Aspects non numériques de la programmation algol

Huu Dung Nguyen

N° d'ordre

T H E S E

présentée à la Faculté des Sciences
de l'Université de GRENOBLE

pour obtenir

le titre de Docteur de Troisième Cycle

" Mathématiques Appliquées "

par

NGUYEN HUU DUNG

ASPECTS NON NUMERIQUES DE LA PROGRAMMATION

ALGOL

-=-=-=-=-=-=-=-=-=-

Thèse soutenue le 25 juin 1965 devant la Commission d'Examen :

MM : KUNTZMANN Président

VAUQUOIS
GASTINEL Examinateurs
BOLLIET

FACULTE DES SCIENCES

LISTE DES PROFESSEURS

DOYENS HONORAIRES M. FORTRAT P.
M. MORET L.

DOYEN M. WEIL L.

PROFESSEURS TITULAIRES

MM.	NEEL L.	MAGNETISME
	HEILMANN R.	CHIMIE ORGANIQUE
	KRAVTCHENKO J.	MECANIQUE RATIONNELLE
	CHABAUTY C.	MATHEMATIQUES PURES
	PARDE M.	POTAMOLOGIE
	BENOIT J.	RADIOELECTRICITE
	CHENE M.	CHIMIE PAPETIERE
	BESSON J.	ELECTROCHIMIE
	WEIL L.	THERMODYNAMIQUE
	FELICI N.	ELECTROSTATIQUE
	KUNTZMANN J.	MATHEMATIQUES APPLIQUEES
	BARBIER R.	GEOLOGIE APPLIQUEE
	SANTON L.	MECANIQUE DES FLUIDES
	OZENDA P.	BOTANIQUE
	FALLOT M.	PHYSIQUE INDUSTRIELLE
	GALVANI O.	MATHEMATIQUES
	MOUSSA A.	CHIMIE NUCLEAIRE ET RADIOACTIVITE
	TRAYNARD P.	CHIMIE GENERALE
	SOUTIF M.	PHYSIQUE GENERALE
	CRAYA A.	HYDRODYNAMIQUE
	REULOS R.	THEORIE DES CHAMPS
	AYANT Y.	PHYSIQUE APPROFONDIE
	GALISSOT F.	MATHEMATIQUES PURES
Mlle	LUTZ E.	MATHEMATIQUES GENERALES
MM.	BLAMBERT M.	MATHEMATIQUES
	BOUCHEZ	PHYSIQUE NUCLEAIRE
	LLIBOUTRY L.	GEOPHYSIQUE
	MICHEL R.	GEOLOGIE ET MINERALOGIE
	BONNIER E.	METALLURGIE
	DESSAUX G.	PHYSIOLOGIE ANIMALE
	PILLET E.	ELECTROTECHNIQUE
	DEBELMAS J.	GEOLOGIE GENERALE
	GERBER R.	MATHEMATIQUES PURES
	PAUTHENET R.	ELECTROTECHNIQUE
	VAUQUOIS B.	CALCUL ELECTRONIQUE
	SILBER R.	MECANIQUE DES FLUIDES
	MOUSSIEGT J.	ELECTRONIQUE
	BARBIER J.C.	PHYSIQUE EXPERIMENTALE

MM.	BUYLE-BODIN M.	ELECTRONIQUE
	KOSZUL J.L.	MATHEMATIQUES
	DREYFUS B.	THERMODYNAMIQUE
	VAILLANT F.	ZOOLOGIE
	KLEIN J.	MATHEMATIQUES PURES
	SENGEL P.	ZOOLOGIE
	ARNAUD P.	CHIMIE
	BARJON R.	PHYSIQUE NULCEAIRE
	BARNOUD F.	BIOSYNTHESE DE LA CELLULOSE

PROFESSEURS ASSOCIES.

MM.	WAGNER H.	BOTANIQUE
	NAPP-ZINN K.	BOTANIQUE

PROFESSEURS SANS CHAIRE

Mme	KOFLER L.	BOTANIQUE
	DEPASSEL R.	MECANIQUE
	PERRET R.	SERVOMECANISME
Mme	BARBIER M.J.	ELECTROCHIMIE
MM.	COHEN J.	PHYSIQUE
	GIDON P.	GEOLOGIE
Mme	SOUTIF J.	PHYSIQUE GENERALE
MM.	GIRAUD P.	GEOLOGIE
	GASTINEL N.	MATHEMATIQUES APPLIQUEES
	LACAZE A.	THERMODYNAMIQUE
	GLENAT R.	CHIMIE ORGANIQUE
	BRISSONNEAU P.	PHYSIQUE GENERALE
	DUCROS P.	MINERALOGIE
	ANGLES D'AURIAC	MECANIQUE DES FLUIDES
	ROBERT A.	CHIMIE PAPETIERE
	COUMES A.	ELECTRONIQUE
	PEBAY-PEROULA	PHYSIQUE
	DEGRANGE C.	ZOOLOGIE
	GAGNAIRE D.	CHIMIE PAPETIERE
	RASSAT A.	CHIMIE
	PERRIAUX J.	GEOLOGIE
	BARRA J.	MATHEMATIQUES APPLIQUEES

PROFESSEURS HONOAIRES

MM.	FORTIER A.	MECANIQUE DES FLUIDES
	BRELOT M.	MATHEMATIQUES
	WOLFERS F.	PHYSIQUE
	DORIER A.	ZOOLOGIE

MAITRES DE CONFERENCES

MM.	BIAREZ J.	MECANIQUE DES FLUIDES
	DODU J.	MECANIQUE DES FLUIDES

MM.	DOLIQUE J.M.	ELECTRONIQUE
	HACQUES G.	MATHEMATIQUES APPLIQUEES
	LANCIA R.	PHYSIQUE AUTOMATIQUE
	POULOUJADOFF M.	ELECTROTECHNIQUE
	KAHANE A.	PHYSIQUE
Mme	BONNIER J.	CHIMIE
Mme	KAHANE J.	PHYSIQUE
MM.	DEPORTES C.	CHIMIE MINERALE
	DEPOMMIER P.	PHYSIQUE NUCLEAIRE
	CAUQUIS G.	CHIMIE GENERALE
	BONNET G.	PHYSIQUE
Mme	BOUCHE L.	MATHEMATIQUES
MM.	COLOBERT L.	PHYSIOLOGIE ANIMALE
	PAYANT J.J.	MATHEMATIQUES
	CAUBET J.P.	MATHEMATIQUES
	LAURENT P.	MATHEMATIQUES APPLIQUEES
	BERTRANDIAS J.P.	MATHEMATIQUES APPLIQUEES
	BRIERE G.	PHYSIQUE
	LAJZEROWICZ J.	PHYSIQUE
	VALENTIN J.	PHYSIQUE
	DESRE P.	METALLURGIE
	BONNETAIN L.	CHIMIE MINERALE

MAITRE DE CONFERENCE ASSOCIE

MM.	RADELLI L.	GEOLOGIE

Je tiens à exprimer ma profonde reconnaissance à

Monsieur le Professeur KUNTZMANN, Directeur de L'institut de Mathématiques Appliquées de GRENOBLE
qui a bien voulu me faire l'honneur de présider le Jury

Monsieur le Professeur VAUQUOIS, Directeur du Centre d'Etudes pour la Traduction Automatique
qui a bien voulu faire partie du Jury,

Monsieur le Professeur GASTINEL, qui s'est intéressé à mon travail, pour ses encouragements bienveillants,

Monsieur BOLLIET, Ingénieur au C.N.R.S. qui a dirigé mes recherches et qui a su les orienter vers les résultats les plus intéressants,

Monsieur COHEN, Attaché de Recherches au C.N.R.S., qui a bien voulu me faire profiter de ses expériences.

Ainsi qu'à

Mon frère NGUYEN MANH TUONG qui a su me donner de bons conseils dans les moments difficiles.

Je remercie également tous les membres du Laboratoire de Calcul dont l'aide me fut précieuse,

En particulier mes camarades COLMERAUER, DARDALHON, de CHASTELLIER, LAGARDE, LECARME et XUAN qui m'ont toujours imposé leur ambiance de travail dans la bonne humeur,

et,

Madame FRAIMEAULT, Mesdemoiselles LAFOSSE, COTON, BICAIS, Monsieur MOUNET, à qui je dois la réalisation pratique de cet ouvrage.

A ma Mère,
A la mémoire de mon Père.

ASPECTS NON NUMERIQUES DE LA PROGRAMMATION ALGOL

INTRODUCTION

La puissance d'ALGOL pour résoudre élégamment des problèmes numériques tels que l'intégration et la différentiation approchées, le calcul des valeurs propres de matrices etc... est bien démontrée. Mais tel que défini dans le rapport de base, il n'a pas d'instruments adéquats pour le traitement non numérique de l'information. Ce domaine est très important, comme exemple on peut citer :

. La compilation des langages artificiels.
. La traduction automatique des langues naturelles.
. La composition et l'impression des textes.
. La documentation automatique.
. La vérification et démonstration de théorèmes mathématiques.
. La dérivation et intégration formelles.
. La gestion etc...

Le traitement non numérique des informations, qui en ALGOL s'appelle le traitement des chaînes, est ressenti absent dans ce langage, beaucoup d'auteurs ont proposé diverses solutions à ce problème. Les uns conçoivent leur proposition par une extension du langage en introduisant des nouveaux symboles, de nouvelles catégories syntaxiques, ce qui modifiera notablement le langage, donc demande en général une compilation complètement modifiée à moins que le compilateur déjà existant soit à syntaxe dirigée (syntax directed). D'autres proposent simplement une généralisation sémantique du langage, aucun symbole nouveau n'est introduit, le compilateur existant reste comme il est ; on adjoint au langage des concepts sémantiques nouveaux par la construction d'un ensemble de procédures dont le corps serait éventuellement en code.

Dans les pages qui suivent nous allons exposer une étude de ces diverses propositions et leurs applications au compilateur ALGOL de Grenoble sur IBM 7044. Notre travail contient trois parties :

. Etude critique des concepts d'entrée-sortie ;

. Etude critique du traitement des chaînes en ALGOL ;

. Définition des procédures de liste en ALGOL, cette dernière partie a été élaborée avec la collaboration bienveillante de Monsieur J. COHEN.

°°°

I. ETUDE CRITIQUE DES CONCEPTS D'ENTREE-SORTIE.

Au moment où ALGOL est défini, les problèmes d'entrée-sortie apparaissaient aux yeux des auteurs du langage comme dépendants du matériel utilisé. Aucune opération d'entrée-sortie n'est décrite ni mentionnée dans le rapport de base.

Depuis lors, de nombreux compilateurs sont construits, et chaque compilateur offre des facilités d'entrée-sortie différentes. Il en résulte des difficultés dans la communication des programmes entre installations. Dans le but de remédier à cet état de fait, différentes propositions sont conçues. Nous allons exposer la solution adoptée pour le compilateur ALGOL de Grenoble puis les propositions de H. Bekic, I. F. I. P. et A. C. M. ; ensuite nous passerons à une description détaillée des applications au compilateur ALGOL de GRENOBLE.

I. 1. PROCEDURES D'ENTREE-SORTIE DU SYSTEME ALGOL DE GRENOBLE.

Nous allons exposer ici les facilités d'entrée-sortie du système ALGOL de GRENOBLE (sur IBM 7044). Elles se présentent sous forme de procédures standard et se divisent en deux groupes :

. Procédures simples ;
. Procédures étendues.

I. 1. 1. Procédures simples.

C'est un ensemble de procédures simples d'entrée-sortie, à format fixe, étudiées pour la mise au point des programmes communiquant seulement avec les unités ordinaires d'entrée et de sortie (entrée par carte, sortie par imprimante).

I. 1. 1. 1. Procédures d'entrée immédiate.

Trois indicateurs de fonction sans paramètre R DONNEE, E DONNEE et B DONNEE, de type respectif réel, entier et booléen, permettent de lire une valeur sur l'unitée d'entrée et d'employer immédiatement cette valeur dans une instruction du programme, de la même

façon qu'un nombre pur ou une valeur logique apparaissant explicitement dans le programme.

Chaque valeur lue doit se présenter sur les cartes données sous la forme définie en ALGOL, en tenant compte de la représentation machine, et conformément au type de la procédure employée.

Exemple :

ALGOL	représentation machine	
$0.\ 314_{10} + 1$	0. 314 : ✱ + 1	pour R DONNEE
$- 314_{10}{}^{2}$	- 314 : ✱ 2	pour E DONNEE
vrai	'TRUE' ou 'VRAI' ou 'V',ou '1'	pour B DONNEE

Sur les cartes données chaque valeur lue doit être séparée du précédent et du suivant par un symbole ";" (::) Les blancs sont ignorés et un point virgule doit obligatoirement terminer chaque carte (qui peut-être perforée sur 72 colonnes).

Une fois une carte épuisée, la première valeur lue par une procédure d'entrée immédiate est la première figurant sur la carte suivante.

Exemple :

Soient les cartes données suivantes :

1) 2 :: 'v' :: 10.0 :: -.1.2 ::
2) 3. 14 :✱-1 :: 10 :✱ 3 ::

le programme

```
début réel  B  ;  entier  A  ;  booléen  R  ;
        A :=  E  DONNEE
        si    B  DONNEE ⊃ (( A + R DONNEE) ≤  R DONNEE)
                      alors  B :=  SIN  (R DONNEE)
                      sinon  R :=  R DONNEE  >  100
fin ;
```

équivaut à

```
début réel  B  ;  entier  A  ;  booléen  R ;
```

```
A := 2 ;
si vrai ⊃ ((A + 10. 0) ≤ -1.2)
        alors B := SIN (3. 14₁₀ - 1)
        sinon R := 10 :* 3
fin ;
```

I. 1. 1. 2. Procédures élémentaires.

1. LIRE

Cette procédure commande la lecture d'une carte à chaque appel. La carte peut comporter plusieurs données séparées les unes des autres par un ";" (::) . Le nombre de paramètres effectifs ne doit pas dépasser le nombre de données sur une carte.

Ces paramètres sont des variables déclarées dans le programme, et auxquelles seront affectées dans l'ordre les valeurs lues. Si le nombre de paramètres est inférieur à celui de données sur la carte, les dernières données sont ignorées.

Si la liste des paramètres comporte des éléments vides, les données correspondant aux emplacements de ces éléments sont ignorées.

Exemple :

Etant donnée une carte perforée comme suit :

```
+ 314 . 16 :* -2 :: 31 :: 10 :: -3 :: 0.5 :: 'VRAI'::'FAUX'
                                                ::34::
```

le programme suivant :

```
début réel A, B, E ; entier C, D ; booléen G ;
        LIRE (A, B, C, D, E,, G)
fin ;
```

affecte à A la valeur réelle 3. 14 16, à B l'entier 31, à C l'entier 10, à D l'entier -3, à E le réel 0.5, à G la valeur logique faux.

La valeur logique vrai et l'entier 34 sur la carte sont ignorés.

2. ECRIRE.

Chaque appel de cette procédure produit l'impression d'une

ligne. Le nombre de positions utilisées pour imprimer chaque donnée est fixe (15) et le nombre de données par ligne également (8).

Les paramètres (au nombre de huit au maximum) peuvent être des expressions arithmétiques ou booléennes quelconques ou des chaînes au sens ALGOL (non imbriquées). Chacun d'eux constitue une donnée dont le type dans le programme source fixe la représentation à l'impression.

Les valeurs réelles sont imprimées sous forme normalisée, c'est-à-dire avec mantisse entre 0 . 1 et 1, et un exposant comportant un signe et deux chiffres.

Les entiers sont cadrés à droite dans les zones de 15 caractères, les zéros de gauche étant remplacés par des espaces.

Les nombres négatifs sont précédés du signe - ; les positifs d'un espace.

Les valeurs logiques sont imprimées VRAI ou FAUX cadrées à droite dans les zones.

Enfin les chaînes sont imprimées cadrées à gauche. Si une chaîne comporte plus de quinze caractères, les caractères de droite sont supprimées ; si elle comporte moins, des espaces sont ajoutés à droite.

Si on désire avoir une zone entière de 15 blancs, on mettra dans la liste, à l'emplacement correspondant, un élément vide.

Exemple :

Le programme suivant :

```
début réel  X  ;  entier  N  ;
     N := 1  ;  X := 13 , 2  ;  ECRIRE  (,N>0,,'X␣=',X)
fin ;
```

provoque l'impression de la ligne suivante

␣...␣␣...VRAI␣... X␣=␣...␣␣␣.13200000+02␣␣...

15 11 15 12

espaces

1ère 2ème 3 4 5 6,7,8ème zone

I. 1. 1. 3. SAUT CARTE, SAUT LIGNE, SAUT PAGE.

Ces trois procédures sans paramètre, permettent respectivement :

- D'ignorer une carte à l'entrée,
- D'imprimer une ligne blanche,
- De sauter à la page suivante.

I. 1. 2. Procédures étendues.

Ces procédures permettent d'effectuer des transferts quelconques d'information avec toutes les unités d'entrée-sortie disponibles sur la machine. Le modèle des transferts d'information est fixé par l'utilisateur grâce à un ordre MODELE du type FORTRAN IV, décrite ici sous forme de carte syntaxique qui sera présentée en fin de chapitre.

Ces procédures sont ENTREE et SORTIE.

Elles sont appelées avec une liste de paramètres en nombre quelconque, construite de la façon suivante :

Premier paramètre : Une expression arithmétique de type entier fixant le canal d'entrée-sortie.

Deuxième paramètre : Une étiquette simple du programme A l'appel de la procédure, une instruction ALLER A est simulée vers cette étiquette, et toutes les instructions du programme sont exécutées à partir de cette étiquette jusqu'au premier ordre MODELE rencontré. Celui-ci fixe alors le modèle de traitement des données de l'appel.

Liste des données : La liste des données est traitée par couples. Le premier paramètre de chaque couple doit être une expression arithmétique de type entier qui donne le nombre de variables à traiter. Le deuxième est une expression arithmétique de type entier qui donne le nombre de variables à traiter. Le deuxième est une expression arithmétique ou booléenne qui fournit l'adresse de la première variable à traiter pour chaque couple. L'ordre des variables traitées pour une * variation des différents indices de la variable, en commençant par le dernier. Dans le cas particulier d'une variable à deux indices,

---

*variable indicée est celui correspondant à une

ceci correspond à un traitement des tableaux par ligne.

Exemples :

1. Les nombres A, B, C définissant les valeurs :

+ 3. 1416, + 2. 72 et -0. 5 ,

les instructions SORTIE (6*, M1, 1, A, 1, B, 1, C) ;
M1 : MODELE (' (3 E 12 . 4)') ;

commandent l'impression sur l'unité normale de sortie de la ligne :

␣␣0. 3142E␣01	␣␣0. 2720E␣01	␣ - 0.5000E␣␣0
A	B	C

2. Si l'on veut lire sur 16 cartes les 16 éléments d'une matrice (4, 4) on écrira :

début tableau A [1; 4, 1 : 4] ;
ME : MODELE (' (E 10.4)') ;
ENTREE (5* , ME, 16, A [1, 1]) ;
fin ;

A coté de cela, il y a aussi des procédures de servitude qui sont :

REBOBINER : Avec une liste de paramètres effectifs qui doivent être des expressions arithmétiques de type entier. Les bandes magnétiques dont les numéros logiques sont les valeurs de ces expressions sont rebobinées.

ECRIRE FDB , même liste de paramètres pour écrire des marques de fin de fichier sur les bandes correspondantes.

ESP ARRIERE : Les paramètres sont pris par couples d'expressions arithmétiques du type entier. Le premier paramètre du couple indique le numéro logique de la bande considérée ; le deuxième paramètre donne le nombre d'espaces arrières à effectuer sur cette bande. Toutefois si ESP ARRIERE est appelé avec un seul paramètre, le deuxième est supposé égal à 1.

---

*Les numéros 5, 6, 7 correspondent aux unités ordinaires : d'entrée, de sortie et de perforation .

PAUSE (N) L'appel de cette procédure provoque l'arrêt de la machine et l'impression au pupitre en code octal de la valeur du paramètre N.

CONTROLE Cette procédure permet de tenir compte des possibilités de points de reprise (check point) offertes par le système et étudiées par chaque installation.

HEURE Cette procédure provoque l'appel du programme de comptabilité de l'installation pour imprimer sur l'unité 6 l'heure du moment de l'appel (après vidage de toutes les zones tampon d'entrée-sortie).

TEMP Indicateur de fonction de type entier égal, si l'installation possède l'horloge interne, au temps passé depuis le début de l'exécution du programme évalué en dixièmes de seconde.

DISCUSSION

Examinons tout d'abord les procédures étendues. Le modèle utilisé est celui de FORTRAN IV. Il fixe exactement la configuration d'une ou plusieurs lignes à imprimer ou d'une ou plusieurs cartes à lire. Il détermine la longueur en caractères de chaque donnée, sa représentation extérieure et intérieure, sa position sur la page. etc..

En ce qui concerne l'instruction d'entrée-sortie, on constate quelques améliorations réalisées dans le système ALGOL de Grenoble par rapport au FORTRAN IV :

. On peut spécifier un modèle variable dans le programme même tandis qu'en FORTRAN IV, le modèle variable doit être lu sur cartes données et n'est pas listé.

. La gestion des tableaux est plus naturelle, elle se fait par lignes au lieu de par colonnes.

. Pour spécifier le nombre d'éléments de tableaux à entrer/sortir on peut utiliser une expression arithmétique variable au lieu d'un entier fixe.

Exemple

B étant une variable booléenne

A un réel tableau

si au cours du programme on désire écrire les 8 premiers éléments de

A avec le modèle E 12.4 si B est vrai, et le message " LA␣SERIE␣N␣EST␣PAS␣CONVERGENTE" si B est faux, le programme suivant constitue une solution au problème;

```
M 1 : allera si B alors M 2 sinon M 3
M 2 : MODELE ( ' (8 E 12.4)') ;
M 3 : MODELE ( ' (30HLA␣SERIE␣N␣EST␣PAS␣CONVERGENTE)') ;
SORTIE (6, M 1, si B alors 8 sinon 0, A|1|) ;
```

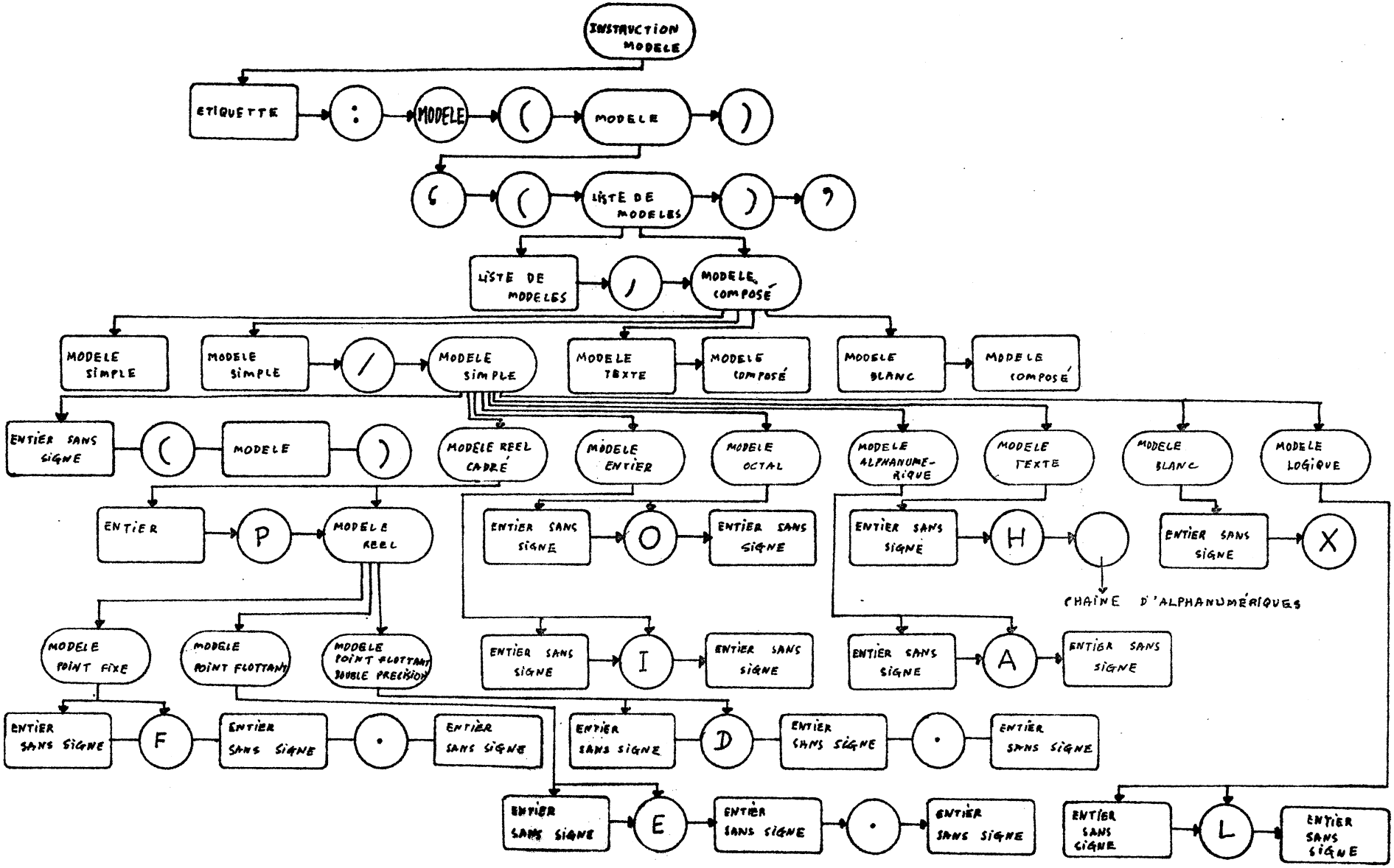
On voit donc que les procédures étendues constituent un outil très puissant pour résoudre des problèmes d'entrée-sortie. Mais son utilisation nécessite un certain apprentissage. Pour les programmeurs non avertis, le système ALGOL de GRENOBLE offre les procédures simples, à format fixe, assez rigides mais extrèmement simples à utiliser.

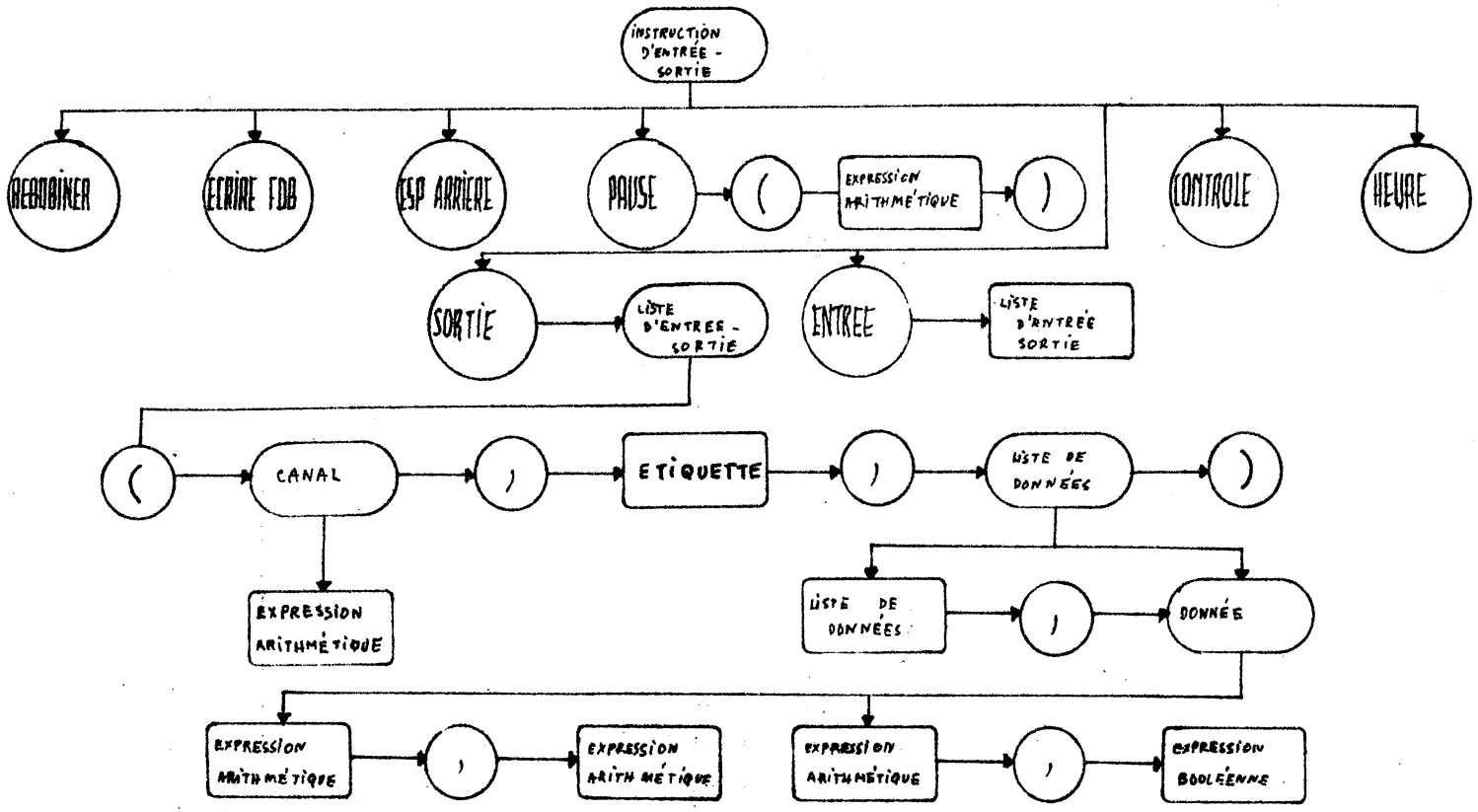
Pour conclure on peut dire que l'ensemble des procédures d'entrée-sortie du système ALGOL de GRENOBLE, constitue une solution satisfaisante au problème d'entrée-sortie de ALGOL. Les seuls défauts qu'on trouve sur ces procédures sont :

. L'imposibilité de manipuler deux quantités sur un enregistrement par 2 commandes successives (exemples: écrire deux nombres sur une même ligne par deux instructions successives), ce qui ne facilite point une mise en page programmée.

. L'impossibilité de mettre des commentaires au milieu d'un nombre, ce qui ne permet pas d'écrire des libellés du genre .
15 FRANCS 75 CENTIMES.

ET voici les cartes syntaxiques des modèles et des instructions d'entrée-sortie (concernant seulement les procédures étendues) du système ALGOL de GRENOBLE.


INSTRUCTION MODELE
ETIQUETTE
:
MODELE
(
MODELE
)
G
(
LISTE DE MODELES
)
?
LISTE DE MODELES
/
MODELE COMPOSÉ
MODELE SIMPLE
MODELE SIMPLE
/
MODELE SIMPLE
MODELE TEXTE
MODELE COMPOSÉ
MODELE BLANC
MODELE COMPOSÉ
ENTIER SANS SIGNE
(
MODELE
)
MODELE REEL CADRÉ
MODELE ENTIER
MODELE OCTAL
MODELE ALPHANUMERIQUE
MODELE TEXTE
MODELE BLANC
MODELE LOGIQUE
ENTIER
P
MODELE REEL
ENTIER SANS SIGNE
O
ENTIER SANS SIGNE
ENTIER SANS SIGNE
H
CHAINE D'ALPHANUMERIQUES
ENTIER SANS SIGNE
X
MODELE POINT FIXE
MODELE POINT FLOTTANT
MODELE POINT FLOTTANT DOUBLE PRECISION
ENTIER SANS SIGNE
I
ENTIER SANS SIGNE
ENTIER SANS SIGNE
A
ENTIER SANS SIGNE
ENTIER SANS SIGNE
F
ENTIER SANS SIGNE
.
ENTIER SANS SIGNE
ENTIER SANS SIGNE
D
ENTIER SANS SIGNE
.
ENTIER SANS SIGNE
ENTIER SANS SIGNE
E
ENTIER SANS SIGNE
.
ENTIER SANS SIGNE
ENTIER SANS SIGNE
L
ENTIER SANS SIGNE


INSTRUCTION D'ENTRÉE - SORTIE
REBOBINER
ECRIRE FDB
ESP ARRIÈRE
PAUSE
(
EXPRESSION ARITHMÉTIQUE
)
CONTROLE
HEURE
SORTIE
LISTE D'ENTREE - SORTIE
ENTREE
LISTE D'ENTRÉE SORTIE
(
CANAL
)
ETIQUETTE
)
LISTE DE DONNÉES
)
EXPRESSION ARITHMÉTIQUE
LISTE DE DONNÉES
)
DONNÉE
EXPRESSION ARITHMÉTIQUE
)
EXPRESSION ARITHMÉTIQUE
EXPRESSION ARITHMÉTIQUE
)
EXPRESSION BOOLÉENNE

I. 2. PROPOSITION DE H. BEKIC

H. Bekic dans "Une proposition d'entrée-sortie pour ALGOL basée sur l'entrée-sortie de FORTRAN IV" (Référence 49) a suggéré l'adoption pour ALGOL des processus d'entrée-sortie de FORTRAN IV avec les modifications suivantes :

. Le concept de modèles répétés sera unifié pour être applicable de la même manière sur les modèles simples et aux listes de modèles. Les primaires arithmétiques sont admises pour spécifier le nombre de fois qu'il faut répéter un modèle. Ce qui veut dire qu'un duplicateur de modèles peut avoir la valeur de toute expression arithmétique (mise entre parenthèses).

. Par l'introduction d'un nouveau type de variable qui est la variable de chaîne, et de la possibilité de lui affecter des expressions de chaîne, au milieu d'un modèle il est permis d'avoir une insertion variable :

. Les modèles et données sont déclarés à l'aide de deux nouveaux symboles FORMAT LIST et ITEM LIST

Fig. 1 et Fig. 2

représentent la définition syntaxique de la proposition de H. Bekic

DISCUSSION

Cette proposition est intéressante dans le sens qu'elle donne une définition concise des processus d'entrée-sortie, surtout en ce qui concerne la liaison entre la liste des données et la liste des modèles suivant lesquels les données doivent être traduites. Par contre elle comporte des défauts de FORTRAN IV déjà mentionnés.

Mais la cause principale qui rend difficile l'adoption de cette proposition réside dans le fait que l'auteur a introduit des nouveaux symboles, des quantités nouvelles (expression de chaîne) qui nécessitent une modification des compilateurs existants. Or cela n'est pas souhaitable dans l'état actuel. En ce qui concerne plus précisément l'introduction en ALGOL des expressions de chaîne, la proposition n'est pas conséquente. Il n'y a aucune idée précise sur la manière de manipuler les chaînes, aucune opération sur les chaînes n'est définie. Il n'est pas logique d'introduire un type nouveau (chaîne) dans le seul but de faciliter les processus d'entrée-sortie.

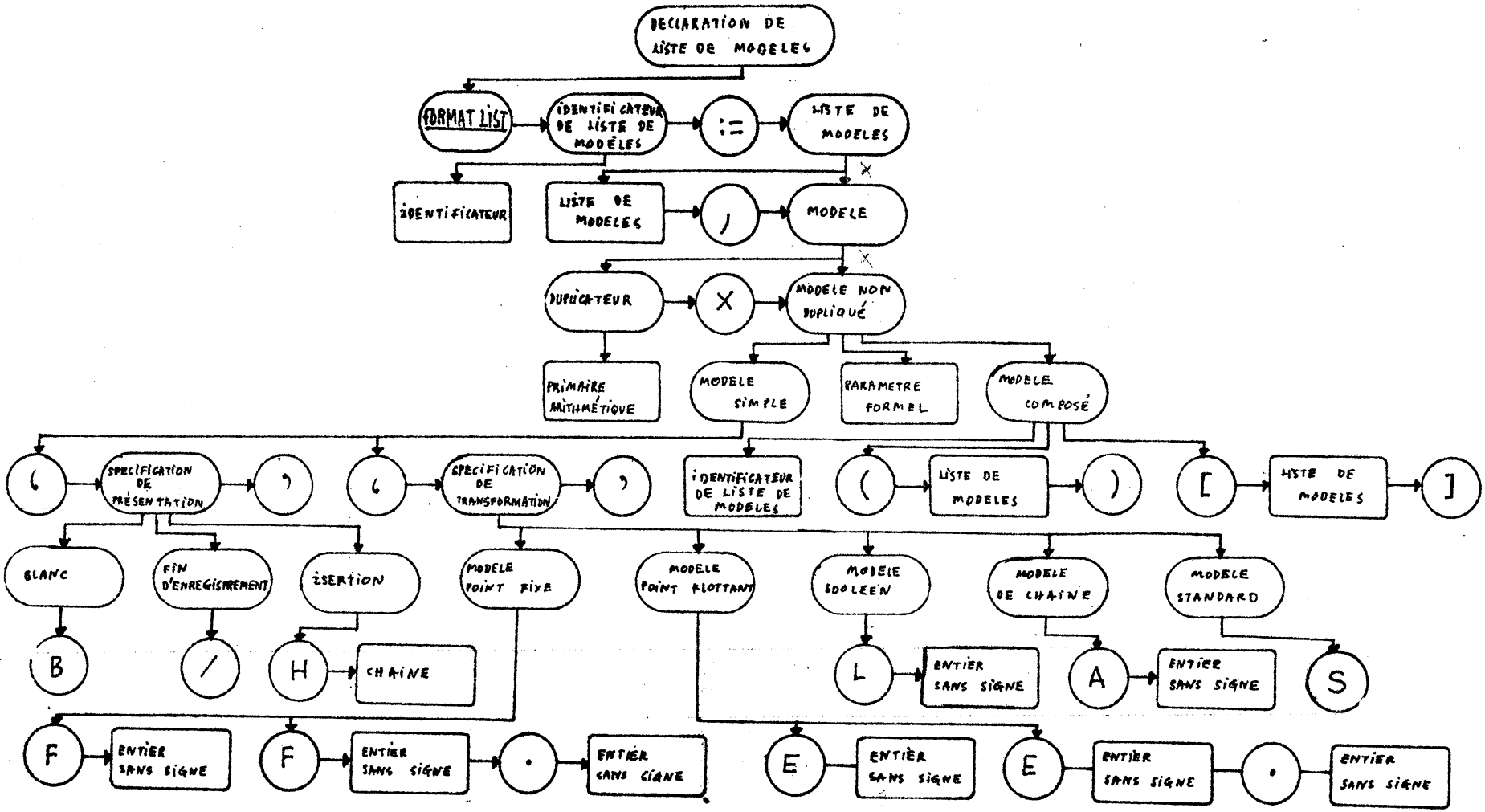
I. 3. PROPOSITION D'IFIP

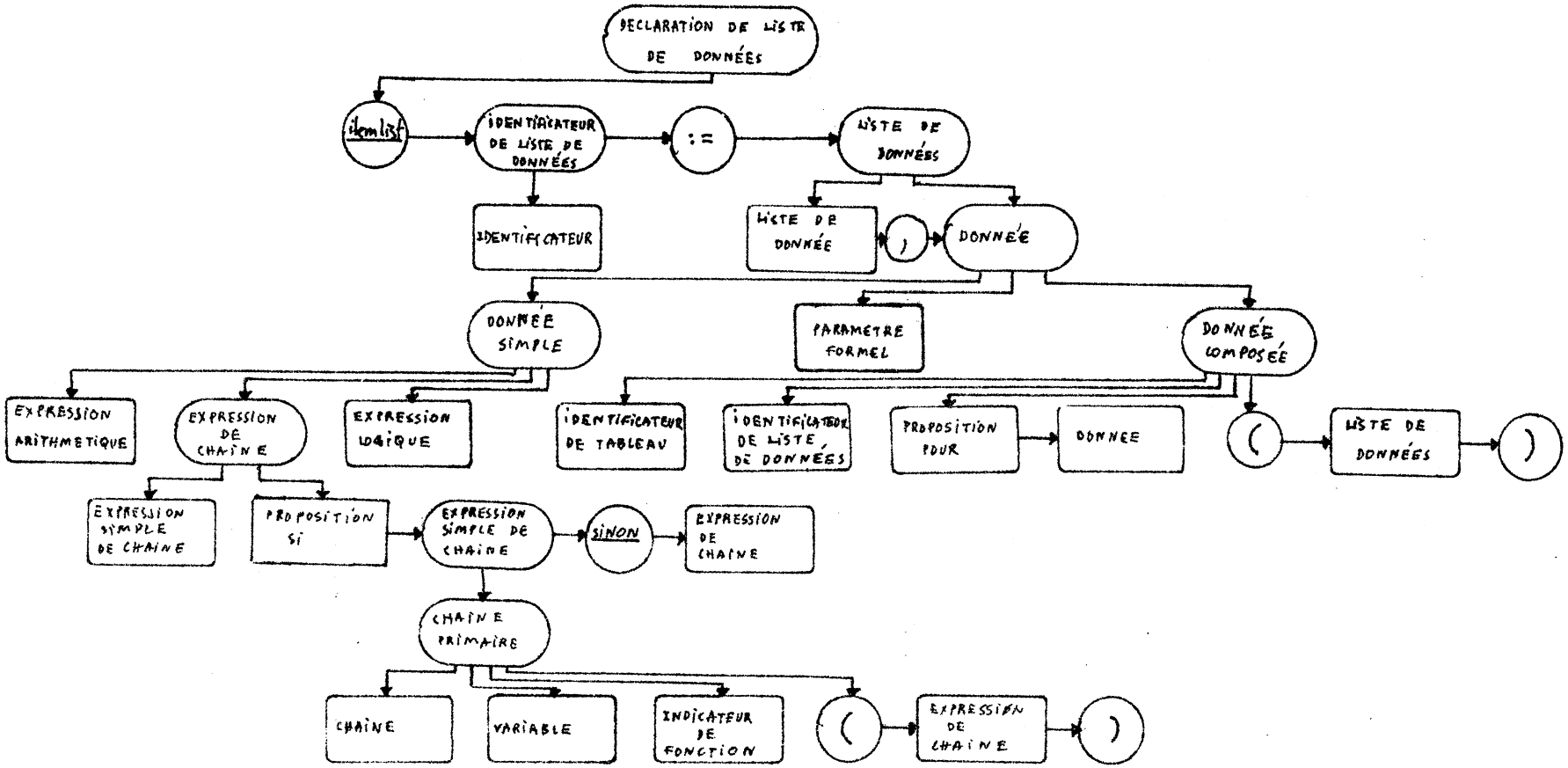
Chaque compilateur ALGOL met à la disposition des utilisateurs des facilités d'entrée-sortie différentes, cela rend difficile la communication des programmes entre installations et même dans une installation, un programme ALGOL écrit pour une machine ne peut-être exécuté par une autre machine. Pour parer à cette difficulté, le groupe de travail W G 2. 1 sur ALGOL d'IFIP a proposé un ensemble de procédures d'entrée-sortie qui sont exprimées en code, dites procédures de base . Voici la liste de ces procédures :

INSYMBOL
OUTSYMBOL
LENGTH
INREAL
OUTREAL
INARRAY
OUTARRAY

Elles se déclarent comme suit :

```
procedure   INSYMBOL (CHANNEL, STRING, DESTINATION) ;
value       CHANNEL ; integer CHANNEL, DESTINATION  ;
string      STRING ;  < procedure body > ;
```


DECLARATION DE LISTE DE MODELES
FORMAT LIST
IDENTIFICATEUR DE LISTE DE MODELES
:=
LISTE DE MODELES
IDENTIFICATEUR
LISTE DE MODELES
)
MODELE
DUPLICATEUR
X
MODELE NON DUPLIQUÉ
PRIMAIRE ARITHMÉTIQUE
MODELE SIMPLE
PARAMETRE FORMEL
MODELE COMPOSÉ
(
SPECIFICATION DE PRESENTATION
)
(
SPECIFICATION DE TRANSFORMATION
)
IDENTIFICATEUR DE LISTE DE MODELES
(
LISTE DE MODELES
)
[
LISTE DE MODELES
]
BLANC
FIN D'ENREGISTREMENT
INSERTION
MODELE POINT FIXE
MODELE POINT FLOTTANT
MODELE BOOLEEN
MODELE DE CHAINE
MODELE STANDARD
B
/
H
CHAINE
L
ENTIER SANS SIGNE
A
ENTIER SANS SIGNE
S
F
ENTIER SANS SIGNE
F
ENTIER SANS SIGNE
.
ENTIER SANS SIGNE
E
ENTIER SANS SIGNE
E
ENTIER SANS SIGNE
.
ENTIER SANS SIGNE


DECLARATION DE LISTE DE DONNÉES
idem list
IDENTIFICATEUR DE LISTE DE DONNÉES
:=
LISTE DE DONNÉES
IDENTIFICATEUR
LISTE DE DONNÉE
)
DONNÉE
DONNÉE SIMPLE
PARAMETRE FORMEL
DONNÉE COMPOSÉE
EXPRESSION ARITHMETIQUE
EXPRESSION DE CHAINE
EXPRESSION LOGIQUE
IDENTIFICATEUR DE TABLEAU
IDENTIFICATEUR DE LISTE DE DONNÉES
PROPOSITION POUR
DONNEE
(
LISTE DE DONNÉES
)
EXPRESSION SIMPLE DE CHAINE
PROPOSITION SI
EXPRESSION SIMPLE DE CHAINE
SINON
EXPRESSION DE CHAINE
CHAINE PRIMAIRE
CHAINE
VARIABLE
INDICATEUR DE FONCTION
(
EXPRESSION DE CHAINE
)

```
procedure   OUT SYMBOL (CHANNEL, STRING, SOURCE) ;
value   CHANNEL, SOURCE; integer  CHANNEL, SOURCE ;
string   STRING ; <procedure body> ;

procedure   LENGTH (STRING) ; string STRING ;
<procedure body> ;

procedure   INREAL (CHANNEL, DESTINATION) ; value CHANNEL ;
integer   CHANNEL ; real DESTINATION ; <procedure body> ;

procedure   OUT REAL (CHANNEL, SOURCE) ;
value   CHANNEL, SOURCE ; integer CHANNEL ; real SOURCE ;
<procedure body> ;

procedure   INARRAY (CHANNEL, DESTINATION) ; value CHANNEL ;
integer   CHANNEL ; array DESTINATION ; <procedure body> ;

procedure   OUT ARRAY (CHANNEL, SOURCE) ; value CHANNEL ;
integer   CHANNEL ; array SOURCE ;
<procedure body> ;
```

Dans les appels de ces procédures, à l'exception de LENGTH, la valeur du premier paramètre effectif (correspondant à CHANNEL) doit être un nombre entier positif précisant le canal d'entrée-sortie valable pour le programme.

ACTION DE CES PROCEDURES :

Les procédures INSYMBOL et OUTSYMBOL fournissent les moyens de communication entre les supports extérieurs et les variables du programme en terme de symboles de base simples ou tout autre symbole supplémentaire. La correspondance entre les symboles de base et les valeurs des variables dans le programme est établie en accordant la séquence des symboles de base de la chaîne donnée en second paramètre, pris de gauche à droite, avec les entiers positifs 1, 2, 3, ...

Ainsi la procédure INSYMBOL affecte à la variable (de type entier) donnée comme troisième paramètre la valeur correspondant au prochain symbole de base apparu sur le support extérieur. Si ce symbole n'est pas dans la liste des symboles de la chaîne qui définit le second paramètre, zéro sera affecté. Si ce symbole n'est pas un symbole de base ALGOL 60, une valeur négative, correspondant

au symbole, sera affectée.

Il en est de même pour la procédure OUT SYMBOL. Si la valeur du troisième paramètre est négative, un symbole correspondant à cette valeur sera transmis. Il est recommandé d'utiliser la même convention pour les symboles supplémentaires communs aux deux procédures précitées.

La procédure LENGTH sert à calculer la longueur des chaînes.

La valeur de LENGTH (c) est égale au nombre de symboles de base de la chaîne ouverte comprise entre les crochets de chaîne les plus extérieurs.

La procédure INREAL affecte la prochaine valeur apparue sur le support extérieur à la variable de type réel donnée en second paramètre. La procédure OUTREAL fait l'opération inverse.

Pour les procédures INARRAY, OUTARRAY, on utilise la convention:

A [K1, K2, ... , Km] précède A [J1, J2, ..., Jm]
si Ki = Ji (i = 1, 2, ... , p-1) $1 \leq p \leq m$
Kp < Jp

La représentation des nombres sur le support extérieur est la même pour les quatre procédures INREAL, OUTREAL, INARRAY, OUTARRAY.

DISCUSSION :

Ce rapport introduit des facilités de manipulation de chaînes car elle permet par l'intermédiaire de INSYMBOL de transformer une chaîne de symboles quelconques en un ensemble de nombres entiers, or les opérations sur les entiers sont très bien définies en ALGOL. On voit donc par là de grandes possibilités pour le traitement des chaînes :

Pour faire un problème de manipulation de symboles on transforme la chaîne donnée en un ensemble d'entiers par INSYMBOL, on applique des opérations convenables sur ces derniers ; une fois finies les opérations, on fait la transformation entier —> symbole par OUTSYMBOL pour avoir le résultat final sur un support extérieur.

A part ce côté très positif du rapport qui est d'ailleurs réalisé sous le nom de procédures de base pour "l'entrée-sortie" il y a des points imprécis. Les procédures INREAL, OUTREAL, IN ARRAY, OUT ARRAY ne sont pas primitives puisqu'elles peuvent être construites à partir des trois procédures INSYMBOL, OUT SYMBOL, LENGTH (pour les exemples voir I. 5. 2.). Ensuite elles se rapportent à des quantités réelles, or la forme des réels sur les supports extérieurs n'est pas précisée. Donc tout indique qu'il faille réduire le nombre des procédures de base à trois et adopter une convention sur la manière d'entrer ou de sortir des réels par exemple au moyen d'un modèle avant de construire INREAL, OUT REAL à partir de INSYMBOL, OUT SYMBOL, LENGTH. Quant à IN ARRAY, OUT ARRAY leur action peut-être effectuée par une simple boucle de pour.

Exemple : INREAL étant supposé déjà construit le programme suivant remplit le tableau A déclaré dans le bloc le plus extérieur.

```
begin array   A [1 : n] ;
procédure  INREAL (CHANNEL, SOURCE) ; value CHANNEL, SOURCE ;
integer   CHANNEL ; real SOURCE ; < procedure body > ;
integer I ;
for   I := 1 step 1   until   n   do
      INREAL   (m, A [1])
end ;
```

m et n étant des entiers convenablement choisis.

I. 4 PROPOSITION D'A. C. M.

Dans le rapport de base d'ALGOL, les différentes quantités telles que entier, réel, booléen sont très bien définies en ce qui concerne leur forme dans le programme. Mais il manque totalement la description de ce qui doit être fait pendant la transmission des quantités du programme ALGOL vers l'extérieur ou vice versa.

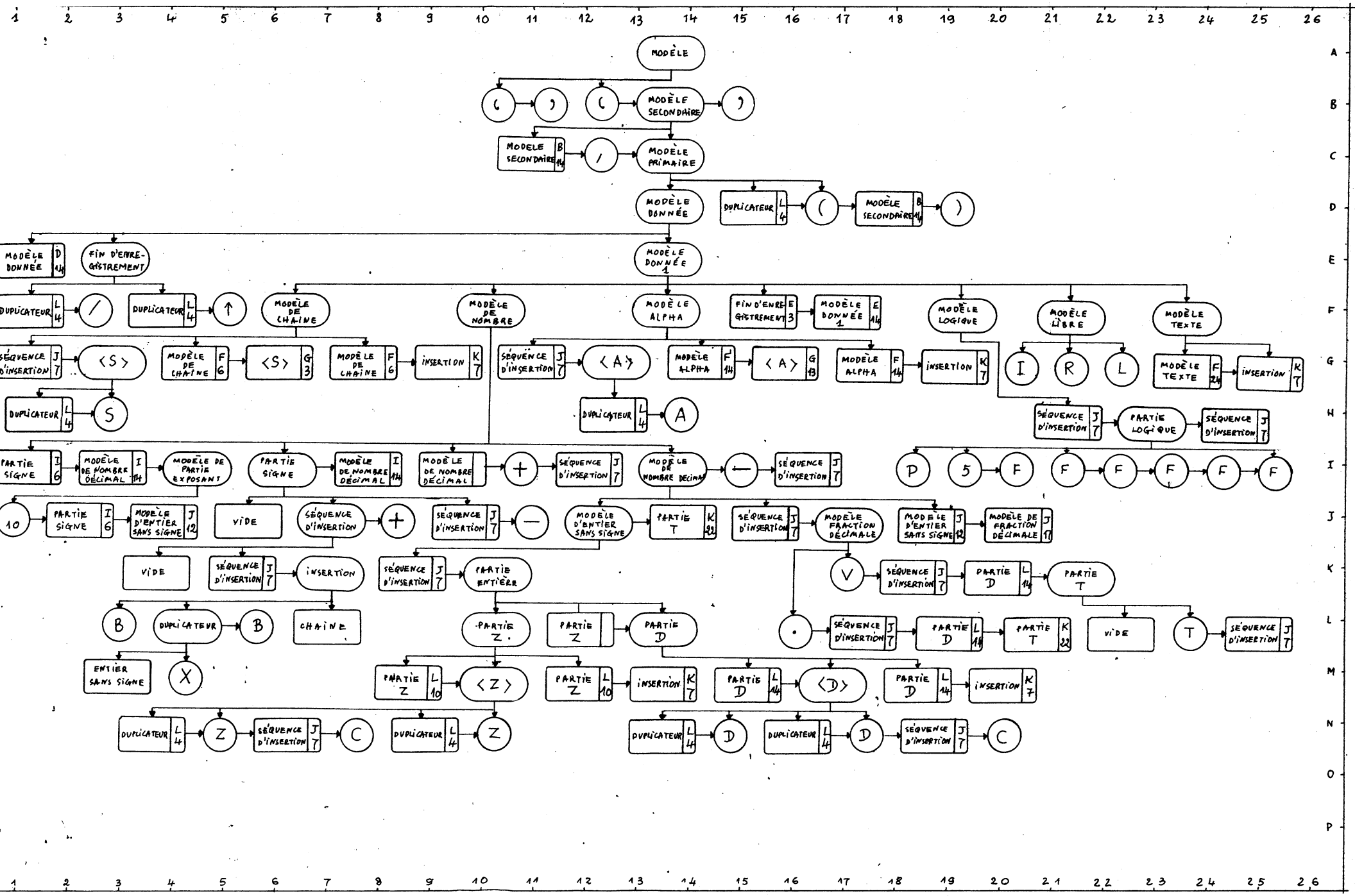
Ceci résulte du fait qu'au moment où ALGOL a été défini, les processus d'entrée-sortie apparaissaient aux yeux des auteurs du langage comme trop dépendants des machines. Depuis lors, de nombreux compilateurs ALGOL ont été réalisés et chaque compilateur offre quelques facilités d'entrée-sortie. L'expérience a montré que

de telles facilités peuvent être incorporées à ALGOL d'une façon presque complètement indépendante des machines.

Se basant sur ce fait, un sous-comité d'A. C. M. a proposé "une convention d'entrée-sortie pour ALGOL 60" (Référence) dont voici les traits caractéristiques :

Elle ne nécessite aucune extension du langage.

La façon d'interpréter les représentations de quantités sur supports extérieurs est spécifié au moyen d'une chaîne "modèle".


MODÈLE
(
)
(
MODÈLE SECONDAIRE
)
MODÈLE SECONDAIRE B 14
/
MODÈLE PRIMAIRE
MODÈLE DONNÉE
DUPLICATEUR L 4
(
MODÈLE SECONDAIRE B 14
)
MODÈLE DONNÉE D 14
FIN D'ENREGISTREMENT
MODÈLE DONNÉE 1
DUPLICATEUR L 4
/
DUPLICATEUR L 4
↑
MODÈLE DE CHAINE
MODÈLE DE NOMBRE
MODÈLE ALPHA
FIN D'ENREGISTREMENT E 3
MODÈLE DONNÉE 1 E 14
MODÈLE LOGIQUE
MODÈLE LIBRE
MODÈLE TEXTE
SÉQUENCE D'INSERTION J 7
<S>
MODÈLE DE CHAINE F 6
<S> G 3
MODÈLE DE CHAINE F 6
INSERTION K 7
DUPLICATEUR L 4
S
SÉQUENCE D'INSERTION J 7
<A>
MODÈLE ALPHA F 14
<A> G 13
MODÈLE ALPHA F 14
INSERTION K 7
DUPLICATEUR L 4
A
I
R
L
MODÈLE TEXTE F 24
INSERTION K 7
SÉQUENCE D'INSERTION J 7
PARTIE LOGIQUE
SÉQUENCE D'INSERTION J 7
P
5
F
F
F
F
F
F
PARTIE SIGNE I 6
MODÈLE DE NOMBRE DÉCIMAL I 14
MODÈLE DE PARTIE EXPOSANT
PARTIE SIGNE
MODÈLE DE NOMBRE DÉCIMAL I 14
MODÈLE DE NOMBRE DÉCIMAL
+
SÉQUENCE D'INSERTION J 7
MODÈLE DE NOMBRE DÉCIMAL
—
SÉQUENCE D'INSERTION J 7
10
PARTIE SIGNE I 6
MODÈLE D'ENTIER SANS SIGNE J 12
VIDE
SÉQUENCE D'INSERTION
+
SÉQUENCE D'INSERTION J 7
—
MODÈLE D'ENTIER SANS SIGNE
PARTIE T K 22
SÉQUENCE D'INSERTION J 7
MODÈLE FRACTION DÉCIMALE
MODÈLE D'ENTIER SANS SIGNE J 12
MODÈLE DE FRACTION DÉCIMALE J 11
VIDE
SÉQUENCE D'INSERTION J 7
INSERTION
SÉQUENCE D'INSERTION J 7
PARTIE ENTIÈRE
V
SÉQUENCE D'INSERTION J 7
PARTIE D L 14
PARTIE T
B
DUPLICATEUR
B
CHAINE
PARTIE Z
PARTIE Z
PARTIE D
.
SÉQUENCE D'INSERTION J 7
PARTIE D L 14
PARTIE T K 22
VIDE
T
SÉQUENCE D'INSERTION J 7
ENTIER SANS SIGNE
X
PARTIE Z L 10
<Z>
PARTIE Z L 10
INSERTION K 7
PARTIE D L 14
<D>
PARTIE D L 14
INSERTION K 7
DUPLICATEUR L 4
Z
SÉQUENCE D'INSERTION J 7
C
DUPLICATEUR L 4
Z
DUPLICATEUR L 4
D
DUPLICATEUR L 4
D
SÉQUENCE D'INSERTION J 7
C

Cette chaîne "modèle" doit obéir à une certaine syntaxe présentée ici sous forme de carte syntaxique.

En ce qui concerne la signification des méta-variables utilisées pour définir la syntaxe du modèle, voici quelques commentaires:

Il est à remarquer que nous nous occupons principalement de la sortie, l'entrée est tout à fait semblable et sera commentée en parallèle quand cela s'avère nécessaire.

Sommaire des symboles :

A	Caractère alphanumérique représenté en entier
B	blanc ou espace
C	virgule
D	chiffre
F	valeur logique vrai ou faux
I	entier non traduit
L	valeur logique non traduite
P	bit logique
R	réel non traduit
S	caractère de chaîne
T	troncature
V	point décimal sous-entendu
X	duplicateur variable
Z	suppression de zéro
+	imprimer le signe
-	imprimer le signe si le nombre est négatif
$10^{}$	indicateur de la partie exposant
( )	délimiteurs des modèles secondaires dupliqués
,	séparateur des modèles - donnée
/	saut ligne
↑	saut page
'	délimiteurs de chaînes insérées
.	point décimal

Duplicateur : Un entier sans signe n utilisé comme duplicateur signifie que la quantité est à répéter n fois ; ainsi 2D3B est équivalent à DDBBB. Le caractère X utilisé comme duplicateur

permet d'avoir un modèle variable, car sa valeur sera spécifiée seulement quand le programme est en cours d'exécution.

Insertion : Il est permis d'insérer des commentaires délimités par les crochets de chaîne partout à l'intérieur d'un modèle de nombre. Ces commentaires à la sortie seront imprimés à la même place par rapport au reste du nombre. De même B qui correspond à un blanc peut-être inséré partout à l'intérieur d'un modèle de nombre. Pour l'entrée : l'insertion sert à spécifier le nombre de positions de caractères à ignorer.

Suppression de signe, zéro et virgule : la partie d'un nombre à gauche du point décimal consiste en un signe facultatif, puis une séquence de Z ou D, avec éventuellement des C derrière un Z ou un D, plus des caractères d'insertion.

La convention sur les signes est la suivante :

. Si aucun signe n'apparait, le nombre est supposé positif, et le traitement des négatifs est indéfini.

. Si un signe plus apparaît, le signe apparaîtra si le nombre est négatif et sera supprimé dans le cas contraire.

La lettre Z correspond à une suppression de zéro, et la lettre D à une impression de chiffre sans suppression de zéro. Chaque Z ou D correspond à une seule position de caractère ; le chiffre zéro spécifié par Z sera supprimé, par exemple remplacé par un blanc quand tous les chiffres à sa gauche sont des zéros. Il est à remarquer qu'un nombre nul imprimé avec un modèle composé de Z uniquement apparaitra sur le support extérieur comme des blancs.

. La lettre C correspond à une virgule. Une virgule suivant un D sera toujours imprimée ; une virgule suivant un Z sera imprimée à moins qu'une suppression de zéro intervienne pour ce Z. Quand il y a une suppression de zéro ou de virgule, le signe éventuel est imprimé à la position du caractère le plus à droite supprimé.

Pour l'entrée : Aucune distinction n'est faite entre D et Z. L'insertion sert uniquement à spécifier le nombre de positions de caractères à ignorer. Si le modèle spécifie un signe à gauche, le signe peut apparaître à la position d'un quelconque Z, D ou C

aussi longtemps qu'il est à gauche du nombre. Un signe spécifié à droite doit apparaître en bonne place.

Point décimal : La position du point décimal est indiqué soit par le caractère "." , soit par la lettre V. Dans le premier cas, le point décimal apparaît sur le support extérieur, dans le deuxième cas il est sous entendu. Seuls les D (non les Z) peuvent apparaître à droite du point décimal.

Modèle de chaîne : Ce modèle est utilisé pour sortir des chaînes. Chaque S dans le modèle correspond à un caractère de la chaîne à transmettre vers l'extérieur. Si la longueur de la chaîne est plus grande que le nombre de S, les caractères à gauche seront transmis ; si la longueur de la chaîne est plus petite, des blancs seront ajoutés à droite de la chaîne.

Modèle alpha : Chaque lettre A indique qu'un caractère est à transmettre. Ceci est similaire au modèle de chaîne mais l'équivalent en ALGOL des alphanumériques est du type entier.

Modèle libre un I, R ou L est utilisé pour indiquer qu'une quantité entière, réelle ou logique est à transmettre en utilisant la représentation interne de la machine.

Modèle logique : quand les quantités booléennes sont à transmettre, les modèles P, F, 5F ou FFFFF doivent être utilisés. La correspondance se définit comme suit :

ALGOL	P	F	5F ou FFFFF
true	1	T	TRUE
false	0	F	FALSE

Modèle texte : pour les autres modèles il doit y avoir une correspondance entre une variable du programme ALGOL et un nombre de caractères à transmettre. Le modèle texte n'en a pas besoin. Il donne lieu, pour la sortie à l'émission des caractères d'insertion et pour l'entrée à des sauts de caractères.

DISCUSSION :

Cette proposition donne la définition vigoureuse d'un "modèle" susceptible de fournir à ALGOL un instrument puissant pour traiter des problèmes d'entrée-sortie.

. La possibilité de mettre des commentaires au milieu d'un nombre, utile pour les libel... 'e banque par exemple (5 DOLLARS 60 CENTS).

. L'utilisation des Z, D, C, V, T pour préciser exactement ce qu'on attend d'un modèle de nombre.

. L'emploi d'un duplicateur variable

. L'introduction d'une convention sur les signes (un signe peut apparaître à gauche ou à droite d'un nombre) constituent des nouveautés par rapport au modèle de FORTRAN IV. De plus, la proposition prévoit que des sorties soient exécutées à la suite sur le même enregistrement ; autrement dit, sur une imprimante, il est permis d'écrire plusieurs fois sur la même ligne ; ce qui facilite grandement une mise en page dynamique.

En ce qui concerne la liaison entre le modèle et la quantité à transmettre, elle se fait au moyen des procédures en code.

Exemple :

```
procedure   OUTPUT 1 (CANAL, MODELE, QUANTITE) ;
valeur      CANAL, QUANTITE 1 ; entier CANAL ; chaîne MODELE ;
            < corps de procédure > ;
procedure   OUTPUT 2 (CANAL, MODELE, QUANTITE 1, QUANTITE 2) ;
valeur      CANAL, QUANTITE 1, QUANTITE 2 : entier CANAL ;
chaîne      MODELE ; < corps de procédure > ;

procedure   OUTPUT n (CANAL, MODELE, ....., QUANTITE n) ;
```

. Le nombre n qui suit l'identificateur OUTPUT indique qu'il y a n quantités à transmettre.

. CANAL précise l'unité logique utilisée.

. QUANTITE , QUANTITE 2, ... sont des quantités à transmettre.

Ainsi si 6 indique l'imprimante et que N = 500, M = 0, X [0] = 18061579 et T = 3.14155926536 l'instruction

OUTPUT 4 (6, '2(BBB4ZD), 3B, +D.60_{10}+3D, 3B, -Z.DDDDBDDDD', N, M, X[M], COS(T))

donne sur le listage le résultat suivant :

500 0+1.806158_{10}+007 1.0000 0000

On voit par là qu'il faudrait construire beaucoup de procédures d'entrée-sortie. Surtout, pour de nombreux compilateurs ALGOL en état de fonctionnement qui demandent une spécification des paramètres

de procédure il faudrait construire un nombre formidable de procédures. En fait on peut transformer n en un paramètre interne et transformer OUTPUT en un procédure standard dont le nombre de paramètres effectifs serait variable.

Il apparaît donc qu'il est souhaitable d'adopter pour ALGOL le modèle d'entrée-sortie décrit ci-dessus. Mais pour ce qui est de la liaison entre la quantité à transmettre et le modèle, il faudrait utiliser une variante de ce qu'offre le compilateur ALGOL de Grenoble, à savoir :

Trois fonctions standard : TRANSMISSION, MODELE, et DICTIONNAIRE définies comme suit :

- procédure TRANSMISSION (CANAL, AIGUILLAGE, N1, A1, N2, A2, ... Nn, An) ;

commentaire :

. TRANSMISSION est l'unique fonction standard utilisée pour la transmission des quantités du programme vers l'extérieur et vice versa.

. CANAL désigne l'unité logique avec lequel le programme entre en relation. A l'appel de TRANSMISSION, CANAL a pour valeur l'entier le plus près de la valeur de l'expression arithmétique mise à sa place.

. AIGUILLAGE est une expression de désignation qui permet de renvoyer le contrôle du programme vers un endroit où MODELE communique un modèle à TRANSMISSION ; comme une expression de désignation peut prendre plusieurs valeurs possibles à l'exécution, ceci permet d'avoir un modèle dynamique.

. Les autres paramètres N1, A1, N2, A2, ... , Nn, An se traitent par couple. La valeur de n est variable. An désignent les quantités à transmettre vers l'extérieur, ou les adresses où doivent se mettre les données à la lecture. Dans le cas où An est une variable simple : Nn doit prendre la valeur 1. Dans le cas d'un tableau, An donne l'adresse du premier élément à traiter, Nn spécifie le nombre d'éléments de tableau qu'on désire considérer, toute expression arithmétique est admise pour spécifier Nn.

Il est à remarquer que :

. Plusieurs ordres TRANSMISSION peuvent partager la lecture d'une seule carte; quand tous les caractères d'une carte sont lus, TRANSMIS-

SION lit automatiquement sur la carte suivante.

. Pour la sortie par imprimante, le processus est semblable; plusieurs TRANSMISSION exécutés à la suite écrivent le résultat sur la même ligne. A l'épuisement de toutes les positions d'une ligne, l'écriture se fait automatiquement sur la ligne suivante.

b - procedure MODELE (CHAINE, N1, N2, ... Nn) ;
commentaire : MODELE sert à communiquer à TRANSMISSION son premier paramètre CHAINE qui est une chaine "modèle" : les autres paramètres N2, ... Nn qui sont des expressions arithmétiques, servent à spécifier les valeurs numériques d'éventuels duplicateurs variables X qui se trouveraient dans le "modèle".

Exemple :
MODELE ('5B'ALLER A'//') ou
MODELE ('XB'ALLER A'//',5) ou
MODELE ('XBXB'ALLER A'//',3, 2)

communiquent le même "modèle" à TRANSMISSION.

Si 6 désigne l'imprimante, le programme suivant

```
début tableau  X[0:10] ; entier M, N ; réel T ;
    M = 0 ; T:=3.14155926536 ; N:=500 ;
    X[M]:= 18061579 ; X[M+1]:=COS (T) ;
    E1 :  MODELE ('2(3B4D),3B,+D.6D10+3D,3B,-Z.4DB4D');
    TRANSMISSION (6,E1,1,N,1,M,siM=0 alors 2 sinon 0, X[M])
fin
```

imprimera sur le papier le résultat suivant :
␣␣␣␣␣500␣␣␣␣␣␣␣␣0+1.806158$_{10}$+007␣␣␣1.0000␣0000

c - procedure DICTIONNAIRE (CHAINE) ; chaîne CHAINE ;

commentaire : La procédure DICTIONNAIRE sert à spécifier le dictionnaire pour le modèle alpha, ce qui signifie :

- si dans le programme il n'y a pas d'ordre DICTIONNAIRE, la correspondance entre les alphanumériques à l'extérieur et leurs équivalents entiers dans le programme est celle définie par la machine utilisée.

- mais si l'on désire modifier cette correspondance, c'est-à-dire attribuer aux équivalents des alphanumériques d'autres valeurs, on

utilisera DICTIONNAIRE de la manière suivante :

Le paramètre de DICTIONNAIRE est une chaîne, on met dans cette chaîne tous les symboles distincts dans un ordre désiré. A partir du moment où l'ordre DICTIONNAIRE est exécuté, l'équivalent du premier alphanumérique dans CHAINE sera 1, l'équivalent du deuxième sera 2, et ainsi de suite.

Exemple :

DICTIONNAIRE ('ABC1␣04 ...') fait correspondre à

A	la valeur	1
B		2
C		3
1		4
␣		5
0		6
⋮		⋮

Voici un exemple d'utilisation des opérations d'alignement et de composition :
le programme suivant

```
debut entier  K, N ; entier tableau A[0:13];
    E1 : MODELE ('↑34B'TRIANGLE␣DE␣PASCAL '//') ;
    E2 : MODELE ('XB,X(6Z)//',39-3xN,N) ;
    TRANSMISSION (6, E1) ;
    pour  N:= 0 pas 1 jusqua 13 faire
      debut  A[N]:=1 ;
        pour  K := N - 1 pas -1 jusqua 1 faire  A[K]:=A[K-1]+A[K];
        TRANSMISSION  (6, E2, N, A[1])
      fin
fin
```

écrira au début d'une nouvelle page le résultat suivant

I. 5. APPLICATIONS AU COMPILATEUR DE GRENOBLE

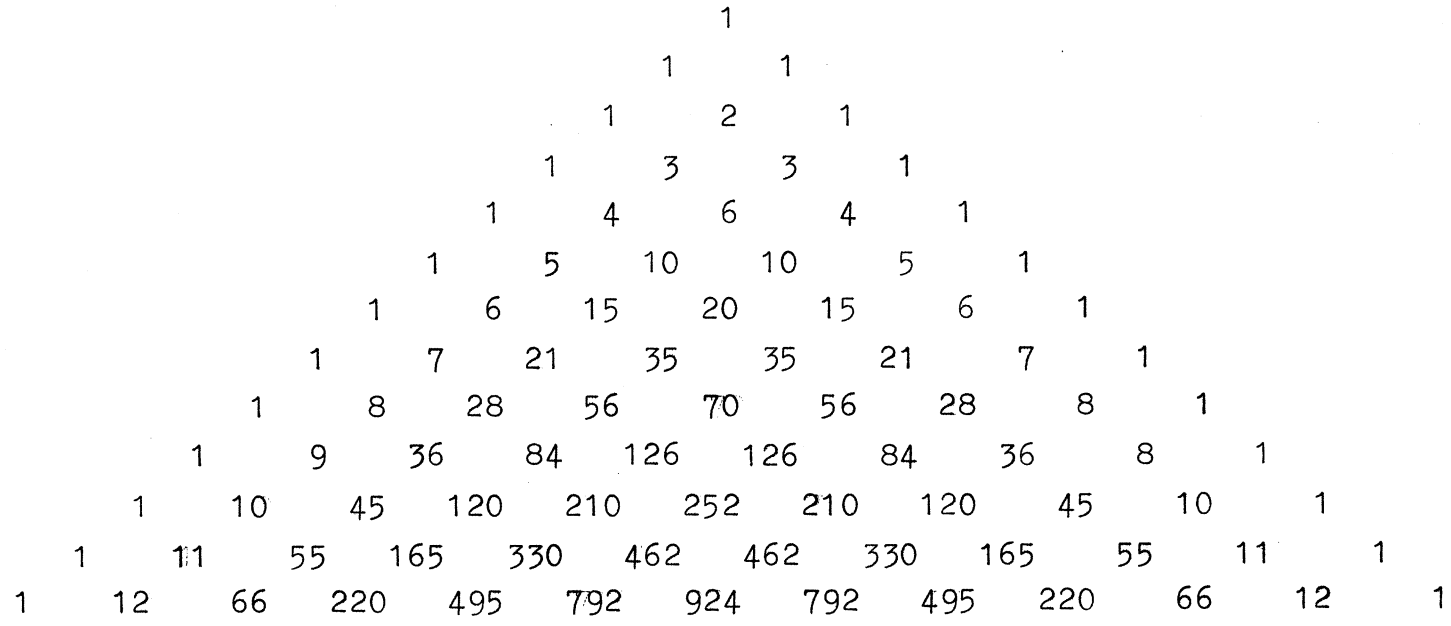
Il existe pour le compilateur ALGOL de GRENOBLE les facilités d'entrée-sortie de FORTRAN IV. C'est pour cette raison que nous n'avons pas voulu réaliser le modèle proposé par A. C. M., qui s'est inspiré de celui de FORTRAN IV avec quelques améliorations déjà citées. Nous avons préféré programmer une variante de la proposition I. F. I.I

Avec les procédures dites de base ainsi construites, nous pouvons écrire n'importe quelle procédure d'entrée-sortie qui s'avérera utile directement en ALGOL. Ceci en effet paraît séduisant car une procédure d'entrée-sortie ainsi écrite en ALGOL a l'avantage d'être facilement modifiable lorsqu'on s'apperçoit qu'elle comporte quelques défauts difficiles à détecter lors de sa construction. Cela se fait sans créer de problème pour la communication entre installations car la modification est faite au niveau d'ALGOL, donc facilement compréhensible.

Dans les pages qui suivent nous allons décrire une réalisation, basée sur cette idée. La description se divise en deux parties :

- procédures de base d'entrée-sortie
- procédures auxiliaires d'entrée-sortie

TRIANGLE DE PASCAL



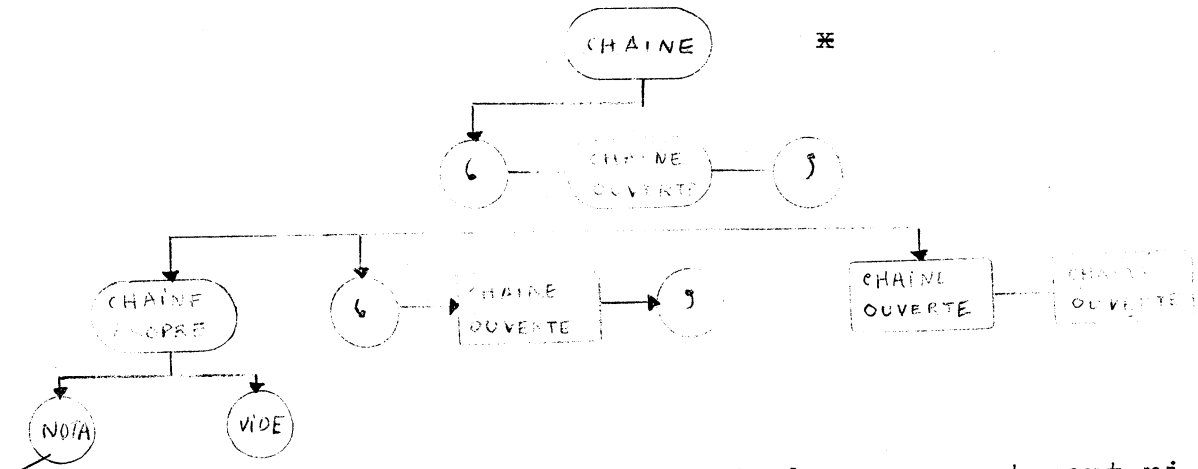
I. 5. 1. PROCEDURES DE BASE D'ENTREE-SORTIE

Nous allons décrire en détail une réalisation des procédures de base pour l'entrée-sortie, elles sont au nombre de trois et s'intitulent :

LONGUEUR
LIRE SYMBOLE
ECRIRE SYMBOLE

Ce travail a été réalisé en utilisant le compilateur ALGOL de GRENOBLE sur IBM 7044 dont le langage d'assemblage est le MAP. Elles sont toutes les trois écrites en code MAP.

Rappelons la définition de chaîne en ALGOL, elle se traduit syntaxiquement par le schéma suivant :



un assemblage quelconque de symboles de base ne contenant ni ni

Exemple

' début fin ' ' allera B ' 0 '

est une chaîne

Le compilateur ALGOL - IBM 7044 ne permet pas l'utilisation de chaînes imbriquées, ni même de "chaînes propres" proprement dites.

---

*Il est à remarquer que cette définition de chaîne est ambigue.

On est donc amené à donner un autre sens au mot "chaîne" ; dans ce qui suit nous entendons par chaîne tout assemblage (éventuellement vide) de symboles (caractères alphanumériques) disponibles sur IBM 7044 à l'exception de , délimité par les crochets de chaîne qui ont comme représentation machine les assemblages de symboles suivants

'(' pour
et ')' pour

Voyons sommairement comment on construit une procédure en code :

Disons qu'à n'importe quel endroit d'un programme ALGOL on peut définir une procédure dont le corps est en code MAP. L'entête de la procédure est en ALGOL et le corps défini comme suit :
<corps de procedure en code> ::= 'CODE'<suite d'instruction MAP>'FCØDE'

Le caractère ' terminant le symbole 'CODE' doit être perforé en colonne 72 de la carte précédant la première instruction MAP tandis que le premier ' du symbole 'FCODE' doit être en colonne 1. de la carte suivant immédiatement la dernière instruction MAP.

Pour chaque procédure d'un programme ALGOL le compilateur réserve deux mémoires représentées par Fn et Pn ; n étant le numéro d'ordre d'apparition de la déclaration de la procédure en question depuis le début du programme sans tenir compte des divers niveaux de bloc. Et pour chaque paramètre formel de la n-ième procédure le compilateur réserve deux mémoires au nom de Pn-m et Fn+m, m étant le numéro d'ordre du paramètre formel dans la liste des paramètres formels de la procédure. Précisons la manière d'utiliser les mémoires Pn, Fn, Pn-m, Fn+m.

a - Pour affecter une valeur à un indicateur de fonction, on met la dite valeur dans la mémoire Fn à la sortie du corps de la procédure.

b - Pour un paramètre formel spécifié comme variable simple, deux cas se présentent :

1. S'il est appelé par valeur, la dite valeur se trouve dans Fn+m.

2. S'il est appelé par nom, on obtient son adresse en machine dans l'index 2 après exécution de l'instruction T S X Pn-m, 4.

c - Le paramètre formel est spécifié comme étiquette, le transfert à la dite étiquette s'obtient par l'instruction T S X Pn-m, 4.

d - Le paramètre formel est spécifié comme tableau, on trouve en partie adresse de Fn+m l'adresse du premier élément du tableau effectif. Les tableaux sont rangés ligne par ligne.

f - Le paramètre est spécifié comme chaîne. A chaque apparition de chaîne dans le programme, le compilateur construit une séquence de mémoire de la forme

Hk		PZE	l
l mémoires	(	BCI	1, caractères
	(	:	
	(	BCI	1, ...

k étant le numéro d'ordre d'apparition de la chaîne en question depuis le début du programme sans tenir compte des divers niveaux de bloc.

L étant le nombre de mémoires nécessaires au stockage des caractères formant la chaîne ; chaque mémoire peut contenir 6 caractères.

Si le nombre de caractères contenus dans la chaîne n'est pas multiple de 6 ; la dernière mémoire de la séquence considérée est complétée par des caractères dont le code actuel est 77 ce qui se traduit sur l'imprimante par des blancs.

Pour trouver l'adresse de Hk , on utilise comme pour une variable simple l'instruction

T S X Pn-m, 4

On aura Hk dans l'index 2.

Ainsi la procédure suivante écrira sur papier une chaîne quelconque. Si le nombre de caractères de la chaîne dépasse la limite permise qui est 131 sur une ligne, les derniers caractères de la

chaine sont ignorés.

On suppose que la procédure est déclarée avant toute autre déclaration de procédure dans le programme

```
Procedure   ECRIRE   TEXTE   (CHAINE) ; chaîne   CHAINE ;  'CODE'

      TSX          P1-1, 4        mettre dans l'index l'adresse de la
                                  chaîne
      SXA          *+1, 2         - mettre dans la zone décrément de
      CLA          **               ZONE + 1  le nombre de mémoires à
      ALS          18               sortir (le nombre de mémoires néces-
      STD          ZONE +1          saires au stockage des caractères
                                    de la chaîne).
      TXI          *+1, 2, 1      - mettre dans la zone adresse de
      SXA          ZONE+1, 2        ZONE + 1  l'adresse de la première
                                    mémoire des mémoires où sont sto-
                                    ckés les caractères de la chaîne.
      CALL         JOBCU (ZØNE)   appel du sous-programme d'écriture.
      TRA          ZØNE+2         sortie du corps de la procédure
      EXTERN       JOBOU          déclaration du sous-programme
                                  d'écriture
ZONE  PZE          1              spécifications nécessaires au
      PZE          ,,             sous programme d'écriture
'FCODE'  ;
```

Passons maintenant à la description des procédures LONGUEUR, LIRE SYMBOLE, ECRIRE SYMBOLE

entier procedure LONGUEUR (CHAINE) ; chaine CHAINE ;

< corps de procédure en code > ;
(voir l'appendice)

L'indicateur de fonction LONGUEUR de type entier prend comme valeur le nombre de caractères contenus dans la chaîne qui est son paramètre effectif sans compter des crochets de chaîne.

L'algorithme de construction de LONGUEUR est très simple et peut se résumer comme suit :

On sait qu'une chaîne est stockée en mémoire sous la forme :

Hk	PZE	l
l mémoires	BCI	1, des caractères
	:	
	BCI	1, ...

et que si le nombre de caractères de "CHAINE" n'est pas un multiple de 6, la dernière mémoire des l mémoires servant à stocker "CHAINE" est complétée par des codes $(77)_8$.

Pour calculer la valeur de LONGUEUR on multiplie donc l par 6, puis on soustrait du produit résultant le nombre de codes $(77)_8$ trouvés dans la mémoire Hk+l

Exemple :

LONGUEUR ('123456') = 6

LONGUEUR ('A␣B␣C:=5+␣-1.2␣') = 15

␣ représente l'espace

LONGUEUR ('') = 0

Tout caractère IBM 7044 est permis dans "CHAINE", paramètre de LONGUEUR à l'exception de '

```
procedure  LIRE  SYMBOLE  (CANAL,  CHAINE,  DESTINATION) ;
    valeur  CANAL ; entier  CANAL, DESTINATION ; chaine  CHAINE ;
            < corps de procédure en code >  ;
```

(voir l'appendice)

La procédure LIRE SYMBOLE sert de moyen de communication entre les supports extérieurs et les variables de type entier du programme dans le sens extérieur —> programme comme son nom l'indique.

La valeur de CANAL détermine le support extérieur à partir duquel l'information est communiquée au programme ALGOL.

Si CANAL = 0	La lecture se fait à partir des cartes
CANAL = 1	La lecture se fait à partir des bandes magnétiques ou disques.

Le fichier au nom de B O G E étant défini au moyen d'un carte contrôle de la forme suivante

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
$ F I L E                        '  B O  G  E  '  U  0  n  ,

26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
U  0  m  ,  R  E  A  D  Y  ,  B  L  0  C  K  =  0  1  4  0  ,

47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66....
R  E  E  L  ,  L  R  L  =  0  1  4  ,  R  C  T  =  0  1  0
```

les indices indiquent le numéro de colonne de la carte ; n, m étant les numéros de bandes ou disques sur lesquels le fichier est défini.

La dimension du bloc est de 140 mémoires qui se divisent en dix enregistrements logiques de 14 mémoires chacun.

LIRE SYMBOLE utilise CHAINE (2ème paramètre) comme un dictionnaire ; elle compare le caractère trouvé sur le support ex-

térieur avec les caractères de CHAINE un à un de gauche à droite jusqu'à ce qu'elle trouve un caractère de CHAINE identique à celui du support extérieur. Alors le numéro d'ordre de ce caractère dans CHAINE, obtenu en associant les entiers 1, 2, 3 ... avec les 1er, 2e, 3e ... caractères de CHAINE, est donné comme valeur à DESTINATION. Si le caractère du support extérieur n'est égal à aucun des caractères de CHAINE , la valeur zéro est donnée à DESTINATION.

Actuellement pour la construction du dictionnaire on réserve 720 mémoires dans chacune desquelles on met un caractère de CHAINE par ordre décroissant, autrement dit :

DICT étant la référence de ces 720 mémoires on met dans
DICT - 1 le premier caractère de CHAINE
DICT - 2 le deuxième " " "

Autrement dit CHAINE peut avoir au maximum 720 caractères ce qui constitue une limitation mais s'avère largement suffisant dans les cas habituels. Si l'utilisateur a besoin de plus de 720 caractères pour CHAINE, pour lever cette limitation il suffit qu'il change une seule carte dans le programme, la carte en question étant :

DICT BES n

n étant supérieur ou égal au nombre de caractères dont il a besoin pou CHAINE.

Par contre s'il sait que la longueur de ses chaînes CHAINE ne dépasse pas un nombre m beaucoup plus petit que 720 ; pour ne pas encombrer inutilement la mémoire rapide, il donnera à n cette valeur m.

Il est à remarquer que la notion de caractère disponible sur un support extérieur est tout à fait artificiel, en réalité il y a aucun moyen de lire un seul caractère à la fois. Chaque fois qu'un ordre de lecture est donné, une unité d'information est transmise de l'extérieur vers l'unité centrale, cette unité d'information ne contient pas un seul caractère mais un groupe de caractère.

Si CANAL = 0

par JOBIN (un sous programme de lecture de bibliothèque) on lit une carte qui comporte 72 caractères utilisables, les huit dernières colonnes de la carte donnée étant réservées pour l'identification.

Par conséquent on doit stocker ces 72 caractères dans une zone tampon de la mémoire rapide, ensuite à chaque appel de LIRE SYMBOLE on examine un caractère de cette zone tampon. Quand ces 72 caractères sont tous utilisés par 72 appels de LIRE SYMBOLE avec CANAL = 0 , un autre cycle recommence : on commande la lecture d'une autre carte et le processus continue ainsi de suite.

Si CANAL = 1

il s'agit d'une lecture d'information à partir des bandes magnétiques ou disques selon la définition du fichier. Le processus est tout à fait semblable, car en général les enregistrements sur bandes ou disques sont des images de cartes. En tout cas nous avons construit le programme en tenant compte de ce fait et la lecture d'une carte (dans le cas où CANAL = 0) est remplacée ici par celle d'un enregistrement logique au moyen des instructions IOBS.

De toute façon LIRE SYMBOLE est conçue de telle façon que dans le cas général, l'utilisateur n'aura pas à se soucier de tous ces détails, il suffit qu'il sache que chaque fois qu'une instruction LIRE SYMBOLE est exécutée, tout se passe comme si un caractère est lu à partir du support extérieur spécifié par la valeur actuelle de CANAL.

Procedure ECRIRE SYMBOLE (CANAL, CHAINE, SOURCE) ;

valeur CANAL ; entier CANAL, SOURCE ; chaine CHAINE ;

< corps de procédure en code >
(voir l'appendice)

Cette procédure est symétrique à LIRE SYMBOLE. En effet elle sert de moyen de communication entre les variables de type entier du programme ALGOL avec le milieu extérieur dans le sens programme —> extérieur.

Comme pour LIRE SYMBOLE, la valeur de "CANAL" détermine le support extérieur avec lequel le programme entre en relation.

CANAL = 0 s'il s'agit de carte
CANAL = 1 bande, disque
CANAL = 2 papier

"CHAINE" est le dictionnaire d'ECRIRE SYMBOLE ; à chaque appel d'ECRIRE SYMBOLE le SOURCE-ième caractère de "CHAINE" est transmis vers le support extérieur actuellement valable. Si la valeur de "SOURCE" est inférieur à 1 ou supérieur à LONGUEUR (CHAINE) des messages d'erreur appropriés sont imprimés sur le listage du programme ALGOL.

Précisons les détails techniques :

a - CANAL = 0 carte

On a dit qu'à chaque appel d'ECRIRE SYMBOLE un caractère est transmis vers le milieu extérieur, ici un caractère serait perforé sur carte ; en fait on stocke ce caractère dans une zone tampon, chaque fois que cette zone est remplie, c'est-à-dire après 72 appels d'ECRIRE SYMBOLE avec CANAL = 0 une carte est perforée par JOBPP (sous-programme de bibliothèque). Par conséquent si on veut perforer un nombre de caractères non multiple de 72 , on doit vider la zone tampon par l'instruction ALGOL. VIDER (CANAL) avec CANAL=0 pour sauvegarder les derniers caractères.

b - CANAL = 1 bandes, disques

Le processus est le même que celui du cas précédent, on doit

vider la zone tampon par l'instruction VIDER (1) quand cela est nécessaire.

c - CANAL = 2 papier

Les caractères sont écrits un par un sur la même ligne par les appels successifs d'ECRIRE SYMBOLE. Quand une ligne est complète (131 symboles) , il faut donner l'ordre ALALIGNE pour pouvoir écrire sur la ligne suivante, sinon les derniers caractères sont perdus. On peut même sauter à la ligne suivante à n'importe quel moment en donnant l'ordre ALALIGNE. Deux instructions ALALIGNE qui se suivent ont pour effet de laisser une ligne blanche au milieu de la feuille d'impression. La mise en page est ainsi très souple, le nombre de caractères qu'on veut mettre sur une ligne peut-être calculé par programme. Cette caractéristique d'ECRIRE SYMBOLE est utilisée très souvent dans la construction de nombreux programmes présentés dans les pages qui vont suivre.

Il est à remarquer que dans le but de pouvoir faire des comparaisons "sur les chaînes", on a introduit un artifice qui consiste à considérer une mémoire spéciale "LOCA" de l'unité centrale de la machine comme un support extérieur au programme. Ainsi on peut écrire chaque fois un caractère dans ce support par ECRIRE SYMBOLE et l'en extraire par LIRE SYMBOLE en donnant à CANAL la valeur 3.

EXEMPLE :

On veut construire une procédure dont le seul paramètre est une chaine ; pour un D trouvé dans cette chaine on écrit sur papier cinq I , pour un Z deux blancs et pour tout autre symbole on saute à la ligne suivante.

La solution correspondante à ce problème sera :

```
procedure ECRIRE TEST (CHAINE) ; chaine CHAINE ;
  debut entier I, J, L ;
    L := LØNGUEUR (CHAINE) ;
    pour I:= 1 pas 1 jusqu a L faire
      debut

          ECRIRE SYMBOLE (3, CHAINE, I) ;
```

```
        LIRE SYMBOLE (3, 'DZ', J) ;
        si J = 0 alors ALALIGNE
                 sinon
        si J = 1 alors pour J= 1 pas 1 jusqua 5 faire
                                ECRIRE SYMBOLE (2, 'I', 1)
                 sinon
        pour J:=1, 2 faire
             ECRIRE SYMBOLE (2, 'U', 1)
    fin
fin   DE ECRIRE TEST ;
```

Avec ces trois procédures primitives en code LONGUEUR, LIRE SYMBOLE, ECRIRE SYMBOLE on a des moyens très puissants pour construire d'autres procédures en ALGOL qui permettent une entrée-sortie convenable, façonnée suivant les besoins, modifiable facilement, et permettant en particulier l'introduction des listes et leur manipulation en ALGOL.

Tout d'abord nous allons exposer la construction des facilités concernant l'entrée-sortie, ensuite nous aborderons les listes et leur manipulation. Les algorithmes seront présentés en ALGOL sous forme de procédures avec des commentaires détaillés.

I. 5. 2. PROCEDURES AUXILIAIRES D'ENTREE-SORTIE

Différents problèmes d'entrée-sortie sont résolus, on les présente sous forme de procédures avec des commentaires détaillés. Remarquons qu'elles sont toutes écrites en ALGOL, donc modifiables à loisir et très souples. Remarquons aussi le caractère d'imbrication de ces constructions, les procédures complexes sont toujours bâties à partir des procédures plus simples.

1 -

```
procedure   ECRIRE SOUS CHAINE (CANAL, CHAINE, I, J) ;
   valeur   CANAL , I, J ; entier CANAL , I, J ; chaine CHAINE ;
   debut commentaire
```

cette procédure écrit sur le support extérieur spécifié par CANAL, une sous-chaine de CHAINE (deuxième paramètre). Le premier caractère de la sous-chaine étant le I -ième caractère de CHAINE et J étant la longueur de la sous-chaine.

Si I est inférieur à 1 les 1-I premiers caractères de la sous-chaine sont ignorés.

Si I + J - 1 est supérieur à LONGUEUR (CHAINE) les derniers I + J - 1 - LONGUEUR (CHAINE) caractères de la sous-chaine ne sont pas considérés.

Exemple :

L'instruction suivante (I et J étant déclarés dans le bloc englobant cette instruction) :

```
pour I := 12, 7, 5, 12, 3 faire
ECRIRE SOUS CHAINE  (2,'u LE uPLOMBIER', I,
                     si I = 12 alors 5 sinon
                     si I = 7  alors 2 sinon 1 ) ;
```

écrit sur papier

ROMPTRE

```
entier K ;
```

```
        J := si LONGUEUR (CHAINE) I+J-1 alors I+J-1
             sinon LONGUEUR (CHAINE) ;
        I := si I < 1 alors 1 sinon I ;
        pour K := I pas 1 jusqua J faire
            ECRIRE SYMBOLE (CANAL, CHAINE, K)
fin de ECRIRE SOUS CHAINE ;
```

2 -

```
procedure ECRIRE CHAINE (CANAL, CHAINE) ;
   valeur CANAL ; entier CANAL ; chaine CHAINE ;
   debut commentaire
```

Cette procédure sort une chaine (explicitement CHAINE) sur le support extérieur spécifié par la valeur de CANAL, il est à remarquer que sur papier le nombre maximum de caractères à imprimer sur une ligne est de 131. Le saut de ligne étant laissé volontairement non automatique (ceci peut très bien se faire en incorporant un compteur dans le corps de la procédure), pour aller à la ligne il faut donner l'ordre ALALIGNE ;

```
entier I, K ;
    K := LONGUEUR (CHAINE) ;
    pour I := 1 pas jusqua K faire
         ECRIRE SYMBOLE (CANAL, CHAINE, I)
fin de ECRIRE CHAINE ;
```

3 -

```
procedure ECRIRE BOOLEEN (CANAL, BOOLEEN)
   valeur CANAL ; entier CANAL ; booleen BOOLEEN ;
   commentaire
```

Cette procédure sort une quantité booléenne sous la forme * VRAI * ou * FAUX *

```
si BOOLEEN alors ECRIRE CHAINE (CANAL,'* VRAI *')
             sinon ECRIRE CHAINE (CANAL,'* FAUX *') ;
```

4 -

```
procedure  LIRE  ENTIER  (CANAL, ENTIER) ; valeur  CANAL ;
   entier  ENTIER ;
   debut commentaire :
```

Cette procédure affecte à ENTIER la valeur d'un nombre entier qui sur le support extérieur apparait comme une séquence de chiffres décimaux éventuellement précédés par un signe et suivis par une virgule. Tout autre symbole précédent le signe est écarté ;

```
entier  N, K ; booléen  B ;
  ENTIER := 0 ;
  B := vrai
  pour  K :=  1,  K + 1  tantque  N = 0  faire
        LIRE SYMBOLE (CANAL,'0123456789-+,',N) ;
  si  N = 11  alors  := faux
  si  N > 10  alors  :=  1 ;
  pour  K := 1 , K + 1  tantque  N ≠ 13  faire
  debut
    si  N = 0  alors
        debut
             ECRIRE CHAINE (2,'ERREUR␣DANS␣DONNEES') ;
             ALALIGNE ; allera EXIT
        fin ;
        ENTIER :=  10 x ENTIER + N - 1 ;
        LIRE SYMBOLE  (CANAL,' 0123456789-+,',N)
    fin ;
    si  ¬ B  alors  ENTIER := - ENTIER    ;  EXIT :
  fin  DE LIRE ENTIER ;
```

5 -

```
procedure  ECRIRE  ENTIER  (CANAL,  ENTIER) ;
   valeur  CANAL,  ENTIER ; entier  CANAL, ENTIER ;
   debut commentaire
```

Cette procédure écrit un nombre entier sur le support extérieur sous forme décimale éventuellement précédé par le signe " - " ,

si le nombre est négatif. Deux ordres ECRIRE ENTIER exécutés à la suite impriment 2 entiers sur la même ligne pour CANAL = 2 ;

```
  entier  N, B ;
  si  ENTIER  < 0  alors
     debut
       ECRIRE SYMBOLE (CANAL,'-', 1)
       ENTIER := - ENTIER
     fin ;
INFER:
si  ENTIER  < 10  alors
     debut
       ECRIRE SYMBOLE (CANAL,'0123456789', ENTIER+1)
       allera EXIT
       fin ;
N := 1 ;  B :=  ENTIER  ÷ 10 ;
SUPER :
si  B  > 10  alors
     debut
       B := B ÷ 10 ; N := N+1  ;
       allera   SUPER
     fin ;
ECRIRE SYMBOLE  (CANAL,'0123456789', B+ 1)
ENTIER := ENTIER - B x 10 ↑ N ;
ZERO:
N := N+1 ;
si  ENTIER  < 10 ↑ N   ∧ N ≠ 0  alors
      debut
        ECRIRE SYMBOLE (CANAL,'0', 1) ;
        aller a   ZERO
      fin ;
allera  INFER
EXIT:
fin  DE ECRIRE ENTIER ;
```

6 -

```
procedure  ENTIER CADRE  (CANAL, SOURCE, CADRAGE) ;
   valeur  CANAL, SOURCE, CADRAGE ; entier CANAL, SOURCE, CADRAGE ;
   debut commentaire :
```

Cette procédure écrit l'entier SOURCE sur le support extérieur spécifié par CANAL. CADRAGE désigne le nombre d'espaces utilisés pour écrire cet entier.

```
Exemple : ENTIER CADRE  (2, 12, 3)  donne u12
tandis que  ENTIER CADRE  (2, -12, 5)  donne uu-12 ;
entier  P ,
pour  P := 0, P + 1 tantque
      ¬((ABS(SOURCE)<10↑P)∧(10↑(P-1)<ABS(SOURCE))) faire
      P:= P ;
si  SOURCE  < 0  alors  P := P+1 ;
CADRAGE := CADRAGE - P ;
pour  P := 1  pas  1  jusqua  CADRAGE  faire
ECRIRE SYMBOLE  (2,'u',1) ; ECRIRE ENTIER (CANAL, SOURCE)
fin  DE  ENTIER CADRE ;
```

7 -

```
procedure  ECRIRE REEL  (CANAL, SOURCE) ;
   valeur  CANAL, SOURCE ; entier  CANAL ; reel  SOURCE ;
   debut commentaire :
```

Cette procédure écrit sur le support extérieur spécifié par la valeur de CANAL, un nombre réel sous format standard c'est-à-dire normalisé avec huit décimales après la virgule, un signe " ± " pour la partie exposant comportant deux chiffres. Le nombre est précédé d'un signe "-" dans le cas où il est négatif. Le nombre total d'espaces occupés par ce nombre est de quinze. Deux ordres ECRIRE REEL écrivent deux nombres sur la même ligne pour CANAL = 2. Zéro est sorti cadré à droite avec 14 blancs à gauche.

Exemple :

L'instruction ECRIRE REEL (2, - 35.0578) écrira sur papier

uu-.35057800+02

Il y a toujours arrondi sur le dernier chiffre décimal.

```
reel B ; entier  PUISSANCE , K ;
si  ABS  (SOURCE) = alors
debut
  ECRIRE CHAINE (CANAL,  '␣␣␣␣␣␣␣␣␣␣␣␣␣␣0') ;
allera  EXIT
fin  ;
si  SOURCE  < 0  alors
debut
  ECRIRE CHAINE  (CANAL, '␣␣-.') ;
  SOURCE := - SOURCE
fin            sinon
ECRIRE CHAINE  (CANAL; '␣␣␣.') ;
PUISSANCE  := 0 ;
EXPOSANT POSITIF :
si  SOURCE  >  1  alors
  debut
    PUISSANCE := PUISSANCE + 1 ; SOURCE := SOURCE /10 ;
    allera  EXPOSANT POSITIF
  fin ;
    EXPOSANT NEGATIF :
  si  SOURCE x 10 < 1  alors
  debut
    PUISSANCE  :=  PUISSANCE-1 ; SOURCE := SOURCE x 10 ;
    allera  EXPOSANT NEGATIF
  fin ;
    SOURCE := SOURCE x 10 ↑ 8 + 0.5 ;
    commentaire
```

L'instruction précédente sert à arrondir le dernier des huit chiffres décimaux après la virgule ;

```
si  SOURCE  ⩾  10 ↑ 8  alors
  debut
     SOURCE := SOURCE / 10 ; PUISSANCE := PUISSANCE + 1
  fin  CONCERNANT LE CAS  0.999 ... 9 ... ;
```

```
ECRIRE ENTIER (CANAL, ENTIER (SOURCE)) ;
ECRIRE SYMBOLE (CANAL, si PUISSANCE ⩾ 0 alors 1 sinon 2) ;
PUISSANCE := ABS (PUISSANCE) ;
si PUISSANCE < 10 alors ECRIRE SYMBOLE (CANAL,'0', 1) ;
ECRIRE ENTIER (CANAL, PUISSANCE) ; EXIT :
  fin DE ECRIRE REEL ;
```

8 -

```
procedure SORTIREELEXP (CANAL, A, M, N) ; valeur CANAL, A, M, N ;
   entier CANAL, M, N ; reel A ;
   debut commentaire :
```

cette procédure écrit sur le support extérieur dont la nature est spécifiée par le premier paramètre, le nombre réel A (deuxième paramètre) avec une partie exposant comportant un signe et deux chiffres. Le nombre total d'espaces utilisés (toujours supérieur ou égal à cinq) est spécifié par le troisième paramètre M. Le nombre est écrit sous forme normalisée et cadré à droite, il peut comporter des blancs à gauche. Le quatrième paramètre spécifie le nombre de chiffres qu'on veut avoir après la virgule. Le dernier décimal est arrondi.

Exemple :

Pour A = 53. 72 les instructions

```
     SORTIREELEXP (2, A, 12, 2)
     SORTIREELEXP (2, A, 10  3)
```

écriront sur papier respectivement

```
    ␣␣␣␣␣␣.54+02
    ␣␣␣.53 +02                ;
```

```
reel B ; entier PUISSANCE , K ;
si ABS (A) = 0 alors
   debut
    pour K := 1, K+1 tantque K ⩽ M-4 faire
         ECRIRE SYMBOLE (CANAL,'␣',1) ;
         ECRIRE CHAINE (CANAL,'0+00') ;
         allera EXIT
  fin    A ≠ 0 on va traiter le cas général ;
    pour K := 1, K+1 tantque K ⩽ M-(5+N) faire
```

```
        ECRIRE SYMBOLE  (CANAL, ' ', 1) ;
    pour  K := si  A < 0  alors  1  sinon  2,  3  faire
        ECRIRE SYMBOLE  (CANAL, '- .' , K) ;
    A := ABS (A) ;
    EXPOSANT  POSITIF
    si    A ⩾ 1  alors
  début
    A := A/10   ; PUISSANCE := PUISSANCE + 1 ;
    allera  EXPOSANT POSITIF
  fin  ;
    EXPOSANT NEGATIF :
    si  A x 10 < 1  alors
  debut
    A := A x 10 ; PUISSANCE := PUISSANCE - 1 ;
    allera  EXPOSANT NEGATIF
  fin  ;
    A := A + 0.5 x 10 - N  ;
    si  A ⩾ 1  alors
  debut
    A := A / 10   ; PUISSANCE := PUISSANCE +1
  fin  cette instruction est nécessaire à cause de la représentation
non exacte des nombres flottants en machine ;
    pour  K := 1, K+1  tantque  K ⩽ N  faire
  debut
    A := A x 10 ;  B := ENTIER (A) ;
        ECRIRE SYMBOLE  (CANAL,'0123456789',B+1) ;
        A := A - B
  fin  ;
ECRIRE SYMBOLE  (CANAL,'+-', si  PUISSANCE ⩾ 0  alors 1  sinon  2) ;
PUISSANCE := ABS  (PUISSANCE)  ;
si  PUISSANCE  < 10  alors  ECRIRE SYMBOLE (CANAL,'0', 1) ;
ECRIRE ENTIER  (CANAL, P)     ; EXIT ;
  fin  DE  SORTIREELEX P ;
```

9 -

procedure MATRICE REELLE (TABLEAU, M, N, TITRE, TITRE COL) ;
valeur M, N ; tableau TABLEAU ; chaine TITRE , TITRE COL ;
debut commentaire :
cette procédure s'occupe du problème de l'impression des matrices sur papier.

Dans la programmation des problèmes d'analyse numérique on a souvent à utiliser des matrices ayant en général des dimensions variables. Pour les imprimer il faut construire chaque fois un programme spécial d'impression, il en résulte une perte de temps appréciable pour la mise au point.
MATRICE REELLE résout ce problème pour des matrices de termes réels de la manière suivante :
- TABLEAU (premier paramètre) étant le nom du tableau à deux dimendimensions où sont rangés les éléments de la matrice en question.
- M (deuxième paramètre) le nombre de lignes de la matrice
- N (troisième paramètre) le nombre de ces colonnes
- TITRE (quatrième paramètre) est une chaîne qui sert de nom pour la matrice.
- TITRE COL (cinquième paramètre) une autre chaine qui sert de désignation pour chaque colonne de la matrice, chaque désignation de colonne occupe quinze positions (y compris des blancs).

Il est à remarquer qu'une chaîne peut-être vide, on voit par là un moyen de supprimer le titre de la matrice ou les désignations de ses colonnes ou les deux à la fois, si l'utilisateur ne les désire pas.

Exemple :

MATRICE REELLE (MATRICE, M, N,'','')

imprimera la matrice MATRICE (M, N) sans titre ni désignation de colonne.

- Les éléments de la matrice sont imprimés au moyen de ECRIRE REEL donc sous forme standard. Ils sont normalisés, avec huit chiffres décimaux dont le dernier est arrondi, une partie signe à gauche, une partie exposant à droite comportant un signe et deux chiffres, le tout occupe quinze positions.

La procédure imprime la matrice par tranches de huit colonnes la dernière tranche pouvant comporter moins de huit colonnes. Le cas d'une matrice ayant moins de huit colonnes est aussi traité. Il y a restauration de page entre chaque tranche. Le titre de la matrice (quatrième paramètre) est répété en tête de chaque tranche.

Exemple :

MATRICE REELLE étant déjà déclarée, la séquence d'instruction suivantes imprime sur papier une matrice de 23 colonnes et 72 lignes ayant pour titre général TABULATION et des désignations de colonnes COLONNE 1, COLONNE 2 .... COLONNE 23.

```
debut entier I, J ; tableau ESSAI [1 : 72, 1 : 23] ;
  pour I := 1 pas 1 jusqu'à 72 faire
    pour J := 1 pas 1 jusqua 23 faire
      ESSAI [I, J] := 0.5 x (I + J) + 41 x I - 9 x J ;
    MATRICE REELLE (ESSAI, 72, 23, ' TABULATION ',
      ' COLONNE 1 COLONNE 2 ..... COLONNE 23 ')
fin ;
```

(voir résultat ci-contre)

```
entier I, J, K, P, Q, T ;
Q:= 0 ;
pour K := si N ≤ 8 alors N sinon 8 ,
          si Q+8 < N alors Q+8 sinon N tantque K ≤ N faire

  debut
    P := Q+1 ; Q:= K ; ALALIGNE ;
    ECRIRE CHAINE (2, TITRE) ;
    pour T := 1, T + 1 tantque T < 4 faire
         ALALIGNE ;
    ECRIRE SOUS CHAINE (2, TITRE COL , (P-1) x 15+1, (Q-P+1) x 15) ;
    ALALIGNE ; ALALIGNE ;
    pour I := 1 pas 1 jusqua M faire
  debut
    ALALIGNE ;
    pour J := P pas 1 jusqua Q faire
         ECRIRE REEL (2, TABLEAU [I, J] ) ;
  fin ;
```

TABULATION

COLONNE1	COLONNE 2	COLONNE 3	COLONNE 4	COLONNE 5	COLONNE 6	COLONNE 7	COLONNE 8
.33000000+02	.24500000+02	.16000000+02	.75000000+01	-.99999999+00	-.94999999+01	-.18000000+02	-.26500000+02
.74499999+02	.66000000+02	.57500000+02	.48999999+02	.40499999+02	.32000000+02	.23500000+02	.15000000+02
.11600000+03	.10750000+03	.98999999+02	.90499999+02	.81999999+02	.73499999+02	.65000000+02	.56500000+02
.15750000+03	.14900000+03	.14050000+03	.13200000+03	.12350000+03	.11500000+03	.10650000+03	.97999998+02
.19900000+03	.19050000+03	.18200000+03	.17350000+03	.16500000+03	.15650000+03	.14800000+03	.13950000+03
.24050000+03	.23200000+03	.22350000+03	.21500000+03	.20650000+03	.19800000+03	.18950000+03	.18100000+03
.28199999+03	.27350000+03	.26500000+03	.25649999+03	.24800000+03	.23950000+03	.23100000+03	.22250000+03
.32350000+03	.31500000+03	.30650000+03	.29800000+03	.28950000+03	.28100000+03	.27250000+03	.26400000+03
.36499999+03	.35649999+03	.34799999+03	.33950000+03	.33100000+03	.32250000+03	.31400000+03	.30550000+03
.40649999+03	.39799999+03	.38950000+03	.38100000+03	.37250000+03	.36399999+03	.35549999+03	.34700000+03
.44800000+03	.43950000+03	.43099999+03	.42250000+03	.41399999+03	.40549999+03	.39700000+03	.38850000+03
.48950000+03	.48099999+03	.47250000+03	.46399999+03	.45549999+03	.44700000+03	.43850000+03	.42999999+03
.53099999+03	.52249999+03	.51399999+03	.50549999+03	.49699999+03	.48850000+03	.47999999+03	.47149999+03
.57249999+03	.56399999+03	.55549999+03	.54699999+03	.53850000+03	.52999999+03	.52149999+03	.51299999+03
.61399999+03	.60549999+03	.59700000+03	.58849999+03	.57999999+03	.57149999+03	.56299999+03	.55449999+03
.65549999+03	.64699999+03	.63850000+03	.63000000+03	.62149999+03	.61299999+03	.60450000+03	.59599999+03
.69699999+03	.68849998+03	.67999999+03	.67149998+03	.66299999+03	.65449999+03	.64599999+03	.63750000+03
.73849999+03	.72999998+03	.72149998+03	.71299998+03	.70449998+03	.69599998+03	.68750000+03	.67899999+03
.77999999+03	.77149999+03	.76299998+03	.75449999+03	.74599999+03	.73749999+03	.72899999+03	.72049999+03
.82149998+03	.81299998+03	.80449999+03	.79599998+03	.78749999+03	.77899999+03	.77049998+03	.76199999+03
.86299998+03	.85449999+03	.84599998+03	.83749999+03	.82899999+03	.82049998+03	.81199998+03	.80349998+03
.90449999+03	.89599999+03	.88749999+03	.87899999+03	.87049999+03	.86199998+03	.85349997+03	.84499999+03
.94599998+03	.93750000+03	.92899999+03	.92049998+03	.91199998+03	.90349998+03	.89499998+03	.88649998+03
.98749999+03	.97899999+03	.97049999+03	.96199998+03	.95349998+03	.94499999+03	.93649998+03	.92799998+03
.10290000+04	.10205000+04	.10120000+04	.10035000+04	.99499998+03	.98649998+03	.97799998+03	.96949999+03
.10705000+04	.10620000+04	.10535000+04	.10450000+04	.10365000+04	.10280000+04	.10195000+04	.10110000+04
.11120000+04	.11035000+04	.10950000+04	.10865000+04	.10780000+04	.10695000+04	.10610000+04	.10525000+04
.11535000+04	.11450000+04	.11365000+04	.11280000+04	.11195000+04	.11110000+04	.11025000+04	.10940000+04
.11950000+04	.11865000+04	.11780000+04	.11695000+04	.11610000+04	.11525000+04	.11440000+04	.11355000+04
.12365000+04	.12280000+04	.12195000+04	.12110000+04	.12025000+04	.11940000+04	.11855000+04	.11770000+04
.12780000+04	.12695000+04	.12610000+04	.12525000+04	.12440000+04	.12355000+04	.12270000+04	.12185000+04
.13195000+04	.13110000+04	.13025000+04	.12940000+04	.12855000+04	.12770000+04	.12685000+04	.12600000+04
.13610000+04	.13525000+04	.13440000+04	.13355000+04	.13270000+04	.13185000+04	.13100000+04	.13015000+04
.14025000+04	.13940000+04	.13855000+04	.13770000+04	.13685000+04	.13600000+04	.13515000+04	.13430000+04
.14440000+04	.14355000+04	.14270000+04	.14185000+04	.14100000+04	.14015000+04	.13930000+04	.13845000+04
.14855000+04	.14770000+04	.14685000+04	.14600000+04	.14515000+04	.14430000+04	.14345000+04	.14260000+04
.15270000+04	.15185000+04	.15100000+04	.15015000+04	.14930000+04	.14845000+04	.14760000+04	.14675000+04
.15685000+04	.15600000+04	.15515000+04	.15430000+04	.15345000+04	.15260000+04	.15175000+04	.15090000+04
.16100000+04	.16015000+04	.15930000+04	.15845000+04	.15760000+04	.15675000+04	.15590000+04	.15505000+04
.16515000+04	.16430000+04	.16345000+04	.16260000+04	.16175000+04	.16090000+04	.16005000+04	.15920000+04
.16930000+04	.16845000+04	.16760000+04	.16675000+04	.16590000+04	.16505000+04	.16420000+04	.16335000+04
.17345000+04	.17260000+04	.17175000+04	.17090000+04	.17005000+04	.16920000+04	.16835000+04	.16750000+04
.17760000+04	.17675000+04	.17590000+04	.17505000+04	.17420000+04	.17335000+04	.17250000+04	.17165000+04
.18175000+04	.18090000+04	.18005000+04	.17920000+04	.17835000+04	.17750000+04	.17665000+04	.17580000+04
.18590000+04	.18505000+04	.18420000+04	.18335000+04	.18250000+04	.18165000+04	.18080000+04	.17995000+04
.19005000+04	.18920000+04	.18835000+04	.18750000+04	.18665000+04	.18580000+04	.18495000+04	.18410000+04
.19420000+04	.19335000+04	.19250000+04	.19165000+04	.19080000+04	.18995000+04	.18910000+04	.18825000+04
.19835000+04	.19750000+04	.19665000+04	.19580000+04	.19495000+04	.19410000+04	.19325000+04	.19240000+04

.20250000+04	.20165000+04	.20080000+04	.19995000+04	.19910000+04	.19825000+04	.19740000+04	.19655000+04
.20665000+04	.20580000+04	.20495000+04	.20410000+04	.20325000+04	.20240000+04	.20154999+04	.20070000+04
.21080000+04	.20995000+04	.20909999+04	.20825000+04	.20740000+04	.20654999+04	.20570000+04	.20484999+04
.21495000+04	.21410000+04	.21325000+04	.21239999+04	.21155000+04	.21070000+04	.20984999+04	.20900000+04
.21910000+04	.21825000+04	.21740000+04	.21654999+04	.21569999+04	.21485000+04	.21400000+04	.21315000+04
.22324999+04	.22240000+04	.22155000+04	.22070000+04	.21984999+04	.21900000+04	.21815000+04	.21730000+04
.22740000+04	.22654999+04	.22570000+04	.22485000+04	.22400000+04	.22315000+04	.22229999+04	.22144999+04
.23155000+04	.23069999+04	.22984999+04	.22899999+04	.22815000+04	.22730000+04	.22645000+04	.22559999+04
.23570000+04	.23485000+04	.23400000+04	.23315000+04	.23229999+04	.23145000+04	.23060000+04	.22975000+04
.23985000+04	.23900000+04	.23815000+04	.23730000+04	.23644999+04	.23559999+04	.23475000+04	.23390000+04
.24400000+04	.24315000+04	.24230000+04	.24145000+04	.24060000+04	.23975000+04	.23890000+04	.23805000+04
.24815000+04	.24730000+04	.24644999+04	.24560000+04	.24475000+04	.24390000+04	.24305000+04	.24219999+04
.25229999+04	.25144999+04	.25060000+04	.24975000+04	.24890000+04	.24805000+04	.24720000+04	.24635000+04
.25644999+04	.25559999+04	.25475000+04	.25390000+04	.25305000+04	.25219999+04	.25135000+04	.25050000+04
.26059999+04	.25974999+04	.25889999+04	.25804999+04	.25719999+04	.25635000+04	.25550000+04	.25465000+04
.26475000+04	.26389999+04	.26304999+04	.26219999+04	.26134999+04	.26050000+04	.25964999+04	.25880000+04
.26889999+04	.26805000+04	.26719999+04	.26634999+04	.26549999+04	.26464999+04	.26380000+04	.26294999+04
.27305000+04	.27219999+04	.27135000+04	.27050000+04	.26964999+04	.26879999+04	.26794999+04	.26710000+04
.27719999+04	.27635000+04	.27550000+04	.27464999+04	.27380000+04	.27294999+04	.27209999+04	.27124999+04
.28134999+04	.28049999+04	.27965000+04	.27880000+04	.27794999+04	.27710000+04	.27625000+04	.27539999+04
.28550000+04	.28464999+04	.28379999+04	.28295000+04	.28210000+04	.28125000+04	.28039999+04	.27955000+04
.28964999+04	.28880000+04	.28794999+04	.28709999+04	.28625000+04	.28540000+04	.28455000+04	.28369999+04
.29380000+04	.29294999+04	.29210000+04	.29125000+04	.29039999+04	.28955000+04	.28870000+04	.28785000+04
.29794999+04	.29709999+04	.29625000+04	.29539999+04	.29455000+04	.29369999+04	.29285000+04	.29200000+04

TABULATION

COLONNE 9	COLONNE10	COLONNE11	COLONNE12	COLONNE13	COLONNE14	COLONNE15	COLONNE16
-.35000000+02	-.43499999+02	-.52000000+02	-.60499999+02	-.68999999+02	-.77499999+02	-.85999998+02	-.94499999+02
.65000000+01	-.20000000+01	-.10500000+02	-.19000000+02	-.27500000+02	-.36000000+02	-.44500000+02	-.52999999+02
.47999999+02	.39500000+02	.31000000+02	.22500000+02	.14000000+02	.55000000+01	-.30000000+01	-.11500000+02
.89499998+02	.80999998+02	.72499999+02	.63999999+02	.55499999+02	.47000000+02	.38500000+02	.30000000+02
.13100000+03	.12250000+03	.11400000+03	.10550000+03	.96999999+02	.88499999+02	.79999999+02	.71499999+02
.17250000+03	.16400000+03	.15550000+03	.14700000+03	.13850000+03	.13000000+03	.12150000+03	.11300000+03
.21400000+03	.20550000+03	.19700000+03	.18850000+03	.18000000+03	.17150000+03	.16300000+03	.15450000+03
.25550000+03	.24700000+03	.23850000+03	.23000000+03	.22150000+03	.21300000+03	.20450000+03	.19600000+03
.29699999+03	.28850000+03	.28000000+03	.27150000+03	.26300000+03	.25449999+03	.24600000+03	.23750000+03
.33850000+03	.33000000+03	.32149999+03	.31300000+03	.30449999+03	.29600000+03	.28750000+03	.27900000+03
.38000000+03	.37149999+03	.36299999+03	.35450000+03	.34600000+03	.33750000+03	.32900000+03	.32049999+03
.42149999+03	.41299999+03	.40449999+03	.39600000+03	.38750000+03	.37899999+03	.37049999+03	.36200000+03
.46299999+03	.45449999+03	.44599999+03	.43750000+03	.42899999+03	.42049999+03	.41199999+03	.40349999+03
.50449999+03	.49600000+03	.48750000+03	.47899999+03	.47049999+03	.46199999+03	.45350000+03	.44500000+03
.54600000+03	.53749999+03	.52899999+03	.52049999+03	.51199999+03	.50349999+03	.49500000+03	.48649999+03
.58750000+03	.57900000+03	.57049999+03	.56199999+03	.55350000+03	.54499999+03	.53649999+03	.52800000+03
.62899999+03	.62049999+03	.61199999+03	.60349999+03	.59500000+03	.58649999+03	.57799999+03	.56949999+03
.67049999+03	.66199999+03	.65349999+03	.64500000+03	.63649999+03	.62799999+03	.61949999+03	.61099999+03
.71199999+03	.70349998+03	.69499999+03	.68649999+03	.67799998+03	.66949999+03	.66099998+03	.65250000+03
.75349998+03	.74499999+03	.73649999+03	.72799998+03	.71949999+03	.71099998+03	.70249999+03	.69399999+03
.79499999+03	.78649999+03	.77799998+03	.76949998+03	.76099998+03	.75249999+03	.74399998+03	.73549998+03
.83649998+03	.82799997+03	.81949999+03	.81099998+03	.80249999+03	.79399999+03	.78549998+03	.77699999+03
.87799998+03	.86949998+03	.86099998+03	.85249999+03	.84399998+03	.83549998+03	.82699999+03	.81849998+03
.91949999+03	.91099998+03	.90249999+03	.89399999+03	.88549998+03	.87699999+03	.86849999+03	.85999998+03
.96099998+03	.95249999+03	.94399999+03	.93549998+03	.92699999+03	.91849999+03	.90999998+03	.90149998+03
.10025000+04	.99399998+03	.98549998+03	.97699999+03	.96849998+03	.95999998+03	.95149998+03	.94299998+03
.10440000+04	.10355000+04	.10270000+04	.10185000+04	.10100000+04	.10015000+04	.99299997+03	.98449999+03
.10855000+04	.10770000+04	.10685000+04	.10600000+04	.10515000+04	.10430000+04	.10345000+04	.10260000+04
.11270000+04	.11185000+04	.11100000+04	.11015000+04	.10930000+04	.10845000+04	.10760000+04	.10675000+04
.11685000+04	.11600000+04	.11515000+04	.11430000+04	.11345000+04	.11260000+04	.11175000+04	.11090000+04
.12100000+04	.12015000+04	.11930000+04	.11845000+04	.11760000+04	.11675000+04	.11590000+04	.11505000+04
.12515000+04	.12430000+04	.12345000+04	.12260000+04	.12175000+04	.12090000+04	.12005000+04	.11920000+04
.12930000+04	.12845000+04	.12760000+04	.12675000+04	.12590000+04	.12505000+04	.12420000+04	.12335000+04
.13345000+04	.13260000+04	.13175000+04	.13090000+04	.13005000+04	.12920000+04	.12835000+04	.12750000+04
.13760000+04	.13675000+04	.13590000+04	.13505000+04	.13420000+04	.13335000+04	.13250000+04	.13165000+04
.14175000+04	.14090000+04	.14005000+04	.13920000+04	.13835000+04	.13750000+04	.13665000+04	.13580000+04
.14590000+04	.14505000+04	.14420000+04	.14335000+04	.14250000+04	.14165000+04	.14080000+04	.13995000+04
.15005000+04	.14920000+04	.14835000+04	.14750000+04	.14665000+04	.14580000+04	.14495000+04	.14410000+04
.15420000+04	.15335000+04	.15250000+04	.15165000+04	.15080000+04	.14995000+04	.14910000+04	.14825000+04
.15835000+04	.15750000+04	.15665000+04	.15580000+04	.15495000+04	.15410000+04	.15325000+04	.15240000+04
.16250000+04	.16165000+04	.16080000+04	.15995000+04	.15910000+04	.15825000+04	.15740000+04	.15655000+04
.16665000+04	.16580000+04	.16495000+04	.16410000+04	.16325000+04	.16240000+04	.16155000+04	.16070000+04
.17080000+04	.16995000+04	.16910000+04	.16825000+04	.16740000+04	.16655000+04	.16570000+04	.16485000+04
.17495000+04	.17410000+04	.17325000+04	.17240000+04	.17155000+04	.17070000+04	.16985000+04	.16900000+04
.17910000+04	.17825000+04	.17740000+04	.17655000+04	.17570000+04	.17484999+04	.17400000+04	.17315000+04
.18325000+04	.18240000+04	.18155000+04	.18070000+04	.17985000+04	.17900000+04	.17815000+04	.17730000+04
.18740000+04	.18655000+04	.18570000+04	.18484999+04	.18400000+04	.18315000+04	.18230000+04	.18145000+04
.19155000+04	.19069999+04	.18985000+04	.18900000+04	.18815000+04	.18730000+04	.18645000+04	.18560000+04

.19570000+04	.19485000+04	.19400000+04	.19315000+04	.19230000+04	.19145000+04	.19059999+04	.18975000+04
.19984999+04	.19900000+04	.19815000+04	.19730000+04	.19645000+04	.19560000+04	.19475000+04	.19390000+04
.20400000+04	.20315000+04	.20230000+04	.20145000+04	.20059999+04	.19975000+04	.19890000+04	.19805000+04
.20815000+04	.20729999+04	.20645000+04	.20560000+04	.20475000+04	.20390000+04	.20305000+04	.20220000+04
.21230000+04	.21145000+04	.21059999+04	.20975000+04	.20890000+04	.20805000+04	.20720000+04	.20634999+04
.21645000+04	.21560000+04	.21475000+04	.21390000+04	.21304999+04	.21220000+04	.21135000+04	.21050000+04
.22060000+04	.21975000+04	.21890000+04	.21805000+04	.21720000+04	.21634999+04	.21550000+04	.21465000+04
.22475000+04	.22390000+04	.22305000+04	.22220000+04	.22135000+04	.22050000+04	.21965000+04	.21880000+04
.22890000+04	.22805000+04	.22719999+04	.22635000+04	.22550000+04	.22465000+04	.22380000+04	.22295000+04
.23305000+04	.23220000+04	.23135000+04	.23050000+04	.22965000+04	.22880000+04	.22795000+04	.22710000+04
.23720000+04	.23635000+04	.23550000+04	.23465000+04	.23380000+04	.23294999+04	.23210000+04	.23125000+04
.24135000+04	.24050000+04	.23965000+04	.23880000+04	.23795000+04	.23710000+04	.23625000+04	.23540000+04
.24550000+04	.24465000+04	.24380000+04	.24295000+04	.24210000+04	.24125000+04	.24040000+04	.23955000+04
.24965000+04	.24880000+04	.24795000+04	.24710000+04	.24625000+04	.24540000+04	.24455000+04	.24370000+04
.25380000+04	.25295000+04	.25209999+04	.25125000+04	.25040000+04	.24955000+04	.24870000+04	.24785000+04
.25794999+04	.25710000+04	.25625000+04	.25540000+04	.25455000+04	.25370000+04	.25285000+04	.25200000+04
.26210000+04	.26125000+04	.26040000+04	.25955000+04	.25869999+04	.25785000+04	.25700000+04	.25614999+04
.26625000+04	.26539999+04	.26455000+04	.26369999+04	.26285000+04	.26199999+04	.26115000+04	.26030000+04
.27040000+04	.26955000+04	.26869999+04	.26785000+04	.26699999+04	.26615000+04	.26529999+04	.26445000+04
.27454999+04	.27370000+04	.27285000+04	.27200000+04	.27114999+04	.27029999+04	.26944999+04	.26859999+04
.27869999+04	.27784999+04	.27700000+04	.27615000+04	.27530000+04	.27444999+04	.27359999+04	.27274999+04
.28285000+04	.28199999+04	.28114999+04	.28030000+04	.27945000+04	.27859999+04	.27775000+04	.27689999+04
.28700000+04	.28614999+04	.28529999+04	.28444999+04	.28360000+04	.28275000+04	.28189999+04	.28105000+04
.29115000+04	.29030000+04	.28944999+04	.28859999+04	.28774999+04	.28690000+04	.28605000+04	.28519999+04

3021,0057,0060,DUNG STRING MA

TABULATION

COLONNE17	COLONNE18	COLONNE19	COLONNE20	COLONNE21	COLONNE22	COLONNE23
-.10300000+03	-.11150000+03	-.12000000+03	-.12850000+03	-.13700000+03	-.14550000+03	-.15400000+03
-.61499999+02	-.69999999+02	-.78499999+02	-.86999998+02	-.95499999+02	-.10400000+03	-.11250000+03
-.20000000+02	-.28500000+02	-.37000000+02	-.45499999+02	-.53999999+02	-.62500000+02	-.70999999+02
.21500000+02	.13000000+02	.45000000+01	-.40000000+01	-.12500000+02	-.21000000+02	-.29500000+02
.63000000+02	.54499999+02	.45999999+02	.37500000+02	.29000000+02	.20500000+02	.12000000+02
.10450000+03	.95999998+02	.87500000+02	.78999999+02	.70499999+02	.62000000+02	.53500000+02
.14600000+03	.13750000+03	.12900000+03	.12050000+03	.11200000+03	.10350000+03	.94999999+02
.18750000+03	.17900000+03	.17050000+03	.16200000+03	.15350000+03	.14500000+03	.13650000+03
.22899999+03	.22050000+03	.21200000+03	.20350000+03	.19500000+03	.18650000+03	.17800000+03
.27050000+03	.26199999+03	.25350000+03	.24500000+03	.23650000+03	.22800000+03	.21950000+03
.31200000+03	.30350000+03	.29500000+03	.28650000+03	.27800000+03	.26949999+03	.26100000+03
.35349999+03	.34500000+03	.33649999+03	.32799999+03	.31950000+03	.31100000+03	.30249999+03
.39500000+03	.38649999+03	.37799999+03	.36950000+03	.36100000+03	.35250000+03	.34399999+03
.43649999+03	.42800000+03	.41949999+03	.41099999+03	.40250000+03	.39399999+03	.38550000+03
.47800000+03	.46949999+03	.46100000+03	.45250000+03	.44399999+03	.43550000+03	.42700000+03
.51949999+03	.51099999+03	.50250000+03	.49399999+03	.48549999+03	.47700000+03	.46849999+03
.56099999+03	.55249999+03	.54399999+03	.53549999+03	.52700000+03	.51849999+03	.50999999+03
.60249999+03	.59399999+03	.58549999+03	.57699999+03	.56849999+03	.55999999+03	.55149999+03
.64399999+03	.63549999+03	.62700000+03	.61849999+03	.60999999+03	.60149999+03	.59299999+03
.68549998+03	.67699999+03	.66849999+03	.66000000+03	.65149999+03	.64299998+03	.63450000+03
.72699999+03	.71849998+03	.70999999+03	.70149999+03	.69299998+03	.68449999+03	.67599999+03
.76849999+03	.75999999+03	.75149998+03	.74299998+03	.73449999+03	.72599998+03	.71749999+03
.80999998+03	.80149998+03	.79299998+03	.78449999+03	.77599999+03	.76749999+03	.75899998+03
.85149998+03	.84299997+03	.83449998+03	.82599998+03	.81749999+03	.80899998+03	.80049998+03
.89299997+03	.88449998+03	.87599998+03	.86749999+03	.85899998+03	.85049998+03	.84199999+03
.93449998+03	.92599998+03	.91749998+03	.90899998+03	.90049998+03	.89199998+03	.88349998+03
.97599998+03	.96749999+03	.95899999+03	.95049998+03	.94199999+03	.93349999+03	.92499999+03
.10175000+04	.10090000+04	.10005000+04	.99199999+03	.98349998+03	.97499999+03	.96649999+03
.10590000+04	.10505000+04	.10420000+04	.10335000+04	.10250000+04	.10165000+04	.10080000+04
.11005000+04	.10920000+04	.10835000+04	.10750000+04	.10665000+04	.10580000+04	.10495000+04
.11420000+04	.11335000+04	.11250000+04	.11165000+04	.11080000+04	.10995000+04	.10910000+04
.11835000+04	.11750000+04	.11665000+04	.11580000+04	.11495000+04	.11410000+04	.11325000+04
.12250000+04	.12165000+04	.12080000+04	.11995000+04	.11910000+04	.11825000+04	.11740000+04
.12665000+04	.12580000+04	.12495000+04	.12410000+04	.12325000+04	.12240000+04	.12155000+04
.13080000+04	.12995000+04	.12910000+04	.12825000+04	.12740000+04	.12655000+04	.12570000+04
.13495000+04	.13410000+04	.13325000+04	.13240000+04	.13155000+04	.13070000+04	.12985000+04
.13910000+04	.13825000+04	.13740000+04	.13655000+04	.13570000+04	.13485000+04	.13400000+04
.14325000+04	.14240000+04	.14155000+04	.14070000+04	.13985000+04	.13900000+04	.13815000+04
.14740000+04	.14655000+04	.14570000+04	.14485000+04	.14400000+04	.14315000+04	.14230000+04
.15155000+04	.15070000+04	.14985000+04	.14900000+04	.14815000+04	.14730000+04	.14645000+04
.15570000+04	.15485000+04	.15400000+04	.15315000+04	.15230000+04	.15145000+04	.15060000+04
.15985000+04	.15900000+04	.15815000+04	.15730000+04	.15645000+04	.15560000+04	.15475000+04
.16400000+04	.16315000+04	.16230000+04	.16145000+04	.16060000+04	.15975000+04	.15890000+04
.16815000+04	.16730000+04	.16645000+04	.16559999+04	.16475000+04	.16390000+04	.16305000+04
.17230000+04	.17145000+04	.17060000+04	.16975000+04	.16890000+04	.16805000+04	.16720000+04
.17645000+04	.17560000+04	.17475000+04	.17390000+04	.17305000+04	.17220000+04	.17134999+04
.18059999+04	.17975000+04	.17890000+04	.17805000+04	.17720000+04	.17635000+04	.17550000+04
.18475000+04	.18390000+04	.18305000+04	.18220000+04	.18135000+04	.18050000+04	.17965000+04

.18890000+04	.18805000+04	.18720000+04	.18635000+04	.18550000+04	.18465000+04	.18380000+04
.19305000+04	.19220000+04	.19135000+04	.19050000+04	.18965000+04	.18880000+04	.18795000+04
.19720000+04	.19635000+04	.19550000+04	.19465000+04	.19380000+04	.19295000+04	.19210000+04
.20135000+04	.20050000+04	.19965000+04	.19880000+04	.19795000+04	.19710000+04	.19625000+04
.20550000+04	.20465000+04	.20380000+04	.20295000+04	.20210000+04	.20125000+04	.20040000+04
.20965000+04	.20880000+04	.20795000+04	.20710000+04	.20625000+04	.20540000+04	.20455000+04
.21380000+04	.21295000+04	.21210000+04	.21125000+04	.21040000+04	.20955000+04	.20870000+04
.21795000+04	.21710000+04	.21625000+04	.21540000+04	.21455000+04	.21370000+04	.21285000+04
.22210000+04	.22125000+04	.22040000+04	.21955000+04	.21870000+04	.21785000+04	.21699999+04
.22625000+04	.22540000+04	.22455000+04	.22370000+04	.22285000+04	.22200000+04	.22115000+04
.23040000+04	.22955000+04	.22870000+04	.22785000+04	.22700000+04	.22615000+04	.22530000+04
.23455000+04	.23370000+04	.23285000+04	.23200000+04	.23114999+04	.23030000+04	.22944999+04
.23869999+04	.23785000+04	.23700000+04	.23615000+04	.23530000+04	.23444999+04	.23360000+04
.24285000+04	.24199999+04	.24115000+04	.24030000+04	.23944999+04	.23859999+04	.23775000+04
.24700000+04	.24614999+04	.24529999+04	.24445000+04	.24360000+04	.24275000+04	.24190000+04
.25115000+04	.25029999+04	.24944999+04	.24859999+04	.24775000+04	.24690000+04	.24605000+04
.25530000+04	.25444999+04	.25359999+04	.25274999+04	.25189999+04	.25105000+04	.25020000+04
.25944999+04	.25860000+04	.25775000+04	.25689999+04	.25604999+04	.25519999+04	.25434999+04
.26359999+04	.26275000+04	.26189999+04	.26105000+04	.26019999+04	.25934999+04	.25849999+04
.26774999+04	.26689999+04	.26605000+04	.26519999+04	.26435000+04	.26350000+04	.26264999+04
.27189999+04	.27104999+04	.27019999+04	.26934999+04	.26850000+04	.26764999+04	.26680000+04
.27604999+04	.27519999+04	.27434999+04	.27350000+04	.27264999+04	.27180000+04	.27094999+04
.28019999+04	.27934999+04	.27849999+04	.27764999+04	.27680000+04	.27594999+04	.27509999+04
.28434999+04	.28350000+04	.28264999+04	.28179999+04	.28094999+04	.28010000+04	.27925000+04

11 HEURES 03 MINUTES 11 SECONDES

```
si  Q = N  alors  allera  FIN DE MATRICE REELLE  ;
PAGE SUIVANTE ;
commentaire  : Cette instruction comme son nom l'indique sert à res-
taurer la page après chaque tranche d'impression ;
  fin  ;
FIN DE MATRICE REELLE :
fin ;
```

10 -

```
procedure  COURBE FONCTION  (FONCTION, VALINI, VALTERMI, PAS, HAUTEUR,
                                                          TABULATION) ;
   valeur  VALINI, VALTERMI, PAS, HAUTEUR ;
   réel  VALINI, VALTERMI, PAS ; entier  HAUTEUR ;
   booleen  TABULATION ; reel procedure  FONCTION ;
   debut commentaire :
```

COURBE FONCTION trace sur papier la courbe d'une fonction à une variable déclarée comme réelle procédure dans le programme. Le nom de cette procédure doit figurer à la place du premier paramètre FONCTION.

Le deuxième paramètre VALINI désigne la valeur initiale de la variable.

Le troisième paramètre VALTERMI désigne la valeur terminale de la variable.

HAUTEUR est le nombre total d'espaces à l'intérieur desquels la courbe est cadrée (pour l'axe Y). HAUTEUR doit-être inférieur à 120.

Si VALINI < 0 < VALTERMI, la position de zéro est indiquée sur la courbe. Les valeurs maximale et minimale de la fonction pour l'intervalle considéré sont calculées et imprimées au début de de la page avant que la courbe soit tracée.

Le quatrième paramètre TABULATION aura pour valeur vrai si on veut avoir une tabulation de la fonction et de sa variable , et faux dans le cas contraire.

Exemple :

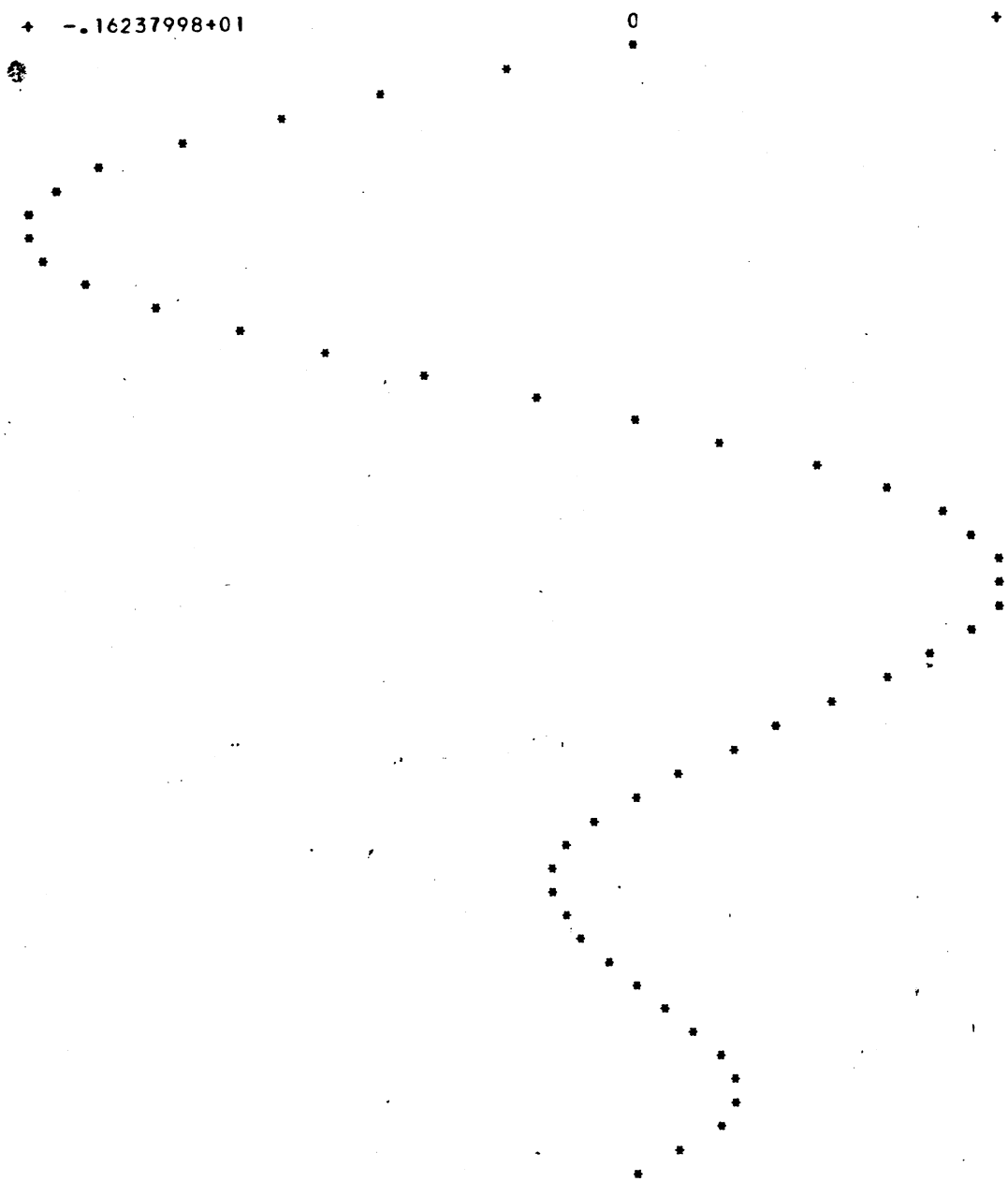
On suppose que la procédure COURBE FONCTION a été déclarée, la séquence d'instructions suivante tracera une courbe déclarée comme procédure FIC et donnera une tabulation de la fonction et de sa variable :

```
debut reel procedure FIC (x) ; reel X ;
          FIC := (X ↑ 2 - 1) x (X - 2) X SIN (2 x 3.1416 x X/3) ;
      COURBE FONCTION (FIC, -1.0, 2.0, 0.0625, 70, vrai)
fin ;          ( voir résultat ci-contre)

entier M , ESPACE ; reel MIN, MAX, INTERVALLE ;
   M := ABS (ENTIER ((VALINI - VALTERMI)/(PAS)) +1 ;
 debut entier   I ; tableau X, Y [1:M] ;
   X[1]:= VALINI ;
   Y[1]:= MIN := MAX := FONCTION (VALINI) ;
   pour I := 2 pas 1 jusqua M faire
 debut
   X [I] := X[I-1]+ PAS ;
   Y [I] := FONCTION (X [I]) ;
   si MIN > Y [I] alors MIN := Y[I] ;
   si MAX < Y[I] alors MAX := Y[I] ;
 fin du calcul de l'intervalle de cadrage ;
   si TABULATION alors
 debut
       ECRIRE CHAINE (2,'      TABULATION') ;
       ALALIGNE ; ALALIGNE ; ALALIGNE ;
   pour I := 1 pas 1 jusqua M faire
       ECRIRE ('X u=u', X[I] ,,,,'Yu=u', Y[I]) ;
   SAUT PAGE
 fin de la tabulation de la fonction et de sa variable ;
INTERVALLE := MAX - MIN ; ALALIGNE ;
ECRIRE SYMBOLE (2, + , 1) ; ECRIRE REEL (2, MIN) ;
si MIN < 0 ∧ 0 < MAX alors
 debut
   NBLANC(ABS(ENTIER((Y[I]-MIN)xHAUTEUR/INTERVALLE))) ;
   ECRIRE SYMBOLE (2,'*', 1) ; ALALIGNE
```

T A B U L A T I O N

X =	Y =
-.10000000+01	0
-.93750000+00	-.32863523+00
-.87500000+00	-.65086723+00
-.81250000+00	-.94763296+00
-.75000000+00	-.12031250+01
-.68750000+00	-.14051123+01
-.62500000+00	-.15451052+01
-.56250000+00	-.16183699+01
-.50000000+00	-.16237999+01
-.43750000+00	-.15636601+01
-.37500000+00	-.14432186+01
-.31250000+00	-.12702867+01
-.25000000+00	-.10546897+01
-.18750000+00	-.80769179+00
-.12500000+00	-.54139809+00
-.62499999-01	-.26815929+00
0	0
.62499999-01	.25190721+00
.12500000+00	.47770420+00
.18750000+00	.66923034+00
.25000000+00	.82031422+00
.31250000+00	.92696595+00
.37500000+00	.98746538+00
.43750000+00	.10023462+01
.50000000+00	.97427994+00
.56250000+00	.90786604+00
.62500000+00	.80934085+00
.68750000+00	.68621763+00
.75000000+00	.54687499+00
.81250000+00	.40011169+00
.87500000+00	.25468718+00
.93750000+00	.11886806+00
.10000000+01	0
.10625000+01	-.95876058-01
.11250000+01	-.16434618+00
.11875000+01	-.20286939+00
.12500000+01	-.21093526+00
.13125000+01	-.19012421+00
.13750000+01	-.14406558+00
.14375000+01	-.78292412-01
.15000000+01	.45984051-05
.15625000+01	.82316573-01
.16250000+01	.15923908+00
.16875000+01	.22096298+00
.17500000+01	.25781632+00
.18125000+01	.26083706+00
.18750000+01	.22235397+00
.19375000+01	.13655228+00
.20000000+01	0



11 HEURES 43 MINUTES 32 SECONDES

```
  fin
 fin
fin  DE COURBE FONCTION ;

procedure  NBLANC  (N) ; valeur  N  ; entier N  ;
  début commentaire
cette procédure écrit sur papier  N  blancs, autrement dit elle saute
N  espaces sur une ligne ;
entier  I ;
pour  I := 1  pas  1  jusqua  N  faire
         ECRIRE SYMBOLE  (2, 'u', 1)
  fin  DE  NBLANC  ;

11 -
procedure  COURBE 7 (X, Y1, Y2, Y3, Y4, Y5, Y6, Y7, N, HAUTEUR,
                                          TABULATION, CHAINE)  ;
  valeur N, HAUTEUR ; entier N, HAUTEUR  ;
  booleen  TABULATION  ; chaîne  CHAINE  ;
  tableau  X, Y1, Y2, Y3, Y4, Y5, Y6, Y7  ;
  debut commentaire
```

Nous présentons ici une procédure qui trace sur la même feuille de papier les graphes de sept fonctions (à une variable) dont les valeurs par pas réguliers sont stockées dans sept tableaux différents.

X	désigne le tableau où les valeurs de la variable sont stockées.		
Y1	désigne le tableau où les valeurs de la 1ère fonction sont stockées.		
Y2	"	"	2ème fonction "
⋮	⋮	⋮	⋮
Y7	"	"	7e fonction "

N désigne la dimension de ces tableaux, ou le nombre de points de chaque courbe.

HAUTEUR le nombre d'espaces à l'intérieur desquels les courbes sont cadrées.

TABULATION aura pour valeur vrai si on veut avoir une tabulation

de ces sept fonctions et de leur variable X, faux dans le cas contraire.

CHAINE sert de dictionnaire pour le dessin de la courbe.

Exemple :

si CHAINE = '1234567', la première courbe sera tracée avec le symbole 1 ; la deuxième avec 2, ... , la 7e avec 7.

alors que si CHAINE = 'ABCDEFG'
la première courbe sera tracée avec le symbole A, la deuxième avec B .... la 7e avec G.

La présentation est donc laissée au désir de l'utilisateur.

Regardons de plus près l'algorithme de construction de la procédure COURBE 7.

Tout d'abord il est à remarquer que cet algorithme est général et peut théoriquement servir à construire des procédures qui tracent un nombre n quelconque de courbes sur une même feuille de papier, pratiquement le nombre n ne doit pas dépasser une certaine limite qui dépend du nombre de positions disponibles sur une ligne d'impression.

L'utilisateur, au besoin peut facilement construire lui-même des procédures COURBE 2, COURBE 3, ... pour tracer deux courbes, troi courbes ... , en modifiant quelques détails dans COURBE 7.

Pour tracer plusieurs courbes, en même temps, il faut calculer les positions où on doit mettre le symbole correspondant à chaque courbe. Mais sur une ligne, dès qu'on met un symbole quelque part il est impossible de revenir en arrière. Il faut donc classer les valeurs correspondantes aux différentes courbes pour savoir laquelle il faut tracer d'abord, et en même temps garder les informations qui permettent de déterminer le symbole convenable à mettre dans une position donnée. Ceci est réalisé par la procédure auxiliaire MINMAXIND dont le fonctionnement sera expliqué en commentaire lors de la présentation de cette procédure.

Exemple :

La séquence d'instructions suivantes produira comme résultat, 7 courbes tracées avec le symbole A, B, C, D, E, F, G.

```
début tableau X, Y1, Y2, Y3, Y4, Y5, Y6, Y7[1 : 25] ;
  entier I ; pour I := 1 pas 1 jusqua 25 faire
```

X	Y1	Y2	Y3	Y4	Y5	Y6	Y7
-.30000000+01	.12340980-03	.11108996-01	.10539922+00	.32465247+00	.56978284+00	.75483961+00	.86881506+00
-.27500000+01	.51957469-03	.22794181-01	.15097742+00	.38855813+00	.62334431+00	.78952157+00	.88855027+00
-.25000000+01	.19304541-02	.43936934-01	.20961139+00	.45783336+00	.67663385+00	.82257756+00	.90696062+00
-.22500000+01	.63297154-02	.79559510-01	.28206295+00	.53109599+00	.72876334+00	.85367637+00	.92394608+00
-.20000000+01	.18315639-01	.13533528+00	.36787944+00	.60653067+00	.77880079+00	.88249691+00	.93941307+00
-.17500000+01	.46770623-01	.21626517+00	.46504319+00	.68194076+00	.82579705+00	.90873377+00	.95327529+00
-.15000000+01	.10539922+00	.32465247+00	.56978284+00	.75483961+00	.86881506+00	.93210250+00	.96545457+00
-.12500000+01	.20961139+00	.45783336+00	.67663385+00	.82257756+00	.90696062+00	.95234480+00	.97588156+00
-.10000000+01	.36787944+00	.60653067+00	.77880079+00	.88249691+00	.93941307+00	.96923324+00	.98449644+00
-.75000000+00	.56978284+00	.75483961+00	.86881506+00	.93210250+00	.96545457+00	.98257548+00	.99124945+00
-.50000000+00	.77880079+00	.88249691+00	.93941307+00	.96923324+00	.98449644+00	.99221795+00	.99610138+00
-.25000000+00	.93941307+00	.96923324+00	.98449644+00	.99221795+00	.99610138+00	.99804878+00	.99902391+00
0	.10000000+01	.10000000+01	.10000000+01	.10000000+01	.10000000+01	.10000000+01	.10000000+01
.25000000+00	.93941307+00	.96923324+00	.98449644+00	.99221795+00	.99610138+00	.99804878+00	.99902391+00
.50000000+00	.77880079+00	.88249691+00	.93941307+00	.96923324+00	.98449644+00	.99221795+00	.99610138+00
.75000000+00	.56978284+00	.75483961+00	.86881506+00	.93210250+00	.96545457+00	.98257548+00	.99124945+00
.10000000+01	.36787944+00	.60653067+00	.77880079+00	.88249691+00	.93941307+00	.96923324+00	.98449644+00
.12500000+01	.20961139+00	.45783336+00	.67663385+00	.82257756+00	.90696062+00	.95234480+00	.97588156+00
.15000000+01	.10539922+00	.32465247+00	.56978284+00	.75483961+00	.86881506+00	.93210250+00	.96545457+00
.17500000+01	.46770623-01	.21626517+00	.46504319+00	.68194076+00	.82579705+00	.90873377+00	.95327529+00
.20000000+01	.18315639-01	.13533528+00	.36787944+00	.60653067+00	.77880079+00	.88249691+00	.93941307+00
.22500000+01	.63297154-02	.79559510-01	.28206295+00	.53109599+00	.72876334+00	.85367637+00	.92394608+00
.25000000+01	.19304541-02	.43936934-01	.20961139+00	.45783336+00	.67663385+00	.82257756+00	.90696062+00
.27500000+01	.51957469-03	.22794181-01	.15097742+00	.38855813+00	.62334431+00	.78952157+00	.88855027+00
.30000000+01	.12340980-03	.11108996-01	.10539922+00	.32465247+00	.56978284+00	.75483961+00	.86881506+00

3021,0057,0075,DUNG STRING MA

+ .12340980-03

+ .99999999+00

```
  debut
    si  I = 1  alors  X[I] := -3.0  sinon  X[I] := X[I-1]+0.25 ;
    Y1[I] := - EXP (-X[I]↑ 2) ; Y2[I] := EXP (-X[I]↑2/2) ;
    Y3[I] := EXP (-X[I]↑2/4) ; Y4[I] := EXP (-X[I]↑2/8) ;
    Y5[I] := EXP (-X[I]↑/16) ; Y6[I] := EXP (-X[I]↑2/32) ;
    Y7[I] := EXP (-X[I]↑64)
  fin
COURBE 7 (X, Y1, Y2, Y3, Y4, Y5, Y6, Y7, 25, 100, vrai, 'ABCDEFG')
  fin ;                    (voir résultat ci-contre)
  debut entier  I, J, DIFFER ; reel  MIN, MAX, INTERVALLE ;
  tableau  TAMPON  [1 : 7 x N] ; entier tableau  INDICE [1 : 7] ;
  pour  I := 1  pas  1  jusqua  N  faire
  debut
        TAMPON [I] := Y1[I] ;
        TAMPON [I+N] := Y2 [I] ;
        TAMPON [I+2xN]:= Y3[I] ;
        TAMPON [I+3xN]:= Y4[I] ;
        TAMPON [I+4xN] := Y5[I] ;
        TAMPON [I+5xN] := Y6[I] ;
        TAMPON [I+6xN] := Y7[I] ;
  fin
MINMAX  (TAMPON, 7 x N) ;
MAX  :=  TAMPON [7 x N] ;
MIN  := TAMPON [1] ;
```

commentaire : La séquence d'instructions précédentes sert à calculer le maximum et le minimum de la fonction dans l'intervalle où on veut tracer la courbe, ce calcul est nécessaire pour pouvoir cadrer la courbe ;

```
INTERVALLE  :=  MAX - MIN ; ALALIGNE ;
ECRIRE SYMBOLE  (2, +  , 1) ; ECRIRE REEL  (2, MIN) ;
NBLANC (HAUTEUR - 16) ; ECRIRE SYMBOLE  (2, '+', 1) ;
ECRIRE REEL (2, MAX) ;
pour  I := 1  pas  1  jusqua  N  faire
  debut
```

```
          ALALIGNE ; ALALIGNE ;
          TAMPON [1] := Y1[I] ;
          TAMPON [2] := Y2[I] ;
          TAMPON [3] := Y3[1] ;
          TAMPON [4] := Y4[I] ;
          TAMPON [5] := Y5[I] ;
          TAMPON [6] := Y6[I] ;
          TAMPON [7] := Y7[I] ;
          MIN MAXIND (TAMPON, INDICE, 7) ;
          NBLANC (ENTIER((TAMPON[1]-MIN) x HAUTEUR/INTERVALLE + 0.5))
          ECRIRE SYMBOLE (2, CHAINE, INDICE [1]) ;
pour J := 2 pas 1 jusqua 7 faire
  debut
          DIFFER := ENTIER ((TAMPON [J]- TAMPON [J-1] x HAUTEUR/
                                                    INTERVALLE + 0.5))
          si DIFFER < 1 alors TAMPON [J] := TAMPON [J-1] sinon
  debut
          NBLANC (DIFFER - 1) ;
          ECRIRE SYMBOLE (2, CHAINE, INDICE [J])
    fin
  fin
fin ;
si TABULATION alors
  debut
          SAUTPAGE ; ALALIGNE ;
          ECRIRE CHAINE (2,' X  Y1  Y2  Y3  Y4  Y5  Y6  Y7 ') ;
commentaire : La chaîne, 2ème paramètre de l'instruction ECRIRE
CHAINE précédente sert à imprimer les désignations des colonnes de l
tabulation de la variable X et des sept fonctions considérées ;
ALALIGNE ; ALALIGNE ; ALALIGNE ;
pour I := 1 pas 1 jusqua N faire
  debut
    pour MAX := X[I], Y1[I], Y2[I], Y3[I], Y4[I], Y5[I], Y6[I] , Y
                                                          fair
          ECRIRE REEL (2, MAX) ;
    ALALIGNE
```

```
        fin ;
    fin de TABULATION ;
fin DE COURBE 7 ;
procedure MIN MAXIND (TABLEAU, INDICE, N) ; valeur N ;
entier N ; tableau TABLEAU ; entier tableau INDICE ;
  debut commentaire : cette procedure classe les valeurs des éléments
du tableau TABLEAU par ordre croissant, et en même temps met dans le
tableau INDICE des entiers qui correspondent aux places qu'occupaient
les éléments de TABLEAU avant que celui-ci soit manipulé par
MINMAXIND ;
tableau TAMPON [1 : N]; entier I, J ; réel INTER ;
    pour I := 1 pas 1 jusqua N faire
        TAMPON [I] := TABLEAU [I] ;
    BOUCLE :
    pour I := 2 pas 1 jusqua N faire
        si TABLEAU [I] < TABLEAU [I - 1] alors
                        debut
                          INTER := TABLEAU [I - 1] ;
                          TABEAU [I - 1] := TABLEAU [I] ;
                          TABEAU [I] := INTER ;
                          aller a BOUCLE
                        fin ;
  pour I := 1 pas 1 jusqua N faire
    pour J := 1 pas 1 jusqua N faire
      si TAMPON [I] = TABLEAU [J] alors
                        INDICE [J] := I
fin procedure MINMAXIND ;
```

II - ETUDE CRITIQUE DU TRAITEMENT DES CHAINES EN ALGOL

Le langage ALGOL conçu initialement pour résoudre des problèmes numériques tel qu'il est défini dans le rapport de base, ne fait aucune référence à la manipulation des symboles. Aucune opération sur les chaînes n'est définie ni mentionnée. Les seules chaînes qu'on peut avoir dans un programme ALGOL doivent être des paramètres de procédures. La structure imbriquée des chaînes fait penser à celle des listes en LISP. Elle donne l'impression que les auteurs du langage ont voulu laisser une porte ouverte à une élaboration future d'instruments adéquats pour le traitement non numérique des informations.

Différentes tentatives dans cette voie sont faites, nous allons faire une étude de celles de G. SOLLIN, N. WIRTH, VAN WIJNGAARDEN et LAARSCHOT-NEDER KOORN.

II - 1 PROPOSITION DE G. SOLLIN.

Le but de cette proposition (Référence 19) est d'étudier le moyen de faire identifier par ALGOL une information quelconque et son affectation sans assurer aucun traitement algorithmique sur cette information.

L'information se trouvant dans un calculateur est en général quelconque et représente soit :

- le programme
- les données et résultats.

Du fait de sa nature, cette information se présente sous la forme de suites de longueurs quelconques que nous devons repérer :

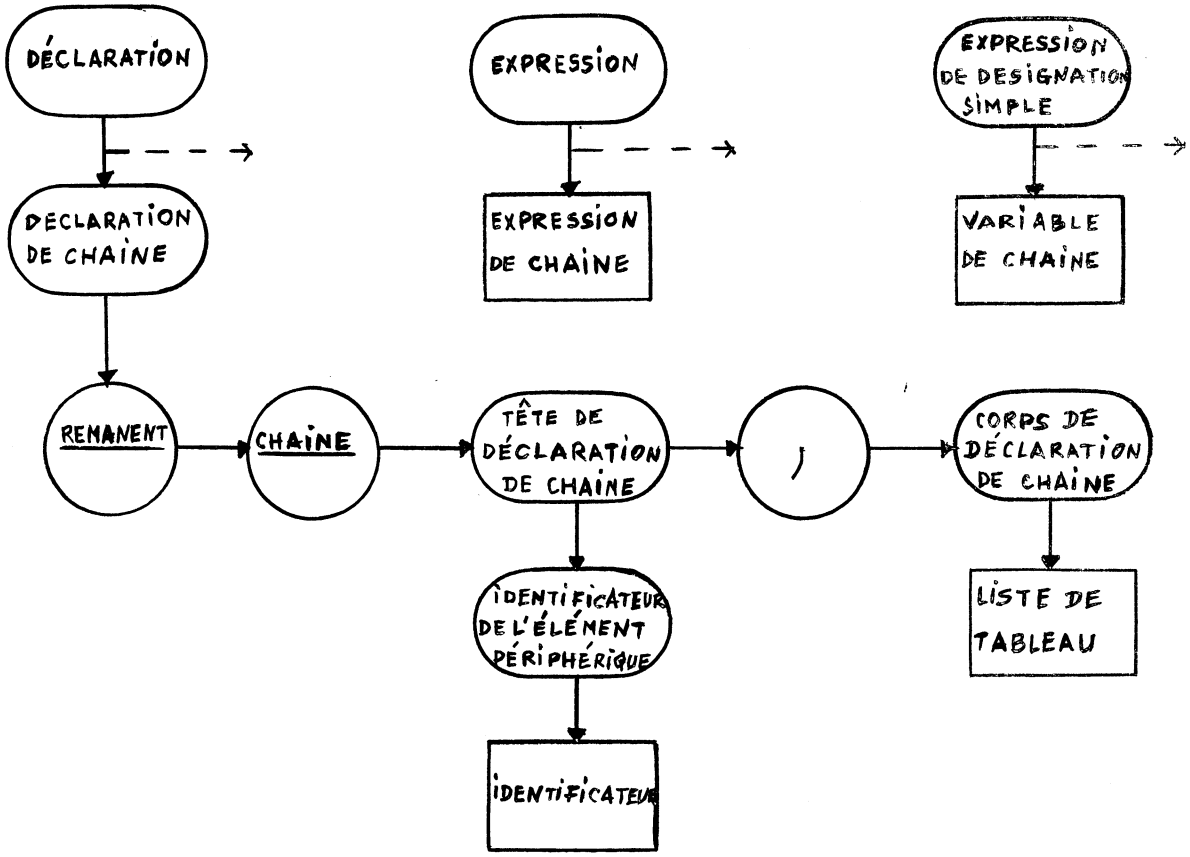
- en position par rapport aux supports du calculateur
- en dimension

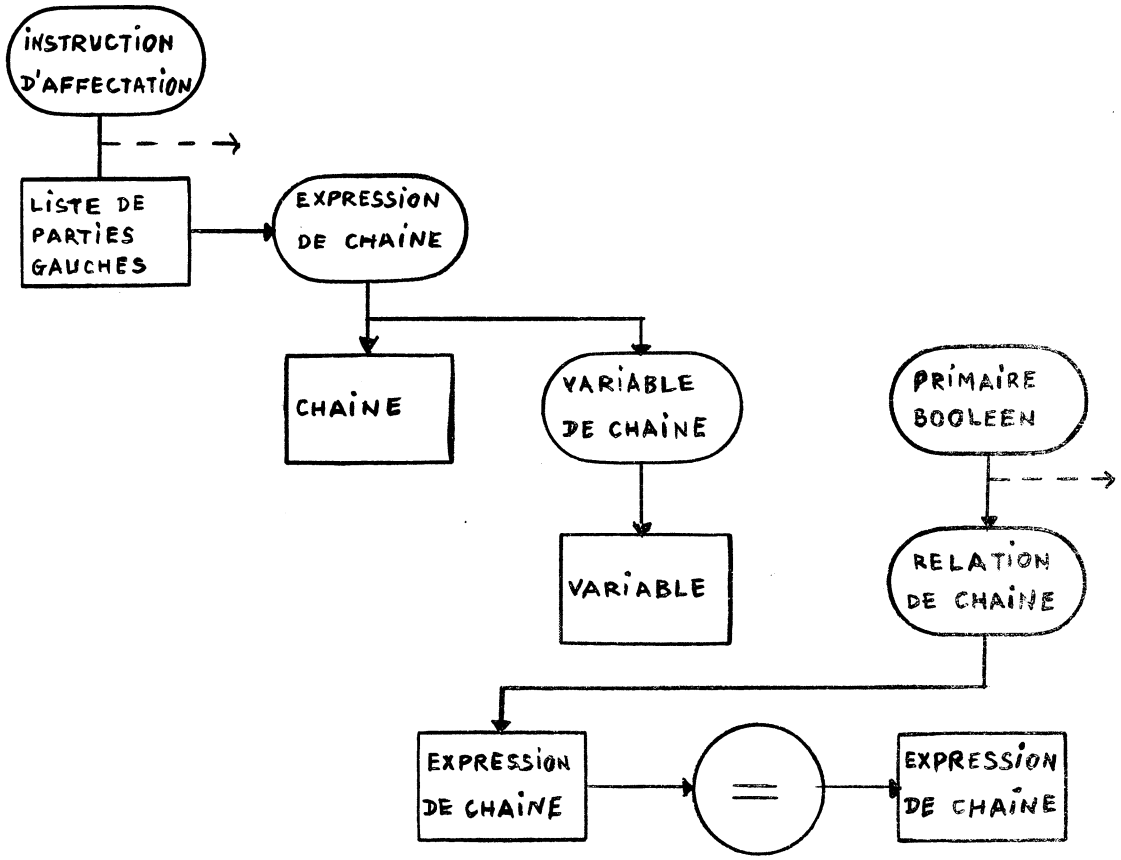
Le repérage de la position devra tenir compte des traitements à l'extérieur du calculateur, c'est à dire repérer les moyens d'entrée-sortie. Il devra également indiquer aussi bien l'adresse d'une donnée ou d'un résultat pour son traitement que celle d'un programme objet en vue de son exécution, puisque jusqu'au moment de l'utilisation de l'information en tant que programme, données ou résultats, elle est d'une nature unique qui est celle de toute suite de symboles identifiables.

Autrement dit, l'identificateur se rapportant à une telle information devra également être une étiquette.

Il faut remarquer que cette notion d'information a bien été considérée par ALGOL sous la forme de CHAINE.

L'auteur propose donc quelques adjonctions au langage qui se traduisent syntaxiquement par :


DÉCLARATION
DÉCLARATION DE CHAINE
REMANENT
CHAINE
TÊTE DE DÉCLARATION DE CHAINE
IDENTIFICATEUR DE L'ÉLÉMENT PÉRIPHÉRIQUE
IDENTIFICATEUR
)
CORPS DE DÉCLARATION DE CHAINE
LISTE DE TABLEAU
EXPRESSION
EXPRESSION DE CHAINE
EXPRESSION DE DESIGNATION SIMPLE
VARIABLE DE CHAINE


INSTRUCTION D'AFFECTATION
LISTE DE PARTIES GAUCHES
EXPRESSION DE CHAINE
CHAINE
VARIABLE DE CHAINE
VARIABLE
PRIMAIRE BOOLEEN
RELATION DE CHAINE
EXPRESSION DE CHAINE
=
EXPRESSION DE CHAINE

En ce qui concerne la sémantique, il faut préciser les points suivants :

VARIABLE DE CHAINE : est soit une variable ordinaire soit une variable déclarée chaîne ; dans le dernier cas les indices précisent le premier élément de la chaîne considérée. Par exemple : une chaîne déclarée par un tableau L [1 : N] peut donner naissance à N chaînes telles que

L [1], L [2], ... L [N]

L [1] ayant N éléments

L [P] ayant N - P + 1 éléments

INSTRUCTION D'AFFECTATION (de chaînes) :

l'affectation de chaînes se fait uniquement suivant les dimensions des chaînes ou valeurs déclarées de la façon suivante :

- si la variable de la partie gauche a une dimension supérieure à l'expression de chaîne de la partie droite, cette dernière est affectée en totalité sans changer les valeurs des éléments excédentaires de la variable de chaîne de la partie gauche.

- si la variable de la partie gauche a une dimension inférieure à l'expression de chaîne de la partie droite, cette dernière est affectée partiellement à partir du premier élément jusqu'à l'épuisement des positions déclarées pour la variable de chaîne de la partie gauche.

Exemple : le programme suivant

```
début chaîne  MC, MA [ 1 : 4 ] , MZ [1 : 50];
chaîne 1, X1, X2, X3, X4, X5 [ 1 : 80] ; chaîne 2, R [1: 240, 1: 100];
chaîne 3, S [1: 150] ; chaîne 4, T [1: 50] ; entier P, V1, V2, W;
pour P:= 1 pas 1 jusqua 100 faire
  début
    R [1,P] := X1 [ 1 ] ; R [80,P] := X2 [1]; R [160, P]:= X3 [1];
    V1:= MA [ 1 ] := X4 [12] ; V2:= MA [ 1] := X5 [ 12 ] ;
    début ⟨ traitement ⟩ fin ;
    MZ [1]:=  S [1] ; MZ [18] := W;
    T [ 1 ] :=  MZ [ 1 ]
  fin
fin
```

effectue 100 fois

- lire 5 cartes sur lecteur 1.
- ranger 3 cartes sur ruban 2.
- identifier sur les deux cartes restantes 2 valeurs arithmétiques V1 et V2 se trouvant à partir du 12e caractère de chaque carte sur 4 caractères.
- effectuer un traitement en mémoire centrale (MC)
- extraire les 50 premiers caractères d'une chaîne de 150 caractères sur ruban 3.
- * extraire les 50 premiers caractères sur l'imprimante 4.

DISCUSSION :

La proposition permet quelques facilités de traitement de chaînes, à savoir :

- l'introduction d'information à partir d'éléments périphériques d'un calculateur.
- l'extraction.
- à l'intérieur du calculateur, des affectations, sur une même chaîne, de chaînes distinctes ou de valeurs arithmétiques ou booléennes. Ce qui permettrait l'analyse des éléments.

En ce qui concerne ce dernier point, il reste à préciser la manière d'affecter une variable réelle, entière ou booléenne à une variable indicée déclarée chaîne et vice versa.

Cette proposition a le mérite de mettre en relief le rôle des unités d'entrée-sortie. Par contre, la forme de la déclaration des variables de chaînes n'est pas précise. En effet, il est tout à fait arbitraire d'imposer à l'avance une longueur fixe à une information, alors que dans la cas général cette longueur n'est pas connue. En outre lorsqu'on veut faire des substitutions de chaînes de longueurs inégales dans une chaîne, la forme indicée des variables de chaîne ne convient pas.

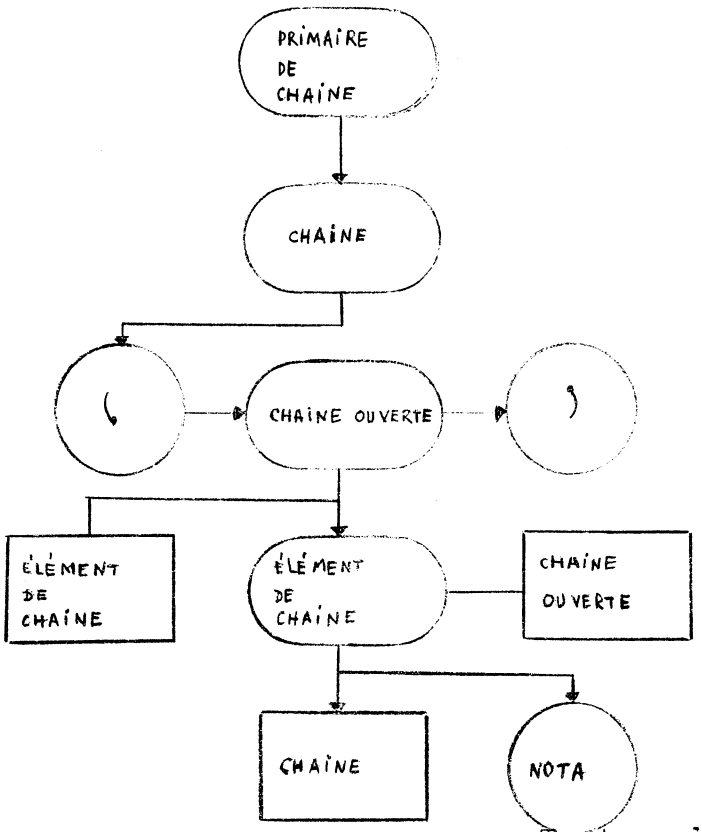
On peut donc considérer cette proposition, non comme une solution, mais une contribution à son étude.

* - porter sur ces caractères un résultat numérique W à partir du caractère 18.

II - 2 PROPOSITION DE N. WIRTH

Le but de cette proposition est de donner à ALGOL des instruments adéquats pour traiter des problèmes de manipulation de chaînes.

N. WIRTH pense que le travail ne peut être convenablement fait en ne définissant qu'un ensemble de fonctions standard. Il suggère donc l'introduction d'une nouvelle variable de type "chaîne" Cela n'implique pas que les affectations du genre ⟨variable de type chaîne⟩ := ⟨expression de chaîne⟩ soient permises. La syntaxe de chaîne serait définie comme suit :



Tout symbole distinct à l'exception de ' et de ?

Comme procédures de base, il y en a cinq qui sont : LENGTH, TYPE, PICK, SET, CUT :

1 - entier procédure LENGTH (CHAINE); chaîne CHAINE; commentaire :
la valeur de LENGTH est n, si S = ' e_1, e_2..... e_n '
où chaque e_i est un élément de chaîne.
Exemple : LENGTH (' A ') = 1
LENGTH (' X ' UV ' Y') = 3

2 - booleen procedure TYPE (SOURCE, CHAINE, I, DESTINATION);
entier I ; chaîne SOURCE, CHAINE, DESTINATION;
commentaire :
- si le premier élément de SOURCE est une chaîne, alors cete chaîne est affectée à DESTINATION. CHAINE et I ne changent pas de valeur.
- sinon on affecte à I le nombre correspondant à la position du premier élément de CHAINE qui est égal au premier élément de SOURCE (si un tel élément n'existe pas alors I prend la valeur 0) et TYPE devient faux. DESTINATION et CHAINE ne changent pas.
Exemple : TYPE (' A', 'ABC ' , I, U) est faux et I = 1
TYPE (' X12 ', ' ZYX ', I, U) est faux et I = 3
TYPE ('' X ' CD ', ' 1 ', I, U) est vrai et U = ' X '
TYPE (' B ', 'CD', I, U) est faux et I = 0

3 - procédure PICK (DESTINATION , SOURCE, I);valeur I;
entier I; chaîne DESTINATION, SOURCE ;
commentaire :
le $I^{ème}$ élément de SOURCE est affecté à la variable (de type chaîne) DESTINATION.
Exemple : PICK (X, ' ABC ' , 2) donne X = ' B '
PICK (Y, 'ABC'+ - */'', 4) donne Y = '' + - */ ''

4 - procedure SET (DESTINATION, I, SOURCE); valeur I ;
entier I; chaîne DESTINATION, SOURCE;
commentaire :
si SOURCE est la chaîne contenant les éléments e_j, j = 1, 2, ... n alors le $(I + j - 1)^{ième}$ élément de DESTINATION sera remplacé par e_j pour j = 1, 2, ..., n

Exemple :

si X = '1 2 3 4 5 6'
Y = 'A B C D E F'
S = 'P Q '+ -' R S'

alors

SET (S, 1, '␣') donne S = '␣Q '+ -' R S'
SET (S, 3, Y) donne S = 'P Q A B C D E F'
SET (X, 7, '7 '10''11'') donne X = '1 2 3 4 5 6 7' 10''11''

5 - procedure CUT (CHAINE, I); valeur I ; entier I;
chaîne CHAINE ;
commentaire :
Les éléments de CHAINE avec les indices supérieurs ou égaux à I sont supprimés.
Exemple : si X = '3 5''7' 81' 3'
alors CUT (X, 3) donne X = '35'

Les instructions procédure doivent être de la forme suivante :

LENGTH (< primaire de chaîne >)
TYPE (< primaire de chaine > , <primaire de chaîne> , <variable > ,<variable de chaîne>).
PICK (<variable de chaîne > ,<primaire de chaîne > , <expression arithmétique >)
SET (< variable de chaîne > ,<expression arithmétique> , <primaire de chaîne>)
CUT (<variable de chaîne > ,<expression arithmétique >)

DISCUSSION :

On dispose ainsi des instruments de base pour le traitement de chaînes. En effet, au moyen de PICK on peut accéder à tout élément d'une chaîne tandis que SET permet son remplacement. LENGTH et TYPE donnent des informations sur la longueur et le degré d'imbrication de la chaîne considérée.

Voici quelques exemples de procédures de traitement de chaînes. Elles sont construites à partir des procédures de base :

- procédure CONCATENER (X, Y) ; chaîne X, Y;
commentaire :
à X on affecte la concaténation de X et Y ;
SET (X, LENGTH (X) + 1, Y);
Exemple : si X = '12', Y = '34 'AB''
alors CONCATENER (X, Y) donne X = '1 2 3 4 'AB''
- procedure INSERER (DESTINATION, I, SOURCE); valeur I;
entier I; chaîne DESTINATION, SOURCE;
commentaire :
SOURCE sera insérée dans DESTINATION devant le I^{ème} élément de cette dernière chaîne ;

```
debut entier  J, L1, L2; chaîne  TAMPON 1, TAMPON 2;
    L1:= LENGTH (DESTINATION); L2:= LENGTH (SOURCE);
    SET (TAMPON 1, 1, DESTINATION);
    SET (TAMPON 1, I, SOURCE);
    pour J:= I pas 1 jusqua L2 faire
         debut
            PICK (TAMPON2,DESTINATION, J)
            SET (TAMPON1, J+L, TAMPON2)
         fin ;
    SET (DESTINATION, 1, TAMPON1)
fin ;
```

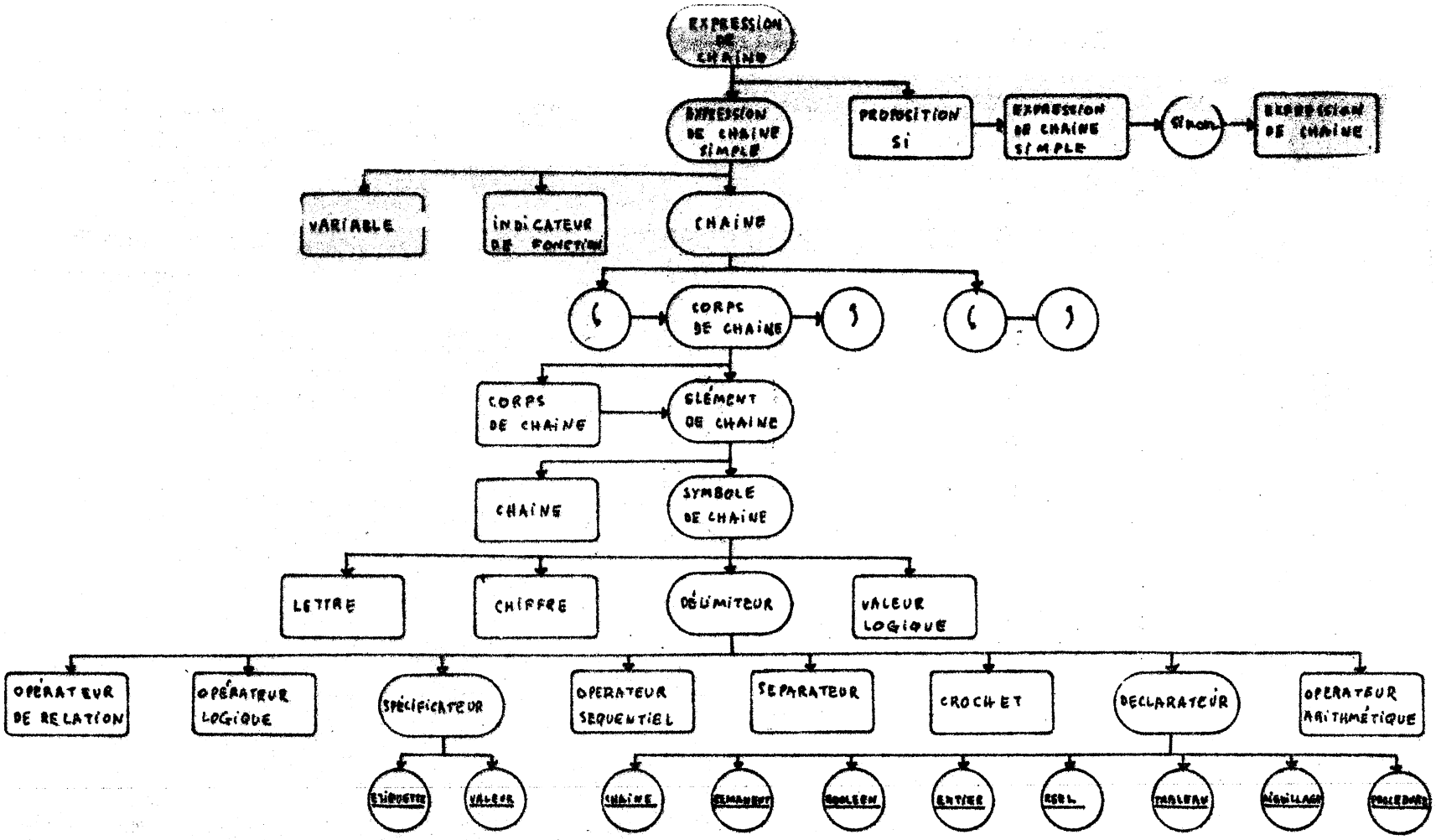
- procedure ABLATION (CHAINE, I, J); valeur I, J;
entier I, J ; chaine CHAINE ;
commentaire :
ABLATION supprime les éléments compris entre le I^{ème} et le J^{ième} élément de CHAINE;

```
debut entier  K, L ; chaîne U;
     l:= LENGTH (CHAINE);
     si I ⩽ J alors
         debut
           pour  K := J+1 pas  1 jusqua L faire
               debut
                 PICK (U, CHAINE, K);
                 SET ( CHAINE, I, U);
                 I:= I+1
               fin ;
```

```
        CUT (S,I)
      fin
fin ;
```

Nous voyons que cette proposition offre une possibilité d'incorporer à ALGOL des facilités de traitement de chaînes. Mais l'interdiction de faire des affectations de chaînes sur les variables de chaîne semble arbitraire. Pour ce qui est du nombre de procédures de base, on peut le réduire à quatre. En effet, il n'y a aucune difficulté pour construire CUT à partir de PICK, SET, TYPE, LENGTH :

```
- procedure CUT (CHAINE,I); chaîne CHAINE; entier I;
debut entier J; chaîne TAMPON1, TAMPON2;
      si I ⩽ LENGTH (CHAINE) alors
        début
          PICK (TAMPON1, CHAINE,1);
          pour J:= 2 pas 1 jusquà I-1 faire
            début
              PICK (TAMPON2, CHAINE, J);
              SET (TAMPON1, J, TAMPON2)
            fin ;
          PICK ( CHAINE, TAMPON1, 1);
          SET (CHAINE, 1, TAMPON1)
        fin ;
```


EXPRESSION DE CHAINE
EXPRESSION DE CHAINE SIMPLE
PROPOSITION SI
EXPRESSION DE CHAINE SIMPLE
sinon
EXPRESSION DE CHAINE
VARIABLE
INDICATEUR DE FONCTION
CHAINE
(
CORPS DE CHAINE
)
(
)
CORPS DE CHAINE
ELEMENT DE CHAINE
CHAINE
SYMBOLE DE CHAINE
LETTRE
CHIFFRE
DELIMITEUR
VALEUR LOGIQUE
OPERATEUR DE RELATION
OPERATEUR LOGIQUE
SPECIFICATEUR
OPERATEUR SEQUENTIEL
SEPARATEUR
CROCHET
DECLARATEUR
OPERATEUR ARITHMETIQUE
ETIQUETTE
VALEUR
CHAINE
ENTIER
REEL
TABLEAU

II - 3 PROPOSITION DE VAN WIJNGAARDEN

En vue de l'introduction des facilités de manipulation de chaînes, Van Wijngaarden a proposé une extension du langage ALGOL. Si on adopte sa proposition la carte syntaxique des expressions de chaîne serait celle de la page ci-contre. On aurait ainsi une nouvelle expression qui est celle de chaîne. En ce qui concerne les opérations sur les chaînes, il n'y aura pas de nouveaux opérateurs mais elles seront faites au moyen des fonctions standard qui sont les suivantes :

1- booleen procedure ATOMIC (S); chaîne S;
⟨corps de procedure⟩;

commentaire : ATOMIC prend la valeur vrai si et seulement si le corps de S est un symbole de chaîne ;

2 - booleen procedure EQUAL (S,T); chaîne S,T;
⟨corps de procedure⟩ ;

commentaire : EQUAL prend la valeur vrai si et seulement si S et T sont identiques ;

3- chaîne procedure CONCATENATE (S, T); chaîne S,T;
⟨corps de procedure⟩ ;

commentaire : CONCATENATE devient la chaîne obtenue par la concaténation de la chaîne S et de la chaîne T en supprimant leurs crochets de chaînes les plus extérieurs.

Exemple : CONCATENATE (' 1 ' , ' 2 ') = ' 12 '
CONCATENATE (' ABC ', ' 102 ') = ' ABC ' 102 '

4 - chaîne procedure FIRST (S) ; chaîne S;
⟨corps de procedure⟩;

commentaire : si la chaîne S est vide alors FIRST devient S, autrement si le premier élément de S est un symbole de chaîne alors FIRST devient la chaîne dont le corps est ce symbole, autrement FIRST devient cet élément.

Exemple : FIRST (' A ') = ' A '
FIRST (' AB ' CD ') = ' AB '

5 - chaîne procedure TAIL (S); chaîne S;
⟨corps de procedure⟩ ;

commentaire : si S est vide alors TAIL devient S autrement TAIL devient la chaîne dont le corps est celui de S enlevé de son premier élément.

Exemple : TAIL ('A B C') = 'B C'

6 - procédure INSYMBOL (S) ; chaîne S;
⟨corps de procedure⟩;

commentaire : S devient la chaîne dont le corps est le symbole de chaîne trouvé sur le support extérieur ;

7 - procedure OUT SYMBOL (S); chaîne S ;
⟨corps de procedure⟩ ;

commentaire : si ATOMIC (S) alors OUT SYMBOL transmet vers l'extérieur le cymbole de chaîne contenu dans S, autrement elle est ineffective ;

8 - procedure INMARK (N) entier N;
⟨corps de procedure⟩ ;

commentaire : INMARK affecte à N l'entier positif M si le symbole trouvé sur le support extérieur est le $M^{ième}$ de la liste des symboles reconnus*.

9 - procedure OUTMARK (N) ; valeur N; entier N;
⟨corps de procedure⟩ ;

commentaire : OUTMARK transmet vers l'extérieur le $N^{ième}$ symbole de la liste des symboles reconnus * ;

* la liste des symboles reconnus doit être scumise à l'acceptation générale.

A partir de ces procédures de base, on peut construire en ALGOL d'autres procédures de traitement de chaînes dont voici quelques exemples donnés par VAN WIJNGAARDEN :

chaîne procedure ELEMENT (CHAINE, RANG);
valeur RANG ; entier RANG; chaine CHAINE ;
commentaire :
ELEMENT devient la chaîne dont le corps est le $RANG^{ième}$ élément de CHAINE, en comptant de gauche à droite;

```
      ELEMENT:= si RANG = 1 alors FIRST (CHAINE)
                sinon  ELEMENT (TAIL (CHAINE), RANG-1);
```

chaine procedure DERNIER ELEMENT (CHAINE); chaîne CHAINE ;
commentaire :
DERNIER ELEMENT devient la chaîne dont le corps est le dernier élément de CHAINE;

```
      DERNIER ELEMENT := si EQUAL (CHAINE, FIRST ( CHAINE))
                         alors  CHAINE   sinon
                         DERNIER ELEMENT ( TAIL (CHAINE)) ;
```

entier procedure LONGUEUR (CHAINE); chaîne CHAINE;
commentaire :
LONGUEUR devient le nombre de symboles de base contenus dans le corps de CHAINE ;

```
LONGUEUR := si EQUAL (CHAINE,' ') alors  0  sinon
            si ATOMIC (CHAINE) alors 1  sinon
            LONGUEUR (FIRST (CHAINE)) +
            si ATOMIC (FIRST (CHAINE)) alors 0 sinon 2 +
            LONGUEUR ( TAIL (CHAINE)) ;
```

DISCUSSION :

Tout d'abord considérons les procédures d'entrée-sortie INMARK, OUTMARK, INSYMBOL et OUTSYMBOL :
il est souhaitable de leur ajouter un paramètre CANAL, qui sert à spécifier le milieu extérieur avec lequel le programme est en contact. Car dans le cas général on peut avoir plusieurs éléments périphériques.

En ce qui concerne plus spécialement INMARK et OUTMARK, il semble intéressant de leur donner la forme des procédures. INSYMBOL, OUTSYMBOL de la proposition d'I.F.I.P. Ce qui permettra d'avoir un dictionnaire variable de symboles, lequel sera explicite dans chaque programme. Ainsi il n'y aura plus besoin d'établir une liste de symboles reconnus qui devrait être soumise à l'acceptation générale.

Les déclarations de INMARK, OUTMARK, INSYMBOL, OUTSYMBOL deviennent :

procédure INMARK (CANAL, CHAINE, DESTINATION) ;
valeur CANAL; entier CANAL, DESTINATION; chaîne CHAINE;
⟨corps de procédure⟩ ;
commentaire :
l'action de INMARK est celle de INSYMBOL de la proposition I.F.I.P.;
procedure OUTMARK (CANAL, CHAINE, SOURCE);
valeur CANAL, SOURCE ; entier CANAL, SOURCE; chaîne CHAINE;
⟨corps de procédure⟩ ;
commentaire :
l'action de OUTMARK est celle de la procedure OUTSYMBOL de la proposition d'I.F.I.P.
procedure INSYMBOL (CANAL, CHAINE); valeur CANAL ;
entier CANAL ; chaîne CHAINE;
⟨corps de procédure⟩ ;
procedure OUTSYMBOL (CANAL, CHAINE); valeur CANAL;
entier CANAL ; chaîne CHAINE;
⟨corps de procédure⟩;

Examinons maintenant les possibilités de traitement de chaîne de la proposition :

La puissance du langage LISP dans le domaine de manipulation de symboles n'est plus à démontrer. VAN WIJNGAARDEN a certainement voulu incorporer en ALGOL quelques possibilités de LISP. Si une comparaison est faite entre cette proposition et LISP, en remarque la correspondance suivante/:

PROPOSITION DE VAN WIJNGAARDEN	LISP
SYMBOLE DE CHAINE	ATOME
CHAINE	LISTE
FIRST	CAR
TAIL	CDR
EQUAL	EQUAL
ATOMIC	ATOM

Mais il reste quelques points obscurs à préciser : Supposons que nous ayons deux chaînes S1 et S2, nous voulons savoir si leurs troisèmes éléments sont identiques, l'expression booléenne

EQUAL (ELEMENT (S1, 3), ELEMENT (S2, 3))

doit nous fournir la réponse.

Maintenant supposons que S1 = 'ABCD'

et S2 = 'AB''C'D'

alors EQUAL (ELEMENT (S1,3)), ELEMENT (S2, 3)) = <u>vrai</u> ce qui signifierait que C et 'C'sont identiques?

D'autre part soit S chaîne''A' B', LENGTH (S) = 2, ce qui n'est pas correct. Car dans le commentaire de la procédure LENGTH, l'auteur précise que LENGTH (S) est le nombre de symboles de base contenus dans le corps de S, ce qui aurait donné à LENGTH (S) la valeur 4. Ces difficultés sont dues à une assimilation non correcte des listes de LISP avec les chaînes en ALGOL. En effet, en LISP une liste peut être soitune liste composée (délimitée par une paire de parenthèses extrêmes), soit un atome. Tandis qu'en ALGOL, une chaîne doit comporter nécessairement une paire de crochets de chaînes extrêmes, et ne peut être un symbole de chaîne.

II - 4 PROPOSITION DE NEDERKOORN - VAN DE LAARSCHOT

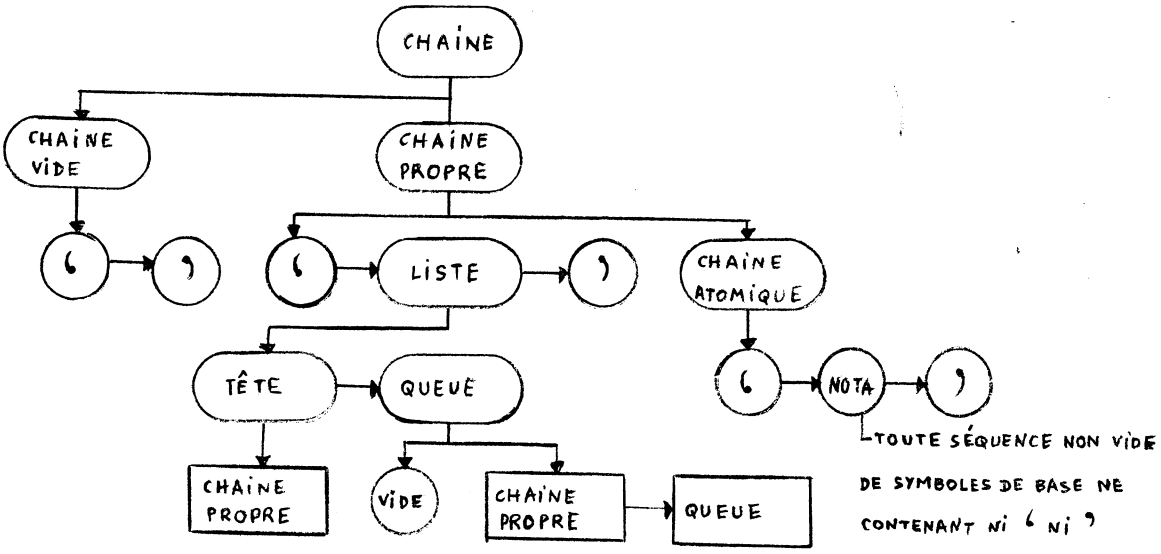
Cette proposition décrit sommairement une expérience de traitement une expérience de traitement de liste que les auteurs ont programmé sur la machine X1 au centre mathématique d'Amsterdam.

L'exposé se divise en deux parties :

- sommaire des chaînes en ALGOL
- sommaire des procédures de liste

A. Sommaire des chaînes en ALGOL

1 - La syntaxe des chaînes d'ALGOL est définie comme suit :



mais les chaînes que manipulent les procédures de liste doivent obéir aux conventions suivantes :

2 - Si la chaîne se trouve sur le support extérieur, elle doit s'exprimer en notation courte, très proche de celle des listes en LISP :

- deux chaînes propres, éléments d'une même liste sont séparés par le symbole '

- pour les chaînes atomiques les crochets de la chaîne sont supprimés.

Exemples de chaînes	Notation courte	Notation de LISP
''	''	NIL
'A'	A	A
''A''B''C''D''	'A'B'(C'D''	(A,B,(C,D))
''A''	'A'	(A)

3 - Si la chaîne apparaît dans le texte du programme ALGOL même, il y a la restriction suivante :

⟨chaîne propre⟩ :: = ⟨chaîne atomique⟩

B. Sommaire des procédures de liste :

En tout il y a 11 procédures, toutes écrites en code non ALGOL. Les chaînes qu'elles manipulent sont stockées en mémoire sous forme de listes. Les pointeurs de listes sont des variables de type entier déclarés dans le programme.

1 - procédure START LISP (N) ; entier N ; ⟨code⟩; commentaire ;

L'instruction START LISP (⟨expression entière⟩) doit apparaître au moins une fois dans le programme, et être exécutée avant l'activation de toute autre procédure de liste. Si N est son paramètre actuel, elle réserve N mémoires de la machine pour la manipulation des listes ;

2 - procédure CALL (A,CHAINE) ; entier A ; chaîne CHAINE ;
⟨ code⟩ ;
commentaire :

L'instruction CALL (⟨variable de type entier⟩,⟨chaîne⟩) introduit un concept sémantique nouveau : la variable est associée à la chaîne telle que la variable représente la chaîne ; cette association est effectuée par une affection implicite de valeur à la variable, autrement dit on affecte à la variable la valeur entière qui représente le pointeur de la liste représentant la chaîne en question ;

3 - procédure READSTRING (A) ; entier A ;⟨code⟩ ; commentaire :

L'instruction READSTRING (⟨variable de type entier⟩) affecte à la variable, paramètre effectif de READSTRING, la valeur du pointeur représentant la chaîne trouvée sur le support extérieur ;

4 - entier procédure PRINTSTRING (A) ; entier A ;⟨code⟩;
commentaire :

L'instruction PRINTSTRING (⟨expression entière représentant une chaîne⟩)

cause l'impression sur papier la chaîne que représente son paramètre effectif. Après l'exécution de l'instruction procédure, PRINSTRING représente la chaîne en question ;

5 - entier procédure PUNCH STRING (A) ; entier A ; ⟨code⟩ ;
commentaire :

L'instruction PUNCH STRING (⟨expression entière représentant une chaîne⟩)
cause la perforation de la chaîne sur carte. Après l'exécution de l'instruction procédure PUNCH STRING représente la chaîne en question ;

6 - entier procédure CAR (A) ; entier A ; ⟨code⟩;
commentaire :

L'indicateur de fonction CAR (⟨expression entière représentant une chaîne propre non atomique⟩)
désigne la tête de la chaîne ;

7 - entier procédure CDR (A) ; entier A ; ⟨code⟩ ;
commentaire :

L'indicateur de fonction CONS (⟨expression entière représentant la première chaîne⟩, ⟨expression entière représentant la deuxième chaîne⟩) représente une nouvelle chaîne. La nouvelle chaîne est '⟨tête⟩⟨queue⟩' , où ⟨tête⟩ est la première chaîne. Pour former ⟨queue⟩, il suffit de supprimer les crochets de chaîne les plus extérieurs à la deuxième chaîne ;

9 - booléen procédure EQ (A,B) ; entier A,B ; ⟨code⟩ ; commentaire :

L'indicateur de fonction EQ (⟨expression représentant la premiè chaîne atomique ou vide⟩ , ⟨expression représentant la deuxième chaîne atomique ou vide⟩) prend la valeur vrai si et seulement si les deux chaînes sont identiques ;

10 - booléen procédure ATOM (A) ; entier A ; ⟨code⟩ ; commentaire :

L'indicateur de fonction ATOM (⟨expression entière représentant une chaîne⟩) prend la valeur vrai si et seulement si la chaîne est atomique ou vide ;

11 - procédure DELETE (A) ; entier A ; ⟨code⟩ ; commentaire :

l'instruction DELETE (⟨expression entière représentant une chaîn a l'effet suivant :

Les procédures CONS, READSTRING etc... créent de nouvelles chaînes qui occupent des mémoires réservées par START LISP. Par l'activation de DELETE, le programmeur libère les mémoires occupées par la chaîne que représente son paramètre effectif. Il en résulte un gain d'espace pour de nouvelles chaînes ;

DISCUSSION :

A la différence des autres propositions, celle-ci n'implique aucune extension syntaxique du langage. L'incorporation à ALGOL des facilités de manipulation de symboles se base sur une généralisation sémantique : on utilise des variables entières pour représenter les chaînes. Et pourtant les possibilités nouvelles obtenues sont grandes, car elles se basent sur le traitement de listes de LISP. Or ce langage est bien connu pour être un instrument de valeur en manipulation de symboles.

Un autre fait remarquable à signaler est le suivant : sans contester la valeur de LISP, si on considère deux programmes équivalents, l'un écrit en ALGOL, l'autre en LISP ; le programme en ALGOL est certainement plus lisible.

Cependant il y a quelques légères critiques à faire sur la manière de représenter les objets que manipulent les procédures de liste :

Puisque le problème de pouvoir traiter n'importe quelle chaîne (au sens du rapport de base) comme une liste n'est pas résolue

- au lieu d'écrire les chaînes en notation courte très proche de celle des listes de LISP, et d'imposer des restrictions sur les chaînes apparaissant dans le texte du programme ALGOL, il est plus simple d'utiliser exactement la notation des listes de LISP, et cela que ce soit sur le support extérieur ou dans le programme même.

Il est à remarquer, en passant, que cette proposition constitue le point de départ de l'élaboration de la "définition de procédures de LISP en ALGOL", qui va être exposée dans le chapitre suivant.

III - DEFINITION DE PROCEDURES DE LISTE EN ALGOL

En 1958 Mc Carthy a proposé un langage de programmation appelé LISP destiné à faciliter la résolution des problèmes non numériques. Parmi les problèmes qu'il se proposait de résoudre avec ce lagage, Mc CARTHY avait mentionné explicitement les suivants :

1 - Réalisation de compilateurs capables de traduire LISP en un langage machine quelconque.

2 - Ecriture d'un programme pour vérifier la démonstration de théorèmes relatifs à une classe spéciale de systèmes formels.

3 - Elaboration de programmes effectuant la différentiation et l'intégration formelles.

4 - Développement de programmes générant les démonstrations de théorèmes en calcul des prédicats.

5 - Ecriture de programmes dont les résultats sont des formules proprement dites et non leurs valeurs numériques.

On peut dire que six ans après la suggestion de Mc CARTHY tous les problèmes mentionnés ci-dessus étaient pratiquement réalisés. Cela est sans aucun doute une indication très importante sur la validité de ce langage.

On voit donc l'intérêt de pouvoir incorporer les caractéristiques principaux de LISP dans un langage tel que ALGOL. Cela permettrait la mise au point rapide d'un outil pour le traitement des listes, qui aurait encore l'avantage de posséder des caractéristiques supplémentaires qui n'existent pas en LISP pur.

III - 1 PROCEDURES DE BASE

Avant de décrire les procédures de base proprement dites nous allons définir les opérandes qu'elles manipulent. Ces opérandes sont des listes de symboles qui doivent respecter les règles syntaxiques suivantes, écrites sous forme normale de Backus (*) :

⟨ liste⟩ : = (⟨ liste d'éléments ⟩) | ⟨atome⟩
⟨ liste d'éléments⟩ :: = ⟨élément⟩ |
⟨liste d'éléments⟩⟨blanc⟩⟨élément⟩

(*) Dans cette description syntaxique la métavariable sous-liste n'est pas strictement nécessaire. Elle a été introduite dans le but de préciser la notion de sous-liste.

LAARSCHOT et NEDERKOORN ont proposé la réalisation de procédures en code machine donnant à un compilateur ALGOL la possibilité de traiter des programmes écrits en un langage assez proche de LISP. Notre travail est fondé principalement sur cette idée, cependant nous avons préféré construire ces procédures directement en ALGOL pour les raisons suivantes :

1 - Fournir aux presonnes connaissant déjà ALGOL la possibilité de se familiariser avec LISP en leur donnant une description détaillée de la façon dont LISP est construit.

2 - Permettre l'incorporation de ces procédures dans un compilateur ALGOL quelconque soit en les traduisant en procédures dont le corps est en code soit en les utilisant directement comme nous l'avons fait.

3 - Les procédures présentées dans ce travail ne représentent absolument pas la seule façon d'incorporer un langage de liste à ALGOL. Le programmeur expérimenté pourra sans peine essayer d'autres solutions en utilisant les idées exposées ici.

4 - Nous avons accès à un compilateur ALGOL sur une machine assez puissante et nous avons pu tester effectivement les procédures que nous présentons.

Dans les pages suivantes, nous décrivons en détail les procédures écrites pour permettre à un compilateur ALGOL de traiter des programmes écrits dans un langage équivalent à LISP.

⟨élément⟩ :: = ⟨sous-liste⟩ | ⟨atome⟩
⟨sous-liste⟩ :: = (⟨liste d'éléments⟩)
⟨atome⟩ :: = ⟨élément atomique⟩ | ⟨nil⟩
⟨élément atomique⟩ :: = ⟨symbole⟩ | ⟨élément atomique⟩ ⟨symbole⟩
⟨nil⟩ :: = ()
⟨symbole⟩:: = tout symbole disponible sauf (,),␣,
⟨blanc⟩ :: = ␣ | ⟨blanc⟩␣

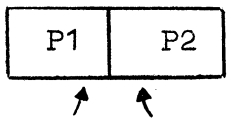
Remarquons que ces règles demandent l'existence des parenthèses extérieures dans le cas où la liste possède plus d'un élément atomique.

Exemple : soit la liste (*) :

(A (BC D) ())

A, (BC D) et () sont des éléments de cette liste ; A,BC,D et () sont chacun un ⟨atome⟩ ; (BC D) est une ⟨sous-liste⟩ .

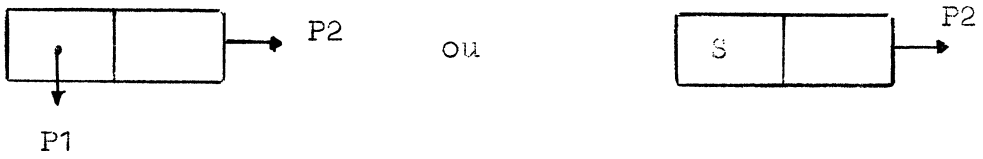
La sémantique associée aux définitions syntaxiques données ci-dessus est étroitement liée à la façon dont les éléments d'une liste sont emmagasinés en mémoire. Il est plus facile d'expliquer graphiquement le concept de liste. Considérons un mot machine représenté par :



partie car ou tête — partie cdr ou queue

(*) Remarquons que les blancs seront par la suite représentés par des espaces.

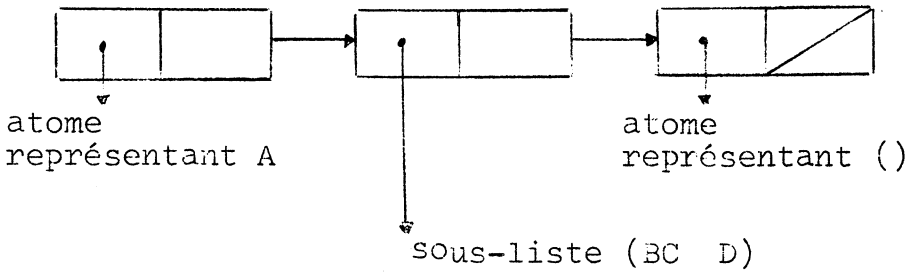
P1 et P2 sont les contenus de la partie gauche et droite d'un mot machine et les parties elles-mêmes seront appelées car et cdr. P1 et P2 représentent en général une adresse mais P1 peut aussi parfois représenter un symbole. On distingue ces deux cas par les représentations suivantes :



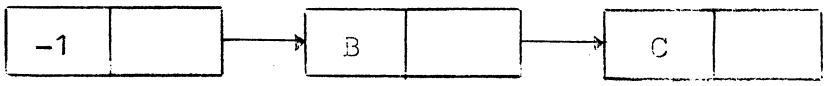
selon que car contient une adresse P1 ou un symbole S. Pour ce qui suit les adresses P1 et P2 sont des pointeurs (représentés par les flèches) qui indiquent les adresses où la prochaine information est emmagasinée : P2 pointe vers le prochain ⟨élément⟩ d'une ⟨liste d'éléments⟩ . P1 pointe soit vers une ⟨sous-liste⟩ soit vers un ⟨atome⟩,c.à.d. que P1 décrit la nature d'un élément d'une liste d'éléments.

La parenthèse ")" qui indique la fin d'une liste ou d'une sous-liste sera représentée par une adresse spéciale en partie cdr. Cette adresse est, dans notre cas, celle de la mémoire 1 et nous considérons les deux représentations ci-dessous comme identiques :

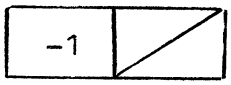
Le squelette de la liste donnée en exemple peut maintenant être ébauché par :



Bien que nous ayons déjà les moyens de représenter graphiquement la sous-liste (BC D), il nous faut encore expliquer la représentation graphique d'un atome. Les atomes sont rangés dans d'autres listes que nous appelons listes de propriété ("property lists"). Les listes de propriété sont caractérisées par leur première mémoire dont la partie car contient la valeur -1. Par exemple la liste de propriété de l'atome BC est la suivante :



L'adresse de la première mémoire d'une liste de prcpriété est utilisée comme pointeur dans la partie car d'un élément du squelette de liste que nous avons déjà décrit. Remarquons que les contenus des parties car de la deuxième mémoire et des mémoires suivantes de la liste de propriété contiennent les symboles qui forment l'atome en question. Observons aussi qu'une partie cdr contenant la valeur 1 est également utilisée pour indiquer la fin d'une liste de propriété et que la liste de propriété de "()" (appelé NIL) est emmagasinée dans la mémoire d'adresse 1. Cette dernière est représentée par



Comme nous décrivons les procédures de base en ALGOL nous allons diviser la mémoire de travail en deux tableaux à une dimension que nous appellerons TCAR et TCDR. Le couple formé par un élément de TCAR et l'élément correspondant de TCDR joue d'ailleurs le même rôle que le mot machine en LISP pur. Remarquons que si la machine pour laquelle on écrit ces procédures n'a pas suffisamment de mémoires il sera nécessaire de les écrire en code et de partager le mot machine comme en LISP pur.

Dans les pages suivantes on fait une description des procédures de base que nous appellerons procédures de liste et qui seront présentées en détail à la fin de ce chapitre .

En voici la liste :

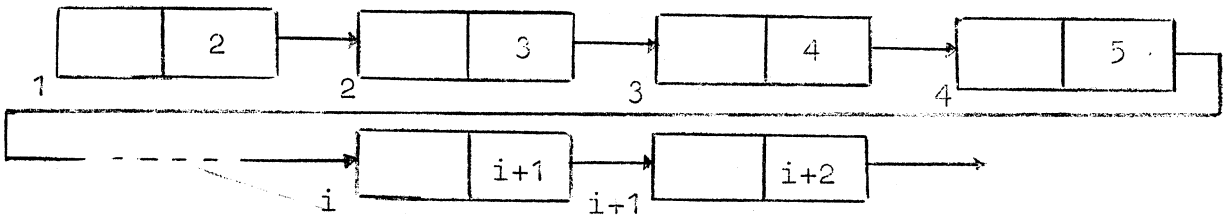
	BUT
1. PREPARER	Préparation de la liste libre
2. CAR	Détermination de la tête d'une liste
3. CDR	Détermination de la queue d'une liste
4. CONS	Construction d'une nouvelle liste
5. ATOM	Prédicat d'atomicité
6. EQ	Prédicat d'égalité
7. LIRE LISTE	Lecture de listes
8. NOMMER	Désignation de listes
9. ECRIRE LISTE	Impression de listes
10. LIBERER	Libération d'une liste

LIRE LISTE, NOMMER et ECRIRE LISTE sont des procédures de transmission (entrée-sortie) ; CAR, CDR et CONS sont des procédures de manipulation de listes ; EQ et ATOM sont des prédicats : le

premier permet la reconnaissance des symboles et le deuxième vérifie si un élément de liste est atomique. PREPARER et LIBERER sont des procédures auxiliaires. Passons maintenant à la description de toutes ces procédures.

Procédure PREPARER (<entier>)

PREPARER sert à initialiser une utilisation ultérieure des procédures de liste. Elle a un paramètre qui doit être le même entier que la borne supérieure des tableaux TCAR et TCDR lors de leur déclaration. Le but principal de cette procédure est de préparer une liste "libre" qui contient toutes les mémoires qui étaient déclarées disponibles. Chaque partie droite (TCDR) contient l'adresse de la prochaine mémoire disponible, qui initialement (c. à d. après l'exécution de PREPARER) est égale à l'adresse de la mémoire suivante. Cette première action est représentée schématiquement de la façon suivante :



Remarquons que toutes les mémoires nécessaires à une opération donnée sont prises dans la liste libre et y sont remises quand elles sont libérées par une istruction appropriée du programmeur (voir LIBERER) ; nous verrons plus tard que l'adresse de la mémoire disponible suivante n'est pas, en général, l'adresse du mot suivant dans mémoire de la machine.

La deuxième action de PREPARER est de construire la liste de propriété de NIL et d'initialiser le pointeur IPD qui indique l'adresse de la première mémoire de la liste libre.

entier procédure CAR (<expression entière représentant une liste>) :

L'indicateur de fonction CAR est assez simple. Sa valeur est le contenu de TCAR [P] ; P étant le paramètre effectif représentant le pointeur d'une liste.

entier procédure CDR (<expression entière représentant une liste >) ;

Cet indicateur de fonction a comme valeur celle du pointeur rangé en partie cdr de P (P étant le paramètre effectif).

entier procédure CONS (<expression entière représentant une liste >,

<expression entière représentant une liste >) ;

Le but de cette procédure est de prendre un couple TCAR-TCDR de la liste libre et de ranger en TCAR la valeur du premier paramètre et dans TCDR la valeur du second. La valeur de l'indicateur de fonction CONS après l'appel est un entier qui indique le pointeur de la nouvelle liste qui vient d'être construite. Le cas particul où le deuxième paramètre effectif représente un élément atomique demande l'utilisation d'une mémoire supplémentaire prise elle aussi de la liste libre et liée à la précédente pour former l'ensemble voulu. Observons que CONS, telle que décrite ici est légèrement différente de CONS utilisée en LISP. Cette différence a été introduite de manière à éviter la notation "dot" de LISP.

booléen procédure ATOM (⟨expression entière représentant une liste⟩);

L'indicateur de fonction ATOM prend la valeur vrai si le paramètre effectif pointe vers une liste de propriété c. à d. si TCAR [P]= -1 où P est le paramètre effectif. Dans les autres cas ATOM prend la valeur faux.

booléen procédure EQ (⟨expression entière représentant une liste⟩,
⟨expression entière représentant une liste⟩) ;

Cet indicateur de fonction a pour valeur vrai si les deux listes considérées sont atomiques et ont des listes de propriété identiques ; autrement il prend la valeur faux.

entier procédure LIRE LISTE ;

Cet indicateur de fonction est utilisé pour :

a. Lire des données sur cartes et les ranger sous forme de liste
b. Affecter à LIRE LISTE la valeur numérique du pointeur du premier élément de la liste lue.

Pour l'opération décrite en a. il est nécessaire d'employer une procédure d'entrée capable de lire un symbole alphanumérique et de lui donner une valeur numérique correspondante de type entier Ceci est réalisé par LIRESYMBOLE.

L'instruction LIRE SYMBOLE (0, '␣()ABCDEF ... ', TEMP)

lit le prochain symbole sur carte et affecte à la variable entière TEMP la valeur numérique correspondant au numéro d'ordre de cet élément dans la chaîne en 2ème paramètre. Ainsi si une "(" est le

premier caractère d'une carte donnée, après l'exécution de LIRE SYMBOLE, TEMP aura la valeur 2. Les blancs ont comme correspondant numérique la valeur 1.

Avant de décrire LIRE LISTE il convient de remarquer que la définition syntaxique des listes données précédemment, est trop rigide en ce qui concerne les blancs et rendrait lourde la tâche de préparer les listes de données : il serait intéressant de la modifier légèrement pour permettre l'introduction de blancs facultatifs entre les parenthèses. On a tenu compte de cette modification dans la procédure LIRE LISTE qui utilise comme procédures auxiliaires LIRE, SAUTER, LIRE ATOME, LIRE FIN ATOME, LIRE SUITE et LIRE LISTE MODIF. Les procédures LIRE et SAUTER vont pour but de placer en TEMP (variable entière déclarée dans un bloc extérieur) le premier symbole non-blanc de la chaîne d'entrée. La procédure LIRE lit un symbole et SAUTER ré-utilise LIRE si le symbole lu est un blanc ; sinon SAUTER ne produit aucun effet.

Le fonctionnement de LIRE LISTE peut être brièvement décrit de la façon suivante : LIRE LISTE fournit à LIRE LISTE MODIF le premier symbole de la liste à lire stocké dans TEMP qui doit contenir initialement ou vien la valeur 2 représentant une "(" ou bien une valeur plus grande que 3. Dans ce dernier cas la liste d'entrée est un atome et le contrôle est transféré à LIRE ATOME qui construit (en utilisant CONS) le début de la liste de propriété. Pour ranger les symboles successifs de l'atome dans la liste de propriété LIRE ATOME utilise LIRE FIN ATOME qui s'appelle elle-même jusqu'à la lecture d'un délimiteur de fin d'atome ("␣", "(" ou ")").

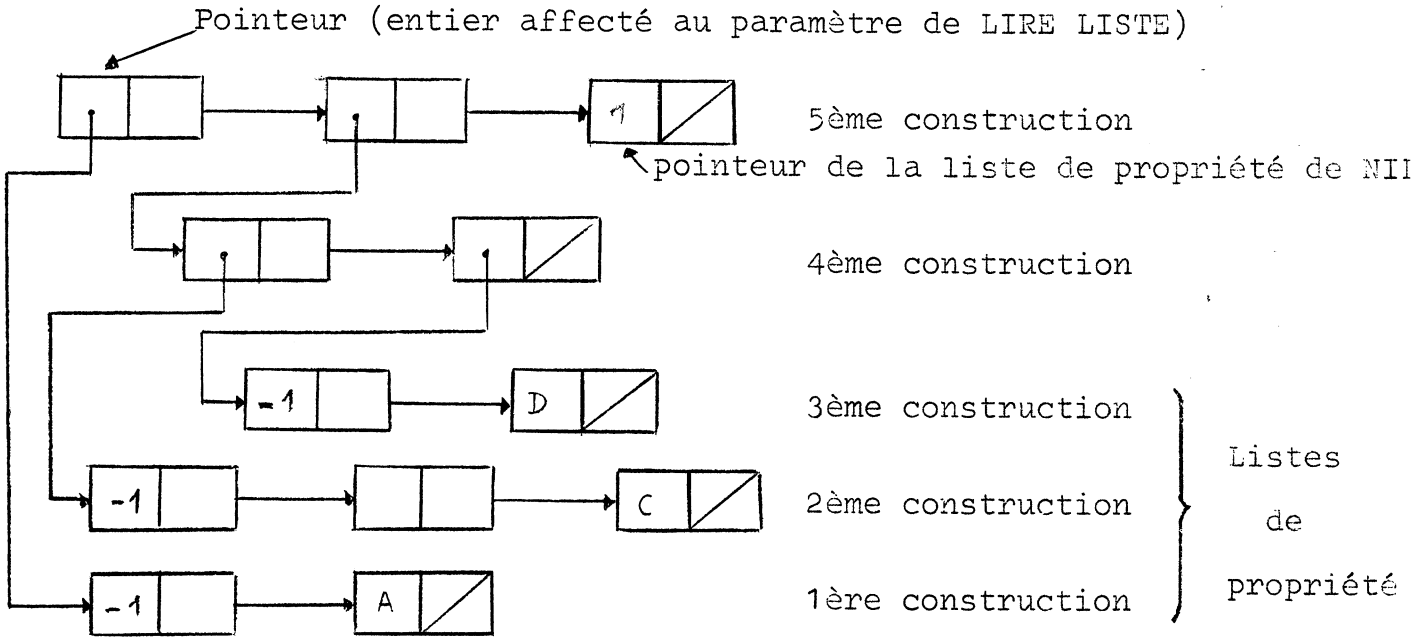
Revenons au cas où TEMP contenait la valeur 2 indiquant que le symbole lu était une "(". Dans ce cas LIRE LISTE MODIF lit le prochain symbole et si ce dernier est une ")" elle a pour valeur 1, c. à d. le pointeur de la liste de propriété de NIL. Autrement elle appelle LIRE SUITE qui construit une liste ayant pour tête LIRE LISTE MODIF et pour queue LIRE SUITE, à moins qu'une ")" soit lue, indiquant la fin d'une liste ou sous-liste.

En résumé on peut dire que LIRE LISTE MODIF lit une liste, LIRE ATOME et LIRE FIN ATOME lisent un atome et que LIRE SUITE lit l'intérieur d'une paire de parenthèses si son contenu est non vide.

Exemple : L'appel de LIRE LISTE pour lire la liste :

(A (BC D) ())

provoque le rangement suivant en mémoire :



entier procédure NOMMER (<chaîne>)

Cette procédure est semblable à LIRE LISTE : alors que LIRE LISTE permet de ranger des listes définies sur cartes (ou bandes) la procédure NOMMER permet de ranger des listes définies à l'intérieur d'un programme. NOMMER diffère de LIRE LISTE par l'utilisation :

- d'une zone de rangement provisoire dans laquelle est écrit le paramètre effectif chaîne

- d'un canal spécial de lecture pour la procédure LIRE SYMBOLE

Un appel de cet indicateur de fonction affecte à l'identificateur NOMMER la valeur du pointeur de la liste représentée par chaîne . On peut remarquer que l'effet de NOMMER est analogue à celui de QUOTE en LISP. Nous n'avons pas listé la procédure NOMMER en fin de chapitre, étant donné qu'elle peut être facilement reconstituée en utilisant LIRE LISTE.

entier procédure ECRIRE LISTE (<expression entière représentant une liste>) ;

Un appel de cette procédure qui a comme paramètre la variable entière représentant une liste construite au moyen des procédures de liste, permet d'imprimer la liste correspondante. L'impression est telle qu'il ne serait pas nécessaire de faire aucune modification si on devait la lire ultérieurement au moyen de LIRE LISTE.

Remarquons que ECRIRE LISTE est un indicateur de fonction du type entier dont la valeur après exécution est celle du pointeur de la liste à imprimer. L'intérêt d'avoir ECRIRE LISTE comme indicateur de fonction est de pouvoir l'utiliser comme paramètre effectif, d'autres procédures (en particulier LIBERER). Cela peut aussi faciliter la détection des erreurs quand on veut introduire des impressions intermédiaires.

procédure LIBERER (< expression entière représentant une liste >) ;

Le fonctionnement de cette procédure est semblable à celui de la procédure ECRIRE LISTE. Au lieu d'imprimer des symboles atomiques et des parenthèses on ajoute à la liste libre les mémoires correspondantes. On recommande une grande prudence dans l'utilisation de cette procédure, car elle efface toutes les listes de propriété qui ont été utilisées dans la construction de la liste que l'on veut effacer. Répétons qu'après l'exécution de LIBERER, la liste libre est complètement modifiée et que la mémoire suivante ne correspond pas en général à la mémoire au sens séquentiel. Cependant cela ne perturbe en rien le fonctionnement des autres procédures. Au contraire elle permet une utilisation rationnelle des mémoires disponibles.

LIBERER est un des éléments les plus importants d'un langage de listes. En LISP, il est appelé "garbage collector" (récupérateur) et est automatiquement en action lorsque toute mémoire libre est épuisée.

Nous réalisons que cette procédure, telle qu'elle est présentée ici, ne constitue pas une solution idéale aux problèmes de récupération de l'espace occupé par les listes devenues inutiles. Un palliatif à ces problèmes serait d'avoir deux procédures de libération des listes avec ou sans récupération des atomes. Une autre solution serait d'incorporer une procédure pour la recopie des listes.

Dans les pages suivantes le lecteur trouvera en ALGOL un listage des procédures LISP.

```
procédure PREPARER  (N) ; valeur N ; entier N ;

    début entier I ;
        commentaire préparation de la liste libre ;
        pour  I := 2 pas 1 jusqua N faire
           début
                TCAR [ I]: = 0 ;
                TCDR [I]: = I + 1
           fin ;

        commentaire construction de la liste de propriété de NIL ;
        TCAR  1 : = -1 ;
        TCDR  1 : =  1 ;

        IPD : = 2

    fin procédure PREPARER ;
```

```
entier procédure CAR (A) ; valeur A ; entier A ;

     CAR := TCAR[ A];

entier procédure CDR  (A); valeur A ; entier A ;

     CDR := TCDR [A];

entier procédure CONS (A,B); valeur A,B; entier A,B;

     début entier I;

          CONS := I := IPD;
          IPD := TCDR [IPD];
          TCAR [ I]:=  A;

          si TCAR [ B]=-1∧B ≠ 1
             alors début
                     I := IPD;
                     IPD := TCDR [IPD];
                     TCAR [ I] := B;
                     TCDR [ I ] := 1
                   fin
             sinon  TCDR [ I] := B

     fin procédure CONS;
```

```
booléen procédure ATOM (A); valeur A; entier A;

        ATOM :=   TCAR[A]= -1;

booléen procédure EQ  (A,B) ; valeur A,B; entier A,B;

     début

          si TCAR A ≠ -1 ∨ TCAR[B]≠ -1 alors allera FAUX;
          si A = B alors allera VRAI;

          BOUCLE :
          A := TCDR[A]; B := TCDR [B];
          allera si A = 1  ∧  B = 1 alors VRAI sinon
                    si TCAR [A] ≠  TCDR[B] alors FAUX sinon BOUCLE;

          FAUX : EQ := faux; allera FINI;
          VRAI : EQ := vrai;
          FINI :
     fin procédure EQ;
```

```
entier procédure LIRE LISTE ;
      début
        LIRE ; SAUTER ;
        LIRE LISTE := LIRE LISTE MODIF
      fin procédure LIRE LISTE ;

entier procédure LIRE LISTE MODIF ;
      si TEMP > 3 alors LIRE LISTE MODIF := LIRE ATOME
                  sinon
                  début
                    LIRE ; SAUTER ;
                    LIRE LISTE MODIF := si TEMP = 3 alors 1
                                        sinon LIRE SUITE
                  fin ;


procédure LIRE ;
     ENTRE SYMBOLE (0, ' ␣() ABCDEF ... ', TEMP) ;


procédure SAUTER ;
     si  TEMP = 1 alors début
                          LIRE ;
                          SAUTER
                        fin ;
```

```
entier procédure LIRE ATOME ;
      LIRE ATOME := CONS (-1, (LIRE FIN ATOME)) ;


entier procédure LIRE FIN ATOME ;
      début entier T ;
         T:= TEMP ; LIRE ;
         LIRE FIN ATOME := CONS (T, si TEMP ⩽ 3 alors 1
                                                sinon
                                    (LIRE FIN ATOME))
      fin procédure LIRE FIN ATOME ;


entier procédure LIRE SUITE ;
      début entier V,T ;
         V:= TEMP ;
         T:= LIRE LISTE MODIF ;
         si V = 2 alors LIRE ;
         SAUTER ;
         LIRE SUITE := CONS (T, si TEMP = 3 alors 1 sinon (LIRE SUITE))
      fin procédure LIRE SUITE ;
```

```
entier procédure ECRIRE LISTE (V) ; valeur V ; entier V ;
      début
        ECRIRE LISTE := V ;
        ECRIRE LISTE MODIF (V)
      fin procédure ECRIRE LISTE ;

procédure ECRIRE LISTE MODIF (V) ; valeur V ; entier V ;
     si ATOM (V) alors ECRIRE ATOME (V)
                 sinon début
                         ECRIRE SYMBOLE (2, '(',1) ;
                         ECRIRE SUITE (V)
                       fin procédure ECRIRE LISTE MODIF ;

procédure ECRIRE SUITE (V) ; valeur V ; entier V ;
     début
       ECRIRE LISTE MODIF (CAR (V)) ;
       si CDR (V) = 1 alors ECRIRE SYMBOLE (2,')',1)
                      sinon ECRIRE LISTE MODIF (CDR (V))
     fin procédure ECRIRE SUITE ;

procédure ECRIRE ATOME (V) ; valeur V ; entier V ;
     début
          si V = 1 alors début
                                ECRIRE  SYMBOLE (2,')',1) ;
                                ECRIRE  SYMBOLE (2,')',1)
                           fin
                   sinon
          pour V := TCDR [ V] tant que V ≠ 1 faire
              ECRIRE SYMBOLE (2 'U() ABC ... ', TCAR [V]) ;
      finprocédure ECRIRE ATOME ;
```

```
procédure LIBERER (Y) ; valeur Y ; entier Y ;
      si ATOM (Y) alors LIBERER ATOME (Y)
                  sinon
                        début
                          LIBERER (CAR(Y));
                          LIBERER (CDR(Y));
                          LIBERER MOT (Y)
                        fin procédure LIBERER

procédure LIBERER ATOME (V); valeur V; entier V ;
     début entier T ;
           pour T := V tantque T ≠ 1 faire
                 début
                    V := TCDR [ T];
                    LIBERER MOT (T)
                 fin
      fin procédure LIVERER ATOME;

procédure LIBERER MOT (U) ; valeur U ; entier U ;

     si TCAR [ U] ≠ 0 alors
           début
              TCAR [ U]:= 0;
              TCDR [U]:= IPD;
              IPD := U
           fin procédure LIBERER MOT;
```

III - 2 PROCEDURES AUXILIAIRES

En LISP on utilise amplement la récursivité pour définir des fonctions plus complexes à partir des fonctions élémentaires CAR, CDR, CONS, EQ et ATOM ; nous avons transcrit quelques unes de ces fonctions plus complexes en ALGOL pour des raisons suivantes :

1 - Nous voulons montrer aux personnes connaissant déjà LISP comment on effectue la traduction de LISP en ALGOL.

2 - Ces procédures sont utiles pour la construction d'autres procédures dont l'utilisateur pourrait avoir besoin. Nous les appelons procédures auxiliaires et elles ont aussi été vérifiées sur calculateur.

Citons d'abord CAAR, CADR, ... CDDDR qui sont très simples mais s'avèrent fréquemment utilisées dans la construction des programmes.

Dans ce qui suit nous dirons qu'un identificateur représente une liste s'il a pour valeur le pointeur de cette liste et nous utiliserons l'abréviation ⟨EERL⟩pour désigner ⟨expression entière représentant une liste⟩ . Nous appelons "rang d'un élément d'une liste" son numéro d'ordre d'apparition dans la liste.

Exemple : pour la liste ((BC) A (D))
(BC) est l'élément de rang 1
A est l'élément de rang 2
(D) est l'élément de rang 3

Passons maintenant aux commentaires sur les procédures auxiliaires qui seront présentées à la fin de ce chapitre.

booleen procedure EGAL (⟨ EERL⟩ , ⟨EERL ⟩)

Le rôle de EGAL est analogue à celui de EQ dont la description a déjà été faite. La seule différence entre EQ et EGAL est que EQ s'applique à deux atomes lors que EGAL vérifie si deux listes (en général composées) sont identiques.

Exemple : Soient les listes A = (1 2)

B = (1 2)

alors EQ (A,B) = faux

EQ (CAR(A),CAR (B))= vrai

EGAL (A,B) = vrai

entier procedure DERNIER (⟨ EERL ⟩)

DERNIER (M) représente le dernier élément (atomique ou non) de la liste M.

Exemple : M = (A (BC) (DE))

DERNIER (M) = (DE)

booleen procedure MEMBRE (⟨ EERL ⟩ , ⟨EERL⟩)

MEMBRE (X,Y) est vrai ou faux selon que X est un élément de la liste Y ou ne l'est pas.

Exemple : X = A

Y = ((BB) A)

Z = (ABC B ((A)))

MEMBRE (X,Y) = vrai

MEMBRE (X,Z) = faux

entier procedure JUXTAPOSER (⟨EERL⟩ , ⟨EERL⟩)

JUXTAPOSER (X,Y) construit et représente une liste nouvelle par juxtaposition des éléments de X et de Y.

Exemple : X = (ASINUS ASINUM)

Y = (FRICAT)

JUXTAPOSER (X,Y) = (ASINUS ASINUM FRICAT)

entier procedure RETOURNER (⟨EERL⟩)

RETOURNER (X) représente la liste retournée des éléments de la liste X

Exemple : X = (A (B C) D)

RETOURNER (X) = (D (B C) A)

entier procedure SUBST (⟨ EERL⟩ , ⟨ EERL⟩ , ⟨ EERL⟩)

SUBT (X,Y,Z) représente la liste obtenue par remplacement de toute apparition de Y dans Z par X, et cela non seulement au niveau des éléments de liste mais aussi dans les sous-listes à tous les niveaux.

Exemple : X = (MB)
Y = A
A = (() BA (A B)A)
SUBST (X,Y,Z) = (() BA ((MB)B) (MB))

entier procedure UNION (⟨EERL⟩ ,⟨EERL⟩)

UNION (X,Y) représente la liste obtenue par juxtaposition des éléments distincts de X, Y et des éléments communs à X, Y. Pour les premiers, les éléments de X précèdent ceux de Y. Pour les derniers, leur rang dépend uniquement de la position qu'ils occupent dans Y.

Exemple : X = (A B C u D)
Y = (C D E)
UNION (X,Y) = (A B C D E)
UNION (Y,X) = (E A B C D)

entier procedure INTERSECTION (⟨EERL⟩ , ⟨EERL⟩)

INTERSECTION (X,Y) représente la liste des éléments communs à X, Y. Leur rang dans la liste résultante dépend du mécanisme suivant : on examine séquentiellement chaque élément de X : s'il appartient à la liste des éléments de Y on le garde, autrement on passe à l'élément suivant. Le processus s'arrête quant la liste des éléments de X est épuisée.

Exemple : X = (A B C D)
Y = (C T A S)
INTERSECTION (X,Y) = (A C)
INTERSECTION (Y,X) = (C A)

entier procedure MATCH (⟨EERL⟩ , ⟨EERL⟩)

MATCH (X,Y) représente un atome qui est le premier élément de même rang, commun aux deux listes X, Y.

Exemple : X =(A B C E)
Y = (D E F E)
Z = (D F E F)
MATCH (X,Y) = E
MATCH (X,Z) = NIL

entier procedure LISTE DES ATOMES (< EERL >)

LISTE DES ATOMES (S) représente la liste des éléments atomiques (pas nécessairement des éléments de liste) de S.

Exemple : S = ((ABC) (D E((F))))
LISTE DES ATOMES (S) = (ABC D E F)

entier procedure RETOURNER SUPER (< EERL >)

RETOURNER SUPER (X) représente la liste retournée des éléments de X qui ont eux-mêmes leurs éléments retournés s'il sont des sous-listes.

Exemple : M = ((A B C) (D E)((F)))
RETOURNER SUPER (M) = (((F)) (E D)(C B A))

entier procédure SUPPRIMER (<expression entière représentant un atome> , <EERL>)

SUPPRIMER (X, Y) représente la liste obtenue par suppression dans Y de la première occurence de X comme élément de liste de Y.

Exemple : X = B
M = (A B C E F)
N = (A (B) C E F)
SUPPRIMER (X,M) = (A C E F)
SUPPRIMER (X,N) = (A (B) (E F)

entier procedure TWIST (<EERL>)

TWIST (X) représente la liste retournée des éléments de X qui ont eux-mêmes leurs éléments retournés s'ils sont des sous-listes. La seule différence entre TWIST et RETOURNER SUPER

est que le premier tient compte de NIL alors que le deuxième ne le fait pas.

Exemple : X = (A B)
TWIST (X) = ((() B) A)
RETOURNER SUPER (X) = (B A)

entier procedure PAIRE (⟨EERL⟩ ,⟨EERL⟩)

PAIRE (X,Y) représente la liste obtenue par assemblage des sous-listes construites à partir des éléments de X et de Y de même rang jusqu'à l'épuisement de la liste X.

Exemple : M = (A B C D)
N = (I J K L)
PAIRE (M,N) = ((A I)(B J) (C K) (I L)

entier procedure PAIRE LISTE (⟨EERL⟩,⟨EERL⟩ ,⟨EERL⟩)

PAIRE LISTE (X,Y,Z) associe à chaque élément de X un élément de Y de même rang; les sous-listes ainsi obtenues forment avec la liste Z une nouvelle liste que représente PAIRE LISTE

Si Y comporte plus d'éléments que X, les derniers éléments de Y n'entrent pas en ligne de compte. Si X comporte plus d'éléments que Y, le résultat de PAIRE LISTE est indéterminé.

Exemple : X = (A B C)
Y = (U V W)
Z = ((D X) (E Y))
PAIRE LISTE (X,Y,Z)= ((A U)(B V)(C W) (D X)(E Y))

entier procedure ASSOCIER (⟨EERL⟩ , ⟨EERL⟩)

ASSOCIER (X,Y) cherche dans Y une sous-liste dont la partie CAR soit identique à X; si une telle sous-liste existe, alors ASSOCIER la représente, autrement la valeur de ASSOCIER est indéterminée.

Exemple : X = B
Y = ((A (M N)) (B(CAR X)) (C (QUOTE M)) (C (CDR X)))
ASSOCIER (X,Y) = (B(CAR X))

entier procedure SUBLIS (⟨EERL⟩ , ⟨EERL⟩)

SUBLIS (X,Y) examine chaque atome de Y: si cet atome est identique à la partie CAR d'une sous-liste de X alors elle remplace dans Y, cet atome par la partie CDR de la sous-liste en question. Le processus continue jusqu'à l'épuisement des éléments de la liste Y; à la fin de l'opération, SUBLIS représente la liste résultante.

Exemple : X = ((X SHAKESPEARE) (Y (THE TEMPEST)))
Y = (X WROTE Y)
SUBLIS (X,Y) = ((SHAKESPEARE) WROTE ((THE TEMPEST)))

entier procedure APPLIQUER (⟨EERL⟩, indicateur de fonction ayant seul paramètre)

APPLIQUER (X,F) représente la liste obtenue par concaténation des listes, résultant de l'application de F sur X, sur CDR (X), CDDR (X), ... CD .... DR (X) jusqu'à l'épuissement de X.

Exemple : X = ((A B) (C D) (E F))
APPLIQUER (X, CAAR) = (A C E)

Et voici le listage en ALGOL des procédures auxiliaires.

```
entier procédure  CAAR (X) ; valeur X ; entier X ;
      CAAR := CAR (CAR (X) ) ;

entier procedure  CADR (X) ; valeur X ; entier X ;
      CADR := CAR (CDR (X) ) ;

entier procedure  CDDR (X) ; valeur X ; entier X ;
      CDDR := CDR (CDR (X) ) ;

entier procedure  CAAAR (X) ; valeur X ; entier X ;
      CAAAR := CAR (CAR (CAR (X) ) )  ;
      .
      .
      .
      .
      .
      . etc.
```

```
booléen procédure EGAL (A,B); valeur A,B; entier A,B;

      EGAL := si A = B alors vrai
                      sinon
              si ATOM (A)∧ATOM (B) alors EQ (A,B)
                                   sinon
              si ¬(ATOM(A)∧ATOM (B))) alors faux
                                      sinon
              si EGAL(CAR(A), CAR(B))) alors
                              EGAL (CDR(A), CDR(B))
                                      sinon faux;


entier procédure DERNIER (M); valeur M; entier M

      DERNIER := si CDR(M) = 1 alors CAR(M)
                               sinon DERNIER (CDR(M));
```

```
booléen procédure MEMBRE (X,Y); valeur X,Y; entier X,Y;
      MEMBRE := si ¬ ATOM (X) ∧ ATOM (Y)    alors  faux
                                            sinon
                si  ATOM (X) ∧ ATOM (Y)  alors EQ (X,Y)
                                         sinon
                si EGAL (X,CAR(Y)) alors vrai
                                   sinon
                MEMBRE (X,CDR(X));

booléen procédure JUXTAPOSER (X,Y); valeur X,Y; entier X,Y;
      JUXTAPOSER := si X = 1 alors Y
                            sinon
                    CONS(CAR(X), JUXTAPOSER(CDR(X),Y));

entier procédure RETOURNER (X); valeur X; entier X;
     RETOURNER := RETOURNER 1 (X,1);

entier procédure RETOURNER 1(X,Y); valeur X,Y; entier X,Y;
     début entier R;
        R := X;
        RETOURNER1 := si X = 1 alors Y sinon
                      RETOURNER1 (CDR(R), CONS(CAR(R),Y))
     fin;
```

```
entier procedure  SUBST (X, Y,Z); valeur X, Y, Z; entier X,Y,Z;
       SUBST := si ATOM (Z) alors
                ( si EQ (Z,Y) alors  X  sinon  Z)
                           sinon
                CONS (SUBST (X,Y, CAR(Z) ), SUBST (X,Y, CDR(Z) ));

entier procedure  UNION (M, N); valeur M, N; entier  M,N;
       UNION :=  si  ATOM (M) alors
                 ( si MEMBRE (M,N) alors N sinon CONS (M, N) )
                                sinon
                 si  MEMBRE (CAR(M), N) alors  UNION (CDR(M),N)
                                          sinon
                CONS (CAR (M), UNION (CDR (M), N) );

entier procedure  INTERSECTION (M,N); valeur M, N; entier M, N ;
       INTERSECTION := si  M = 1 alors 1
                                  sinon
                       si MEMBRE (CAR(M),N) alors
                       CONS (CAR(M), INTERSECTION (CDR(M),N) )
                                              sinon
                       INTERSECTION (CDR (M), N) ;

entier procedure  MATCH (KK, M); valeur KK, M; entier KK, M;
       MATCH := si KK = 1 alors 1
                            sinon
                si M = 1 alors 1
                           sinon
                si EQ (CAR(KK), CAR(M) ) alors CAR (KK)
                                         sinon
                MATCH (CDR (KK), CDR (M) ) ;
```

```
entier procedure     LISTE DES ATOMES (S); valeur S; entier S ;
       LISTE DES ATOMES := si  ATOM (S) alors  S
                                          sinon
                           UNION (LISTE DES ATOMES (CAR (S) ),
                                  LISTE DES ATOMES (CDR (S) ) );

entier procedure   RETOURNER SUPER (M); valeur M; entier M;
       RETOURNER SUPER := si  ATOM (M) alors M
                                       sinon
                          si  M = 1 alors 1
                                    sinon
                          JUXTAPOSER ( RETOURNER SUPER (CDR (M) ),
                          CONS (RETOURNER SUPER (CAR (M)), 1) );

entier procedure  SUPPRIMER (X, M); valeur X, M; entier X, M;
       SUPPRIMER := si  M = 1 alors 1
                              sinon
                     si ATOM (M) alors (si EQ (X,M) alors 1
                                                    sinon Y)
                                    sinon
                     si EQ (X, CAR (M) ) alors CDR (M)
                                         sinon
                     CONS (CAR (M), SUPPRIMER (X, CDR (M) ) );

entier procedure  TWIST (S); valeur S; entier S;
       TWIST : = si  ATOM (S) alors S
                              sinon
                      CONS (TWIST (CDR(S) ), TWIST (CAR (S) );

entier procedure  PAIRE (M, N); valeur M, N; entier M, N;
       FAIRE := si M = 1 alors 1

                           sinon
                  CONS (CONS(CAR(M), CAR(N) ), PAIRE (CDR(M),
                         CDR (N) ) );
```

```
entier procedure  PAIRE LISTE (X,Y,A);valeur X,Y,A; entier X,Y,A);
      PAIRE LISTE:= si  X = 1 alors A
                              sinon
                    CONS (CONS(CAR(X), CAR (Y) ), PAIRE LISTE
                                (CDR(X), CDR (Y), A) );

entier procedure ASSOCIER (X,A); valeur X,A; entier X,A;
      ASSOCIER := si EGAL (CAAR (A), X) alors  CAR (A)
                                          sinon
                  ASSOCIER (X, CDR (A) );

entier procedure  SUBLIS (A,Y) ; valeur A, Y ; entier A, Y ;
      SUBLIS := si ATOM (Y) alors SUB2 (A,Y)
                           sinon
                CONS (SUBLIS (A,CAR(Y)),SUBLIS (A,CDR(Y) ) );

 entier procedure  SUB2 (A,Z); valeur A,Z ; entier A,Z;
      SUB2:= si  A = 1 alors Z
                       sinon
            si EQ (CAAR (A),Z); alors  CDAR (A)
                                sinon
             SUB2 (CDR(A), Z);

entier procedure  APPLIQUER (X, F); valeur X; entier X;
                                     entier procedure F;
      APPLIQUER := si X = 1 alors 1
                           sinon
                  CONS (F(X), APPLIQUER (CDR(X),F) );
```

III - 3 APPLICATION A UN PROBLEME DE DERIVATION FORMELLE

Dans ce chapitre nous allons exposer un exemple d'utilisation des procédures de liste, qui traite un problème de dérivation formelle. Cet exemple illustre bien la puissance des procédures de liste dans la manipulation de symboles.

entier procédure DIFF (⟨expression entière représentant la fonction à dériver⟩ , ⟨expression entière représentant la variable par rapport à laquelle on veut dériver⟩).

La procédure DIFF calcule la dérivée de la fonction Y par rapport à la variable X, et ne traite que le cas où la fonction à dériver est polynomiale ou trigonométrique. La fonction Y peut aussi être une fonction de fonctions.

Exemple : Y = SIN (SIN (X + A) * COS (X))

La donnée et le résultat sont écrits dans une variante de la notation préfixée : l'opérateur est toujours le premier élément d'une liste (ou sous-liste) et il doit être appliqué aux autres éléments de cette liste (ou sous-liste).

Exemple : L'expression :

X * (X + A) * Y,

pour être traitée par DIFF, doit s'écrire :

(* X (+ X A)*Y)

Le programme examine séquentiellement la liste qui représente la fonction à dériver : quand il reconnaît un opérateur, il exécute une action déterminée qui dans le cas général est la

construction d'une nouvelle chaîne au moyen de la procédure CONS.

Voyons en détail le fonctionnement de la procédure DIFF. Supposons qu'on veuille dériver une fonction Y par rapport à la variable X. Plusieurs cas peuvent se présenter :

1.- La fonction se réduit à un seul symbole : alors le programme examine si ce symbole est X; si c'est le cas elle donne "1" comme résultat, autrement "0".

2.- La fonction comporte l'opérateur multiplicatif : alors la liste ou(sous-liste) est nécessairement de la forme :

(* u v w .....)

et comme

(u * v * w * .....)' = (u' * v * w ....) +
(u * v' * w ....) +
(u * v * w' ...) + ...

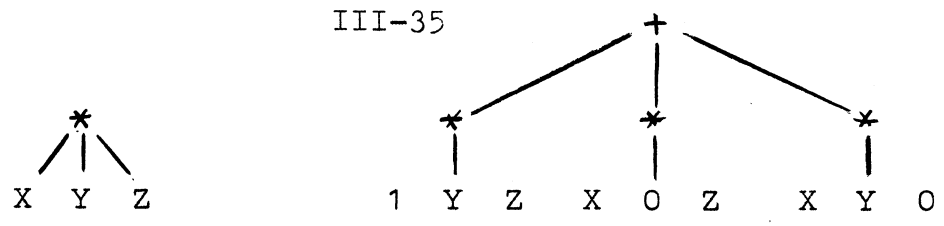
le programme construit la liste dont le premier élément est "+" puis juxtapose autant de sous-listes qu'il y a d'éléments sous la portée de l'opérateur "*". Ces sous-listes recopient la liste d'entrée en ne remplaçant chaque fois qu'un élément par sa dérivée. Exemple : soit la formule à dériver :

(* X Y Z),

alors la dérivée sera la liste :

(+ (* 1 Y Z) (* X 0 Z) (* X Y 0))

On peut remarquer que la vaiante de la nctation préfixée adoptée peut être représentée au moyen d'un arbre. Chaque noeud de l'arbre représente un opérateur et chaque feuille une opérande. D'un noeud peuvent sortir des feuilles (opérandes) ou d'autres branches (sous-listes). Ainsi à l'exemple donné, correspondent les arbres suivants :



Donnée Dérivée

3.- La fonction comporte l'opérateur additif: alors le programme construit la liste dont la partie car est "+", et la partie cdr les dérivées des éléments sous la portée de l'opérateur "+". Ce processus obéit à l'identité :

(u + v + w + ...) ' = u' + v' + w' + ...

exemple : Donnée : (+ A X X)
Dérivée: (+ 0 1 1)

ou graphiquement :

Donnée Dérivée

4.- La fonction comporte l'opérateur soustractif: alors le processus est exactement le même que dans le cas d'addition, le symbole "-" remplaçant le symbole "+". Si l'opérateur est " θ " (moins unitaire). Le programme met le signe θ devant la dérivée de l'unique élément qui est sous sa portée.

5.- L'opérateur est SIN ; le programme construit la liste dont la partie car est "* ", la partie cdr étant constituée de COS, de l'opérande de SIN et finalement de sa dérivée par rapport à X. Exemple : Donnée : (SIN (+ X A))
Dérivée : (*(COS (+ X A) (+ 1 0))

ou graphiquement :

Donnée Dérivée

6.- L'opérateur est COS, le processus de construction de la dérivée est analogue à celui utilisé dans le cas où l'opérateur est SIN. I l faut dans ce cas remplacer l'atome COS par l'atome SIN dans la liste résultante qui doit comporter aussi un signe ⊖ comme premier élément.

Les opérations de base étant définies, on utilise leur combinaison et la récursivité pour construire l'algorithme général.

Passons maintenant aux considérations nécessaires à l'utilisation de DIFF. Il faut que les variables entières utilisées dans le corps de la procédure soient déclarées préalablement. Toutes ces variables servent à pointer les listes, qui doivent être rangées en mémoire préalablement en utilisant NOMMER et LIRE LISTE. Voici l'exemple d'un programme qui imprime la dérivée d'une fonction Y perforée sur carte donnée. Les procédures LISP, les variables IPD, TEMP et les tableaux TCAR et TCDR [I:N] doivent être déclarés dans un bloc englobant le bloc ci-dessous :

```
début entier ZERO, UN, PLUS, MOINS, MOINS UNITAIRE,
             MULTIPLIER, SINUS, COSINUS, NIL, X, Y;

    <en-têtes et corps des procédures DIFF, EGAL et APPLIQUER>

PROGRAMME :

PREPARER (1000) ;
ZERO                    := NOMMER ('0U') ;
UN                      := NOMMER ('1U') ;
PLUS                    := NOMMER ('+U') ;
MOINS                   := NOMMER ('-U') ;
MOINS UNITAIRE          := NOMMER ('⊖U') ;
MULTIPLIER              := NOMMER ('*U') ;
```

```
SINUS                    := NOMMER ('SIN');
COSINUS                  := NOMMER ('COS');
NIL                      := NOMMER ('()');
X                        := NOMMER ('X');

Y := LIRE LISTE ;
LIBERER (ECRIRE LISTE (DIFF (Y,X) ))

fin;
```

En utilisant DIFF il est facile d'écrire un programme pour obtenir la n-ième dérivée d'une fonction. Nous le présentons sous la forme de la procédure DIFFN à trois paramètres le premier représente la fonction à dériver, le deuxième la variable par rapport à laquelle on veut obtenir la dérivée et la troisème spécifie l'ordre de la dérivée. Cette procédure est très simple et se dispense de commentaire; elle sera présentée avec DIFF à la fin de ce chapitre.

A titre d'exemples supplémentaires nous avons trouvé intéressant de donner le listage des deux procédures PREFIXER et INTERFIXER qui permettent, l'une de réduire une expression arithmétique complètement parenthèsée à la forme préfixée utilisée par DIFF, l'autre d'effectuer la transformation inverse. Ainsi l'appel :

```
LIBERER (ECRIRE LISTE (INTERFIXER (DIFFN (PREFIXER (Y),X,N) )))
```

permet l'évaluation et l'impression de la N-ième dérivée d'une expression complètement parenthèsée représentée par Y. Dans les pages trouvera le listage des procédures DIFF, DIFFN, INTERFIXER, PREFIXER .

```
entier procédure DIFF (Y,X); valeur X,Y; entier X,Y;
    début entier procédure DIFFCAR (Z); valeur Z; entier Z;
          DIFFCAR := DIFF(CAR(Z),X);

            entier procédure PROD(Z); valeur Z; entier Z;

              début entier procédure FACTEUR (W);valeur W;entierW;

                FACTEUR := si ¬ EGAL (W,Z) alors CAR(W)
                                           sinon DIFF (CAR(W),X);

              PROD:=CONS(MULTIPLIER,APPLIQUER(CDR(Y),FACTEUR) )
              fin procédure PROD;

DIFF:=si ATOM(Y)alors(si EQ(X,Y)alors UN sinon ZERO)
                    sinon
       si EQ(CAR(Y),PLUS)
            alors CONS(PLUS,APPLIQUER(CDR(Y),DIFFCAR) )
                                       sinon
       si EQ(CAR(Y),MOINS)
            alors CONS(MOINS,APPLIQUER(CDR(Y),DIFFCAR) )
                                       sinon
       si EQ(CAR(Y),MULTIPLIER)
            alors CONS(PLUS,APPLIQUER(CDR(Y),PROD) )
                                       sinon
       si EQ(CAR(Y),SINUS)
            alors CONS(MULTIPLIER,
                       CONS(CONS(COSINUS,CDR(Y)),
                            CONS(DIFF(CDR(Y),X),NIL ) ))
                                       sinon
       si EQ(CAR(Y),COSINUS)
            alors CONS(MOINS UNITAIRE,
                       CONS(CONS(MULTIPLIER,
                            CONS(CONS(SINUS,CDR(Y) ),
                                 CONS(DIFF(CDR(Y),X), NIL) )),
                                                      NIL))
                                       sinon
```

```
        si EQ(CAR(Y),MOINS UNITAIRE)
              alors CONS(MOINS UNITAIRE,
                         CONS(DIFF(CDR(Y),X),NIL))
                    sinon DIFF(CAR(Y),X)
    fin procédure DIFF;

entier procédure PREFIXER (Y); valeur Y; entier Y;
      PREFIXER := si ATOM (Y) alors Y
                             sinon
                  si CDR (Y) = NIL alors PREFIXER (CAR(Y)
                                   sinon
                  si EQ (CADR(Y), PLUS) alors AUX (PLUS,Y)
                                        sinon
                  si EQ (CADR(Y),MOINS) alors AUX (MOINS,Y)
                                        sinon
                  si EQ (CADR(Y),MULTIPLIER) alors AUX (MULTIPLIER,Y)
                                             sinon CONS
                       (PREFIXER (CAR(Y)),
                       CONS (PREFIXER
                       (CDR(Y)),NIL));
entier procédure AUX (OPERATION,Y); valeur OPERATION,Y;
entier OPERATION,Y;
      AUX := CONS (OPERATION, CONS (AUX1(CAR(Y)), RESTE (CDDR(Y))));

entier procédure AUX1(Y), valeur Y; entier Y;
      AUX1 := si ATOM(Y) alors Y sinon PREFIXER(Y);

entier procédure RESTE(Y); valeur Y; entier Y;
      RESTE := CONS (AUX1(CAR(Y)) , si CDR(Y) = NIL alors NIL sinon
                                                      RESTE
                                                      (CDDR(Y)))
```

```
entier procédure INTERFIXER (Y); valeur Y; entier Y;

      INTERFIXER := si ATOM (Y) alors Y
                                 sinon
                    si EQ (CAR(Y), PLUS) alors AIDE (PLUS,CDR(Y))
                                         sinon
                    si EQ (CAR(Y), MOINS)alors AIDE (MOINS,CDR(Y))
                                         sinon
                    si EQ (CAR(Y), MULTIPLIER) alors AIDE
                          (MULTIPLIER,CDR(Y))
                                                sinon CONS
                          (INTERFIXER(CAR(Y)),
                          CONS (INTERFIXER (CDR(Y)), NIL));

entier procédure AIDE (OPERATION,Y); valeur OPERATION, Y;
entier OPERATION,Y;

      AIDE := si Y = NIL alors NIL
                         sinon CONS(INTERFIXER(CAR(Y),
                                    si CDR(Y) = NIL alors NIL
                                                    sinon CONS
                                     (OPERATION,AIDE(OPERATION,CDR(Y))))

entier procédure DIFFN (Y,X,N); valeur Y,X,N; entier Y,X,N;

      DIFFN := si N = 1 alors DIFF (Y,X) sinon DIFFN (DIFF(Y,X),X,N-1);
```

B I B L I O G R A P H I E

1 - A.C.M. Conférence on symbol manipulation
May 20-21 Philadelphia, Pa.
Comm. A.C.M. Vol. 3/N°4/ April 1960

2 - ROBERT F. ROSIN
An introductory problem in symbol manipulation for the student. Comm. A.C.M. Vol. 3/N° 9/September 1960

3 - J.H. WEGSTEIN and W.W. YOUDEN
A string language for symbol manipulation based on ALGOL 60
Comm. A.C.M. VOL 5/ N° 1/ January 1962

4 - Remarks on ALGOL and symbol manipulation
A.C.M. 2/September 1959

5 - R. FLOYD
A descriptive language for symbol manipulation
J. of the A.C.M. 8,4 October 1961

6 - SMITH D.D.
Character manipulation in 7090 FORTRAN
Comm. A.C.M. 6/8/ 1963

7 - A suggested method of making juller use of strings in ALGOL
Comm. A.CM. 6/4/1963

8 - A proposal for input-Output conventions in ALGØL 60.
A report of the subcommitee on ALGOL of the A.C.M.
Comm. AC.M. Vol. 7/N° 5/ May 1964

9 - M.J. BAILEY, M.P. BARNETT and P.B. BURLESON
Symbol manipulation in FORTRAN - SASPI subroutines
Comm. A.C.M. Vol. 7/N° 6/June 1964

10 - DUCAN F.G.
Input and output for ALGØL 60 on KDF 9
Computer Journal 5/4/1963

11 - D.J. FARBER, R.E. GRISWOLD and I.P. POLONSKY, SNOBOL
A string manipulation language
Journal A.C.M. January 1964 Vol. 11 N° 1

12 - WEIZENBAUM J.
Knotted list structures.
Comm. A.CM. Vol. 5/N° 3/ March 1962

13 - GREEN B. F.
Computer languages for symbol manipulation
IRE Trans. on Human Factors in Elec. HFE 2,1 March 1961
reprinted in IRE Trans on Electronic Computers Vol. EC 10,
N° 4 Déc. 1962

14 - BARNETT M.P.
Low level language subroutines for use within FORTRAN
Comm. A.C.M. 4/1961

15 - HENRY J. BOWLDEN
A list-type storage technique for alphanuméric information
Comm. A.C.M. Vol. 6/N°8/August 1963

16 - JULIEN GREEN
IBM White plains NEW YORK
Symbol manipulation in X TRAN
Comm A.C.M. Vol. 3/N° 4/April 1960

17 - KLAUS G.K. BROKATE
On the proposal of D. KNUTH concerning
Input-Output within ALGOL
ALGOL BULLETIN N° 17 1964 P. 12

18 - NIKLAUS WIRTH
A proposal on string manipulation in ALGOL 60
ALGOL BULLETIN N° 17 p. 13

19 - G. SOLLIN 40 Rue Perronet NEUILLY SUR SEINE
Extension de ALGOL au traitement des textes
février 196

20 - VAN WIJNGAARDEN
A possible extension to ALGOL 60 : string 1964

21 - Report on Input-Output procédures for ALGOL 60 b I.F.I.P.
Secretary UTMAN
Comm. A.C.M. October 1964

22 - F.G. DUNCAN and A. VAN WIJNGAARDEN
ALGOL BULLETIN N° 16 May 1964 p. 24

23 - R.L. COOK
String handling in ALGOL 60
ALGOL BULLETIN N° 18 October 1964

24 - H. BEKIC
Some comments on the A.C.M. I/O proposal
ALGOL BULLETIN N° 18

26 - B. RANDELL
String quotes
ALGOL BULLETIN N° 18

27 - W. L. VAN DER POEL
A procedure for string handling
ALGOL BULLETIN N° 18

28 - Comments on the X3.4.2's report on Input-Output facilities for ALGØL
AFNOR June 30/1964

29 - P.J. VAN DE LARSCHOT, J. NEDERKOORN
Strings in ALGOL, communication personnelle
Amsterdam 1963

30 - D. BOBROW, J. WEIZENBAUM
List processing and Extension of language facility by embedding
I.E.E.E. Transactions on Electronic Computers
August 1964

31 - D. BOBROW, B. RAPHAEL
A comparison of list processing computer languages
Comm. A.C.M. April 1964

32 - E. BERKELEY, D. BOBROW
The programming language LISP : Its operation and applicatior
Information International, Inc., Cambridge, Mass, 1964

33 - Mc CARTHY
Recursive functions of symbolic expressions and their
computation by machine, Part I, comm. A.C.M., April 1960

34 - J. Mc CARTHY
LISP 1,5 programmer's manual
The M.I.T Press, Cambridge, Mass., 1962

35 - I.M.P. by B. SVEJGAARD and P. LINDBLAD
Note personnelle.
Københavns Universitets Matematiske Institut

36 - The programming language LISP
Computers and Automation September 1964

37 - F. G. DUCAN , A. VAN WIJNGAARDEN
ALGOL 60 as a basis for ALGOL development
ALGOL BULLETIN N° 16 May 1960

38 - Etude prospective de la théorie des langages et des processus
de transposition dans l'autre
Deuxième compte rendu d'activités
D.G.R.S.T.

39 - J. E. SAMMET and E.R. BØND
Introduction to FORMAC
I.E.E.E. Trans. on Electronic Computers
N° 4/August 1964

40 - V. YNGVE
Comit programmer's reference manual
Massachusetts Institute of Technology Press
Cambridge 1961

41 - M. V. WILKES
List and why they are useful
Proceedings of the 19 th national conférence
PHILADELPHIA, PENNSYLVANIA August 25-27, 1964

42 - E. BOND, M. AUSLENDER
FORMAC. An experimental formula manipulation.

43 - WILKES M. V.

An expriment with a self-compiling compiler for a simple list-processing language. Annual review of automatic programming
Vol. 4 PERGAMON PRESS OXFORD 1964

44 - Etude des langages de listes
D.G.R.S.T. rapport d'activité
Société d'information appliquée

45 - Language FORTRAN IV Référence 20 1055

46 - BAECKER H. D.
Mapped - list structures.
Comm. A.C.M. Vol. 6/ N° 8 August 1963

47 - BANERJI R. B.
The description list of concepts
Comm. A.C.M. Vol. 5/N°8/ August 1962

48 GELERNTERH, J.R. HANSEN and C.L. GERBERICH
A fortran. Compiled list processing language
J. A.C.M. Vol. 7/N°2/ April 1960

49 H. BEKIC
IBM laboratory VIENNA
An Input-Outpu proposal for ALGOL based on FORTRAN IV Input-Output
Communication personnelle 6 March 1964

50 A.J. PERLIS and RENATO ITURRIAGA
carnegie Institute of Technology, Pittsbugh, PA).
An extension to ALGOL for manipulating Formulae.

51 - J. COHEN - NGUYEN HUU DUNG
Définition de procédures LISP en ALGOL. Exemples d'utilisation
Note technique Février 1965
(Institut de Mathématiques appliquées. St Martin d'Hères
Faculté des Sciences. GRENOBLE)

52 WEIZENBAUM J.
Symmetric list Processor
Comm. A.C.M. Vol. 6/N°9 September 1963

53 - J.C. BOUSSARD
Etude et réalisation d'un compilateur ALGOL 60 sur calculatrice électronique du type IBM 7090/94 et 7040/44
Thèse de Sciences appliquées présentée le 16/6/1964 à GRENOBLE

54 - L. BOLLIET - N. GASTINEL - P.J. LAURENT
Un nouveau langage scientifique ALGOL, manuel pratique
HERMANN PARIS 1964

A P P E N D I C E

Voici le listage des procédures LIRE SYMBOLE, ECRIRE SYMBOLE,

LONGUEUR et leurs procédures auxiliaires.

```
'PROCEDURE' LIRE SYMBOLE( CANAL,CHAINE,DESTINATION ) FF
'VALEUR' CANAL FF 'ENTIER' CANAL, DESTINATION FF
'CHAINE' CHAINE FF                                                    'CODE'
          TRA     DEBUEN
BANLIR    CLA     ZBOOL
          TZE     BANDEN
          TSX     S.OPEN,4
          PZE     BOGE
          STZ     ZBOOL
          TRA     BANDEN
ZBOOL     DEC     1
TROPE     CALL    JOBOU(LUTS)
          TRA     FINNE+1
ZTAMPO    MACRO   ZP0,ZP1,ZP2,ZP3,ZP4,ZP5,ZP6,ZP7
ZP0       AXT     1,1
          TSX     ZP1,4
          SXA     *+1,2
          CLA     **
          STO     ZP2
          SUB     =120
          TPL     ZP3
          LDQ     ZP2
          MPY     =6
          STQ     CONTRO
ZP7       CLA     =6
          STO     ZP4
          TXI     *+1,2,1
          SXA     *+1,2
          LDQ     **
ZP6       LGL     6
          STQ     ZP5
          ANA     =077
          STO     DICT,1
          TXI     *+1,1,1
          CLA     ZP4
          SUB     =1
          STO     ZP4
          LDQ     ZP5
          TNZ     ZP6
          CLA     ZP2
          SUB     =1
          STO     ZP2
          TNZ     ZP7
          ENDM    ZTAMPO
```

```
        EXTERN  BOGE
ZZI     DEC     11
ZZJ     DEC     5
PROVEN  BSS     14
MEMOZZ  BSS     1
MEMOIB  BSS     1
MEMOII  BSS     1
CONTRO  BSS     1
ZM      BSS     1
ZN      BSS     1
ZI      DEC     11
ZJ      DEC     5
MEMOZ   BSS     1
MEMOI   BSS     1
        ZTAMPO  DEBUEN,P1-2,ZN,TROPE,ZM,MEMOII,MIMO,MIJBIN
        CLA     F1+1
        TMI     ABSURE
        TZE     CARTEN
        SUB     =1
        TZE     BANLIR
        SUB     =2
        TZE     UNECEN
        TMI     ABSURE
        TPL     ABSURE
ABSURE  CALL    JOBOU(CEREN)
        TRA     FINNE+1
CEREN   PZE     1
        PZE     EREN,,5
EREN    BCI     5,*ERREUR CANAL LIRESYMBOLE
UNECEN  CLA     LOCA
        TRA     CENTRA
BANDEN  CLA     ZZJ
        ADD     =1
        STO     ZZJ
        SUB     =6
        TNZ     MIMOB
        STZ     ZZJ
        CLA     ZZI
        ADD     =1
        STO     ZZI
        SUB     =12
        TNZ     MIJOBB
        STZ     ZZI
ZBANDE  TSX     S.GETL,4
GET     PZE     BOGE,,PROVEN
        LXD     GET,2
        SXA     MEMOZZ,2
MIJOBB  CLA     MEMOZZ
        ADD     ZZI
        STO     MEMOIB
```

```
'PROCEDURE' ECRIRE SYMBOLE( CANAL,CHAINE,SOURCE ) FF
'VALEUR' CANAL, SOURCE FF 'ENTIER' CANAL,SOURCE FF
'CHAINE' CHAINE FF                                                    'CODE'
         TRA      DEBUEC
BANECR   CLA      ZBOOE
         TZE      BANDES
         TSX      S.OPEN,4
         PON      BOGE
         STZ      ZBOOE
         TRA      BANDES
ZBOOE    DEC      1
LIST1    PZE      1
         PZE      LOCA,6,1
LOCA     BSS      1
TROPS    CALL     JOBOU(LUTS)
         TRA      FINAL
FINAL    EQU      FINFIN+1
LARTE    PZE      MBLOC
NBLOC    DEC      1
NCARAC   PZE
MBLOC    BSS      12
         BCI      2,
SACA     SAC      MBLOC+12,2,
         EXTERN   JOBOU
         EXTERN   JOBPP
DICT     BES      720
RX2      BSS      1
MEMOE    BSS      1
MZ       BSS      1
NZ       BSS      1
LUTS     PZE      1
         PZE      ERO,,6
ERO      BCI      7,*DEPASSEMENT POUR LE DICTIONNAIRE
RX2B     BSS      1
NBLOB    DEC      1
NCARAB   PZE
SACB     SAC      MBLOB+12,2,
MBLOB    BSS      12
         BCI      2,
ZOTO     MACRO    CARTES,RX2,NBLOC,SUTES,NCARAC,SACA,MBLOC,VIDER
CARTES   LXA      RX2,2
         CLA      NBLOC
         TZE      SUTES
         AXT      12,2
         STZ      NBLOC
SUTES    CLA      NCARAC
         ALS      18
         ORA      SACA
         SLW      *+2
         CLA      LOCA
```

```
         SAC     MBLOC+12,2,
         CLA     NCARAC
         ADD     =1
         STO     NCARAC
         SUB     =6
         TNZ     *+4
         STZ     NCARAC
         TXI     *+1,2,-1
         TXL     VIDER,2,0
         SXA     RX2,2
         TRA     FINAL
         ENDM    ZOTO
ZOT1     MACRO   NBLOC,MBLOC,FINAL
         CLA     =1
         STO     NBLOC
         AXT     12,2
         CAL     =O-377777777777
         STO     MBLOC+12,2
         TIX     *-1,2,1
         TRA     FINAL
         ENDM    ZOT1
         ZOTO    CARTES,RX2,NBLOC,SUTES,NCARAC,SACA,MBLOC,VIDER
VIDER    CALL    JOBPP(LARTE)
         ZOT1    NBLOC,MBLOC,FINAL
         ZOTO    BANDES,RX2B,NBLOB,SUTEB,NCARAB,SACB,MBLOB,VIDEB
VIDEB    TSX     S.PUTL,4
         PZE     BOGE,,MBLOB
         ZOT1    NBLOB,MBLOB,FINAL
         ZTAMPO  DEBUEC,P2-2,NZ,TROPS,MZ,MEMOE,MO,BIN
         CLA     =5
         STO     CINQ
COCO     TXI     *+1,1,-1
         CLA     DICT,1
         SUB     =077
         TZE     COCO
         STZ     CONTRO
         SXA     CONTRO,1
KIFIN    CLA     F2+3
         TMI     ABSURD
         TNZ     MOL
         CALL    JOBOU(LISTER)
         TRA     FINAL
MOL      SUB     CONTRO
         TMI     NORMAL
         TZE     NORMAL
         CALL    JOBOU(LISTEC)
         TRA     FINAL
ABSURD   CALL    JOBOU(LISTAB)
         TRA     FINAL
LISTER   PZE     1
         PZE     ERREUR,0,7
ERREUR   BCI     7,*CE CARACTERE N EXISTE PAS A L ENTREE*
CINQ     BSS     1
LISTAB   PZE     1
         PZE     ER1,0,2
```

```
MIMOB   LDQ*    MEMOIB
        LGL     6
        STQ*    MEMOIB
        TRA     CENTRA-1
CARTEN  CLA     ZJ
        ADD     =1
        STO     ZJ
        SUB     =6
        TNZ     MIMOT
        STZ     ZJ
        CLA     ZI
        ADD     =1
        STO     ZI
        SUB     =12
        TNZ     MIJOB
        STZ     ZI
        EXTERN  JOBIN
        CALL    JOBIN
        STO     MEMOZ
        ALS     2
        PBT
        TRA     MIJOB
        ALS     16
        ORA     SAUVE
        TSX     S.SCCR,4
        TMT     14
        TSX     S.SCCR,4
        MSM     S.SCDI
        TRA     S.JXIT
SAUVE   PZE     S.SAVE
MIJOB   CLA     MEMOZ
        ADD     ZI
        STO     MEMOI
MIMOT   LDQ*    MEMOI
        LGL     6
        STQ*    MEMOI
        ANA     =077
CENTRA  STO     F1+3
        AXT     0,1
CONTI1  TXI     *+1,1,1
        CLA     F1+3
        SUB     DICT,1
        TZE     FINEN
        CLA     CONTRO
        ALS     18
        STD     *+1
        TXL     CONTI1,1,**
        AXT     0,1
FINEN   TSX     P1-3,4
        SXA     *+2,2
        PXA     ,1
FINNE   STO     **
'FCODE' FF
```

```
ER1     BCI     5,*ERREUR CANAL ECRIRESYMBOLE
LISTEC  PZE     1
        PZE     ERREC,0,7
ERREC   BCI     7,*CE CARACTERE N EXISTE PAS A LA SORTIE*
NORMAL  LXA     F2+3,1
        CLA     DICT,1
        STO     LOCA
        CLA     F2+1
        TMI     ABSURD
        TZE     CARTES
        SUB     =1
        TZE     BANECR
        SUB     =1
        TZE     PAPIER
        SUB     =1
        TZE     FINAL
        TPL     ABSURD
PAPIER  CALL    JOBOU(LIST1)
FINFIN  MSM     LIST1+1
'FCODE'           FF
```

```
        'ENTIER' 'PROCEDURE' LONGUEUR(CHAINE) FF 'CHAINE' CHAINE FF 'CODE'
        TRA     *+2
MOSCOU  BSS     1
        TSX     P3-1,4
        SXA     *+1,2
        LDQ     **
        MPY     =6
        STQ     F3
        PXA     ,2
        SXA     *+1,2
        ADD     **
        STA     *+1
        CLA     **
ONTINU  LGR     6
        STO     MOSCOU
        PXA     ,
        LGL     6
        SUB     =077
        TNZ     FIN+1
        CLA     F3
        SUB     =1
        STO     F3
        CLA     MOSCOU
FIN     TRA     ONTINU
'FCODE'  FF
```

```
'PROCEDURE' VIDER(CANAL) FF 'VALEUR' CANAL FF 'ENTIER' CANAL FF    'CODE'
        CLA     F4+1
        TZE     CARVID
        SUB     =1
        TZE     BANVID
        CALL    JOBOU(ZEZ)
        TRA     ZSORT
CARVID  CALL    JOBPP(LARTE)
        ZOT1    NBLOC,MBLOC,ZSORT
BANVID  CLA     NBLOB
        TNZ     ZCLOSE
        TSX     S.PUTL,4
        PZE     BOGE,,MBLOB
        ZOT1    NBLOB,MBLOB,ZCLOSE
ZEZ     PZE     1
        PZE     ZEZE,0,3
ZEZE    BCI     3,*ERREUR POUR VIDER
ZCLOSE  TSX     S.CLSE
        PZE     BIGE
ZSORT   NOP
'FCODE'   FF


'PROCEDURE' PAGE SUIVANTE FF                                        'CODE'
        TRA     ZPAGE
ZLIS1   PZE     1
        PON
ZPAGE   CALL    JOBOU(ZLIS1)
'FCODE'  FF


'PROCEDURE' ALALIGNE        FF                                      'CODE'
        TRA     ZALA
LRCH    PZE     1
        PZE
ZALA    CALL    JOBOU(LRCH)
'FCODE'      FF
```

VU,

Grenoble, le

Le Président de la Thèse

VU,

Grenoble, le

Le Doyen de la Faculté des Sciences

VU et permis d'imprimer,

Le Recteur de l'Académie de Grenoble