



**HAL**  
open science

# SYPAC : un système expérimental de calcul formel en Pascal

Robert-Michel Di Scala

► **To cite this version:**

Robert-Michel Di Scala. SYPAC : un système expérimental de calcul formel en Pascal. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1982. Français. NNT : . tel-00300279

**HAL Id: tel-00300279**

**<https://theses.hal.science/tel-00300279>**

Submitted on 18 Jul 2008

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# THESE

*présentée à*

**l' Université Scientifique et Médicale de Grenoble**

*pour obtenir le grade de*

**DOCTEUR DE 3<sup>ème</sup> CYCLE**

**. Informatique.**

*par*

**Robert-Michel di SCALA**



**SYPAC :**

**UN SYSTEME EXPERIMENTAL  
DE CALCUL FORMEL EN PASCAL**



**Thèse soutenue le 30 mars 1982 devant la Commission d'Examen :**

**Monsieur P. JULLIEN : Président**

**Messieurs M. BERGMAN  
J. CALMET  
J. DELLA DORA  
P.C. SCHOLL** } **Examineurs**



UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

année scolaire 1980-1981

Président de l'Université : M. J.J. PAYAN

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS DE 1ère CLASSE

Mlle	AGNIUS DELORD Claudine	Biophysique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Clinique dermatologie
	AMBROISE THOMAS Pierre	Parasitologie
	ARNAUD Paul	Chimie
	ARVIEU Robert	Physique nucléaire
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale A
	BEAUDOING André	Clinique pédiatrie et puériculture
	BELORISKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
Mme	BERIEL Hélène	Pharmacodynamie
M.	BERNARD Alain	Mathématiques pures
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques pures
	BEZES Henri	Clinique chirurgicale & traumatologie
	BILLET Jean	Géographie
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET EYMARD Joseph	Clinique Hépto-gastro-entérologie
Mme	BONNIER Jane-Marie	Chimie générale
MM.	BOUCHERLE André	Chimie et toxicologie
	BOUCHET Yves	Anatomie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie

.../...

MM. BUTEL Jean	Orthopédie
CABANEL Guy	Clinique rhumatologie et hydrologie
CARLIER Georges	Biologie végétale
CAU Gabriel	Médecine légale et toxicologie
CAUQUIS Georges	Chimie organique
CHARACHON Robert	Clinique O.R.L.
CHATEAU Robert	Clinique neurologique
CHIBON Pierre	Biologie animale
COEUR André	Chimie analytique et bromotologique
COUDERC Pierre	Anatomie pathologique
CRABBE Pierre	C.E.R.M.O.
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude	M.I.A.G.
DELORMAS Pierre	Pneumo-phtisiologique
DENIS Bernard	Clinique cardiologique
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DODU Jacques	Mécanique appliquée IUT 1
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique
GASTINEL Noël	Analyse numérique
GAVEND Jean-Michel	Pharmacologie
GEINDRE Michel	Electro-radiologie
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
JANIN Bernard	Géographie
JEANNIN Charles	Pharmacie galénique
JOLY Jean-René	Mathématiques pures
KAHANE André	Physique
KAHANE Josette	Physique
KLEIN Joseph	Mathématiques pures
KOSZUL Jean-Louis	Mathématiques pures
LACAZE Albert	Hermodynamique
LACHARME Jean	Biologie cellulaire
LAJZEROWICZ Joseph	Physique

Mme	LAJZEROWICZ Jeannine	Physique
MM.	LATREILLE René	Chirurgie thoracique
	LATURAZE Jean	Biochimie pharmaceutiques
	LAURENT Pierre	Mathématiques appliquées
	LE NOC Pierre	Bactériologie virologie
	LLIBOUTRY Louis	Géophysique
	LOISEAUX Jean-Marie	Sciences nucléaires
	LOUP Jean	Géographie
	LUU DUC Cuong	Chimie générale et minérale
	MALINAS Yves	Clinique obstétricale
Mlle	MARIOTTE Anne-Marie	Pharmacognosie
MM.	MAYNARD Roger	Physique du solide
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	NEGRE Robert	Mécanique IUT 1
	MOZIERES Philippe	Spectrométrie physique
	OMONT Alain	Astrophysique
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY PEYROULA Jean-Claude	Physique
	PERRET Jean	Sémeiologie médicale (neurologie)
	PERRIER Guy	Géophysique
	PIERRARD Jean-Marie	Mécanique
	RACHAIL Michel	Clinique médicale B
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
Mme	RENAUDET Jacqueline	Bactériologie
M.	REVOL Michel	Urologie
Mme	RINAUDO Marguerite	Chimie CERMAV
MM.	DE ROUGEMONT Jacques	Neuro-chirurgie
	SARRAZIN Roger	Clinique chirurgicale B
Mme	SEIGLE MURANDI Françoise	Botanique et crytogamie
MM.	SENGEL Philippe	Biologie animale
	SIBILLE Robert	Construction mécanique IUT 1
	SOUTIF Michel	Physique
	TANCHE Maurice	Physiologie
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire

MM. VAN CUTSEM Bernard  
 VAUQUOIS Berrard  
 VERAÏN Alice  
 VERAÏN André  
 VIGNAIS Pierre

Mathématiques appliquées  
 Mathématiques appliquées  
 Pharmacie galénique  
 Biophysique  
 Biochimie médicale

**PROFESSEURS DE 2ème CLASSE**

MM. ARNAUD Yves  
 AURIAULT Jean-Louis  
 BEGUIN Claude  
 BOITET Christian  
 BOUTHINON Michel  
 BRUGEL Lucien  
 BUISSON Roger  
 CASTAING Bernard  
 CHARDON Michel  
 CHEHIKIAN Alain  
 COHEN Henri  
 COHENADDAD Jean-Pierre  
 COLIN DE VERDIERE Yves  
 CONTE René  
 CYROT Michel  
 DEPASSEL Roger  
 DOUCE Roland  
 DUFRESNOY Alain  
 GASPARD François  
 GAUTRON René  
 GIDON Maurice  
 GIGNOUX Claude  
 GLENAT René  
 GOSSE Jean-Pierre  
 GROS Yves  
 GUITTON Jacques  
 HACQUES Gérard  
 HERBIN Jacky  
 HICTER Pierre  
 IDELMAN Simon  
 JOSELEAU Jean-Paul  
 JULLIEN Pierre  
 KERCKOVE Claude

Chimie IUT 1  
 Mécanique IUT 1  
 Chimie organique  
 Mathématiques appliquées  
 E.E.A. IUT 1  
 Energétique IUT 1  
 Physique IUT 1  
 Physique  
 Géographie  
 E.E.A. IUT 1  
 Mathématiques pures  
 Physique  
 Mathématiques pures  
 Physique IUT 1  
 Physique du solide  
 Mécanique des fluides  
 Physiologie végétale  
 Mathématiques pures  
 Physique  
 Chimie  
 Géologie  
 Sciences nucléaires  
 Chimie organique  
 E.E.A. IUT 1  
 Physique IUT 1  
 Chimie  
 Mathématiques appliquées  
 Géographie  
 Chimie  
 Physiologie animale  
 Biochimie  
 Mathématiques appliquées  
 Géologie

MM.	KRAKOWIACK Sacha	Mathématiques appliquées
	KUHN Gérard	Physique IUT 1
	KUPKA Yvon	Mathématiques pures
	LUNA Domingo	Mathématiques pures
	MACHE Régis	Physiologie végétale
	MARECHAL Jean	Mécanique
	MICHOULIER Jean	Physique IUT 1
Mme	MINIER Colette	Physique IUT 1
MM.	NEMOZ Alain	Thermodynamique
	NOUGARET Marcel	Automatique IUT 1
	OUDET Bruno	Mathématiques appliquées
	PEFFEN René	Métallurgie IUT 1
	PELMONT Jean	Biochimie
	PERRAUD Robert	Chimie IUT 1
	PERRIAUX Jean-Jacques	Géologie minéralogie
	PERRIN Claude	Sciences nucléaires
	PFISTER Jean-Claude	Physique du solide
	PIERRE Jean-Louis	Chimie organique
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	RICHARD Lucien	Biologie végétale
	ROBERT Gilles	Mathématiques pures
	ROBERT Jean-Bernard	Chimie physique
	ROSSI André	Physiologie végétale
	SAKAROVITCH Michel	Mathématiques appliquées
	SARROT REYNAUD Jean	Géologie
	SAXOD Raymond	Biologie animale
Mme	SOUTIF Jeanne	Physique
MM.	STUTZ Pierre	Mécanique
	VIALON Pierre	Géologie
	VIDAL Michel	Chimie organique
	VIVIAN Robert	Géographie

#### CHARGES D'ENSEIGNEMENT PHARMACIE

MM.	ROCHAS Jacques	Hygiène et hydrologie
	DEMENGE Pierre	Pharmacodynamie

#### PROFESSEURS SANS CHAIRE (médecine)

M.	BARGE Michel	Neuro-chirurgie
----	--------------	-----------------



MM.	BOST Michel	Pédiatrie
	BOUCHARLAT Jacques	Psychiatrie
	CHAMBAZ Edmond	Biochimie (hormonologie)
	CHAMPETIER Jean	Anatomie
	COLOMB Maurice	Biochimie
	COULOMB Max	Radiologie
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GROULADE Joseph	Biochimie A
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Gérontologie
	JALBERT Pierre	Histologie
	MAGNIN Robert	Hygiène
	PHELIP Xavier	Rhumatologie
	REYMOND Jean-Charles	Chirurgie générale
	STIEGLITZ Paul	Anesthésiologie
	VROUSOS Constantin	Radiothérapie

#### MAITRES DE CONFERENCES AGREGES (médecine)

MM.	BACHELOT Yvan	Endocrinologie
	BENABID Alim Louis	Médecine et chirurgie
	BERNARD Pierre	Gynécologie obstétrique
	CONTAMIN Charles	Chirurgie thoracique
	CORDONNIER Daniel	Néphrologie
	CROUZET Guy	Radiologie
	DEBRU Jean-Luc	Médecine interne
	DYON Jean-François	Chirurgie infantile
	FAURE Claude	Anatomie et organogénèse
	FAURE Gilbert	Urologie
	FLOYRAC Roger	Biophysique
	FOURNET Jacques	Hépatogastro-entérologie
	GAUTIER Robert	Chirurgie générale
	GIRARDET Pierre	Anesthésiologie
	GUIDICELLI Henri	Chirurgie générale
	GUIGNIER Michel	Thérapeutique (réanimation)
	JUNIEN-LAVILLAULOY Claude	Clinique O.R.L.
	KOLODIE Lucien	Hématologie biologique
	MALLION Jean-Michel	Médecine du travail
	MASSOT Christian	Médecine interne
	MOUILLON Michel	Ophtalmologie

**MM. PARAMELLE Bernard**  
**RACINET Claude**  
**RAMBAUD Pierre**  
**RAPHAEL Bernard**  
**SCHAEFER René**  
**SEIGNEURIN Jean-Marie**  
**SOTTO Jean-Jacques**  
**STOEBNER Pierre**

**Pneumologie**  
**Gynécologie-Obstétrique**  
**Pédiatrie**  
**Stomatologie**  
**Cancérologie**  
**Bactériologie-virologie**  
**Hématologie**  
**Anatomie-pathologique**



Je tiens à remercier,

Monsieur P.JULLIEN, professeur à l'Université Scientifique et Médicale de Grenoble, d'avoir bien voulu présider ce jury,

Messieurs P.C.SCHOLL et J.DELLADORA de l'Université Scientifique et Médicale de Grenoble, de me faire l'honneur d'examiner et de juger ce travail.

Jacques CALMET qui, malgré ses nombreuses activités, a accepté de diriger ce travail et d'en être le rapporteur. Je le remercie de m'avoir aidé dans les démarches administratives et de m'avoir fait bénéficier de son appui sans compter le temps et la gentillesse, en m'intégrant dans l'équipe du groupe Calcul Formel du laboratoire Informatique de l'IMAG.

Tout particulièrement Marc BERGMAN, car sans lui rien n'aurait été possible. Il a su grâce à sa sympathie et son dévouement abolir l'isolement psychologique dû aux 10 000 kms séparant le Laboratoire Informatique de La Réunion et la métropole. Je lui suis profondément reconnaissant d'avoir suivi ce travail en me guidant au cours de longues conversations téléphoniques, vers la réalisation définitive de ce travail. Je le remercie de m'avoir donné confiance en moi-même et initié à la recherche, j'espère que nous pourrons continuer dans cette voie.

Pierre GIGORD, responsable de l'IREM de La Réunion, pour les contacts qu'il a pris afin de faciliter mon travail, les encouragements prodigués afin de démarrer cette thèse. Je le remercie pour la possibilité qu'il m'a donnée d'utiliser les moyens de l'IREM et l'intérêt pédagogique qu'il a manifesté pour ce travail.

Mon épouse Dominique pour sa patience affectueuse, son aide morale et matérielle pour la correction et la rédaction de ce document, et pour les sacrifices financiers consentis pour l'achat du micro-ordinateur qui a servi à l'implantation du système SYPAC.

Madame S.ROCHE et monsieur D.IGLESIAS, pour la frappe d'une partie et le tirage rapides de ce document.

Enfin, le lecteur pour sa patience qui sera soumise à rude épreuve, en passant au fil des pages d'une composition dactylographiée à une impression par un éditeur de texte et vice versa.

\*\*\*\*\*



## PLAN DU DOCUMENT

-----

Les noms des paragraphes apparaissant dans les chapitres sont ceux qui forment le titre répété sur chaque feuille en haut de page, les numeros de pages se réfèrent aux numeros marqués en bas de page.

INTRODUCTION SYSTEME SYPAC .....p.1

### CHAPITRE I: LE LANGAGE DE SURFACE PALDES

DESCRIPTION DU LANGAGE PALDES MINIMAL .....	p.13
1- Les données en PALDES .....	p.13
2- Les variables en PALDES .....	p.19
3- Les algorithmes en PALDES .....	p.22
4- Les instructions exécutables en PALDES .....	p.23
5- Grammaire LL(1) de PALDES .....	p.31

### LE TRAVAIL COMME ETUDE DE FAISABILITE

-----

### CHAPITRE II:

COMPILATEUR PALDES .....	p.39
1- Fonctionnement général du compilateur .....	p.40
2- Organisation et modules du compilateur .....	p.43
3- Types de données internes au compilateur .....	p.43

### CHAPITRE III:

MODULES DU COMPILATEUR PALDES .....	p.47
1- Schéma de hiérarchie .....	p.48
2- Module d'analyse lexicographique .....	p.49
3- Module des initialisations .....	p.52
4- Module de gestion de la table des symboles .....	p.54
5- Module de reconnaissance .....	p.64
6- Module de debugging .....	p.65
7- Module des déclarations .....	p.65
8- Module de génération de code .....	p.66

### CHAPITRE IV:

MEMOIRE DYNAMIQUE DE LISTE DANS SYPAC .....	p.72
---	------

1- Représentation des listes PALDES .....	p.73
2- Examen des différentes procédures Pascal gérant la mémoire dynamique .....	p.75
3- Restitution et désallocation des listes dans SYPAC .....	p.76
4- Allocation mémoire .....	p.79
5- Interpretation des listes .....	p.80
6- Exemple d'interprétation .....	p.81
7- Etat et transformation de la mémoire dans SYPAC en PALDES .....	p.83
8- Schema d'occupation de la mémoire lors de l'exécution d'un programme PALDES .....	p.86
9- Procédures internes écrites en Pascal .....	p.89
10- Procédures internes écrites en PALDES .....	p.92

CHAPITRE V:

MODULE DE PRECISION INFINIE DE SYPAC .....	p.95
1- Les entiers en précision infinie .....	p.96
2- Les algorithmes classiques .....	p.98
3- Système de calcul en précision infinie en PALDES .....	p.104

CHAPITRE VI:

MODULE POLYNOMIAL DE SYPAC .....	p.116
1- Les polynomes de $Z[X]$ en PALDES .....	p.117
2- Les algorithmes composant le module .....	p.117
3- Système de calcul polynômial en PALDES .....	p.121

BIBLIOGRAPHIE:

* Grammaire LL(1) et compilation .....	p.128
* Le langage Pascal .....	p.129
* La version UCSD .....	p.130
* ALDES et systèmes SAC-1 et SAC-2 .....	p.131
* Autres systèmes .....	p.131

ANNEXES .....	p.134
---------------	-------

INTRODUCTION AU CALCUL FORMEL  
ET  
AU SYSTEME SYPAC

... Dans cette introduction figurerons quelques generalites sur le Calcul Formel afin de placer ce travail dans son contexte naturel.

DEFINITION ET USAGE DU CALCUL FORMEL:

- 1- DEFINITION COMME DISCIPLINE CARREFOUR .....P.1
- 2- LES PROBLEMES QUI SE POSENT EN CALCUL FORMEL .....P.1
- 3- LES LANGAGES HOTES DU CALCUL FORMEL .....P.3
- 4- LES MATERIELS UTILISES .....P.4

PRESENTATION ET POSITION DU TRAVAIL:

- L'EXPERIENCE DE F. TEER .....P.6
- LES OBJECTIFS DE CE TRAVAIL .....P.7
- ALDES ET LE SYSTEME SAC-1 .....P.8
- ELEMENTS DE SYPAC .....P.10
- PORTABILTE ET INTERET DE SYPAC .....P.10

\*\*\*\*\*





## 1-DEFINITION COMME DISCIPLINE CARREFOUR:

-----

On emploie les mots CALCUL FORMEL au sens des manipulations symboliques et algébriques sur ordinateur. C'est un domaine en expansion dans la mesure où depuis environ trente ans c'est une discipline carrefour entre les trois disciplines suivantes:

1°) La PHYSIQUE: elle a posé et pose encore de nombreux problèmes qui ont permis d'élaborer des techniques et des concepts généraux pour donner naissance à des systèmes de Calcul Formel d'intérêt général.

2°) Les MATHÉMATIQUES: elles ont apporté avec le développement de l'Algorithmique non-numérique, une contribution fondamentale à la manipulation d'objets symboliques sur ordinateur.

3°) L'INFORMATIQUE: de par l'utilisation que l'on fait du Calcul Formel, l'Informatique joue un rôle qui va croissant. Avec d'une part les langages de programmation, et d'autre part la représentation des objets. Les développements couvrent tous les aspects de la discipline de la théorie à la technique.

## 2-LES PROBLEMES QUI SE POSENT EN CALCUL FORMEL:

-----

Nous présentons ici quelques aspects, parmi les plus importants de cette problématique.

a) Au niveau des Mathématiques deux branches se sont développées parallèlement, ce sont:

\* Les méthodes heuristiques, comme dans la simplification de formule et dans l'intégration symbolique telles que l'on peut en trouver dans le système SIN de J.MOSES en 1967.

\* Les méthodes algorithmiques exactes avec le développement des calculs sur les polynômes à une ou plusieurs variables.

Ces deux méthodologies apparaissent dans la pratique, comme complémentaires, des heuristiques donnant de très bons résultats soit lorsqu'un algorithme n'existe pas, soit lorsqu'il existe et qu'il est peu efficace, soit lorsqu'il est en situation d'échec. Signalons à ce sujet des systèmes

algorithmiques comme le système PM de G. COLLINS, devenu SAC-1 en 1969 et proposant une collection importante d'algorithmes sur les polynômes à une ou plusieurs variables.

\* Certains théorèmes ont permis d'ouvrir et de faire progresser de nouvelles voies, citons pour mémoire les algorithmes de calcul du PGCD et l'important algorithme de BERLEKAMP (1970) sur la factorisation des polynômes en arithmétique modulaire. Cet algorithme a permis de faire progresser le calcul polynômial et l'intégration symbolique. Cette dernière technique devant elle-même beaucoup à l'algorithme de RISCH (1970) sur l'intégration en série finie dans une extension algébrique différentielle fixée. L'implantation des derniers raffinements de cet algorithme restant encore en cours.

b) Problématique au niveau de la représentation des objets:

Dans un calcul symbolique, il n'est généralement pas possible de prévoir la taille d'un calcul, ce qui n'est pas le cas dans un problème d'algorithmique numérique.

c) Au niveau de la communication avec le langage de surface:

L'utilisation d'un système de Calcul Formel d'intérêt général, impose une communication avec l'utilisateur de type conversationnelle.

Le langage de surface outre ses facilités dans les commandes, devra être proche d'un langage de programmation connu par les scientifiques, qui ne sont pas toujours disposés à apprendre un nouveau langage.

Les systèmes de Calcul Formel doivent satisfaire à certaines exigences:

d) Posséder les calculs en précision infinie, cette arithmétique bien que coûteuse, est utile à l'intérieur de certains calculs formels.

e) Etre extensibles par adjonction de nouvelles expressions, soit en définissant de nouveaux types d'objets avant d'implanter les algorithmes les manipulant, soit en programmant en langage hôte. Car d'une façon générale un système de Calcul Formel est implanté au moyen d'un langage dans lequel est écrit le traducteur du langage de surface.

Terminons ce paragraphe en faisant remarquer que les problèmes d'implantation d'algorithmes déjà existants et parfaitement connus, ne sont pas tous triviaux. Citons certains algorithmes de multiplication rapide de polynômes qui sont inefficaces une fois implantés alors que mathématiquement ils sont les plus performants. Enfin aucun ouvrage regroupant tous les aspects du Calcul Formel n'a été à ce jour publié.

### 3-LES LANGAGES HOTES DU CALCUL FORMEL:

-----

Etant donné son caractère combinatoire, et du fait de la représentation d'un grand nombre d'objets sous forme de listes, le Calcul Formel est un gros utilisateur du langage LISP et de ses dérivés.

\* LISP comme langage applicatif est un outil puissant dans le traitement des listes c'est le langage le plus utilisé.

Mais pour un système de Calcul Formel la portabilité dépend pour une grande part du langage hôte et du compilateur de ce langage. Or LISP n'étant pas normalisé, il existe de nombreuses versions différentes de l'interpréteur LISP, et le manque de compatibilité entre ces versions nuit à la portabilité de tels systèmes.

\* Un autre langage hôte plus portable et plus efficace au niveau du compilateur est le FORTRAN, qui est après LISP le deuxième langage le plus utilisé. Dans les systèmes supportés par FORTRAN on commence d'abord par construire un module de manipulations de listes, citons les systèmes ALTRAN, SAC-1 et SAC-2 ayant FORTRAN comme langage hôte.

\* Certains systèmes spécialisés sont écrits, pour des raisons d'efficacité, en langage machine comme le système SCOONSCHIP utilisé en Physique des Hautes Energies. Cette particularité est un vice rédhibitoire pour leur diffusion.

\* Pour mémoire PL/1 a servi de support au système FORMAC qui à ce jour n'est plus utilisé.

Pour finir notons la remarque suivante:

les ressources nécessaires en mémoire, sont directement liées à la taille des expressions engendrées par les calculs intermédiaires, les nécessités propres à l'algorithme utilisé interviennent aussi, de sorte qu'afin de minimiser l'occupation mémoire, dans tous les cas on trouve dans un système de Calcul Formel, une GESTION DYNAMIQUE de la mémoire avec un algorithme de récupération mémoire généralement par Garbage Collection.

#### 4-LES MATERIELS UTILISES EN CALCUL FORMEL:

-----

Il existe parmi la soixantaine de systèmes de Calcul Formel utilisables trois principaux systèmes accessibles à un grand nombre de personnes, il s'agit de:

MACSYMA développé au MIT par J.MOSES depuis 1969, disponible uniquement aux abonnés américains du réseau ARPA. Sa diffusion sur d'autres matériels est annoncée (le centre de calcul de Grenoble en dispose), le langage hôte est LISP, il nécessite 500 Ko de mémoire.

REDUCE développé à l'université de l'UTAH par A.HEARN depuis 1969, est le plus distribué (plus de 500 centres scientifiques en dispose). C'est un système interactif. Le langage hôte est LISP, il nécessite pour la version minimale environ 200 Ko de mémoire.

SAC-2/ALDES développé en 1980 conjointement par G.COLLINS et R.LOSS, c'est une nouvelle version du système SAC-1, non interactif de langage hôte FORTRAN, il est distribué sur tout matériel possédant un compilateur FORTRAN IV et au moins 100 Ko de mémoire. C'est un système dirigé vers l'utilisation mathématique d'algorithmes polynômiaux.

Malgré tous ces systèmes on continue tous les ans à écrire des systèmes, comme SMP (1981) par WOLFRAM développé en langage-C. On peut remarquer que malgré leur modularité, ces systèmes nécessitent des partitions mémoires relativement importantes ce qui limite leur diffusion.

Sur mini-ordinateurs des développement se font:

VAXYMA une version de MACSYMA pour VAX en grosse configuration.

Une machine REDUCE est développée sur PDP 11.

Une machine ALDES est en cours de réalisation sur LSI 11/23.

Depuis 1979 avec l'apparition des micro-processeurs deux petits systèmes de Calcul Formel aux capacités bien sur incomparables aux précédents, ont été développés il s'agit de:

## >>> Introduction système SYPAC <<<<

MUMATH développé en 1979 par R.STOUTMEYER et RICH, il est conçu pour le micro-processeur INTEL 8080 à l'aide d'un LISP interprété. Il est disponible sous CP/M et TRS/DOS sur tous les micro-ordinateurs à base de micro-processeurs INTEL 8080 et Z 80, il nécessite de 32 Ko à 64 Ko selon le nombre de modules chargés. C'est un système à vocation pédagogique

NLARGE développé en 1980 par J.FITCH conçu pour le micro-processeur Z 80 sur la base d'un LISP compilé sous CP/M, il utilise de 32 Ko à 64 Ko et permet de faire des opérations de base sur des polynômes à plusieurs variables à coefficients rationnels  $p/q$  avec  $p$  et  $q$  inférieurs à 4096 en valeur absolue. Un module d'entiers infinis a été développé mais non encore intégré à NLARGE.

Il semble que la diffusion importante des micro-ordinateurs et des mini-ordinateurs provoquera une grande expansion de petits logiciels de Calcul Formel, soit pour des utilisations pédagogiques, soit pour des petits systèmes experts.

Pour résumer les possibilités du Calcul Formel voici quelques phrases clef:

Le Calcul Formel invente, implante et utilise des algorithmes algébriques.

En Calcul Formel l'utilisation des algorithmes algébriques permet de faire des représentations sans perte de précision ni de signification.

Le Calcul Formel fait dans sa partie algébrique tout ce qui est trop numérique pour les algébristes et tout ce qui est trop algébrique pour les numériciens.

A la fin d'un calcul symbolique on aboutit toujours à un interface entre le Calcul Formel et le calcul numérique.

PRESENTATION ET POSITION DU TRAVAIL PAR RAPPORT AU CALCUL FORMEL:

F. TEER a proposé en 1977 une extension du Pascal appelée FORMULAPASCAL, cette extension est obtenue par adjonction d'un type de donnée permettant de représenter tout à la fois, un entier infini, un rationnel, un polynôme à une ou plusieurs variables, à coefficients entiers infinis ou rationnels.

Cette extension a été conçue grâce aux possibilités de description de structure qu'offre le Pascal, la facilité et le style de programmation qu'il autorise, et sa gestion dynamique de la mémoire. F. TEER a développé cette extension comme interface aux modules du système SAC-1, en appelant à l'intérieur de son programme engendré, des procédures SUBROUTINE FORTRAN.

\* L'implantation est faite sur un gros matériel, l'auteur disposait déjà d'un système de Calcul Formel écrit en ALGOL 68 et d'un autre système écrit en ABC ALGOL, ses mesures de performances sont en défaveur du FORMULAPASCAL. Il note même des temps deux fois plus long que sur SAC-1 seul. TEER explique ceci par le fait que le FORMULAPASCAL doit, avant d'appeler une routine SAC-1, faire des vérifications complètes de compatibilité des représentations des polynômes.

\* Enfin l'auteur fait remarquer qu'il y a ambiguïté entre l'interprétation des rationnels et celle des polynômes, de telle sorte que certaines fonctions de manipulations de rationnels peuvent considérer par ERREUR un polynôme comme un rationnel et le manipuler comme tel. Le FORMULAPASCAL a été abandonné et reste la seule tentative d'utiliser Pascal en Calcul Formel à ce jour.

Il est intéressant de noter deux remarques sur l'extension de F. TEER ceci afin d'éclairer les options qui sont prises plus tard dans SYPAC:

a) La récupération de mémoire nécessaire comme nous l'avons dit dans les pages précédentes, est faite en FORMULAPASCAL par la méthode du compteur de référence, méthode faisant augmenter la taille de la cellule de base, et oblige à une mise à jour permanente de ce compteur dès que l'on utilise la cellule correspondante.

b) La restitution des objets à la pile de mémoire du Pascal est confiée au Pascal à travers la procédure

>>>> introduction système SYPAC <<<<

DISPOSE().

Une question restait ouverte : pouvait-on, vu le développement du langage Pascal sur les micro-ordinateurs, utiliser ce langage pour le Calcul Formel sur de petits matériels.

Dès lors les objectifs de ce travail ont été fixés aux trois points suivants:

1°) Développer un petit système de Calcul Formel sur micro-ordinateur, pour en faire un laboratoire d'essai et de test sur les possibilités du Pascal en Calcul Formel sur micro. Ce système a été appelé SYPAC.

2°) Définir SYPAC de façon modulaire et portable sur un grand nombre de micro-ordinateurs.

3°) Les contraintes du système sont-elles compatibles avec celles d'un développement ultérieur en système à vocation pédagogique.

Le modèle proposé pour le langage de surface était ALDES, car c'est le meilleur langage de description d'algorithmes algébriques, le langage de surface de SYPAC se dénomme PALDES pour Pascal-ALDES, il reprend l'essentiel d'ALDES en utilisant la structuration du Pascal, il est décrit dans ce document plus loin.

Voici afin de situer PALDES un très bref rappel sur ALDES et le système qu'il supporte:

ALDES est le langage de surface du système SAC-2, qui est une implantation en ALDES sous compilateur FORTRAN, des modules de SAC-1:

- ALDES est une implantation de la METHODE DE DESCRIPTION DES ALGORITHMES donnée par D.E.KNUTH.

- ALDES est un langage de type ALGOL dont la grammaire est donnée sous forme de diagrammes syntaxiques par R.LOOS.

- Un programme ALDES est compilé par un compilateur écrit en FORTRAN IV et engendrant du FORTRAN IV.

- ALDES ne manipule que des ENTIERS MACHINES et des LISTES, ceux-ci dans un "TAS", les pointeurs de listes sont obtenus par une partition sur les entiers.



NOMBRES ALGEBRIQUES

INTERVALLE ARITHMETIQUE

ARITHMETIQUE RATIONNELLE

ZERO REELS DES POLYNOMES

FACTORISATION

RESULTANTS

PGCD DE POLYNOMES DE PLUSIEURES VARIABLES

POLYNOMES A PLUSIEURES VARIABLES A COEFF.  
ENTIERS INFINIS

POLYNOMES A PLUSIEURES VARIABLES A COEFF  
DANS UN CORPS

ARITHMETIQUE MODULAIRE

ARITHMETIQUE EN PRECISION INFINIE

GESTION DES LISTES DE BASE

MODULES DE SAC 1

>>> introduction système SYFAC <<<<

APERÇU SUR LES DIFFÉRENCES AVEC ALDES:

\*\*\*\*\*

Les objets de FALDES:

-----

Comme la version de Loos, FALDES supporte les entiers représentables par la machine et les listes.

FALDES supporte en plus les types prédéfinis suivant:

- caractères --> CARAC.
- logique --> LOGIQUE.
- chaîne de longueur < 80 caractères --> CHAINE.

Ces types d'objets n'apportent rien de nouveau au système ALDES de Loos, qui peut manipuler des codages de caractères ou de Booleens. Toutefois il est plus aisé pour l'utilisateur de se servir d'une variable typée que d'un codage. Ces facilités étant dues au fait que le Pascal traduit, par le préprocesseur, supporte déjà ce genre de type.

Déclarations en FALDES:

-----

En FALDES, toute variable utilisée dans un algorithme, doit avoir été déclarée soit, comme étant une variable globale, soit comme une variable locale à l'algorithme. Ceci a les deux avantages connus dans les langages déclaratifs:

- la vérification statique est plus aisée pour l'interpréteur,
- l'utilisateur connaît la portée de ses identificateurs en lisant son programme.

Toute variable doit être déclarée avec son type, sinon le type par défaut est: entier représentable en machine.

Les numéros de paragraphes (étiquettes) utilisés doivent être comme en Pascal, déclarés avant d'être référencés dans un ALLERA. Les numéros non assignés par un ALLERA pourront ne pas être déclarés.

Les paramètres d'un algorithme suivent les mêmes principes qu'en ALDES, avec l'obligation à la définition de l'algorithme de définir leur type.

Les caractéristiques des paramètres sont les mêmes qu'en ALDES.

Les listes seront décrites plus précisément plus loin, leur manipulation étant sensiblement la même qu'en ALDES.

Elles sont construites dans l'optique de la représentation des polynômes. Par souci de gain de place sur un petit système, le zéro n'est pas représenté.

## >>>> introduction système SYPAC <<<<

les algorithmes de gestion des listes du système SYPAC sont totalement différents de ceux d'ALDES. La récupération de place se faisant en ALDES par Garbage Collector dans un grand tableau.

Dans SYPAC, en PALDES par l'intermédiaire d'un algorithme de gestion de pile, on s'est servi de la mémoire dynamique du Pascal afin d'avoir sur un petit système le maximum de mémoire disponible.

### ELEMENTS PHYSIQUES DE SYPAC:

-----

1) Préprocesseur PALDES: 3000 lignes de Pascal.

2) PSL (bibliothèque de base) environ 800 lignes de Pascal et 60 lignes de PALDES.

ESL1 (bibliothèque pour entiers arbitraires) environ 400 lignes de PALDES.

ESL2 (bibliothèque pour polynômes) environ 150 lignes de Pascal minimal et 300 lignes de PALDES.

Le coefficient relatif entre nombre de lignes PALDES et nombre de lignes Pascal minimal généré est 1.10 environ.

pour l'occupation mémoire effective tout dépendra du système Pascal de la machine hôte. Afin de fixer les idées, le système SYPAC a été écrit en Pascal UCSD sur Apple II (le code objet est alors du P-code interprété), le préprocesseur PALDES occupe moins de 30 K0 sur diskette, sur 36 K0 de mémoire disponible PSL+ESL1+ESL2 résidents occupent à peu près 26 K0.

### PORTABILITE DE SYPAC:

-----

SYPAC est un système relativement portable d'un matériel à un autre, actuellement il est portable immédiatement sur tout système Pascal UCSD, sous forme d'UNIT mises dans une library.

Une deuxième version est en cours d'écriture pour être utilisée sur un autre système Pascal.

\* Le manque de portabilité se situe au niveau de la segmentation puisque le Pascal ISO ne supporte pas cette éventualité, dans l'attente d'une norme internationale incluant cette possibilité, l'UCSD permettant de faire cette opération, et de plus conçu pour fonctionner sur des micro-ordinateurs, sert de base à SYPAC.

\* Sans vouloir faire de SYPAC un système aussi performant que MUMATH disponible sur tout matériel possédant un microprocesseur Intel 8080 ou Zilog Z80 sous système CPM ou DOS, la disponibilité croissante de systèmes variés constitués de microprocesseurs différents, supportant Pascal UCSD, fait que SYPAC pourrait être un outil à

vocation pédagogique.

En effet, l'implantation des polynômes dans  $Q[X]$  à coefficients arbitraires, et le calcul vectoriel dans un espace vectoriel sur  $Q$  de dimension finie, est facilement réalisable.

\* Les faiblesses actuelles des temps d'exécution sur micro-ordinateurs "8 bits" pour des calculs importants (nombres à plus de 30 chiffres...), ne touchent pas les exemples pédagogiques courants utilisant des nombres "raisonnables".

\* outre sa disponibilité sur un grand nombre de matériels différents, l'intérêt de SYPAC réside, dans cette optique, dans le fait que l'enseignant disposerait d'un langage (PALDES) structuré d'apprentissage aisé, lui permettant d'écrire ses algorithmes et de les expérimenter.

\* Enfin la sécurité de ne pas avoir à être limité par la taille des éléments à manipuler, devrait soulager le pédagogue des contrôles de validité des calculs et lui apporter un champ plus vaste d'expérimentation.

\* L'évolution ultérieure de SYPAC sera d'en établir une version plus conversationnelle et finie, puis d'en faire l'expérimentation avec des professeurs de Mathématiques et de Physique de l'Enseignement secondaire.

\*\*\*\*\*

## CHAPITRE I

Dans cette partie on décrit la syntaxe et la sémantique  
du langage PALDES

=====

>>>> description PALDES minimal <<<<

\*\*\* DESCRIPTION SYNTAXIQUE ET SEMANTIQUE DU LANGAGE PALDES \*\*\*

Un programme PALDES consiste en un algorithme principal, avec ou non des sous-algorithmes.

Le jeu des caractères est celui du Pascal:

les chiffres:

C = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

les lettres:

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

les caractères spéciaux:

K = { !, ", #, \$, %, &, ', (, ), \*, -, +, /, =, <, >, (, ), ^, l, l, ..., }

les mots réservés:

R = { allera, alors, assertion, carac, chaine, entrer, et, étiquette, faire, faux, global, imprimer, intrinseq, jusque, liste, local, logique, non, ou, permanent, polynôme, pour, pragma, reel, répeter, retour, selon, si, sinon, sortir, stop, tableau, tantque, vrai }

Le vocabulaire terminal Vt est donc:

(en notant + le symbole de l'union ensembliste)

Vt = C + L + K + R

1) Les données en PALDES:

Les seuls objets internes au système PALDES sont :

- Les nombres entiers représentables en machine (dans l'implantation inférieurs en valeur absolue à 32768 ).
- Les caractères de C+L+K.
- Les booléens .
- Les chaînes de caractères.
- Les listes d'entiers machine.

>>> description PALDES minimal <<<<

- Les polynômes à une variable à coefficients entiers.
- Les entiers en précision infinie.
- Les polynômes à une variable à coefficients liste.

a) Les entiers machine:  
-----

Ce sont les entiers que le système Pascal supportant PALDES, est capable de représenter. Sur le micro-ordinateur de l'implantation (Apple II+) ce sont les nombres  $x$  tels que:

$$-32768 < x < 32767$$

Ces nombres peuvent être entrés ou sortis par les instructions de base du langage ENTRER, IMPRIMER.

Le système PALDES comporte des opérations et des fonctions sur ces nombres:

- Les opérateurs classiques: +, -, \*
- La division entière: / (quotient dans la division euclidienne).
- Certaines fonctions:
  - t:= MIN(x,y) --> t est le min de x et de y.
  - t:= MAX(x,y) --> t est le max de x et de y.
  - t:= SGNN(x) --> t vaut -1 si x<0, 0 si x=0, +1 si x>0.
  - t:= PAIR(x) --> t est VRAI si x est pair FAUX si x impair.
  - t:= RESTE(x,y) --> t est le reste de la division euclidienne de x par y.
  - t:= PGCDN(x,y) --> t est le pgcd de x et de y.
  - t:= PPCMN(x,y) --> t est le ppcm de x et de y.
  - t:= PUISS(x,y) --> t vaut x à la puissance y.

b) les caractères:  
-----

Tous les caractères du Pascal sont accessibles dans des variables simples de type CARAC.

Les seules opérations que l'on peut faire sur ces caractères sont des E/S, des comparaisons lexicographiques, des affectations.

c) les booléens:  
-----

Il existe des variables de type LOGIQUE, qui peuvent être affectées des valeurs VRAI ou FAUX, puis utilisées dans des expressions conditionnelles.

Il est aussi possible de se servir de ces variables comme

test de véracité d'une proposition, comme dans l'affectation suivante:

t:= ( x < B ) ou ( x+y > z );

Ici à l'exécution de cette instruction, selon les valeurs respectives de x,y,z la variable t prendra la valeur FAUX ou VRAI.

- opérateurs sur les booléens:

NON , OU , ET

Ce sont les opérateurs classiques de l'algèbre de Boole et du calcul propositionnel.

- une fonction booléenne: (x et y sont de type LOGIQUE)

t:= OUX(x,y) --> t est le ou exclusif de x et de y.

d) les chaînes de caractères:

-----  
Elles ne sont accessibles qu'en affectation sur des variables de chaîne (longueur < 80), ou avec les instructions ENTRER et IMPRIMER.

e) les listes d'entiers machines:

-----  
La conception du système de liste est détaillée plus loin.

Il est nécessaire de savoir que les listes PALDES sont orientées vers la manipulation polynômiale, et donc que la cellule de base d'une liste comporte deux informations utilisables par le programmeur.

Chacune de ces deux informations correspond au coefficient d'un monôme et à sa puissance.

La puissance est un élément de N, donc essentiellement positif.

Le nombre de cellules d'une liste est donc limité par le nombre maximal représentable en machine.

Cette contrainte apparente n'est pas gênante, car la mémoire centrale n'offre généralement pas autant de cellules de liste possibles:

Dans l'implantation la mémoire centrale maximale (sur 16 K-octets) offre au plus 7900 éléments pour une liste, alors que le nombre maximal représentable est 32767.

Les listes sont ordonnées par le système, soit par ordre croissant sur l'élément puissance, soit par ordre décroissant sur ce même élément.



>>>> description PALDES minimal <<<<

Il est tout à fait possible de se servir des listes d'entiers machine pour un usage autre que polynômial, on dispose alors du rang de l'entier dans la liste et l'on peut donc faire des tris, des concaténations de listes.

LES LISTES PALDES NE COMPORTENT PAS DE ZEROS.

En fait ceci a été choisi pour optimiser la place en mémoire centrale, au prix d'une légère augmentation de la complexité des algorithmes de traitement de liste.

Tout nombre nul n'est pas rentré, sa présence est indiquée par le fait qu'il manque des éléments dans la suite des rangs consécutifs.

Les éléments manquant sont interprétés par le système comme étant des nombres nuls:

soit la liste : (23, -87, 345, 0, 5, 0, 0, 56)

La liste implantée comprendra 5 cellules:

rang:	0	1	2	4	7
élément:	23	-87	345	5	56

Les éléments de rang 3, 5 et 6 manquent, ils sont donc nuls.

OPERATIONS SUR LES LISTES:  
=====

C'est le système de gestion de liste qui fait les opérations d'insertion, d'effacement, d'ordonnancement (cf. en haut).

Le programmeur peut utiliser des algorithmes systèmes de traitement de liste:

- 1) déclaration: AFFECT(L=liste;R=liste)  
utilisation: AFFECT(L;R);

Sert à transférer le contenu total de la liste L dans la liste R. A la fin de l'opération R est identique à L, L est conservée.

- 2) déclaration: CHER(L=liste;X:Y)  
utilisation: CHER(L,X;Y);

Sert à trouver dans la liste, L l'entier qui a pour rang x, le résultat se trouve dans y.

- 3) déclaration: COPIE(L=liste;R=liste)  
utilisation: COPIE(l;r);

Sert à fusionner la liste L sur la liste R. Il y a

>>>> description PALDES minimal <<<<

insertion des éléments non nuls de L, qui remplacent intégralement ceux de R de même rang.

A la fin de l'opération L est conservée, R est une copie de L comportant tous les éléments de R qui avaient un correspondant nul dans L:

rang:	0	1	2	3	4	5
élément L:	-5	0	45	-98	0	9
élément R:	12	-4	68	0	0	100
-----						
résultat :	-5	-4	45	-98	0	9

Ceci revient à recopier dans R tous les éléments de L qui figurent explicitement dans L.

4) déclaration: LONG(L=liste;N)  
utilisation: LONG(L;N);

Sert à communiquer la longueur en nombre d'éléments non nuls d'une liste:

Dans l'exemple précédent la liste L a pour longueur 4, la liste R avant COPIE a pour longueur 4, après copie de L sur R, R a pour longueur 5.

5) déclaration: EGAL(L1,L2=liste;T=logique)  
utilisation: EGAL(L1,L2;T);

Sert à tester l'égalité de deux listes, élément par élément et rang par rang.

La liste L=(8,0,9,-6,7) et la liste R=(8,9,0,-6,7) bien que de même longueur, contenant en notation compactée les mêmes éléments non nuls, ne sont pas égales (T sera mis à FAUX), car l'élément de rang 1 n'est pas le même.

6) déclaration: DESALL(;L=liste)  
utilisation: DESALL(;L);

Sert à supprimer entièrement une liste, et à la rendre inaccessible par la suite (désallocation programmée et restitution de la place libérée).

7) déclaration: EFFEL(X;L=liste)  
utilisation: EFFEL(X;L);

Sert à supprimer dans la liste L, l'élément dont le rang est X. (restitution de la place libérée)

8) déclaration: EFFLS(;L=liste)  
utilisation: EFFLS(;L);

Sert à effacer tout le contenu de la liste L. La

>>> description PALDES minimal <<<<

liste L se retrouve de longueur nulle, comme après son initialisation, on dit que la liste est vide. (restitution de la place libérée)

9) déclaration: ETAT(L=liste;X)  
utilisation: ETAT(L;X);

Sert à connaître l'état de liste maintenu par le système.

SI x=0 ALORS la liste est vide .  
SI x=1 ALORS la liste contient au moins un élément non nul.  
SI x=2 ALORS la liste n'est plus accessible.

10) déclaration: INV(L=liste;R=liste)  
utilisation: INV(L;R);

Sert à inverser une liste, cette inversion fait passer de l'ordre des rangs décroissants à celui des rangs croissants, la liste R contient exactement et seulement tous les éléments de L dans l'ordre inverse.

Soit L= (34,0,0,0,0,2,-5,9,78)

Après INV(L;R) on aura

R= (78,9,-5,2,0,0,0,0,34)

11) déclaration: MET(X,Y;L=liste)  
utilisation: MET(X,Y;L);

Sert à "mettre" l'élément Y nul ou non, de rang X, dans la liste L.

Il s'agit de l'insertion d'un élément s'il n'était pas présent, sinon de l'écrasement de l'ancienne valeur.

Soit la liste L= (12,5,0,-6,98) ordonnée par rang décroissant (ordre standard).

On a donc rang(12)=4  
rang(5) =3  
rang(-6)=1  
rang(98)=0

Si l'on fait MET(3,1232;L); la liste L deviendra:

L= (12,1232,0,-6,98)

Si l'on fait MET(2,100;L); la liste L devient:

L= (12,1232,100,-6,98)

12) déclaration: PREM(L=liste;X,Y)  
utilisation: PREM(L;X,Y);

>>> description PALDES minimal <<<<

Sert à donner le premier élément non nul de la liste, de rang le plus élevé si l'ordre est standard, le moins élevé si l'ordre est inverse.

En pratique, si l'on lit la liste de gauche à droite, cela revient à mettre l'élément le plus à gauche dans Y et son rang dans X.

Soit L = (23,0,0,0,4,-6)

Après PREM(L;X,Y) X contiendra 5 (rang de 23), et Y la valeur 23.

13) déclaration: REDUC(:L=liste)  
utilisation: REDUC(;L);

Sert à supprimer l'élément "le plus à gauche" de la liste L.

Soit L = (23,0,0,0,4,-6)

Après REDUC(;L) L devient:

L = (4,-6)

23 ayant été supprimé, le premier élément non nul étant 4 de rang 1, la liste L qui était au départ de longueur 6, devient une liste de longueur 2.

Une partie de ces algorithmes a été écrite en Pascal, l'autre en PALDES.

Nous verrons plus loin les trois autres classes de données du langage. Car tous les algorithmes les manipulant ont été écrits en PALDES.

Le type liste en PALDES, est essentiel, car il sert de base aux types du système SYPAC obtenu par extension du système de base.

## 2) LES VARIABLES EN PALDES: =====

Comme en Pascal, toute variable utilisée dans un programme PALDES, doit avoir été préalablement déclarée.

Les variables sont représentées dans le programme par des identificateurs dont la longueur dépend de l'implantation.

Les identificateurs sont les mêmes que dans les langages FORTRAN, Pascal, ALGOL, ...etc (cf. grammaire de PALDES)

Nous avons vu qu'une variable était d'un type pris parmi les types prédéfinis, ou qu'elle était, par défaut du type entier machine.

>>> description PALDES minimal <<<<

Seuls les variables indicées ne peuvent être que du type entier machine, on les appelle tableaux. Le nombre des indices n'est pas limité.

Trois niveaux de déclaration sont permis en PALDES:

- Le niveau GLOBAL, les variables de ce niveau sont accessibles à tous les algorithmes du programme.

- Le niveau LOCAL, ce niveau se trouve déclaré dans un sous-algorithme, et les variables de ce niveau ne sont accessibles que dans le corps de cet algorithme.

Les variables locales ne peuvent transmettre une information que si elles sont passées comme paramètres dans l'appel à un autre algorithme.

NIVEAU GLOBAL:

exemples de déclarations:

global X1, X2, X3;

Les variables X1, X2 et X3 sont de type entier machine et accessibles dans tout le programme

global logique VERITAS, REPONSE;

Les variables VERITAS et REPONSE sont des booléens.

déclaration de tableaux:

global tableau INDIC(-5, 7, 0, -1), ..., INDIC(19, 20, 12, 7) ;

Remarque:

en PALDES les indices peuvent être négatifs.

Les déclarations de niveau GLOBAL se trouvent au début du programme comme en Pascal.

NIVEAU LOCAL:

exemples de déclarations:

local X1, X2, X3;

local logique VERITAS, REPONSE;

local liste LIST1, LIST2;

local tableau TAB(1, 34), ..., TAB(100, 20);

>>>> description FALDES minimal <<<<

La syntaxe des déclarations reste la même (cf. grammaire) que pour le niveau global

Les déclarations de niveau LOCAL, se trouvent après la déclaration de l'algorithme et de ses paramètres.

NIVEAU PARAMETRE:

---

La définition sémantique des paramètres en FALDES est identique à celle d'ALDES.

La seule façon de communiquer des informations autrement que par des variables GLOBALES, est de se servir des paramètres.

A la différence de Pascal, il n'y a pas en FALDES de paramètre mixte.

L'ensemble des paramètres est composés de la réunion de deux sous-ensembles DISJOINTS:

- Les paramètres d'entrée.
- Les paramètres de sortie.

Les paramètres d'entrée sont passés par "valeur".

Les paramètres de sortie sont passés par "adresses".

Un des intérêts du paramètre mixte en Pascal est que lorsque la variable à passer en entrée occupe une grande place en mémoire, la transmission par valeur fait effectuer une recopie qui est coûteuse en place mémoire et en temps.

Aussi peut-on utiliser, dans ce cas, le passage par adresse VAR...

Dans le cas de FALDES, cette possibilité est inutile car, les listes qui sont les plus encombrantes dans le système, ne sont référencées que par deux variables, la racine et la fin de liste.

Donc quelque soit la longueur et l'occupation mémoire d'une liste le coût du passage par valeur reste le même.

Une des contraintes de programmation de cette technique de séparation des paramètres, est que lorsque l'on veut modifier une variable par un algorithme, il faut utiliser une variable intermédiaire, que l'on affectera à la variable initiale après fin de l'exécution.

Les raisons de cette contrainte sont de deux ordres:

- éviter l'alourdissement des procédures systèmes lorsque ce cas ne se produit pas.

>>> description PALDES minimal <<<<

- surtout permettre le travail modulaire de façon à savoir exactement quelles sont les interfaces d'un algorithme à l'autre.

exemple:

-----  
faire A <-- A+1

soit :

```
    somme(x;y)
      (1) y:=x+1
      #
```

On ne peut pas écrire : somme (A;A);

il faudra faire:

```
somme(A;B);
A:=B;
.....
```

Le reste de la sémantique d'utilisation des paramètres est semblable au Pascal.

3) LES ALGORITHMES EN PALDES:  
=====

PALDES comporte trois types d'algorithmes différents.

- a) Les algorithmes.
- b) Les algorithmes fonctions.
- c) L'algorithme principal.

Un programme PALDES comporte obligatoirement un seul ALGORITHME PRINCIPAL, mais il peut comporter avant cet algorithme, zéro ou plusieurs algorithmes de type a) ou b).

Une règle syntaxique veut qu'un algorithme soit déclaré avant d'être utilisé. Il n'existe pas d'instruction FORWARD comme en Pascal.

Ceci pour éviter la récursivité croisée.

L'algorithme principal:  
-----

Il n'a pas de paramètres, c'est le programme principal du Pascal.

déclaration: < identif > .

exemple: SOMME .

Il ne travaille que sur les variables globales.

>>>> description PALDES minimal <<<<

L'algorithme fonction:

Il a un nombre quelconque de paramètres d'entrée mais un SEUL DE SORTIE, noté de façon spéciale.

déclaration < identif > := < identif > (< liste param >)

exemple: S:=SOMME(X,Y)

Ceci lorsque le paramètre de sortie est de type entier machine sinon il faut spécifier le type de l'algorithme (de son paramètre de sortie:

l'algorithme de parité PAIR ayant comme résultat une valeur logique RES:

RES := PAIR(X)=logique

l'algorithme:

Il a autant de paramètres de sortie et d'entrée que l'on veut.

Des exemples ont été donnés dans la description des algorithmes sur les listes d'entiers machine.

4) LES INSTRUCTIONS EXECUTABLES EN PALDES:

=====

PALDES comporte des instructions simples et des instructions composées.

Les instructions composées sont une composition séquentielle d'instructions simples, ou une instruction simple:

si  $I_a, I_b, \dots, I_n$  sont des instructions simples ( $n > 0$ ):

(  $I_a; I_b; \dots; I_n$  )

est une instruction composée, les symboles { et } étant des parenthèses d'énoncé.

Nous noterons {} par la suite l'instruction vide qui n'effectue aucune exécution.

INSTRUCTIONS SIMPLES:

-----



Elles sont au nombre de neuf et sont regroupées en cinq familles:

1- AFFECTATION:

---

affectation ordinaire sur tous les objets de type suivant:

- Entiers machines.
- Caractères.
- Chaines.
- Booléens.

Elle prend la forme:

$X := t ;$

Où X est soit un identificateur de variable simple, soit un identificateur de tableau.

t est un terme comme défini dans la C-grammaire, mais de même type que X.

L'évaluation dans un terme se fait de gauche à droite.

2- APPEL D'ALGORITHME:

---

De la forme :

$X ( Ta, Tb, \dots, Tk ; Va, \dots, Vp ) ;$

Où X est l'identifiacteur d'algorithme,  $Ta, \dots, Tk$  les paramètres effectifs d'entrée et  $Va, \dots, Vp$  les paramètres effectifs de sortie.

S'il n'y a pas de paramètres d'entrée on écrira:

$X ( ; Va, \dots, Vp )$

le point-virgule ne devra pas être omis.

Dans le cas où il n'y a pas de paramètres de sortie on aura deux possibilités:

$X ( Ta, \dots, Tk ; )$

ou bien  $X ( Ta, \dots, Tk )$

3- INSTRUCTIONS CONDITIONNELLES:

---

>>> description PALDES minimal <<<<

si P alors E1 alors E2;

Sémantique classique, si la proposition P est vraie l'instruction composée E1 est exécutée, si P est fausse on exécute alors l'instruction composée E2.

cas particulier:

si P alors E1 sinon () ;

Si P est fausse on ne fait rien exécuter et l'on passe en séquence.

On pourra noter en PALDES

si P alors E1 ;

4-INSTRUCTION DE TRANSFERT:

Malgré la forte structuration du langage ALDES/81, le transfert inconditionnel est autorisé, avec toutefois des contraintes qui permettent de le contrôler.

Un corps de programme PALDES est structuré en paragraphes étiquetés par un nombre entier, paragraphes se terminant par un point.

Il est possible en PALDES, dans un même corps de programme, de faire un transfert de l'intérieur d'un paragraphe vers le début d'un autre paragraphe du même corps.

l'instruction est notée ALLERA

exemple:

(23) .....

si ....

X := 567 .

(12) .....

ALLERA 23;

.....

>>>> description PALDES minimal <<<<

L'effet est le même que le GOTO en FORTRAN, transférant la suite de l'exécution à l'instruction étiquetée 23.

Une contrainte supplémentaire sécurisant l'emploi du ALLERA en PALDES, est que tout numéro de paragraphe qui est référencé dans un ALLERA, doit être déclaré au niveau des déclarations de l'algorithme.

Dans l'exemple précédent, le numero 23 doit avoir été déclaré par une instruction de déclaration de la forme:

étiquette 23;

Si cette déclaration n'est pas faite, il y aura production d'un message d'erreur.

## 5- INSTRUCTIONS ITERATIVES:

---

### 5-1 LA BOUCLE TANTQUE: =====

notation:  
-----

tantque P faire E1 ;

semantique:  
-----

A le même effet que:

(1) si non P alors ALLERA 2;

E1;

ALLERA 1;

.....

(2) .....

Qui signifie donc d'exécuter les actions de E1 tant que la proposition P reste VRAI.

### 5-2 LA BOUCLE POUR: =====

Forme générale:  
-----

pour X = Ta, Tb, ..., Tc faire E1;

>>> description PALDES minimal <<<<

Sémantique:

A le même effet que:

```
X := Ta;  
YO := Tb - X;  
Z := Tc;  
Y1 := SIGNN(YO);  
tantque Y1 * X <= Y1 * Z faire (E1; X := X + YO )
```

Forme particulière:

pour X = Ta, ..., Tc faire E1;  
sémantique:

A le même effet que:

```
X := Ta;  
Z := Tc;  
tantque X <= Z faire ( E1; X := X + 1 )
```

Elle correspond au cas particulier souvent rencontré, où l'indice de boucle X, a pour pas d'incrément 1.

Remarque:

l'indice de boucle est encore défini à la fin de la boucle, il a la dernière valeur augmentée du pas, soit m cette valeur de X après la boucle:

en posant  $p = T_b - T_a$

- si la suite des indices  $T_a, T_b, \dots, T_c$  est croissante alors  $p > 0$ :

m est le plus petit entier tel que:

$$T_a + (m-1) * p \leq T_c < T_a + m * p$$

- si la suite des indices  $T_a, T_b, \dots, T_c$  est décroissante alors  $p < 0$ :

m est le plus petit entier tel que:

$$T_a + m * p \leq T_c < T_a + (m-1) * p$$

5-3 LA BOUCLE REPETER:

notation:

répéter E1 jusqu'à P;

sémantique:

>>> description PALDES minimal <<<<

A le même effet que:

- (1) E1;  
si non P alors ALLERA 1;

5-4 LE TEST MULTIPLE:

=====

notation:

selon X { a1,...,an faire E1;  
          b1,...,bp faire E2;  
          .....  
          q1,...,qm faire Er }

sémantique:

semblable au CASE OF du Pascal.

A le même effet que:

si (X=a1) ou (X=a2) ou....ou (X=an) alors E1  
sinon  
si (X=b1) ou (X=b2) ou....ou (X=bp) alors E2  
sinon  
.....  
sinon  
si (X=q1) ou (X=q2) ou....ou (X=qm) alors Er;

X est une expression de type entier machine, ou une variable de type caractère.

Soit l'ensemble totalement ordonné:

$$T = \{ a1, a2, \dots, an, b1, b2, \dots, q1, \dots, qm \}$$

remarques:

1) Comme en ALDES, il est préférable pour des raisons de place mémoire, de ne pas avoir une "distance" trop grande entre les valeurs des choix possibles:

Dans l'ensemble T défini plus haut, soient:

$$E_i = \min T$$
$$E_j = \max T$$

La traduction se faisant sous forme de CASE OF, il faut tenir compte du fait suivant:

>>> description PALDES minimal <<<<

Le compilateur Pascal UCSD, comme certains autres compilateurs Pascal, afin d'optimiser le temps d'exécution du CASE OF, réserve autant de mots mémoire qu'il y a d'éléments dans l'intervalle (Ei..Ej)

exemple:

```
selon X ( 1 faire X:=1;
          32000 faire x:=2;
          567 faire x:=3 )
```

Ei=1 , Ej=32000 ,le compilateur Pascal réservera 32000-1 soient 31999 mots.

2) si l'expression X du selon n'est pas dans l'ensemble T, on voit que le traitement se poursuit en séquence.

5-5 LES ASSERTIONS PALDES:

notation:

assertion < chaine >: F;

semantique:

Cette instruction suit la définition qu'en donne J.ARSAC dans (3):

"Une assertion est un prédicat liant certaines variables du programme et placée en un point précis de celui-ci. La meilleur façon d'en rendre compte est de considérer l'assertion comme une instruction de vérification:

- lors de l'exécution du programme, elle est calculée quand on la rencontre, comme n'importe quelle autre instruction.

Elle, DOIT avoir la valeur VRAI, autrement il y a incohérence entre les instructions et les assertions. Une assertion n'est pas un élément de l'exécution: elle ne modifie aucune variable du programme, et a, par rapport à celui-ci la valeur d'un commentaire.

Une assertion est localisée dans le programme, comme le serait une instruction:

- elle précède une instruction c'est une PRE-ASSERTION,

- elle suit une instruction c'est une

POST-ASSERTION."

L'instruction assertion en PALDES, suit rigoureusement cette définition.

Elle possède une partie <chaine> permettant de mettre un commentaire référant cette assertion. Elle travaille sur des propositions F telles qu'on les trouve dans instructions conditionnelles en PALDES.

Le système PALDES, détectant une incohérence (assertion fausse) enverra un message au terminal avec la référence <chaine> de l'assertion.

Cette instruction s'insère dans une stratégie d'aide à la preuve de programme en PALDES, chaque instruction du langage a un effet particulier sur les variables intervenant dans sa pré-assertion. On peut alors dire qu'une instruction "transforme" sa pré-assertion en sa post-assertion. Nous avons mis le mot transformation entre guillemets, pour qu'il n'y ait pas de confusion avec une transformation au sens mathématique, car il n'y a pas unicité de la post-assertion.

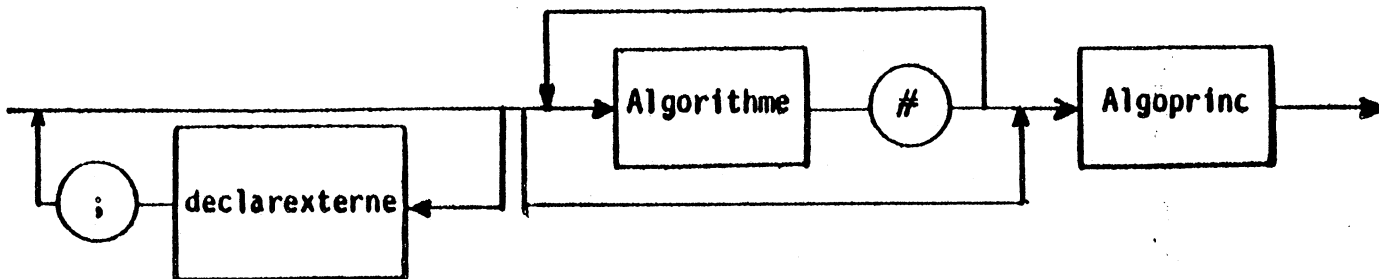
DIAGRAMMES SYNTAXIQUES PALDES

GRAMMAIRE LL(1)

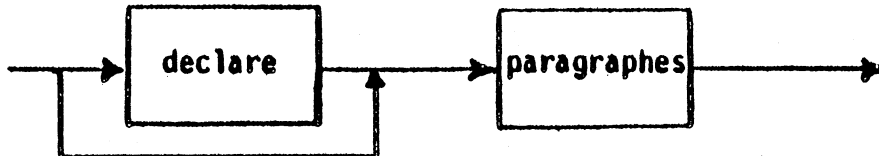
Un programme ALDES est constitué de paragraphes.



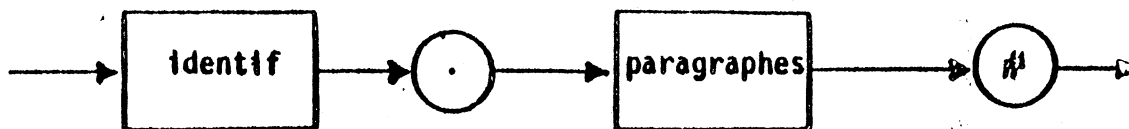
\* PROGRAMME :



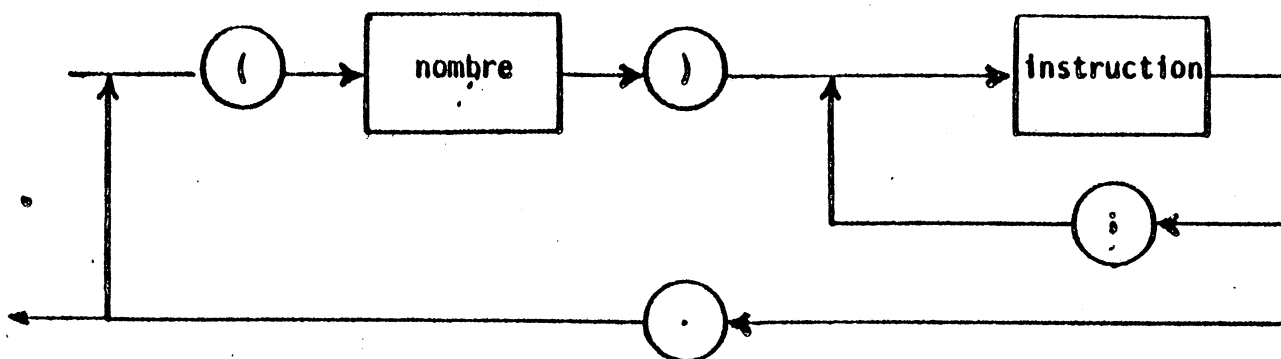
\* ALGORITHME :



\* ALGO PRINC :

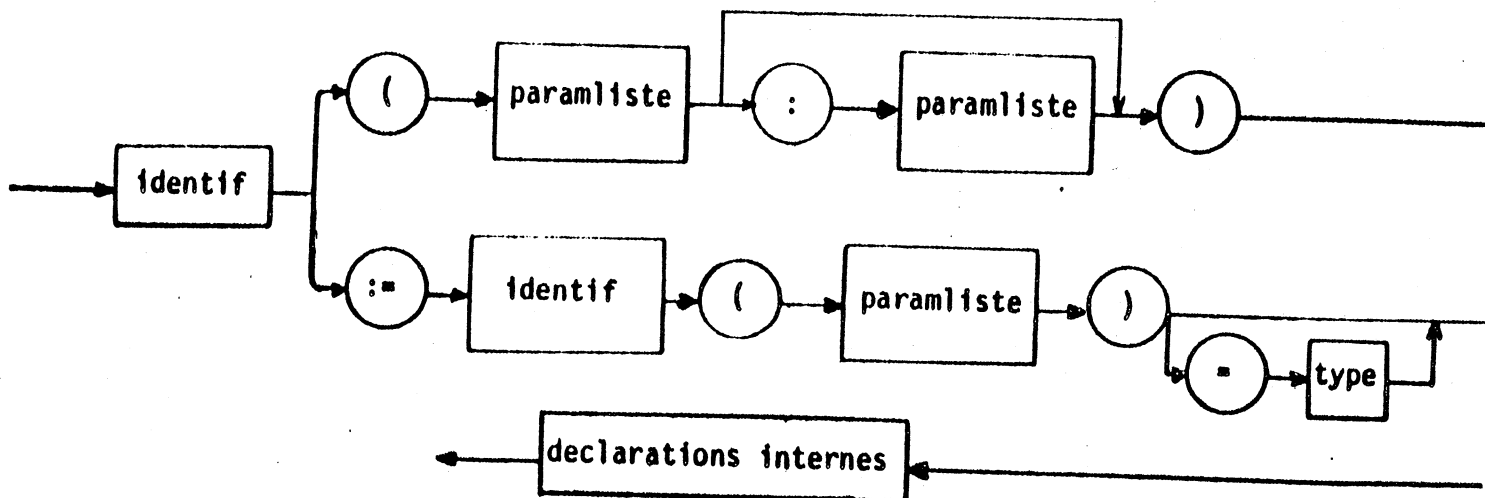


\* PARAGRAPHES :

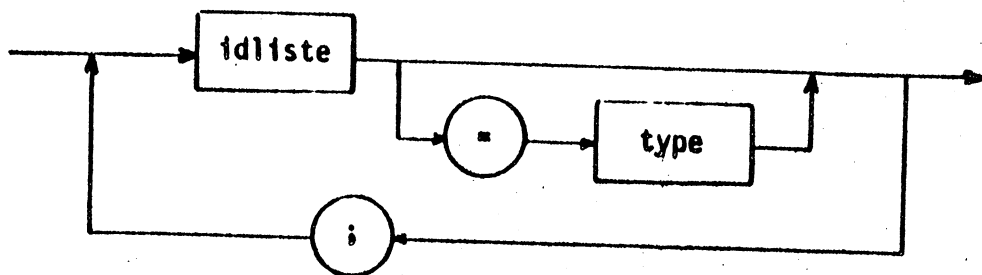




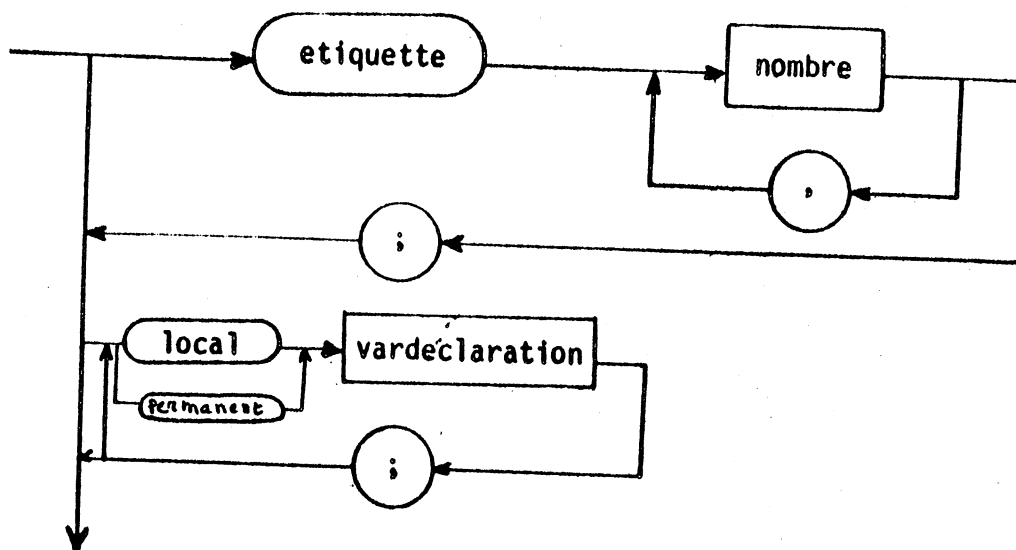
\* DECLARE :



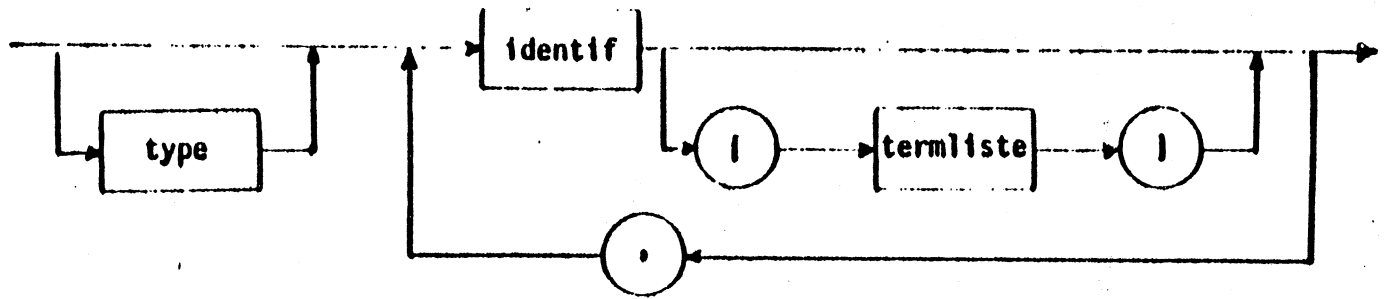
\* PARAMLISTE :



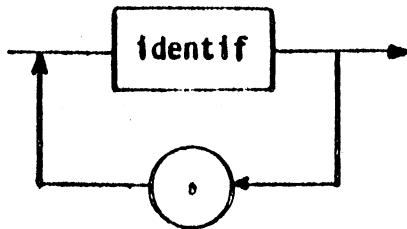
\* DECLARATIONS INTERNES



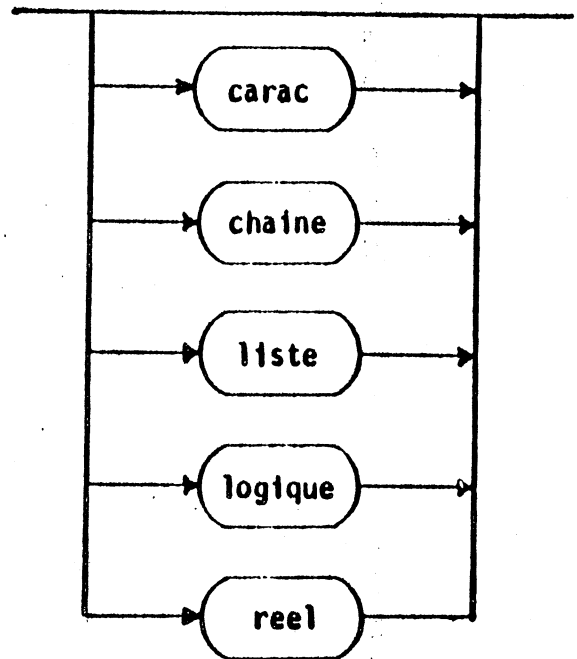
\* VARDECLARATION :



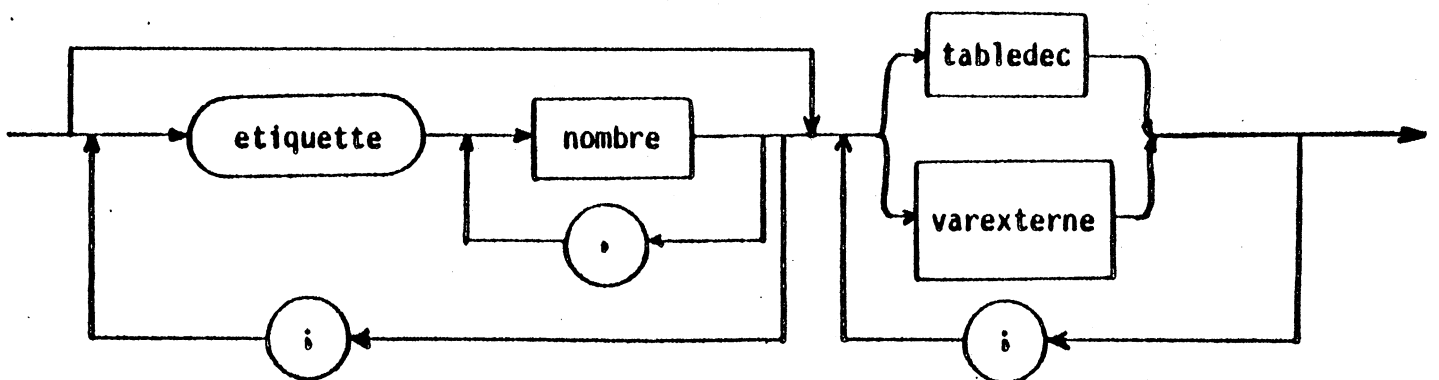
\* IDLISTE :



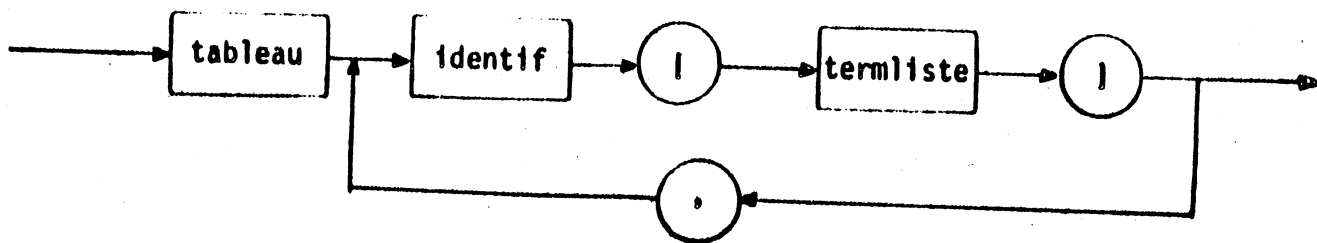
TYPE :



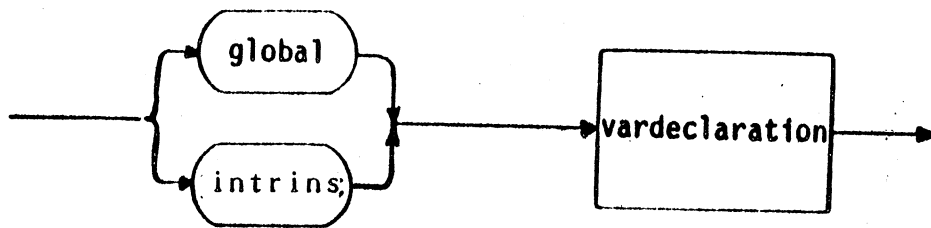
\* DECLAREXTERNE :



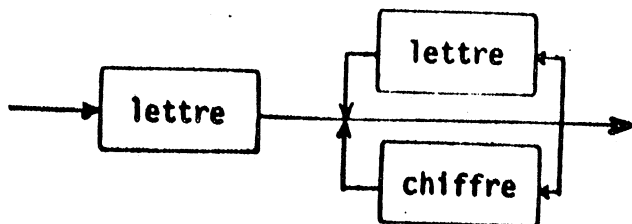
\* TABLEDEC :



\* VAREXTERNE :



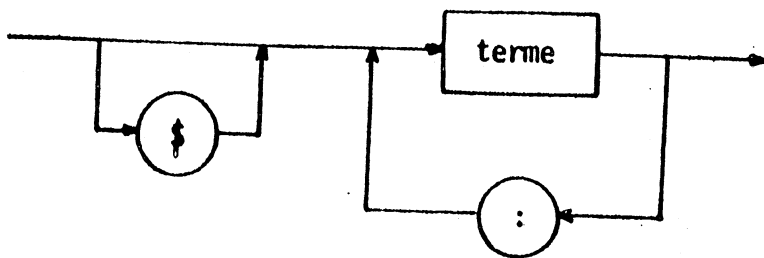
\* IDENTIF



\* NOMBRE :

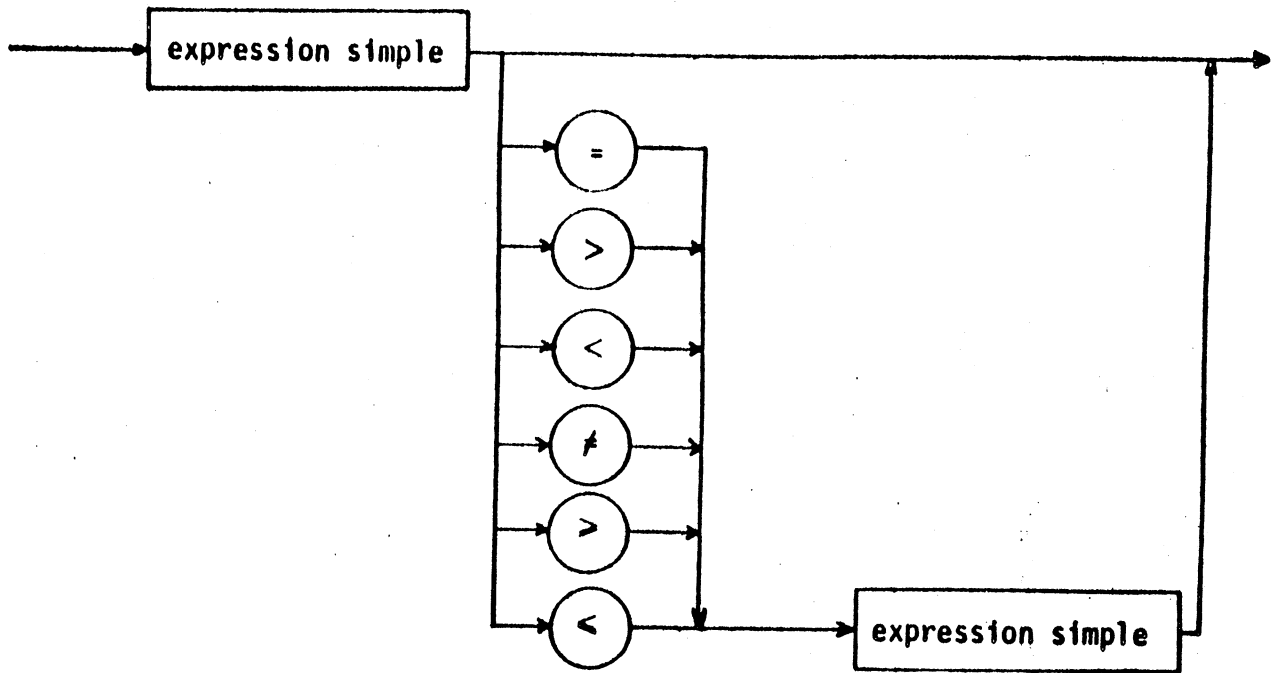


\* TERMIO

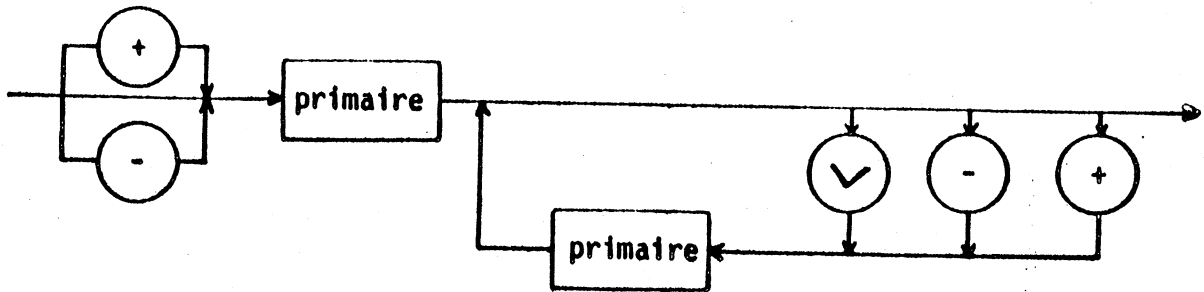




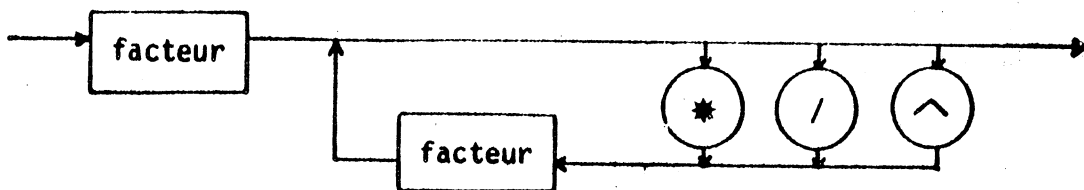
\* TERME :



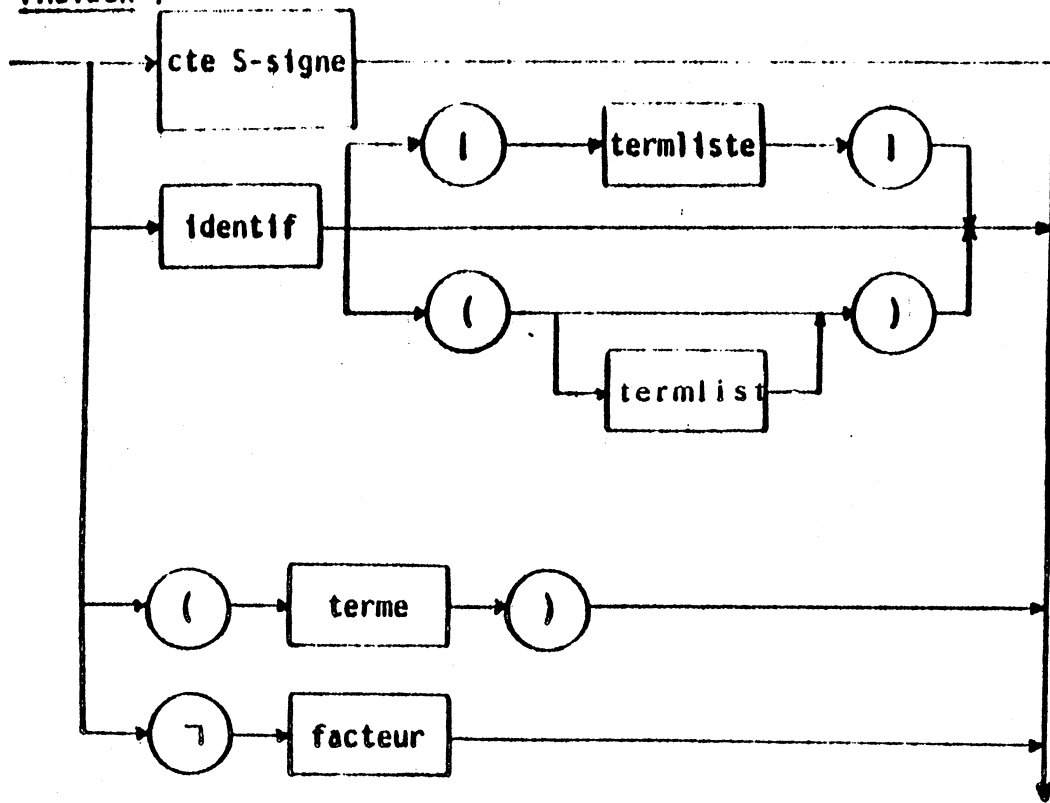
\* EXPRESSION SIMPLE



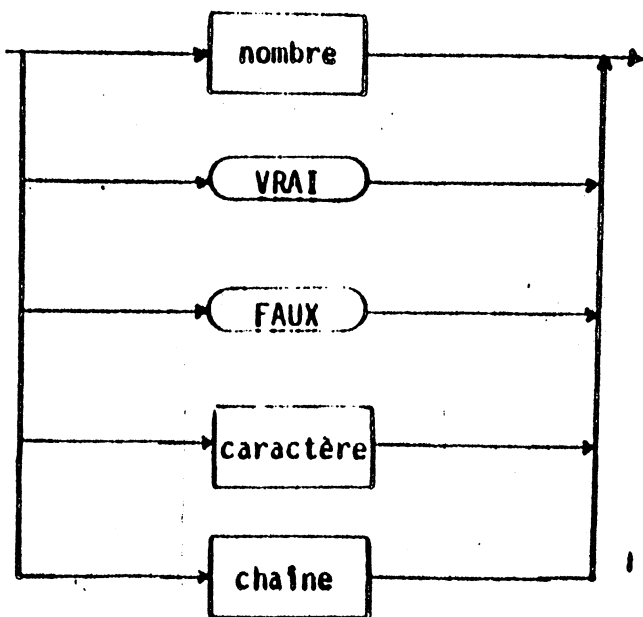
\* PRIMAIRE



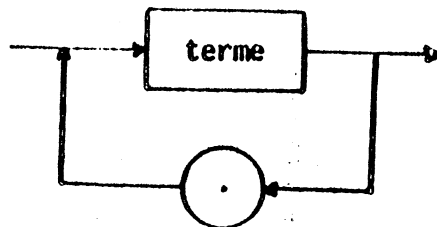
\* FACTEUR :



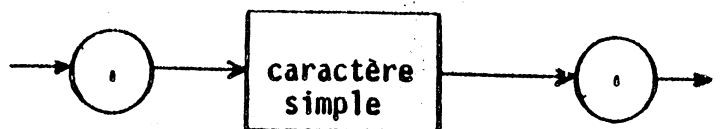
\* CTE S-SIGNE :



\* TERMLISTE :



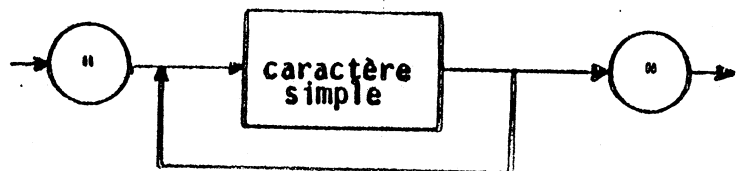
\* CARACTERE :



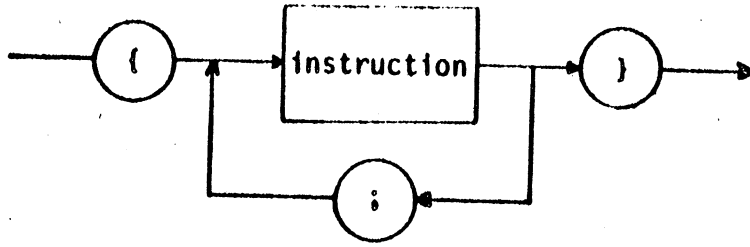
\* CONDITION :



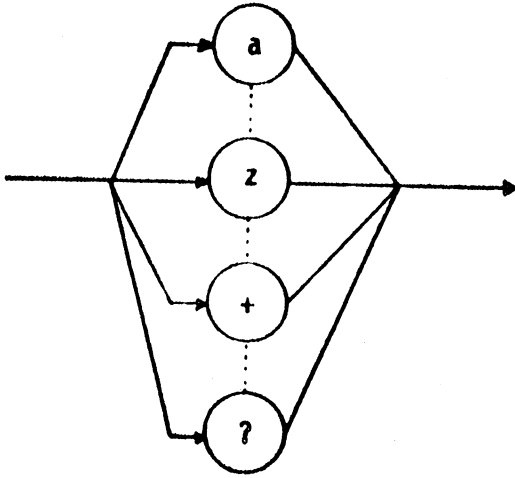
\* CHAINE



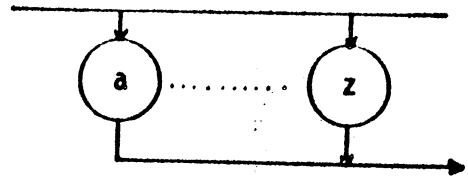
**\* SEQUENCE :**



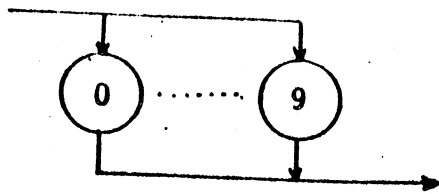
**\* CARACTERE SIMPLE :**



**\* LETTRE :**



**\* CHIFFRE :**



CHAPITRE II



## COMPILATEUR PALDES-PASCAL

-----

Par la suite le sous-ensemble Pascal qui se trouve être l'intersection entre le Pascal Wirth (du User Manual Report) et le Pascal UCSD sera dénommé Pascal minimal, ou Pascal.

### FONCTIONNEMENT GENERAL DU COMPILATEUR

Le compilateur est écrit en Pascal UCSD sur un micro-ordinateur individuel du commerce.

Il analyse un texte source écrit en PALDES et le traduit en Pascal minimal. Dans toute la suite de ce document le texte traduit sera dénommé code généré par analogie avec le code machine généré par un compilateur général.

le code généré est du Pascal wirth, il est en suite soumis au compilateur Pascal UCSD du micro-ordinateur pour exécution.

Ce choix fait donc d'une compilation PALDES, une opération qui se passe en deux phases:

- analyse, interprétation, traduction,
- compilation Pascal minimal.

Ceci permet de répondre d'une certaine façon à l'objectif numéro 2) de portabilité en Pascal.

Le compilateur PALDES se compose sémantiquement de deux parties distinctes bien que liées dans la rédaction du programme:

- l'analyseur syntaxique,
- le générateur de code .

Le compilateur est prévu pour être intégré dans un système PALDES interactif dénoté SYPAC, il est conçu de façon conversationnelle au niveau de l'analyseur, il peut être étendu avec de nouvelles fonctions systèmes qui formeront un système plus évolué.

Cette conception modulaire est obtenue par la mise en bibliothèque des nouvelles unités systèmes et par mise de la table de correspondance des références d'algorithmes systèmes sur un fichier à accès séquentiel.

Grâce à cette organisation le système PALDES a pu déjà subir des évolutions et des extensions.

ORGANISATION GENERALE DU COMPILATEUR

---

Modules composants le compilateur:

=====

- déclarations
- initialisations
- génération
- reconnaissance
- gestion table des symboles
- analyse lexicographique
- debugging

Voyons rapidement le rôle de chacun de ces modules avant de les détailler.

ANALYSE LEXICOGRAPHIQUE:

-----

Son rôle est de lire sur un fichier le texte source PALDES qui a déjà été saisi, de faire un premier tri entre des mots ou des symboles possibles en PALDES.

Il n'a qu'une action sémantique très limitée au niveau du vocabulaire terminal de la grammaire de PALDES.

RECONNAISSANCE:

-----

Dans ce module, se trouvent toutes les procédures correspondant aux paragraphes syntaxiques de la grammaire LL(1) de PALDES, les paragraphes décrivant la partie instruction du langage, et les paragraphes associés comme: expression, facteur, ...etc.

On trouve aussi des procédures assurant l'interface avec les modules de génération, comme fournir un nouvel identificateur, reconnaître et coder le type d'un élément.

DECLARATIONS:

-----

Ce module contient le reste des procédures associées aux paragraphes syntaxiques décrivant les parties déclarations globales, locales ou paramètres, ainsi que les procédures décrivant la structure générale d'un programme PALDES comme: programme, algorithme, ...etc.

INITIALISATIONS:

-----

## >>>> compilateur PALDES <<<<

Contient toutes les procédures permettant l'initialisation de tous les paramètres, les mots d'états et les tableaux du compilateur. Ce module contient aussi les mots réservés en français, en anglais et le jeu de caractères de base de PALDES.

Ce module initialise aussi la table des algorithmes systèmes à partir d'un fichier disque, c'est ainsi qu'on obtient l'extensibilité du système sans avoir à recompiler le texte source du compilateur.

### GENERATION:

---

Cette famille de procédures permet de générer dans un fichier disque le Pascal minimal qui est la traduction du programme PALDES en cours de compilation.

Par exemple chacun des identificateurs PALDES reçoit un codage, puis est stocké dans la table des symboles, une des procédures de la famille est chargée de transmettre au fichier disque le codage de l'identificateur à chaque apparition d'une de ses occurrences.

### GESTION DE LA TABLE DES SYMBOLES:

---

Une famille de procédures est chargée de ranger, trier, accéder à certains éléments de la table des symboles et même de tasser certaines parties de cette table. Certaines de ces procédures servent de liaison avec d'autres modules du compilateur.

### DEBUGGING:

---

Pour l'instant, cette famille de procédures a pour rôle l'impression détaillée d'un listing des références de la table des symboles au cours de l'analyse.

### PROGRAMME PRINCIPAL:

---

Enfin cet ensemble de modules est inséré dans un programme principal qui constitue la trame du compilateur.

Dans ce programme se trouvent toutes les déclarations des éléments utilisés par les modules pour communiquer, on trouve aussi les types de base permettant des actions sémantiques précises.

Le programme ordonne selon la syntaxe du Pascal les différentes procédures de chaque module. En effet la conception récursive de l'analyseur amène très souvent à des récursivités croisées entre certaines procédures de l'analyseur.

Ce programme est aussi chargé de lancer conversationnellement l'interprétation.

les types de données internes du compilateur:

=====  
Pour permettre à chacun des modules précédents de fonctionner, les unités de données sont parfaitement définies par des variables globales en Pascal.

Afin d'augmenter la modularité et surtout la lisibilité du programme des classes de données ont été créés en utilisant les TYPES Pascal.

Les équivalents symboliques:

-----  
Pour les éléments du vocabulaire terminal (Vt) un type de même cardinal que Vt a été créé, il s'agit du type SYMBOLE:

TYPE SYMBOLE= (non, plus, moins, dollar, égal, neg, ptvir, identif, virg, fair, pourc, parg, pard, crog, crod, pt, apost, guill, carac, entree, ou, et, sup, inf, supeg, infeg, diese, étoile, sortie, assert, compile, arret, logik, divi, puiss, imprim, allera, si, alors, sinon, nul, retour, réel, poly, vérité, cas, tantque, repete, jusque, pour, etc, global, chaine, intrins, safe, nomb, point, arrai, affect, local, branch, liste, dept )

Les éléments de ce type seront appelés les 'équivalents symboliques' des éléments de Vt.

Il y a donc une bijection entre Vt et l'ensemble des équivalents symboliques, cette conception permet une structuration du système qui, comme dans le cas de la table des algorithmes systèmes, autorise une grande souplesse.

Ici chaque implementation peut se permettre d'avoir des éléments de Vt différents (cas de terminaux ne possédant pas certains caractères), sans que l'on ait à modifier quoi que ce soit dans le compilateur. Les mots réservés peuvent donc être modifiés, le système implementé propose d'ailleurs deux jeux de tels mots, l'un en français, l'autre en anglais. Il est donc possible en respectant le sens profond des symboles, de choisir d'autres mots pour le langage PALDES. Il en est de même pour les caractères.

Il faut toutefois respecter le nombre exact d'éléments symboliques.

Les types sémantiques de PALDES:

-----  
Nous savons qu'un programme PALDES est composé de déclarations et d'instructions, chacun des identificateurs du programme a donc un type précis, le compilateur possède donc deux ensembles de type:

TYPE TYPAL= (algorithme, algo princ, fonction, fctsys )

## >>>> compilateur FALDES <<<<

TYPE TYPVAR=(varcar, sortifct, varpol, varchain, varent, vareel, varlist, varlogic, tableauent, param, varprec)

Le TYPAL sert à caractériser les différents algorithmes que l'on peut rencontrer dans un programme FALDES

Le TYPVAR caractérise les types et les états des variables d'un programme.

Remarquons que sortifct et param ne sont pas des types de variables FALDES, mais des types internes au compilateur, en vue d'actions particulières.

### Les états sémantiques du compilateur:

---

Le compilateur dispose de plusieurs types d'états permettant à chaque module de savoir à tout instant de l'analyse ou de l'interprétation, ce qui se passe.

TYPE SEMANTIK = (creafsys, declaralg, declarext, declarint, declaretiq, utilisation, declarpar)

TYPE STATU=(sglobal, ssafe, sintrinsic, slocal, sbranch, sparame, sparams)

TYPE IDETAT=(vide, absent, present)

Le type SEMANTIK permet de communiquer par une variable globale Pascal, l'état actuel de l'avancement de l'analyse.

Le type STATU est un sous-état du précédent permettant d'affiner la connaissance de l'avancement de l'analyse.

Il sert surtout à traduire à l'intention du module de gestion de la table des symboles, le statut des identificateurs analysés.

Le type IDETAT est un mot d'état de communication entre les procédures de recherche ou d'insertion dans la table des symboles, et le reste des modules.

### Les données de travail du compilateur:

---

le compilateur travaille avec trois fichiers séquentiels, deux en entrée un en sortie, et un fichier à accès direct.

Le fichier de sortie PLIGNE, est rempli à travers le tampon du système Pascal, par le module de génération.

Le fichier d'entrée G, contenant la table des algorithmes systèmes est utilisé par le module d'initialisations.

Le fichier d'entrée F, contenant le texte source FALDES est utilisé par le module d'analyse lexicographique à

>>> compilateur PALDES <<<

travers un tampon matérialisé par un tableau de 80 caractères noté LIGNE.

Le fichier MESSERR sert à donner en clair les messages d'erreur qui sont générés par le compilateur, il est donc possible la aussi de modifier les messages d'erreurs a volonté.

Il existe onze mots logique d'états particuliers du compilateur servant de drapeaux à la partie interprétation et génération, dont certains proviennent de messages d'options de compilation:

var ETIVAR, PREMVAR, SECVAR, OK, STOP, POINTVIR, IO, IMPTAB, GAUCHE, DEJA: boolean

Par exemple:

- Le mot OK provient du message de l'option 'mode analyseur seul', alors le mot ok fourni au module de génération, a pour effet de supprimer la génération, ce mot est aussi positionné par une erreur syntaxique.

- Le mot STOP permet au sous-module de l'error recovery', du module reconnaissance, de savoir qu'une erreur grave à l'analyse ou dans le transfert, s'est produite, et demande l'abandon de l'interprétation.

- Le mot GAUCHE sert à reconnaître le type d'un identificateur dans une affectation, si l'on rencontre cet identificateur dans la partie gauche, son type doit être conservé pour la vérification statique des types de la partie droite.

- Le mot IMPTAB provient d'une option de compilation qui valide le débugging.

- Le mot DEJA est un mot de communication interne a plusieurs modules permettant au module de gestion de la table des symboles d'accélérer l'entrée d'un code d'identificateur dans la table des symboles: si l'identificateur a déjà été codé il sera inutile pour le ranger de faire appel à la procédure engendrant un nouveau code, le codage de l'identificateur se trouve déjà dans IDCODE.

- Le mot POINTVIR sert à signaler à la génération qu'un point-virgule a déjà été généré pour l'instruction traduite.

- Les mots ETIVAR, SECVAR et PREMVAR permettent au module de génération de fabriquer un programme Pascal syntaxiquement correct.

Le reste des données générales du compilateur se réparti en mots contenant les contraintes d'utilisation et

>>> compilateur PALDES <<<

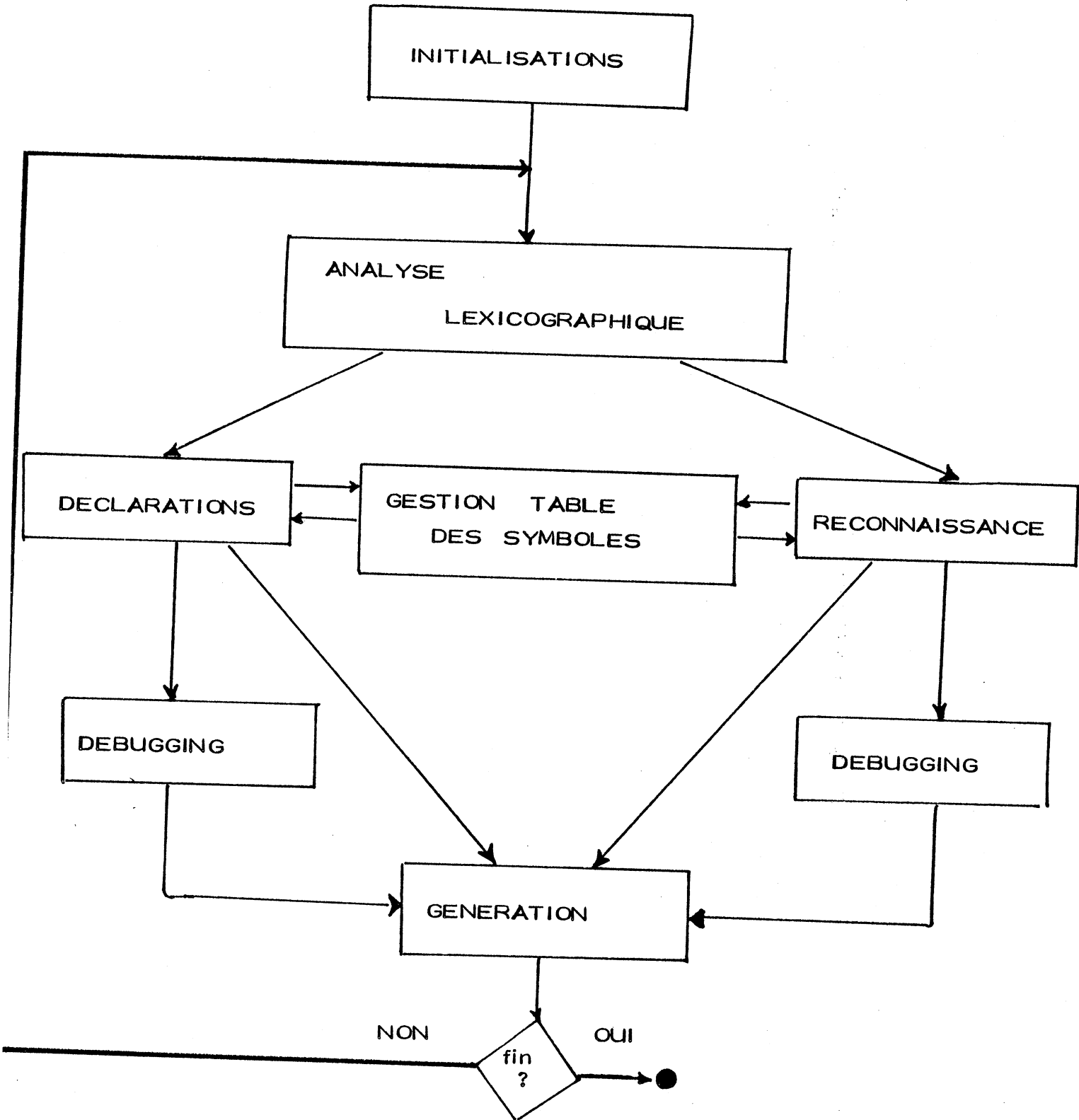
d'implémentation, comme la longueur utile des identificateurs,  
le nombre d'éléments de la table des symboles ou des autres  
tables de service...etc.

## CHAPITRE III

Ce chapitre décrit les modules composant le compilateur PALDES, leur fonctionnement, leur hiérarchisation et leur utilisation au cours de la compilation d'un programme PALDES



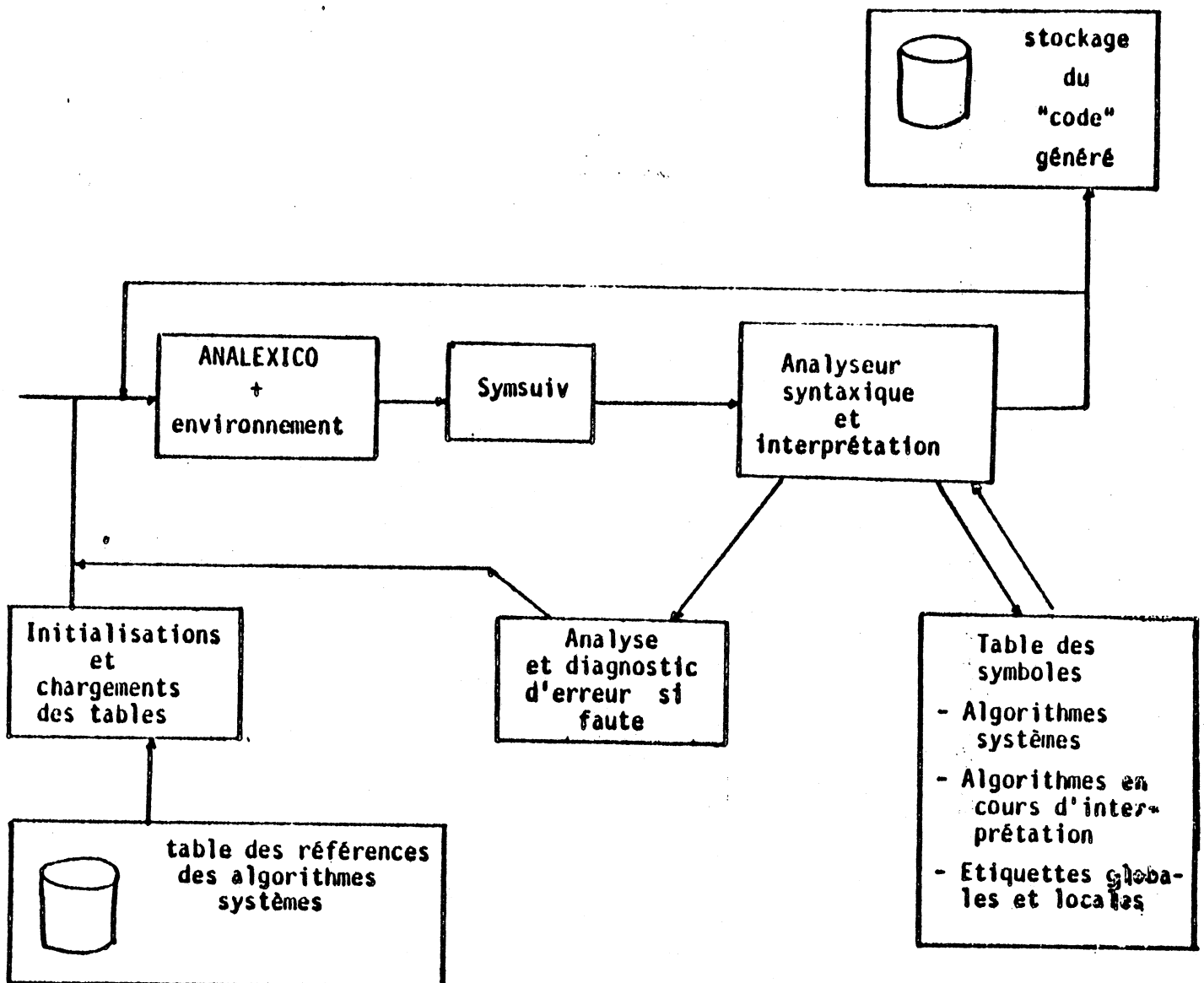
ORGANISATION GENERALE et MODULES  
du  
COMPILATEUR PALDES



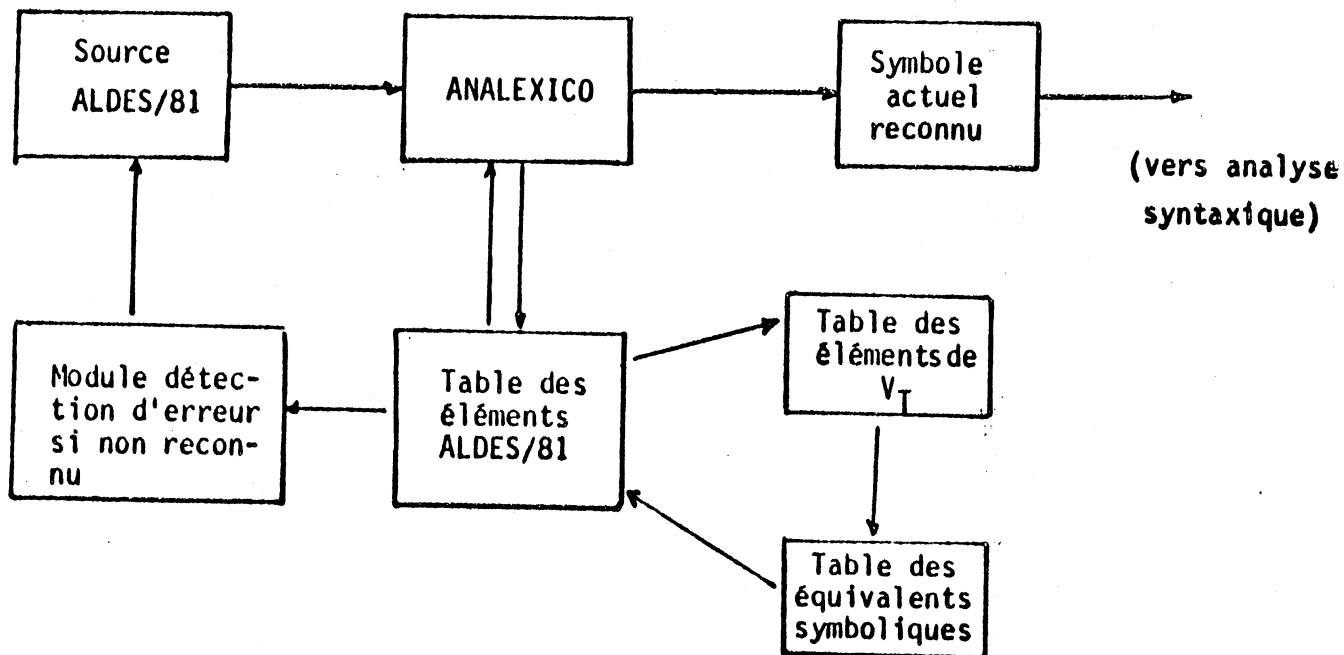
>>>> modules du compilateur PALDES <<<<

MODULE D'ANALYSE LEXICOGRAPHIQUE ANALEXICO:  
=====

INSERTION DE L'ANALYSEUR ANALEXICO DANS LE COMPILATEUR



ANALEXICO et son environnement:



Le module ANALEXICO reconnaît les éléments du langage PALDES, les mots réservés, les identificateurs, les caractères. Les identificateurs peuvent avoir une longueur quelconque, seuls les  $n$  premiers caractères permettent de les différencier. Le nombre  $n$  est fixé dans le compilateur comme une constante globale.

Le module ANALEXICO travaille sur trois tables:

- La table des éléments de  $V_T$  (vocabulaire terminal)
- La table des équivalents symboliques.
- La table des éléments PALDES.

La table des éléments de  $V_T$ :

Elle est composée de deux tableaux:

MOT (1..MMAX) et SSYM (CHAR)

a) Le tableau MOT(1..MMAX) contient les mots réservés PALDES.

b) Le tableau SSYM(CHAR) dont l'indice varie dans l'ensemble des caractères simples de PALDES, contient l'équivalent symbolique du caractère indexé.

La table des équivalents symboliques:

---

Elle est composée de deux tableaux, le tableau SSYM déjà cité, puisqu'il contient des équivalents symboliques et le tableau MSYM (1..MMAX) qui contient les équivalents symboliques des mots de MOT(1..MMAX)

La table des éléments PALDES:

---

C'est une succession de tests sur des suites d'équivalents symboliques afin de reconnaître certains mots du langage comme les identificateurs, les symboles '=', '>=', ',...', '...etc

Le module ANALEXICO par l'intermédiaire de cette table envoie des informations au module de gestion des erreurs, si par contre un symbole est reconnu, son équivalent symbolique est rangé dans le mot SYM, mot global du compilateur.

Dans le programme l'appel à ANALEXICO se fait par la procédure SYMSUIV.

MODULE DES INITIALISATIONS:

=====

c'est ce module qui initialise les tables d'ANALEXICO:

PROCEDURE INITFR;

VAR I, N: INTEGER;

BEGIN

```

MOT[ 1 ]:= 'ALLERA      ' ) MOT[ 2 ]:= 'ALORS        ' ) MOT[ 3 ]:= 'CARAC      ' )
MOT[ 4 ]:= 'CHAINE     ' ) MOT[ 5 ]:= 'ENTRER      ' ) MOT[ 6 ]:= 'ET         ' )
MOT[ 7 ]:= 'ETIQUETTE ' ) MOT[ 8 ]:= 'FAIRE       ' ) MOT[ 9 ]:= 'FAUX      ' )
MOT[10 ]:= 'GLOBAL     ' ) MOT[11 ]:= 'IMPRIMER   ' ) MOT[12 ]:= 'INTRINSEQ ' )
MOT[13 ]:= 'JUSQUE    ' ) MOT[14 ]:= 'LISTE      ' ) MOT[15 ]:= 'LOCAL     ' )
MOT[16 ]:= 'LOGIQUE   ' ) MOT[17 ]:= 'NON        ' ) MOT[18 ]:= 'OU        ' )
MOT[19 ]:= 'PERMANENT ' ) MOT[20 ]:= 'POUR      ' ) MOT[21 ]:= 'REEL     ' )
MOT[22 ]:= 'REPETER   ' ) MOT[23 ]:= 'RETOUR    ' ) MOT[24 ]:= 'SELON    ' )
MOT[25 ]:= 'SI        ' ) MOT[26 ]:= 'SINON     ' ) MOT[27 ]:= 'SORTIR   ' )
MOT[28 ]:= 'STOP      ' ) MOT[29 ]:= 'TABLEAU  ' ) MOT[30 ]:= 'TANTQUE  ' )
MOT[31 ]:= 'VRAI     ' ) MOT[32 ]:= 'XCHIFFRE  ' ) MOT[33 ]:= 'ZASSERTION' )
MOT[34 ]:= 'ZPRAGMA  ' )

```

END;

PROCEDURE INSYMFR;

BEGIN

```

MSYM[ 1 ]:= ALLERA; MSYM[ 2 ]:= ALORS; MSYM[ 3 ]:= CARAC; MSYM[ 4 ]:= CHAINE;
MSYM[ 5 ]:= ENTREE; MSYM[ 6 ]:= ET; MSYM[ 7 ]:= BRANCH; MSYM[ 8 ]:= FAIR;
MSYM[ 9 ]:= VERITE; MSYM[10 ]:= GLOBAL; MSYM[11 ]:= IMPRIM; MSYM[12 ]:= INTRINS;
MSYM[13 ]:= JUSQUE; MSYM[14 ]:= LISTE; MSYM[15 ]:= LOCAL; MSYM[16 ]:= LOGIK;
MSYM[17 ]:= NON; MSYM[18 ]:= OU; MSYM[19 ]:= SAFE; MSYM[20 ]:= POUR;
MSYM[21 ]:= REEL; MSYM[22 ]:= REPETE; MSYM[23 ]:= RETOUR; MSYM[24 ]:= CAS;
MSYM[25 ]:= SI; MSYM[26 ]:= SINON; MSYM[27 ]:= SORTIE; MSYM[28 ]:= ARRET;
MSYM[29 ]:= ARRAI; MSYM[30 ]:= TANTQUE; MSYM[31 ]:= VERITE; MSYM[32 ]:= PRECIS;
MSYM[33 ]:= ASSERT; MSYM[34 ]:= COMPILER

```

END;

BEGIN

LANGUE;

INIT2;

```

SSYM[ ' " ' ]:= GUILL; SSYM[ ' # ' ]:= DIESE;
SSYM[ ' $ ' ]:= DOLLAR; SSYM[ ' % ' ]:= POURC;
SSYM[ ' ' ' ]:= APOST; SSYM[ ' ( ' ]:= PARG;
SSYM[ ' ) ' ]:= PARD; SSYM[ ' * ' ]:= ETOILE;
SSYM[ ' + ' ]:= PLUS; SSYM[ ' , ' ]:= VIRG;
SSYM[ ' - ' ]:= MOINS; SSYM[ ' . ' ]:= PT;
SSYM[ ' / ' ]:= DIVI; SSYM[ ' ' ' ]:= PTVIR;
SSYM[ ' > ' ]:= SUP; SSYM[ ' < ' ]:= INF;
SSYM[ ' = ' ]:= EGAL; SSYM[ ' [ ' ]:= CROG;
SSYM[ ' ] ' ]:= CROD; SSYM[ ' ^ ' ]:= PUISS;
SSYM[ ' : ' ]:= DEPT;

```

PAGE(OUTPUT)

END; (\* SEGMENT 3 INITIALE \*)

Ceci, correspond a l'initialisation des trois tableaux cites dans ANALEXICO, pour une implantation avec des mots

Français.

On trouvera à la suite de ce texte, une initialisation pour une implantation en anglais.

SEGMENT PROCEDURE INITIALE)

PROCEDURE INITANGL;

VAR I,N: INTEGER;

BEGIN

```
MOT(1) := 'AND'      ; MOT(2) := 'ARRAY'    ; MOT(3) := 'CASE'      ;
MOT(4) := 'CHARAC'  ; MOT(5) := 'DO'        ; MOT(6) := 'ELSE'     ;
MOT(7) := 'FALSE'   ; MOT(8) := 'FOR'        ; MOT(9) := 'GLOBAL'   ;
MOT(10) := 'GOTO'   ; MOT(11) := 'IF'         ; MOT(12) := 'INPUT'   ;
MOT(13) := 'INTRINSIC' ; MOT(14) := 'LABEL'    ; MOT(15) := 'LIST'    ;
MOT(16) := 'LOCAL'  ; MOT(17) := 'LOGICAL'   ; MOT(18) := 'NOT'     ;
MOT(19) := 'OR'     ; MOT(20) := 'PRINT'    ; MOT(21) := 'REAL'    ;
MOT(22) := 'REPEAT' ; MOT(23) := 'RETURN'   ; MOT(24) := 'SAFE'    ;
MOT(25) := 'STOP'   ; MOT(26) := 'STRING'   ; MOT(27) := 'THEN'    ;
MOT(28) := 'TRUE'   ; MOT(29) := 'UNTIL'    ; MOT(30) := 'WHILE'   ;
MOT(31) := 'XDIGIT' ; MOT(32) := 'XOUT'    ; MOT(33) := 'ZASSERT' ;
MOT(34) := 'ZPRAGMA' ;
```

END;

PROCEDURE INSYMAN;

BEGIN

```
MSYM(10) := ALLERA; MSYM(27) := ALORS; MSYM(4) := CARAC; MSYM(26) := CHAINE;
MSYM(12) := ENTREE; MSYM(1) := ET; MSYM(14) := BRANCH; MSYM(5) := FAIR;
MSYM(7) := VERITE; MSYM(9) := GLOBAL; MSYM(20) := IMPRIM; MSYM(13) := INTRINS;
MSYM(29) := JUSQUE; MSYM(15) := LISTE; MSYM(16) := LOCAL; MSYM(17) := LOGIK;
MSYM(18) := NON; MSYM(19) := OU; MSYM(24) := SAFE; MSYM(8) := POUR;
MSYM(21) := REEL; MSYM(22) := REPETE; MSYM(23) := RETOUR; MSYM(3) := CAS;
MSYM(11) := SI; MSYM(6) := SINON; MSYM(31) := SORTIE; MSYM(25) := ARRET;
MSYM(2) := ARRAT; MSYM(30) := TANTQUE; MSYM(28) := VERITE; MSYM(31) := PRECIS;
MSYM(33) := ASSERT; MSYM(34) := COMPILER
```

END;

C'est dans ce module que les ensembles INIT construits à partir de la grammaire LL(1) de PALDES, sont initialisés, ainsi que tous les autres ensembles ou sous-ensembles utilisés dans le programme. Les mots d'états sont aussi tous initialisés ici.

Enfin c'est à partir de ce module qu'un dialogue s'établit au terminal pour pouvoir décider de certaines options de compilation (listing, debug,...)

MODULE DE GESTION DE LA TABLE DES SYMBOLES:  
=====

Organisation de la table des symboles:  
-----

Nous avons définis deux TYPE supplémentaires TYP qui est la réunion des TYPE TYPAL et TYPVAR, qui sert à donner le type de n'importe quel identificateur.

Puis le TYPE TYPARAM permettant d'avoir la liste des paramètres et leur type, pour un algorithme donné:

```
TYPE TYPARAM = ^ CODPARAM;  
CODPARAM= RECORD  
    CODE: INTEGER;  
    SUIV: TYPARAM  
END;
```

La table des symboles est constituée de deux parties:

- a) la table des identificateurs,
- b) la table des étiquettes.

On peut adjoindre à la table des symboles, une famille de tables permettant de mémoriser temporairement des éléments d'interprétation spéciaux comme: la liste des dimensions d'un tableau, les identificateurs de liste locale, ...etc.

Une partie de la table des symboles est de type statique, l'autre est de type dynamique.

L'élément de base composant la table des symboles prend deux formes représentées par deux types Pascal:

```
TYPE TABLESYMB = ARRAY (1..TMAX) OF RECORD  
    NOM: STRING(10);  
    NOMCODE: STRING(MAXID);  
    TIP: TYP;  
    DIM: INTEGER;  
    ETAT: STATU;  
    LPARAM: TYPARAM  
END;
```

Le second type utilisé:

```
TYPE TABLETIQ = ARRAY (1..TMAX) OF RECORD  
    RVAL: integer;  
RNETIQ: integer;  
RDECLAR: BOOLEAN;  
RAFFECT: BOOLEAN  
END;
```

1) la table des identificateurs:  
-----

Elle est du type TABLESYMB et comporte trois sections:

a) La table des algorithmes, fonctions et algorithmes principal, cette table notée TAP évolue au cours de l'analyse de façon croissante à chaque déclaration d'un nouvel algorithme. Elle est permanente pendant toute l'interprétation.

b) La table des variables globales du programme, notée TVGF, est remplie lors des déclarations globales, reste permanente pendant l'interprétation, mais n'évolue pas.

c) La table des variables locales et paramètres d'un algorithme, notée TVA, cette table est réinitialisée à la fin de l'analyse d'un algorithme.

La section TAP de la table des symboles, est remplie par la procédure ALGODEC correspondant au paragraphe syntaxique DECLARE de la grammaire, cette procédure analyse et interprète la déclaration d'un algorithme.

La section TVGF est remplie par la procédure EXTERDEC correspondant au paragraphe syntaxique DECLAREXTERNE de la grammaire, cette procédure et celles qui lui sont associées, traitent les déclarations externes (étiquettes et variables globales), si le programme en possède.

La section TVA est remplie par les procédures DECLARPARAM incluse dans ALGODEC, traitant les paramètres formels, et EXTERDEC avec un aiguillage lui permettant de savoir qu'il s'agit du paragraphe syntaxique des déclarations internes.

Un certain nombre d'aiguillages et de repères dans l'avancement de l'interprétation, sont matérialisés par les mots d'ETATS SEMANTIQUES DE NIVEAU, ce sont des mots de type SEMANTIK, déjà défini, le compilateur peut donc se trouver à l'un des six niveaux suivants:

- 1) DECLARALG : niveau de déclaration d'un algorithme.
- 2) DECLAREXT : niveau des déclarations globales.
- 3) DECLARINT : niveau des déclarations locales.
- 4) DECLARETIQ : niveau des déclarations d'étiquettes.
- 5) DECLARPAR : niveau de déclaration des paramètres.
- 6) UTILISATION: niveau des instructions de PALDES.



En PALDES, on sait que tous les identificateurs sont pourvus d'un type, comme pour les équivalents symboliques, chaque type est implanté dans le compilateur selon un "type symbolique", associé à un équivalent symbolique du type décrit dans le programme source PALDES, par exemple:

dans un programme PALDES, on pourra avoir la déclaration suivante:

```
local logique TEST ;
```

- Le mot 'logique' est reconnu dans le tableau MOT(16) avec l'indice 16.

- L'équivalent symbolique de 'logique', se trouve dans MSYM(16), c'est 'logik', le compilateur reconnaît en ce mot, un type PALDES, donc il lui associe un type symbolique noté 'varlogic'.

Remarque:

---

certains types symboliques, peuvent ne pas correspondre à un mot PALDES, mais plutôt à une déclaration implicite de type comme dans un algorithme.

En effet, on voit ici tout l'intérêt des mots d'états sémantiques de niveau, qui, pour ceux d'entre eux qui sont rattachés aux algorithmes, sont des véritables prédéclarations de déclarations implicites, car c'est la position et la syntaxe qui permettent de différencier un algorithme d'une fonction, d'un algorithme principal, ...etc.

Detail des types symboliques du compilateur:

---

ALGORITHME: pour un algorithme procédure.  
ALGOPRINC : pour l'algorithme principal.  
FONCTION : pour un algorithme fonction.  
FCTSYS : pour un algorithme système.  
PARAM : pour un paramètre formel.  
SORTIFCT : pour l'identificateur de résultat d'une fonction.  
TABLEAUENT: pour un tableau d'entiers machine.  
VARCAR : pour une variable de type caractère.  
VARCHAIN : pour une variable de type chaîne.  
VARENT : pour une variable de type entier machine.  
VARLISTE : pour une variable de type liste.  
VARLOGIC : pour une variable de type logique.  
VARPOL : pour une variable de type polynôme.  
VARPREC : pour une variable en précision infinie.

Tous ces types sont implantés en PALDES. Il faut noter que les deux derniers type VARPOL et VARPREC, ne font pas partie du système SYFAC de base, mais sont des évolutions du système,

>>> modules du compilateur PALDES <<<<

par adjonction d'algorithmes permettant de manipuler ces types.

Indépendamment des six états sémantiques de niveaux, le compilateur possède cinq états sémantiques de statut.

Ces états sont en fait un raffinement des niveaux précédents, ils correspondent aux différentes étapes possibles dans les déclarations internes (locales...) ou dans les déclarations externes (globales,...).

- 1) SGLOBAL : les identificateurs sous ce statut, sont accessibles par tous les algorithmes.
- 2) SLOCAL : les identificateurs sous ce statut ne sont accessibles que dans l'algorithme où ils ont été déclarés.
- 3) SSAFE : les listes de base de PALDES et donc tous les types en dérivant par extension (entier infini, polynôme, ...), sont protégés, lorsqu'ils sont paramètres, d'un éventuel effet de bord dont nous parlerons plus loin.
- 4) SINTRINSIC: les objets ayant ce statut restent protégés et constants tout au long de l'exécution du programme, ils sont protégés de l'écrasement de leur contenu initial.
- 5) SBRANCH : le nombre reconnu actuellement doit être interprété comme une étiquette.

Remarques sur les statuts SINTRINSIC et SSAFE:

-----  
Statut SINTRINSIC:

=====

La protection des objets de statut SINTRINSIC se fait dans un programme PALDES en les déclarant au début, dans les parties globales ou locales comme INTRINSEQ. On voit donc que ces objets peuvent être locaux et restent constants dans l'algorithme où ils sont déclarés, ils sont désactivés à la fin de l'algorithme.

La protection effective a lieu lors de l'interprétation:

c'est à dire qu'une vérification statique aura lieu, afin de voir si l'identificateur de la partie gauche d'une affectation n'est pas SINTRINSI, ou qu'un paramètre effectif de sortie d'un algorithme n'est pas SINTRINSIC.

Statut SSAFE:

=====

Ce statut correspond à une protection statique des listes dans la programmation en PALDES, afin de les protéger d'un effet de bord spécifique du langage. On utilise dans le programme le mot reserve PERMANENT, son équivalent

symbolique étant SAFE.

L'effet de bord pour les listes PALDES est le suivant:

Si le passage par paramètre d'entrée protège automatiquement les autres objets de base de PALDES contre une réécriture, il ne protège pas les listes d'une modification d'un ou plusieurs de ses éléments, seules la racine et la findeliste sont protégées.

Ceci est dû au fait que les listes PALDES sont représentées dans la mémoire dynamique du Pascal qui n'assure pas la protection des objets "pointeurs".

Normalement, un paramètre d'entrée ne doit pas être modifié dans un algorithme, si le cas se produit cela correspond à une erreur logique de programmation, le programmeur pourra prendre une "assurance" contre de telles erreurs en déclarant ces objets PERMANENT, au prix bien sûr d'un coût supplémentaire en temps d'exécution et en place mémoire.

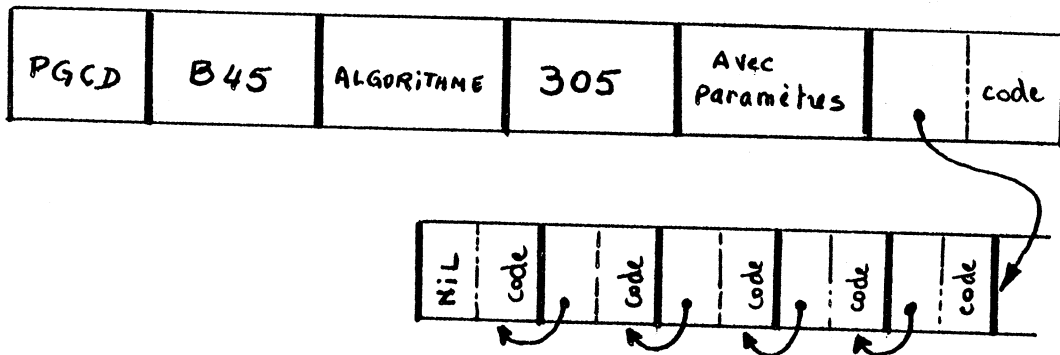
Ce sont d'ailleurs ces deux raisons qui ont motivées la non protection automatique des listes par le système de base.

Signalons qu'une vérification statique à la compilation aurait pu être faite, comme pour les objets INTRINSEQ. Cette vérification étant d'autant plus facile que les listes ne sont manipulées que comme paramètres dans des appels d'algorithmes. Pour ne pas alourdir les temps de compilation on a préféré laisser la possibilité de ne pas faire cette vérification, en proposant en contrepartie une déclaration de protection en l'occurrence PERMANENT.

Nous verrons dans le module GENERATION comment la traduction est faite en Pascal.

### CONSTITUTION D'UN ENREGISTREMENT DE LA TABLE DES IDENTIFICATEURS:

exemple:  
=====



- La première cellule contient l'identificateur PALDES.

>>>> modules du compilateur PALDES <<<<

- La seconde cellule contient l'identificateur interprété en Pascal (code).
- La troisième cellule contient le type symbolique de l'identificateur.
- La quatrième cellule contient un code entier.
- La cinquième cellule contient le statut de l'identificateur.
- La sixième cellule contient un pointeur vers la pile des paramètres.

Le nombre entier contenu dans la cellule quatre est un code servant à différentes utilisations:

- a) pour un algorithme fonction, il indique le nombre de paramètres (entrée seulement),
- b) pour un algorithme procédure, il indique le nombre de paramètres d'entrée, et le nombre de paramètres de sortie,
- c) pour un tableau il indique le nombre d'indices.

Dans tous les autres cas il est mis à 0.

Pour améliorer la lisibilité d'un programme PALDES, les identificateurs externes (global), ne peuvent être redéfinis ni comme paramètres formels, ni comme variables locales d'un algorithme quelconque. Ce choix évite la "name précedence" du Pascal, où un identificateur du programme principal peut être redéfini comme interne à une procédure, grâce à un "masquage" temporaire de cet identificateur.

Par contre, pour éviter la lourdeur du choix d'un trop grand nombre d'identificateurs distincts, les identificateurs des paramètres formels et ceux des variables locales d'un algorithme, lui sont propres et, peuvent être réutilisés dans un autre algorithme:

```
global C,D;
.....

Algor01 (ENT1,ENT2;SORTIE=liste)
  local X,Y,I;
  local carac A,B;

(1) .....

#

t:=Algor (ENT1;ENT2,ENT3=liste)
```

```
local liste X,J,I;  
local A,B,C;
```

(O) .....

#

Dans cet exemple il n'y aura qu'une seule erreur de décelée:

- La variable C est déclarée deux fois (global et local).

Les identificateurs ENT1 et ENT2 sont utilisés deux fois, la première en type entier, la deuxième en type entier et liste. Il en va de même pour les identifiacteurs X,Y,I,A,B qui peuvent être réutilisés autant de fois que l'on veut, avec le même type, ou sous un autre type.

#### CONCLUSION:

---

La table des identificateurs sert à plusieurs utilisations:

- Reconnaissance des doubles déclarations d'identificateurs..
- Reconnaissance d'identificateurs non déclarés.
- Vérification de la cohérence des types.
- Vérification de la cohérence entre le nombre de paramètres formels et le nombre de paramètres effectifs, en entrée et en sortie.
- Communiquer des informations au module de génération.

#### 2) la table des étiquettes:

---

Elle est du type TABLESYMB et comporte deux sections:

a) La table des étiquettes de l'algorithme principal, cette table notée TPETIK, est remplie lors des déclarations externes, et reste permanente, comme la TVGF, et n'évolue pas pendant toute l'interprétation. Elle est remplie par la procédure PBRANCH qui correspond au paragraphe syntaxique de déclaration d'étiquette dans EXTERDEC, au niveau sémantique DECLAREXT.

b) La table des étiquettes d'un algorithme non principal, cette table notée TETIQ, est remplie lors des déclarations internes, et comme la TVA elle est réinitialisée à la fin de l'analyse de l'algorithme, elle est remplie par la même procédure PBRANCH, mais dans la partie correspondant au niveau sémantique DECLARINT.

Remarque sur le niveau sémantique DECLARETIQ:

---

Il n'y a pas qu'un seul indicateur de niveau, le type SEMANTIK sert à plusieurs indicateurs; dans le cas de la table des étiquettes, il y a un indicateur général qui donne le niveau DECLAREXT ou DECLARINT (l'un excluant l'autre), mais il y a un autre indicateur qui signale la présence d'une déclaration d'étiquette par le fait qu'il vaut DECLARETIQ, sinon la valeur du niveau actuel.

CONSTITUTION D'UN ENREGISTREMENT DE LA TABLE DES ETIQUETTES:

---

45	10	oui	NON
----	----	-----	-----

- La première cellule contient l'étiquette PALDES.
- La deuxième cellule contient le codage en étiquette Pascal.
- La troisième cellule permet la reconnaissance de l'apparition de cette étiquette comme numéro de paragraphe dans le programme PALDES.
- La quatrième cellule permet de reconnaître si le branchement vers le paragraphe qui est référencé par cette étiquette, a été fait au moins une fois par une instruction ALLERA.

La table des étiquettes sert à:

- Voir si une étiquette à laquelle renvoi un ALLERA existe comme titre de paragraphe, exemple:  
ALLERA 45; 45 n'ayant pas été déclarée, mais existant comme numéro de paragraphe (45).
- Voir si une étiquette a été déclarée, mais n'existe pas comme numéro de paragraphe.
- Voir si une étiquette est inutile; dans les cas où 45 est déclarée, elle existe comme paragraphe mais n'est assigné par aucun ALLERA, ou dans le cas: elle est déclarée mais n'existe pas comme paragraphe et n'est assignée par aucun ALLERA.

Pour faire toutes ces vérifications un module d'analyse de la table des étiquettes est appelé à la fin de chaque algorithme interprété et fournit, s'il y a lieu un diagnostic, il en est de même dans l'algorithme principal, ce module travaillant sur les deux sections de la table des étiquettes.

CONCLUSION SUR LA GESTION DE LA TABLE DES SYMBOLES;  
=====

Extension de la table:  
-----

Si dans la table des symboles, un dépassement de capacité se produisait, ce qui arrive lorsqu'il y a plus de déclarations qu'elles ne peut en contenir (dans l'implantation sur le micro-ordinateur Apple II+, 40 variables globales, 40 paramètres et variables locales par algorithme), le système a la possibilité de proposer pendant la compilation une extension d'une partie de la table. En cours d'interprétation, le compilateur signale un overflow au prochain symbole, si l'extension n'est pas demandée l'insertion d'un autre symbole dans la table provoquera un abort de la compilation. Sinon la compilation continuera avec 50% de plus pour la (ou les) table(s) en question. Cette option d'extension ne provoque pas un ralentissement de la compilation.

Structure et ordre:  
-----

La structure générale de la table des symboles, avec ses composantes statiques et dynamiques, est en organisation triée sur les identificateurs PALDES, par ordre lexicographique, et sur les numéros croissants pour les numéros de paragraphes étiquettes.

Cette organisation permet une recherche 'dichotomique' pour un élément quelconque de la table. Le rangement par insertion d'un nouvel élément déclenchant automatiquement la procédure de tri.

La procédure de tri choisie est du type dichotomique récursif (C.A.HOARE). Ces choix comme nombre d'autres ont été faits pour avoir des temps de réponse faibles vis à vis du cycle de fonctionnement du microprocesseur, et surtout du temps d'exécution de l'instruction Pascal UCSD, qui est généralement interprétée en P-code.  
procédure de tri de l'élément de base de la table des symboles:

PROCEDURE TRI (VAR TABLE: TABLESYMB; INF, MAXI: INTEGER)

VAR I, J, D1: INTEGER;  
KOD, X, W1: STRING;  
S1: STATU;  
T1: TYP;

```

BEGIN
I:=INF;
J:=MAXI;
IF J>I THEN
BEGIN
X:=TABLE((I+J) DIV 2) NOM;
REPEAT
WHILE TABLE(I) NOM<X DO I:=I+1;
WHILE TABLE(J) NOM>X DO J:=J-1;
IF I<=J THEN
BEGIN
WITH TABLE(I) DO
BEGIN
N1:=NOM;
KOD:=NOMCODE;
T1:=TIP;
D1:=DIM;
S1:=ETAT;
END;
TABLE(I) NOM:=TABLE(J) NOM;
TABLE(I) NOMCODE:=TABLE(J) NOMCODE;
TABLE(I) TIP:=TABLE(J) TIP;
TABLE(I) DIM:=TABLE(J) DIM;
TABLE(I) ETAT:=TABLE(J) ETAT;
WITH TABLE(J) DO
BEGIN
NOM:=N1;
NOMCODE:=KOD;
TIP:=T1;
DIM:=D1;
ETAT:=S1;
END;
J:=J-1;
I:=I+1;
END
UNTIL I>J;
IF INF<J THEN TRI(TABLE, INF, J);
IF MAXI>I THEN TRI(TABLE, I, MAXI);
END
END;

```



>>>> modules du compilateur PALDES <<<<

#### MODULE DE RECONNAISSANCE:

=====  
Pour décrire pratiquement ce module, il suffit de savoir que les différentes procédures le composant ont été conçues en suivant les diagrammes syntaxiques de la grammaire LL(1) de PALDES donnée dans ce document.

Les ensembles INIT et FOLLOW sont ceux qui sont construits lors de la vérification de la LL(1)-té de la grammaire de PALDES cf <4>.

#### RECUPERATION D'ERREUR DANS L'ANALYSE:

-----  
Signalons une sous-unité importante de module, permettant la liaison avec la table des symboles, notée TABLALDES, qui contient toutes les fonctions d'accès à la table des symboles, et l'unité d'ERROR RECOVERY.

La récupération d'erreur se fait donc dans le module de reconnaissance, elle s'effectue comme dans tous les analyseurs descendants récursifs, par annulation de la branche à partir de l'élément erroné, jusqu'au prochain marqueur syntaxique.

Pour améliorer l'efficacité dans le choix des marqueurs syntaxiques à partir desquels l'analyse reprend, on choisit de préférences des marqueurs situés à des charnières sémantiques:

on prendra dans les ensembles FOLLOW(), les éléments qui correspondent à une demande de génération.

ci - dessous la procédure TEST effectuant la récupération d'erreur:

le type Pascal SYMSET correspond à un ensemble forme sur les équivalents symboliques de SYMBOLE, la variable globale SYM contient le symbole en cours fourni par SYMSUIV (cf. ANALEXICO).

```
PROCEDURE TEST(ENS1,ENS2:SYMSET;N1:INTEGER);  
  
BEGIN  
  IF NOT(SYM IN ENS1) THEN  
    BEGIN  
      ERR(N1,'***',NL);  
      ENS1:=ENS1+ENS2;  
      WHILE NOT(SYM IN ENS1) DO SYMSUIV  
    END  
  END;  
END; (* TEST *)
```

Toutefois, l'efficacité dépend beaucoup du type d'erreur commise, et un saupoudrage astucieux d'erreurs permettrait

>>> modules du compilateur PALDES <<<

de faire ignorer par l'analyseur, beaucoup de symboles.

Dans les utilisations actuelles, trois passages au plus ont été nécessaires pour découvrir des erreurs qui se succédaient, et avaient été masquées par les précédentes.

#### MODULE DE DEBUGGING:

=====

Pour la version actuelle, ce module donne le listing complet des références de tous les éléments de la table des symboles au fur et à mesure de son évolution.

Ce listing s'ajoute au listing source, à la fin de l'analyse de chaque algorithme.

On obtient dans l'ordre:

- Un listing de tous les algorithmes systèmes en bibliothèque à ce jour.
- Puis ensuite un listing de la TVGF .
- Un listing de TPETIK.
- Pour chaque algorithme la TVA et la TETIQ.
- Un listing de la TAP et des algorithmes systèmes utilisés par le programme.

Ce module est active conversationnellement, au terminal au début de l'interprétation. Dans le cas contraire seul un listing source apparait

Un exemple de listing produit par ce module, est donné à la fin de ce document.

#### MODULE DECLARATIONS:

=====

Comme le module RECONNAISSANCE, ce module est conçu à partir de la méthode établie pour les paragraphes syntaxiques de la grammaire. Pour la méthodologie de construction se reporter au même exemple que pour le module de reconnaissance.

Il est utile de savoir que ce module doit reconnaître la syntaxe des déclarations, leur hiérarchie, et traduire cette hiérarchie en éléments qui permettront au module de génération de faire une traduction en déclarations Pascal correspondantes.

MODULE DE GENERATION DU CODE:

=====

La génération se fait à des moments bien précis, dépendants de chacun des éléments analysés. Dans tous les cas il y a appel au module de génération, dès qu'un groupe d'éléments a été analysé et reconnu comme valide. La génération n'est pas une traduction "mot à mot" du texte PALDES en Pascal, il existe en mémoire centrale un certain nombre de tables de stockage pour certaines particularités sémantiques.

La génération se fait donc en deux temps, le stockage d'éléments sémantiques préparant la génération définitive sur un fichier séquentiel disque.

Comme cela a déjà été dit, le compilateur est conçu en deux parties logiques distinctes, mais reliées à des "carrefours" sémantiques, ceci permet de déconnecter à tout instant le module de génération par la variable 'ok', et ce dès le début. Et il est donc possible de se servir du compilateur comme d'un analyseur du langage PALDES, ce qui permettra de se familiariser avec la syntaxe de PALDES et l'écriture de programmes en PALDES avec un minimum de perte de temps. Lors de l'apparition de la première erreur de syntaxe le module de génération est déconnecté, mais l'analyse se poursuit de manière plus rapide.

Nous allons voir sur quelques exemples d'interprétation comment fonctionne ce module.

Le compilateur utilise, pour le code généré, dans la mesure du possible les facilités du Pascal généré:

les variables globales, locales, les types entier machine, caractères, tableau, logique, les fonctions, les algorithmes, l'algorithme principal, la récursivité, sont des éléments du langage PALDES qui ont des interprétations très simples en Pascal.

Les listes (d'entiers machine) ont déjà été examinées dans le détail, lors du chapitre sur l'espace mémoire et les listes. Le module de génération produira pour les éléments de type liste, et les types associés, un code plus important puisque pour les faire fonctionner, il faudra faire appel à certains algorithmes du système PALDES de gestion de liste.

INTERPRETATION ET CODE GENERE D'ELEMENTS DE PALDES:

1) Les étiquettes sont traduites en LABEL Pascal.

2) Les variables globales sont des VAR du programme principal Pascal nommé INTERPRET.

>>> modules du compilateur PALDES <<<

- 3) Les éléments intrinseq sont des CONST Pascal.
- 4) Les entiers sont des INTEGER.
- 5) Les logique sont des BOOLEAN.
- 6) Les carac sont des CHAR.
- 7) Les chaines sont des STRING(80).
- 8) Les tableau sont des ARRAY(..) OF INTEGER.
- 9) Les algorithmes sont traduits par des procédures Pascal.
- 10) Les variables locales sont des VAR d'une procédure.

ELEMENTS PARTICULIERS:

---

algorithme fonction:  
=====

Y:=FONC1(X1,X2;A=liste)=carac

.....

Y:= < expression >;

.....

#

Cet algorithme fonction PALDES, sera interprété en Pascal par:

```
FUNCTION F56(F57,F58:integer;F59,FDLF59:liste):char;  
BEGIN
```

.....

F56:= < expression >;

.....

END;

A l'appel de la fonction dans une expression, on traduira de la même façon en Pascal, on remarquera que l'identificateur Y est interprété comme l'identificateur de la fonction, lors d'une affectation.

INTERPRETATION DES INSTRUCTIONS PALDES:

---

Les parenthèse d'énoncé { et } sont traduites par BEGIN et END en Pascal.

>>> modules du compilateur FALDES <<<<

Les affectations sont traduites telles quelles.

Les appels d'algorithmes sont remplacés par les appels des procédures correspondantes avec les paramètres effectifs correspondants.

instruction "pour" FALDES:

---

L'interprétation suit la définition donnée dans la description des instructions FALDES:

pour X=T1,T2,...,T3 faire E1 ;

Est interprété en Pascal par:

```
X :=T1;
Y0:=T2-X;
Z :=T3;
Y1:=SIGN(Y0);
WHILE Y1*X <= Y1*Z DO
BEGIN
.
.
.
E1
.
.
.
X:=X+Y0
END;
```

Cette traduction nécessite l'introduction de variables de travail internes, comme Y0, Z, Y1.

Pour pouvoir gérer correctement une suite d'imbrications de boucles "pour", le système dispose d'une pile de niveaux d'imbrications (20 niveaux autorisés dans l'implantation ). Cette pile est simulée par trois tableaux dont l'indice est le niveau d'imbrication en cours. Le compilateur possède un mot d'état de niveau, qui communiquant sa valeur au module de génération permet de générer la valeur correcte du niveau au système.

Ce sont les variables internes Y0, Z, et Y1 qui deviennent des tableaux.

La pile contient à chaque niveau les différentes valeurs des variables internes de ce niveau; le niveau augmente de 1 à chaque début d'analyse d'une nouvelle instruction "pour" imbriquée, et diminué de 1 à la fin de l'analyse du "pour", le module de génération engendre donc des éléments de tableaux tout au long de l'interprétation:

Exemple:

---

>>> modules du compilateur PALDES <<<<

```
pour X1=T1,A1,...,B1 faire
pour X2=T2,A2,...,B2 faire
( A:=X1;
  B:=X2 )
```

Cette imbrication est traduite et interprétée:

```
  X1:=T1;
YO(1):=A1-X1;
  Z(1):=B1;
Y1(1):=SIGN(YO(1));
WHILE Y1(1) <= Y1(1)*Z(1) DO
BEGIN
  X2:=T2;
YO(2):=A2-X2;
  Z(2):=B2;
Y1(2):=SIGN(YO(2));
WHILE Y1(2)*X2 <= Y1(2)*Z(2) DO
BEGIN
  A :=X1;
  B :=X2;
  X2:=X2+YO(2)
END;
  X1:=X1+YO(1)
END;
```

Il faudra faire attention de ne pas imbriquer trop de boucle "pour", car en Pascal le code généré risque de saturer rapidement le segment de code du compilateur Pascal. Ceci ne représente d'ailleurs pas un problème, car il est toujours possible de remplacer une instruction de boucle par une autre. Dans l'implantation la limite a été fixée à 19 boucles imbriquées au plus, ce qui dépasse largement les cas courants de programmes.

INSTRUCTIONS TRADUITES IMMEDIATEMENT:

tantque.....faire:

Traduite sans difficulté par WHILE.....DO.

répéter.....jusque:

Traduite par REPEAT.....UNTIL.

si.....alors.....sinon:

Traduite par IF....THEN....ELSE.

entrer et entrer\$:

Traduite par READLN(input, ) et READ(input, ).

imprimer et imprimer#:

-----  
Traduite par WRITELN(f, ) et WRITE(f, ), ou f est le fichier de type texte du système SYPAC, associé au périphérique de sortie. Dans l'implantation c'est soit le terminal écran, soit l'imprimante.

allera:

-----  
Traduite par GOTO .

selon....{...}:

-----  
Traduite par CASE....OF....END.

assertion "....":< condition >:

-----  
Le module de génération engendre un code spécial, qui est pour l'instant une procédure notée ASSERT, ayant deux paramètres d'entrée par valeur. L'un d'eux est le commentaire référant l'assertion, l'autre une variable BOOLEAN, ayant la valeur de vérité de la condition. Cette procédure est chargée d'envoyer un message de détection d'incohérence entre instructions et assertions. Elle joue un rôle semblable à l'instruction ASSERT de l'ALGOL W ou de l'ADA.

Exemple d'éléments générés par le module génération:

```
assertion " n est pair, ici":(FAIR(n));
```

Sera traduit en Pascal:

```
un appel à la procédure ASSERT,  
ASSERT('n est pair, ici',A98(N));
```

(en supposant que A98 soit le codage de l'algorithme FAIR).

description de la procédure ASSERT:

=====

```
PROCEDURE ASSERT(TEXTE:string(80);AFFIRM:boolean);  
BEGIN  
  IF NOT AFFIRM THEN  
  BEGIN  
    WRITELN(F,'ASSERTION: ',TEXTE,' FAUSSE)  
    (* DANS UNE VERSION ULTERIEURE ON INCLUERA ICI UN ETAT  
    DU SYSTEME PERMETTANT LE DEBUGGING A L'EXECUTION *)  
  END;
```

Le débbugging n'a pas été inclu, car cela oblige à élaborer

>>> modules du compilateur PALDES <<<<

tout un système de mémorisation de la gestion de l'exécution (nom d'algorithme, état des variables globales, locales, ...etc), qui aurait alourdi considérablement sur le micro-ordinateur le code généré, la place mémoire du système, et les temps d'exécution. Ce pourra être une option à envisager dans un développement ultérieur.

Il est tout à fait possible de prévoir une extension simple de l'assertion en debugging, en faisant imprimer au moment de l'incohérence une liste de variables, qui aurait été indiquée dans l'instruction (un peu comme le TRACE du FORTRAN). Au lieu de faire interpréter l'assertion par une procédure il faut alors générer le code correspondant à chaque assertion, car le Pascal ne permet pas d'avoir des procédures à nombre de paramètres variables. On générera par exemple des instructions d'écriture des variables à imprimer:

```
assertion "n est pair,dans la boucle":(PAIR(N)),n,i;
```

où i est l'indice de la boucle dans laquelle l'assertion est placée.

Le module de génération pourrait engendrer à partir de la table des symboles le code suivant:

```
ASSERT('N EST PAIR DANS LA BOUCLE',PAIR(N));  
WRITELN(F,'N:=',N);  
WRITELN(F,'I:=',I);
```

Ceci bien que simple à réaliser n'a pas été implanté par soucis de gain de place dans le segment de code du Pascal sur le micro-ordinateur de l'implantation.



## CHAPITRE IV

La gestion de la mémoire des listes se fait dans la mémoire dynamique, par les procédures NEW et NEWALDES, pour ce qui est de l'allocation mémoire.

La restitution de mémoire se fait dans une pile d'adresses située dans la mémoire dynamique et notée ESPALIB.

La désallocation mémoire suit immédiatement toute restitution, par empilement de l'adresse de cette cellule libérée, dans ESPALIB.

=====

>>>> mémoire dynamique de liste dans SYPAC <<<<

\*\*\* ORGANISATION GENERALE DE L'ESPACE MEMOIRE \*\*\*  
DES LISTES EN PALDES

---

En ALDES le TAS dans lequel tous les éléments sont représentés, est géré par un algorithme de garbage collector, c'est la différence fondamentale avec le système SYPAC.

REPRESENTATIONS CHOISIES:

---

- L'espace mémoire est dynamique avec allocation homogène.
- Les blocs de mémoire sont appelés des cellules et sont composés de trois éléments d'information.
- L'organisation générale est celle de liste chaînée.

Il y a deux zones de mémoire utilisées:

---

La première est l'ESPACE D'EXECUTION, zone dans laquelle se font tous les calculs sur les listes. C'est dans cette zone que se trouvent aussi bien les listes GLOBALES du programme, que les listes LOCALES d'un algorithme.

La seconde zone est l'ESPACE LIBRE, zone servant de réservoir de cellules mémoires prêtes à être utilisées par le programme.

Il y a un échange constant entre ces deux zones de mémoire.

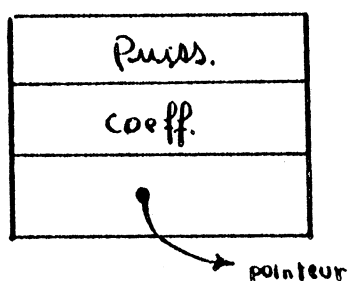
Dans le sens ESPACE D'EXECUTION vers ESPACE LIBRE, il s'agit de RESTITUTION de cellules ne servant plus.

Dans le sens inverse il s'agit d'ALLOCATION de cellules pour un usage temporaire.

Pour les deux zones, dans l'implantation physique en Pascal il n'y a pas de distinction, elles sont toutes deux gérées dans la MEMOIRE DYNAMIQUE du Pascal.

On sait qu'en Pascal la mémoire dynamique est accessible aux structures munies de pointeurs. C'est le cas de l'espace d'exécution, et de l'espace libre dans SYPAC en effet:

la cellule de base de SYPAC a la forme suivante:



>>>> mémoire dynamique de liste dans SYPAC <<<<

Le système de gestion de listes de SYPAC est simulé en Pascal par une famille de procédures assurant une gestion par pile de l'espace dynamique du Pascal.

La structure de liste de SYPAC est interprétée en Pascal de la manière suivante:

```
LIST=RECORD
    PUISS,COEFF: INTEGER;
    SUIV:LISTE
END;
```

Ceci correspond à une cellule mémoire de SYPAC.

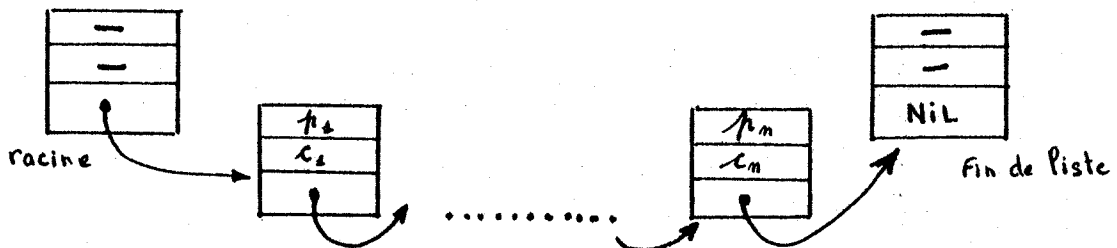
On voit donc que chacun des éléments d'information PUISS et COEFF, contient l'information utilisable de la cellule, l'élément SUIV servant à pointer vers la cellule suivante.

Nous aurons donc une liste de SYPAC en créant le type LISTE en Pascal:

```
LISTE = ^LIST
```

Donc chaque liste de SYPAC est représentée par une liste linéaire chaînée; pour simplifier et accélérer certains algorithmes comme la recherche d'un élément où l'insertion la liste est encadrée comme dans N.Wirth (2) par:

une cellule de tête contenant sa racine et une cellule de fin de liste que WIRTH appelle sentinelle que nous noterons FDL.



Cette conception de l'espace mémoire utilise au maximum les possibilités de l'espace dynamique du Pascal.

Le langage Pascal n'assure quelque soit sa version, qu'une faible part de la gestion de l'espace dynamique. Soit le système possède les procédures DISPOSE et NEW, soit il possède les procédures RELEASE et NEW, comme dans les systèmes UCSD. Certains Pascal possèdent en plus de NEW et DIPOSE, une autre procédure de désallocation, il s'agit de la procédure RESET.

EXAMEN SOMMAIRE DES FONCTIONS DE CHACUNE DE CES PROCEDURES:

---

\* La procédure NEW est chargée de l'allocation de blocs de mémoire du type défini par le paramètre de NEW, sa fonction essentielle est de fournir un nouveau pointeur vers un bloc libre.

\* La procédure DISPOSE effectue une restitution du bloc pointé par la variable qu'elle contient.

\* La procédure RELEASE fonctionne en Pascal UCSD, de pair avec la procédure MARK. Le rôle de MARK est de faire au moment de son appel un marquage de la pile de mémoire dynamique, et de sauvegarder cette marque dans une variable.

Lorsque l'on veut restituer de l'espace mémoire, on fait appel à la procédure RELEASE, avec comme paramètre une variable qui a déjà été marquée par MARK. Il se produit alors une désallocation et une restitution de toute la mémoire dynamique comprise entre le sommet actuel de la pile de mémoire et la marque.

\* La procédure RESET est analogue à la procédure RELEASE, mais moins rudimentaire, puisqu'il est inutile de lui fournir une variable préalablement marquée.

Il suffit de lui communiquer un pointeur de bloc précédemment alloué par NEW, et tous les blocs de quelque nature, qui ont été alloués depuis, seront désalloués et restitués.

On voit donc que, dans tous les cas l'allocation mémoire se fait par la procédure NEW(A), qui va puiser dans la mémoire dynamique libre de quoi constituer un bloc d'enregistrements pointés par A.

Par contre la désallocation et la restitution mémoire se font de façons différentes, les systèmes UCSD disponibles sur la plupart des micro-ordinateurs restituant tout sans discrimination, les autres systèmes Pascal pouvant restituer une cellule de base voir (35), (36) et (37).

Pour tous les types de restitution, il y a un risque de mauvaise utilisation de blocs de mémoire déjà restitués. En effet la libération d'une variable dynamique se fait sous le contrôle du programme et toute réutilisation de cette variable peut provoquer un BUG (cf problème de la référence folle). Certains compilateurs mettent une variable désallouée à NIL.

Il semble enfin que la gestion de la mémoire dynamique par NEW et DISPOSE combinés, soit coûteuse en temps machine.

Afin d'assurer une gestion protégeant le programmeur de ce genre d'erreur, et afin de ne pas pénaliser le système par des NEW et DISPOSE nombreux, le système de gestion de liste SYPAC

>>>> mémoire dynamique de liste dans SYFAC <<<<

n'utilise que la procédure NEW.

Une famille de procédures systèmes est chargée de la désallocation et de la restitution.

Outre qu'aucun calcul d'adresse n'est fait pour chercher ou insérer une cellule, les désallocations comme nous allons le voir ne nécessitent pas d'algorithme de restitution.

#### RESTITUTION ET DESALLOCATION DE LA MEMOIRE DANS SYFAC:

---

Pour réaliser ce système, il a fallu tenir compte des contraintes suivantes:

- Le système doit fonctionner en Pascal UCSD.
- Le système doit pouvoir être implémenté sur un micro-ordinateur, avec une mémoire dynamique d'exécution de 30 k-octets au plus, pour le Pascal.
- Les temps pour la restitution mémoire ne doivent pas être trop lourds.

Pour essayer de satisfaire à ces impératifs, les choix suivants ont été faits:

- 1) L'ESPACE D'EXECUTION est celui du Pascal, ce qui veut dire que l'on se sert des piles d'exécution dynamique du système Pascal lorsque c'est nécessaire.
- 2) L'ESPACE LIBRE est représenté par une pile LIFO de cellules de liste chaînées notée ESPALIB.

En Pascal on aura:

---

```
VAR ESPALIB:RECORD
      NACT,NALL,NSTO:INTEGER;
      TETE:LISTE
END;
```

Les variables NACT,NALL,NSTO sont des paramètres statistiques sur le nombre de cellules disponibles, allouées, et en stockage cumulé de ESPALIB.

3) On crée une nouvelle procédure système d'allocation de mémoire notée NEWALDES, dont le rôle est d'allouer une cellule de mémoire (contenant 3 informations).

4) La désallocation et la restitution sont simultanées, puisqu'une cellule désallouée est empilée sur ESPALIB en tête, avec possibilité de l'allouer immédiatement avec NEWALDES par dépilement de ESPALIB.

>>> mémoire dynamique de liste dans SYPAC <<<<

Le système de liste ALDES/81 comporte deux procédures de restitution mémoire:

- a) la procédure RESTITUE, qui désalloue une cellule mémoire quelconque d'une liste:

```
PROCEDURE RESTITUE(VAR L:LISTE);
BEGIN
  IF L=NIL THEN
    ETSYS:=1
  ELSE
    WITH ESPALIB DO
      BEGIN
        L^.SUIV:=TETE;
        TETE:=L;
        NACT:=NACT+1;
        NSTO:=NSTO+1
      END
    END;
END;
```

La variable ETSYS sert de mot d'état système et signale ici que la cellule L est inaccessible, ce qui signifie qu'elle a déjà été désallouée.

- b) La procédure DESALLOUE qui libère une liste entière correspondant à un chainage de cellules. La désallocation se fait en restituant séquentiellement chacune des cellules composant la liste, par empilement sur ESPALIB:

```
PROCEDURE DESALLOUE(VAR L:LISTE);
VAR LOC:LISTE;
BEGIN
  IF L=NIL THEN
    ETSYS:=1
  ELSE
    BEGIN
      LOC:=L;
      WHILE LOC<>NIL DO
        BEGIN
          LOC:=LOC^.SUIV;
          RESTITUE(L);
          L:=LOC
        END;
      L:=NIL
    END
  END;
END;
```

Il y a désallocation automatique dans SYPAC dès que:

- Au moins un lien de chainage a été supprimé.
- Une liste est locale à un algorithme.

L'allocation et la désallocation de mémoire dans SYPAC se font sous deux modes.

Soit automatiquement pour tous les éléments internes au



## >>> memoire dynamique de liste dans SYFAC <<<<

systeme, comme avec les deux procedures RESTITUE et DESALLOUE et leur famille, soit manuellement par programme PALDES, sous controle du systeme SYFAC.

Pour la desallocation programme, il existe des procedures permettant de faire les memes operations dans un programme PALDES.

Ces procedures sont accessibles a n'importe quel algorithme PALDES, comme un algorithme-systeme.

### SITUATION DE LA MEMOIRE DYNAMIQUE PENDANT L'EXECUTION

#### ----- DANS SYFAC -----

#### 1) L'ALLOCATION MEMOIRE: -----

Tantque l'espace memoire dynamique du Pascal dispose de place, toutes les allocations de cellules se font a partir de la procedure NEW.

Dès qu'une desallocation intervient les adresses (pointeurs) de cellules sont empilees dans ESPALIB.

Les allocations de cellules par NEW sont cumulatives, en ce sens que malgré la restitution de cellules desallouees, on continue a prelever les nouvelles cellules par NEW jusqu'a un certain seuil.

Dans l'implémentation en cours, le seuil est atteint lorsqu'il ne reste plus que 1200 octets dans l'espace dynamique, cette valeur a été fixée pour laisser une marge de manoeuvre minimale pour le fonctionnement du systeme Pascal.

Cette valeur peut être modifiée a chaque execution d'un programme.

#### PARTITION DE LA MEMOIRE: =====

En effet, le seuil correspond a une partition de la memoire dynamique du Pascal, l'un des elements de la partition permettant de faire des manipulations de listes, l'autre comportant l'espace d'execution du Pascal.

C'est dans la portion de memoire non reservee aux listes, que se trouveront les piles d'executions des algorithmes pour des objets comme les entiers machine, les tableaux, les caracteres, ...etc.

Le systeme dispose d'un indicateur SYSPASCAL qui lui permet de connaitre l'etat de saturation de la memoire dynamique.



## >>> mémoire dynamique de liste dans SYPAC <<<<

C'est un drapeau qui, dès qu'il est positionné, fait fonctionner toutes les allocations dans l'espace des listes par NEWALDES.

L'overflow mémoire se produisant lorsqu'une demande de NEWALDES dans ESPALIB ne peut être satisfaite.

La conception du système SYPAC permet de pallier à cet incident, dans la limite de la mémoire disponible, en changeant conversationnellement le seuil de la partition mémoire.

En résumé, selon le type de programme, l'utilisateur peut s'il le désire, laisser plus ou moins de place à la partition des listes, un overflow sur les listes, repris en compte par le système SYPAC permettant d'étendre en cours d'exécution la place mémoire.

Par contre un overflow sur l'espace dynamique, par exemple le manque de place pour la quatrième activation récursive d'un tableau local à un algorithme, produit un STACK OVERFLOW généré par le système Pascal, avec une fin d'exécution du programme.

Si le système Pascal ne comporte pas de reprise sur incident, il faut réexécuter le programme, mais non le recompiler, et changer le seuil de partition au début de l'initialisation du système SYPAC.

## 2) INTERPRETATION DES LISTES:

---

Les listes FALDES peuvent avoir trois statuts, LOCAL, GLOBAL, ou PARAMETRE.

Le système SYPAC comporte une procédure d'initialisation de liste, permettant la création de la liste, par sa racine L et sa fin de liste FDL.

Le nom de cette procédure est INITLIST.

### a) les listes globales:

=====

Elles sont implantées définitivement lors de leur déclaration en variables globales.

Leur initialisation est faite au début de l'activation de l'algorithme principal.

Leur désactivation se fait automatiquement à la fin du programme, ou par programme dans un algorithme.

Il est possible de les protéger d'une désallocation intempestive, même programmée, en les déclarant de type PERMANENT.

### b) les listes paramètres:

=====

## >>> mémoire dynamique de liste dans BYPAC <<<

=====

nous ne parlons, dans ce paragraphe, que des listes paramètres de sortie, car pour les listes paramètres d'entrée, un système de protection particulière par une déclaration spécialisée, permet d'assurer leur non modification à l'intérieur d'un algorithme.

Elles sont transmises comme paramètre effectif lors d'un appel d'algorithme dans un autre algorithme.

Pour l'algorithme appelant elles se comportent comme des listes globales, toute modification faite sur la liste dans le corps de l'algorithme appelant est effective.

Même dans le cas d'un algorithme récursif, la liste de sortie est modifiée à chaque appel récursif.

La visibilité de la racine d'une liste (pointeur) est identique à celle des autres types d'objets de PALDES. (cf exemple)

### c) les listes locales:

=====

elles sont initialisées en début d'activation de l'algorithme ou elles sont déclarées, puis désallouées à la fin de l'activation de cet algorithme.

Cette façon de traiter les listes, permet d'assurer leur protection et comportement de variable locale dans un algorithme récursif. Chaque appel voit se créer dans l'espace des listes une nouvelle liste, chaque fin d'appel récursif voit se désallouer la liste du niveau récursif en cours.

La méthode est donc classique, avec ici l'avantage de faire gérer la pile des adresses de listes locales, comme pour les autres objets, par le système d'exécution Pascal.

### 3) EXEMPLE DE LISTES LOCALES ET PARAMETRES:

-----  
algo(A;ENT=liste;LIS=liste)  
local liste LOC;

(1) .....

'partie 1'  
.....

algo(12,ENT;LOC);

'partie 2'

>>> mémoire dynamique de liste dans SYFAC <<<<

```
.....  
#  
  
Cet algorithme récursif sera traduit par le  
préprocesseur en Pascal de la façon suivante:  
  
procédure algo( A:integer; ENT,FDENT:liste; var  
LIS,FDLIS:liste );  
var LOC,FDLOC:liste;  
  
begin  
  INITLIST(LOC,FDLOC);  
  
  'partie 1'  
  .....  
  
  algo(12,ENT,FDENT,LOC,FDLOC);  
  
  'partie 2'  
  .....  
  
  DESALLOUE(LOC)  
end;
```

On voit bien qu'à chaque appel récursif de "algo" le système va créer une liste nouvelle LOC,FDLOC (racine,findeliste).

La liste ENT,FDENT ne subit aucune modification car c'est un paramètre d'entrée.

La liste LOC,FDLOC voit ses différentes activations subir des modifications dans la 'partie 1'.

Ce n'est que la fin du dernier niveau récursif n, qui fait passer la nième création de la liste LOC dans la 'partie 2', pour enfin désallouer au niveau n cette création et passer par dépilement au niveau n-1.

Donc à chaque appel récursif, les nouvelles listes figurent dans la pile d'exécution de la procédure Pascal "algo".

Lors du dépilement de la pile d'exécution d'algo, ce seront les différentes cellules créées, qui seront utilisées dans la suite de l'activation d'algo.

EN CONCLUSION:

=====

Tous les objets manipulés par PALDES à travers le système SYFAC, se comportent comme en Pascal.

>>>> mémoire dynamique de liste dans SYPAC <<<<

ETAT ET TRANSFORMATION DE LA MEMOIRE DANS SYPAC EN PALDES

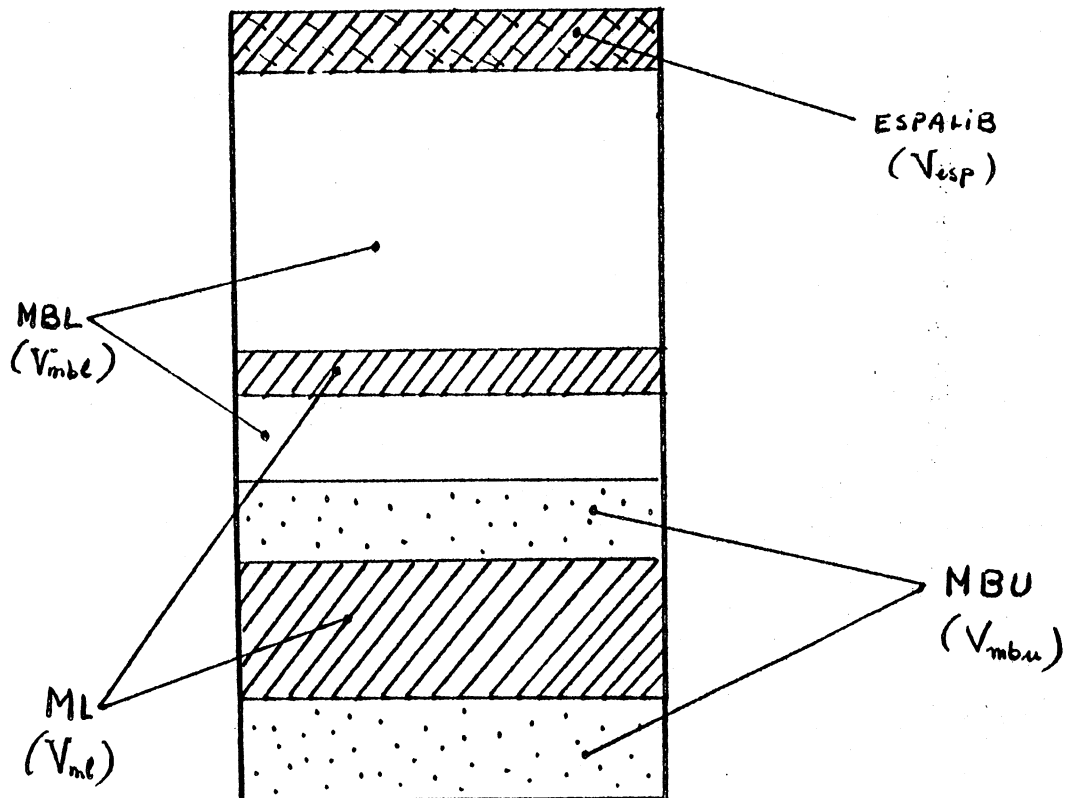
on distingue quatre états de la mémoire sous SYPAC

1°- La Mémoire Banalisée Libre (MBL): c'est la mémoire de base du Pascal, c'est à partir d'elle que se feront les représentations de tous les objets de PALDES. Dans cet état elle est inutilisée et disponible. Soit  $V_{mbl}$  le volume qu'elle occupe en unités mémoires.

2°- La Mémoire Banalisée Utilisée (MBU): c'est la mémoire de base utilisée pour représenter les objets de PALDES autres que les listes, le système d'exécution Pascal établit un échange permanent entre la MBL et la MBU. Soit  $V_{mbu}$  le volume qu'elle occupe en unités mémoires.

3°- La Mémoire de Listes (ML): elle est composée de cellules formées à partir de transformation de mémoire MBL en cellule par la procédure NEW(). Soit  $V_{ml}$  le volume qu'elle occupe en unités mémoires.

4°- La mémoire des listes retituées (ESPALIB): c'est un bloc occupant un volume fixé  $V_{esp}$ , car c'est la tête de liste des cellules desallouées.



>>> mémoire dynamique de liste dans SYPAC <<<<

Rappelons que nous notons  $S$  le seuil de saturation de la mémoire ML, c'est le seuil à partir duquel la transformation de mémoire MBL en mémoire ML ne se fait plus.

Notons  $V_t$  le volume total de la mémoire utilisable sur le calculateur, en Pascal.

Avant l'initialisation de SYPAC on a :

$$V_{mb1} = V_t$$

Puis tout au long de l'exécution on a :

$$V_{mb1} + V_{mbu} + V_{m1} + V_{esp} = V_t$$

La restitution des cellules ML se fait pendant toute l'exécution dans ESPALIB.

Remarquons enfin qu'il y a deux régimes d'utilisation de la mémoire, le régime transitoire, et le régime permanent.

Nous allons visualiser ce qui vient d'être dit sur l'exemple ci-après:

```
global liste L1,L2;  
global I,J,K;  
global carac REP,UN;  
global logique VERIF,TEST;
```

```
ALGO1( )  
local tableau T(100);
```

.....

#

```
ALGO2( )  
local liste L3,L4,L5;
```

.....

#

```
.....  
ALGOn( )  
local liste LA1,LA2;  
local tableau T(100);
```

.....

#

>>>> mémoire dynamique de liste dans SYPAC <<<<

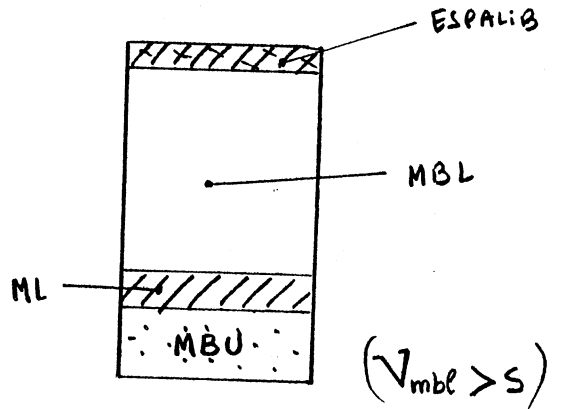
```
PRINCIPAL .  
(0) ALGO1( ) .  
(1) ALGO2( ) .  
.....  
(n-1) ALGOn( )  
##
```

A la page suivante on trouvera un schéma figurant en parallèle, à l'exécution de chaque algorithme, les plages d'occupation des différentes sortes de mémoires dans SYPAC.

>>>> mémoire dynamique de liste dans SYPAC <<<<

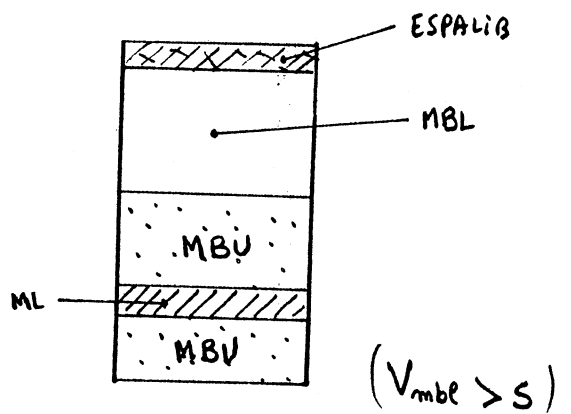
```

global liste L1,L2,L3;
global I,J,K;
global carac RECONS,UN;
global logique VERIF,TEST;
    
```



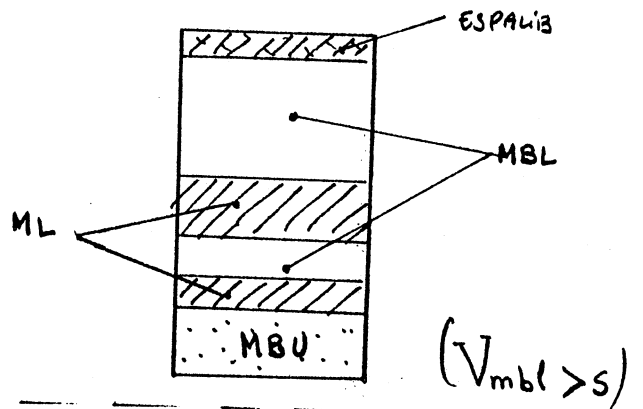
```

ALGO1 ( )
local tableau T(100);
local carac A,E,L;
    
```



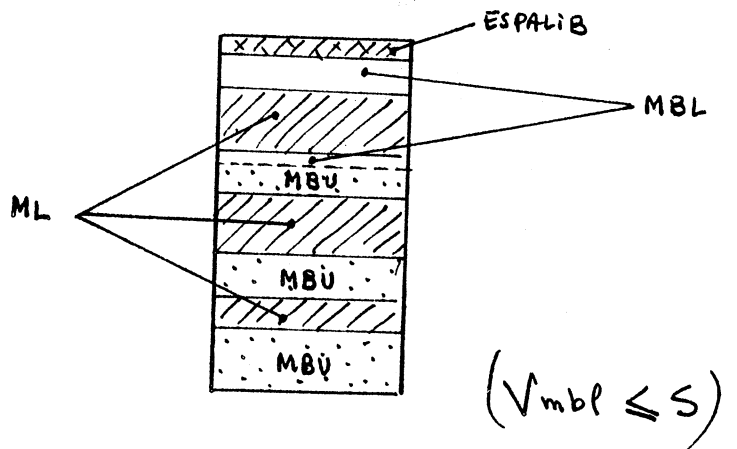
```

ALGO2 ( )
local liste L3,L4,L5;
    
```



```

ALGO n ( )
local liste LA1,LA2;
local tableau T(100);
    
```



```

# PRINCIPAL.
(0) ALGO1 ( )
(1) ALGO2 ( )
...
    
```

On trouvera dans les pages qui suivent, les différentes procédures Pascal et PALDES composant le noyau résident du système SYPAC.

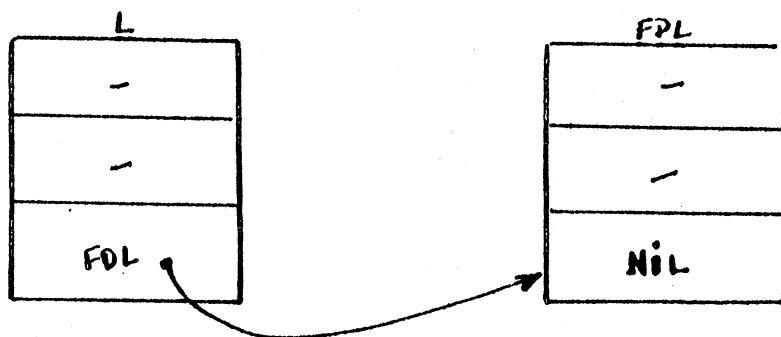
ces procédures concernent la gestion de la mémoire de liste et les opérations élémentaires sur les listes de SYPAC en PALDES.



PROCEDURES INTERNES A SYPAC

## PARTIE DES PROCEDURES INTERNES AU SYSTEMES SYPAC (écrites en Pascal)

- Stand** : cette procédure initialise la pile d'espace libre et met à zéro les variables destinées au comptage statistique des cellules transférées à l'espace libre.
- Restitue** : rend une cellule de la liste L à l'espace libre par empilement dans ESPALIB.
- TESTEM** : sert à positionner l'indicateur du seuil de saturation mémoire Pascal sur Pascal ou surPALDES.
- NEWALDES** : lorsque l'indicateur est surPALDES cette procédure remplace le NEW. Elle dépile ESPALIB et fourni l'adresse de la cellule disponible.
- DESLLOUE** : efface complètement une liste polynômiale L, de sa racine à sa fin de liste, en restituant séquentiellement chacune des cellules de la liste à ESPALIB.
- Après l'exécution de cette procédure, la liste L est en état inaccessible.
- INITLIST** : selon que l'on est sous Pascal ou PALDES pour l'attribution de cellule mémoire, cette procédure sert à initialiser une liste avec sa racine et sa fin de liste



**RECHERANG** : donne le rang c'est à dire le numéro de la cellule dans la liste.

**INSERER** : cette procédure et la suivante, sont le pivot du traitement de liste PALDES. Elle sert à rechercher une cellule par l'élément puissance. Elle sert à insérer une cellule entre deux autres. Elle sert à changer le coefficient pour une puissance donnée. Enfin, elle sert aussi à cumuler un résultat dans un coefficient (c'est à dire additionner une valeur au coefficient d'un monôme de puissance donnée). Elle peut travailler sur les polynômes ordonnés suivant les puissances croissantes ou bien suivant les puissances décroissantes.

**CREATLIST** : permet de recopier une liste déjà existante dans une autre liste, soit avec le même ordre, soit avec l'ordre inverse (puissances croissantes ou décroissantes).

**EFFACE** : efface sur n'importe quel type de liste (croissante ou décroissante) une cellule de puissance donnée et la restitue ESPALIB.

Toutes ces procédures sont écrites en Pascal et figurent dans les pages suivantes.

## SECTION DES PROCEDURES DE SYPAC ACCESSIBLES AUX PROGRAMMES PALDES

- A<sub>1</sub> fourni le premier élément de la liste
- A<sub>2</sub> efface et restitue la première cellule de la liste
- A<sub>3</sub> permet de se servir de CREAT LIST (fonction identique sur les puissances croissantes)
- A<sub>4</sub> permet de se servir de CREAT LIST (fonction identique sur les puissances décroissantes)
- A<sub>5</sub> met ou change le coefficient d'une puissance existante ou efface
- A<sub>6</sub> fourni le coefficient d'une puissance donnée
- A<sub>7</sub> sert à entrer les éléments d'une liste
- A<sub>8</sub> imprime une liste sur le terminal
- A<sub>9</sub> efface une cellule (cas décroissant)
- A<sub>10</sub> permet de changer de terminal en cours d'exécution
- A<sub>11</sub> donne la longueur en nombre d'éléments d'une liste
- A<sub>12</sub> donne le degré du polynôme
- A<sub>13</sub> désalloue une liste non vide
- A<sub>14</sub> permet d'examiner l'état du système SYPAC
- A<sub>15</sub> fourni l'état de la liste selon qu'elle est inaccessible = 2, vide = 0 ou pleine = 1
- A<sub>16</sub> permet d'additionner deux monômes de même puissance
- A<sub>17</sub> efface le contenu d'une liste sans la désallouer en restituant tous ses éléments

La liste est donc réutilisable dans le programme.

VALNUM : donne la valeur numérique d'un polynôme

DERIVE : donne le polynôme dérivée d'un polynôme.

PROCEDURES INTERNES AU SYSTEME SYPAC (écrites en PALDES)

RESTE : donne le reste de la division de deux entiers de  $\mathbb{Z}$

PGCDN : donne la valeur absolue du pgcd de deux entiers de  $\mathbb{Z}$

PPCM : donne la valeur absolue du ppcm de deux entiers de  $\mathbb{Z}$

PUISS : calcule la puissance  $n^{\text{ième}}$  d'entier (méthode de Dijkstra)

AFFECT : réalise une affectation sur les listes

AFFECT (L;R) (R ← L, L est conservée)

EGAL : vérifie l'égalité de deux listes résultats de type logique

MOD : reste de la division par 10 d'un entier de  $\mathbb{Z}$

MAX : le plus grand de deux entiers de  $\mathbb{Z}$

MIN : le plus petit de deux entiers de  $\mathbb{Z}$

PAIR : parité d'un entier de  $\mathbb{Z}$

X:=RETE (I0,J0)

(1) SI J0 ≠ 0 ALORS X:=I0-(I0/J0)\*J0%  
SINON IMPRIMER"ERRERU DIVISION PAR 0"

#

N:=PGCDN(I0,J0)

LOCAL R;

(0) SI I0 < J0 ALORS

%R:=I0;

I0:=J0;

J0:=R

%.

(1) REPETER

R:=RETE(I0,J0) ;

I0:=J0;

J0:=R;

JUSQUE R=0.

(3) SI I0 < 0 ALORS %N:=-I0%

SINON N:=I0

#

N:=PPCMN(I0,J0)

(1) N:=I0\*J0/PGCDN(I0,J0)

#

R:=MOD(X)

(1) R:=X-(X/I0)\*I0

#

T:=MAX(X,Y)

(1) T:=X;

SI X < Y ALORS T:=Y

#

T:=MIN(X,Y)

(1) T:=X;

SI X > Y ALORS T:=Y

#

P:=PAIR(K)=LOGIQUE

(1) SI K=2\*(K/2) ALORS %P:=VRAI%

SINON P:=FAUX

#

```
R:=PUISS(X,N)
LOCAL I,J,K;
```

```
(1) I:=X;
    J:=1;
    K:=N;
    SI K=1 ALORS %R:=X%
    SINON
    % TANTQUE K>0 FAIRE
      % TANTQUE PAIR(K) FAIRE
        % K:=K/2;
        I:=I*I;
        %;
        K:=K-1;
        J:=J*I;
        %;
      R:=J;
    %
```

```
#
```

```
X:=STATU(L=LISTE)
LOCAL Y;
(1) ETAT(L;Y);
    X:=Y
```

```
#
```

```
AFFECT(LINIT=LISTE;LRES=LISTE)
```

```
(0) EFFLS(;LRES);
    COPIE(LINIT;LRES)
```

```
#
```

```
T:=EGAL(L1,L2=LISTE)=LOGIQUE
ETIQUETTE 10;
LOCAL I0,J0;
LOCAL X1,Y1,X2,Y2;
LOCAL LISTE L01,L02;
LOCAL LOGIQUE OK;
```

```
(0) COPIE(L1;L01);
    COPIE(L2;L02);
    OK:=VRAI.
```

```
(1) LONG(L01;I0);
    LONG(L02;J0);
    SI I0<>J0 ALORS
    %OK:=FAUX;
    ALLERA 10
    % .
```

```
(2) TANTQUE OK ET (STATU(L01)=1) FAIRE
    %PREM(L01;X1,Y1);
    PREM(L02;X2,Y2);
    REDUC(;L01);
    REDUC(;L02);
    SI (X1<>X2) OU (Y1<>Y2) ALORS OK:=FAUX
    % .
```

```
(10) T:=OK
```

```
#
```

**CHAPITRE V**



## LES ENTIERS EN PRECISION INFINIE EN PALDES

=====

C'est un sous-type du type liste, ils sont censés représenter un sous-ensemble "très grand de Z".

Les entiers en précision infinie, font partie de l'extension du système PALDES minimal, le but est de disposer de nombres entiers de longueur quelconque et des quatre opérations habituelles sur les entiers:

- Addition.
- Soustraction.
- Multiplication.
- Division euclidienne.

Ceci afin de montrer que PALDES permet de faire des calculs avec des nombres entiers dépassant la capacité limitée de la machine.

Le choix des listes PALDES comme support pratique, permet d'obtenir un système souple de gestion des entiers infinis, et transparent pour l'utilisateur.

La dénomination d'entiers infinis a été choisie pour signaler que les entiers relatifs peuvent être quelconque et ne voient leur taille (nombre de chiffres les composants), limitée que par la place en mémoire centrale. Les contraintes de taille étant donc absolument les mêmes que pour les listes.

Si l'on tient compte des données physiques de l'implémentation, il serait possible de faire une multiplication de deux nombres à 1900 chiffres. Il est bien entendu hors de question de manipuler couramment de tels nombres, les temps de calcul, sur micro-ordinateur 8 bits, seraient hors de portée d'une vie humaine.

Les algorithmes définissant les quatre opérations ont été tout naturellement ceux de Knuth, puisque PALDES comme d'autres systèmes plus complet, a pour principal but de mettre ces algorithmes à la disposition de l'utilisateur sur micro-ordinateur.

### Représentation physique des entiers infinis:

-----

Un entier infini est une liste dont chacune des cellules est particulière:

- L'ordre sur les éléments puissance "PUISS" est l'ordre standard, puissances décroissantes. Ce choix semblable à la représentation écrite d'un nombre, permet une manipulation plus simple de ce nombre.

>>> module précision infinie de SYPAC <<<<

- La première cellule (la plus à gauche), contient le signe du nombre, toutes les autres cellules ont leur élément coefficient "COEFF" strictement positif.

- Tous les éléments COEFF d'une liste représentant un entier infini, sont compris entre 0 et 9:

$$0 < \text{COEFF} \leq 9$$

- La base dans laquelle les nombres sont représentés, peut être quelconque.

Par la suite nous noterons  $\text{ent}(X)$ , le plus grand entier relatif inférieur ou égal à  $X$ , c'est la partie entière mathématique dans  $\mathbb{Z}$ :

$$\text{ent}(X) \leq X < \text{ent}(X)+1$$

Les algorithmes seront écrits en base  $B$ , un entier supérieur ou égal à 2, il est à noter que le système PALDES peut, avec quelques modifications minimes, permettre du calcul en base  $B$ , car il comprend les éléments permettant de faire les opérations en base  $B$  ( $B > 1$ ). Dans l'implémentation actuelle  $B=10$ .

Pour implémenter les algorithmes des quatre opérations en précision infinie, il faut avoir déjà à sa disposition les opérations primitives suivantes:

-a) Addition et soustraction des entiers à un chiffre avec comme résultat un nombre à un chiffre et éventuellement une retenue.

-b) Multiplication de deux entiers à un chiffre avec comme résultat un entier à deux chiffres (celui de gauche pouvant être nul).

-c) Division euclidienne d'un entier à deux chiffres, par un entier à un chiffre, avec un quotient et un reste à un chiffre.

Par construction la machine contient ces opérations, elles ne sont pas toujours accessibles directement selon les langages, PALDES permet de faire implicitement b) et il donne dans c) le quotient entier; des algorithmes élémentaires ont été construits afin d'obtenir le reste dans la division, et la retenue dans l'addition ou la soustraction.

#### CONSTITUTION DU SYSTEME PALDES PRECISION INFINIE:

=====

Le type entier infini est noté XCHIFFRE dans l'implémentation.

Dans la suite nous écrirons "nombre infini" ou entier infini, pour nombre en précision infinie.

Liste des algorithmes le composant:

---

LIRNBR(:A=liste) sert à lire un nombre infini, au terminal et à le stocker sous forme compactée dans la liste A. Le nombre est compacté par épuration de ses zéros (cf. liste).

ECRNBR(A=liste) sert à écrire sur le périphérique de sortie un nombre infini, stocké dans la liste A, sous la forme habituelle, d'écriture sur le papier; de gauche à droite avec ses zéros.

ENTVERLS(N:A=liste) converti un nombre entier machine en un nombre infini, ce dernier est stocké dans la liste A.

SGNLS(A=liste:N) donne le signe d'un nombre infini:  
si  $N=1$  alors  $A>0$   
si  $N=0$  alors  $A=0$   
si  $N=-1$  alors  $A<0$

OPFLS(A=liste:B=liste) prend le nombre infini A et met son opposé dans la liste B, A reste inchangé.

T:=PAIRLS(A=liste)=logique donne la parité d'un nombre infini, cet algorithme fonctionne comme pour les entiers machines, T est VRAI si A est pair, FAUX si A est impair.

ABSLS(A=liste:B=liste) prend la valeur absolue du nombre infini A et la met dans B. A reste inchangé.

COMPLS(A,B=liste:N) compare deux nombres infinis:  
si  $N=1$  alors  $A>B$   
si  $N=0$  alors  $A=B$   
si  $N=-1$  alors  $A<B$

SOUSLS(A,B=liste:C=liste) effectue la soustraction de deux nombres infinis A et B, strictement positifs, et tels que  $A>=B$ ; le résultat se trouve dans C.

ADDLS(A,B=liste:C=liste) effectue l'addition de deux nombres infinis A et B, strictement positifs, le résultat se trouve dans C.

MULTLS(A,B=liste:C=liste) effectue la multiplication de deux nombres infinis A et B, strictement positifs, le résultat est dans C.

DIVCT(A=liste;B:Q=liste;R) divise le nombre infini A strictement positif, par l'entier machine B compris entre 1 et 9, Q est le quotient, R est le reste.

DIVLS(A,B=liste:Q,R=liste) divise le nombre infini A, strictement positif, par le nombre infini B, strictement positif et  $B<A$ , pour mettre le quotient dans Q, et le reste dans R.

ADDITION(A,B=liste:C=liste) additionné dans Z (addition ou

>>>> module précision infime de SYPAC <<<<

soustraction d'entiers signes), les nombres infinis A et B, le resultat est dans C.

MULTIPLICATION(A,B=liste;C=liste) multiplie deux nombres infinis quelconques.

DIVISION(A,B=liste;Q,R=liste) division euclidienne généralisée à Z, Q est le quotient généralisé et R le reste généralisé.

#### DIVISION EUCLIDIENNE GENERALISEE A Z:

---

Soient a et b deux entiers relatifs, l'algorithme de la division euclidienne n'est défini que dans N, on l'étend à Z de la manière suivante, a' et b' dénoteront les valeurs absolues respectives de a et de b:

Lorsque a et b sont strictement positifs, c'est la division euclidienne dans N.

On suppose  $a' \geq b' > 0$ .

on par division euclidienne de a' par b' notons (1) la relation:

$$(1) \quad a' = b' * q + r \text{ et } 0 \leq r < b'$$

on peut réécrire (1) :

si  $(a < 0)$  et  $(b > 0)$  alors  $a' = -a$  et  $b' = b$   
donc: (1)  $\Leftrightarrow -a = b * q + r \Leftrightarrow a = b * (-q) + (-r)$

si  $(a > 0)$  et  $(b < 0)$  alors  $a' = a$  et  $b' = -b$   
donc: (1)  $\Leftrightarrow a = (-b) * q + r \Leftrightarrow a = b * (-q) + r$

si  $(a < 0)$  et  $(b < 0)$  alors  $a' = -a$  et  $b' = -b$   
donc: (1)  $\Leftrightarrow -a = (-b) * q + r \Leftrightarrow a = b * q - r$

L'algorithme de division dans Z consistera à changer les signes de Q et de R, selon les signes de A et de B:

```
SIGNLS(A;SA);
SIGNLS(B;SB);
si (SA<0) et (SB>0) alors ( OPPLS(Q;Q');
                           OPPLS(R;R') );
si (SA>0) et (SB<0) alors OPPLS(Q;Q');
si (SA<0) et (SB<0) alors OPPLS(R;R');
```

#### DESCRIPTION DES ALGORITHMES DE BASE:

---

Les nombres infinis sont supposés être écrits en base B, un nombre infini A est écrit  $a_1a_2a_3a_4\dots a_n$  (nombre a n chiffres) la numérotation se faisant de gauche à droite, les entiers infinis sont essentiellement positifs.

La description des algorithmes est celle donnée par D.E Knuth

>>> module précision infime de SYFAC <<<<

dans (2), on trouvera, aussi une preuve formelle de ces algorithmes dans le même ouvrage. Des algorithmes structurés, équivalents et ne comportant plus de ALLERA sont décrits et implantés en PALDES, on les trouvera à la fin de ce document.

algorithme de comparaison:

COMPABS (A et B entiers infinis, de même nombre de chiffres n, R résultat

A=a1a2...an  
B=b1b2...bn)

(1) j := 1.

(2) si aj < bj alors r := -1; fin (A < B)  
si (aj = bj) et (j = n) alors r := 0; fin (A = B)  
si (aj = bj) et (j < n) alors { j := j + 1;  
ALLERA 2 };  
si aj > bj alors r := 1; fin (A > B)

#

Voyons seulement sur cet exemple comment cet algorithme est structuré et écrit en PALDES de base:

COMPABS(A,B=liste;n:R)

!" A et B sont des entiers de même nombre de chiffres n";  
!" on suppose qu'il existe un algorithme permettant d'obtenir aj et bj";  
!" , on notera cet algorithme CHER(A,j;x), il donne l'élément aj de la liste A dans la variable x."

local aj,bj,j;

(1) j:=1.

(2) tantque (j < n) et (aj = bj) faire j := j + 1;  
si aj = bj alors r := 0  
sinon  
si aj < bj alors r := -1  
sinon r := 1

#

algorithme d'addition :

Addition de deux entiers infinis de même nombre de chiffres n, et strictement positifs (on peut toujours considérer deux nombres comme ayant le même nombre de chiffres quitte à rajouter des zéros à gauche du plus court):

ADDLS

(A=a1a2....an  
B=b1b2....bn)

>>> module précision infinie de SYPAC <<<<

base X, le resultat est dans  $C=c_0c_1c_2\dots c_n$  ou  $c_0$  est la retenue eventuelle)

```
(a1) (initialisations)
      j:=n;
      k:=0.

(a2) cj:= RESTE(aj+bj+k,X);
      k:=(aj+bj+k)/X .

(a3) j:=j-1;
      si j>0 alors ALLERA a2
      sinon c0:=k
#
```

remarques:

-----  
La variable j est un indice, la variable k conserve la retenue a chaque etape du calcul. La dernière retenue 1 ou 0 est mise dans c<sub>0</sub>.

algorithme de soustraction:

-----  
Soustraction de deux entiers infinis positifs A et B, tels que  $A \geq B$ , de meme nombre de chiffres n.

SOUSLS

(A=a<sub>1</sub>a<sub>2</sub>....a<sub>n</sub>  
B=b<sub>1</sub>b<sub>2</sub>....b<sub>n</sub>)

base X, le resultat est dans  $C=c_1c_2\dots c_n$ )

```
(s1) j:=n;
      k:=0 .

(s2) cj:=RESTE(aj-bj+k,X);
      k:=(aj-bj+k)/X .

(s3) j:=j-1;
      si j>0 alors ALLERA s2
#
```

remarques:

-----  
La variable j est un indice, la variable k conserve la retenue 0 ou -1 a chaque etape du calcul.

algorithme de multiplication:

-----  
Multiplication d'un entier infini à n chiffres A, par un entier infini B à m chiffres, le produit etant un entier infini a au plus m+n chiffres.

>>> module précision infinie de SYPAC <<<<

MULTLS

(A=a1a2...an  
B=b1b2...bm

base X, le resultat est dans C=C1C2...Cm+n)

(m1) pour i=1,2,...n faire Cn+i:=0;  
      j:=m.

(m2) si bj=0 alors { Cj:=0;  
                    ALLERA m6 }.

(m3) i:=n;  
      k:=0 .

(m4) t:=ai\*bj + Ci+j + k;  
      Ci+j:=RESTE(t,X) .

(m5) i:=i-1;  
      si i>0 alors ALLERA m4  
      sinon Cj:=k.

(m6) j:=j-1;  
      si j>0 alors ALLERA m2

#

remarques:

-----  
Cet algorithme, comme les precedents est un algorithme classique, correspondant au calcul effectue "à la main", en multipliant le multiplicande a1a2...an, par chaque chiffre bj du multiplicateur, ici l'addition des resultats intermediaires a lieu en meme temps que les multiplications, cf. paragraphe (m4) de l'algorithme. Il existe d'autres algorithmes signales dans (1), plus performants en temps, le but actuel etant de montrer que PALDES permet de faire du calcul en précision infinie, dans un deuxième temps on implémentera des algorithmes plus performants.

algorithme de la division:

-----  
Division d'un entier infini A=A1A2...Am+n et d'un entier infini B=B1B2...Bn ou B1<>0 et n>1. On a donc A>B, A et B tous les deux strictement positifs.

DIVLS

(n prévoit un chiffre de plus pour A normalise, A=A0A1A2...Am+n apres normalisation A0 pouvant etre nul.  
base X, le quotient est dans Q=q0q1...qm et le reste dans R=r1r2...rn )

(d1) (normalisation)

>>> module precision infinie de SYPAC <<<<

```
d:=X/(b1+1);
A:=A*d;
B:=B*d .
```

(d2) j:=0.

```
(d3) si Aj=B1 alors q' := X-1
      sinon q' := (Aj*X + Aj+1)/B1;
      tantque (B2*q' > X*(Aj*X + Aj+1 - q'*B1) + Aj+2) faire
q' := q'-1;
```

(d4) W:=q'\*B.

```
(d5) Qj:=q';
      si Aj...Aj+n < W alors ALLERA d6
      sinon ALLERA d7 .
```

```
(d6) q' := q'-1;
      W := q'*B ;
      Qj := q' .
```

```
(d7) Aj...Aj+n := Aj...Aj+n - W;
      j := j+1;
      si j <= m alors ALLERA d3 .
```

```
(d8) rest:=0;
      pour i=1,2,...,n faire
      ( z:=X*rest + Ai;
        rest:=RESTE(z,d);
        Ri:=z/d ).
```

#

remarques:

-----  
Cet algorithme est légèrement différent par l'écriture de celui de Knuth (2), mais il est parfaitement équivalent, pour améliorer sa lisibilité au niveau des paragraphes (d4), (d5) et (d6) une variable W entier infini a été introduite. L'étape de la dénormalisation du paragraphe (d8) a été terminée.



SYSTEME DE CALCUL EN PRECISION INFINIE

EN PALDES

```
ETIQUETTE 0;  
GLOBAL CARAC REP, GSI;  
GLOBAL LISTE POL1, POL2, POL3, POL4, POL5, POL6;  
GLOBAL GI;  
GLOBAL GJ, GK;
```

```
X:=STATU(L=LISTE)  
LOCAL Y;  
(1) ETAT(L, Y);  
X:=Y  
#
```

```
ECRNBR(L=LISTE)  
LOCAL I, X, Y, N;  
LOCAL TI(160);
```

```
(0) DEGRE(L, N).  
(1) SI (STATU(L)=0) ALORS IMPRIMER "0"  
SINON  
% I:=N;  
TANTQUE (I)=0 FAIRE  
% CHER(I, Y, L);  
X:=I+1;  
TI(X):=Y;  
I:=I-1  
%;  
I:=N;  
TANTQUE (I)=0 FAIRE  
% X:=I+1;  
IMPRIMER$ TI(X);  
I:=I-1  
%;  
IMPRIMER" "  
%  
#
```

```
SORESUL(L1, L2, L3=LISTE, COEF)  
LOCAL I;
```

```
(1) ECRNBR(L1);  
ECRNBR(L2);  
POUR I=1, ..., 40 FAIRE  
IMPRIMER "$"-";  
IMPRIMER " ";  
ECRNBR(L3)  
#
```

ENTVERLS(N:L=LISTE)

(1) EFFLS(;L).

(2) MET(0,N;L)

#

SGNLS(L=LISTE:N)

LOCAL X,Y;

(1) SI (STATU(L)=0) ALORS N:=0

SINON

% PREM(L;X,Y);

SI (Y>0) ALORS N:=1

SINON N:=-1

%

#

OPPLS(L=LISTE:R=LISTE)

LOCAL X,Y;

(1) EFFLS(;R);

SI (STATU(L)=1) ALORS

% PREM(L;X,Y);

COPIE(L;R);

MET(X,-Y;R)

%

#

DIVCT(L=LISTE;K:Q=LISTE;R)

LOCAL Z,N,J,Y;

(1) EFFLS(;Q);

R:=0;

DEGRE(L;N).

(2) SI (STATU(L)=1) ALORS

SI (K=1) ALORS

% COPIE(L;Q) %

SINON

% POUR J=N,N-1,...,0 FAIRE

% CHER(J;Y,L);

Z:=10\*R+Y;

R:=Z-(Z/K)\*K;

MET(J,Z/K;Q)

%

%

#

ABSLS(L1=LISTE:L2=LISTE)

LOCAL X,Y;

(1) EFFLS(;L2).

(2) SI (STATU(L1)=1) ALORS

% COPIE(L1;L2);

PREM(L1;X,Y);

SI (Y<0) ALORS MET(X,-Y;L2)

%

#

```

COMPABS(L1,L2=LISTE:R)
LOCAL LISTE LOC1,LOC2;
LOCAL D1,D2,Y1,Y2;
(1) ABSLS(L1;LOC1);
    ABSLS(L2;LOC2).
(2) DEGRE(LOC1;D1);
    DEGRE(LOC2;D2);
    SI (D1>D2) ALORS R:=1
    SINON
    SI (D2>D1) ALORS R:=-1
    SINON
    % PREM(LOC1;D1,Y1);
    PREM(LOC2;D2,Y2);
    REDUC(;LOC1);
    REDUC(;LOC2);
    TANTQUE ((D1=D2) ET (Y1=Y2) ET ((STATU(LOC1)=1) OU (STATU(LOC2)=1))) FAIRE
    % SI ((STATU(LOC1)=0) OU (STATU(LOC2)=0)) ALORS
        % Y1:=0;
        Y2:=0;
        D1:=0;
        D2:=0
    %;
    PREM(LOC1;D1,Y1);
    PREM(LOC2;D2,Y2);
    REDUC(;LOC1);
    REDUC(;LOC2)
    %;
    SI (D1<D2) ALORS R:=-1
    SINON
    % SI (D1>D2) ALORS R:=1
    SINON
    % SI (Y1<Y2) ALORS R:=-1
    SINON
    % SI (Y1>Y2) ALORS R:=1
    SINON
    % SI ((STATU(LOC1)=0) ET (STATU(LOC2)=0)) ALORS R:=0
    SINON
    % SI (STATU(LOC1)<>1) ALORS R:=-1
    SINON
    SI (STATU(LOC2)<>1) ALORS R:=1
    %
    %
    %
    %
    %
    %
    %

```

```

COMPLS(L1, L2=LISTE:R)
LOCAL N1, N2, Y;
(1) SGNLS(L1; N1);
    SGNLS(L2; N2);
    SI ((N1=0) ET (N2=0)) ALORS R:=0
    SINON
    % SI (N1=0) ALORS
        % SI (N2>0) ALORS R:=-1 SINON R:=1 %
        SINON
        % SI (N2=0) ALORS
            % SI (N1>0) ALORS R:=1SINON R:=-1 %
            SINON
            % COMPABS(L1, L2; Y);
            SI (Y=0) ALORS R:=0
            SINON
            % SI (Y=1) ALORS
                % SI (N1>0) ALORS R:=1 SINON R:=-1 %
                SINON
                % SI (N2>0) ALORS R:=-1 SINON R:=1 %
            %
        %
    %
#

```

```

SOUSUP(N, M; L=LISTE:W; R=LISTE)
LOCAL J, Z, Y;

(1) POUR J=N+1, N+2, ..., M FAIRE
    % CHER(J; Y, L);
    Z:=Y+W;
    W:=0;
    SI (Z<0) ALORS % W:=-1;
                    Z:=Z+10
                    %;
    SI (Z<>0) ALORS MET(J, Z; R)
%
#

```

```

SOUSLS<P,Q=LISTE;R=LISTE>
LOCAL J, N, M, W, D1, D2, Z, Y1, Y2;

```

```

(1) EFFLS<R>;
SI ((STATU<P>=0) OU (STATU<Q>=0)) ALORS
% SI (STATU<P>=0) ALORS
  %SI (STATU<Q>=1) ALORS OPPLS<Q;R>%
  SINON
  SI (STATU<Q>=0) ALORS COPIE<P;R>
%

```

```

(2) SI ((STATU<P>=1) ET (STATU<Q>=1)) ALORS
% W:=0;
  DEGRE<P;D1>;
  DEGRE<Q;D2>;
  SI (D1<=D2) ALORS
    % N:=D1;
    M:=D2
  %
  SINON
    % N:=D2;
    M:=D1
  %;
  POUR J=0, 1, ..., N FAIRE
    % CHER<J;Y1,P>;
    CHER<J;Y2,Q>;
    Z:=Y1-Y2+W;
    W:=0;
    SI (Z<0) ALORS % W:=-1;
      Z:=Z+10
    %;
    SI (Z<0) ALORS MET<J,Z;R>
  %;
  SI (D1<>D2) ALORS SOUSUP<N,M,P;W,R>
%

```

```

ADDSUP<N,M;L=LISTE;W;R=LISTE>
LOCAL J, Z, Y;

```

```

(1) POUR J=N+1, N+2, ..., M FAIRE
% CHER<J;Y,L>;
  Z:=Y+W;
  W:=0;
  SI (Z)=10) ALORS % W:=1;
    Z:=0
  %;
  SI (Z<0) ALORS MET<J,Z;R>
%

```

```

ADDLS(P, Q=LISTE; R=LISTE)
LOCAL ZR, J, N, M, W, D1, D2, Z, Y1, Y2;

```

```

(1) EFFLS(; R);
  SI ((STATU(P)=0) OU (STATU(Q)=0)) ALORS
  % SI (STATU(P)=0) ALORS
  %SI (STATU(Q)=1) ALORS COPIE(Q; R)%
  SINON
  SI (STATU(Q)=0) ALORS COPIE(P; R)
  %
  SINON
  % W:=0;
  DEGRE(P; D1);
  DEGRE(Q; D2);
  SI (D1<=D2) ALORS
  % N:=D1;
  M:=D2
  %
  SINON
  % N:=D2;
  M:=D1
  %
  %
  %

```

```

(2) SI ((STATU(P)=1) ET (STATU(Q)=1)) ALORS
  % POUR J=0, 1, ..., N FAIRE
  % CHER(J; Y1, P);
  CHER(J; Y2, Q);
  Z:=Y1+Y2+W;
  W:=0;
  SI (Z>=10) ALORS % W:=1;
  RESTE(Z, 10; ZR);
  Z:=ZR
  %;
  SI (Z<>0) ALORS MET(J, Z; R)
  %;

  SI (D1<>D2) ALORS
  % SI (D1=M) ALORS ADDSUP(N, M, P; W, R)
  SINON ADDSUP(N, M, Q; W, R)
  %;
  SI (W=1) ALORS MET(M+1, 1; R)
  %

```

```
MULTLS(P, Q=LISTE:R=LISTE)
LOCAL ZR, I, J, K, N, M, Y1, Y2, Y3, W, Z;
```

```
(1) EFFLS(R);
    DEGRE(P, N);
    DEGRE(Q, M);
```

```
(2) SI ((STATU(P)=1) ET (STATU(Q)=1)) ALORS
    % POUR J=0, 1, ..., M FAIRE
    % CHER(J, Y2, Q);
    SI (Y2<>0) ALORS
    % W:=0;
    POUR I=0, 1, ..., N FAIRE
    % K:=I+J;
    CHER(K, Y3, R);
    CHER(I, Y1, P);
    Z:=Y1*Y2+Y3+W;
    W:=Z/10;
    RESTE(Z, 10, ZR);
    Z:=ZR;
    MET(I+J, Z, R)
    %;
    MET(N+J+1, W, R)
    %
    %
    %
    %
```

```
PREMIER(N, M, D, V1, V2; A, B=LISTE)
LOCAL I, J, K;
LOCAL LISTE LD, T;
```

```
(1) DEGRE(A, I);
    DEGRE(B, J);
    PREM(B, K, V1);
    N:=J+1;
    M:=I-J;
    D:=10/(V1+1);
    ENTVERLS(D, LD);
    MULTLS(A, LD, T);
    AFFECT(T, A);
    MULTLS(B, LD, T);
    AFFECT(T, B);
    PREM(B, K, V1);
    K:=J-1;
    CHER(K, Y2, B)
```

```
RESTDIV(M, N, D; A=LISTE:R=LISTE)
LOCAL J, Z, UX, REST;
```

```
(1) REST:=0;
    POUR J=N-1, N-2, ..., 0 FAIRE
    % CHER(J, UX, A);
    Z:=10*REST+UX;
    REST:=Z-(Z/D)*D;
    MET(J, Z/D, R)
    %
```



```

DIVLS(A, B=LISTE:Q, R=LISTE)
LOCAL LISTE LD, LQCH, T, W, X;
LOCAL I, J, K, N, M, D, V1, V2, U1, U2, U3;
LOCAL Z, QS, UX, TEST, REST;

```

```

(1) PREMIER(N, M, D, V1, V2, A, B).
(2) J:=M+N;
    TANTQUE (J>=N) FAIRE
    % CHER(J; U1, A);
      K:=J-1;
      CHER(K; U2, A);
      Z:=10*U1+U2;
      SI (U1=V1) ALORS QS:=9
      SINON QS:=Z/V1;
      K:=J-2;
      CHER(K; U3, A);
      TANTQUE (V2*QS>10*(Z-QS*V1)+U3) FAIRE QS:=QS-1;
      ENTVERLS(QS; LQCH);
      MULTLS(B, LQCH; W);
      MET(J-N, QS; Q);
      I:=J;
      TANTQUE (I>=J-N) FAIRE
      % CHER(I; UX, A);
        MET(N-J+I, UX; X);
        I:=I-1
      %;
      COMPLS(X, W; TEST);
      SI (TEST<0) ALORS
      % QS:=QS-1;
        MET(J-N, QS; Q);
        ENTVERLS(QS; LQCH);
        MULTLS(B, LQCH; W)
      %;
      SOUSLS(X, W; T);
      AFFECT(T; X);
      I:=N;
      TANTQUE (I>=0) FAIRE
      % CHER(I; UX, X);
        MET(I-N+J, UX; A);
        I:=I-1
      %;
      J:=J-1
    %;

```

```

(3) RESTDIV(M, N, D, A; R)

```

\*

DIVISION(A, B=LISTE:Q, R=LISTE)

LOCAL SA, SB;

LOCAL T, D2, Y2;

LOCAL LISTE LA, LB;

(0) ABSLS(A; LA);

ABSLS(B; LB);

SGNLS(A; SA);

SGNLS(B; SB).

(1) EFFLS(; Q);

EFFLS(; R).

(2) SI (STATU(LB)=0) ALORS IMPRIMER"DIVISION IMPOSSIBLE"

SINON

SI (STATU(LA)=1) ALORS

% COMPLS(LA, LB; T);

SI (T=0) ALORS MET(0, 1; Q)

SINON

SI (T=1) ALORS

% PREM(LB; D2, Y2);

SI (D2=0) ALORS

% DIVCT(LA, Y2; Q, T);

NET(0, T; R)

%

SINON DIVLS(LA, LB; Q, R)

%

SINON COPIE(LA; R)

%

(3) SI ((SA<0) ET (SB<0)) ALORS

% PREM(Q; D2, Y2);

MET(D2, -Y2; Q);

SI (STATU(R)=1) ALORS

% PREM(R; D2, Y2);

MET(D2, -Y2; R)

%

%;

SI ((SA>0) ET (SB<0)) ALORS

% PREM(Q; D2, Y2);

MET(D2, -Y2; Q)

%;

SI ((SA<0) ET (SB<0)) ALORS

% SI (STATU(R)=1) ALORS

% PREM(R; D2, Y2);

MET(D2, -Y2; R)

%

%

#

```
MULTIPLICATION(P, Q=LISTE:R=LISTE)  
LOCAL SIGNE, X1, Y1, X2, Y2;  
LOCAL LISTE LP, LQ;
```

```
(0) ABSLS(P; LP);  
    ABSLS(Q; LQ).
```

```
(1) PREM(P; X1, Y1);  
    PREM(Q; X2, Y2);  
    SI (Y1*Y2>0) ALORS SIGNE:=+1  
    SINON SI (Y1*Y2<0) ALORS SIGNE:=-1  
    SINON SIGNE:=0;  
    SI (Y1<0) ALORS Y1:=-Y1;  
    SI (Y2<0) ALORS Y2:=-Y2;  
    MET(X1, Y1; P);  
    MET(X2, Y2; Q).
```

```
(2) MULTLS(LP, LQ; R);  
    SI (STATU(R)=1) ALORS  
    % PREM(R; X1, Y1);  
      Y1:=SIGNE*Y1;  
    MET(X1, Y1; R)  
    %
```

```
ADDITION(P, Q=LISTE; R=LISTE)
LOCAL SP, SQ, RES, X, Y;
LOCAL LISTE LP, LQ;
```

```
(0) ABSLS(P; LP);
    ABSLS(Q; LQ);
```

```
(1) SGNLS(P; SP);
    SGNLS(Q; SQ);
```

```
(2) SI (SP*SQ)=0) ALORS
    % SI (SP*SQ=0) ALORS ADDLS(P, Q; R)
    SINON
    % ADDLS(LP, LQ; R);
    SI ((SP=-1) ET (SQ=-1)) ALORS
    % PREM(R; X, Y);
    MET(X, -Y; R)
    %
    %
    %
    SINON
    % COMPABS(P, Q; RES);
    SI (RES)=0) ALORS
    % SOUSLS(LP, LQ; R);
    SI (SP=-1) ALORS
    % PREM(R; X, Y);
    MET(X, -Y; R)
    %
    %
    SINON
    % SOUSLS(LQ, LP; R);
    SI (SQ=-1) ALORS
    % PREM(R; X, Y);
    MET(X, -Y; R)
    %
    %
    %
```

```
MENU()
```

```
(1) IMPRIMER "A... POUR ADDITION DE 2 LISTES. ";
    IMPRIMER "M... POUR MULTIPLICATION DE 2 POLY. ";
    IMPRIMER "D... POUR DIVISION DE 2 LISTES. ";
    IMPRIMER "P... POUR PGCD DE 2 LISTES. ";
    IMPRIMER "F... POUR FINIR. ";
    ENTRER REP
```

CHAPITRE VI

LES POLYNOMES DE  $Z(X)$  EN PALDES:

=====

C'est un autre sous-type du type liste de PALDES, il est censé représenter les polynômes à une indéterminée notée  $X$ , à coefficients dans  $Z$ .

Le type polynôme fait aussi partie de l'extension du système PALDES minimal, avec cette extension, l'utilisateur dispose d'opérations sur les polynômes de  $Z(X)$ :

- Addition de deux polynômes.
- Multiplication de deux polynômes.
- Division lorsqu'elle est possible.
- PGCD de deux polynômes.
- Valeur numérique d'un polynôme.
- Dérivée d'un polynôme par rapport à  $X$ .

Les listes de PALDES sont un bon support des polynômes, elles ont été conçues pour cet objectif. Les polynômes devant avoir une représentation dense, les coefficients nuls ne sont pas représentés.

Un polynôme PALDES est une liste dont chacune des cellules a la signification suivante:

- Les polynômes de PALDES sont ordonnés suivant les puissances décroissantes de la variable  $X$ , une cellule représente un monôme complet du polynôme.
- L'élément PUISS représente la puissance de  $X$  dans le monôme, l'élément COEFF, le coefficient signe du monôme.
- Les éléments PUISS sont positifs, les COEFF des entiers machine quelconque.

CONSTITUTION DU SYSTEME POLYNOMIAL PALDES:

=====

le type polynôme de  $Z(X)$  est noté POLYNOME dans l'implémentation, par la suite nous écrirons polynôme pour polynôme de  $Z(X)$ .

liste des algorithmes composant le systeme:

-----

On ne les détaillera pas, comme ceux du système de précision infinie, ils sont écrits en PALDES à la fin du document. Les opérations de base sur les polynômes sont semblables à celles de la précision infinie.

addition:

-----

On reprend l'algorithme de précision infinie, en supprimant le problème de la retenue (on obtient addition et soustraction qui sont ici confondues car les coefficients sont signes).

multiplication:

---

Même démarche que pour l'addition, les retenues sont éliminées

division: (polynômes unitaires)

---

Il est impossible de diviser deux polynômes quelconques dans  $Z(X)$ , une extension ultérieure utilisant les polynômes dans  $Q(X)$  le fera. Par contre certains polynômes sont toujours divisibles par d'autres, comme dans le cas intéressant de polynômes "unitaires".

Un polynôme est dit unitaire si son coefficient du monôme de plus haut degré est égal à 1.

Dans l'algorithme de division dans les opérations effectuées sur les coefficients sont des additions, des multiplications, des soustractions, opérations autorisées dans un anneau comme l'est  $Z$ , il y a une seule instruction où l'on divise des coefficients par le coefficient du monôme de plus haut degré:

algorithme:

```
(d1) pour k=m-n, m-n-1, ..., 0 faire
    { Qk:= Un+k / Vn;
      pour j=n+k-1, n+k-2, ..., k faire
        Uj:=Uj - Qk * Vj-k }
```

#

Les seules divisions sont faites par  $V_n$ , lorsque  $V$  est unitaire cet algorithme donne le quotient et le reste dans  $Z$ .

Dans ce cas, tout polynôme de  $Z(X)$  peut être divisé euclidiennement par un polynôme unitaire. Or ce cas permet de traiter le PGCD de polynômes dans  $Z(X)$ , qui lui a toujours une solution, puisque le PGCD est défini à une constante multiplicative près.

On trouvera dans (1), un détail complet des algorithmes utilisés ici, une implémentation en FROLOG de tous ces algorithmes et bien d'autres a été réalisée dans le système SYCOPHANTE (cf. (5)). Les objectifs premiers n'étant pas de faire une étude complète sur les différents

>>>> module polynômial de SYPAC <<<<

algorithmes, comme pour la precision infinie, les algorithmes les plus classiques ont ete utilises.

valeur numérique:

---

Le schéma de horner est utilisé pour le calcul.

polynome dérivé:

---

La formule classique  $(aX^n)' = n a X^{n-1}$  est utilisée.

PGCD de deux polynômes:

---

Pour le calcul du PGCD de deux polynômes, l'algorithme euclidien generalise a ete essaye. Comme le systeme PALDES doit pouvoir évoluer en système de calcul dans  $\mathbb{Q}(X)$ , avec des coefficients infinis, l'algorithme de G.E.Collins decrit dans (2) a ete prefere, parce que plus rapide. Le defaut de cet algorithme etant de générer un overflow plus tot dans le calcul, alors que, pour un meme calcul, l'algorithme euclidien generalise le genere plus tard, ceci permettra de mesurer l'efficacite des entiers infinis dans ce genre de calcul.

Algorithme de G.E.Collins:

---

$U = U_m U_{m-1} \dots U_0$   
 $V = V_n V_{n-1} \dots V_0$   
reste  $r = r_0 r_1 \dots r_{n-1}$

(c1) (rendre les polynomes primitifs)

$d := \text{pgcd}(\text{cont}(U), \text{cont}(V));$   
 $U(x) := \text{pp}(U(x));$   
 $V(x) := \text{pp}(V(x));$   
 $g := 1;$   
 $h := 1.$

(c2) (pseudo-division)  
 $\text{delt} := \text{deg}(U) - \text{deg}(V).$

(r1) pour  $k = m - n, m - n - 1, \dots, 0$  faire  
pour  $j = n + k - 1, n + k - 2, \dots, 0$  faire  
 $U_j := V_n * U_j - U_{n+k} * V_{j-k};$   
 $r_0 := U_0; \dots r_{n-1} := U_{n-1};$   
si  $r(x) = 0$  alors ALLERA c4;  
si  $\text{deg}(r) = 0$  alors  
{  $V(x) := 1;$   
ALLERA c4 } .

(c3) (ajustement du reste)  
 $U(x) := V(x);$   
 $V(x) := r(x) / (g * h^{\text{delt}});$   
 $g := 1(U);$   
 $h := h^{(1-\text{delt})} * g^{\text{delt}};$



ALLERA c2.

(c4) le résultat est  $d*pp(V(x))$

#

Les algorithmes employés sont tous écrits en FALDES:

cont(U(x)):

donne en sortie le pgcd des coefficients de U(x).

pp(U(x)):

est défini par  $pp(U(x)) = U(x) / cont(U(x))$

deg(U):

donne le degré du polynôme U

conclusion sur le système polynôme:

Le but d'implantation de manipulations polynômes élémentaires, pour l'instant, en FALDES, semble atteint. On pourra voir à la fin du document quelques exemples d'exécution de ce système sur des polynômes simples. Une version plus performante est possible, les algorithmes employés n'étant pas les plus rapides. Une utilisation de ce système dans un ensemble à usage pédagogique est envisagée. Car dans les exemples scolaires les polynômes courants n'ont que des coefficients à trois ou quatre chiffres au plus. Les algorithmes implantés dans le système FALDES, peuvent donc servir de base à une évolution ultérieure dans  $\mathbb{Q}(X)$ .

## SYSTEME DE CALCUL POLYNOMIAL EN PALDES

Le reste du système a été écrit en PALDES afin de se servir des algorithmes de gestion de liste définis précédemment. Cette famille d'algorithmes effectue des opérations formelles sur les polynômes de  $\mathbb{Z}[X]$

Addition

·  
·  
·  
·

pgcd

Presque tous ces algorithmes sont accessibles aux programmes PALDES :

**ADDPOL** : réalise l'addition polynômiale

se sert de 2 algorithmes RECOP et SOMME

**MULTPOL** : réalise la multiplication polynômiale

**LAMBDA P** : multiplie ou divise tous les coefficients d'un polynôme par un entier relatif (la division est entière)

**DIVPOL** : effectue la division euclidienne de 2 polynômes de  $\mathbb{Z}[X]$  (lorsque c'est possible) et fourni le quotient et le reste

**PGCDPOL** : ici algorithmes et ceux auxquels il fait appel

RESTEPOL (reste généralisé)

CONT

PP

Sont décrits dans Knuth et ont été immédiatement implémentés.

Le calcul du PGCD dans  $\mathbb{Z}[X]$  est réalisé par l'algorithme de Collins malgré l'overflow rapide qu'il peut générer.

La méthode d'euclide généralisée a été testée et prend environ 10 % de temps supplémentaire à l'exécution pour un overflow plus tardif.

```

AFFECT<LINIT=LISTE:LRES=LISTE>
(0) EFFLS<;LRES>;
    COPIE<LINIT;LRES>
#

T:=EGAL<L1,L2=LISTE>=LOGIQUE
ETIQUETTE I0;
LOCAL I0,J0;
LOCAL X1,Y1,X2,Y2;
LOCAL LISTE L01,L02;
LOCAL LOGIQUE OK;

(0) COPIE<L1;L01>;
    COPIE<L2;L02>;
    OK:=VRAI.

(1) LONG<L01;I0>;
    LONG<L02;J0>;
    SI <I0<>J0> ALORS
        %OK:=FAUX;
        ALLERA I0
    % .

(2) TANTQUE <OK ET <STATU<L01>=1>> FAIRE
    %PREM<L01;X1,Y1>;
    PREM<L02;X2,Y2>;
    REDUC<;L01>;
    REDUC<;L02>;
    SI <<X1<>X2> OU <Y1<>Y2>> ALORS OK:=FAUX
    % .

(10) T:=OK
#

RECOP<N:P;L,R=LISTE>
LOCAL EXPOS,COEFF;
(1) TANTQUE <<P>N> ET <STATU<L>=1>> FAIRE
    %PREM<L;EXPOS,COEFF>;
    MET<EXPOS,COEFF;R>;
    REDUC<;L>;
    DEGRE<L;P>
    %
#

SOMME<;PL,QL,RL=LISTE>
LOCAL X1,Y1,X2,Y2,Z;

(1) PREM<PL;X1,Y1>;
    PREM<QL;X2,Y2>;
    REDUC<;QL>;
    REDUC<;PL>;
    Z:=Y1+Y2;
    SI <Z<>0> ALORS MET<X1,Z;RL>
#

```

```
ADDPOL(P, Q=LISTE:R=LISTE)
LOCAL LISTE PL, QL;
LOCAL N1, N2, LONG1, LONG2;
```

```
(0) COPIE(P; PL);
    COPIE(Q; QL);
    LONG1:=STATU(PL);
    LONG2:=STATU(QL).
(1) TANTQUE ((LONG1=1) ET (LONG2=1)) FAIRE
    %DEGRE(PL; N1);
    DEGRE(QL; N2);
    SI (N1>N2) ALORS
    %RECOP(N2; N1, PL, R)%
    SINON
    %SI (N1<N2) ALORS
    %RECOP(N1; N2, QL, R)%
    SINON SOMME(, PL, QL, R)
    % ;
    LONG1:=STATU(PL);
    LONG2:=STATU(QL)
    % .
(2) SI (LONG1=0) ALORS
    %LONG2:=STATU(QL);
    TANTQUE (LONG2=1) FAIRE
    %RECOP(-1; N2, QL, R);
    LONG2:=STATU(QL)
    %
    %
    SINON
    SI (LONG2=0) ALORS
    %LONG1:=STATU(PL);
    TANTQUE (LONG1=1) FAIRE
    %RECOP(-1; N1, PL, R);
    LONG1:=STATU(PL)
    %
    %
```

```
#
MULTI(EXP, COEF=, Q=LISTE:R=LISTE)
LOCAL LISTE LOC;
LOCAL X, Y;
```

```
(1) COPIE(Q; LOC).
(2) TANTQUE (STATU(LOC)=1) FAIRE
    %PREM(LOC; X, Y);
    CUMUL(X+EXP, Y*COEF; R);
    REDUC(, LOC)
    %
```

```
#
```

```
MULTPOL(P, Q=LISTE:R=LISTE)
LOCAL I, J;
LOCAL LISTE LOC;
```

```
(1) COPIE(P; LOC);
```

```
(2) TANTQUE (STATU(LOC)=1) FAIRE
    %PREM(LOC; I, J);
    MULTI(I, J, Q; R);
    REDUC(; LOC)
    %
```

```
#
```

```
LAMBDA(P; P=LISTE:R=LISTE)
LOCAL S, EX, COE;
LOCAL LISTE LOC;
```

```
(1) COPIE(P; LOC);
    TANTQUE (STATU(LOC)=1) FAIRE
    %PREM(LOC; EX, COE);
    MET(EX, L*COE; R);
    REDUC(; LOC)
    %
```

```
#
```

```
DIVPOL(A, B=LISTE:Q, R=LISTE; KOEF)
LOCAL D1, D2;
LOCAL X1, X2, Y1, Y2, EXP, COEFF;
LOCAL LISTE AP, BP, S, MUL;
```

```
(0) FREM(B; X2, Y2);
    KOEF:=1;
    SI (Y2<>1) ALORS
    %PREM(A; X1, Y1);
    PPCMN(Y1, Y2; KOEF);
    KOEF:=KOEF/Y1;
    LAMBDA(P; KOEF, A; AP)
    %
    SINON COPIE(A; AP);
    DEGRE(AP; D1);
    DEGRE(B; D2);
```

```
(1) TANTQUE ((D1)=D2) ET (STATU(AP)=1) FAIRE
    %PREM(AP; X1, Y1);
    EXP:=X1-X2;
    COEFF:=Y1/Y2;
    MET(EXP, COEFF; Q);
    MET(EXP, COEFF; MUL);
    MULTPOL(MUL, B; S);
    REDUC(; MUL);
    LAMBDA(P; -1, S; S);
    ADDPOL(AP, S; BP);
    AFFECT(BP; AP);
    EFFLS(; BP);
    EFFLS(; S);
    DEGRE(AP; D1)
    %
```

```
(2) SI (STATU(AP)=1) ALORS COPIE(AP; R)
```

```
#
```

COEFF(POL=LISTE;N:X)

(0) CHER(N;X,POL)

#

RESTEPOL (A,B=LISTE;R=LISTE)

LOCAL I, J, K, M, N, VN, UJ, UNK, VJK, X, Y;

LOCAL LISTE U, V;

(0) COPIE(A;U);  
COPIE(B;V);  
PREM(V;I,VN);  
N:=I;  
DEGRE(U;M).

(1) POUR K=M-N, M-N-1, ..., 0 FAIRE  
% COEFF(U, N+K; UNK);  
POUR J=N+K-1, N+K-2, ..., K FAIRE  
% COEFF(U, J; X);  
COEFF(V, J-K; Y);  
UJ:=VN+X-UNK\*Y;  
MET(J, UJ; U)  
%;  
POUR J=K-1, K-2, ..., 0 FAIRE  
% COEFF(U, J; UJ);  
UJ:=VN\*UJ;  
MET(J, UJ; U)  
%  
%.

(2) DEGRE(U; I);  
TANTQUE ((I)=N) ET (STATU(U)=1) FAIRE  
%REDUC(U);  
DEGRE(U; I)  
%;  
COPIE(U; R)

#

CONT(U=LISTE;X)

ETIQUETTE 2;

LOCAL LISTE L;

LOCAL T, Y, Z;

(0) COPIE(U; L).

(1) PREM(L; T, Y);  
REDUC(L);  
Z:=Y;  
TANTQUE (STATU(L)=1) FAIRE  
% PREM(L; T, Y);  
PGCDN(Z, Y; T);  
Z:=T;  
SI (Z=1) ALORS ALLERA 2;  
REDUC(L)  
%.

(2) X:=Z

#

```
PP(L0=LISTE:L1=LISTE)
LOCAL LISTE L;
LOCAL I, EX, CO;
```

```
(0) COPIE(L0;L);
    CONT(L;I).
```

```
(1) TANTQUE (STATU(L)=1) FAIRE
    % PREM(L;EX,CO);
    MET(EX,CO/I;L1);
    REDUC(L)
    %
```

```
#
```

```
PGCDPOL(A,B=LISTE:P=LISTE)
ETIQUETTE 2,3;
LOCAL LISTE R,U,V,UN;
LOCAL D;
```

```
(1) IMPRIMER"PGCDPOL:EUCLIDE GENERALISE";
    MET(0,1;UN);
    PP(A;U);
    PP(B;V).
```

```
(2) RESTEPOL(U,V;R);
    DEGRE(R;D);
    SI (STATU(R)=0) ALORS ALLERA 3;
    SI (D=0) ALORS
    % AFFECT(UN;V);
    ALLERA 3
    % ;
    AFFECT(V;U);
    EFFLS(V);
    PP(R;V);
    EFFLS(R);
    ALLERA 2 .
```

```
(3) COPIE(V;P)
```

```
#
```

```
SORESUL(L1,L2,L3=LISTE;COEF)
LOCAL I;
```

```
(1) ECRLS(L1);
    ECRLS(L2);
    POUR I=1,...,40 FAIRE
    IMPRIMER "$"-";
    IMPRIMER " ";
    SI (COEF<>1) ALORS IMPRIMER $"1/":COEF;
    ECRLS(L3)
```

```
#
```

```
P:=PAIR(K)=LOGIQUE
```

```
(1) SI (K=2*(K/2)) ALORS %P:=VRAI%
    SINON P:=FAUX
```

```
#
```

```
PUISS(X, N:R)
LOCAL I, J, K;
```

```
(1) I:=X;
    J:=1;
    K:=N;
    SI (K=1) ALORS %R:=X%
    SINON
    % TANTQUE (K>0) FAIRE
      % TANTQUE (PAIR(K)) FAIRE
        % K:=K/2;
        I:=I*I;
      %;
      K:=K-1;
      J:=J*I;
    %;
    R:=J;
  %
```

```
VALNUM(VAL, P=LISTE:V)
LOCAL LISTE LOC;
LOCAL X, Y, S, T;
```

```
(1) COPIE(P; LOC);
    S:=0;

(2) TANTQUE (STATU(LOC)=1) FAIRE
    % PREM(LOC; X, Y);
    PUISS(VAL, X; T);
    S:=S+T*Y;
    REDUC(LOC);
  %;

(3) V:=S;
```

```
DERIVE(P=LISTE:R=LISTE)
LOCAL N, X, Y;
LOCAL LISTE LOC;
```

```
(1) COPIE(P; LOC);
    EFFLS(LOC);

(2) REPETER
    PREM(LOC; X, Y);
    MET(X-1, X*Y; R);
    REDUC(LOC);
    DEGRE(LOC; N)
  JUSQUE ((N=0) OU (STATU(LOC)=0))
```



BIBLIOGRAPHIE:

GENERALITES:

=====

(1) D.E.KNUTH

The art of computer programming vol 1:  
fundamental algorithms  
addison wesley 1973 (2nd edition).

(2) D.E.KNUTH

The art of computer programming vol 2:  
semi-numerical algorithms  
addison wesley 1981 (2nd edition).

GRAMMAIRES LL(1) ET COMPILATION:

=====

(3) J.ARSAC

la construction de programmes structures dunod  
1977.

(4) N.WIRTH

algorithms+data structures=programs prentice hall,  
int. series in computer science 1976.

• (5) AHO & ULLMAN

Theory of parsing, translation and compiling vol  
1: Parsing  
prentice hall, int.series in computer science, 1972.

(6) AHO & ULLMAN

Theory of parsing, translation and compiling vol  
2: Compiling  
prentice hall, int.series in computer science, 1973.

(7) H.GALLAIRE

Techniques de compilation méthodes d'analyse  
syntaxique  
Ecole Nationale Supérieure de l'Aéronautique et de  
l'Espace, CEPADUES édition, 1977.

(8) D.THALMAN & B.LEVRAT

Conception et implantation de langage de  
programmation une introduction à la compilation  
édition G.Morin, 1979

(9) P.Y.CUNIN

M.GRIFFITHS

J.VOIRON

Comprendre la compilation  
Springer Verlag, 1980

\*\*\* references et ouvrages \*\*\*

- (10) F.L.BAUER  
J.EICKEL  
Compiler construction an advanced course  
Springer Verlag, 2nd edition, 1976
- (11) A.B.FYSTER  
Compiler design and construction  
Van nostrand reinhold company, 1980
- (12) D.GRIES  
Compiler construction for digital computer  
John Wiley and sons, 1971
- (13) A.D.Mc GETTRICK  
The definition of programming languages  
Cambridge University press, 1980
- (14) R.BORNAT  
Understanding and writing compilers  
The Macmillan press limited, 1979 .
- (15) P.CALINGAERT  
Assemblers, compilers and program translation  
Pitman publishing limited, 1979
- (16) J.WELSH  
M.Mc KEAG  
Structured system programming  
Prentice Hall, int. series in computer science 1980
- (17) IRIA  
Le point sur la compilation, cours de la  
communaute européenne,  
IRIA 1978
- LE LANGAGE Pascal:  
=====
- (18) N.WIRTH  
The programming language Pascal  
Acta Informatica 1, 35-63, 1972
- (19) K.JENSEN  
N.WIRTH  
Pascal user manual and report  
Springer Verlag 1978
- (20) C.A.R HOARE  
N.WIRTH  
An axiomatic definition of the programming  
language Pascal  
Acta Informatica 2, 335-355, 1973
- (21) B.W.RAVENEL  
Toward a Pascal standard  
IEE computer 12, 4, 68-82, 1979

\*\*\* references et ouvrages \*\*\*

- (22) A.M.ADDYMAN  
Pascal standardisation  
Sigplan notices 15,4,67-69,1980
- (23) L.MOORE  
Foundations of programming with Pascal  
Halsted press j.Willey & sons 1980
- (24) J.TIBERGHIE  
The Pascal Handbook  
SYBEX berkeley 1980
- (25) J.WELSH  
J.ELDER  
Introduction to Pascal  
Prentice Hall 1979
- (26) N.WIRTH  
Systematic programming, an introduction  
Prentice Hall int. series in computer science 1973
- LA VERSION Pascal UCSD:  
=====
- (27) R.ZAKS  
An introduction to Pascal  
Sybex berkeley 1980
- (28) K.L.BOWLES  
Update on UCSD Pascal activities  
Pascal newsletter 8,16-18,1977
- (29) K.L.BOWLES  
An introduction to the UCSD Pascal system  
Behavior research methods & instruments  
10,4,531-538,1978
- (30) K.L.BOWLES  
Status of UCSD project  
Pascal news 11,36-40,1978
- (31) K.L.BOWLES  
Beginner's guide for the UCSD Pascal system  
Byte/ mc graw-hill 1979
- (32) K.L.BOWLES  
A machine independent software for micro and mini  
computers  
Sigplan notes 1,1,6-15,1978
- (33) M.OVERGAARD  
UCSD Pascal: a portable software environment for  
small computers  
AFIPS conference proceedings 747-754,1980

\*\*\* references et ouvrages \*\*\*

- (34) K.A.SHILLINGTON  
G.M.ACKLAND  
UCSD Pascal version 1.5, institute for  
information systems, university of california of  
san diego 1978
- (35) TAHLMAN  
MAGENAT  
VAUCHER  
Le langage Pascal ISO avec Pascal 6000 et Pascal  
UCSD  
Ed. g. Morin 1980
- (36) Apple Computer Inc.  
Apple Pascal: operating system reference manual +  
addendum version 1.1 1980
- (37) Apple Computer Inc.  
Apple Pascal: language reference manual +  
addendum version 1.1 1980
- (38) D.V.MOFFAT  
Index to the periodical literature:  
1981 Pascal Bibliography june,1981  
Sigplan notices 16,1,nov 1981

ALDES ET SYSTEMES SAC-1 ET SAC-2:

=====

- (39) G.E.COLLINS  
University of Wisconsin Technical report n0  
129,july 1971:  
The SAC-1 list processing system
- (40) G.E.COLLINS  
University of Wisconsin Technical report n0  
115,march 1971:  
The SAC-1 polynomial system
- (41) G.E.COLLINS  
University of Wisconsin Technical report n0  
8,september 1971:  
The SAC-1 rational function system
- (42) G.E.COLLINS  
University of Wisconsin Technical report n0  
10,june 1979:  
The SAC-1 modular arithmetic system

- (43) G.E.COLLINS  
University of Wisconsin Technical report n0  
80, february 1970:  
The SAC-1 partial fraction, decomposition and  
rational function integration system
- (44) G.E.COLLINS  
E.HOROWITZ  
University of Wisconsin Technical report n0 19,  
august 1970:  
The SAC-1 polynomial real zero system
- (45) G.E.COLLINS  
University of Wisconsin Technical report n0  
145, february 1972:  
The SAC-1 polynomial GCD and resultant system
- (46) G.E.COLLINS  
University of Wisconsin Technical report n0  
156, march 1973:  
The SAC-1 polynomial factorization system
- (47) G.E.COLLINS  
M.T.Mc CLELLAN  
University of Wisconsin Technical report n0  
154, april 1972:  
The SAC-1 polynomial linear algebra system
- (48) R.LOOS  
Algebraic algorithm description as programs  
University of UTAH , august 1972
- (49) R.LOOS  
The algorithm description language ALDES report  
ACM-SIGSAM bulletin, 10, feb.1976
- (50) R.LOOS  
ALDES implementation guide and system SAC-2  
Universitat Karlsruhe october 1980
- (51) J.CALMET  
A user's presentation of SAC-2/ALDES  
Bulletin CALSYF n0 1, décembre 1981
- (52) R.LOOS  
A kwick index for the algebraic algorithms of the  
SAC-2 and ALDES system  
Universitat Karlsruhe oct.1980

AUTRES SYSTEMES SUR MICRO-ORDINATEUR:

(53) F. TEER

Formula manipulation in Pascal using SAC-1  
Informatika rapport IR-25, nov. 1977

(54) D.R. STOUTEMYER  
A.D. RICH

Capabilities of the muMATH-79 computer algebra  
system for the Intel-8080 Microprocessor  
Proceedings of EUROSAM 1979 lect. notes in  
computer science Springer Verlag n0 72

(55) J. FITCH  
J. MARTI

NLARGEing a Z80 Microprocessor  
University of UTAH summer 1980

(56) J. CALMET & VAN HULSEN

Twente University, to appear in computing  
supplementum Springer Verlag 1982:  
Memorandum n0 360 computer algebra systems

(57) M. BERGMAN & H. KANQUI

sycophante Système de calcul formel et  
d'intégration symbolique sur ordinateur.  
DRME contract 73/828, final report 1975.

(58) M. BERGMAN

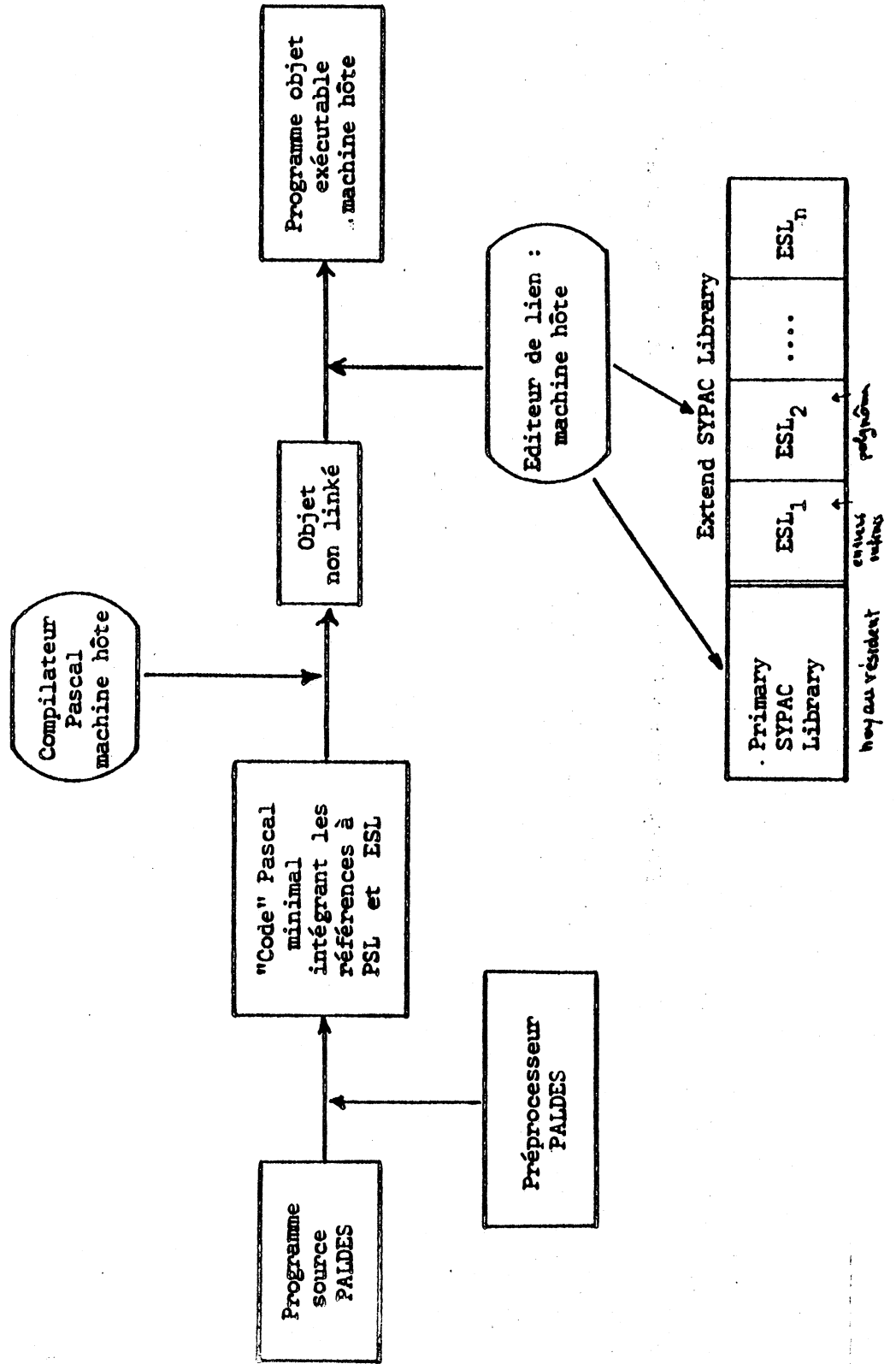
Application de la démonstration automatique aux  
manipulations algébriques et à l'intégration  
formelle sur ordinateur, thèse de 3 ième cycle  
Université d'Aix-Marseille II, 1973

ANNEXES

=====

- \* Organisation générale et utilisation de SYPAC .....p.135
- \* Exemples de calculs en précision infinie .....p.136
- \* Exemples de calculs sur des polynômes en régime  
de gestion transitoire de la mémoire par Pascal ....p.139
- \* Exemples de calculs sur des polynômes en régime  
permanent de gestion par PALDES .....p.145
- \* Exemple complet de compilation et exécution  
d'un programme en PALDES sous SYPAC .....p.148

ORGANISATION GENERALE ET UTILISATION DE SYPAC





1 ADDITION:  
123456789123456789123456789123  
112233445566778899111122234455  
-----  
235690234690235688234579023578

2 ADDITION:  
1234567891234567897687564513145278969749  
987654321098765432109876543210987654321  
-----  
4733665187217021679286102160980777735186

3 ADDITION:  
123456789123456756433421437899875643245634523465432176543456789076543211122334455  
234567891234567890765432123456789987654321234567890765432345678776544332223411997  
-----  
525890255367574408974267225777440867775799025530871977802466753097643335645433

4 MULTIPLICATION:  
214000000557791  
220014004456789  
-----  
46801400176839500168002515308

5 MULTIPLICATION:  
1234567891011234567891  
1234567891011234567891  
-----  
01310358489495059917737711582

6 MULTIPLICATION:  
215701112132141565788327688999  
348821993247567893985773712133  
-----  
75262221199245624471007738074098178696268529613773376924867

7 MULTIPLICATION:  
3740640000003100970092003500540047002100  
1170015204350078001300350022006200330090  
-----  
37630094523224488503050297504728725924742507511111157702445078344766649021890

8 MULTIPLICATION:  
4357261325659127667672899459482375463293  
2723347234491637588985634119237611548524  
-----  
1186633558119115201333726342546433415688378430185901048915058272318603165032953

9

DIVISION:  
200021000024561  
200021000024561

200021000024561  
RESTE : 200021000024561

10

DIVISION:  
101118356489885059917737711582  
101118356489885059917737711582

101118356489885059917737711582  
RESTE : 0

11

DIVISION:  
10000708300  
10000708300

10000708300  
RESTE : 823

12

DIVISION:  
1129073602  
1129073602

1129073602  
RESTE : 1129073602

13

CALCUL DU PGCD:  
1425  
9975

1425

14

CALCUL DU PGCD:  
156750  
9975

1425

15

CALCUL DU PGCD:  
2030625  
2030625

2030625

16

CALCUL DU PGCD:  
1038106265625  
28755680625

2030625

17

CALCUL DU PGCD:  
1077064618729883056840625  
828883169107000390625

4123437890625

Les temps ont été mesurés sur un micro-ordinateur 8 bits l'Apple II et la partie PSL + ESL<sub>2</sub> a été implantée sur un micro-ordinateur 16 bits, possédant un Pascal compilé en code LSI 11/23.

Les performances globales montrent une vitesse d'exécution de 24 fois à 31 fois plus rapide pour le micro-ordinateur 16 bits.

Ce qui laisse penser que SYPAC doit donner des temps raisonnables sur micro-ordinateur 16 bits.

Exemple	8 bits interprété	16 bits compilé
1	4"	-
2	7"	-
3	22"	-
4	5"	-
5	13"	-
6	1'14"	2"
7	1'08"	2"
8	2'40"	5"
9	48"	2"
10	1'08"	3"
11	20"	1"
12	6"	-
13	7"	-
14	14"	-
15	25"	1"
16	47"	1"
17	4'33"	10"

**EXTRAITS D'EXEMPLES  
D'OPERATIONS FORMELLES  
POLYNOMIALES**

-----

**LE SYSTEME DE GESTION DE L'ESPACE MEMOIRE  
ÉTANT LE PASCAL**

-----

**SUR LE LISTING CELA CORRESPOND À LA  
LIGNE SYSTEME DE GESTION LISTE : PASCAL**

-----

ADDITION:

$$+1X^{25}+25X^2-1X^1$$
$$-1X^3+1X^1$$

$$+1X^{25}-1X^3+25X^2$$

NBRE DE MOTS DISPONIBLES: 15892  
SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 55

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 55

ADDITION:

$$+1X^8+1X^5-2X^3-6X^0$$
$$+2X^3-4X^2+1X^1+6X^0$$

$$+1X^8+1X^5-4X^2+1X^1$$

NBRE DE MOTS DISPONIBLES: 15946  
SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 33

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 33

ADDITION:

$$+1X^2+2X^1+1X^0$$

$$+3X^8+4X^6+9X^0$$

$$+3X^8+4X^6+1X^2+2X^1+10X^0$$

NBRE DE MOTS DISPONIBLES: 16018

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 10

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 10

DERIVATION:

$$+1X^7-3X^6+2X^1+10X^0$$

SE DERIVE EN:

$$+7X^6-18X^5+2X^0$$

DERIVATION:

$$+1X^3-5X^2+3X^1-2X^0$$

SE DERIVE EN:

$$+3X^2-10X^1+3X^0$$

DERIVATION:

$$-3X^{23}+2X^5$$

SE DERIVE EN:

$$-69X^{22}+10X^4$$

DERIVATION:

$$+1X^3+3X^2+3X^1+1X^0$$

SE DERIVE EN:

$$+3X^2+6X^1+3X^0$$

Temps inferieurs ou egaux à la secode

MULTIPLICATION:

+1X<sup>1</sup>  
+1X<sup>1</sup>

+1X<sup>2</sup>

NBRE DE MOTS DISPONIBLES: 15985

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 29

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 29

MULTIPLICATION:

+1X<sup>4</sup>+1X<sup>3</sup>+1X<sup>2</sup>+1X<sup>1</sup>

LISTE VIDE

LISTE VIDE

NBRE DE MOTS DISPONIBLES: 16018

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 17

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 17

MULTIPLICATION:

+1X<sup>1</sup>+1X<sup>0</sup>

+1X<sup>1</sup>+1X<sup>0</sup>

+1X<sup>2</sup>+2X<sup>1</sup>+1X<sup>0</sup>

NBRE DE MOTS DISPONIBLES: 16024

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 12

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 12

MULTIPLICATION:

+1X<sup>3</sup>+1X<sup>2</sup>+1X<sup>1</sup>

+1X<sup>0</sup>

+1X<sup>3</sup>+1X<sup>2</sup>+1X<sup>1</sup>

NBRE DE MOTS DISPONIBLES: 16036

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 8

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 8

MULTIPLICATION:

+1X<sup>2</sup>+3X<sup>1</sup>+2X<sup>0</sup>

+1X<sup>1</sup>+1X<sup>0</sup>

+1X<sup>3</sup>+4X<sup>2</sup>+5X<sup>1</sup>+2X<sup>0</sup>

NBRE DE MOTS DISPONIBLES: 13261

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 367

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 367

MULTIPLICATION:

+1X<sup>2</sup>-2X<sup>1</sup>+4X<sup>0</sup>

+1X<sup>2</sup>+1X<sup>1</sup>-2X<sup>0</sup>

+1X<sup>4</sup>-1X<sup>3</sup>+8X<sup>1</sup>-8X<sup>0</sup>

NBRE DE MOTS DISPONIBLES: 12274

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 695

Temps inferieurs ou egaux à la seconde sur micro Aplle II

DIVISION:  
 $+1X^2+2X^1+1X^0$   
 $+1X^1+1X^0$

-----  
 $+1X^1+1X^0$   
 RESTE= NUL  
 NBRE DE MOTS DISPONIBLES: 15358  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 234  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 234

$$x^2 + 2x + 1 = (x+1)^2$$

DIVISION:  
 $+1X^4-1X^3+8X^1-8X^0$   
 $+1X^2+1X^1-2X^0$

-----  
 $+1X^2-2X^1+4X^0$   
 RESTE= NUL  
 NBRE DE MOTS DISPONIBLES: 14032  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 109  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 109

$$x^4 - x^3 + 8x - 8 = (x^2 + x - 3)(x^2 - 2x + 4)$$

DIVISION:  
 $+1X^4-1X^3+10X^1$   
 $+1X^2+1X^1-2X^0$

-----  
 $+1X^2-2X^1+4X^0$   
 RESTE=  $+2X^1+8X^0$   
 NBRE DE MOTS DISPONIBLES: 13684  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 224  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 224

$$x^4 - x^3 + 10x = (x^2 + x - 2)(x^2 - 2x + 4) + 2x + 8$$

DIVISION:  
 $+1X^5+1X^4+1X^3+1X^2+1X^1+26801X^0$   
 $+1X^0$

-----  
 $+1X^5+1X^4+1X^3+1X^2+1X^1+26801X^0$   
 RESTE= NUL  
 NBRE DE MOTS DISPONIBLES: 15559  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 161  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 161

$$2^5 + 2^4 + x^3 + 2^2 + x + 26801 = 1 \cdot (2^5 + 2^4 + x^3 + 2^2 + x + 1)$$

DIVISION:  
 $+3X^5+2X^4+1X^0$   
 $+1X^3+1X^1+2X^0$

-----  
 $+3X^2+2X^1-3X^0$   
 RESTE=  $-8X^2-1X^1+7X^0$   
 NBRE DE MOTS DISPONIBLES: 13330  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 341  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 341

$$3x^5 + 2x^4 + 1 = (3x^2 + 2x - 3)(x^3 + x + 1) + -7x^2 - x + 2$$

DIVISION:  
 $+1X^3+4X^2+6X^1+1X^0$   
 $+1X^1+1X^0$

-----  
 $+1X^2+3X^1+3X^0$   
 RESTE=  $-2X^0$   
 NBRE DE MOTS DISPONIBLES: 12742  
 SYSTEME GESTION LISTE: PASCAL

NMBRE DE CELLULE ACTUELLES: 539  
 NBRE DE CELL DONNEES: 0  
 NBRE DE CELL STOCKEES: 539

$$x^3 + 4x^2 + 6x + 1 = (x+1)(x^2 + 3x + 3) - 2$$

Temps moyen entre 1 s. et 2 s. sur micro Apple II

Temps moyen de 3 s. à 5 s. sur micro Apple II

PGCD:

$+1X^5-1X^3-1X^2+1X^0$

$+5X^3-6X^2-3X^1+4X^0$

$+1X^2-2X^1+1X^0$

NBRE DE MOTS DISPONIBLES: 14692

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 207

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 207

PGCD:

$+1X^{12}-1X^9-1X^8+1X^5$

$+1X^3-1X^2-1X^1+1X^0$

$+1X^3-1X^2-1X^1+1X^0$

NBRE DE MOTS DISPONIBLES: 13477

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 611

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 611

PGCD:

$+1X^2-3X^1+2X^0$

$+1X^2-1X^0$

$-1X^1+1X^0$

NBRE DE MOTS DISPONIBLES: 15133

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 64

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 64

PGCD:

$+4X^2-12X^1+8X^0$

$+5X^2-5X^0$

$-1X^1+1X^0$

NBRE DE MOTS DISPONIBLES: 15133

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 64

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 64

PGCD:

$+1X^3+2X^2-1X^1-2X^0$

$+1X^2-1X^1+1X^0$

$+1X^0$

NBRE DE MOTS DISPONIBLES: 14425

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 299

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 299

PGCD:

$+1X^3+2X^2-1X^1-2X^0$

$+7X^1+14X^0$

$+1X^1+2X^0$

NBRE DE MOTS DISPONIBLES: 14674

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 216

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 216

PGCD:

$+1X^3+1X^0$

$+1X^2-1X^1+1X^0$

$+1X^2-1X^1+1X^0$

NBRE DE MOTS DISPONIBLES: 14833

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 163

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 163

PGCD:

$+1X^2+1X^1+1X^0$

$+1X^1+1X^0$

$+1X^0$

NBRE DE MOTS DISPONIBLES: 14906

SYSTEME GESTION LISTE: PASCAL

NBRE DE CELLULE ACTUELLES: 114

NBRE DE CELL DONNEES: 0

NBRE DE CELL STOCKEES: 114



# CALCULS SUR LES POLYNOMES : EXEMPLES

ADDITION:

$$\begin{array}{r}
 +30X+12-20X+9+5X+7-10X+3+17X+2+168X+1-456X+0 \\
 +20X+10-5X+7+10X+3+13X+2-100X+1 \\
 \hline
 +30X+12+20X+10-20X+9+30X+2+68X+1-456X+0
 \end{array}
 \quad (1'')$$

MULTIPLICATION:

$$\begin{array}{r}
 +1X+3+1X+1+2X+0 \\
 +3X+2+2X+1-3X+0 \\
 \hline
 +3X+5+2X+4+8X+2+1X+1-6X+0
 \end{array}
 \quad (2'')$$

DIVISION:

$$\begin{array}{r}
 +1X+10-1X+0 \\
 +1X+1+1X+0 \\
 \hline
 +1X+9-1X+8+1X+7-1X+6+1X+5-1X+4+1X+3-1X+2+1X+1-1X+0 \\
 \text{RESTE= NUL}
 \end{array}
 \quad (7'')$$

DIVISION:

$$\begin{array}{r}
 +3X+5+2X+4+1X+0 \\
 +1X+3+1X+1+2X+0 \\
 \hline
 +3X+2+2X+1-3X+0 \\
 \text{RESTE= } -8X+2-1X+1+7X+0
 \end{array}
 \quad (4'')$$

DERIVATION:

$$\begin{array}{r}
 +2X+45-3X+30-1X+12-10X+8+3X+2+70X+1+987X+0 \\
 \text{SE DERIVE EN:} \\
 +90X+44-90X+29-12X+11-80X+7+6X+1+70X+0
 \end{array}
 \quad (1'')$$

PGCDPOL: EUCLIDE GENERALISE

$$\begin{array}{r}
 \text{PGCD:} \\
 +1X+5-1X+3-1X+2+1X+0 \\
 +5X+3-6X+2-3X+1+4X+0 \\
 \hline
 +1X+2-2X+1+1X+0
 \end{array}
 \quad (2'')$$

PGCDPOL: EUCLIDE GENERALISE

$$\begin{array}{r}
 \text{PGCD:} \\
 +1X+12-1X+9-1X+8+1X+5 \\
 +1X+3-1X+2-1X+1+1X+0 \\
 \hline
 +1X+3-1X+2-1X+1+1X+0
 \end{array}
 \quad (2'')$$

ENTRER LA VALEUR: 11

$$\begin{array}{r}
 +1X+2+2X+1+1X+0 \\
 \text{VALEUR NUMERIQUE: 144}
 \end{array}$$

( < 1'')

ENTRER LA VALEUR: -1

$$\begin{array}{r}
 +1X+2+2X+1+1X+0 \\
 \text{VALEUR NUMERIQUE: 0}
 \end{array}$$

( < 1'')

Les temps indiqués entre parenthèses ont été mesurés sur le micro-ordinateur 8 bits.

(CAS DU CALCUL DU PGCD)

REPRISE DES EXEMPLES IDENTIQUES AUX  
PRECEDENTS, MAIS APRES SATURATION  
DE L'ESPACE MEMOIRE DYNAMIQUE DU  
PASCAL

LE SYSTEME DE GESTION EST PALDÉS

-----

PGCD:

+1X↑3+2X↑2-1X↑1-2X↑0

+1X↑2-1X↑1+1X↑0

---

+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULES ACTUELLES: 4898

NBRE DE CELL DONNEES: 963

NBRE DE CELL STOCKEES: 5861

PGCD:

+1X↑3+2X↑2-1X↑1-2X↑0

+7X↑1+14X↑0

---

+1X↑1+2X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULES ACTUELLES: 4898

NBRE DE CELL DONNEES: 880

NBRE DE CELL STOCKEES: 5778

PGCD:

+1X↑3+1X↑0

+1X↑2-1X↑1+1X↑0

---

+1X↑2-1X↑1+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULES ACTUELLES: 4898

NBRE DE CELL DONNEES: 827

NBRE DE CELL STOCKEES: 5725

PGCD:

+1X↑2+1X↑1+1X↑0

+1X↑1+1X↑0

---

+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULES ACTUELLES: 4900

NBRE DE CELL DONNEES: 776

NBRE DE CELL STOCKEES: 5676

PGCD:

+1X↑5-1X↑3-1X↑2+1X↑0

+5X↑3-6X↑2-3X↑1+4X↑0

---

+1X↑2-2X↑1+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULE ACTUELLES: 4895

NBRE DE CELL DONNEES: 1639

NBRE DE CELL STOCKEES: 6534

PGCD:

+1X↑12-1X↑9-1X↑8+1X↑5

+1X↑3-1X↑2-1X↑1+1X↑0

---

+1X↑3-1X↑2-1X↑1+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULE ACTUELLES: 4894

NBRE DE CELL DONNEES: 1542

NBRE DE CELL STOCKEES: 6436

PGCD:

+1X↑2-3X↑1+2X↑0

+1X↑2-1X↑0

---

-1X↑1+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULE ACTUELLES: 4899

NBRE DE CELL DONNEES: 595

NBRE DE CELL STOCKEES: 5494

PGCD:

+4X↑2-12X↑1+8X↑0

+5X↑2-5X↑0

---

-1X↑1+1X↑0

NBRE DE MOTS DISPONIBLES: 628

SYSTEME GESTION LISTE: ALDES

NMBRE DE CELLULE ACTUELLES: 4899

NBRE DE CELL DONNEES: 727

NBRE DE CELL STOCKEES: 5626

<<<<< DEBUT INTERPRETEUR >>>>>  
 ANALYSE ET GENERATION DU PREPROCESSEUR  
 PASCAL UCSD-ALDES. VERSION 15/11/81

GLOBAL LISTE LI, L2, N;  
 GLOBAL I, P;

CONVMUENT(N:L=LISTE)  
 LOCAL I0, I1, I2;

```
(0) SI (N<10) ALORS ENTVERLS(N;L)
    SINON
    SI (N<100) ALORS
      % I1:=N/100;
      I0:=N-I1;
      ENTVERLS(I0;L);
      MET(1, I1;L)
    %
    SINON IMPRIMER"NOMBRE TROP GRAND!"
#
```

LISTING DU SOURCE PALDES compilé

-----  
 TABLE DES SYMBOLES PASSAGE N0: 1  
 =====

SECTION DES VARIABLES GLOBALES:  
 -----

IDENTIF	CODAGE	TYPE	NE/NS	STATU
I	A48	ENTIER	0 0	GLOBAL
L2	A46	LISTE	0 0	GLOBAL
LI	A45	LISTE	0 0	GLOBAL
N	A47	LISTE	0 0	GLOBAL
P	A49	ENTIER	0 0	GLOBAL

SECTION DES VARIABLES LOCALES:  
 -----

IDENTIF	CODAGE	TYPE	NE/NS	STATU
I0	A53	ENTIER	0 0	LOCAL
I1	A54	ENTIER	0 0	LOCAL
I2	A55	ENTIER	0 0	LOCAL
L	A52	LISTE	0 0	PARAM/OUT
N	A51	ENTIER	0 0	PARAM/IN

SECTION DES ALORITHMES SYSTEMES:  
 -----

IDENTIF	CODAGE	TYPE	NE/NS	STATU
ABSLS	A40	FONC-SYS	1 1	INTRINSE
ADDITION	A44	FONC-SYS	2 1	INTRINSE
ADDPOL	A23	FONC-SYS	2 1	INTRINSE

AFFECT	A22	FONC-SYS	1	1	INTRINSE
CHER	A6	FONC-SYS	1	2	INTRINSE
COMPLS	A41	FONC-SYS	2	1	INTRINSE
CONT	A28	FONC-SYS	1	1	INTRINSE
COPIE	A3	FONC-SYS	1	1	INTRINSE
CUMUL	A16	FONC-SYS	2	1	INTRINSE
DEGRE	A12	FONC-SYS	1	1	INTRINSE
DERIVE	A32	FONC-SYS	1	1	INTRINSE
DESALL	A13	FONC-SYS	1	0	INTRINSE
DIVISION	A42	FONC-SYS	2	2	INTRINSE
DIVPOL	A26	FONC-SYS	2	3	INTRINSE
ECRLS	A8	FONC-SYS	1	0	INTRINSE
ECRNBR	A36	FONC-SYS	1	0	INTRINSE
EFFEL	A9	FONC-SYS	1	1	INTRINSE
EFFLS	A17	FONC-SYS	0	1	INTRINSE
ENTVERLS	A37	FONC-SYS	1	1	INTRINSE
ETAT	A15	FONC-SYS	1	1	INTRINSE
INV	A4	FONC-SYS	1	1	INTRINSE
LAMBDA P	A25	FONC-SYS	3	1	INTRINSE
LIRLS	A7	FONC-SYS	0	1	INTRINSE
LIRNBR	A35	FONC-SYS	0	1	INTRINSE
LIRPOL	A34	FONC-SYS	0	1	INTRINSE
LONG	A11	FONC-SYS	1	1	INTRINSE
MET	A5	FONC-SYS	2	1	INTRINSE
MULTIPLIC	A43	FONC-SYS	2	1	INTRINSE
MULTPOL	A24	FONC-SYS	2	1	INTRINSE
OPPLS	A39	FONC-SYS	1	1	INTRINSE
PERIF	A10	FONC-SYS	1	0	INTRINSE
PERIF	A10	FONC-SYS	1	0	INTRINSE
PGCDPOL	A30	FONC-SYS	2	1	INTRINSE
PP	A29	FONC-SYS	1	1	INTRINSE
PPCM	A20	FONC-SYS	2	1	INTRINSE
PREM	A1	FONC-SYS	1	2	INTRINSE

PRIMPOL	A33	FONC-SYS	1	2	INTRINSE
PUISS	A21	FONC-SYS	2	1	INTRINSE
REDUC	A2	FONC-SYS	0	1	INTRINSE
RESTE	A18	FONC-SYS	2	1	INTRINSE
RESTEPOL	A27	FONC-SYS	2	1	INTRINSE
SGNLS	A38	FONC-SYS	1	1	INTRINSE
SYST	A14	FONC-SYS	0	0	INTRINSE
VALNUM	A31	FONC-SYS	2	1	INTRINSE

\*\*\*\*\*  
 TABLE DES ETIQUETTES:  
 =====

SECTION ETIQUETTES INTERNES:

.... AUCUNE ....

SECTION ETIQUETTES EXTERNES:

.... AUCUNE ....

FACTOR(<)  
 LOCAL J;  
 LOCAL CARAC T;

- (1) I:=1;  
 REPETER  
 CONVMUENT(I;LI);  
 MULTIPLICATION(N,LI;L2);  
 AFFECT(L2;N);  
 I:=I+1  
 JUSQUE (I=P+1).
- (2) POUR J=1,...,60 FAIRE IMPRIMER\$"-";  
 IMPRIMER.
- (3) IMPRIMER\$"FACTORIELLE ":P:" = ";  
 ECRNBR(N)

#

-----  
 TABLE DES SYMBOLES PASSAGE N0: 2  
 =====

SECTION DES VARIABLES LOCALES:

IDENTIF	CODAGE	TYPE	NE/NS	STATU
J	A57	ENTIER	0 0	LOCAL
T	A58	CARACTERE	0 0	LOCAL

\*\*\*\*\*  
 TABLE DES ETIQUETTES:  
 =====

SECTION ETIQUETTES INTERNES:

.... AUCUNE ....

-----  
PRINCIP.

(0) PERIF(0).

(1) IMPRIMER"ENTREZ LA VALEUR A CALCULER",  
ENTRER P;  
ENTVERLS(1;N).

(2) FACTOR(<)

#

-----  
TABLE DES SYMBOLES PASSAGE N0: 3

=====

SECTION ALGORITHMES:

-----

IDENTIF	CODAGE	TYPE	NE/NS	STATU
AFFECT	A22 ALGORITHM.		1 1	INTRINSE
CONVNUENT	A50 ALGORITHM.		1 1	PARAM/OUT
ECRNBR	A36 ALGORITHM.		1 0	INTRINSE
ENTVERLS	A37 ALGORITHM.		1 1	INTRINSE
FACTOR	A56 ALGORITHM.		0 0	GLOBAL
MET	A5 ALGORITHM.		2 1	INTRINSE
MULTIPLIC	A43 ALGORITHM.		2 1	INTRINSE
PERIF	A10 ALGORITHM.		1 0	INTRINSE
PRINCIP	A59 ALGOPRINC		0 0	GLOBAL

-----

SECTION DES VARIABLES LOCALES:

-----

.... NEANT ....

\*\*\*\*\*

TABLE DES ETIQUETTES:

=====

SECTION ETIQUETTES EXTERNES:

-----

.... AUCUNE ....

#

\*\*\*\*\*

AUCUNE ERREUR DECELEE.

\*\*\*\*\*

>>>>> FIN INTERPRETEUR



LISTING DU "CODE Pascal" engendré

par le compilateur PALDES

```
PROGRAM INTERPRETE;  
(*$G+*)  
(*$S+*)  
(*$U #5:LIBRALDES *)  
USES SYSTALDES, POLALDES, PRECINFINI;  
VAR YBAR0, YBAR1, ZBAR0:ARRAY[1..20] OF INTEGER;  
A45, FDLA45:LISTE;  
A46, FDLA46:LISTE;  
A47, FDLA47:LISTE;  
  
A48:INTEGER;  
A49:INTEGER;  
  
PROCEDURE A50(A51:INTEGER; VAR A52, FDLA52:LISTE);  
VAR RE:INTEGER;  
A53:INTEGER;  
A54:INTEGER;  
A55:INTEGER;  
  
BEGIN  
IF A51<10 THEN  
A37(A51, A52, FDLA52)  
ELSE  
IF A51<100 THEN  
BEGIN  
A54:=A51 DIV 100;  
A53:=A51-A54;  
A37(A53, A52, FDLA52);  
A5(1, A54, A52, FDLA52);  
END  
ELSE  
WRITELN(F, 'NOMBRE TROP GRAND!');  
;  
;  
END;
```

```

PROCEDURE A56;
VAR RE: INTEGER;
A57: INTEGER;

A58: CHAR;

BEGIN
A48:=1;
REPEAT
A50(A48, A45, FDLA45);
A43(A47, FDLA47, A45, FDLA45, A46, FDLA46);
A22(A46, FDLA46, A47, FDLA47);
A48:=A48+1;
UNTIL A48=A49+1;
A57:=1;
YBAR0[1]:=1;
IF YBAR0[1]=0 THEN YBAR1[1]:=0
ELSE
IF YBAR0[1]>0 THEN YBAR1[1]:=1
ELSE
YBAR1[1]:=-1;
ZBAR0[1]:=60;
WHILE YBAR1[1]*A57<=YBAR1[1]*ZBAR0[1] DO
BEGIN
WRITE(F, '-');
;
A57:=A57+YBAR0[1]
END
;
Writeln(F);
WRITE(F, 'FACTORIELLE ', A49, ' = ');
A36(A47, FDLA47);
END;
BEGIN
PAGE(OUTPUT);
STAND;
Writeln(F, 'PALDES VERSION 15/11/81');
Writeln(F);
INITLIST(A46, FDLA46);
INITLIST(A45, FDLA45);
INITLIST(A47, FDLA47);
A10(0);
Writeln(F, 'ENTREZ LA VALEUR A CALCULER');
READLN(A49);
A37(1, A47, FDLA47);
A56;
END.

```

EXECUTION après compilation  
du programme par le système  
Pascal

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 10 = 3628800

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 11 = 39916800

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 12 = 479001600

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 13 = 6227020800

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 20 = 2432902008176640000

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 21 = 51090942171709440000

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 22 = 112400072777607680000

ENTREZ LA VALEUR A CALCULER

-----  
FACTORIELLE 23 = 25852016738884976640000

Dernière page d'une thèse

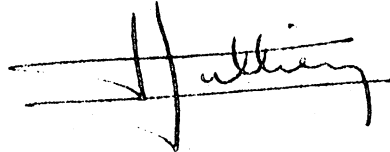
---

Di SCALA Robert - Michel

VU

Grenoble, le

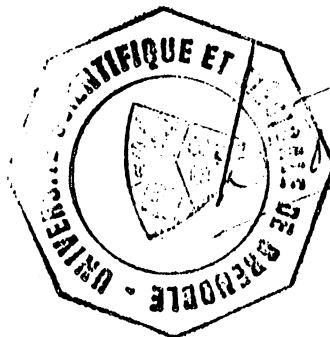
Le Président de la thèse



Vu, et permis d'imprimer,

Grenoble, le 16 mars 1982

Le Président de l'Université Scientifique et Médicale



*Tanche*  
Pour le Président,  
**M. TANCHE**  
Vice-Président Assesseur

