre la forme caractère du support externe et la forme interne propre à chaque type d'information.

3. Conclusion

Le mode STREAM est un mode de transmission relativement coûteux mais d'un emploi beaucoup plus aisé que le mode RECORD. Il est donc adapté à la transmission des informations de faibles volumes.

§ . 3 . 1 . 2 INSTRUCTIONS DU MODE STREAM

1. Forme générale

- en lecture

[préfixes:] GET spécif-données [COPY] [SKIP (exp)] ;

- en écriture

[préfixes:] PUT spécif-données [SKIP (exp) | LINE (exp) | PAGE LINE (exp)] ;

- avec specif-données prenant 3 formes

DATA (listes-données)

LIST (liste-données)

EDIT (liste-données) (liste-formats)

...

- Dans les formes DATA et LIST le flot de transmission est quasiment libre de toute organisation programmée alors qu'au contraire avec la forme EDIT son organisation et son interprétation seront précisés dans le programme au moyen de liste-format.

- préfixes représente des étiquettes et/ou préfixes de condition.

2. Liste-données

- C'est l'ensemble des (pseudo-) variables concernées par l'instruction de transmission. En sortie, dans les formes LIST et EDIT, elle peut comporter des constantes et des expressions.

- les éléments de la liste sont séparés par une virgule ; la liste doit être placée entre parenthèse.

- Au moment de la transmission

  . la liste est explorée et réalisée de gauche à droite, selon l'ordre des éléments.

  . en entrée, à chaque élément est affectée sa valeur prise dans le flot de lecture, après conversion de la forme externe dans la forme interne de la variable. Dès qu'une valeur est ainsi lue elle est immédiatement disponible, et en particulier, peut servir à contrôler la valorisation des autres éléments de la liste.

  . en sortie, chaque valeur est convertie en forme caractère et insérée à sa place dans le flot de sortie.

3. Les options du GET

- COPY : précise que le flot de lecture doit être copié sur imprimante, tel qu'il se présente sur la carte lue

- SKIP [(exp)] indique que "exp" cartes sont abandonnées dans le flot de lecture, c'est-à-dire que la carte en cours de lecture est abandonnée et que les "exp - 1" suivantes sont sautées.
  SKIP est équivalent à SKIP (1)

...

4. Les options du PUT

- SKIP [(exp)] indique que "exp" lignes sont abandonnées dans le flot d'écriture c'est-à-dire que la ligne en cours est abandonnée et que les "exp-1" suivantes seront sautées. Il y aura donc "exp-1" lignes blanches.

  SKIP est équivalent à SKIP (1)

Si exp est < = Ø , le saut à la ligne suivante est inhibé ; il y aura donc sur-impression

- LINE (exp) provoque le saut à la ligne précisée par "exp" dans la page en cours. Si la ligne est déjà passée, on passe à la page suivante, première ligne.
  La page standard comporte 40 lignes de 120 positions.

- PAGE provoque l'abandon de la page en cours et le saut à la page suivante, à la ligne n° "exp" si LINE (exp) a été spécifiée ou à la 1ère ligne dans le cas contraire.

5. Remarques générales

- exp représente une expression calculable dont la partie entière du résultat converti en BIN FIXED (15) est utilisée au moment de la réalisation de l'option.

- Les options SKIP, LINE et PAGE peuvent apparaitre avant ou après la "spécif-données" ; elles sont toujours réalisées AVANT transmission.

- Les options SKIP et LINE , s'excluent mutuellement ; de même SKIP et PAGE.

- Les transmissions STREAM s'accompagnant de conversion, les tentatives avortées provoqueront une interruption et en général l'arrêt du programme.

...

§ . 3 . 1 . 3 FORME DATA

1. Liste-données

- Elle peut être omise. Dans ce cas, la transmission concerne la totalité des variables connues dans le bloc de l'instruction.

- En entrée, elle comporte des noms de variables élémentaires non-indicés et/ou des noms de tableau. Ces noms peuvent apparaitre dans n'importe quel ordre.

- En sortie, elle comporte des noms de variables, indicés ou non, des noms de tableau, des variables étiquettes, points d'entrée... Les constantes et les expressions sont interdites.

2. Flot de lecture

- Il se présente sur les cartes sous la forme d'une suite de "pseudo-affectations".

  variable-élémentaire = c^{te}, ... ;

- Celles-ci sont séparées de blanc (s) et/ou par une virgule. L'ensemble du flot destiné un GET DATA est clôt par un ";" .

- Variable-élémentaire doit se retrouver dans la liste-données du GET et s'il s'agit d'un élément de tableau, les indices doivent être des constantes.

- C^{te} doit être une constante valide

  - numérique décimale réelle ou complexe (I)
  - chaine de caractères ou de bits, sans facteur de répétition.

- La rencontre du ; interrompt la transmission, même si liste-donnée n'a pas été explorée en totalité.

3. Flot d'écriture

- la forme interne est convertie en caractère ; de plus les valeurs numériques binaires sont converties en décimal.

  Les chaines sont délimitées par une paire d'apostrophes. Le caractère apostrophe est dédoublé. Une chaine de bits est suivie de B.

- Ces valeurs apparaissent sous forme de pseudo-affectations :

  variable = valeur

  Sur la ligne d'impression celles-ci sont placées dans les zones prédéfinies.

- Un tableau est transmis selon la séquence de stockage de ses éléments en mémoire.

4. Exemples

```
        /*
        *   GET DATA; PUT DATA;
        */

3           DCL A(2,3) FIXED(6) INIT((6)0),
                B CHAR(10) VAR  INIT(''),
              C BIT(4) INIT ((2)'01'B),
               LAB LABEL;

4             {GET DATA COPY;
5             {PUT DATA LINE(A(2,1)/3);
6             {LAB=L1;

 9       L1: GET DATA(C,A,U);
10            {PUT SKIP(2) DATA(A);
11            {PUT DATA(LAB) SKIP(2);
12            {PUT SKIP(C) DATA(B,C);
```

```
1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.
   A(1,1) = 222,  A(2,1) = 18,  A(1 ,
    2 ) = 3333  A(2,3)=77777
  B = 'L''ENTREE' ;

A(1,1)=      222          A(1,2)=     3333          A(1,3)=          0          A(2,1)=         18          A(2,2)=          0
U= 0.000000E+00           A(2,3)=    77777          B='L''ENTREE'               C='0101'B                   LAB
                          I=            13;
1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.

1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.

A(1,1)=      11           A(1,2)=     3333          A(1,3)=          0          A(2,1)=         18          A(2,2)=         22
                          A(2,3)=       33;
LAB;

B='L''ENTREE'             C='0011'B;
1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.
```

§ . 3. 1 . 4 FORME LIST

1. Liste-données

- elle est obligatoire

- En entrée : elle peut comprendre des noms de variables, indicés ou non, de tableau ou de pseudo-variable.

- En sortie : en plus, elle peut comporter des constantes ou des expressions qui sont alors évaluées et dont la valeur est insérée dans le flot de sortie.

2. Flot de lecture

- Il se présente sur les cartes sous la forme d'une suite de constantes valides, séparées de blanc (s) et/ou par une virgule. Les chaines doivent être délimitées par une paire d'apostrophes, les constantes numériques doivent être décimales, la lettre I doit apparaitre dans une constante complexe.

- Ces constantes doivent apparaitre selon l'ordre des variables de liste-données. Un tableau est équivalent à l'ensemble de ses éléments.

- La fin des lectures dans le flot est commandée par l'épuisement de liste-données.

- Si 2 virgules consécutives apparaissent dans le flot, elles définissent une zone nulle. La variable qui lui correspond dans liste-données est sautée dans la transmission et sa valeur n'est donc pas modifiée.

3. Flot d'écriture

- La forme interne est convertie en caractère. Les valeurs numériques binaires sont converties en décimal.

- Les valeurs apparaissent sous forme de constantes mais une chaine de caractères n'est pas délimitée par une paire d'apostrophes alors qu'une chaine de bits se présente sous la forme habituelle ('01...1'

...

- Sur la ligne d'impression, ces valeurs sont placées dans les zones prédéfinies.

4. Exemples

```
        /*
        *   GET LIST; PUT LIST;
        */
3           DCL A(2,3) FIXED(6) INIT((6)0),
                B CHAR(12) INIT('L'AUTOMOBILE'),
                C BIT(4) INIT('0'B);
4               GET (A) COPY;
5               GET LIST (I,J,SUBSTR(B,I,J)) COPY;
6               GET LIST(C) COPY;
7           PUT SKIP(2) LIST(A);
8           PUT (B,C) SKIP(3);
```

```
1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.
   111, 222, , 21 , , 66666
 , 5,'NE',
'111'B

   111                222                0                21                0
 66666

L'AUTOMNE          '1110'B
1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.1234.6789.
```

§ . 3 . 1 . 5 REMARQUES POUR LES FORMES LIST et DATA

1. Cas des tableaux

Dans la transmission, un nom de tableau est traité comme l'ensemble de ses éléments, pris un par un selon la séquence de stockage en mémoire.

2. Cas des structures

Cf ch. 9

...

3. Zones d'impression

En standard, la ligne d'impression est décomposée en 5 zones de 24 Positions :

1 à 24, 25 à 48, 49 à 72, 73 à 96, et 97 à 120.

Toute valeur transmise est ajustée dans une zone. Si cette valeur est trop longue, elle s'étendra sur plusieurs zones consécutives. Si dans un PUT, une ligne est remplie, l'écriture se poursuit automatiquement sur la ligne suivante .

4. Reprise de lecture

La notion de carte de 80 colonnes étant "effacée", toute lecture est reprise là où elle avait été précédemment abandonnée. Plus précisément la reprise se fera à la 1ère position suivant une virgule séparatrice ou bien au 1er caractère non-blanc suivant des blancs séparateurs.

SECTION 3.2

INSTRUCTIONS DE CONTROLE DE SEQUENCE

§. 3 . 2 . 1 INSTRUCTION GOTO

1. Forme générale et rôle

[etiq :] GOTO destination ;

Réalise le branchement inconditionnel à l'instruction identifiée par "destination"

2. Règles

- "destination" doit avoir pour valeur une constante-étiquette connue et peut prendre 3 formes principales
- constante-étiquette
- variable-étiquette indicée ou non
- référence de fonctions renvoyant une constante-étiquette

Dans les 2 derniers cas on réalise un aiguillage puisque la valeur de la destination est variable.

3. Restrictions

- Le bloc où se trouve "destination" doit être actif, c'est-à-dire que au moment du débranchement la destination doit être connue.

...

-"Destination" ne doit pas se trouver à l'intérieur d'un groupe DO itératif.

4. Exemples

```
        /*
        *    TENTATIVE DE BRANCHEMENT
        *         DANS  UN  GROUPE  DO ITERATIF
        */
 29          DCL A(2) LABEL;

 30              DO I=1 TO 2;
 31              PUT SKIP(2) DATA(I);
 32              GOTO A(I);
 33           B: END;
 34              GOTO FIN;

 35     A(1): PUT SKIP LIST ('A1');
 36           GOTO B;
 37     A(2): PUT SKIP LIST ('A2');
 38           GOTO B;

 39          FIN: END PG32;
```

```
IEL0522I S   36, 38    'GOTO' STATEMENT SPECIFIES INVALID BRANCH INTO
                       AN ITERATIVE 'DO' GROUP.    STATEMENT IGNORED.
```

```
        /*
        *    AIGUILLAGE  PAR  GOTO  INDICE
        */
 2           ON ENDFILE(SYSIN) GOTO FIN;
 3           DCL Z BIT(1),
                 (LA,LB)(0:1) LABEL;

 4     LEC:  LA(0): LB(0):
                 GET LIST(A) COPY;
 5               Z = A = 2;
 6               GOTO LA(Z);

 7            LA(1):
                 GET LIST(B) COPY;
 8               Z = B >= A;
 9               GOTO LB(Z);

10            LB(1):
                 PUT SKIP(2) DATA(A);
11               PUT SKIP DATA (B);
12               GOTO LEC;
```

```
   1,
2,
3,

A= 2.00000E+00;
B= 3.00000E+00;
1,
3,
2,
6,

A= 2.00000E+00;
B= 6.00000E+00;
3,
2,
-1
```

§ . 3. 2 . 2 . INSTRUCTION IF THEN ; ELSE ;

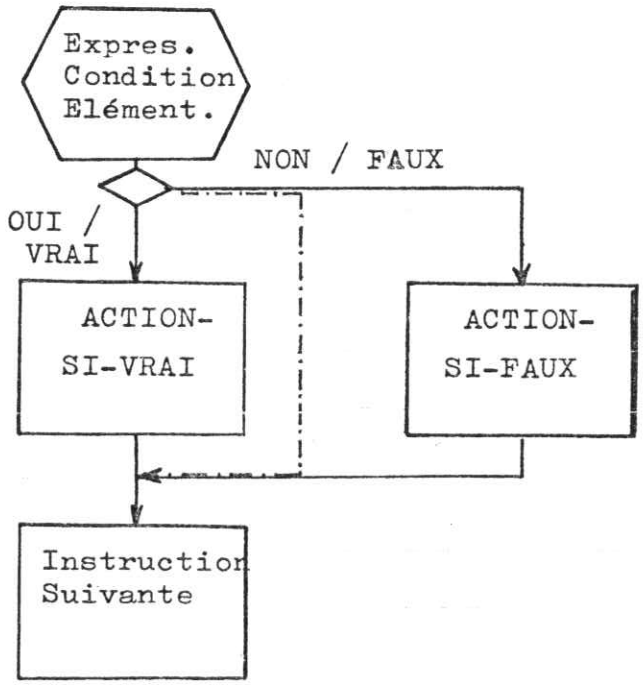
1. Forme générale et rôle

```
[préfixes :]  IF  exp-cond-élem  THEN  action-si-vrai ;
                                 [ELSE  action-si-faux ;]

              instruction suivante ;
```

Le IF permet de choisir entre 2 traitements symbolisés par "action-si-vrai" et "action-si-faux" selon que "exp-cond-élem" est vrai ou faux. C'est donc une instruction de choix binaire entre les 2 termes d'une alternative dont un seul sera réalisé.

2. Réalisation du IF

- exp-cond-élem est évaluée
- si elle est "vrai" ('1'B) alors action-si-vrai est réalisée et l'on passe à instruction-suivante
- si elle est "faux" ('0'B) alors action-si-faux est réalisée et l'on passe à instruction-suivante
- dans le dernier cas si action-si-faux n'existe pas, on passe directement à instruction-suivante



- la reprise de la séquence à instruction-suivante n'a pas lieu si les clauses du THEN et/ou ELSE sont des débranchements impératif

3. Règles

préfixe — est un ensemble de préfixes de condition ou d'étiquettes. La portée d'un préfixe-condition est limitée à l'évaluation de exp-cond-élem.

exp-cond-élem — doit finalement restituer une valeur élémentaire unique qui se traduira par '1'B ou '0'B - C'est une expression de comparaison plus ou moins compliquée mais qui ne doit pas faire intervenir d'agrégats.

action-si-vrai
action-si-faux — Ces clauses du THEN et du ELSE ne peuvent prendre que l'une des 3 formes :

- instruction unique
- groupe DO non-itératif, groupe DO itératif devant être inclus dans un groupe non-itératif
- bloc Begin

ELSE action-si-faux — n'est pas indispensable en particulier si action-si-vrai est un GOTO.
S'il n'existe pas, la séquence reprend immédiatement avec instruction-suivante dans le cas où exp-cond-élem est faux.

4. Imbrication de IF

- chacune des actions peut elle-même comporter un IF et se décomposer ainsi en une alternative dont les termes, à leur tour, peuvent se décomposer et ainsi de suite.
- on obtient donc une imbrication de IF pour lesquels il faut veiller soigneusement à la correspondance des THEN et ELSE.
- la règle de correspondance est la suivante :

La mise en correspondance se fait à partir du ELSE le plus interne vers le plus externe de sorte qu'un ELSE est toujours associé au THEN le plus interne encore solitaire.

- dans certains cas, un THEN interne ne devant pas comporter de ELSE associé, afin de maintenir l'équilibre dans les correspondances, on devra placer un ELSE nul c'est-à-dire

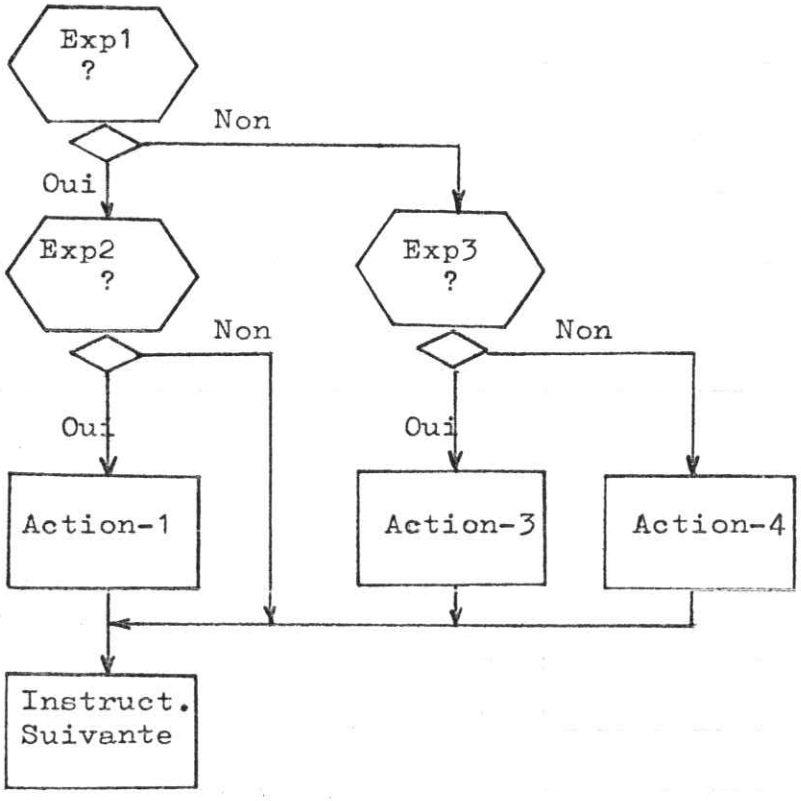
ELSE ;

n'indiquant aucune action

- par exemple,

```
IF exp1  THEN  IF  exp2   THEN  action-1 ;
                               ELSE ;
         ELSE  IF  exp3   THEN  action-3 ;
                               ELSE  action-4 ;
instruction suivante ;
```

correspondra à l'organigramme ci-dessous.



5. Exemple

```
/*
*   CAS DE FIGURES
*          D'UN  IF
*/
2       DCL  (A,B,C,R)  FIXED(3);
3       ON  ENDFILE(SYSIN) GOTO FIN;
4   LEC: GET LIST(A,B,R) COPY;
5       G=0;
6       IF R+B>A THEN G=1;
7                ELSE G=22;
8       PUT SKIP(2) DATA(G);
9       G=0;
10      IF R+B>A THEN G=13;
11                    G=17;
12      PUT SKIP DATA(G);
13      G=0;
14      IF R+B>A  THEN G=55;
15                ELSE;
16      PUT SKIP DATA (G);
17      GOTO LEC;
```

```
7,5,3

G=    1;
G=   17;
G=   55;
7,4,0

G=   22;
G=   17;
G=    0;
```

§ . 3 . 2 . 3 Groupe DO non-itératif

1. Forme

[étiq :] DO ;

séquence d'instructions

END [étiq :]

2. Rôle

On encadre une séquence d'instructions par une instruction DO non-itérative et un END associé de façon à définir un groupe DO non-itératif. Ces instructions sont alors considérées comme un tout, traité comme tel par le compilateur.

Un groupe DO non-itératif est principalement employé dans un IF pour décrire une action nécessitant plusieurs instructions.

On évite ainsi de multiplier les renvois par GOTO à d'autres régions du programme.

3. Règles

- l'exécution de la séquence d'instruction n'est pas itérée et ne se fait qu'une seule fois.
- il est possible de pénétrer à l'intérieur du groupe, par exemple par GOTO, sans initialiser le groupe par DO.

4. Exemple

```
4       LEC:   GET LIST(A,B,C);
5              PUT SKIP DATA(A,B,C);
6              DLTA = B**2 - 4*A*C ;

7       IF DLTA < 0 THEN DO;
8                  C1 = (-B +(SQRT(ABS(DLTA)))*1I) / (2*A);
9                  C2 = CONJG(C1);
10                 PUT SKIP LIST('2 RACINES COMPLEXES');
11                 PUT DATA(C1,C2);
12                 END;

13                 ELSE  IF DLTA > 0 THEN DO;
14                                X1 = (-B+SQRT(DLTA))/(2*A);
15                                X2 = (-B-SQRT(DLTA))/(2*A);
16                                PUT SKIP LIST('2 RACINES REELLES');
17                                PUT DATA(X1,X2);
18                                END;

19                                ELSE DO;
20                                X = -B/(2*A);
21                                PUT SKIP LIST('1 RACINE DOUBLE');
22                                PUT DATA(X);
23                                END;
```

§ . 3 . 2 . 4 GROUPE DO ITERATIF, 1ère FORME

1. Forme générale

```
[préfixes :]   DO var = exp1 [ TO exp2 [BY exp3] ] ;
                             [ BY exp3 [TO exp2] ]

                    Corps de boucle

               END [étiq.] ;
               Instruction suivante ;
```

préfixes : ensemble d'étiquettes et/ou de préfixes conditions dont la portée est limitée à l'évaluation des "exp_i".

var : variable de contrôle des itérations

exp1 : valeur initiale de var

exp2 : valeur finale que var ne doit pas dépasser au cours des itérations

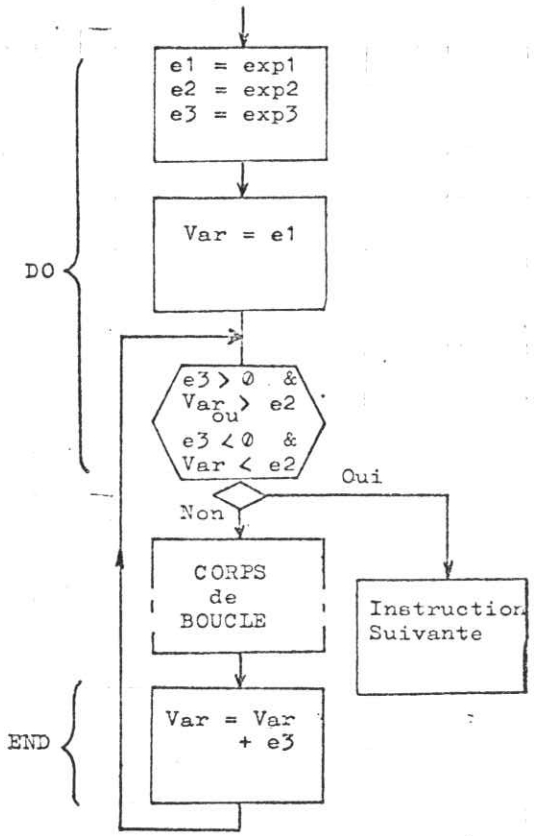
exp3 : pas de variation des valeurs de var lorsque l'on passe d'une itération à la suivante

étiq : l'une des étiquettes de DO. Elle ne doit pas être indicée.

2. Réalisation de la boucle

- exp 1, exp2 et exp3 sont successivement évalués, convertis dans les attributs de var et stockés en mémoire. Ils ne peuvent donc plus être modifiés au cours des itérations.
- le corps de la boucle sera exécuté tant que var, variant depuis exp1, au pas exp2, ne dépasse pas exp3

- En fin de chacune des itérations la valeur courante de var est modifié de exp3 avant d'être comparée à exp2
- Dès que var dépasse exp2 les itérations sont abandonnées et le déroulement reprend avec instruction suivante.



3. Règles principales

- Var et les exp_i doivent être de type élémentaire, numérique ou chaines convertibles de sorte que les calculs soient réalisables.
- Var peut être lui-même indicé ou un pseudo-variable.
- exp3 peut être négatif et dans ce cas si l'on veut que la boucle soit réalisée au moins une fois il faut que exp2 < = exp1
- Si exp3 est omis il est pris égal à +1. Un pas égal à -1 doit toujours être spécifié.
- Si exp2 seul est omis, aucune condition n'existe qui puisse mettre fin aux itérations ; elle devra donc être programmée par ailleurs dans le corps de boucle
- Si exp2 et exp3 sont omis, on se trouve en présence d'une forme dégénérée (cf § 3.2.8)
  La boucle est réalisée une fois avec var = exp1

```
      /*
      *   DO  ITERATIF  1.ERE FORME
      */
 2          PUT SKIP(2) LIST ('*1*');
 3          N=0;
 4          DO X = 2 TO '1' BY -0.2;
 5             PUT SKIP DATA(X);
 6             N = N+1;
 7          END;
 8          PUT SKIP DATA(X,N);
```

```
*1*
X=  2.00000E+00;
X=  1.79999E+00;
X=  1.59999E+00;
X=  1.39999E+00;
X=  1.19999E+00;
X=  9.99996E-01
N=          5;
```

```
 9          PUT SKIP(2) LIST('*2*');
10             N=0;
11             DO X=10 TO 5 BY -1;
12                PUT SKIP LIST(X/5);
13                N = N+1;
14             END;
15             PUT SKIP LIST(X/5); PUT DATA(N);
```

```
*2*
 2.00000E+00
 1.79999E+00
 1.59999E+00
 1.39999E+00
 1.19999E+00
 1.00000E+00
 7.99999E-01
N=          6;
```

```
17          PUT SKIP(2) LIST('*3*');
18          N=0;
19          DO X=1 TO 2 BY .2;
20             PUT SKIP DATA(X);

21             N = N+1;
22          END;
23          PUT SKIP DATA(X,N);
```

```
*3*
X=  1.00000E+00;
X=  1.19999E+00;
X=  1.39999E+00;
X=  1.59999E+00;
X=  1.79999E+00;
X=  1.99999E+00;
X=  2.19999E+00
N=          6;
```

```
24          PUT SKIP(3) LIST('*4*');
25             N=0;
26             DO X=2 TO 1 BY -0.25;
27                PUT SKIP DATA(X);
28                N=N+1;
29             END;
30             PUT SKIP DATA(X,N);
```

```
*4*
X=  2.00000E+00;
X=  1.75000E+00;
X=  1.50000E+00;
X=  1.25000E+00;
X=  1.00000E+00;
X=  7.50000E-01
N=          5;
```

§. 3. 2. 5. GROUPE DO ITERATIF 2ème FORME

1. Forme générale

[préfixes :] DO WHILE (exp4)

Corps de boucle

END [étiq :] ;

Instruction suivante ;

préfixes : ensemble d'étiquettes et/ou préfixes conditions dont la portée est limitée à l'évaluation de exp4.

exp4 : expression condition élémentaire qui prend pour valeurs '1'B ou '0'B selon qu'elle est réalisée ou non

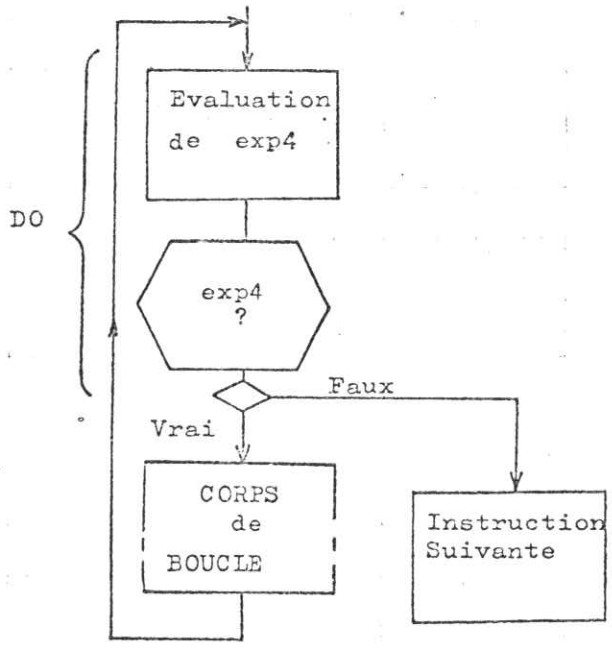
étiq : comme précédemment

2. Réalisation de la boucle

Le corps de boucle est réalisée tant que (aussi longtemps que) exp4 reste vrai.

Après chaque itération elle est à nouveau évaluée.

Dès qu'elle est trouvée fausse, les itérations sont abandonnées et la séquence est reprise à instruction-suivante.



3. Règles principales

- Les parenthèses qui entourent exp4 sont obligatoires

4. Exemple

```
      /*
      *    DO  ITERATIF  2EME FORME
      *        DO  WHILE(...);
      */

2         DCL MOT CHAR(30),L(30) CHAR(1)
                  DEF MOT;
3         ON ENDFILE(SYSIN) GOTO FIN;

4         LEC: GET LIST(MOT);
5              I=0;
6              DO WHILE(L(I)<'A');
7              I=I+1;
8              END;

9              J=30;
10             DO WHILE(L(J)<'A');
11             J=J-1;
12             END;

13             K = J-I+1 ;
14             PUT SKIP(2) LIST(MOT);
15             PUT SKIP EDIT((3)'1234567890')
                        (COL(I), A(K) );
16             GOTO LEC;
```

```
**LUNDI***
  12345
-----MARDI
     12345
+++++++++MERCREDI++++++
          12345678
```

§. 3 . 2. 6 GROUPE DO ITERATIF 3ème FORME

1. Forme générale

```
[préfixes :]  DO var = exp1 [ TO exp2  [BY exp3] ] WHILE (exp4) ;
                            [ BY exp3  [TO exp2] ]

              Corps de boucle

            END [étiq] ;
```

- C'est la forme obtenue par association de la 1ère forme et de la 2ème forme.
- Les éléments de l'instruction ont les mêmes significations que précédemment

2. Réalisation de la boucle

- Pour que le corps de boucle soit réalisé il faut :

  ET que var ne dépasse pas exp2 (1ère forme)
  que exp4 soit vrai (2ème forme)

- Dès que 1'une au moins de ces 2 conditions n'est plus réalisée, les itérations sont interrompues et la séquence se poursuit avec instruction suivante.
- La 1ère condition (1ère forme) est d'abord testée avant celle de la 2ème forme.

3. Remarque

Si exp4 s'exprime en fonction de la variable de contrôle var, il ne faut pas oublier que dans l'évaluation de exp4 la valeur de var utilisée est celle de la dernière itération, modifiée du pas exp3

4. Exemple

```
/*
*   DO  ITERATIF  3EME  FORME
*       DO  ...   WHILE(...);
*/
10      DCL MOT CHAR(30), L(30) CHAR(1)
            DEF MOT;
11      ON ENDFILE(SYSIN) GOTO  FIN;
12      LEC:  GET LIST(MOT);

13          DO I=1 BY 1 TO 30  WHILE(L(I)<'A');
14          END;

15          DO J=30 BY -1 TO 1 WHILE(L(J)<'A');
16          END;

17          K=J-I+1;

18          PUT SKIP(2) LIST(MOT);
19          PUT SKIP EDIT ((3)'1234567890')
                          (COL(I),A(K));
20          GOTO LEC;
```

```
**LUNDI***
  12345
-----MARDI
     12345
+++++++++MERCREDI+++++
          12345678
```

§ . 3 . 2 . 7 SPECIFICATIONS SUCCESSIVES D'ITERATION

1. Forme générale

[préfixes :] DO spécif$_1$, spécif$_2$,..., spécif$_n$;

Corps de boucle

END [étiq] ;

Instruction suivante :

préfixes et étiq ont les significations habituelles

spécif$_i$: une spécification d'itération de l'une des 3 formes définies précédemment séparées des autres par une virgule.

2. Réalisation

- Chacune des "spécif" est réalisée successivement sur le même corps de boucle
- On ne passe à instruction suivante que si toutes les "spécif" ont été réalisées.

3. Exemple

```
      /*
      *    DO  ITERATIF
      *        SPECIFICATIONS  SUCCESSIVES
      */

23          DCL (I,J) FIXED(3);
24          I,J=2;
25          DO I=1 TO I,                                  I=   1          J=   3;
                                                          I=   2          J=   4;
                    I TO I+J,                             I=   3          J=   5;
                                                          I=   4          J=   6;
                                                          I=   5          J=   7;
                                                          I=   6          J=   8;
                                                          I=   7          J=   9;
                          I TO I+1.5*J BY 2;              I=   8          J=  10;
                                                          I=  10          J=  11;
                                                          I=  12          J=  12;
                                                          I=  14          J=  13;
                                                          I=  16          J=  14;
26          J=J+1;                                        I=  18          J=  15;
27          PUT SKIP DATA(I,J);                           I=  20          J=  16;
28          END;
29          PUT SKIP(2) DATA(I,J);                        I=  22          J=  16;
```

§ . 3 . 2 . 8 FORME DEGENEREE

1. Forme générale

[préfixes :] DO var = v_1 , v_2, ..., v_n ;

Corps de boucle

END [étiq] ;
Instruction suivante ;

préfixes, étiq et var ont la signification habituelle

Les spécifications se réduisent à une seule valeur ou expression v_i.

2. Réalisation

- Les v_i sont initialement évalués et stockés en mémoire.
- Puis le corps de boucle est réalisé n fois, var prenant successivement les valeurs v_1, v_2, ..., v_n
- Lorsque ces n valeurs ont été utilisées, le contrôle passe à instruction suivante.

3. Remarque

Sous cette forme, var peut être de n'importe quel type de donnée-problème ou de donnée-contrôle de programme : étiquettes, point d'entrée, fichier... Les valeurs v_i doivent être convenablement choisies

4. Exemples

```
      /*
      *   DO ITERATIF
      *        FORME  DEGENEREE
      */
31          I=1;
32          DO I = I+1,
                   I+2,
                         I+3;
33          PUT SKIP DATA(I);
34          END;
35          PUT SKIP(2) DATA(I);
```

```
I= 2;
I= 4;
I= 7;
I= 7;
```

```
      /*
      *   DO  DEGENERE SUR CHAR
      */
            PROC OPTIONS(MAIN);
 2          DCL X CHAR(1),Y CHAR(4);

 3          DO X='A','B','C','D';
 4            NX=UNSPEC(X);
 5            NX=NX+1;
 6            UNSPEC(Y)=UNSPEC(NX);
 7          PUT SKIP DATA(X,Y);
 8          END;
```

```
X='A'  Y=' B  ';
X='B'  Y=' C  ';
X='C'  Y=' D  ';
X='D'  Y=' E  ';
```

```
 9          PUT SKIP(3);
10          DO X=' ','A','B','C';
11          N=UNSPEC(' ')+UNSPEC(X);
12          PUT SKIP EDIT(X,N)(A(3),F(6));
13          END;
```

```
        128
A       257
B       258
C       259
```

```
 2          DCL X CHAR(15) INIT('PSEUDO-VARIABLE');
      /*
      *   DO  AVEC  PSEUDO-VARIABLE
      */
 3          PUT PAGE (X);
 4          DO I = 1 TO 3;
 5          PUT SKIP(2) EDIT(I)(F(3));
 6               DO SUBSTR(X,I,4)= 'AUX','EAUX';
 7               PUT SKIP DATA(X);
 8               END;
 9          END;
```

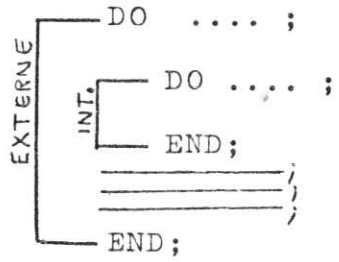
```
PSEUDO-VARIABLE

   1
X='AUX DO-VARIABLE';
X='EAUXDO-VARIABLE';

   2
X='EAUX O-VARIABLE';
X='EEAUXO-VARIABLE';

   3
X='EEAUX -VARIABLE';
X='EEEAUX-VARIABLE';
```

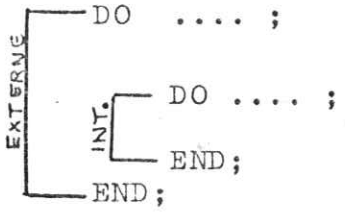
---

§ . 3. 2 . 9 REMARQUES GENERALES

1. Imbrication de boucle

- Un corps de boucle peut comporter lui-même une autre boucle controlée par un DO itératif.
- Il faut que la boucle interne soit totalement imbriquée à la boucle externe, DO et END compris
- On peut donc avoir les 2 schémas ci-contre





- Si les 2 boucles ont une sortie "tangente" elles pourront être fermées avec un seul END réalisant une fermeture multiple (cf § 4.1.)
- La fréquence d'exécution des boucles croît avec la profondeur d'imbrication.
- Il ne faut confondre les spécifications successives qui réalisent une sommation d'itérations alors que l'imbrication réalise un produit d'itérations.

2. Autres remarques

- Si la condition qui met fin aux itérations est satisfaite d'emblée, le corps de boucle n'est pas exécuté du tout.
- On ne peut pénétrer dans un groupe DO itératif que par l'instruction DO. On ne peut pas quitter le groupe pour y revenir par GOTO.
- Si var est de type numérique (ou convertible) sa partie entière obtenue par troncature peut servir d'indice.

§. . 3 . 2 . 10 SPECIFICATION REPETITIVE DANS LISTE-DONNEES

1. Forme

(..., (liste-données-partielle DO spécification),...)

- liste-données-partielle : est une partie de la liste-donnée de l'instruction de transmission dont l'utilisation sera itérée sous le contrôle du DO
- spécification : est l'une des 5 formes de DO étudiée précédemment.
- l'itération dans la transmission porte sur l'ensemble de la liste partielle et non pas sur chacun des éléments de la liste pris individuellement.

2. Règles

- Le END normalement associé au DO ne doit pas apparaitre.
- La portée de spécification d'itération est limitée par un niveau de parenthèses.
- Des itérations de transmission peuvent être imbriquées. Chacune d'elles doit eîre limitée par un niveau de parenthèses.

3. Exemple

```
        /*
        *    SPECIFICATION REPETITIVE
        *        DANS  GET ET PUT
        */
        PROC OPTIONS(MAIN);

2       DCL X(5,5);

3       GET LIST (M,N,(( X(I,J)  DO I=1 TO M)
                                 DO J=1 TO N));

4       PUT PAGE LIST('*1*');
5       PUT SKIP EDIT(((X(I,J) DO I=1 TO M)
                               DO J=1 TO N))
                        ( (N) F(5),SKIP);

6       PUT SKIP(3) LIST('*2*');
7            DO I=1 TO M;
8            PUT SKIP EDIT((X(I,J) DO J=1 TO N))
                           (F(5));
9            END;

10      PUT SKIP(3) LIST('*3*');
11      PUT SKIP;
12           DO I=1 TO M;
               .
13           DO J=1 TO N;
14             PUT EDIT (X(I,J))(F(5));
15           END;
16           PUT SKIP;
17           END;
```

```
*1*
    11   12   13
    21   22   23

*2*
    11   13   22
    12   21   23

*3*
    11   13   22
    12   21   23
```

§ . 3 . 2 . 11 INSTRUCTION NULLE

1. Forme

[étiq :] ;

2. Rôle

Ne se traduit par aucun code résultant

Cette instruction est principalement utilisé dans 2 cas :

pour un ELSE nul

pour une unité-ON nulle (cf § 5.2.7)

3. Remarque

L'étiquette d'une instruction nulle n'a pas la même valeur que l'étiquette de l'instruction suivante.

SECTION 3.3.

PRINCIPALES FONCTIONS ET PSEUDO-VARIABLES INCORPOREES A PLIOPT

§. 3 . 3 . 1 GENERALITES

1. Référence de fonctions

- Les fonctions incorporées au compilateur étendent largement les possibilités des opérateurs et permettent de résoudre facilement de nombreux traitements simples sur des données de type numérique ou chaine.
- Pour cela on fait appel à la fonction au moyen d'une "référence de fonction" dans laquelle on précise les valeurs sur lesquelles elle devra s'appliquer et que l'on appelle arguments

Nom-fonction [(liste-arguments)]

- Une telle référence de fonction apparait en général dans une expression. Auquel cas, elle est immédiatement résolue avant tout autre opérateur et le résultat qu'elle retourne se substitue à la référence de fonction.

2. Liste-arguments

La liste, placée entre parenthèses à la suite du nom de fonction, est formée d'un ensemble d'arguments séparés par une virgule. Chaque argument peut être une constante, une variable, ou une expression pouvant elle même contenir une référence à d'autres fonctions.

- Dans la plupart des cas, les arguments pourront être des tableaux ou même des expressions-tableaux. Il faudra alors que tous les tableaux aient des attributs de dimensionnement identiques.
- Certaines fonctions n'ont pas d'argument . Comme les mots-clés du PL/I ne sont pas réservés, il est nécessaire de préciser si le nom représente ou non la fonction. S'il s'agit de la fonction on pourra :
  soit utiliser une liste-argument vide : ( )
  soit déclarer le nom avec l'attribut BUILTIN. (cf § 4.2.)

3. Résultat d'une fonction

Les attributs du résultat restitué par une fonction seront en général ceux des arguments ; dans certains cas la précision pourra être modifiée. Ainsi la fonction SIN pourra s'appliquer indifféremment à un argument réel ou complexe. Le calcul et le résultat seront adaptés au mode de l'argument.

Si la fonction restitue une valeur de tableau ces valeurs seront organisés selon le dimensionnement commun aux arguments.

4. Pseudo-variable

Certaines fonctions celles que SUBSTR, UNSPEC... peuvent s'employer comme pseudo-variable c'est-à-dire en position de cible dans une instruction d'affectation, sous la forme.

Nom-fonction (liste-arguments) = source ;

Dans ce cas l'appel à la fonction, utilisée comme pseudo-variable, a pour effet d'appliquer la valeur de la source à l'un des arguments.

§ . 3 . 3 . 2 Fonctions mathématiques

1. Généralités

- Les arguments doivent être (convertibles) en FLOAT. Ils doivent rester compris entre certaines limites liées soit à la définition de la fonction, soit à des considérations de hardware.
- Dans certains cas l'argument peut être réel ou complexe ; la réalisation de la fonction est alors adaptée au mode de l'argument.

- Le résultat est exprimé en FLOAT dans la plus grande des précisions des arguments.

2. Fonctions circulaires

COS, SIN, TAN pour un argument réel en radians ou complexe.

COSD, SIND, TAND argument réel en degrés.

ACOS, ASIN restitue une valeur réelle en radians.

ATAN (x) renvoie Arctg x entre $-\frac{\pi}{2}$ et $+\frac{\pi}{2}$

x peut être complexe.

ATAN (x, y) renvoie arctg x/y, x et y étant réels non tous nuls.

ATAND (x) et ATAND (x, y) restituent des résultats en degrés, pour des arguments réels.

3. Fonctions hyperboliques

EXP
COSH, SINH, TANH
ATANH

} argument réels ou complexes.

4. Fonctions logarithmes, racine carrée

LOG : $\log_e$ réel ou complexe

LOG_2 : $\log_2$
LOG_{10} : $\log_{10}$
} réels

SQRT : racine carrée : réel ou complexe.
renvoie la détermination principale.

5. Exemples

```
      /*
      *   FONCTIONS   ATAND
      */

2         ON ENDFILE(SYSIN) GOTO SUITE;
3         LEC:  GET LIST(X,Y);
4               PUT SKIP EDIT(X,Y,ATAND(Y/X),
                                  ATAND(Y,X))
                                 (4 F(9,3));
5            PUT SKIP;
6            GOTO LEC;
```

```
 0.500    0.866   59.999   59.999
 0.866   -0.500  -30.001  -30.001
-0.500    0.866  -59.999  120.001
-0.866   -0.500   30.001 -149.999
```

```
      /*
      *   FONCTIONS  HYPERBOLIQUES
      */

 8            DO Z = 1 TO 2 BY .3;
 9                ASH=LOG(Z+SQRT(Z*Z+1));
10                ACH=LOG(Z+SQRT(Z*Z-1));
11            PUT SKIP(2) EDIT(Z,ASH,SINH(ASH))
                                 (R(FOR));
12            PUT SKIP EDIT(ACH,COSH(ACH))
                              (R(FAR),  2 R(FUR));
13            END;
14     FAR:  FORMAT(COL(5));
15     FOR:  FORMAT(F(4,1),2 F(8,5));
16     FUR:  FORMAT ( F(8,5));
```

```
1.0 0.88137 1.00000
    0.00000 1.00000
1.3 1.07845 1.30000
    0.75643 1.30000
1.6 1.24898 1.60000
    1.04697 1.60000
1.9 1.39800 1.90000
    1.25719 1.90000
```

§ . 3 . 3 . 3 FONCTIONS ARITHMETIQUES

1. Généralités

Les arguments doivent être numériques , et en général réels.

Le résultat aura les mêmes attributs que les arguments, sauf peut être la précision.

2. Valeur absolue

ABS l'argument peut être complexe et alors la fonction renvoie son module.

3. Troncature - arrondi - partie entière

TRUNC : partie entière par troncature de la partie fractionnaire

ROUND (x, n) : x doit être en FIXED, n constante entière, positive ou négative.
Le résultat est la valeur de x, arrondie sur sa n^{ième} position à droite (n > 0) ou à gauche (n < 0) du point fractionnaire.

FLOOR (x) restitue le plus grand (algébriquement) des entiers minorant x

CEIL (x) restitue le plus petit (algébriquement) des entiers majorant x

EXEMPLE

```
      /*
      *    FONCTIONS
      *         TRUNC,CEIL,FLOOR
      *         &    ROUND
      */

 2          DCL X FIXED (5,3);
 3          ON ENDFILE(SYSIN) GOTO FIN;

 4          LEC: GET LIST(X);
 5               PUT PAGE EDIT(X,-X)(R(FOR));

 6               PUT SKIP(2) EDIT('TRUNC',
                       TRUNC(X),TRUNC(-X))(A,R(FOR));

 7               PUT SKIP EDIT('CEIL',
                       CEIL(X),CEIL(-X))(A,R(FOR));

 8               PUT SKIP EDIT('FLOOR',
                       FLOOR(X),FLOOR(-X))(A,R(FOR));

 9               PUT SKIP EDIT('ROUND')(A);
10               PUT SKIP(0) EDIT(+2, ROUND(X,+2),
                                     ROUND(-X,+2),
                                  0, ROUND(X,0),
                                     ROUND(-X,0),
                                 -1, ROUND(X,-1),
                                     ROUND(-X,-1))
                    (COL(7),F(2), 2 F(8,3),SKIP);
11               GOTO LEC;
12     FOR : FORMAT(COL(9), 2 F(8,3));
```

```
                    12.437 -12.437
TRUNC               12.000 -12.000
CEIL                13.000 -12.000
FLOOR               12.000 -13.000
ROUND    2          12.440 -12.440
         0          12.000 -12.000
        -1          10.000 -10.000
```

```
                    42.678 -42.678
TRUNC               42.000 -42.000
CEIL                43.000 -42.000
FLOOR               42.000 -43.000
ROUND    2          42.680 -42.680
         0          43.000 -43.000
        -1          40.000 -40.000
```

4. Minimum, maximum

MIN
MAX $(\ldots, x_i, \ldots)$ - restituent respectivement le plus petit ou le plus grand au sens algébrique, des x_i qui doivent être réels.

- i peut varier de 2 à 64.

5. Reste de division

MOD (x, y)

Renvoie le reste de la division entière de x par y . En FIXED, ce résultat peut être tronqué à gauche

6. Exemple

```
/*
*   FONCTION  MOD
*/
    DCL (I,J,(A,B) FIXED(5,2), X,Y )(3);
    GET LIST (I,J);
    PUT SKIP(2)EDIT(' I ',I,' J ',J,'MOD',
                    MOD(I,J))(R(FOR));
    GET LIST (A,B);
    PUT SKIP(2) EDIT(' A ',A,' B ',B,'MOD'
                    MOD(A,B))(R(FOR));
    GET LIST (X,Y);
    PUT SKIP(2)EDIT(' X ',X,' Y ',Y,'MOD',
                    MOD(X,Y))(R(FOR));
FOR : FORMAT(A(3), 3 F(8,3),SKIP);
```

I	3.000	3.000	3.000
J	2.000	3.000	5.000
MOD	1.000	0.000	3.000
A	6.000	6.000	6.000
B	2.000	4.000	9.000
MOD	0.000	2.000	6.000
X	4.500	4.500	4.500
Y	2.250	4.000	13.500
MOD	0.000	0.500	4.500

7. Signe d'une quantité

SIGN (x) renvoie -1, 0 ou + 1 selon que x est négatif, nul ou positif.

8. Fonctions et pseudo-variables complexes

COMPLEX (A, B) ou CPLX (A, B)

A et B étant réels, restitue le nombre complexe A +iB

CONJG (Z)

Z étant complexe, restitue son conjugué $\bar{Z}$

- Fonctions REAL (Z) et IMAG (Z)

  Z étant complexe, restituent respectivement la partie réelle et la partie imaginaire de Z

- Pseudo-variables REAL (Z) et IMAG (Z)

  Ces pseudo-variables, apparaissant en position de cible, affectent respectivement à la partie réelle et à la partie imaginaire de Z la valeur de la source.

  pour avoir Z = A + iB on peut écrire

  REAL (Z) = A ;

  IMAG (Z) = B ;

  ou bien Z = CPLX (A, B) ;

```
      /*
      *    NOMBRES COMPLEXES
      *         FORMULE DE  MOIVRE
      */
 2          DCL (Z1, Z2, Z3) CPLX;

 3              X=30;
 4          DO K=1 TO 6;
 5              Z1=CPLX(COSD(X),SIND(X));
 6              Z2=Z1**K;
 7              REAL(Z3) = COSD(K*X);
 8              IMAG(Z3)=SIND(K*X);
 9              PUT SKIP DATA(Z2);
10              PUT EDIT('Z3=',Z3)
                        (COL(35),A,C(F(8,5)) );
11          END;
```

```
Z2= 8.66025E-01+4.99999E-01I;     Z3= 0.86603 | 0.50000
Z2= 5.00000E-01+8.66025E-01I;     Z3= 0.50000 | 0.86603
Z2= 1.19209E-07+9.99999E-01I;     Z3= 0.00000 | 1.00000
Z2=-4.99999E-01+8.66025E-01I;     Z3=-0.50000 | 0.86603
Z2=-8.66024E-01+5.00000E-01I;     Z3=-0.86603 | 0.50000
Z2=-9.99999E-01+2.38418E-07I;     Z3=-1.00000 | 0.00000
```

§ . 3 . 3 . 4 FONCTIONS DE CHAINES

1. Généralités

Elles s'appliquent en général sur un argument de type chaine ou convertible en chaine, d'autres arguments pouvant servir à préciser cette application.
Si l'argument principal est de type numérique BINARY, il sera converti en chaine de bits. Il sera converti en chaine de caractères s'il est DECIMAL.

2. Fonction SUBSTR (X, I [J])

- X est la chaine de base ; ce peut être une expression ou un tableau

  I position de départ dans X, comptée à partir de la gauche.

  J longueur de la sous-chaine à recopier, comptée à partir de la position I

I et J peuvent être des constantes ou des expressions dont seule la partie entière est utilisée.

- La fonction restitue la sous-chaine prise dans X à partir de la Ième position sur une longueur J. C'est-à-dire des positions I à I + J - 1 .
- I et J doivent être tels que l'extraction soit réalisable.
  Si J = 0, on récupère la chaine nulle.
  Si J est omis, on récupère le reste de la chaine à partir de I.
- X reste invariant.

EXEMPLE

```
     /*
     *   FONCTION    S U B S T R
     */
2         DCL  X(3) CHAR(10) VAR INIT('JANVIER','JUIN',
                                  'JUILLET'),
               J(3) INIT(1,2,4);
3              X = SUBSTR (X, J,LENGTH(X)/2);
4              PUT PAGE EDIT  ( X )( A,SKIP);
```

```
JAN
UI
LLE
```

3. Pseudo-variable SUBSTR (X, I, [,J])

X, I, J ont les mêmes significations que ci-dessus. Mais comme SUBSTR apparait en position de cible, la source est placée dans X à partir de la position I et sur une longueur J. Il s'agit d'une véritable affectation, avec cadrage à gauche (position I) et ajustement à droite (position I + J -1). Les autres positions de x demeurent inchangées.

EXEMPLE

```
      /*
      *   PSEUDO-VARIABLE  S U B S T R
      */

6         DCL   Y(3) CHAR(10) VAR INIT ((3)(6)'0'),
                Z(3) CHAR(10)     INIT ((3)(10)'B');

7            PUT SKIP EDIT(Y(1),Z(1))(X(14),2 A(11));

8             DO I=1 TO 3;
9             SUBSTR (Y(I),2,5) = REPEAT('L',I**2);
10            SUBSTR (Z(I),2,5) = REPEAT('L',I**2);
11            PUT SKIP EDIT(I,REPEAT('L',I**2),Y(I),Z(I))
                      (F(3), 3 A(11));

12            END;
```

```
                    00000                BBBBBBBBBB
1LL                 0LL                  BLL      BBB
2LLLLL              0LLLLL               BLLLLLBBBB
3LLLLLLLLLL         0LLLLL               BLLLLLBBBB
```

4. Fonction INDEX (x, y)

- X est la chaine de base, Y une autre chaine dont on recherche l'existence et la position dans X.

  Le résultat est une valeur BIN FIXED (15) qui indique :

  - La 1ère position de la 1ère occurence de X dans Y
  - Ou bien, 0, si Y ne se retrouve pas dans X

```
      /*
      *   I N D E X
      */

2         DCL N(3) INIT ((3)0),
              CB(3) CHAR(2) INIT('HE','ER','RH'),
              A CHAR(10) INIT('RECHERCHER');

3         PUT PAGE LIST(A);
4         PUT SKIP EDIT(CB)(3 A(3));

5         N = INDEX(A, CB);
6         PUT SKIP(2) EDIT(N)(3 F(3));

7         PUT SKIP(2) EDIT(SUBSTR(A,N+2))(A,X(1));
8         N = INDEX(SUBSTR(A,N+2), CB);
9         PUT SKIP(2) EDIT(N)(F(3),F(6),F(6));
```

```
    ↓↓
RECHERCHER
HE ER RH

 4  5  0

RCHER CHER ECHERCHER
  ↑    ↑
  3    3      0
```

5. Fonction LENGTH

L'argument est de type chaine, généralement de longueur variable. La fonction renvoie une valeur BIN FIXED (15) qui est la longueur instantanée de la chaîne.

6. Fonction REPEAT (X, K)

X est la chaine de base

K est le facteur de répétition de chainage ; ce peut être une expression

Le résultat est la chaine obtenue en chainant X, K fois à lui-même. On obtient donc (K + 1) fois X .

Si K est négatif ou nul, on retrouve X seul

EXEMPLE

```
        /*
        *   R E P E A T
        */
11          DCL A(9) CHAR(9),LJ(9);
12          DO I=1 TO 9;
13          LJ(I) = 9*COSD(10*I-10);
14          END;
15              A = REPEAT('*',LJ-1);
16              PUT SKIP(5) EDIT(A) (COL(1),A);
```

```
*********
********
*******
******
*****
****
***
*
```

7. Fonction UNSPEC (X)

- L'argument peut être de n'importe quel type : donnée-problème ou donnée-contrôle de programme. Le résultat sera la copie de la configuration binaire interne représentant la valeur de X, c'est-à-dire une chaine de bits dont la longueur qui varie selon X est au moins égal à 16.

  Si x est une chaine variable, le préfixe longueur de 2 octets est inclus dans la traduction

- La fonction UNSPEC peut-être intéressante au cours de la phase de mise au point du programme afin de connaître très précisément la valeur d'une variable.

```
        /*
        *   FONCTION   U N S P E C
        */
2           DCL     A FIXED(5,2)   INIT (12.53),
                    B BIN FIXED    INIT (19),
                    C FLOAT(6)     INIT (12.53),

                    D CHAR(3)      INIT('E19'),
                    E LABEL        INIT (E19);
3           DCL      F ENTRY VARIABLE INIT(PCD);
4       E19:  PUT PAGE LIST(UNSPEC(A),UNSPEC(-A));
5             PUT SKIP(2) LIST(UNSPEC(B),UNSPEC(-B));
6             PUT SKIP(2) LIST(UNSPEC(C),UNSPEC(-C));
7             PUT SKIP(4) LIST(UNSPEC(D));
8             PUT SKIP(2) LIST(UNSPEC(E));
9             PUT SKIP(4) LIST(UNSPEC(F));
```

```
A '0000|0001|0010|0101|0011|1100'B                    -A '0000|0001|0010|0101|0011|1101'B
B '0000|0000|0001|0011'B    -B '1111|1111|1110|1100'B
C '0100|0001|1100|1000|0111|1010|1110|0001'B          -C '1100|0001|1100|1000|0111|1010|1110|0001'B

D '1100|0101|1111|0001|1111|1001'B
E '0000|0000|0000|0110|0010|0111|1111|1000|0000|0000|0000|0110|0111|0001|0111|0000'B

F '0000|0000|0000|0110|0010|0000|1000|1100|0000|0000|0000|0000|0000|0000|0000|0000'B
```

8. Pseudo-variable UNSPEC (x)

UNSPEC étant en position de cible, la source qui doit être (convertible en) une chaine de bits, est affectée dans les positions de mémoire de X.

Le cadrage se fait à gauche et l'ajustement à droite. Si X est variable, le préfixe longueur est inclus dans l'affectation.

```
        /*
        *   PSEUDO-VARIABLE   U N S P E C
        */

10          DCL  X  CHAR(6) VAR INIT('UNSPEC');
11        {PUT PAGE EDIT  (X,LENGTH(X), UNSPEC(X))(R(FOR));

12        {     X = 'VAR';
13        {PUT SKIP(2) EDIT(X,LENGTH(X), UNSPEC(X))(R(FOR));

14        {UNSPEC(X) = (14)'0'B||'11'B||'110110011100000111100101'B;
15        {PUT SKIP(4) EDIT(X,LENGTH(X), UNSPEC(X))(R(FOR));

16        {UNSPEC(X) = (13)'0'B||'101'B||(5)'11000C01'B;
17        {PUT SKIP(2) EDIT(X,LENGTH(X),UNSPEC(X))(R(FOR));

18      FOR: FORMAT( A(6),COL(9),F(2),COL(14), B);
```

UNSPEC	6	0000.00C0	C0000110	11100100	11010101	11100010	11010111	11000101	11000011
VAR	3	00C00000	00000011	11100101	11000001	11011001	11010111	11000101	11000011
RAV	3	00000C00	C0000011	11011001	11000001	11100101	00C00000	00000000	00000000
AAAAA	5	0C000000	00000101	11000001	11000001	11000001	11000001	11000001	00000000

9. Fonction BOOL (x, y, B)

- Les arguments sont (convertibles en) des chaines de bits
- x et y sont les chaines opérandes sur lesquelles s'appliquera la fonction. Ils sont de longueurs quelconques ; la chaine la plus courte est complètée de '0'B à droite.
- B est formée de 4 bits et sert à définir de résultat de l'applicatio de la fonction entre X et Y

  B a donc la forme $'b_1\ b_2\ b_3\ b_4'$ B
- Les chaines X et Y sont explorées en parallèle, bit par bit. Selon la configuration des couples de bits (x_i, y_i), le bit de rang i de la chaine résultat sera l'un des 4 bits de B.
- La table de correspondance est :

x_i \ y_i	'0'B	'1'B
'0'B	b_1	b_2
'1'B	b_3	b_4

```
      /*
      *   FONCTION   B O O L
      */

2         DCL   X BIT(4) INIT ('0011'B),
                Y BIT(4) INIT ('0101'B);
3         DCL   Z BIT(4), U BIT(7), I   FIXED(2);

4         PUT PAGE LIST(' I      X     Y      Z    BOOL ');
5         PUT SKIP EDIT(X,Y,'(X,Y,Z)')
                  (COL(6),B(4),X(1),B(4),X(9),A);

6         DO I = 0 TO 15;
7         U = I ;
8         Z = SUBSTR(U,4,4);
9         PUT SKIP EDIT(I,Z, BOOL(X,Y,Z))
                  (F(3),COL(17),B(4),X(2),B(4));
10        END;
```

I	X 0011	Y 0101	Z	BOOL (X,Y,Z)
0			0000	0000
1			0001	0001
2			0010	0010
3			0011	0011
4			0100	0100
5			0101	0101
6			0110	0110
7			0111	0111
8			1000	1000
9			1001	1001
10			1010	1010
11			1011	1011
12			1100	1100
13			1101	1101
14			1110	1110
15			1111	1111

§ . 3 . 3 . 5 FONCTIONS DE TABLEAU

1. Généralités

L'argument est un tableau ou bien une expression tableau mais le résultat est une valeur unique, dépendant des éléments du tableau. Bien que le résultat soit élémentaire, de telles fonctions ne doivent pas apparaitre dans l'expression d'un IF. Les tableaux ne doivent pas être redéfinis par ISUB.

2. ALL (x) et ANY (x)

- L'argument x doit être un tableau de chaines de bits, ou bien convertible en chaines de bits
- Le résultat sera une chaine de bits
  - dont la longueur est la plus grande des longueurs des éléments du tableau
  - dont le bit de rang i sera :

    pour ALL : '1'B si tous les bits de rang i des éléments sont '1'B et 'Ø'B dans les autres cas.

    pour ANY : '1'B si l'un au moins des bits de rang i des éléments est '1'B et 'Ø'B si tous sont 'Ø'B

3. SUM (x) et PROD (x)

- L'argument x doit être de type numérique

  Si les éléments ne sont pas des entiers ils sont convertis en flottant ; si ce sont des chaines, ils sont convertis en entiers.

- Le résultat est la somme ou le produit de tous les éléments de X. Si les éléments sont en FIXED, comme un dépassement de précision du résultat n'est pas exclu, le compilateur imprime un message de niveau W.

```
        /*
        *    FONCTIONS DE TABLEAUX
        *         SUM  &  PROD
        */
12          DCL A(4) FIXED, B(4) BIT(1), BB BIT(1);
13          ON ENDFILE(SYSIN) GOTO FIN;
14      LEC:PUT SKIP(2)LIST((25)'-');
15          GET LIST(A)COPY;
16              C = PROD(A);
17              IF C=0 THEN PUT SKIP LIST
                           ('AU MOINS UN ELEMENT NUL');
18              C = SUM(ABS(A));
19              IF C=0 THEN PUT SKIP LIST
                           ('TOUS LES ELEMENTS NULS');


        /*
        *    FONCTIONS DE TABLEAUX
        *         ALL  &  ANY
        */
20          BB = ALL(B);
21              IF BB   THEN PUT SKIP LIST
                           ('TOUS LES ELEMENTS NULS');
22          B = A = 0;
23          BB = ANY(B);
24              IF BB   THEN PUT SKIP LIST
                           ('AU MOINS UN ELEMENT NUL');
25          PUT SKIP(2); GOTO LEC;

27          FIN:  PUT ('F I N') SKIP(3);
```

```
   1,2,0,1
AU MOINS UN ELEMENT NUL
AU MOINS UN ELEMENT NUL

0,0,1,0
AU MOINS UN ELEMENT NUL
AU MOINS UN ELEMENT NUL

   1,2,3,4

0,0,0,0
AU MOINS UN ELEMENT NUL
TOUS LES ELEMENTS NULS
AU MOINS UN ELEMENT NUL

F I N
```

```
EL08711 I   16      RESULT OF BUILTIN FUNCTION 'PROD' WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS
EL08711 I   18      RESULT OF BUILTIN FUNCTION 'SUM' WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS.

ND OF COMPILER DIAGNOSTIC MESSAGES
```

§ . 3 . 3 . 6 AUTRES FONCTIONS

1. Généralités

Elles sont sans argument et doivent donc soit être déclarées BUILTIN soit utilisées avec une liste-argument nulle.

Elles restituent des chaines de caractères.

2. DATE [( )]

Restitue une chaine de 6 caractères précisant la date du jour sous la forme :

AAMMJJ

AA = millésime MM = mois JJ = jour

La fonction DATE est coûteuse ; ne l'utiliser qu'une seule fois par programme.

3. TIME [( )]

Restitue une chaine de 9 caractères précisant l'heure de l'horloge interne initialisée à l'IPL. Cette chaine à la forme

HHMMSSTTT

HH = heure MM = minute SS = seconde TTT = milliseconde par 300^{ième} de seconde.

SECTION 3.4

ENTREES/SORTIES

FORME EDIT

§. 3 . 4 . 1 INSTRUCTIONS DE LA FORME EDIT

1. Forme des instructions

[préfixes :] GET EDIT (liste-données) (liste-formats) [COPY | SKIP (exp)] ;

[préfixes :] PUT EDIT (liste-données) (liste-formats) [PAGE | LINE (exp) | SKIP (exp)] ;

Les options COPY, SKIP, PAGE ou LINE sont celles étudiées au §. 3.1.2.

2. Principe de la transmission : liste-formats

Le flot de transmission du mode STREAM est un flot continu de valeurs exprimées sous forme caractère. La forme externe étant différente de la forme interne, des conversions sont nécessaires.

En forme EDIT, l'interprétation et l'organisation du flot de transmission est programmée par l'intermédiaire de liste-formats qui décrit l'information telle qu'elle se présente (lecture) ou se présentera (écriture) sur le support externe. Liste-formats est obligatoire.

3. Liste-données

Elle est constituée exactement comme pour la forme LIST (cf 3.1.4.1). Seules des données de type numérique ou chaines peuvent apparaitre

§ . 3 . 4 . 2 FLOT DE TRANSMISSION

1. Flot d'entrée

- Il se présente sous la forme d'une suite continue de caractères, sans séparateur reconnu (,oub) ni délimiteur de chaine (' et B). Le caractère I d'une constante complexe ne doit pas apparaitre mais seulement la partie réelle puis la partie imaginaire. Le découpage et l'interprétation du flot de lecture sont programmés par l'intermédiaire de liste-formats.

2. Flot de sortie

Il se présente sous la forme d'une suite continue de caractères. Son interprétation et sa présentation sur le support externe sont programmés par liste-formats. Aucun séparateur n'est introduit automatiquement dans le flot. Aucun délimiteur n'apparait. Les données binaires sont converties en décimal.

3. Remarques

- Aux conversions autorisées près, les descriptions de la liste-formats doivent correspondre aux variables associées.

§ . 3 . 4 . 3 LISTE-FORMATS

1. Forme et composition

Elle est placée entre parenthèses et peut se composer

- de descriptions de données externes : F, E, C, A ou B
- de contrôleurs du flot de transmission
  X, COL, SKIP, LINE, PAGE
- d'indicateur de format à distance : R

Ces éléments de formats, (sauf R) peuvent être paramétrés et ces paramètres peuvent être des expressions qui seront évaluées au moment de l'exécution et dont seule la partie entière sera utilisée. Les descriptions de données, l'indicateur R ou une sous-liste de formats délimitée par un niveau interne de parenthèses peuvent être précédés par un facteur d'itération. Ce facteur sera, soit une constante décimale entière suivie d'un blanc, soit une expression entre parenthèses dont la partie entière de la valeur sera utilisée.

Un facteur d'itération nul a pour effet d'éliminer l'élément de format ou la sous-liste de formats qu'il préfixe.

2. Place de liste-formats

- Soit dans l'instruction de transmission, à la suite de liste-données et dans ce cas elle n'est utilisable que par l'instruction,
- Soit dans une instruction FORMAT, avec étiquette

étiquette : FORMAT (liste-formats).

Dans ce cas, le GET ou le PUT qui l'utilise spécifie l'étiquette du FORMAT que l'on associe à la transmission au moyen de l'indicateur de format à distance R.

$$\left\{\begin{matrix}\text{GET}\\\text{PUT}\end{matrix}\right\} \text{(liste-données) (..., R(étiquette),...) ;}$$

Un FORMAT peut être utilisé aussi bien par un GET que par un PUT.

3. Réalisation de la transmission

- Liste-données et liste-formats sont explorés séquentiellement, en parallèle, de gauche à droite.
- A un élément de donnée est associé la 1ère description rencontrée.
- Si au cours de cette exploration séquentielle un contrôleur est rencontré, il est immédiatement exécuté et l'exploration se poursuit à la recherche de la description suivante.

- L'arrêt de la transmission est commandé par l'épuisement de liste-données. Le reste de liste-formats est immédiatement abandonné.

- Si liste-formats est trop courte, elle est réutilisée depuis le début.

4. Autres règles

- Un tableau est équivalent à l'ensemble de ses éléments. Ils sont transmis selon leur disposition en mémoire. A chaque élément doit correspondre une description ; une même description peut être réutilisée indéfiniment.

- En sortie, les valeurs numériques sont ajustées sur la droite de la zone réservée ; les valeurs de chaines sont ajustées à gauche.

§ . 3 . 4 . 4 DESCRIPTIONS DE DONNEES NUMERIQUES

1. Généralités

Ils décrivent la forme caractère d'une valeur numérique décimale externe. Tous les paramètres peuvent être des expressions de valeur non négative.

Chaque description peut être précédée d'un facteur d'itération qui peut être également une expression.

En sortie, les valeurs numériques sont arrondies sur le chiffre d'extrême droite.

2. Virgule fixe F (w [, d] [, p])

w : étendue de la zone réservée dans le flot,

d : nombre de positions réservées aux chiffres fractionnaires. Ces positions sont comptées à partir de la droite.

p : facteur d'échelle ; il peut être négatif
Il a pour effet de multiplier la valeur par 10^p au moment de la transmission.

En entrée : - La constante doit se trouver dans la zone. Si elle comporte un point, celui-ci est prioritaire sur d.
- Si le nombre est entier, d qui est nul peut être omis.

En sortie - La valeur interne est convertie en décimal virgule fixe et cadrée à droite de la zone réservée ; les zéros non-significatifs sont remplacés par des blancs.

- Si w est seul, seule la partie entière est imprimée.
- Si d apparait, la partie fractionnaire est ajustée.

3. Virgule flottante E(w, d [, s])

w : étendue de la zone réservée dans le flot ; elle doit tenir compte de tous les symboles nécessaires à l'expression de la valeur numérique.

d : nombre des chiffres fractionnaires dans la mantisse

s : nombre total des chiffres de la mantisse

En entrée : La constante doit se trouver dans la zone sous forme virgule flottante. Si le E ou l'exposant sont omis, l'exposant est pris égal à 0. Le point de la mantisse a priorité sur d.

En sortie : La valeur interne est convertie en décimal virgule flottante et cadrée à droite. L'exposant est ajusté de sorte que le 1er chiffre de la mantisse soit significatif.
Si s est omis on aura :

$$s = d+1 \quad \text{et} \quad w = s+6$$

4. Nombres complexes C(desprip$_r$ [, descrip$_i$])

- descrip$_r$ et descrip$_i$ sont les descriptions (E ou F) de la partie réelle et de la partie imaginaire.
  Si descrip$_i$ est omis, descrip$_r$ est utilisé 2 fois.
- En entrée comme en sortie, partie réelle et partie imaginaire sont juxtaposées dans le flot de transmission. En particulier le caractère I n'apparait pas.

5. Remarque

L'emploi des descriptions E ou F n'implique nullement que les variables associées soient DEC FIXED ou DEC FLOAT.

6. Exemples

```
1       PLED:  PROC OPTIONS(MAIN);
2           DCL A  DEC FLOAT  ;
3           DCL FOR(12) LABEL ;
4           PUT SKIP EDIT('    123456789012345678901')(A);
5           A = -764.25
6               DO  I = 1 TO 12 ;
7               PUT SKIP EDIT ( I, A)(F(2),X(1),R(FOR(I)));
8               END ;
9   FOR(1): FORMAT  (F(8,3));
10  FOR(2): FORMAT  (F(8));
11  FOR(3): FORMAT  (F(9,3,1));
12  FOR(4): FORMAT  (F(9,3,-1));
13  FOR(5): FORMAT  (F(8,1));
14  FOR(6): FORMAT (F(8,0,1));
15  FOR(7): FORMAT (E(10,3));
16  FOR(8): FORMAT (E(10,2));
17  FOR(9): FORMAT (E(10,3,4));
18  FOR(10): FORMAT (E(12,5));
19  FOR(11): FORMAT (E(15,2,5 ));
20  FOR(12): FORMAT (E(9,3));
21          PUT SKIP(3) LIST('F I N');
22      END PLED;
```

```
 1 -764.250
 2    -764
 3 -7642.500
 4  -76.425
 5  -764.3
 6   -7643
 7 -7.643E+02
 8  -7.64E+02
 9 -7.643E+02
10 -7.64250E+02
11     -764.25E+00
12 7.643E+02
```

§ . 3 . 4 . 5 DESCRIPTIONS DE CHAINES A [(x)] , B [(w)]

1. Généralités

A décrit une valeur chaine de caractères et B une valeur chaine de bits. w est le nombre des positions de la zone ; ce peut être une expression.

2. En entrée

w est obligatoire sinon il est pris égal à 1 ; il précise donc le nombre de caractères à lire dans le flot d'entrée.

Si w = 0, une chaine nulle est transmise.
Pour la description B, la zone ne doit contenir que les caractères 0 ou 1 .
Les délimiteurs de chaine (' et B) ne doivent pas apparaitre dans le flot.

3. En sortie

La valeur de chaine est cadrée à gauche et ajustée à droite jusqu'à la wème position. Si w est omis il est pris égal à la longueur de la valeur transmise ; il n'y a alors ni padding ni troncature à droite.
Si w est nul, l'élément de donnée n'est pas transmis.
Les bits '0'B et '1'B sont convertis en caractères et chacun occupe donc une position dans le flot de sortie.

4. Exemples

```
 8      PUT SKIP EDIT('NOUS SOMMES LE :', JR   ,MONTH(MOI),'19'||AN)
                     (A(17)   ,  X(2) ,A(2),X(1),A,X(2),  A  );

 9      PUT SKIP(3) EDIT('IL EST EXACTEMENT : ')(A);
10      DO I = 1,3,5,7 ;
11        PUT EDIT(SUBSTR(TEMPS,I,2+(I=7)))(A);
12        PUT EDIT(CAR(I/2))(X(1),A(4));
13        END;


13          BR = 2**I ;
14           PUT EDIT(I,BR,UNSPEC(BR),UNSPEC(-BR))
                    (COL(1),F(2),F(5),COL(28),B(16),COL(58),B(16));
```

§ . 3 . 4 . 6 CONTROLEURS DE LA TRANSMISSION

1. Saut de caractères X (w)

w est le nombre des caractères du flot à sauter, donc à ne pas transmettre En lecture, les w positions de la carte sont sautées. En sortie, aucun caractère n'est transmis dans ces positions ; la ligne d'impression comporte donc autant de blancs.

2. Positionnement de caractère : COLUMN (w)

w est la position de caractère à l'intérieur de l'enregistrement (carte de 80 colonnes, ligne de 120 ou 132 positions) à partir de laquelle la transmission doit débuter.

Si w = 0 ou si w dépasse la longueur de l'enregistrement, il est ramené à 1.

En entrée comme en sortie, les caractères intermédiaires sont ignorés La ligne d'impression comportera donc des blancs.

Si la position w est déjà dépassée dans l'enregistrement en cours, celui-ci est alors abandonné et le contrôleur s'applique à l'enregistrement suivant.

3. Saut d'enregistrements SKIP [(w)]

Ce contrôleur du flot a le même rôle que l'option SKIP du GET et du PUT. Cf § . 3.1.2.3. et 3.1.2.4.

4. Positionnement de la ligne d'impression LINE (w)

Ce contrôleur a le même rôle que l'option LINE du PUT. Cf § 3.1.2.4.

5. Saut de page PAGE

Cf § 3.1.2.4.

6. Remarques

- Les différents contrôleurs sont compatibles et peuvent apparaitre concurremment dans une même liste-formats.

- Au contraire des options correspondantes, ils sont exécutés au moment où ils sont rencontrés dans l'exploration de liste-formats, à condition que liste-données ne soit pas déjà épuisé.

7. Exemples

```
8         PUT SKIP;
9       PUT SKIP EDIT('NOUS SOMMES LE :', JR   ,MONTH(MOI),'19'||AN)
                      (A(17)   ,  X(2) ,A(2),X(1),A,X(2),  A   );


13              PUT SKIP LIST(PRENOM || ' ' || NOM);
14              LP=LENGTH(PRENOM);
15              LN=LENGTH(NOM);
16              PUT EDIT (LP,LN)
                         (COL(LP+LN+3),2 F(4));


18        PUT EDIT(B)(COL(2),A);
19        PUT LINE(71)EDIT('PAGE',NPAG)(COL(17),A,F(5));

20        IF OR='1'B THEN DO;
21                        PUT LINE(80)LIST('F I N');
22                        STOP;
23                         END;

24        TITRE:  NPAG=NPAG+1;
25             PUT PAGE LINE(2) EDIT(C,
                                A,'F A C T O R I E L L E',A,
                                A,'N', A,'DE N',A,
                                A,A,'IC**',A,
                                B,
                                A,A,A,A)
                                (COL(2),A,
                                 COL(2),A,COL(14),A,COL(41),A,
                                 COL(2),A,COL(7),A,COL(11),A,COL(21),A,
                                                           COL(41),A,
                                 COL(2),A,COL(11),A,COL(35),A,COL(41),A,
                                 COL(2),A,
                                 COL(2),A,COL(11),A,COL(32),A,COL(41),A);
```

§. 3. 4. 7 INSTRUCTION FORMAT

1. Instruction FORMAT

étiq : FORMAT (liste-formats) ;

étiq : est obligatoire ; elle peut être indicée.

liste-formats est construite comme précédemment avec la seule restriction que l'indicateur de format à distance R est exclu. Un FORMAT à distance ne peut pas renvoyer à un autre FORMAT à distance.

Dans n'importe quelle instruction de transmission du bloc il est possible de s'y référer au moyen de l'indicateur R.

2. Indicateur de format à distance R (étiq)

- étiq doit être l'étiquette d'une instruction FORMAT connu dans le bloc.
- Tout se passe comme si l'expression R (étiq) était remplacée par la liste-format du FORMAT.
- R (étiq) peut être précédé d'un facteur d'itération.

3. Exemples

```
29          IF LINENO(SYSPRINT) > 1 THEN PUT EDIT((30)'-',' ')
                                           (SKIP(2),A);
30          PUT SKIP EDIT('TABLEAU',((A(I,J) DO I=1 TO 3)
                                        DO J=1 TO 3))
                                       (A,SKIP(2), 3 R(FORME));
31          PUT SKIP(2);
32      FORME:  FORMAT(COL(1), 3 F(4));


11              PUT SKIP(2) EDIT(Z,ASH,SINH(ASH))
                                (R(FOR));
12              PUT SKIP EDIT(ACH,COSH(ACH))
                             (R(FAR),  2 R(FUR));
13              END;
14      FAR: FORMAT(COL(5));
15      FOR: FORMAT(F(4,1),2 F(8,5));
16      FUR: FORMAT ( F(8,5));
```

ANNEXE

PROGRAMMES D'INTRODUCTION

```
// OPTION LOG
// OPTION LINK
  ACTION NOMAP
// EXEC PLIOPT
```

```
DOS PL/I OPTIMIZING COMPILER          V1 RL 4.2 PTF32          TIME: 08.13.39    DATE: 13 MAY 76          PAGE    1
IEL0010I I      FOR BEST COMPILE SPEED, ALLOCATE WORKFILES TO FULL CYLINDERS ON SEPARATE DEVICES.  IF ONLY ONE
                SPILL DEVICE, USE EQUAL SPLIT-CYLINDER ALLOCATION.
```

```
PL/I OPTIMIZING COMPILER          MOYENNE: PROCEDURE OPTIONS(MAIN);          PAGE    2

                    SOURCE LISTING

    STMT LEV NT

      1       0     |  MOYENNE: PROCEDURE OPTIONS(MAIN);
      2   1   0     |      NBRX=0;
      3   1   0     |      SOMME=0;
      4   1   0   R |  EAD: GET LIST(X);
      5   1   0     |      PUT DATA( X );
      6   1   0     |      NBRX=NBRX+1
      7   1   0     |      SOMME=SOMME+X;
      8   1   0     |      GOTO READ;
      9   1   0     |  PUT DATA (NBRX,SOMME);
     10   1   0     |  X_BARRE = SOMME/NBRX;
     11   1   0     |      PUTLIST(X-BARRE);
     12   1   0     |  END MOYENNE;
```

PROGRAMME N° 1

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
4	EAD	/* STATEMENT LABEL CONSTANT */
1	MOYENNE	EXTERNAL ENTRY RETURNS(BINARY FIXED (15,0))
********	NBRX	AUTOMATIC ALIGNED BINARY FIXED (15,0) 2,6,6,9,10
********	READ	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6)
********	SOMME	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 3,7,7,9,10
********	SYSIN	EXTERNAL FILE 4
********	SYSPRINT	EXTERNAL FILE PRINT 5,9
********	X	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 4,5,7
********	X_BARRE	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 10

COMPILER DIAGNOSTIC MESSAGES

ERROR ID	L	STMT	MESSAGE DESCRIPTION
SEVERE AND ERROR DIAGNOSTIC MESSAGES			
IEL0399I	E	6	SEMICOLON ASSUMED AFTER 'NBRX=NBRX+1'.
IEL0514I	S	8	IDENTIFIER 'READ' AFTER 'GOTO' IS NOT A LABEL KNOWN IN THE CURRENT BLOCK. STATEMENT IGNORED.
IEL0304I	S	11	INVALID SYNTAX IN ASSIGNMENT STATEMENT AFTER 'PUTLIST(X-BARRE)'. 'PUTLIST(X-BARRE)' IGNORED.
END OF COMPILER DIAGNOSTIC MESSAGES			

COMPILE TIME 0.24 MINS SPILL FILE: 0 RECORDS, SIZE 1691

COMPILATION ENDED BY 'NOCOMPILE' OPTION

// EXEC LNKEDT

PLIOPT CPU=00006*10-4H NSIO=000018 #0000 F #

1S13I STATEMENT OUT OF SEQUENCE.
1170I JOB TNIACW11 CANCELLED DUE TO CONTROL STATEMENT ERROR
EOJ TNIACW11 DATE 13/05/76,CLOCK 08/13/56,DURATION 00/00/28

EOJ TNIACW11*010BELHACHE DATE=13/05/76(081355)

LNKEDT CPU=00000*10-4H NSIO=000002 #0000 F #
COUT TOTAL OU JOB ---------- #0000 F #

SOURCE LISTING

```
STMT LEV NT

   1      0   |  MOYENNE: PROC OPTIONS(MAIN);                          |
   2    1 0   |       NBRX=0;                                          |
   3    1 0   |       SOMME=0;                                         |
   4    1 0   |  READ: GET LIST(X);                                    |
   5    1 0   |        PUT DATA(X);                                    |
   6    1 0   |        NBRX=NBRX+1;                                    |
   7    1 0   |        SOMME=SOMME+1;                                  |
   8    1 0   |        GOTO READ;                                      |
   9    1 0   |       PUT DATA(NBRX,SOMME);                            |
  10    1 0   |       X_BARRE=SOMME/NBRX;                              |
  11    1 0   |       PUT SKIP LIST(X-BARRE) ;                         |
  12    1 0   |  END MOYENNE;                                          |
```


ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
********	BARRE	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 11
1	MOYENNE	EXTERNAL ENTRY RETURNS(BINARY FIXED (15,0))
********	NBRX	AUTOMATIC ALIGNED BINARY FIXED (15,0) 2,6,6,9,10
4	READ	/* STATEMENT LABEL CONSTANT */ 8
********	SOMME	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 3,7,7,9,10
********	SYSIN	EXTERNAL FILE 4
********	SYSPRINT	EXTERNAL FILE PRINT 5,9,11
********	X	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 4,5,11
********	X_BARRE	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 10

COMPILER DIAGNOSTIC MESSAGES

ERROR ID L STMT MESSAGE DESCRIPTION

COMPILER INFORMATORY MESSAGES

IEL0972I I 'BUFFERS(1)' DEFAULTED FOR FILE 'SYSPRINT'. EXECUTION TIME CAN BE IMPROVED BY DECLARING 'BUFFERS(2)'.

IEL0972I I 'BUFFERS(1)' DEFAULTED FOR FILE 'SYSIN'. EXECUTION TIME CAN BE IMPROVED BY DECLARING 'BUFFERS(2)'.

END OF COMPILER DIAGNOSTIC MESSAGES

COMPILE TIME 0.51 MINS SPILL FILE: 16 RECORDS, SIZE 1691

PLIOPT CPU=00014*10-4H NSIO=000123 #0001 F #

LNKEDT CPU=00013*10-4H NSIO=001251 #0004 F #

1? 11? 6? 4? 10? 3? 2? 5? 5? 8?

```
X= 1.00000E+00;          X= 1.10000E+01;          X= 6.00000E+00;          X= 4.00000E+00;          X= 1.00000E+01;
X= 3.00000E+00;          X= 2.00000E+00;          X= 5.00000E+00;          X= 5.00000E+00;          X= 8.00000E+00;

IBM1811I 'ONCODE'=0070 'ENDFILE' CONDITION RAISED     ('ONFILE'= SYSIN)
   IN STATEMENT 4 AT OFFSET +0000A2 IN PROCEDURE WITH ENTRY MOYENNE


0S02I JOB TNIACW12 CANCELED DUE TO PROGRAM REQUEST
0S07I PROBLEM PROGRAM PSW     072D0E0000063BC0


EOJ  TNIACW12*010BELHACHE   DATE=13/05/76(081544)


          CPU=00003*10-4H  NSID=000005  #0000 F #
 COUT TOTAL DU JOB ----------------------  #0006 F #
```

SOURCE LISTING

```
STMT LEV NT

   1      0  | MOYENNE:  PROC OPTIONS(MAIN);
              | /*
              | *   MOYENNE DE NBRX NOMBRES
              | */

              |  /*  GESTION DE LA FIN DES LECTURES  */
   2    1  0 |      ON ENDFILE(SYSIN) GOTO FIN;
              |  /*  EN CAS D'ERREUR      */
   3    1  0 |      ON ERROR BEGIN; ON ERROR SYSTEM;
   5    2  0 |      PUT SKIP(3) DATA; END;

              |  /*  DECLARATIONS - INITIALISATIONS  */
   7    1  0 |      DCL NBRX REAL DEC FIXED(3);
   8    1  0 |      DCL SOMME DEC(6) INIT(0);

   9    1  0 |      READ: GET LIST( X ) COPY;
  10    1  0 |            PUT SKIP DATA(X);
  11    1  0 |            SOMME=SOMME + X;
  12    1  0 |            NBRX=NBRX+1;
  13    1  0 |            GOTO READ;

  14    1  0 | FIN: PUT LINE(33) LIST( (20)'-');
  15    1  0 |      PUT SKIP DATA(NBRX,SOMME);
  16    1  0 |      PUT LIST (SOMME/NBRX);

  17    1  0 | END MOYENNE;
```

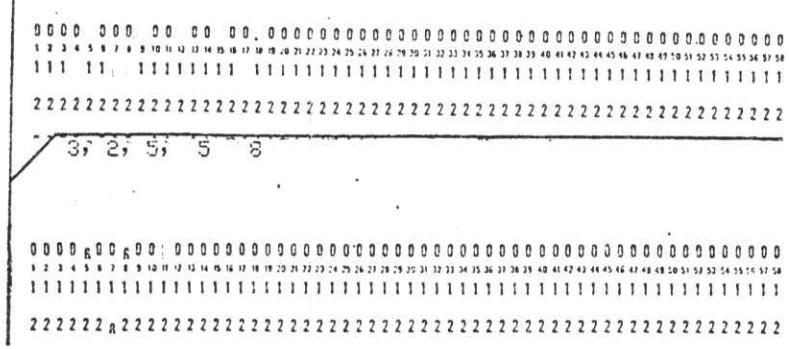

ATTRIBUTE AND CROSS-REFERENCE TABLE

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
14	FIN	/* STATEMENT LABEL CONSTANT */ 2
1	MOYENNE	EXTERNAL ENTRY RETURNS(BINARY FIXED (15,0))
7	NBRX	AUTOMATIC ALIGNED DECIMAL FIXED (3,0) 12,12,15,16
9	READ	/* STATEMENT LABEL CONSTANT */ 13
8	SOMME	AUTOMATIC ALIGNED INITIAL DECIMAL /* SINGLE */ FLOAT (6) 1,11,11,15,16
********	SYSIN	EXTERNAL FILE 2,9
********	SYSPRINT	EXTERNAL FILE PRINT 9,10,14,15,16 5
********	X	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 9,10,11

```
   1,
X= 1.00000E+00;
IBM537I  'ONCODE'=8097  DATA EXCEPTION
   IN STATEMENT 12 AT OFFSET +000148 IN PROCEDURE WITH ENTRY MOYENNE

SOMME= 1.00000E+00          NBRX=00148

IBM537I  'ONCODE'=8097  DATA EXCEPTION
   IN STATEMENT 5 AT OFFSET +0000A0 IN A 'ERR ' ON-UNIT



OS02I JOB TNIACW13 CANCELED DUE TO PROGRAM REQUEST
OS07I PROBLEM PROGRAM PSW     072D0E0000063DA0


EOJ  TNIACW13*010BELHACHE   DATE=13/05/76(081747)


          CPU=00003*10-4H  NSIO=000005  #0000 F #
 COUT TOTAL DU JOB ----------------------  #0007 F #
```

```
PL/I CPTIMIZING COMPILER          MOYENNE:

                    SOURCE LISTING

 STMT

    1    MOYENNE:
         /*
         *    CALCUL DE LA MOYENNE DE
         *         NBRX  NOMBRES
         */
              PROC OPTIONS(MAIN);

    2         ON ENDFILE(SYSIN)GOTO FIN;

    3         ON ERROR BEGIN;
    4           ON ERROR SYSTEM;
    5                 PUT SKIP(2)DATA;
    6                 END;

         /*
         *    DECLARATIONS : INITIALISATIONS
         */
    7         DCL (NBRX FIXED(3), SOMME DEC FLOAT(6))
                   INIT(0);

    8         READ: GET LIST(X) COPY;
    9               PUT SKIP DATA(X);
   10                 NBRX=NBRX+1; SOMME=SOMME+X;
   12                 GOTO READ;

   13         FIN:   PUT LINE(33) LIST((20)'-');
   14                PUT SKIP DATA(NBRX,SOMME);
   15                PUT(SOMME/NBRX) SKIP;

   16    END MOYENNE;
```

```
 1, 11, 6, 4, 10
```



```
 3, 2, 5, 5  8
```



```
    1,
 X= 1.CCCCCE+00;
   11,
 X= 1.10000E+01;
    6,
 X= 6.00000E+00;
    4,
 X= 4.00000E+00;
   10
 X= 1.00000E+01;
    3
 X= 3.00000E+00;
 2
 X= 2.00000E+00;
 5
 X= 5.00000E+00;
 5
 X= 5.00000E+00;
 8
 X= 8.00000E+00;

 ---------------------
 NBRX=      10                    SOMME= 5.50000E+01;
  5.50000E+00

 EOJ   TNIACW14*010BELHACHE    DATE=13/05/76(081946)


                      CPU=00001*10-4H  NSIO=000005   #0000 F #
 COUT TOTAL DU JOB ---------------------              #0007 F #
```

```
1       SECOND3:   PROC    OPTIONS(MAIN);
        /*
        *   EQUATION     A*X**2 + B*X + C = 0
        *   NOMBRE  &  NATURE  DES  RACINES
        */
2           ON ENDFILE(SYSIN)  GOTO FIN;

        /*  DECLARATION DES VARIABLES     CHARACTER(LONGUEUR)    */
3           DCL  ( MESSAGE CHAR(30), (NB,S1,S2) CHAR(1),
                   NATUR CHAR(9), MULT CHAR(11) ) VAR;
4           LEC: GET LIST(A,B,C) COPY;
5                DLTA = B**2 - 4*A*C;
6                IF DLTA = 0 THEN DO;
7                            NB = '1' ;
8                            S1, S2 = '' ;
9                            MULT = ' DOUBLE' ;
10                               END;
11                           ELSE DO;
12                           NB = '2' ;
13                           S1,S2 = 'S' ;
14                           MULT = ' DISTINCTES';
15                               END ;
16               IF DLTA >= 0 THEN NATUR = ' REELLE';
17                            ELSE NATUR = ' COMPLEXE';
18               MESSAGE = NB || ' RACINE' || S1 || NATUR || S2 || MULT ;
19                     PUT SKIP LIST(MESSAGE);
20                     PUT SKIP(2);
21                     GOTO LEC;
22               FIN:PUT SKIP(3) LIST('FIN DES MESSAGES');
23                  END SECOND3;
```

```
   1, 2,1
1 RACINE REELLE DOUBLE
   1 -5  6
2 RACINES REELLES DISTINCTES
   1, 1 ,1
2 RACINES COMPLEXES DISTINCTES


FIN DES MESSAGES
```

DCL NO.	IDENTIFIER	ATTRIBUTES AND REFERENCES
********	A	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 4,5
********	B	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 4,5
********	C	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 4,5
********	DLTA	AUTOMATIC ALIGNED DECIMAL /* SINGLE */ FLOAT (6) 5,6,16
22	FIN	/* STATEMENT LABEL CONSTANT */ 2
4	LEC	/* STATEMENT LABEL CONSTANT */ 21
3	MESSAGE	AUTOMATIC UNALIGNED CHARACTER (30) VARYING 18,19
3	MULT	AUTOMATIC UNALIGNED CHARACTER (11) VARYING 9,14,18
3	NATUR	AUTOMATIC UNALIGNED CHARACTER (9) VARYING 16,17,18
3	NB	AUTOMATIC UNALIGNED CHARACTER (1) VARYING 7,12,18
1	SECOND3	EXTERNAL ENTRY RETURNS(DECIMAL /* SINGLE */ FLOAT (6))
********	SYSIN	EXTERNAL FILE 2,4
********	SYSPRINT	EXTERNAL FILE PRINT 4,19,20,22
3	S1	AUTOMATIC UNALIGNED CHARACTER (1) VARYING 8,13,18
3	S2	AUTOMATIC UNALIGNED CHARACTER (1) VARYING 8,13,18

```
1       (CHECK):
        SECOND3:   PROC    OPTIONS(MAIN);
        /*
        *   EQUATION    A*X**2 + B*X + C = 0
        *   NOMBRE  &  NATURE  DES  RACINES
        */
2       ON ENDFILE(SYSIN)  GOTO FIN;

        /*  DECLARATION DES VARIABLES    CHARACTER(LON
```

```
LEC;
   1, 2,1
A= 1.00000E+00;
B= 2.00000E+00;
C= 1.00000E+00;
DLTA= 0.00000E+00;
NB='1';
S1='';
S2='';
MULT=' DOUBLE';
NATUR=' REELLE';
MESSAGE='1 RACINE REELLE DOUBLE';
1 RACINE REELLE DOUBLE

LEC;
   1 -5  6
A= 1.00000E+00;
B=-5.00000E+00;
C= 6.00000E+00;
DLTA= 1.00000E+00;
NB='2';
S1='S';
S2='S';
MULT=' DISTINCTES';
NATUR=' REELLE';
MESSAGE='2 RACINES REELLES DISTINCTES';
2 RACINES REELLES DISTINCTES

LEC;
   1, 1 ,1
A= 1.00000E+00;
B= 1.00000E+00;
C= 1.00000E+00;
DLTA=-3.00000E+00;
NB='2';
S1='S';
S2='S';
MULT=' DISTINCTES';
NATUR=' COMPLEXE';
MESSAGE='2 RACINES COMPLEXES DISTINCTES';
2 RACINES COMPLEXES DISTINCTES

LEC;


FIN;

FIN DES MESSAGES

EOJ  TNIACW21*010BELHACHE    DATE=13/05/76(
```

```
1     TRI1:
      /*
      *    CONSTRUCTION ET IMPRESSION
      *         D'UN   ''TRIANGLE''  DE  'O'
      */
           PROC OPTIONS(MAIN);

2          DCL A(5) CHAR(5) INIT((5)(3)'*');
3          DCL DIREK LABEL, N FIXED(1) INIT(0);

4     CAS1: DIREK=CAS2;
5             A(1)='O';
6             DO I=2 TO 5;
7             A(I)=A(I-1)||'O';
8             END;
9          GOTO ECRI;

10    CAS2: DIREK=CAS3;
11            A(1)='O';
12            DO I=2 TO 5;
13            A(I)='O'||A(I-1);
14            END;
15            GOTO ECRI;

16    CAS3: DIREK=FIN;
17            A(5)=(5)'O';
18            DO I=4 TO 1 BY -1;
19            A(I)=' '||A(I+1);
20            END;
21            GOTO ECRI;

22    ECRI: PUT SKIP(3);
23          N=N+1;
24          PUT SKIP LIST('NO.'||N);
25          PUT SKIP;
26          DO I=1 TO 5;
27          PUT SKIP LIST(A(I));
28          END;

29            A=' ';
30            GOTO DIREK;

31    FIN : END TRI1;
```

```
NO.    1

O
O
O
O
O

NO.    2

O
OO
OOO
OOOO
OOOOO

NO.    3

    O
   OO
  OOO
 OOOO
OOOOO

EOJ   TNIACW23*
```

PROGRAMME N° 6

1.113

IELC892I W 13, 19 TARGET STRING SHORTER THAN SOURCE. RESULT TRUNCATED ON ASSIGNMENT.

```
1     TRI2:
      /*
      *    TRIANGLE  DE  '0'
      *         CHAINES  VARYING
      */
           PROC OPTIONS(MAIN);

2          DCL A(5) CHAR(5) VAR INIT((5)(' ')),
               DIREK(2) LABEL INIT(CAS2,FIN);

3     CAS1: N=1;
4             A(1)='0';
5             DO I=2 TO 5;
6             A(I)=A(I-1)||'0';
7             END;
8             GOTO ECRI;

9     CAS2: N=2;
10            A(1)='0';
11            DO I=2 TO 5;
12            A(I)='0'||A(I-1);
13            END;
14            GOTO ECRI;

15    ECRI: PUT SKIP(3);
16          PUT SKIP LIST('NO.'||N);
17          PUT SKIP;

18          DO I=1 TO 5;
19          PUT SKIP LIST(A(I),LENGTH(A(I)) );
20          END;

21         A='';
22         GOTO DIREK(N);

23    FIN:  END TRI2;
```

```
NO.          1

0                                                    1
00                                                   2
000                                                  3
0000                                                 4
00000                                                5

NO.          2

0                                                    1
00                                                   2
000                                                  3
0000                                                 4
00000                                                5

EOJ  TNIACW24*010BELHACHE    DATE=13/05/76(115251)
```

```
1       SUBINDX:    PROC OPTIONS(MAIN);

        /*
        *    FONCTIONS DE MANIPULATION DE CHAINES
        *       SUBSTR  -  INDEX  -  LENGTH
        */

2           DCL     TXT CHAR(32),
                    (NOM, PRENOM)    CHAR(20)  VAR   ;

3           ON ENDFILE(SYSIN)  GOTO FIN;

4       LEC:    GET LIST(TXT)   COPY ;
5               DO I = 1 TO 32 ;
6                   IF  SUBSTR(TXT,I,1) = '/'  THEN     DO ;
7                                          NOM = SUBSTR(TXT,1,I-1);
8                                          GOTO SUITE;
9                                              END;
10              END;

11          SUITE:
                J = INDEX (TXT, '*') ;
12              PRENOM = SUBSTR(TXT,I+1,J-I-1);

13              PUT SKIP LIST(PRENOM || ' '  || NOM);
14              LP=LENGTH(PRENOM);
15              LN=LENGTH(NOM);
16              PUT EDIT (LP,LN)
                         (COL(LP+LN+3),2 F(4));

17              PUT SKIP(2) LIST ((50)'-');
18              GOTO LEC;

19      FIN:    PUT SKIP(3) LIST('FIN');
20              END SUBINDX;
```

```
'DE SAINT-EXUPERY / ANTOINE*'
 ANTOINE DE SAINT-EXUPERY      8   17
------------------------------------
'IBSEN/ HENRIK*'
 HENRIK IBSEN     7    5
------------------------------------
'MORE/THOMAS*'
THOMAS MORE      6    4
------------------------------------
'ROUSSEAU/JEAN-JACQUES*'
JEAN-JACQUES ROUSSEAU    12    8
------------------------------------
  'SHAW / GEORGE BERNARD*        '
 GEORGE BERNARD SHAW      15    5
------------------------------------
'VON GOETHE / JOHANN WOLFGANG*'
 JOHANN WOLFGANG VON GOETHE      16   11
------------------------------------

FIN
```

```
1     RACUB:
      /*
      *    TABLE DE RACINES CUBIQUES
      */
           PROC OPTIONS(MAIN);

2          DCL RC(2,10) INIT ((10)*,(10)0)
                        FIXED(9,6);
3          ON ENDFILE(SYSIN) GOTO FIN;

4     LEC: GET LIST(A,N,PAS,EPS) COPY;
5          IF N>10 | EPS<1E-6 | A>999 THEN DO;
6                  PUT SKIP(2) LIST('HORS-LIMITES');
7                              GOTO LEC;
8                                   END;

9          DO I=1 TO N BY 1;
10         RC(1,I)=A+(I-1)*PAS;
11         END;

12         X1=RC(1,1)/3;

13    BCLX: DO I=1 BY 1 TO N;
14             A=RC(1,I);
15             EKR=1;

16             DO WHILE(EKR>=EPS);
17             X0=X1;
18             X1=(2*X0+A/(X0*X0))/3;
19             EKR=ABS(X1-X0);
20             END;

21             RC(2,I)=X1;
22         END BCLX;

23         PUT SKIP(2);
24         DO I=1 TO N;
25         PUT EDIT(RC(*,I)) ( 2 F(11,6));
26         IF MOD(I,2)=0 THEN PUT SKIP;
27         END;

28         PUT SKIP(2) LIST((44)'*');
29         GOTO LEC;

30    FIN: END RACUB;
```

```
   1 5, 2,  1E-3

   1.000000    1.000000    3.000000    1.442249
   5.000000    1.709975    7.000000    1.912930
   9.000000    2.080083
**********************************************
   1, 11, 10, 1E-7

HORS-LIMITES
   10, 10, 10, 1E-6

  10.000000    2.154434   20.000000    2.714417
  30.000000    3.107232   40.000000    3.419951
  50.000000    3.684031   60.000000    3.914867
  70.000000    4.121285   80.000000    4.308869
  90.000000    4.481405  100.000000    4.641589

**********************************************

EOJ   TNIACW31*010BELHACHE    DATE=13/05/76(182657)
```

```
1     P42: PROC OPTIONS(MAIN);

      /*
      *    CONSTRUCTION DU TABLEAU
      *
      *         MMMMM
      *         MOOOM
      *         MOOOM
      *         MOOOM
      *         MMMMM
      */

2          DCL   (X,A,B, W(5,5) ) CHAR(1),
                 Y(4) LABEL, N FIXED(1);

3          ON ENDFILE(SYSIN) GOTO FIN;

4          Y(4): N = 1;
5          LEC: PUT PAGE;  W = ' ';
7               GET LIST (A,B) COPY;

      /*
      *    1.ERE  METHODE
      */
8     Y(1): Y1:  DO I=2 TO 4;DO J=2 TO 4;
10         W(I,J) = B;
11    ----> END Y1;

12    ZU: DO I=1, 5 ; DO J = 1 BY 1 TO 5;
14        W(I,J), W(J,I) = A;
15    ----> END ZU;
16         GOTO ECRI;
```

CAS N. 1

```
MMMMM
MOOOM
MOOOM
MOOOM
MMMMM
```

```
      /*
      *    2.EME METHODE
      */
17    Y(2):    K=0;
18         Z2: DO  X =A, B;
19             K = K+1 ;
20                 DO I = K TO 6-K;
21                 DO J = 6-K TO K BY -1;
22                 W(I,J) = X;
23         ----> END Z2;
24             GOTO ECRI;
```

CAS N. 2

```
MMMMM
MOOOM
MOOOM
MOOOM
MMMMM
```

```
      /*
      *    3.EME METHODE
      */
25    Y(3):    W = B;
26             DO I =1,5;
27             W(I,*), W(*,I) = A;
28             END;
29             GO TO ECRI;
```

CAS N. 3

```
MMMMM
MOOOM
MOOOM
MOOOM
MMMMM
```

```
30         ECRI: PUT SKIP(5) LIST('CAS N.'||N);
31               PUT SKIP(3)EDIT((W(I,*) DO I=1 TO 5))
                       ( COLUMN(1), 5 A(1) );
32               N = N+1;
33               GOTO Y(N);

34         FIN:
           PUT SKIP(3) LIST('F I N');
35         END P42;
```

```
IEL0385I I   11    MULTIPLE CLOSURE OF BLOCK.    1 EXTRA 'END' STATEMENT(S) ASSUMED.
IEL0385I I   15    MULTIPLE CLOSURE OF BLOCK.    1 EXTRA 'END' STATEMENT(S) ASSUMED.
IEL0385I I   23    MULTIPLE CLOSURE OF BLOCK.    2 EXTRA 'END' STATEMENT(S) ASSUMED.
```

```
1     DATHEUR:    PROC OPTIONS(MAIN);

       /*
      *  FONCTIONS  DATE &  TIME
      *
      */

2        DCL |JOUR CHAR(6),
             |AN CHAR(2) DEF JOUR,
             |MOI CHAR(2) DEF JOUR POS(3),
             |JR CHAR(2) DEF JOUR POS(5);

3        DCL MONTH(12) CHAR(17) VAR INIT('JANVIER','FEVRIER','MARS',
              'AVRIL','MAI','JUIN','JUILLET','AOUT','SEPTEMBRE',
              'OCTOBRE','NOVEMBRE','DECEMBRE');


4        DCL |TEMPS   CHAR(9),
             |TIME  BUILTIN,
             |CAR(0:3)CHAR(2) VAR INIT('H','MN','S','MS');

5         \JOUR = DATE();
6         \TEMPS = TIME ;

7         PUT PAGE EDIT(('123456789 ' DO I=1 TO 5))(A);

8         PUT SKIP;
9        PUT SKIP EDIT('NOUS SOMMES LE :', JR   ,MONTH(MOI),'19'||AN)
                       (A(17)  ,  X(2) ,A(2),X(1),A,X(2),  A   );

10        PUT SKIP(3) EDIT('IL EST EXACTEMENT : ')(A);
11        DO I = 1,3,5,7 ;
12          PUT EDIT(SUBSTR(TEMPS,I,2+(I=7)))(A);
13          PUT EDIT(CAR(I/2))(X(1),A(4));
14         END;

15         PUT SKIP(2) EDIT(('123456789 '  DO I=1 TO 5))(A);

16     END DATHEUR;
```

```
123456789 123456789 123456789 123456789 123456789
NOUS SOMMES LE :  13 MAI  1976

IL EST EXACTEMENT : 18 H   32 MN  46 S   789 MS
123456789 123456789 123456789 123456789 123456789
```

```
1       FTABLO:     PROC OPTIONS(MAIN);

        /*
        *    EXISTENCE & COMPTAGE  DANS UN TABLEAU
        *           D'ELEMENT(S)    =    2
        *
        *    FONCTIONS    ALL, ANY, SUM & PROD
        *        LBOUND,   HBOUND
        */

2           DCL     A(3,3) FIXED,  BI(3,3) BIT(1), Z BIT(1) ,
              REP(0:1) CHAR(5) INIT('N O N','O U I'),
              (LA,LB)(0:1) LABEL ;

3           DCL  D(2,2)  BIN FIXED(15);

4           ON ENDFILE(SYSIN) GOTO FIN;

5       LEC:  LB(0): LA(0):
              GET EDIT(A)(COL(1), 9 F(1));

6             CALL WRITE ;

7             BI = (A=2) ;
8             PUT SKIP LIST('Y A-T-IL AU MOINS UN ELEMENT ? ');
9           Z = ANY(BI) ;
10            PUT EDIT(REP(Z))(R(FA));
11            GOTO LA(Z);

12          LA(1):
              PUT SKIP LIST('COMBIEN SONT-ILS ?');
13            PUT EDIT( SUM(BI)) (F(4));
14            PUT SKIP(2) LIST('SONT-ILS TOUS EGAUX A 2 ?');
15          Z = ALL( BI );
16            PUT EDIT(REP(Z))(R(FA));
17            GOTO LB(Z);

18          LB(1):
              DO I=1,2;
19            D(I,1)=LBOUND(A,I);
20            D(I,2)=HBOUND(A,I);
21            END;

22            D(*,1) = D(*,2) - D(*,1) + 1;
23            NBL = PROD(D(*,1));

24            PUT SKIP(2) LIST('SONT-ILS'||NBL||' ?');
25            PUT EDIT(REP( PROD(A)=2**NBL))(R(FA));
26            GOTO LEC;

27          FA:     FORMAT(X(3),A);
```

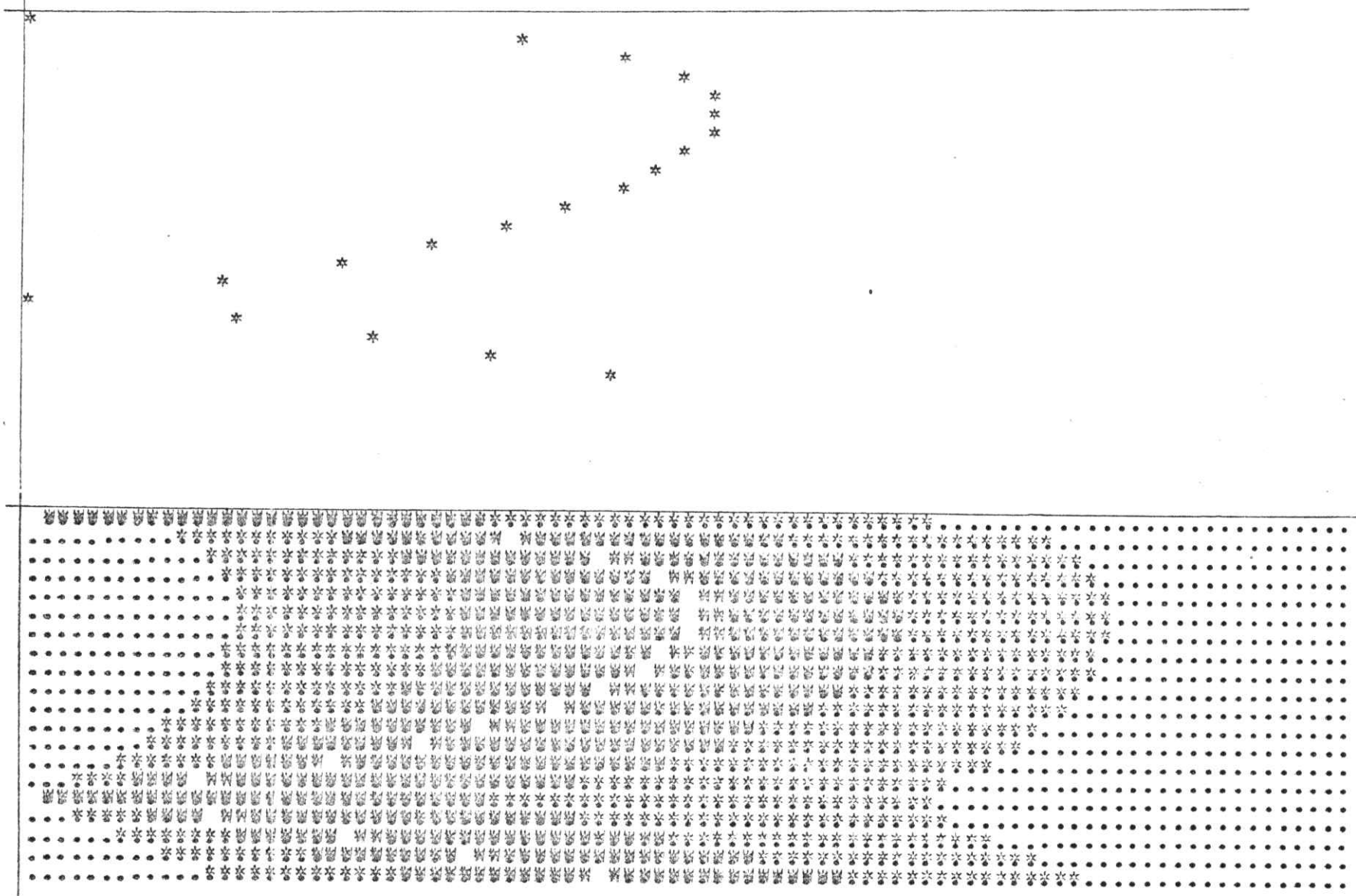
```
        /*
        *    PROCEDURE  INTERNE
        *         D'IMPRESSION  DE  A
        */
28                  WRITE:  PROC;
29                      IF LINENO(SYSPRINT) > 1 THEN PUT EDIT((30)'-',' ')
                                                     (SKIP(2),A);
30                      PUT SKIP EDIT('TABLEAU',((A(I,J) DO I=1 TO 3)
                                                    DO J=I TO 3))
                                                    (A,SKIP(2), 3 R(FORME));
31                      PUT SKIP(2);
32                  FORME:  FORMAT(COL(1), 3 F(4));
33                  END WRITE;

34          FIN:    PUT SKIP(3) LIST('F I N');
35          END FTABLO;
```

```
IEL0871I I    13   RESULT OF BUILTIN FUNCTION 'SUM' WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS.
IEL0906I I    13   DATA CONVERSION WILL BE DONE BY SUBROUTINE CALL.
IEL0871I I    23   RESULT OF BUILTIN FUNCTION 'PROD' WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS.
IEL0871I I    25   RESULT OF BUILTIN FUNCTION 'PROD' WILL BE EVALUATED USING FIXED POINT ARITHMETIC OPERATIONS.
```

```
TABLEAU

   0   5   1
   0   7   9
   1   3   1

Y A-T-IL AU MOINS UN ELEMENT ?     N O N
------------------------------

TABLEAU

   1   3   0
   0   4   1
   2   7   5

Y A-T-IL AU MOINS UN ELEMENT ?     O U I
COMBIEN SONT-ILS ?    1
SONT-ILS TOUS EGAUX A 2 ?    N O N
------------------------------

TABLEAU

   2   2   2
   2   2   2
   2   2   2

Y A-T-IL AU MOINS UN ELEMENT ?     O U I
COMBIEN SONT-ILS ?    9
SONT-ILS TOUS EGAUX A 2 ?    O U I
SONT-ILS        9 ?    O U I


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

TABLEAU

   2   2   0
   3   5   1
   4   7   0

Y A-T-IL AU MOINS UN ELEMENT ?     O U I
COMBIEN SONT-ILS ?    2
SONT-ILS TOUS EGAUX A 2 ?    N O N

F I N
```

```
1       GRAPHE:     PROC OPTIONS(MAIN);

        /*
        *    GESTION DE LA FIN DE PAGE :
        *       EDITION DU TITRE
        */
2          ON ENDPAGE(SYSPRINT) PUT PAGE EDIT
                     ('GRAPHE DE (X**3-3*X+2)**(1/3) ',
                     (50)'-',' ')
                           (COL(37),A,COL(25),A,SKIP) ;

        /*
        *    GRAPHE EN  NOIR  SUR FOND BLANC
        */
3             SIGNAL ENDPAGE(SYSPRINT);

4             DO X=-2 TO +2 BY .2;
5             Y =  30*((X**3-3*X+2)**(1/3));
6             PUT SKIP EDIT('*')( X(Y-1), A(1) );
7             END;

        /*
        *    GRAPHE  EN  BLANC
        *       SUR  FOND   NOIR DEGRADE
        */
8             DCL (R,Z) BIN FIXED;
9             PUT PAGE;
10            DO X=-2 TO +2 BY .2;
11            Y =  30*((X**3-3*X+2)**(1/3));
12            R = 90-Y;
13            Z = Y-1;

14            PUT SKIP EDIT(REPEAT('.',Z-3),REPEAT('.',R-1),
                           ('*' DO I=1 TO 2*Z/3, 1 TO 2*R/3),
                           ('W' DO I=1 TO Z/3, 1 TO R/3 ))
                           (A(Z),X(1),A(R),SKIP(0),
                           COL(Z/3),(2*Z/3)(A(1)),X(1),(2*R/3)(A(1)),
                           SKIP(0), COL(2*Z/3),(Z/3)(A(1)),X(1),
                           (R/3)(A(1)) ) ;
15            END;

16            PUT EDIT('FIN DU GRAPHE')(SKIP(5),A);
17         END GRAPHE;
```

GRAPHE DE (X**3-3*X+2)**(1/3)



FIN DU GRAPHE