UNIVERSITE DE NANCY I

U.E.R. DE SCIENCES MATHEMATIQUES

DEFINITION DE FACE

LANGAGE POUR L'ECRITURE DES COMPILATEURS

IMPLEMENTATION D'UN SOUS-ENSEMBLE

THESE

présentée à l'UER des Sciences Mathématiques

pour l'obtention du grade de

DOCTEUR INGENIEUR

par

JEAN MAROLDT

soutenue le 28 novembre 1972

devant la Comission d'Examen

Jury :

M. J. LEGRAS	,	Président
M. M. DEPAIX	,	Examinateur
M. C. PAIR	,	Examinateur
M. J.C. DERNIAME	,	Examinateur

Compléments

La définition du langage Face a été présentée au Congrés de l'Afcet en septembre 1970.

Une première version du compilateur $Face_0$ était opérationnelle en décembre 1971.

occupation mémoire du compilateur $Face_0$:	5 K-mots

Le compilateur Face, écrit en $Face_0$, comprend à peu près 1800 cartes. Par carte, il y a en moyenne : 1 appel de procédure, une affectation et 1 a 2 expressions simples.

Temps de compilation de ces 1800 cartes (sur CII 10070) :	temps attente E/S : 4.0 mn temps E/S : 0.42 " temps exécution unité centrale : 0.55 "
Temps de métaassemblage de ce programme généré par $Face_0$:	48 minutes

Le compilateur Face ainsi généré occupe près de 20 K-mots (sans les fonctions standard et les tables d'identificateurs).

Le choix du langage métassembleur comme langage intermédiaire se justifie dans la mesure où le compilateur $Face_0$ ne sert qu'une fois pour générer le compilateur Face. Il a permis d'éviter une étude lexicographique avec construction de tables d'identificateurs. Il suffisait de générer un marqueur au niveau de la compilation. C'est le métaassembleur qui en fonction de ce marqueur générait le code objet relatif aù type de l'identificateur (global, paramètre formel, etc). Par ailleurs il permettait une grande autonomie entre le langage engendré et son exécution.

UNIVERSITE DE NANCY I

U.E.R. DE SCIENCES MATHEMATIQUES

DEFINITION DE FACE

LANGAGE POUR L'ECRITURE DES COMPILATEURS

IMPLEMENTATION D'UN SOUS-ENSEMBLE

par JEAN MAROLDT

Que Monsieur le Professeur LEGRAS, Directeur de l'Institut Universitaire de Calcul Automatique, trouve ici l'expression de ma profonde gratitude pour son acceuil à l'Institut et l'honneur qu'il me fait en présidant ce jury.

Ce travail a été effectué sous la direction de Monsieur le Professeur PAIR à qui j'exprime mes plus vifs remerciements pour la formation qu'il m'a donnée et pour les conseils qu'il n'a cessés de m'apporter tout au long de mon travail.

Je remercie Monsieur le Professeur DEPAIX, Président de l'Unité d'Enseignement et de Recherche des Sciences Mathématiques à l'Université de Nancy I, et Monsieur DERNIAME pour l'honneur qu'ils me font en acceptant de participer au jury.

De même, je remercie M. H. PISTRE pour la partie analyse syntaxique du projet Face ainsi que Mme F. BELLEGARDE et Melle A. M. BOUCHET pour l'implémentation de ce langage.

Je tiens également à remercier toute l'équipe de l'Institut de Calcul Automatique et plus particulièrement Melle D. COLIN qui a réalisé matériellement ce travail.

SOMMAIRE

INTRODUCTION

Le langage Face est conçu pour l'écriture des compilateurs. Ses objectifs sont de guider et de faciliter le travail de l'utilisateur tout en lui laissant une grande souplesse dans les choix importants, tel que le nombre de passages que peuvent avoir les compilateurs écrits dans ce langage. De même le programmeur peut faire appel une ou plusieurs fois à l'analyse syntaxique, ou alternativement aux parties analyse et générationEn outre ce langage permet la définition de plusieurs grammaires.

Une première définition de Face a été présentée à l'un des congrés de l'Afcet [10] . Depuis quelques modifications ont été apportées au langage. Elles concernent essentiellement la définition des rangées de procédures. La nouvelle version est exposée au chapitre 1.

Les réalisations dans le cadre du projet Face comprennent le compilateur du langage [1] , le programme d'analyse syntaxique [12] et plusieurs programmes de service (chap 5).

Afin de respecter la notion de compilateur de compilateur qui est à la base de ce projet, le compilateur Face lui-même est réalisé dans un langage, sous-ensemble de Face, appelé $Face_0$ (chapitre 2).

$Face_0$ lui-même est implémenté en langage machine (Symbol 10070). Le compilateur a un seul passage et le langage objet généré est une suite d'appels de procédures de métaassemblage (Métasymbol). La description de cette implémentation constitue la partie la plus importante du présent travail (chap. 3 et 4).

Parmi les programmes de service (chap. 5) se trouvent notamment des programmes de gestion de tables. En effet, la compilation nécessite l'emploi de tables et prennent une part importante dans le temps de compilation. Pour réduire au minimum ce temps de gestion des tables, les programmes correspondants sont écrits en langage machine.

CHAPITRE I

DEFINITION DU LANGAGE FACE

1. DECRIPTION GENERALE

1.1. Les objectifs du langage

Comme la plupart des autres langages permettant l'écriture de compilateurs (voir par exemple [3], [4], [7], [17], [13], [14]) FACE s'applique à des langages dont la syntaxe est définie par une grammaire à contrexte libre [2]. Un programme écrit en FACE[1] contient la description d'une telle grammaire G et il peut décrire, grâce à un certain nombre de procédures, une traduction du langage engendré par G ; sa donnée est donc une chaîne de caractères (texte source) et son résultat peut être une traduction de cette chaîne (texte objet) ou un certain nombre de messages d'erreur si la chaîne donnée n'appartient pas au langage engendré par G ou si elle ne satisfait pas à certaines conditions supplémentaires.

La compilation comporte l'analyse syntaxique du texte source (avec éventuellement messages d'erreur du premier type) et la génération du texte objet et des messages d'erreur du second type ; la génération demande certaines opérations de service, comme la construction de tables d'identificateurs (étude lexicographique). L'analyse syntaxique et la génération peuvent ou non être séparées dans le temps ; de même, la génération peut être effectuée en un ou plusieurs passages, un passage pouvant par exemple être réservé à l'étude lexicographique.

On peut donner à un système d'écriture de compilateurs les objectifs suivants :

a) Le programmeur doit définir son compilateur sans avoir à se préoccuper en aucune façon de l'analyse syntaxique. La première conséquence est que l'analyse doit être effectuée automatiquement, par une procédure (l'analyseur) faisant partie du système, dont les données sont la grammaire considérée et la chaîne à analyser. Il en est ainsi pour tous les langages d'écriture de compilateurs, dont chacun est fondé sur un algorithme d'analyse. Mais il semble souhaitable d'aller plus loin : le programmeur ne doit

---

(1) On peut penser que FACE signifie "fabrication automatique de compilateurs efficaces". Mais ce nom ne vient-il pas plutôt du fait que FACE utilise une pile ?

pas être obligé de connaître la méthode d'analyse, qui pourrait même dépendre du langage étudié et être choisie automatiquement ; en tout cas, la méthode d'analyse doit être sans influence sur la description de la génération du texte objet. Par exemple, il arrive fréquemment qu'une analyse syntaxique "ascendante" soit plus efficace, alors que la génération se conçoit plus commodément de manière "descendante". On choisira donc la sortie de l'analyseur la plus commode pour la génération, et cette sortie sera indépendante de l'algorithme d'analyse.

b) La génération doit être décrite aisément et clairement. Pour réaliser cet objectif, le plus simple est d'associer des procédures de génération aux diverses règles de la grammaire ; l'ordre dans lequel ces procédures sont appelées, qui dépend du résultat de l'analyse syntaxique, ne doit pas avoir à être spécifié par le programmeur : le système doit prendre en charge le résultat de l'analyse et les procédures, pour commander la succession des appels ; il contient pour cela un programme de commande.

c) Le système doit fournir au programmeur des outils convenables, définis clairement et indépendamment de leur représentation en machine. Les chaînes de caractères sont bien entendu nécessaires mais on doit disposer aussi de tables. Une table peut être considérée comme une fonction à un argument (l'"indicatif") dont la valeur est une "adresse", ou un tableau d'adresses. Pour préciser cette notion d'adresse, nous introduirons en FACE des repères, semblables aux "noms" de [11]. Ces repères permettent aussi le traitement de listes, lorsque c'est nécessaire [13].

d) Une grande souplesse doit être laissée à l'utilisateur. Par exemple, on restreint considérablement le domaine d'emploi en ne décrivant, comme certains systèmes, que les compilations en un seul passage. Au contraire il faut pouvoir décrire un ou plusieurs passages, séparés ou non de l'analyse syntaxique, sans refaire à chacun d'eux cette analyse ou au contraire en utilisant des grammaires différentes aux divers passages : par exemple, l'analyse lexicographique peut être guidée par une grammaire beaucoup plus pauvre que celle qui doit guider la génération.

e) Enfin, le compilateur obtenu doit être efficace. Cette efficacité tient notamment à la manière dont est exploité le résultat de l'analyse par le

programme de commande (cf. b) et donc à la forme que possède la sortie de l'analyseur. La difficulté est de concilier la souplesse, la clarté et la facilité d'écriture avec l'efficacité ; pour cela, il convient de privilégier les manières de travailler les plus fréquentes.

1. 2. Examen du processus de compilation

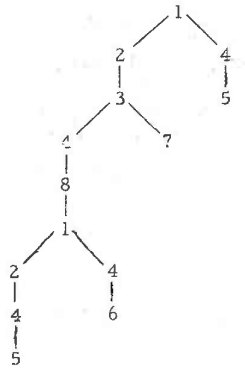
Le résultat de l'analyse syntaxique est une ramification à une racine. Cette notion est définie en [8], [9] ; bornons-nous à la suggérer en disant qu'il s'agit d'une arborescence, orientée de gauche à droite, dont chaque noeud est étiqueté (exemple 1) ; ici, on supposera que les étiquettes sont des numéros de règles de la grammaire pour laquelle est faite l'analyse.

Exemple 1 :

Soit la grammaire G dont les règles (numérotées 1, 2 ...) s'écrivent :

E ::= E + T / 1 / T / 2
T ::= T x F / 3 / F / 4
F ::= a / 5 / b / 6 / c / 7 / (E) / 8

avec pour axiome E, elle engendre des expressions arithmétiques employant les opérateurs x et + et les variables a, b, c . Par exemple (a + b) x c + a appartient au langage engendré et la ramification engendrée est alors :



Une telle ramification r est composée d'une racine N numéro de règle et de p ramifications à une racine $r_1 \ldots r_p$.

Un passage du compilateur qui utilise la ramification r, résultat de l'analyse peut être décrit par une procédure g qui admet r en paramètre [11].

. La procédure g est récursive ; $g(r)$ dépend de $g(r_1), \ldots, g(r_p)$.

Pour l'exemple ci-dessus :

- $g(r_1)$ admet un résultat x_1 : adresse et type du résultat intermédiaire $(a+b) \times c$.
- $g(r_2)$ admet un résultat x_2 : adresse et type de la variable a.

Le calcul $g(r)$ peut être exprimé par :

$x_1 := g(r_1)\ ;\ x_2 := g(r_2)\ ;\ \varphi_1(x_1\ ;\ x_2)\ ;$

où φ_1 est une procédure associée à la règle numéro 1 utilisant en paramètre les résultats x_1, x_2 et fournissant le résultat de $g(1)$. Ceci se généralise ; dans la plupart des cas on compile toutes les ramifications $r_1 \ldots r_p$ à une racine composant $r = N \times (r_1 + \ldots + r_p)$, avant d'utiliser leur résultat :

$g(r) = (x_1 := g(r_1)\ ; \ldots ; x_p := g(r_p)\ ; \varphi_N(x_1 \ldots x_p))$

Il n'en est pas ainsi cependant quand on doit effectuer des actions entre les compilations des ramifications composantes $r_1 \ldots r_p$. Prenons par exemple à la compilation d'une instruction conditionnelle engendrée par la règle :

C ::= si E alors I sinon S/1

Soient r_E, r_I, r_S les ramifications issues de E, I, S.

On a $g(r) = (x := g(rE)\ ;\ e := h(x)\ ;\ g(rI)\ ,\ e' := h'(x)\ ;\ g(rS)\ ;\ h''(e'))$

h est une procédure qui teste si E est de type booléen, génère un branchement conditionnel dont on retient l'étiquette e ;
h' génère un saut inconditionnel dont on retient l'étiquette e' et génère la définition de e ;
h'' génère la définition de e'.

On peut se ramener au cas précédent en décomposant la règle 1

C ::= C' sinon S	$g(r_B) = (x := g(r_E)\ ;\ h(x))$
C' ::= B alors I	$g(r_c) = (e := g(r_B)\ ;\ g(r_I)\ ;\ h'(x))$
B ::= si E	$g(r) = (e' := g(r_c)\ ;\ g(r_S)\ ;\ h''(e'))$

$g(r_I)$ et $g(r_S)$ sont sans résultat.

On montre [11] que ceci est général et qu'on peut se ramener au cas précédent.

. Pour réaliser le calcul de $g(r)$ de façon pratique, il faut supprimer la récursivité de la procédure g :

Il faut sauvegarder x_1 avant de calculer $g(r_2)$, qui risque de le détruire ; de même, il faut sauvegarder $x_2 \ldots x_{p-1}$. Pour cela on utilise une pile : φ_N prendra ses p paramètres sur la pile et y replacera son résultat.

Le calcul de g est alors très facile si on représente la ramification r sous forme post fixée [11]. Bornons-nous ici à un exemple, la ramification de l'exemple 1 est représentée sous cette forme par

5 4 2 6 4 1 8 4 7 3 2 5 4 1

et pour calculer g(r), on aura à exécuter successivement

$\varphi_5 \varphi_4 \varphi_2 \varphi_6 \varphi_4 \varphi_1 \varphi_8 \varphi_4 \varphi_7 \varphi_3 \varphi_2 \varphi_5 \varphi_4 \varphi_1$.

(certaines procédures φ_i) n'ont aucune action).

. Une difficulté subsiste : les paramètres et le résultat des procédures φ_k n'ont pas un format fixe : la compilation d'un opérande d'une expression arithmétique donne un résultat adresse type, celle d'une instruction est sans résultat ... On peut se ramener le plus souvent à un format unique en admettant que chaque φ_k peut avoir un nombre quelconque de résultats :

φ_k est une procédure prenant ses d_k arguments sur une pile et les y remplaçant par s_k résultats. d_k et s_k sont des entiers liés à φ_k. En Face a:guments et résultats seront tous du même type, éléments : un élément est représenté par le contenu d'un mot de mémoire. Il existe cependant des cas où on ne peut ainsi lier à φ_k un nombre fixe d'argument:

Considérons par exemple les déclarations de tableaux d'Algol 60 engendrés par les règles :

<déclaration de tableaux> ::= <type de tableau><section de tableaux> /1
<déclaration de tableaux> , <section de tableaux> /2
<section de tableaux> ::= <suite d'identificateurs> [<paires de bornes>] /3
<suite d'identificateurs> ::= <identificateur> /4
<suite d'identificateur> , <identificateur> /5

Il faut conserver tous les identificateurs de la suite d'identificateurs avant de connaître à la fois les paires de bornes et le type, renseignements communs à tous les identificateurs.

Or les résultats de la procédure g ayant pour paramètres les ramifications qui engendrent la suite d'identificateurs sont en nombre variable selon le nombre d'identificateurs de la suite :

Le nombre d'arguments des procédures φ_1 et φ_2 devrait donc varier. On considère donc un autre type d'argument formé d'une liste d'éléments, ou rangée d'éléments : le dernier argument d'une procédure φ_N pourra être une rangée d'éléments placée au sommet de la pile et appelée rangée en pile.

2. LE FONCTIONNEMENT D'UN COMPILATEUR ECRIT EN FACE

L'analyseur de FACE est une procédure standard nommée analyse, qui a pour résultat la représentation postfixée d'une ramification, c'est à dire une liste, ou rangée d'entiers. Les fonctions φ_N introduites ci-dessus dépendent de l'entier N : elles forment une rangée de procédures. Il existe une procédure standard nommée appl, qui applique les procédures d'une rangée, dans l'ordre spécifié par une autre rangée formée d'entiers qui sont les indices successifs des procédures, les arguments étant pris et les résultats placés sur une même pile. Les procédures standard analyse et appl peuvent être combinées en une autre procédure standard anapplyse, qui effectue appl au fur et à mesure de la construction du résultat d'analyse, tout au moins lorsque l'analyse est déterministe.

Il suffit donc à l'auteur du compilateur d'écrire la rangée de procédures φ_N associées aux diverses règles ; il n'a pas besoin à ce moment d'avoir présente à l'esprit la manière dont est exécutée appl. Les procédures analyse, appl, anaplyse, peuvent être appelées plusieurs fois dans un même programme pour réaliser plusieurs passages du compilateur. L'écriture est donc très modulaire.

3. LES TYPES D'INFORMATION TRAITES PAR FACE

Le premier type d'information est formé par les entiers ou éléments. Extérieurement, un entier peut être représenté dans diverses bases de numération, par exemple 10, 16, 8, 2, et aussi par une chaîne de caractères : pour une implémentation déterminée, on précisera la correspondance entre chaînes de caractères et entiers, ainsi que la valeur absolue maximum des entiers. Si besoin est, l'entier 0 servira de valeur logique FAUX, tout autre entier servant de valeur logique VRAI. On pourra faire des opérations arithmétiques d'addition, de soustraction, de multiplication, de division, des comparaisons d'éléments et des opérations booléennes ET, OU, NON sur des valeurs logiques.

A côté des éléments existeront les repères (cf. la notion de pointeur et les "noms" de [16]) : à chaque instant, un repère R peut repérer un élément : après l'affectation à R d'un élément E, R repère E jusqu'à la prochaine affectation à R.

Face permet l'utilisation de rangée d'éléments et de rangées de repères, qui sont des tableaux unidimensionnels de borne inférieure égale à 1. Une composante d'une rangée t est déterminée par son indice i : t[i] est donc un élément ou un repère selon que t est une rangée d'éléments ou de repères. Une rangée de repères t est dite repérer la rangée d'éléments t1 telle que, pour tout indice i, t[i] repère t1[i] . En particulier, la procédure standard analyse affecte une rangée d'éléments à une rangée de repères.

Un programme FACE est essentiellement formé de procédures qui peuvent être sans résultat ou avoir comme résultat un élément ou un repère, ou encore, dans le cas de procédures standard ou écrites en code machine, une rangée d'éléments ou de repères. Le passage d'un compilateur traduisant un langage L en un langage L' à un compilateur traduisant L en

un autre langage L" devrait, dans bien des cas, pouvoir être effectué en réécrivant seulement un petit nombre de procédures. Les procédures appl et anapplyse ont pour paramètre une rangée de procédures.

La notion de repère permet de clarifier le passage des paramètres aux procédures [16] notamment "d'appel par nom" : si un paramètre est un repère, la procédure peut modifier l'élément qu'il repère. En particulier, une procédure φ_N d'une rangée de procédures modifie en général le contenu des cellules au sommet de la pile : ces cellules sont des paramètres de type repère ; au cas où l'un des paramètres est une rangée en pile, il est du type rangée de repères.

Face permet l'utilisation et la construction d'un certain nombre de tables, par exemple de tables d'identificateurs. Dans une telle table les indicatifs sont les identificateurs qui permettent l'accès aux renseignements correspondants : type et adresse

identificateur ⟶ (adresse, type)

Pour gagner du temps à la consultation qui est très fréquente dans ce type de table, on utilise les techniques de hashing pour les stocker :

Un certain nombre de procédures standard appelées procédures table (table, table1 . . .) ont pour résultat une rangée de repères qui est associée à un indicatif, qui est une rangée d'éléments. En réalité dans un langage à blocs, les identificateurs peuvent admettre plusieurs définitions chacune valable dans un bloc. Aussi on peut associer plusieurs entrées à un même indicatif, une procédure table à 2 arguments :

- une rangée d'éléments qui est l'indicatif ,
- un entier indiquant le rang d'apparition de l'entrée sélectionnée parmi celles qui possèdent cet indicatif.

Dans la pratique, une procédure table sera utilisée comme paramètre effectif d'autres procédures standard permettant d'effectuer des opérations sur les tables : adjonction, recherche, suppression, modification. Ces procédures admettent en paramètre une procédure à résultat booléen qui permet de sélectionner l'entrée voulue (filtre) :

Si l'on considère par exemple un langage à structure de blocs et que l'on utilise la table dans un passage ultérieur à sa construction, on peut numéroter les blocs dans l'ordre de leur apparition et associer à chaque déclaration son numéro de bloc.

A chaque consultation, il faut associer à un identificateur sa dernière déclaration appartenant à un bloc ouvert. Les déclarations seront placées dans la table dans l'ordre inverse de leur apparition ; le rôle de la procédure filtre est de tester si le bloc d'une déclaration est actif. On va donc chercher dans la table la première occurence de l'identificateur (indicatif) donnant à la procédure filtre la valeur vrai.

Tout ceci est permis par les deux procédures suivantes :

- recherche (table, indic, filtre) a pour résultat la rangée de repères table (indic, i) où i est le premier entier tel que la rangée repérée par table (indic, i) vérifie (c'est à dire donne un résultat non nul) à la procédure filtre, s'il existe un tel entier sinon appelle la procédure erreur.

- entrée (table, indic, inf, filtre) est un appel de procédures sans résultat qui :
  . s'il n'existe aucun entier j tel que la rangée repérée par table (indic, i) vérifie la procédure booléenne filtre, affecte la rangée d'éléments inf à table (indic, i) où i est le premier entier tel que table (indic, i) ne repère aucune rangée ;
  . s'il existe un tel entier j et si j_0 est le premier d'entre eux pour $i > j_0$ affecte à table (indic, i) la rangée repérée par table (indic, i-1) et affecte inf à table (indic, j_0).

Les procédures écrites en code Machine permettent de compléter ou d'adapter l'ensemble des procédures standard ; on peut par exemple vouloir décrire des passages plus simples qui n'utilisent pas de grammaire et se contentent de traiter séquentiellement le texte sans employer de pile.

4. DESCRIPTION DU LANGAGE FACE

4.1. Eléménts de base du langage

4.1.1. Symboles de base

a) <lettre>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
b) <chiffre octal>::=0|1|2|3|4|5|6|7
c) <chiffre décimal>::=<chiffre octal>|8|9
d) <chiffre hexadécimal>::=<chiffre décimal>|a|b|c|d|e|f
e) <vide>::=

4.1.2. Chaînes

a) <chaîne>::=<caractère>|<chaîne><caractère>
b) <caractère>::=<lettre>|<chiffre décimal>|<caractère spécial>
c) <caractère spécial>::="|:|.|,|;|(|)|[|]|+|-|x|/|<|≤|>|≥|=|≠|$|#|␣
d) <chaîne fermée>::='<chaîne>'

Sémantique : Les chaînes servent à définir les symboles terminaux des grammaires (4.3) et des quantités de type élément (4.1.5). Elles sont alors mises entre apostrophes, de sorte que ' ne peut être un caractère spécial ; mais " est un caractère spécial. D'autres caractères spéciaux peuvent être ajoutés à volonté pour une implémentation particulière. Le blanc n'est pas significatif dans un programme FACE. Le blanc dans une chaîne sera représenté par le caractère ␣ . Le changement de ligne dans le texte source est noté $ (4.3).

4.1.3 - Entiers

a) <entier>::=<entier octal>|<entier décimal>|<entier hexadécimal>

b) <entier octal>::=$<chiffre octal>|<entier octal><chiffre octal>

c) <entier décimal>::=<chiffre décimal>|<entier décimal>
<chiffre décimal>

d) <entier hexadécimal>::=@<chiffre hexadécimal>|
<entier hexadécimal><chiffre hexadécimal>

Sémantique : Un entier peut être utilisé pour représenter une constante de type élément (4.1.5), comme indice de rangée (4.7.4), et comme numéro de règle (4.3).

4.1.4 - Identificateurs

a) <identificateur>::=<ident>|<idres>

b) <idres>::=#<chiffre>|<idres><chiffre>

c) <ident>::=<lettre>|<ident><lettre>|<ident><chiffre>

Sémantique : Un identificateur peut désigner un élément, une rangée d'éléments, un repère, une rangée de repères (4.4), une procédure (4.5), une rangée de procédures (4.6) ou un symbole non terminal d'une grammaire (4.3). Un repère permet d'accéder à une valeur (4.6.4). Cette valeur peut être modifiée par une instruction d'affectation (4.7.2). Certains paramètres formels des procédures formant une rangée de procédures devront être désignés par les identificateurs particuliers #1, #2, ... formés du caractère # suivi d'un entier décimal (4.6).

4.1.5 - Eléments

a) <élément>::=<entier>|<chaîne fermée>|nil

Sémantique : Un élément est représenté par un entier (4.1.3) ou une chaîne fermée (4.1.2) ou la valeur nil (4.6). Pour chaque implémentation du langage, on précisera :
- l'entier maximum et la longueur maximum d'une chaîne pouvant représenter un élément,
- le passage d'une chaîne à l'entier représentant la même valeur.

4.2 - Programme

a) <programme>::=<définition des grammaires><liste de déclarations>
<expression fermée>

Sémantique :
La définition des grammaires (4.3) sert à décrire la syntaxe des langages étudiés. La liste de déclaration (4.4) sert à déclarer les identificateurs dont la portée est le programme.
L'expression fermée (4.7) qui termine le programme est sans résultat.

4.3 - Définition des grammaires

a) <définition des grammaires>::=<suite de grammaires>

b) <suite de grammaires>::=<vide>|<suite de grammaires><grammaire>

c) <grammaire>::=grammaire <suite de règles>

d) <suite de règles>::=<règle>|<suite de règles><règle>

e) <règle>::=<non terminal>:<second membre>;|:
<liste de caractères à sauter>

f) <non terminal>::=<identificateur>

g) <second membre>::=<membre>/<numéro>|<second membre>/
<membre>/<numéro>

h) <numéro>::=<vide>|<entier décimal>

i) <membre>::=<terminal>|<non terminal>|<membre>,<terminal>|
<membre>,<non terminal>

j) <terminal>::=<chaîne fermée>

k) <liste des caractères à sauter>::='caractère'/<numéro>|$/<numéro>
<liste des caractères à sauter>'caractère'/<numéro>|
<liste des caractères à sauter>$/<numéro>

Sémantique : Les grammaires sont exprimées en notation de Backus où ::= est remplacé par :, | par /<numéro>/ et où la dernière alternative est suivie de /<numéro>. De plus les symboles non terminaux sont des identificateurs, les terminaux sont placés entre guillemets, deux symboles étant séparés par une virgule et chaque règle suivie d'un point virgule. Les numéros servent à repérer les règles. Lorsqu'un numéro est sans importance pour le programme, il peut être laissé vide. Une grammaire est utilisée par la procédure standard analyse, qui possède deux paramètres :
- Un élément dont la valeur entière est un numéro de règle : la grammaire utilisée par l'analyse sera celle qui contient cette règle et l'axiome utilisé sera le premier membre de la règle ;
- Une rangée de repères à laquelle sera affectée le résultat de l'analyse.
La procédure effectue l'analyse de la chaîne de caractères lue sur un organe d'entrée fixé, pour la grammaire et l'axiome déterminés par son premier paramètre ; le résultat de l'analyse est une ramification sur l'ensemble des numéros de règles, qui est mise sous forme postfixée tout en supprimant les numéros de règle qui ont été laissés vides par le programmeur ; la rangée d'éléments obtenue est affectée au second paramètre. Tout non terminal ne doit apparaître qu'une fois comme premier membre d'une règle. Les règles de premier membre vide permettent de transmettre à la procédure analyse la liste des caractères non significatifs du texte source. Pour tout caractère non significatif, pour chaque fin de ligne (4.1.2) du texte source, la procédure analyse insère le numéro correspondant dans la ramification résultat. Il n'y a qu'un membre $ pour une grammmire donnée.

Exemple :

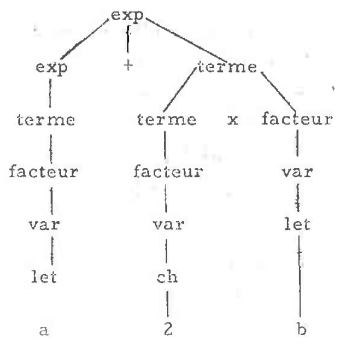
grammaire en notation de Backus

<exp>::=<terme>|<exp>+<terme>
<terme>::=<facteur>|<terme>x<facteur>
<facteur>::=<var>
<var>::=<let>|<ch>
<ch>::=2
<let>::=a|b

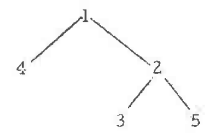
section des grammaires

```
grammaire
exp : terme// exp, '+', terme/1;
terme : facteur // terme, 'x',
        facteur/2 ;
facteur : var / ;
var : let // ch / ;
ch : '2' / 3 ;
let : 'a' / 4 / 'b' / 5 ;
```

Si la chaîne lue est a+2xb, l'analyse syntaxique pour cette grammaire a pour résultat :



Mais à cause des numéros vides, la procédure analyse 'simplifie' cette arborescence en



et le résultat est la rangée d'éléments 4, 3, 5, 2, 1.

4.4 - Liste de déclarations

a) <liste de déclarations>::=<vide>|<liste de déclarations><déclaration>

b) <déclaration>::=<déclaration d'élément>;|<déclaration de repère>;|
<déclaration de rangée d'éléments>;|
<déclaration de rangée de repère>
<déclaration de procédure>;|
<déclaration de procédure en code>;|
<déclaration de rangée de procédures>;

c) <déclaration d'élément>::=elt <identificateur>=<élément>|
<déclaration d'élément>,<identificateur>=<élément>

d) <déclaration de repère>::=rep <identificateur><initialisation>
<déclaration de repère>, <identificateur><initialisation>

e) <initialisation>::=<vide>|:=<élément>

f) <déclaration de rangée d'éléments>::=rangée (<entier décimal>)
elt <identificateur>=(<liste d'éléments>)|
<déclaration de rangée d'éléments>, |identificateur>)
=(<liste d'éléments>)

g) <liste d'éléments>::=<élément>|<liste d'éléments>, <élément>

h) <déclaration de rangée de repères>::=rangée (<entier décimal>)
rep <identificateur><rangée initiale>|
<déclaration de rangée de repères>, <identificateur>
<rangée initiale>

i) <rangée initiale>::=<vide>|:=(<liste d'éléments>)

Sémantique : Cette partie sert à déclarer les identificateurs dont la portée est le programme. Pour les identificateurs d'éléments ou de rangée d'éléments, elle précise l'élément ou la rangée désigné par l'identificateur. Pour les identificateurs de repère ou de rangée de repères, on peut leur faire repérer une valeur initiale. Le nombre d'éléments ou de repères composant une rangée est indiqué par l'entier qui suit le déclareur rangée.

Exemples : rep ml:=1000, n0:=0, nb:=0, niveau:=1, eti:='e000' ;
rangée (6) elt ri=(1, 2, 3, 4, 5, 6) ;
rangée (50) rep ad, cours ;

4.5 - Déclarations de procédure

4.5.1 - Déclarations de procédures

a) <déclaration de procédure>::=proc <résultat><identificateur>
=<corps de procédure>

b) <résultat>::=<vide>|elt|rep

c) <corps de procédure>::=(<liste de paramètres formels>)<expression>|
(<vide>)<expression>

d) <liste de paramètres formels>::=<liste de paramètres formels>;
<déclareur><liste d'identificateurs>|<déclareur>
<liste d'identificateurs>

e) <déclareur>::=elt |rep|rangée elt|rangée rep

f) <liste d'identificateurs>::=<identificateur>|<liste d'identificateurs>,
<identificateur>

Sémantique : On peut définir deux sortes de procédures :
- Celles qui possèdent un résultat ; leur déclaration contient, avant l'identificateur, le déclareur du type du résultat : élément ou repère.
- Celles qui n'admettent aucun résultat.
Selon le cas, l'expression du corps de procédure est sans résultat ou donne le résultat voulu. Les paramètres formels, s'ils existent, peuvent être des repères, des rangées de repères, des éléments, des rangées d'éléments.
Les procédures peuvent être récursives.

Exemples :

```
proc elt to = (elt type) type = 'réel' : 2, 1 ;
proc decrement = (elt ai, type, résint) résint=0 : ( ), ai='*' : (ml:=ml-1),
                  (ml:=ml - to (type)) ;
proc restaureg = (rep 1 ; elt k)
                  (k>1 : ( ), (ordre ('lw'; 1 ; 'o' ; ml ; niveau) ; ml :=ml-1) ;
                  restaureg (1-1 ; k)) ;
```

4.5.2 - Déclarations de procédures en code

a) <déclaration de procédure en code>::=
CODE <rescode><identificateur> =<corps de procédure>

b) <rescode>::=<résultat>|rangée rep| rangée elt

c) <corps de procédure>::=(<liste des types des paramètres>)|(<vide>)

d) <liste des types des paramètres>::=
<décode><liste d'identificateurs>|
<liste des types des paramètres> ; <décode>

e) <décode>::=<déclareur>| proc

Sémantique :

On peut déclarer des procédures qui sont directement écrites en code.

Leur écriture dépend directement de l'implémentation du langage FACE. En plus des résultats élément repère, elles peuvent avoir pour résultat une rangée de repères et leurs paramètres peuvent être des procédures.

Exemple :

code rangée rep rechtable = (rangée rep inder ; proc filtre)

4.6 - Déclaration de rangée de procédures

a) <déclaration de rangée de procédures>::=rangée (<entier décimal>)
proc <identificateur>;<liste de procédures de la rangée>

b) <liste des procédures de la rangée>::=<identificateur>
[<entier décimal>] =<désignation de la procédure>|
<liste de procédures de la rangée>;<identificateur>
[<entier décimal>] =<désignation de la procédure>

c) <désignation de la procédure>::=(<entête>)<expression>|
<identificateur> [<entier décimal>]

d) <entête>::=<entier décimal><vide>, <entier décimal>|
<entier décimal>, rangée, <entier décimal>

Sémantique :

Une déclaration de rangée de procédures définit les procédures de la rangée, notées par l'identificateur de la rangée indicée par un entier, grâce aux désignations de procédures. Une désignation de procédure peut être explicite (entête et expression sans résultat) ou désigner une procédure appartenant à la même rangée ou à une autre rangée déclarée précédemment. Les indices ne sont pas nécessairement croissants et certains d'entre eux peuvent être absents ; dans ce cas la procédure correspondante n'effectue aucune action.

Les rangées de procédures sont formées de procédures sans résultat.

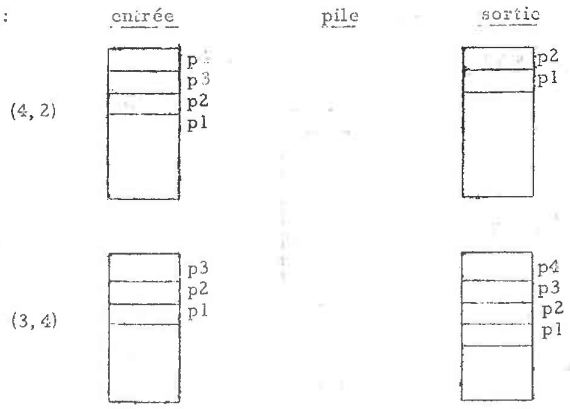
Il n'existe pas d'identificateur indicé de type procédure (4.7.4);

Une rangée de procédures ne peut être un paramètre formel. Une rangée de procédures ne peut donc être utilisée que comme paramètre d'une procédure standard, les seules procédures standard actuellement prévues avec ce type de paramètre sont appl et anapplyse.

Les paramètres formels des procédures d'une rangée sont tous des identificateurs formés du caractère # suivi d'un entier (4.1.4). Si l'entête est (K, L), ils sont tous de type repère ; si l'entête est (K, rangée, L) ils sont de type repère sauf pk (voir plus bas). Dans les deux cas, leur nombre est le plus grand des entiers K et L.

La procédure appl admet deux paramètres : une rangée d'éléments rel et une rangée de procédures rp. Elle utilise une pile. Elle considère successivement les éléments de la rangée rel et appelle la procédure de la rangée rp dont l'indice est égal à l'élément considéré. Si l'entête de procédure est (K, L) : les K paramètres p1, ...pK désignent, de "bas en haut", les K cellules supérieures de la pile ; si L>K, on crée L - K nouvelles cellules au sommet de la pile, pK+1, ..., pL ; si L<K, après exécution de la procédure seules les cellules p1, ..., pL subsistent sur la pile.

Exemples :

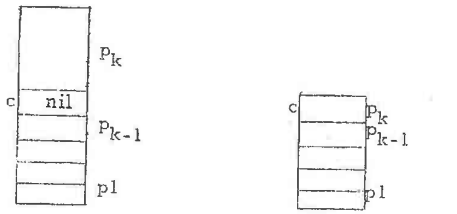


Si l'entête de procédure est (K, rangée, L) p_K est un paramètre de type particulier qui peut désigner successivement une rangée puis un repère. La pile doit alors contenir au moins une cellule repérant nil : (4.1.5) ; désignons par c la plus haute de ces cellules.

Au départ, p_K désigne la rangée formée des cellules situées au-dessus de c . Dès qu'une affectation est faite à p_K , il désigne un repère qui est la cellule c , et la rangée précédemment désignée par p_k devient donc inaccessible.

Si K>1, $p_1, \ldots, p_{k-1}$ désignent, de bas en haut, les k-1 cellules se trouvant en-dessous de c .

Exemple :



a l'entrée de la procédure k=5 — après p_k:=...

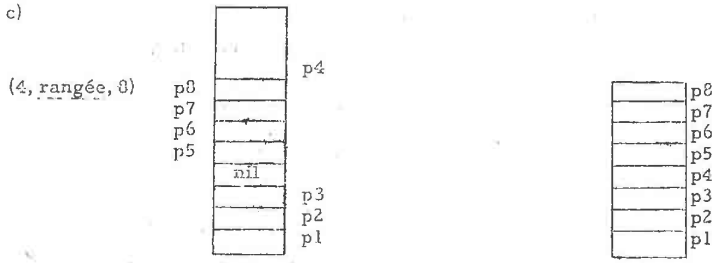
Si $L>K$, $p_{k+1} \cdots p_L$ désignent les cellules situées au-dessus de c et, à la sortie de la procédure, p_L est le sommet de la pile. Certaines de ces cellules appartiennent donc à la rangée désignée par p_k : p_{k+i} est synonyme de $p_k[i]$ si i est au plus égal à la longueur lg de la rangée ; lorsque $K-L>lg$, $K-L-lg$ cellules sont créées au sommet de la pile.

Exemples :

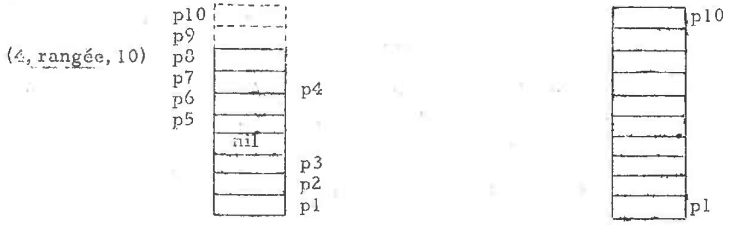
a)

b)

c)



d)



La procédure anapplyse admet trois paramètres de types respectifs : élément, rangée de repères, rangée de procédures ; anapplyse (n ; struct ; rproc) équivaut à analyse (n ; struct) ; appl (struct ; rproc) (4.3).

4.7 - Expression fermée

a) <expression fermée>::=(<liste de déclarations locales>
<fin d'expression fermée>)

b) <fin d'expression fermée>::=<vide>|<suite d'instructions>

Sémantique : Une expression fermée sert à définir un calcul et à fournir un résultat éventuel qui peut être un élément ou un repère. Les déclarations locales (4.7.1) définissent des repères ou rangées de repères dont la portée est l'expression fermée. Le calcul est défini par la suite des instructions (4.7.2) .
Une expression fermée peut être vide. Elle a alors la forme ( ) .

Exemples : (r = 1 : 0, 'erreur')
(anapplyse (1 ; chaîne ; lex) ; appl (chaîne ; gen))
(rangée (4) rep r = recherche (table ; pl ; b) ; rep co ;
r[1] = 'r' : (co:='1d'), (co:=' 1w') ;
ordre (co ; '14' ; 0 ; r[2] ; r[3]))

4.7.1 - Liste de déclarations locales

a) <liste de déclarations locales>::=<vide>|<liste de déclarations locales>
<déclaration locale>

b) <déclaration locale>::=<déclaration locale de repère>;|
<déclaration locale de rangée de repères>;

c) <déclaration locale de repère>::=rep<identificateur><valuation>|
<déclaration locale de repère>, <identificateur>
<valuation>

d) <valuation>::=<vide>|=<secondaire>|:=<expression simple>

e) <déclaration locale de rangée de repères>::=rangée(<entier décimal>)
rep<identificateur> <valrang>
|<déclaration locale de rangée de repères>, <identificateur>
<valrang>

f) <valrang>::=<vide>|=<primaire>|:=(<liste d'expressions imples>

g) <liste d'expressions simples>::=<expression simple>|
<liste d'expressions simples>, <expression simple>

Sémantique : Les déclarations locales déclarent des identificateurs de repère ou de rangée de repères qui ont pour portée l'expression fermée commençant par ces déclarations. Si l'identificateur x est déclaré dans une expression fermée contenue dans le domaine de validité d'un même identificateur x , ce dernier perd sa validité pour la durée de l'expression fermée contenant la nouvelle déclaration de x .
Pour une rangée, l'entier qui suit rangée précise sa taille.
Dans le cas où la valuation (resp. le valrang) commence par = , le secondaire (resp. primaire) doit avoir pour valeur un repère (resp. une rangée de repères dont la taille est celle de la rangée déclarée) : l'identificateur déclaré désigne alors cette valeur pendant toute l'expression fermée. Dans le cas où la valuation est vide ou commence par := , la déclaration crée un nouveau repère (resp. une nouvelle rangée

de repères) qui n'existe que jusqu'à la sortie de l'expression fermée et y est désignée par l'identificateur ; si la valuation (resp. le valrang) commence par := , l'expression simple doit (resp. les expressions simples doivent) avoir une valeur de type élément et la déclaration fait repérer cette valeur (resp. la rangée des valeurs de ces expressions qui doivent être en nombre convenable) par le repère créé.
Comme le nom créé n'existe que dans l'expression fermée, il ne doit pas être utilisé à l'extérieur, c'est par exemple une erreur de déclarer une procédure par

proc rep pr = (rep x ; ... ; x)

Exemples :

rep étiq:=creeti+1 ; rangée (3) rep r = recherche(table ; id ; b) ;
rep t, cod ;

4.7.2 - Suite d'instructions

a) <suite d'instructions>::=<vide>|<suite d'instructions> ; <instruction>

b) <instruction>::=<instruction d'affectation> |<expression>

c) <instruction d'affectation>::=<secondaire>:=<expression>

Sémantique :

Lorsqu'une instruction est une affectation, elle est sans valeur. Lorsqu'elle est une expression, elle a pour valeur le résultat de l'expression s'il existe ou est sans valeur sinon.

Dans une affectation, l'expression doit avoir un résultat qui soit un élément . Le secondaire doit avoir pour valeur un repère (4.7.5).

Exemple : ad:=ad+2 ;
to:=type = 'réel' : 2, 1 ;
info [3]:=adr+4x(c-1) ;
p2:=compacter ('b' ; 's' ; 'x' ; 0) ;

4.7.3 - Expression

a) <expression>::=<expression simple>|<expression simple>:
<expression simple>, <expression>

Sémantique : Soit une telle expression de la forme $\mathcal{E}1 : \mathcal{E}2, \mathcal{E}3$; alors $\mathcal{E}1$ doit avoir une valeur de type élément ; $\mathcal{E}2$ et $\mathcal{E}3$ doivent toutes les deux être sans valeur ou toutes les deux avoir une valeur de même type élément ou repère. La valeur de cette expression est calculée de la façon suivante :
Si $\mathcal{E}1$ a une valeur différente de zéro, alors elle est égale à la valeur de $\mathcal{E}2$, sinon elle est égale à la valeur de $\mathcal{E}3$.

Exemples :

pl = 'r' : (col:='ld'), (col:='lw')
resint = 0 : ( ) , ai = 1 : (ml:=ml-1), (ml:=ml-tà(type))

4.7.4 - Expression simple

a) <expression simple>::=<terme booléen>|<expression simple>
∨<terme booléen>

b) <terme booléen>::=<facteur booléen>|<terme booléen>∧
<facteur booléen>

c) <facteur booléen>::=<primaire booléen>|¬<primaire booléen>

d) <primaire booléen>::=<expression arithmétique><opr>
<expression arithmétique>|<expression arithmétique>

e) <opr>::= =|<|≤|>|≥|≠

f) <expression arithmétique>::=<terme>|<expression arithmétique>
<opa><terme>|<opa><terme>

g) <opa>::= +|-

h) <terme>::=<facteur>|<terme><opm><facteur>

i) <opm>::= x|/

j) <facteur>::=<secondaire>|<élément>|<expression fermée>

Sémantique :

Les priorités des opérateurs sont par ordre décroissant : x /
+ -
=, ≠, <, ≤, >, ≥
¬
∧
∨

De plus, en cas d'égalité de priorité, tous les opérateurs opèrent de gauche à droite.
Une expression qui contient un opérateur a pour valeur un élément. Les opérateurs arithmétiques +, -, x, / ont leur signification habituelle sur les entiers. Pour les opérateurs booléens ¬, ∨, ∧, les opérandes sont d'abord convertis en valeur logique. 0 donne faux et tout autre entier VRAI ; le résultat de l'opération booléenne est ensuite converti en un entier : 1 pour vrai et 0 pour faux. Le résultat d'une expression fermée a été étudié en (4.7).
Lorsqu'une expression a pour valeur un repère (c'est alors nécessairement une expression fermée ou un secondaire) et qu'elle se trouve dans un contexte demandant un élément (second membre d'affectation, opérande d'opérateur, paramètre de type élément...) le repère est remplacé par la valeur qu'il repère.

Exemple : (rep x = elt rep(y) ; x<2 : x, ML+1)+z

4.7.5 - Secondaire

a) <secondaire>::=<primaire>|<identificateur>[<indice>]

b) <primaire>::=<identificateur>|<appel de procédure>

c) <indice>::=<expression>

d) <appel de procédure>::=<identificateur>(<liste de paramètres effectifs>)|<identificateur>(<vide>)

e) <liste de paramètres effectifs>::=<paramètre effectif>|
<liste de paramètres effectifs>;<paramètre effectif>

f) <paramètre effectif>::=<expression>

Sémantique :

Un identificateur indicé a pour valeur un élément ou un repère selon le type de l'identificateur qui doit désigner une rangée d'éléments ou de repères ; l'indice détermine l'ordre du résultat (élément ou rangée) dans ce rang.
Le type d'un identificateur est précisé par sa déclaration (4.4 et 4.7.1). Dans un appel de procédure, l'identificateur est celui d'une procédure déclarée (4.5.1), en code (4.5.2) ou standard. Les paramètres effectifs doivent correspondre en nombre et en type aux paramètres formels

puis dans le même ordre. Ils peuvent être de type élément, repère, rangée d'éléments, rangée de repères dans tous les cas, procédure pour les procédures en code ou standard, rangée de procédure dans le seul cas des procédures standard. Un paramètre qui n'est ni de type élément, ni de type repère est nécessairement réduit à un identificateur.
La valeur d'un appel de procédure est le résultat de la procédure appelée s'il existe ; c'est alors un élément ou un repère ou, pour des procédures standard ou en code, une rangée d'éléments ou de repères.

Exemples : anapplyse (1, chaîne ; lex)
ordre ('leow'; reg ; 0 ; ml+1 ; r_1)

5. EXEMPLE

Ce paragraphe donne un aperçu du programme FACE décrivant un compilateur d'Algol 60 en Symbol 10070. On traite en partie la structure de blocs et les déclarations de types ainsi que l'instruction d'affectation. La compilation est faite en deux passages par appels successifs des procédures anapplyse puis appl, le premier passage réalisant l'analyse syntaxique et l'étude lexicographique par l'intermédiaire d'anapplyse et de la rangée de procédures lex, le second terminant la génération par l'intermédiaire de appl et de la rangée de procédures gen.

Grammaire	Commentaire
bloc non éti : début, partie dec, fin de bloc/5 ; début :"'début"'/6 ; partie dec : dec//partie dec, ', ', dec ; dec : dec type/7/decaig//dectab//decproc/; dectype : dectype, ', ', idec/8/type, idec/8 ; idec : id/ ;	partie de la grammaire Algol 60 engendrant blocs et déclarations de type, décrits en langage FACE
ins de base sans éti : insaf/40/insallera// insproc//insvide/ ; insaf : partie gauche, ':=', expr/51/ partie gauche, ':=', insaf/52 ;	instruction d'affectation.
rangée(2000) rep chaîne ;	chaîne est la rangée de repères résultat de la procédure analyse ou anapplyse.
rangée(100) rep ad, cours ;	ad est une rangée de repères permettant pour chaque bloc, de repérer l'adresse de la première mémoire libre après les réservations en zone statique. cours est une rangée de repères permettant de connaître les blocs en cours d'activation.
rep no := 0, nb:=0, ml:=0, niveau:=1 ;	no repère le numéro du bloc en cours d'activation. nb indique le maximum des numéros du bloc déjà rencontrés. niveau est le niveau de la procédure en cours.
rangée(3) rep acclib := (1, 1, 1)	acclib contient des booléens indicateurs d'occupation des registres
rangée(3) elt acc = (8, 10, 12)	acc numéro des registres.

```
rangée(120) proc lex ;
lex[5] = (2, 0)
(ad[no] := ml+1;ml:=#1;no:=#2);
lex[6] = (0, 2)
(#1:=ml; #2:=no; nb:=nb+1; no:=nb);
lex[7] = (1, 0) ();
lex[8] = (2, 1)
```

étude lexicographique

fin d'un bloc

début d'un bloc

fin d'une déclaration de type
déclaration d'une variable simple
transport en table d'identificateurs et incrémentation du compteur ml.

```
rangée(4) rep inf :=(no, niveau, #1, ml);
entrée(table; #2; inf; f); ml:=ml+to(#1);
```

(Pour les autres règles citées rien n'est à faire à l'étude lexicographique).

```
rangée(120) proc gen ;
gen[5] = (1, 0)
(cours[no] := 0 ; no:=#1);
gen[6] = (0, 1)
(#1:=no; nb:=nb+1; no:=nb; ml:=ad[no];
cours[no]:=1; ordre('lw';'4'; 0; ad[#1]
-1; niveau);
ordre('stw'; '14'; 0; ad[no]-1; niveau));
gen[40] = (2, 0) libérer(#1);
gen[51] = (10, 2).....
```

génération :
fin d'un bloc.

fin d'une affectation multiple ou simple

début d'une affectation multiple ou simple
gen[51] doit établir les conversions éventuelles de type entre l'expression à affecter et la première partie gauche, puis effectuer l'affectation ce qui permet d'avoir, dans tous les cas, la valeur de l'expression dans un registre. Le numéro de ce registre et le type de l'affectation sont seuls conservés

```
gen[52] = (7, 2)
(rep co; #2; #7; erreur( ),(
co:=#2='r':'std', stw';
libérer(p3); décrement(#3; #2; #4);
#3=0; ordre('co'; #6; 0; #1; #5),
(ordre('lw'; '14'; 0; #1; #5);
ordre('co'; #6; '*'; 14; 0))));
```

affectations successives


```
proc elt to = (elt type)
type = 'r' : 2, 1;
proc libérer = (elt adr)
testacc(adr)=1 : (acclib[adr]:=1), ();
proc elt testacc = (elt adr)
(adr = 8) ∨ (adr=10) ∨ (adr=12);
1, 0;
proc décrement = (elt ai,type, resint)
resint=0 :( ), ai=1 : (ml:=ml-1),
(ml:=ml-to(type));
proc elt f=(elt x) no > x:0,
no = x : erreur( ), 1.
(anapplyse(1; chaîne; lex); appl(chaîne; gen))
```

Note :

ordre est une procédure standard plaçant dans un tampon de sortie une partie d'instruction générée. Ses cinq arguments sont, dans l'ordre, pris comme code opération, numéro de registre, marque d'adressage indirect, adresse et numéro de registre d'index.

CHAPITRE II

RESTRICTION DU LANGAGE $FACE_0$ PAR RAPPORT A FACE

Le langage implémenté $Face_0$ est un sous-ensemble du langage Face. Ce noyau retient en partie les notions définies dans Face et sert pour la compilation de ce langage [1].

Pour faciliter la réalisation du compilateur $Face_0$ écrit en langage machine, des restrictions ont été apportées au langage. Elles concernent essentiellement les parties suivantes :

1) les types d'information
Les identificateurs éléments et les rangées d'éléments ne sont pas admis en déclaration. La partie initialisation des informations est supprimée.

2) les grammaires
$Face_0$ ne possède pas de déclaration de grammaires. Celles-ci sont remplacées par des tables codifiées (cf. 4. 3).

3) les procédures
Seules les déclarations de procédures à résultat élément et les procédures standard sont retenues en $Face_0$.

4) les paramètres des rangées de procédures
Les rangées de repères ne sont pas admises comme paramètres des procédures d'une rangée.

5) les déclarations locales
Les déclarations locales sont profondément modifiées. Elles permettent uniquement la déclaration des identificateurs qui désignent des informations déjà existantes (cf. 2. 7. 1).

6) les expressions
Seules les opérateurs additifs (+, -) et de relation (=, ≠, . .) sont admis en $Face_0$.

1. TRAITEMENT DES GRAMMAIRES DANS LES COMPILATEURS FACE et $FACE_0$

Dans Face, l'implémentation d'un langage T s'effectue de la façon suivante :

1) pendant la compilation du langage source Face, les grammaires déclarées sont transformées en tables qui comportent :

- la liste des terminaux
- la liste des non-terminaux
- les pointeurs nécessaires pour transformer les grammaires en un graphe.

Un préanalyseur parcourt ces grammaires pour repérer quel type d'analyse (ascendante ou descendante) il vaut mieux leur appliquer. En cas d'analyse descendante, il modifie légèrement les tables pour supprimer les récursivités à gauche ; il imprime un message si l'une des grammaires n'est pas analysable.

2) pendant l'exécution du programme Face, l'analyseur (procédure standard analyse) active l'analyse la mieux adaptée à la grammaire traitée. Celle-ci lit le programme T et le transforme en une rangée d'éléments de numéros de règles en notation postfixée. Cette rangée d'éléments est reprise par le programme de commande appl pendant la phase de génération. L'analyse et la génération peuvent être regroupées dans l'appel de la procédure standard anapplyse.

Exemple d'un compilateur du langage T :

```
grammaire de T ;
déclarations ;
(anapplyse (numéro de règle ; paramaire ; rangée de procédures) ;...)
```

En Face$_0$ le préanalyseur n'est pas inclus dans le compilateur mais est appelé comme une procédure standard ayant pour paramètre le nom de la table qui contient la grammaire codifiée. La procédure anapplyse n'est pas définie dans ce langage.

2. DEFINITION DU LANGAGE FACE$_0$ PAR RAPPORT A FACE

2.1 - (cf. I.4.1)* Eléments de base du langage

2.1.1 - (cf. I.4.1.1.) Symboles de base

La syntaxe reste inchangée excepté la règle

d) <chiffre hexadécimal>

qui est supprimée

* Les numéros de paragraphes renvoient à ceux utilisés pour la définition de Face.

2.1.2 - (cf. I.4.1.2.) Chaînes

Sémantique :

Les chaînes servent à définir des quantités de type élément. Elles sont mises entre apostrophes. Dans une chaîne l'apostrophe est représenté par deux apostrophes successifs et les blancs y gardent leur signification. Si la longueur est inférieure à la longueur maximale, les caractères sont cadrés à droite et précédés de zéros binaires. Le changement de ligne a lieu quand 72 caractères (y compris les blancs non significatifs) ont été lus sur une ligne.

Exemple :

```
9 ␣A␣B 9
9 B99C 9
```

2.1.3 - (cf. I.4.1.3.) Entiers

Ne sont admis que les entiers décimaux d'où suppression des règles :

b) <entier octal>

d) <entier hexadécimal>

et changement de la règle a) en

<entier>::=<entier décimal>

2.1.4 - (cf. I.4.1.4.) Identificateurs

Les 3 règles de syntaxe de ce paragraphe sont remplacées par la suivante :

a) <identificateur>::=<lettre>|<identificateur><lettre>|
<identificateur><chiffre>

Sémantique :

Un identificateur formé au maximum de 8 caractères peut désigner un repère, une rangée de repères, une procédure ou une rangée de procédures. Les paramètres formels des procédures (cf. 2.5) et les identificateurs de déclarations locales (cf. 2.7.1.) sont formés de la lettre p et d'une suite de lettres ou de chiffres ayant au moins une

lettre. Les paramètres formels des procédures formant une rangée de procédures doivent être désignés par les identificateurs particuliers $p_1, p_2, \ldots$ formés de la lettre p suivie d'un entier.

2.1.5 - (cf. I.4.1.5.) Eléments

En $Face_0$ nil est supprimé car les paramètres formels de type rangée de repère ne sont pas définis dans les rangées de procédures :

a) <élément>::=<entier>|<chaîne fermée>

Sémantique :

L'entier maximum admis est l'entier algébrique représentable sur 32 positions binaires. Sa valeur est de $2,15\ 10^9$. La longueur maximum d'une chaîne fermée est de 4 caractères. Un caractère occupe 8 chiffres binaires et sa valeur est celle retenue par le code EBCDIC.

2.2 - (cf. I.4.2) Programme

Les définitions des grammaires ne sont pas comprises dans le langage. On peut donc écrire

a) <programme>::=<liste des déclarations>
<expression fermée>

2.3 - (cf. I.4.3.) Définition des grammaires

En $Face_0$ les grammaires ne sont pas introduites au niveau du langage mais sous forme de données. Toutes les règles syntaxiques (I.4.3 a-k) de ce paragraphe sont donc supprimées.

Sémantique :

Les grammaires sont données directement sous forme de tables qui sont du même type que celles créées par le compilateur Face (cf. II.1.). A l'exécution elles sont traitées comme données du pré-analyseur (procédure standard préan). Celui-ci, puis l'analyseur sont activés pour former la rangée de repères en notation postfixée.

Exemple d'une compilation de Face écrite en $Face_0$:

déclarations ;

(préan (table des grammaires) ; analyse (numéro de règle ; rangées de repères); appl (rangées de repères ; rangées de procédures) ; ...)

2.4 - (cf. I.4.4.) Liste de déclarations

Les restrictions de ce paragraphe portent sur les initialisations et la déclaration des éléments, rangée d'éléments et procédures en code.
La règle b) est modifiée en :

b) <déclaration>::=<déclaration de repère>|<déclaration de rangée de repères>;|<déclaration de procédure>;|<déclaration de rangée de procédures>;

Les règles suivantes sont sans signification en $Face_0$:

c) <déclaration d'élément>::=<vide>

e) <initialisation>::=<vide>

f) <déclaration de rangée d'éléments>::=<vide>

g) <liste d'éléments>::=<vide>

i) <rangée initiale>::=<vide>

Sémantique :

Les initialisations sont remplacées par des affectations successives au début de l'expression finale ou par une procédure standard qui lit les valeurs en données et les affecte aux repères ou rangées de repères. Par ailleurs les identificateurs éléments sont à remplacer par des repères.

2.5. (cf. I.4.5.) Déclaration de procédure

2.5.1 - (cf. I.4.5.1.)

Les résultats repères et les paramètres formels éléments ou rangée d'éléments ne sont pas définis en $Face_0$:

b) <résultat>::=<vide>|elt

c) <déclareur>::=rep|rangée rep

Sémantique :

Les paramètres formels repères ou rangées de repères sont formés de la lettre p et d'une suite de lettres et de chiffres dont au moins une lettre.

Un identificateur de procédure doit être déclaré avant son premier appel.

2.5.2 - (cf. I.4.5.2.) Déclaration de procédure en code

Le système $Face_0$ inclut les procédures standard nécessaires pour créer un compilateur Face et n'admet pas la déclaration de procédures supplémentaires écrites en code. Ainsi toutes les règles syntaxiques (I.4.5.2. a) c)) sont à supprimer.

Sémantique :

Les procédures standard sont déclarées implicitement lors de leur premier appel. Suivant leur définition en code, elles ont un résultat ou non qui peut être :

1) un élément
2) un repère
3) une rangée de repères

Les procédures à résultat repère ou rangée de repères ne servent que dans les déclarations locales.

En plus des paramètres formels d'une procédure déclarée dans le texte une procédure standard admet les paramètres suivants :

1) élément
2) procédure
3) rangée de procédure

Les principales procédures standard (cf. V) du système $Face_0$ sont :

1) analyse
2) préan
3) entrée
4) recherche
5) appl

2.6 - (cf. I.4.6) Déclaration de rangées de procédures

$Face_0$ ne permet pas de répéter une même action pour des numéros de règles différentes sans avoir à recopier cette action et n'admet pas les rangées de repères comme paramètres d'une procédure. D'où modification des règles syntaxiques :

c) <désignation de la procédure>::=(<entête>)<expression>

d) <entête>::=<entier>,<entier>

Sémantique :

Les procédures de la liste sont obligatoirement dans l'ordre de leurs indices et certains indices peuvent être absents ; dans ce cas la procédure correspondante n'effectue aucune action.

2.7 - (cf. I.4.7.) Expression fermée

La syntaxe reste inchangée

Sémantique :

Le résultat d'une expression est un élément. Le résultat repère n'est pas admis.

2.7.1 - (cf. I.4.7.1.) Liste de déclaration locales

Dans les règles syntaxiques du paragraphe seules les suivantes sont modifiées :

d) <valuation>::=<secondaire>

f) <valrang>::=<primaire>

Sémantique :

L'identificateur d'une déclaration locale est composé de la lettre P et d'une suite de lettres et de chiffres dont au moins une lettre.

Par ailleurs, un identificateur déclaré paramètre formel dans une procédure ne peut être pris comme identificateur de déclaration locale dans cette même procédure. Dans une expression et toutes celles qui lui sont imbriquées un identificateur déclaré local ne peut être utilisé qu'une seule fois.

Les déclarations locales servent en $Face_0$ à repérer des éléments, des repères ou des rangées de repères déjà existantes et permettent ainsi un repérage efficace.

Exemple :

Pendant la compilation d'un programme, on crée des tables d'identificateurs dans lesquelles on associe à chaque identificateur du programme une rangée de repères d'informations. En $Face_0$, la procédure standard recherche permet de retrouver les informations souhaitées à partir de l'identificateur qui est le critère. Une rangée de repères déclarée locale permet de désigner la rangée recherchée.

(ran rep Pa = recherch(....) ;

2.7.2 - (cf. I.4.7.2.) Suite d'instructions

La syntaxe et la sémantique restent inchangées.

2.7.3 - (cf. I.4.7.3.) Expression

La syntaxe reste inchangée.

Sémantique :

Les expressions n'admettent comme valeur que des éléments.

2.7.4 - (cf. I.4.7.4.) Expression simple

Seuls les opérateurs de relation (=, ≠, --) et additifs (+, -) sont admis en $Face_0$. Les règles de syntaxe (I.4.7.4. a-j) sont remplacées par les suivantes :

a) <expression simple>::=<expression arithmétique><opr><expression arithmétique>|<expression arithmétique>

b) <opr>::= =|≠|>|≥|<|≤

c) <expression arithmétique>::=<facteur>|<expression arithmétique><opa><facteur>

d) <facteur>::=<secondaire>|<élément>|<expression fermée>

e) <opa>::= +|-

Sémantique :

Dans un programme les opérateurs booléens doivent être remplacés par des comparaisons successives ou imbriquées. Les priorités des opérateurs sont par odre décroissant :

\+ -

=, ≠, <, ≤, >, ≥

2.7.5 - (cf. I.4.7.5.) Secondaire

Deux modifications sont amenées dans ce paragraphe. Dans <indice> et <paramètre effectif> le terme <expression> est remplacé par des notions plus restrictives.

c) <indice>::=<identificateur>|<entier>

f) <paramètre effectif>::=<identificateur>|<élément>

On admet le paramètre effectif <élément> seulement dans les procédures standard.

Sémantique :

Le type d'un identificateur est précisé par sa déclaration. Un identificateur indicé a pour valeur un repère et le type de l'identificateur doit désigner une rangée de repères ; l'indice détermine

l'ordre du résultat dans la rangée.

Dans un appel de procédure, l'identificateur est celui d'une procédure déclarée ou standard. Les paramètres effectifs peuvent être de type repère, rangée de repères dans tous les cas, élément, procédure et rangée de procédures dans le cas des procédures standard.

La valeur d'un appel de procédure est le résultat de la procédure appelée s'il existe ; c'est un élément dans le cas des procédures déclarées ou un repère et une rangée de repères dans le cas des procédures standard.

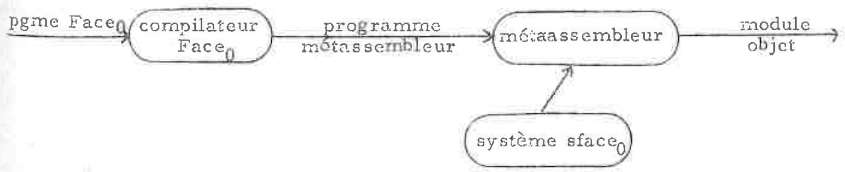
CHAPITRE III

IMPLEMENTATION DE FACE$_0$

1. PLAN GENERAL DU COMPILATEUR

Le compilateur du langage Face$_0$ est écrit en langage assembleur du CII 10070. Les instructions générées sont des macro-instructions du langage métassembleur. Elles sont le résultat de l'analyse faite pour un programme Face$_0$.

Un langage (Face comme Face$_0$) où on remplace les expressions par une lettre est un langage régulier. Pour l'analyser le programme représente un automate fini, déterministe si on lit une unité syntaxique en avance. Pour chaque règle de syntaxe appliquée on génère un ou plusieurs appels de macro-instructions qui sont ensuite repris par le métassembleur pour générer le module objet. Celui-ci a donc une fonction semblable à celle de la procédure standard appl en Face . Les rangées de procédures génératrices du langage objet en Face sont remplacées dans le compilateur Face$_0$ par un système de procédures métassembleur appelé le système sface$_0$.



Les différents modules du compilateur sont composés en fonction de la grammaire de Face$_0$. Chacune des catégories syntaxiques suivantes est traitée par un module à part :

a) <déclaration de repères> (cf. 4.1.)

b) <déclaration de rangée de repères> (cf. 4.1.)

c) <déclaration de procédures> (cf. 4.2.)

d) <déclaration de rangées de procédures> (cf. 4.3.)

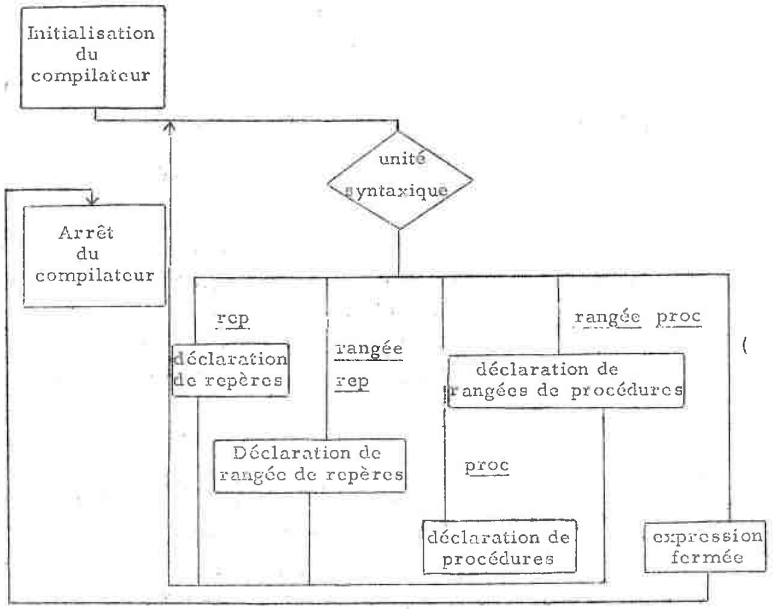
e) <expression fermée> (cf. 4.4.)

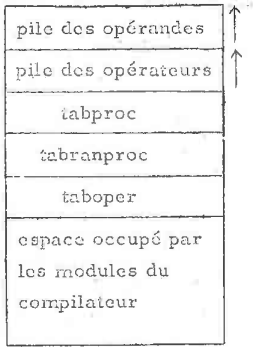
Pour l'analyse des expressions, le compilateur appelle un sous-programme d'analyse syntaxique qui utilise une pile des opérateurs et des opérandes. Une table donnant la priorité des opérateurs entre eux permet le branchement aux traitements à effectuer. La pile des opérateurs (oper) et la pile des opérandes (opra) sont limitées et permettent une imbrication maximale de 24 parenthèses dans une expression.

Le compilateur utilise trois tables, chacune ayant une taille maximale :

- la table tabproc contient le nom des procédures déclarées et le type des paramètres formels,
- la table tabranproc contient les étiquettes de début des procédures d'une rangée,
- la table taboper contient les adresses du traitement des expressions en fonction des priorités du sommet de la pile des opérateurs et de l'opérateur venant d'être lu.

Le schéma général et la gestion mémoire du compilateur sont les suivants :





gestion mémoire

2. REPRESENTATION DES DIVERS TYPES D'INFORMATION

Les informations utilisées en $Face_0$ sont les éléments, les repères et les rangées de repères. Leur portée est le programme. Ils sont dits globaux. Leur modèle de représentation est le suivant :

<élément> est une constante numérique ou une chaîne fermée (chaîne de caractères). Pour ce type d'information le compilateur génère des libellés. Ceux-ci sont transformés par le métassembleur en références à des valeurs de données

Exemple : LW, op1 ='123' ⇒ LW, lop adresse

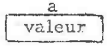
:

adresse data '123'

<repère> Un mot mémoire est réservé par repère. Pour ne pas alourdir l'exécution du programme $Face_0$, chaque repère est dérepéré dès son apparition. La notion de repère ne sera gardée que pour les identificateurs commençant par la lettre P (paramètres formels d'une procédure et déclarations locales)

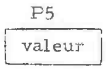
Exemples :

1) rep a



2) Soit Pa un paramètre formel d'une procédure, son paramètre effectif correspondant étant le repère a :

3) Soit P5 le paramètre d'une procédure d'une rangée. A l'adresse F5 on range la valeur de ce paramètre



<rangée de repères> La notion de 'repère' est maintenue. Un pointeur repère la composante zéro de la rangée. Par ce pointage, le transfert des informations d'une rangée dans une autre (cf. 2. 7. 1) se fait sans copie des composantes. Pour pouvoir contrôler pendant

l'exécution le débordement des indices, la composante zéro contient la taille de la rangée.

Exemples :

1) rangée(12) rep cd ;

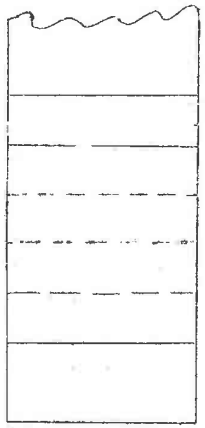
2) Soit Px le paramètre formel rangée de repères d'une procédure et le paramètre effectif correspondant étant la rangée cd :

3. GESTION MEMOIRE POUR LE CODE GENERE

Les programmes $Face_0$ utilisent une pile où sont rangés les résultats intermédiaires et les informations nécessaires aux appels récursifs des procédures. Cette pile appelée ml est du type statique. Chaque information rangée sur cette pile a une adresse relative à un registre d'index appelé le pointeur de procédure (abréviation : pop) .

La mémoire occupée par un programme est divisée en 4 régions :

- les procédures standard
- les déclarations : elles peuvent se suivre dans un ordre quelconque excepté une procédure qui doit être déclarée avant son appel
- l'expression finale
- la pile ml qui s'étend vers le haut de la mémoire.



4. ETUDE DES INSTRUCTIONS GENEREES PAR LE COMPILATEUR

Le compilateur $Face_0$ génère des appels de macro-instructions de format :

etiquette commande argument, ..., argument

L'étiquette et les arguments sont optionnels. On admet jusqu'à 5 arguments.

Les appels de macro-instructions sont repris par le méta assembleur qui avec l'aide du système $sface_0$ génère le langage machine dont le format est le suivant :

a, b c d, e

a : commande
b : registre
c : * marque d'adressage indirect
= marque de libellé . Les libellés sont transformés en références à des valeurs de données
d : opérande mot-mémoire
e : registre d'indexation

Les commandes utilisées sont :

RES	réservation d'une zone mémoire
DATA	génération d'une valeur de donnée
EQU	attribution d'une valeur à un symbole
SET	attribution d'une valeur à un symbole (peut-être redéfini)
LW	chargement d'un mot
STW	rangement d'un mot
LI	chargement immédiat
AI	addition immédiate
AW	addition d'un mot
SW	soustraction d'un mot
CW	comparaison d'un mot
XW	échange d'un mot
B	branchement inconditionnel
BG	branchement si supérieur à
BL	branchement si inférieur à
BE	branchement si égal à
BLEZ	branchement si inférieur ou égal à zéro
⋮	

Le caractère $ désigne le compteur d'emplacement mémoire. On utilise quatre registres pendant l'exécution d'un programme $Face_0$

lop	1 registre d'opérande
i_1	2 registres d'indexation et de mémoires de travail
i_2	
pop	registre de pointeur de procédures

La description du langage généré est faite en fonction des règles de syntaxe traitées dans un programme $Face_0$.

4.1 - Déclaration de repère et de rangée de repères

Dans ce paragraphe sont décrites les instructions générées par le compilateur lors de l'analyse des deux règles suivantes :

<déclaration de repères> (cf. II.2.4.-d)
<déclaration de rangées de repères> (cf. II.2.4.-h)

Pour un repère, seule la réservation d'un mot mémoire est générée. Dans le cas d'une rangée de repères on génère la macro-instruction RANR dont l'argument est la taille de la rangée.

Exemple : rep a, b ; rangée(10) rep file ;

instructions générées

```
a     RES    1
b     RES    1
file  RANR   10
```

Ces lignes sont traduites par le métaassembleur en :

```
a     RES    1
b     RES    1
file  DATA   $+1
      DATA   12
      RES    12
```

4.2 - Déclaration de procédure

<déclaration de procédure> (cf. II.2.5.1.-a)

Les procédures sont des expressions qui peuvent s'exécuter récursivement.

<résultat> (cf. II.2.5.1.-b)

Le langage objet généré pour une procédure élément et une procédure sans résultat sont les mêmes sauf que dans le premier cas on doit sauvegarder le registre lop avant l'exécution de l'expression et tenir compte du résultat élément.

<identificateur>

Cet identificateur définit le début du corps de procédure. On génère :

```
identificateur   EQU   $
```

<liste de paramètres formels> (cf. II.2.5.1.-d)

On ne fait pas de distinction entre les paramètres repères et rangées de repères lors de leur déclaration. Leurs emplacements sont connus par rapport au sommet de la pile ml. Ils sont définis par l'appel de la macro-instruction EQPA dont les arguments sont :

a) une étiquette (créée par le compilateur) qui désigne le nombre total de paramètres (af(1))

b) le rang du paramètre dans la liste (af(2))

Cette macro-instruction est traduite en :

```
étiquette   SET   -af(1)+af(2)-2
```

étiquette = identificateur du paramètre formel

En fin de déclaration, le nombre des paramètres est défini par l'instruction :

```
@i   EQU   n
```

n = nombre de paramètres

<expression>

Avant l'exécution de <expression> dans le cas des procédures élément, on sauvegarde le registre lop à l'emplacement du résultat par l'appel de la macro-instruction SAUV dont l'argument (af(1)) est le nombre de paramètres.

```
SAUV   n
```

n = nombre de paramètres

est traduit par le métassembleur en

```
STW, lop   -af(1)-2, pop
```

Les instructions générées pour l'expression sont identiques à celles de l'expression finale.

Dans le cas d'une procédure élément, le résultat est mis à l'emplacement qui lui est réservé et le registre lop est restauré par l'appel de RESL :

	RESL	n
qui est traduit en	XW, lop	-af(1) -2, pop

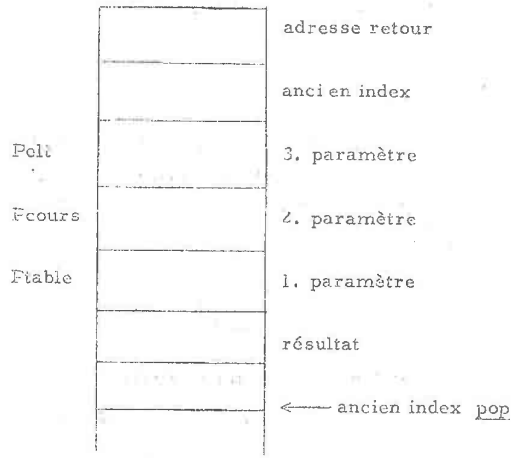
n = nombre de paramètres

Le retour au programme appelant s'effectue par la macro-instruction RET (sans arguments) qui génère :

LW, i1	*pop
B	*i_1

Exemple de compilation d'une procédure élément à trois paramètres :

proc elt f = (rangée rep ptable, pcours ; rep pelt) .... ;



f	EQU	$	f	EQU	$
Ptable	EQPA	@, 1	Ptable	SET	-4
Pcours	EQPA	@, 2	Pcours	SET	-3
Pelt	EQPA	@, 3	Pelt	SET	-2
@	EQU	3	@	EQU	3
	SAUV	3		STW, lop	-5, pop
	. . .	expression		. . .	
	RESL	3		XW, lop	-5, pop
	RET			LW, i_1	*pop
				B	*i_1

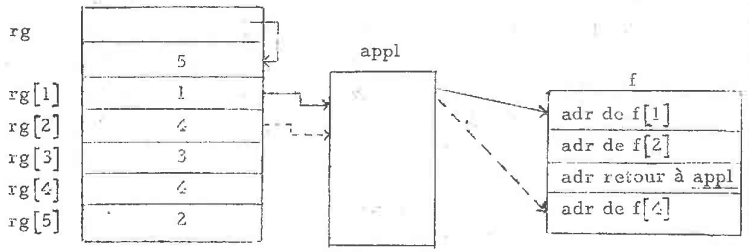
4.3 - Déclaration de rangée de procédures

<déclaration de rangée de procédures> (cf. II.2.6. -a)

Une rangée de procédures est une suite de procédures sans résultat dont les paramètres sont les emplacements mémoire au sommet de la pile ml. L'appel s'effectue par la procédure standard appl. Les composantes d'une rangée de repères (paramètres de appl) sont balayées par cette procédure. En fonction de la valeur v des composantes, la $v^{ième}$ procédure de la rangée est exécutée. Si la $v^{ième}$ procédure est vide, il y a retour immédiat à appl. L'exécution de la rangée des procédures se termine quand toutes les composantes de la rangée de repères ont été explorés.

Exemple :

```
rangée (5) rep rg ;
rangée (4) proc gen ; gen[1] = (0, 5)... ;
                      gen[2] = (7, 0)... ;
                      gen[4] = (2, 3)... ;
```



<désignation de la procédure> (cf. II.2.6.-c)

Par procédure d'une rangée on génère une étiquette @i créée par le compilateur marquant le début de la désignation :

```
@i      EQU     $
```

<entête> (cf. II.2.6.-d)

Les paramètres d'une procédure sont référencés dans l'expression par p1, p2, ... pn avec pn au sommet de la pile ml. Les macro-instructions PARE et PARS permettent la mise à jour de cette pile en fonction des paramètres en entrée et en sortie. Leur argument (af(1)) est le nombre de paramètres.

mouvement de la pile ml en entrée

```
        PARE    n
```

n = nombre de paramètres en entrée

Cet appel de macro-instructions est traduit en :

```
        AI, pop     -af(1)
```


<expression>

Le code généré pour cette règle de syntaxe est le même que pour l'expression finale.

Après l'exécution de l'expression, la pile ml est mise à jour en fonction des paramètres de sortie par la macro-instruction PARS

```
        PARS    n
```

n = nombre de paramètres en sortie

qui est traduit en :

```
        AI, pop     af(1)
```

Ensuite le contrôle est rendu au programme de commande appl par l'instruction

```
        B       @RPL
```

En fin de chaque rangée on constitue une table des adresses des procédures. Dans le cas où une procédure manque, son adresse est remplacée par @RPL, (cf. V.1.).

```
        RET     1
étiquette DATA  ....
          :
          DATA  étiquette de début de      1 ≤ i ≤ q
        q :     la i ième procédure
          :     ou @RPL
          DATA  ....
```

étiquette = nom de la rangée

q = nombre de procédures déclarées

Exemple : compilateion de la rangée de procédures définie plus haut

```
@i    EQU     $        @k    EQU     $
      PARE    0              PARE    2
      :                      :
      PARS    5              PARS    3
      B       @RPL           B       @RPL
@J    EQU     $              RES     1
      PARE    7        gen   DATA    @i
      :                      DATA    @j
      :                      DATA    @RPL
      PARS    0              DATA    @k
```

4.4. Expression fermée

Une <expression fermée> (cf. II. 2. 7.) comprend des déclarations locales des instructions d'affectation ou des expressions. Le code généré dépend de la fonction (addition, affectation, ...) mais aussi du type de l'opérande. Dans les appels de macro-instructions, celui-ci est indiqué comme argument par les marqueurs suivants :

1) 1RP un repère ou une rangée de repères globale
2) 1PAR un paramètre d'une procédure
3) 1ML un résultat intermédiaire ou un paramètre d'une rangée de procédures
4) 1CTE un élément

Pendant l'exécution d'une expression on utilise un seul registre d'opérande [1] (1op). Les résultats intermédiaires d'un calcul restent dans le registre 1op. Si le contexte le demande il est rangé sur la pile ml en tant que résultat intermédiaire. Cette méthode accélère la compilation car les tests à faire au sommet de la pile ml sont limités. Elle nuit peu à l'exécution car le langage $Face_0$ possède peu d'opérateurs de priorités différentes (il ne possède pas les opérateurs multiplicateurs et logiques). En outre un programme écrit en $Face_0$ pour la compilation d'un langage possède peu

(1) Le système $sface_0$ est prévu pour l'utilisation de un ou plusieurs registres d'opérandes.

d'expressions arithmétiques compliquées.

L'expression finale d'un programme $Face_0$ commence par l'appel de la macro-instruction INIT sans arguments qui comprend les références externes aux procédures standard et l'initialisation de l'index pop par l'instruction :

```
LI, pop        ml-1
```

où ml est l'adresse de départ de la pile ml .
On termine l'expression finale par un appel de la macro-instruction ARRT qui rend le contrôle au moniteur par l'instruction :

```
CALI, 9        1
```

4.4.1 - Liste de déclarations locales

<liste de déclarations locales> (cf. II.2.7.1.-a)

En $Face_0$ les déclarations locales permettent uniquement de repérer des informations déjà existantes. Les pointeurs des informations repérées se trouvent sur la pile ml. Pour l'expression qui les contient ils ont la même fonction que les paramètres formels d'une procédure. Les valuations ou valrang définissent les informations repérées et correspondent aux paramètres effectifs d'une procédure.

<déclaration locale de repères> (cf. II.2.7.2.-c)
<déclaration locale de rangée de repères> (cf. II.2.7.1.-c)

L'adresse d'une déclaration est définie par rapport à l'index pop. Elle est générée par l'instruction suivante :

```
étiquette      SET      n
```

étiquette = nom d'une déclaration locale
n = une adresse définie par rapport à pop

<valuation> (cf. II.2.7.1.-d)

En fonction du <secondaire> de la valuation des appels de macro-instructions différents sont générés. Dans le cas où le <secondaire> est :

1) un identificateur
on génère la macro-instruction PREP (cf. 4.4.6.)

2) un identificateur indicé
on génère la macro-instruction ETPA identique à ETAF utilisée pour l'affectation à un identificateur indicé (cf. 4.4.2).

3) un appel de procédure
on range le résultat élément sur la pile mi (cf. 4.2). Pour réaliser le repérage qui existe pour les autres déclarations locales on est obligé de le créer sur la pile. Pour cela on utilise la macro-instruction DLPR dont l'argument af(1) est une adresse définie par rapport à pop.

DLPR n

n = adresse définie par rapport à pop

est traduit par le métassembleur en :

LI, i1	n+1
ADW, i1	pop
STW, i1	n, pop

<valrang> (cf. II.2.7.1.-f)

Le primaire peut-être un identificateur ou un appel d'une procédure standard. Dans le premier cas on génère la macro-instruction PRAN (cf. 4.4.6.) et dans le deuxième seulement un appel de procédure.

Exemple : de déclarations locales

rangée(10) rep chaîne ; rangée(2) rep indic ;
rangée(20) rep info ; proc elt filtre (rep Perit)... ;
(rep Prep = entrée 2 (table ; indic ; info ; filtre) ; rangée
rep Prangée = recherch (table ; indic ; filtre ; erreur),
Pr = chaîne ; ...)

Le langage généré par le compilateur est le suivant :

Prep	SET	1
	DLPR	1
	PREP	table, 3, 1RP
	PRAN	indic, 4, 1RP
	PRAN	info, 5, 1RP
	PREP	filtre, 6, 1RP
	APRO	entrée 2, 8
Prangée	SET	3
	PREP	table, 4, 1RP
	PRAN	indic, 5, 1RP
	PREP	filtre, 6, 1RP
	PREP	erreur, 7, 1RP
	APRO	recherch, 9
Pr	SET	4
	PRAN	chaîne, 4, 1RP

Ces appels de macro-instructions sont traduits en :

```
Prep      SET     1
          LI, il  2
          AW, il  pop
          STW, il 1, pop
          LI, il  table
          STW, il 3, pop
          LW, il  indic
          STW, il 4, pop
          LW, il  info
          STW, il 5, pop
          LI, il  filtre
          STW, il 6, pop
          STW, pop 7, pop
          LI, il  $+4
          STW, il 8, pop
          AI, pop 8
          B       entrée2
          LW, pop -1, pop
Prangée   SET     3
          LI, il  table
          STW, il 4, pop
          LW, il  indic
          STW, il 5, pop
          LI, il  filtre
          STW, il 6, pop
          LI, il  erreur
          STW, il 7, pop
          STW, pop 8, pop
          LI, il  $+4
          STW, il 9, pop
          AI, pop 9
          B       recherch
          LW, pop -1, pop
Pr        SET     4
          LW, il  chaîne
          STW, il 4, pop
```


4.4.2 - Suite d'instructions

<instruction> (cf. II.2.7.2.-b)

Une instruction peut-être une expression ou une instruction d'affectation. Pour permettre l'exécution d'une expression simple et le passage à l'instruction suivante, chaque expression se termine par une étiquette.

Exemple : (a = b : alpha(n), alpha(m) ; ....)

après l'appel de alpha(m) une étiquette de fin d'expression est générée.

<instruction d'affectation> (cf. II.2.7.2.-c)

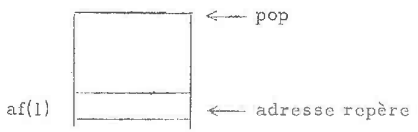
La valeur de l'expression à droite du signe d'affectation se trouve dans le registre lop.
Les instructions nécessaires à l'affectation sont générées par un appel de la macro-instruction AFFC dont les arguments sont l'opérande (af(1)) et son type (af(2)) :

a) dans le cas d'un repère global

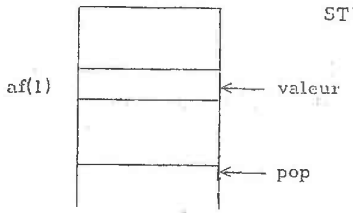
```
STW, lop        af(1)
```

b) dans le cas d'un paramètre repère ou d'une adresse rangée sur ml .
Si une affectation à un identificateur indicé a lieu, l'adresse de la composante est rangée sur ml (cf. 4.4.3.1.).

```
LW, il          af(1), pop
STW, lop        *il
```



c) dans le cas d'un repère paramètre d'une rangée de procédures



Exemple : (Pelt:='elt' ; P5:=a ;....

Sortie du compilateur :

[LW, lop = 'elt']

1) AFFC Pelt, lpar
[LW, lop a]

2) AFFC P5, lrep

Les deux lignes de l'appel de AFFC sont traduites en :

1) LW, il Pelt, pop
STW, lop *il

2) STW, lop 5, pop

4.4.3 - Expression

<expression> (cf. II.2.7.3.-a)

L' <expression simple> à gauche d'un signe ':' sert de condition. Dans le cas où elle est vérifiée le registre lop est différent de zéro. En fin de condition lop est testé pour savoir si le programme se poursuit en séquence ou s'il y a un branchement à une <expression simple> plus loin.

L'argument (af(1)) de la macro-instruction FCOND est l'étiquette de cette <expression simple> . Les instructions générées sont les suivantes :

CI, lop o
BE af(1)

Exemple :

x+5=z+delta :....,.... ;

la comparaison effectuée, on génère la fin de la condition

FCOND @i

@i = étiquette générée par le compilateur

qui est traduite en :

CI, lop o
BE @i

4.4.4 - Expression simple

Avant d'effectuer une opération d'addition ou de comparaison l'un des deux opérandes doit se trouver dans un registre. Les instructions nécessaires au transfert des informations entre registre et mémoire sont généré par les macro-instructions de chargement (CHAR) et de rangement (RAN) de registre.

1) Chargement d'un registre

Les arguments de la macro-instruction CHAR sont les suivants :

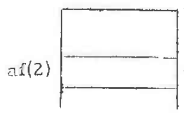
- le nom du registre : af(1)
- l'opérande : af(2)
- le type de l'opérande : af(3)

Les instructions générées en fonction du type de l'opérande sont :

a) dans le cas d'un repère global (af(3)=1RP)

LW, af(1) af(2)

b) dans le cas d'un repère paramètre formel (af(3)=1PAR) une indexation et une indirection sont nécessaires pour amener la valeur de l'opérande dans le registre

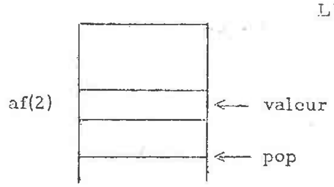


LW, il af(2), pop
LW, af(1) *il

c) dans le cas d'un résultat intermédiaire sur la pile ml ou d'un paramètre formel d'une rangée de procédures (af(3)=1ML)

Une indexation suffit pour charger la valeur

LW, af(1)	af(2), pop



d) dans le cas d'un élément (af(3)=1CTE)

LW, af(1)	=af(2)

Exemple : chargement du paramètre Pa dans le registre 2op

CHAR	2op, Pa, 1PAR

Cet appel est traduit par le métassembleur en :

LW, il	Pa, pop
LW, 2op	*il

2) Le rangement d'un registre

Le contenu d'un registre est rangé sur la pile ml.
Les arguments de la macro-instruction RANG sont :

- le nom du registre : af(1)
- une adresse de la pile ml définie par rapport à pop

Un appel de RANG est traduit en :

STW, af(1)	af(2), pop

Exemple :

l'appel	RANG	1op, 5
générera	STW, 1op	5, pop

<expression simple> (cf. II. 2. 7. 4. -a)

L'expression simple à gauche d'un repère ':' peut contenir un opérateur de relation. Cette opération est réalisée par une macro-instruction de comparaison à noms multiples :

PG	donnant les relations	plus grand
GE		plus grand ou égal
EG		égal
DF		différent
PE		plus petit ou égal
PF		plus petit

Les arguments de cette macro-instruction sont :

- le registre d'opérande : af(1)
- l'opérande : af(2)
- le type de l'opérande : af(3)

On compare d'abord les deux opérandes puis en fonction de la relation donnée on fait des branchements conditionnels.

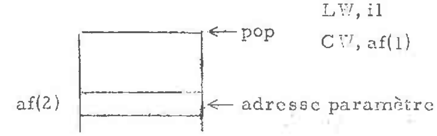
Les instructions générées pour la comparaison diffèrent selon le type de l'opérande qui peut être :

a) un repère global

CW, af(1)	af(2)

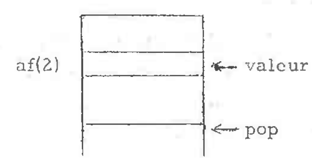
b) un repère paramètre formel

LW, il	af(2), pop
CW, af(1)	*il



c) un résultat intermédiaire ou un paramètre d'une rangée de procédures

CW, af(1)	af(2), pop



d) un élément

CW, af(1)	=af(2)

Les branchements conditionnels permettent de charger 0 ou 1 dans le registre lop :

```
com       $+3
LI, lop   o
B         $+2
LI, lop   1
```

com = commande de branchement (BG, BGE, BE, BNE, BLE, BL) générée en fonction du nom donné à l'appel

Exemple : Pml > 0 : ..... ;

Le compilateur génère les appels suivants :

```
CHAR      lop, Pml, lPAR
PG        lop, o, lCTE
```

qui sont traduits en :

```
LW, il    Pml, pop
LW, lop   *il
CW, lop   =o
BG        $+3
LI, lop   o
B         $+2
LI, lop   1
```

4.4.5 - Expression arithmétique

<expression arithmétique> (cf. II.2.7.4.-c)

Les opérations admises sont l'addition et la soustraction. Elles sont générées par une macro-instruction à deux noms : ADDI et SOUS. Les arguments et les instructions générées sont les mêmes que pour les opérations de relations avec deux exceptions :

1) la commande CW est changée en AW ou SW
2) la partie pour les branchements conditionnels n'est pas générée.

Exemple : ml+no+2 ; .....

Les appels générés par le compilateur sont les suivants :

```
CHAR      lop, ml, lrp
ADDI      lop, no, lrp
ADDI      lop, 2, lCTE
```

Ils sont traduits en :

```
LW, lop   ml
AW, lop   no
AW, lop   =2
```

4.4.6 - Secondaire

4.4.6.1 - Identificateur indicé

Deux cas se distinguent suivant que la composante se trouve à gauche d'un signe d'affectation ou non. Dans le premier cas, c'est l'adresse de la composante qui est rangée sur la pile ml (macro-instruction ETAF) sinon ou j range sa valeur (macro-instruction ETEX).

Les arguments des deux macro-instructions sont les mêmes :

- l'identificateur de la rangée : af(1)
- le type de l'identificateur : af(2)
- une adresse définie par rapport au registre pop af(3)
- l'indice : af(4)
- le type de l'indice : af(5)

<indice> (cf. II.2.7.5.-c)

Les instructions générées par ETAF et ETEX pour l'indice sont les suivantes :

α) dans le cas d'un indice repère global

```
LW, il    af(4)
```

β) dans le cas d'un indice paramètre formel

```
LW, il    af(4), pop
LW, il    *il
```

γ) dans le cas d'un indice paramètre d'une rangée de procédures

LW, i1	af(4), pop

δ) dans le cas d'un élément

LW, i1	=af(4)

<identificateur> [<indice>] (cf. II. 2. 7. 5. -a)

L'indice est vérifié à l'exécution. Dans le cas de débordement le programme se branche à l'étiquette lerrex où un message est imprimé et l'exécution du programme arrêtée. La suite des instructions générées dépend du type de l' <identificateur>.

Instructions générées pour ETAF		Instructions générées pour ETEX	

α) dans le cas d'une rangée de repères globale

BLEZ	lerrex
CW, i1	*af(1)
BG	lerrex

AW, i1	af(1)	LW, i1	*af(1), i1
STW, i1	af(3), pop	STW, i1	af(3), pop

β) dans le cas d'une rangée de repères paramètre formel

BLEZ	lerrex
LW, i2	af(1), pop
CW, i1	*i2
BG	lerrex

AW, i2	i1	LW, i1	*i2, i1
STW, i2	af(3), pop	STW, i1	af(3), pop

Exemple :

info [3] := delta+1 ;

Le compilateur génère les appels suivants de macro-instructions :

ETAF	info, 1RP, n, 3, 1CTE
CHAR	1op, delta, 1RP
ADDI	1op, 1, 1CTE
AFFC	n, 1ML

n = adresse définie par rapport à pop

ce qui traduit en :

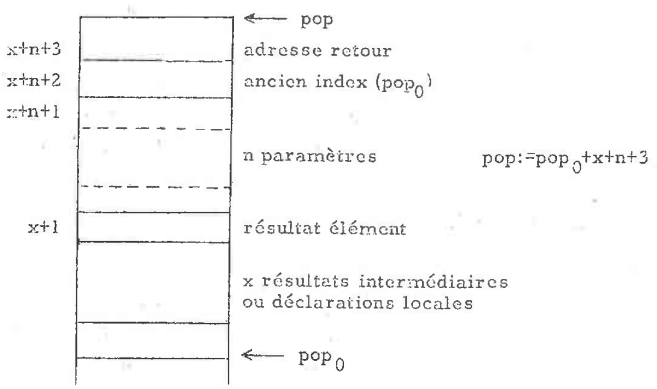
LW, i1	=3
BLEZ	lerrex
CW, i1	*info
BG	lerrex
AW, i1	info
STW, i1	n, pop
LW, 1op	delta
AW, 1op	=1
LW, i1	n, pop
STW, 1op	*i1

4.4.6.2 - Appel de procédure

L'appel récursif des procédures exige une pile statique où sont rangées les informations relatives à un appel donné :

1) le résultat éventuel de la procédure
2) l'adresse des paramètres effectifs
3) l'ancien index pop
4) l'adresse de retour

Exemple de disposition de la pile ml dans le cas d'une procédure élément à n paramètres :



L'appel d'une procédure se fait en plusieurs étapes :

1) ranger sur la pile ml l'adresse des paramètres, l'ancien index, l'adresse de retour

2) incrémenter le pointeur pop

3) branchement effectif à la procédure

4) au retour restauration de l'ancien pop

<paramètre effectif> (cf. II.2.7.5.-f)

Le passage d'un paramètre est différent selon qu'il est un élément, un repère ou une rangée de repères.

* premier cas : paramètre élément

La macro-instruction (PELT) effectue le passage des paramètres élément. Ses arguments sont l'élément (af(1)) et une valeur ml de la pile (af(2)).

```
LW, il          =af(1)
STW, il         af(2), pop
```


* deuxième cas : paramètre repère

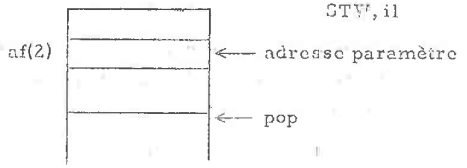
Le passage d'un paramètre repère global, procédure, rangée de procédure s'effectuent de la même façon. Les arguments de la macro-instruction (PREP) sont :

- l'identificateur du paramètre : af(1)
- une adresse de la pile ml définie par rapport à pop :(af(2))
- le type du paramètre : af(3)

Les instruction générées en fonction du type du paramètre sont :

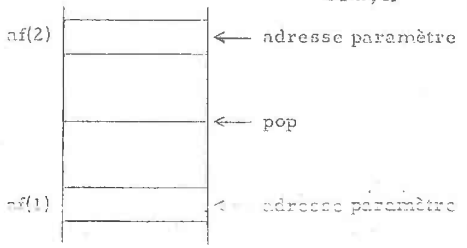
α) dans le cas d'un repère global, d'une procédure ou d'une rangée de procédures

```
LI , il         af(1)
STW, il         af(2), pop
```



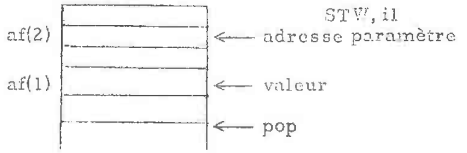
β) dans le cas d'un repère paramètre formel

```
LW, il          af(1), pop
STW, il         af(2), pop
```



γ) dans le cas d'un paramètre d'une rangée de procédures

```
LI, il          af(1)
AW, il          pop
STW, il         af(1), pop
```



* troisième cas : paramètre rangée de repères

Dans le cas des rangées de repères la macro-instruction (PRAN) effectue le passage du paramètre. Ses arguments sont ceux de PREP.

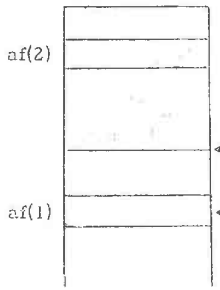
Les instructions générées par FRAN en fonction du type de paramètre sont :

α) dans le cas d'une rangée de repères globale, l'adresse de la composante 0 est rangée sur ml

LW, i1	af(1)
STW, i1	af(2), pop

← adresse composante zéro

β) dans le cas d'une rangée de repères paramètre formel, l'adresse de la composante 0 trouvée dans l'appel précédent est rangée sur ml.



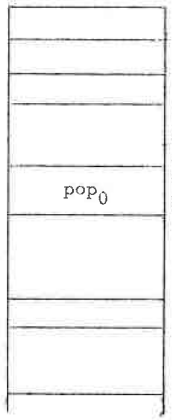
LW, i1	af(1), pop
STW, i1	af(2), pop

← pop

← adresse composante zéro

<appel de procédure> (cf. II.2.7.5.-d)

Avant le branchement à la procédure, l'ancien pointeur et l'adresse de retour sont rangés sur la pile ml. Les arguments de la macro-instruction (APRO) sont l'identificateur de la procédure (af(1)) et une adresse (af(2)) définie par rapport à pop.



STW, pop	af(2)-1, pop
LI, i1	$+4
STW, i1	af(2), pop
AI, pop	af(2)
B	af(1)
LW, pop	-1, pop

Dans le cas d'une procédure instruction, l'emplacement réservé au résultat n'est pas généré.

Exemple : appel d'une procédure sans résultat ayant trois paramètres, 2 repères et une rangée de repères.

... ; f(p3 ; a ; ranrep) ; .....

Nous supposons que l'incrément de la pile ml par rapport à pop avant l'appel est 3.

Le compilateur génère les appels de macro-instructions suivantes :

		Ces lignes sont traduites en :	
PREF	3, 4, 1ML	LI, i1	3
PREP	a, 5, 1RP	AW, i1	pop
PRAN	ranrep, 6, 1RP	STW, i1	4, pop
APRO	f, 8	LI, i1	a
		STW, i1	5, pop
		LW, i1	ranrep
		STW, i1	6, pop
		STW, pop	7, pop
		LI, i1	$+4
		STW, i1	8, pop
		AI, pop	8
		B	f
		LW, pop	-1, pop
		⋮	

CHAPITRE IV

ANALYSE DES MODULES DU COMPILATEUR

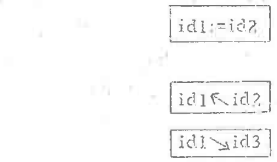
Le compilateur est formé d'un programme principal et de plusieurs modules. Chaque module correspond à un paragraphe de règles syntaxiques. Son adresse de retour dans le programme appelant est l'adresse qui suit l'appel.

Pendant la compilation on ne connaît pas la taille que peut prendre la pile ml à l'exécution, mais on connaît les variations qu'elle subit par <expression>. Elles sont suivies à la compilation par un compteur qui a le même nom que la pile c. a. d. ml.

La description des modules du compilateur est faite en fonction des paragraphes de la grammaire de $Face_0$.

1 - CONVENTION DE DESCRIPTION

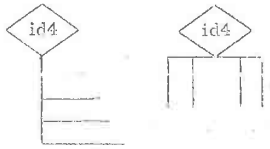
Une partie des modules est décrite par des organigrammes d'analyse organique. Les principales conventions utilisées dans les figures sont les suivantes :



affectation de la valeur de id2 à id1

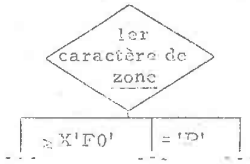
empiler id2 sur la pile id1

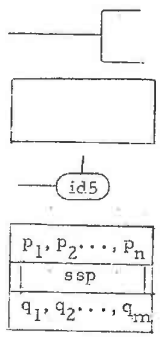
retirer l'information du sommet de la pile id1 et ranger dans id3



comparaison de id4 avec les éléments indiqués dans les branches. Les conditions sont vérifiées de gauche à droite ou de haut en bas

Exemple :





commentaire

fonctions expliquées par les libellés qu'elles contiennent

branchement à l'étiquette id5

appel du module ssp avec n paramètres p_i en entrée et m paramètres q_j en sortie

X'α' α = nombre hexadécimal

'ρ' ρ = suite de caractères

Pour simplifier les organigrammes certaines actions répétées ne sont pas indiquées :

1. Les appels du module bindec dans le cas où un nombre binaire devient argument d'un appel de macroinstruction.
2. Les paramètres en entrée du module gen qui sont eti, com, arg1, ..., arg5 (cf. 2). Souvent la notation plus concise gen(eti EQU $) est utilisée.
3. Les paramètres en sortie zone, drap, attente, du module d'analyse morphologique (anamorph) (cf. 3).
4. Les appels d'anamorph (cf. 3) n'ayant pas de signification dans la suite de la compilation.

2 - LES MODULES DE SERVICE

A part les modules qui constituent le corps du compilateur, il existe quatre modules de service :

- conversion binaire - décimal (module bindec)

  Un nombre binaire est converti en sa représentation de chiffres EBCDIC. Le passage des paramètres se fait par les registres. Dans R6 se trouve la valeur binaire à convertir et dans R7 l'octet à partir duquel le nombre converti est cadré à gauche.

- conversion décimal-binaire (module decbin)

  Il s'agit de convertir un nombre entier lu dans le programme source en un nombre binaire en vue d'un traitement dans le compilateur. A l'appel, l'adresse d'octet du chiffre le plus significatif du nombre à convertir se trouve dans le registre R6. Le passage du résultat au programme appelant se fait par nbbin.

- générateur d'étiquettes (module creeti).

  L'étiquette est formée de la lettre @ suivie d'un nombre décimal. Ce nombre est incrémenté à chaque fois qu'on fait appel au module. Le paramètre en sortie est zoneti qui contient l'étiquette générée.

- générateur des appels de macroinstructions (module gen)

  Ce module concatène les arguments d'un appel de macroinstruction, insert des virgules entre eux et sort les appels dans le fichier exec repris ensuite par le métaassembleur. Les paramètres en entrée du module sont eti, com, arg1, ..., arg5.

3 - ELEMENTS DE BASE DU LANGAGE

L'analyse morphologique du programme source est effectuée par le module anamorph. En $Face_0$ les espaces n'ont pas de signification excepté dans les chaînes formées. Pour savoir où se termine une unité syntaxique il faut donc lire le caractère significatif qui suit. Il est rangé dans attente. Même pour les signes, on procède de cette façon excepté le signe ')' où l'on met C dans le caractère d'attente. Le dernier caractère d'un programme $Face_0$ étant une parenthèse fermante ceci permet de terminer la compilation sans avoir recours à un enregistrement 'fin de fichier' (!EOD).

Les paramètres en sortie d'anamorph sont :

1. zone qui contient l'unité syntaxique autre qu'un caractère spécial
2. drap qui contient le type de l'unité syntaxique lue :

	indicateur dans drap
identificateur	'ID'
constante	'NB'
chaîne fermée	'CH'
déclareur	'DC'
caractère spécial	un nombre hexadécimal compris entre X'01' et X'11'

3. attente contient zéro ou le caractère significatif qui suit l'unité syntaxique traitée.

En $Face_0$, chaque déclareur est précédé et suivi d'un point pour pouvoir le différencier des identificateurs. Il est transféré dans zone et son indicatif dans drap est 'DC'.

<caractère spécial> (cf. II. 2. 1. 2-c)

La correspondance entre les caractères spéciaux et leur valeur numérique est faite par une table tsigne. On lit deux caractères EBCDIC et puis on les compare à ceux contenus dans tsigne. Si le signe est formé d'un seul caractère EBCDIC on range le $2^{ème}$ dans attente sinon on fait une lecture supplémentaire et c'est le caractère lu qu'on y range (exception : parenthèse fermante).

Le paramètre de sortie drap contient une valeur binaire comprise entre 0 et 17.

Définition $Face_0$	caractères implémentés	valeur numérique
÷	+	1
-	-	2
<	<	3
≤	=<	4
=	=	5
≠	<>	6
≥	>=	7
>	>	8
:=	:=	9
.	:	10
,	,	11
;	;	12
(	(	13
)	)	14
[	(.	16
]	.)	17

<chaîne fermée> (cf. II. 2. 1. 2-d)

Une chaîne fermée est reconnue par l'apostrophe qui constitue son premier caractère. On atteint la fin d'une chaîne fermée si on trouve un nombre pair d'apostrophes et que le caractère dans attente est différent d'une apostrophe. La chaîne est transférée dans zone et l'indicatif 'CH' dans drap.

<entier> (cf. II. 2. 1. 3-c)

L'entier est reconnu par le 1. chiffre qui le constitue. Il est transféré dans zone et drap contient l'indicatif 'NB'.

<identificateur> (cf. II. 2. 1. 2-c)

L'identificateur reconnu par la première lettre qui le constitue est transféré dans zone et l'indicatif 'ID' rangé dans drap.

4 - PROGRAMME

<programme> (cf. II. 2. 2-a)

Au début de la compilation le langage objet généré est initialisé par :

- l'ouverture du fichier exec ;
- la génération de l'appel au système métaassembleur $sface_0$:

SYSTEME $SFACE_0$

Ensuite le contrôle est donné au module aiguillage. Après la compilation du langage source, le contrôle est rendu au programme principal pour :

- générer la fin du langage objet et indiquer l'adresse de début d'exécution du programme $Face_0$ (adresse $@_0$)
- fermer le fichier exec,
- rendre le contrôle au système d'exploitation

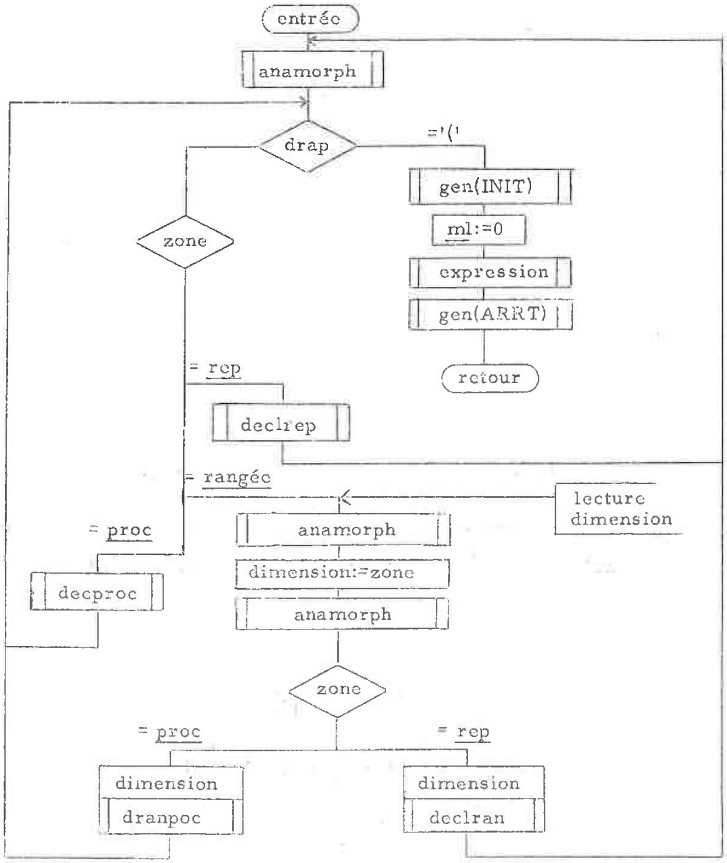
5 - LISTE DE DECLARATIONS

<liste de déclarations> (cf. II. 2. 4-a)

<déclaration> (cf. II. 2. 4-b)

Le module aiguillage examine le déclareur ou le signe devant lequel il se trouve et donne le contrôle au module correspondant.

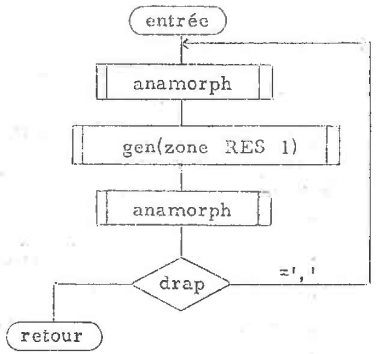
Module aiguillage.



Le paramètre en sortie dimension qui contient la taille des rangées de repères et de procédures.

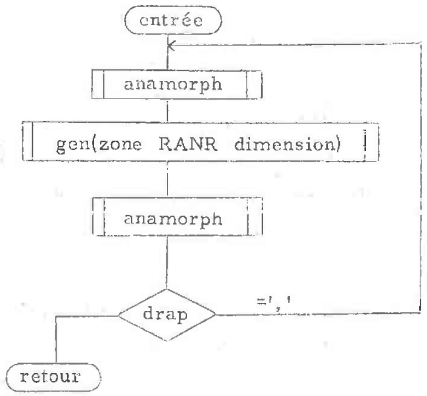
<déclaration de repère> (cf. II. 2. 4-d)

Les déclarations de repères sont générées par le module declrep.



<déclaration de rangée de repères> (cf. II. 2. 4-h)

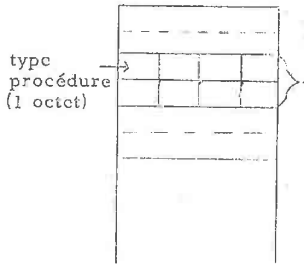
Les déclarations de rangées de repères sont générées par le module declran dont le paramètre en entrée est dimension.



6 - DECLARATION DE PROCEDURE

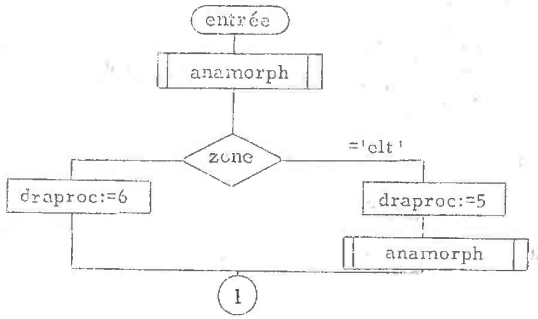
6. 1. Déclaration de procédure (module decproc)

Pour chaque procédure déclarée, son type (élément ou sans résultat) et le type de ses paramètres formels (repères ou rangées de repères) sont rangés dans une table (tabproc). Ceci permet de générer correctement les instructions de mise en place des paramètres effectifs à l'appel de la procédure. La structure de cette table est la suivante :



	n°
élément	1
repère	3
rangée rep.	4
proc. elt	5
proc. sans résultat	6
rangée proc.	7

<résultats> (cf. II. 2. 5-b)

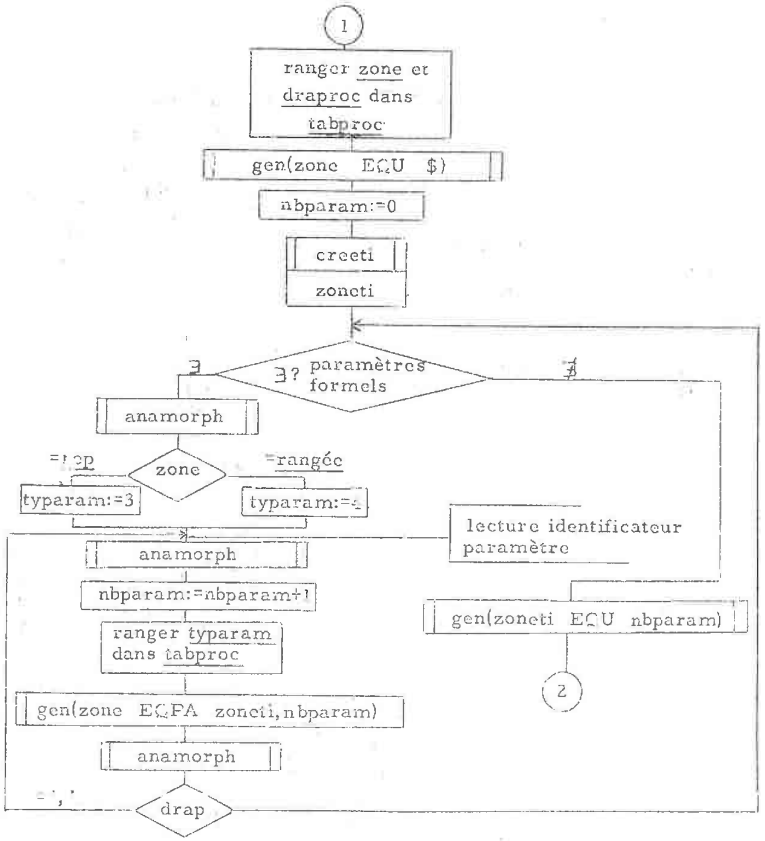


<corps de procédure> (cf. II. 2. 5-c)

Le corps de procédure est divisé en deux parties : la liste des paramètres formels et l'expression. Avant le traitement de <expression>

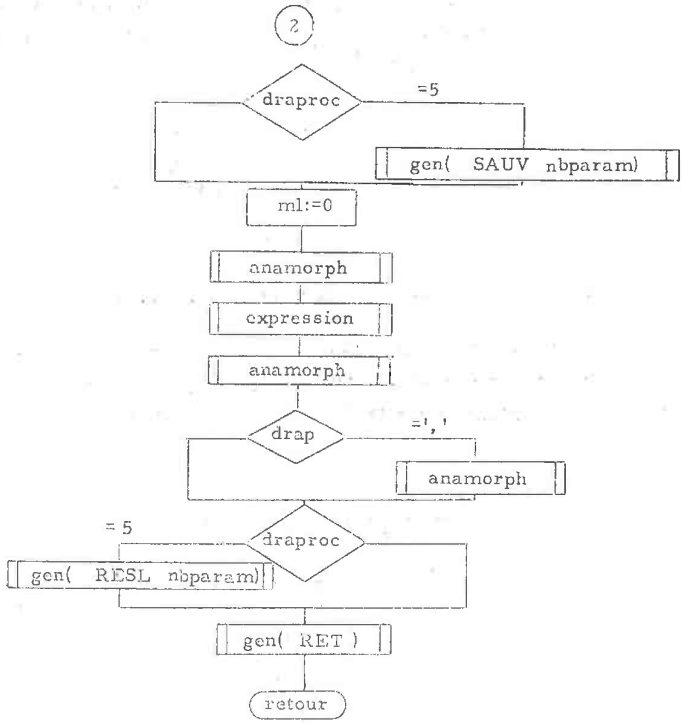
il faut tester si la procédure déclarée est une procédure élément et dans ce cas générer la sauvegarde du registre lop.

<liste de paramètres formels> (cf. II. 2. 5-d)



Dans le cas où un paramètre formel est rangée de repères, on range la valeur 4 dans tabproc sinon la valeur 3. Les appels de macro-instructions générés sont EQU et EQPA.

<expression> (cf. II. 2. 5-c)



Le résultat d'une procédure élément (draproc=5) est rangé sur la pile ml.

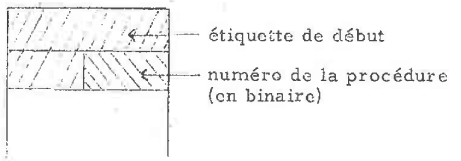
6. 2. Déclaration des procédures standard

Les procédures standard sont écrites en code machine et n'ont pas besoin d'être déclarées dans un programme $Facc_0$. Pour permettre une génération correcte de l'appel par procédure standard on range plusieurs informations dans tabproc :

1) l'identificateur de la procédure
2) son type
3) le type de chaque paramètre.

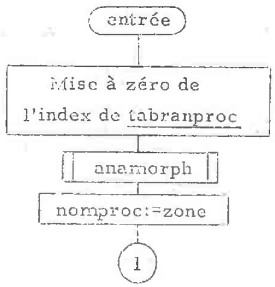
7 - DECLARATION DE RANGEE DE PROCEDURES (module dranproc)

Chaque numéro de procédure d'une rangée et son étiquette de début sont rangés dans la table tabranproc. La taille de cette table est de 1000 mots. Son organisation est la suivante :



<déclaration de rangée de procédures> (cf. II. 2. 6-a)

Le début d'une rangée de procédures est détecté par le module aiguillage qui donne le contrôle à dranproc. Le paramètre en entrée est dimension qui contient la taille de la rangée.



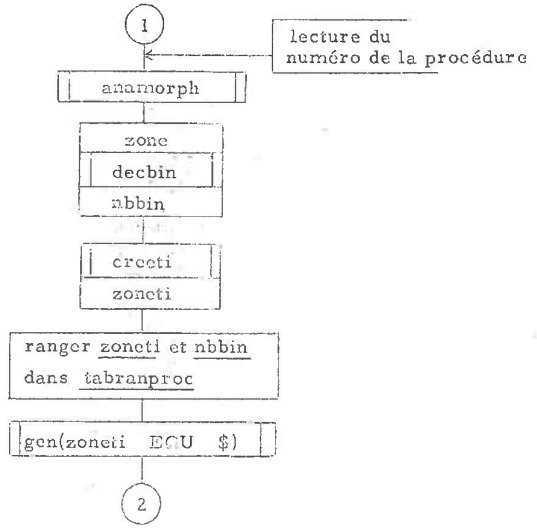
Les appels de macroinstructions générés par ce module sont :

PARE		gestion de la pile ml en fonction des parametres en entrée
PARS		même action mais pour les paramètres en sortie
B	@RFL	retour au programme de commande appl

En fin de compilation de la rangée, on génère la table contenant les adresses de début des procédures.

<liste de procédures de la rangée> (cf. II. 2. 6-b)

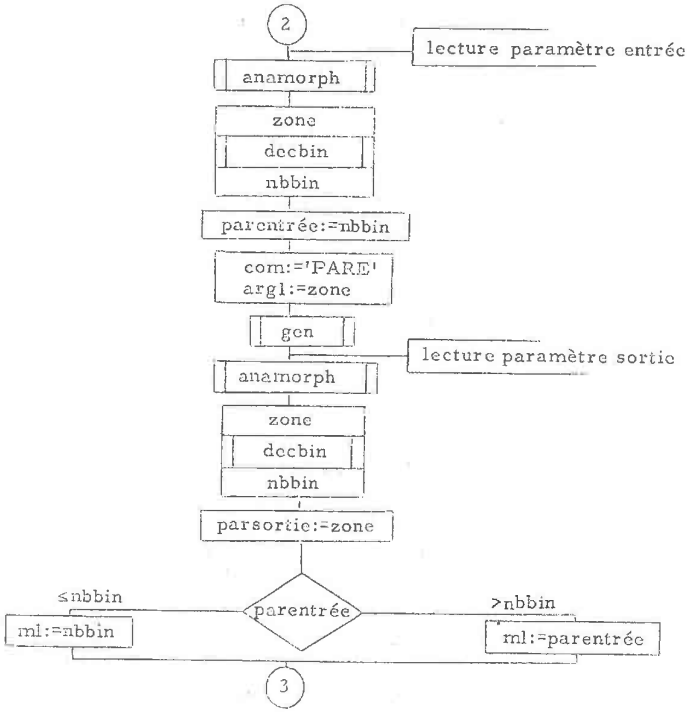
<identificateur> [<entier décimal>]



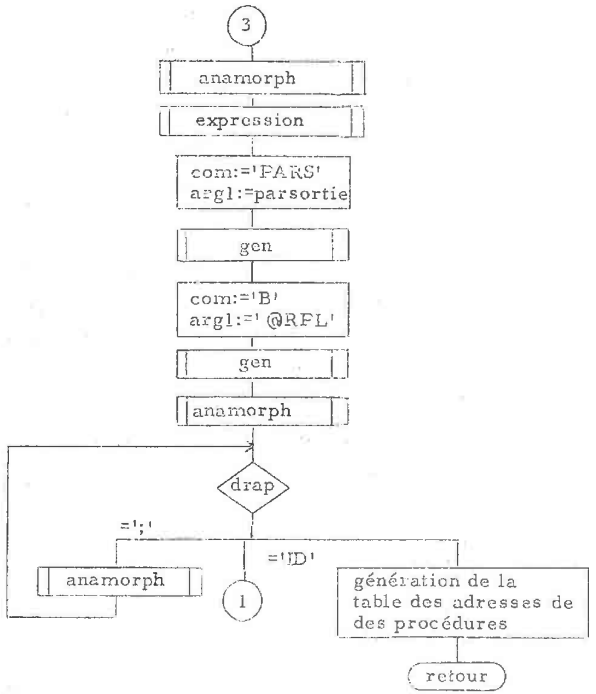
<désignation de la procédure> (II. 2. 6-c)

<en tête>

A l'exécution, le pointeur pop varie en fonction des paramètres en entrée. S'il y a n paramètres en entrée et m paramètres en sortie le compteur ml est incrémenté de sup(n, m).



<expression> (cf. II.2.6-c)



Dans le cas où une procédure de la rangée est vide, on remplace son adresse de début par l'adresse de retour dans appl (@RPL) (cf. III.4.3).

8 - EXPRESSION FERMEE (module expression)

Le module expression est géré par la table des traitements (tabtraitement). Elle permet le branchement aux opérations à effectuer. Le retour se fait par un branchement à l'étiquette suitexp ou tabtrait pour continuer la génération de l'expression.

Les appels de macroinstructions générés par ce module sont :

INIT	initialisation
AFFC	affectation
FCOND	fin/condition
CHAR	chargement registre
RANG	rangement registre
FG...PP	comparaison
ADDI, SOUS	opérations arithmétiques

La pile des opérandes (opra) comporte 30 éléments, 3 mots mémoire sont nécessaires par élément. Plusieurs types d'informations sont rangés sur cette pile. Chacun d'eux est reconnaissable par son premier caractère. Pour le registre lop, le test se fait sur les 3 caractères.

type d'information	critère de reconnaissance
registre	lop
repères, rangées de repères globaux	toute lettre sauf p
repères, rangées de paramètres formels	lettre p
repère paramètres d'une procédure de rangée	$
résultat intermédiaire	00(binaire)
constante numérique	chiffre
constante alphanumérique	' (apostrophe)
étiquette	@...

La pile des opérateurs (oper) comporte 50 éléments. L'élément occupe un mot mémoire.

L'accès dans la table des traitements dépend du signe au sommet de oper et celui lu en entrée. Trois cas fondamentaux se présentent :

- le signe lu en entrée est plus prioritaire que celui au sommet de oper. On l'empile sur oper et on continue la lecture (suitexp).

- dans le cas contraire, on effectue le traitement lié à l'opérateur qui est au sommet et on le retire de la pile. Ensuite on compare la priorité du nouveau sommet avec le même signe toujours en entrée.
- certains signes de fin d'instruction ou d'expression simple (: ; ,) demandent un traitement sans pour autant décrémenter la pile. On passe directement à la lecture suivante (suitexp).

Disposition de la table des traitements

entrée

sommet oper ↓	+	-	<	≤	=	≠	≥	>	:=	:	,	;	(	)	#
1	addition	addition	addition	addition	addition	addition	addition	addition		addition	addition	addition	parouvr	addition	
+	addition	addition	addition	addition	addition	addition	addition	addition		addition	addition	addition	parouvr	addition	
16	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction		sous-traction	sous-traction	sous-traction	parouvr	sous-traction	
-	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction	sous-traction		sous-traction	sous-traction	sous-traction	parouvr	sous-traction	
31	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFF	compFF	compFF	compFF	parouvr	compFF	
<	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFE	compFE	compFE	compFE	parouvr	compFE	
46	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFE	compFE	compFE	compFE	parouvr	compFE	
≤	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFE	compFE	compFE	compFE	parouvr	compFE	
61	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compEG	compEG	compEG	compEG	parouvr	compEG	
=	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compEG	compEG	compEG	compEG	parouvr	compEG	
76	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compDF	compDF	compDF	compDF	parouvr	compDF	
≠	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compDF	compDF	compDF	compDF	parouvr	compDF	
91	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compGE	compGE	compGE	compGE	parouvr	compGE	
≥	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compGE	compGE	compGE	compGE	parouvr	compGE	
106	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFG	compFG	compFG	compFG	parouvr	compFG	
>	empiler	empiler	empiler	empiler	empiler	empiler	empiler	empiler	compFG	compFG	compFG	compFG	parouvr	compFG	
121	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp	fincond	fincond	finex-sim	affecta-tion	parouvr	affecta-tion	
:=	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp	fincond	fincond	finex-sim	affecta-tion	parouvr	affecta-tion	
136															
:															
151															
,															
166															
;															
181	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp	empiler	fincond	finex-sim	ignorer	parouvr	retirer	
(	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp							
196															
)															
211	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp		fincond	finex-sim	finexp	parouvr		
#	empiler	empiler	empiler	empil-comp	empil-comp	empil-comp	empil-comp	empil-comp							

<espression fermée> (cf. II. 2. 7-a)

<expression>

Le module expression initialise les piles opra et oper. Il crée une première étiquette qui sera rangée dans etifinex. Elle servira d'étiquette de fin d'instruction.

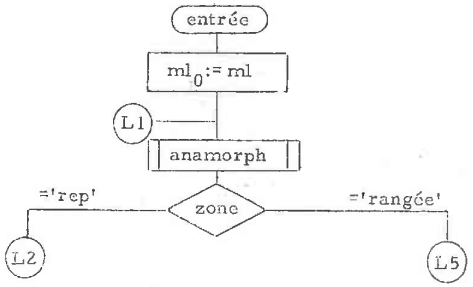
i = numéro du signe lu

j = numéro du signe au sommet de oper

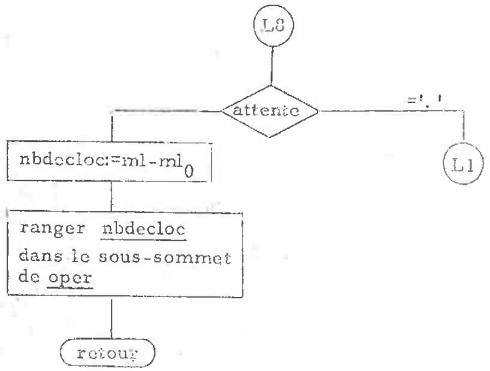
8. 1. Liste de déclarations locales (module declocal)

Les identificateurs déclarés locaux ont lieu uniquement à droite d'une parenthèse ouvrante. C'est le module expression qui donne le contrôle à declocal dans le cas où il détecte un déclareur après le signe (.

<liste de déclarations locales> (cf. II. 2. 7. 1-a)

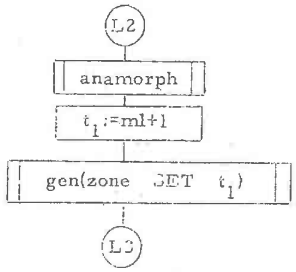


En fin de la liste de déclarations, la variation de la pile ml est rangée dans la pile oper. Ceci permet de décrémenter correctement le compteur ml en fin d'expression fermée.

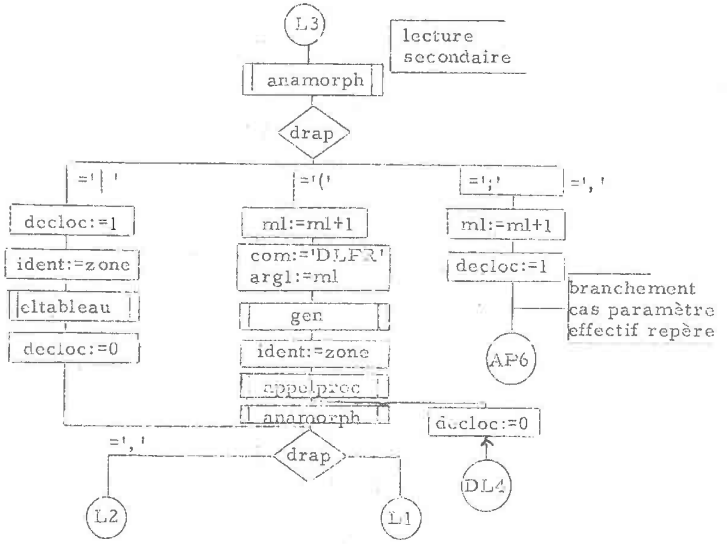


<déclaration locale de repère> (cf. II. 2. 7. 1-c)

Un repère peut-être initialisé par : - un identificateur repère
- un identificateur indicé
- un appel de procédure standard à résultat repère



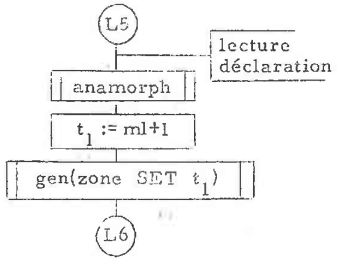
<valuation> (cf. II. 2. 7. 1-d)



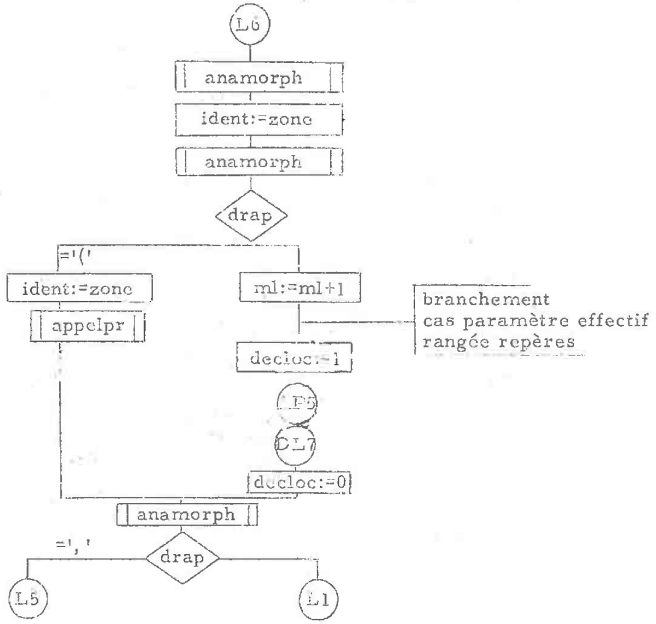
<déclaration locale de rangées de repères> (cf. II. 2. 7. 1-e)

Une rangée de repères est initialisée par

- une rangée de repères ou
- un appel de procédure standard à résultat rangée de repères



<valrang> (cf. II. 2. 7. 1-f)

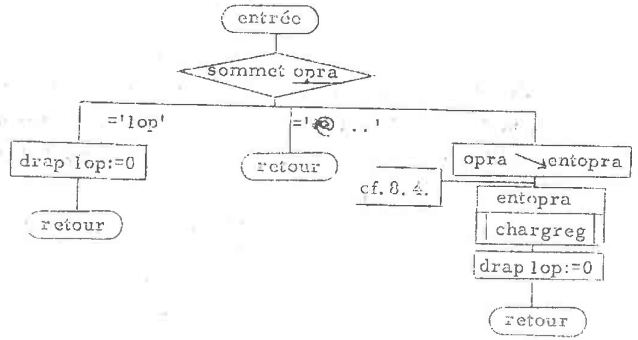


8. 2. Suite d'instructions

<instruction> (cf. II. 2. 7. 2-b)

La fin d'une instruction est détectée par la lecture d'une parenthèse fermante ou d'un point-virgule. Dans le premier cas c'est la fin de l'expression fermée et la parenthèse est traitée par retirer. Le deuxième cas est traité par ignorer ou finexp.

Le module testreg.
Si une expression a un résultat, il doit se trouver dans le registre lop. Le module testreg teste le sommet de la pile opra et génère un chargement de registre s'il y a lieu.

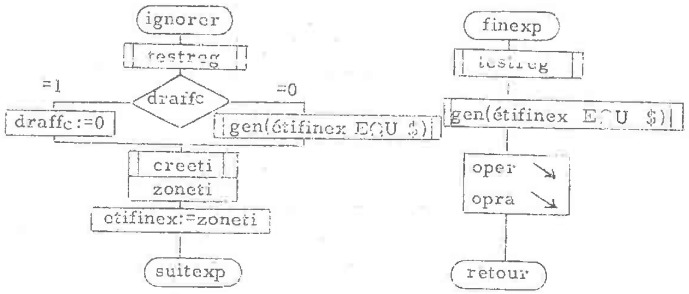


fonction ignorer :

l'instruction qui se termine est incluse dans une expression fermée

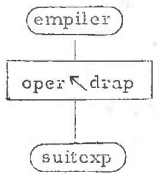
fonction finexp :

l'instruction est unique. C'est la fin de l'expression

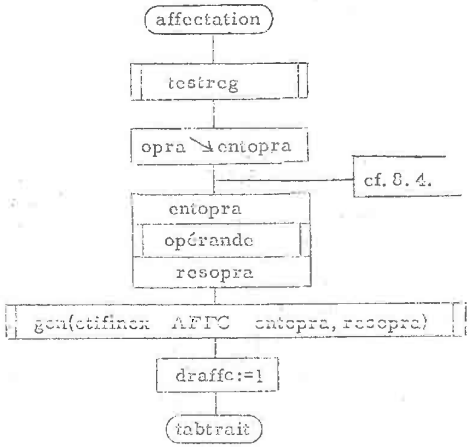


<instruction d'affectation> (cf. II. 2. 7. 2-c)

Deux traitements sont à prévoir : la reconnaissance de l'affectation et sa génération. Quand on lit le signe ':=' on le range sur la pile oper par la fonction empiler.



On génère l'affectation lorsqu'on lit un opérateur moins prioritaire que le signe d'affectation. Dans testreg on vérifie que le résultat de l'expression est bien dans le registre lop, sinon on le charge.



Dans le cas où draffc vaut 1 l'étiquette de fin d'expression a déjà été générée et ne le sera plus en fin d'instruction (fonction ignorer) ou en fin d'expression fermée (fonction retirer).

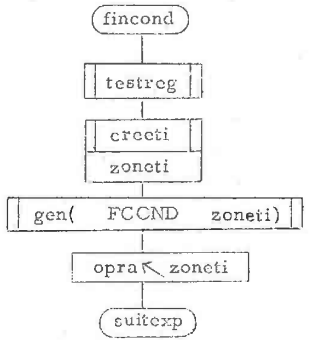
<expression> (cf. II. 2. 7. 3-a)

Dans le cas où une expression se termine par un point-virgule ou une parenthèse fermante ce sont les fonctions retirer, ignorer ou finexp qui la traitent.

Fin d'une condition. (fonction fincond)

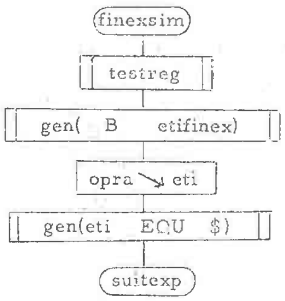
Soit $\mathcal{E}1 : \mathcal{E}2, \mathcal{E}3$ une expression

La lecture du signe ':' indique la fin d'une condition. Le compilateur génère un branchement conditionnel à $\mathcal{E}3$ et empile son étiquette de début.



Fin d'une expression fimple (fonction finexsim)

Il s'agit de faire exécuter un branchement inconditionnel en fin d'expression et de générer l'étiquette de début d'expression empilée par la fonction fincond.



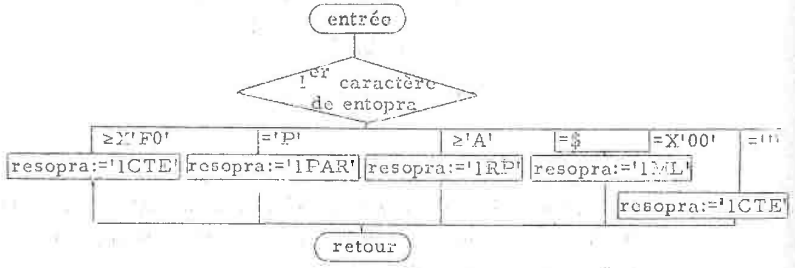
8.4. Expression simple

Le compilateur dispose de quatre modules qui préparent ou génèrent les appels de macroinstructions nécessaires pour les opérations d'une expression.

Module opérande.

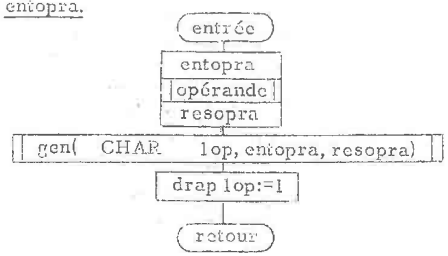
Le langage objet généré pour les opérations est différent selon que les opérandes sont des repères globaux des paramètres formels ou des résultats intermédiaires. Opérande reconnait le type d'un opérande et le range dans resopra. Le paramètre en entrée est entopra et contient l'identificateur à tester. En sortie resopra contient les indicateurs :

1RP = global
1PAR = paramètre formel
1ML = résultat intermédiaire
1CTE = constante



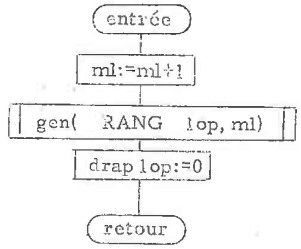
Module chargreg.

chargreg génère le chargement du registre. Le drapeau indiquant la disponibilité du registre (drap lop) est forcé à un. Le paramètre en entrée est entopra.



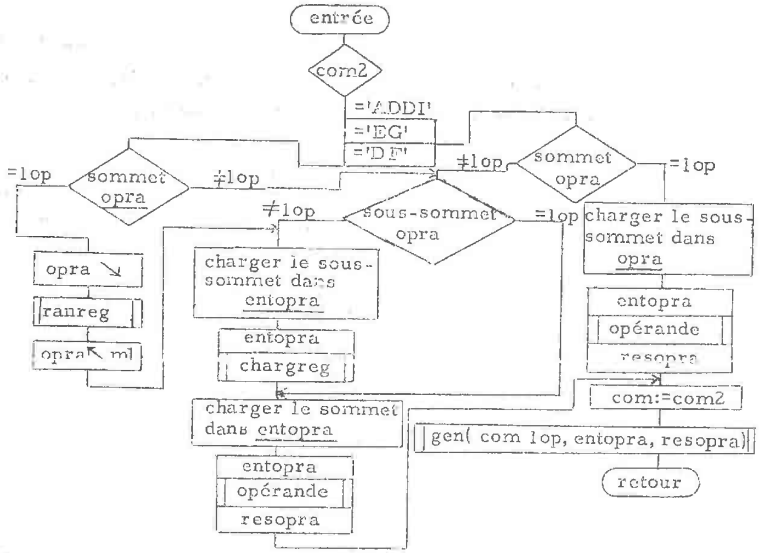
Module ranreg.

ranreg génère le rangement du registre. Le contenu est rangé sur la pile ml en tant que résultat intermédiaire.



Module genmacro.

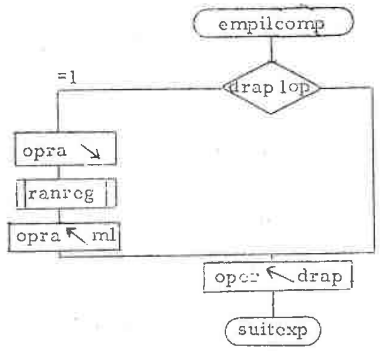
Ce module génère les appels des macroinstructions d'addition, de soustraction et de comparaison. Dans le cas où le registre lop est vide il faut d'abord générer un chargement de celui-ci.



Dans le cas des opérations commutatives le registre peut contenir le deuxième opérande.

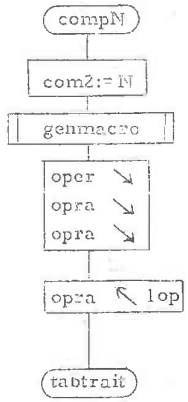
<expression simple> (cf. II. 2. 7. 4-a) (fonction empilcomp)

Lorsque le compilateur rencontre un opérateur de relation et que toutes les opérations plus prioritaires ont été générées, il empile cet opérateur sur oper. Si le registre lop est occupé sa valeur est rangée sur la pile ml.



(fonction compPP, . .)

C'est la lecture du signe ':' qui provoque la génération de l'opération de relation (fonctions compPP, compPE, . .)



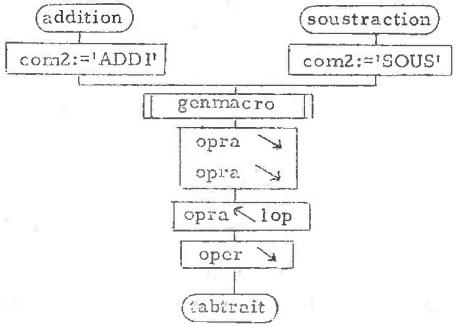
N = PP, PE, EG, DF, GE, FG

<expression arithmétique> (cf. II. 2. 7. 4-c) (fonction empiler)

Les signes + - sont les opérateurs binaires de plus haute priorité en Face$_0$. Dès leur rencontre ils sont empilés sur oper par la fonction empiler.

(fonctions addition, soustraction)

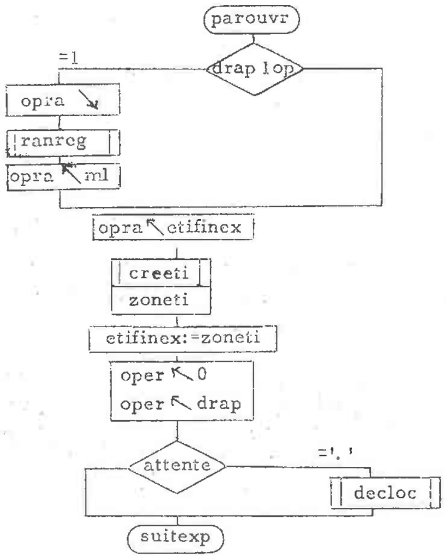
On génère les opérations arithmétiques si l'un des signes + ou - se trouve au sommet de la pile et qu'on vient de lire un signe ayant la même ou une moins haute priorité.



<facteur> (cf. II. 2. 7. 4-d) (fonction parouvr)

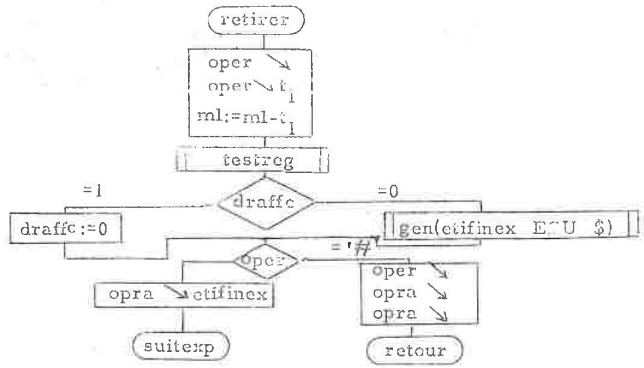
<expression fermée>

Une expression fermée est entourée de parenthèses. Avant de ranger la parenthèse ouvrante sur oper on empile la valeur 0. Cette valeur sera modifiée par le module declocal, s'il y a lieu. Elle représente la variation de la pile ml par les déclarations locales.



(fonction retirer)

Lorsque le compilateur lit une parenthèse fermante il vérifie que toutes les opérations ayant une priorité plus importante sont générées. La parenthèse ouvrante qui se trouve au sommet de oper est retirée. Puisqu'il s'agit de la fin d'une expression, on teste que le résultat éventuel se trouve dans le registre lop. Le compteur ml est décrémenté de la valeur qui se trouve au sommet de oper (cf. fonction parouvr).



8. 6. Secondaire

<secondaire> (cf. II. 2. 7. 5-a)

En Face$_0$ les secondaires sont considérées comme des opérandes. Dès leur reconnaissance dans le module expression, le compilateur donne le contrôle aux modules eltableau ou appelproc.

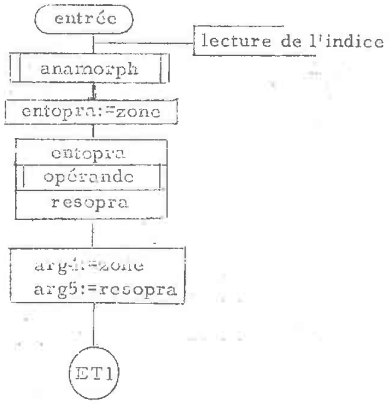
8. 6. 1. Identificateur indicé (module eltableau)

<identificateur> [<indice>] (cf. II. 2. 7. 5-a)

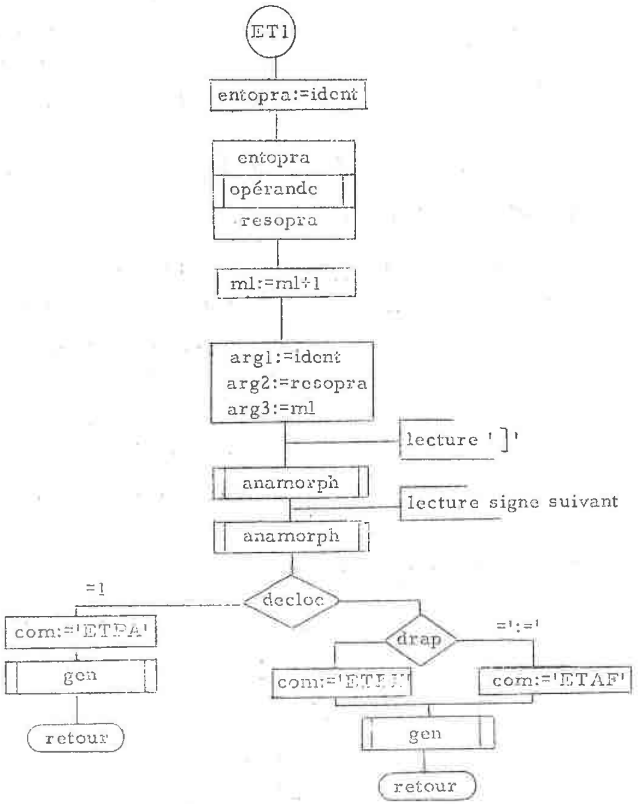
A l'appel du module eltableau l'identificateur de la rangée de repères se trouve dans ident. Les appels de macroinstruction générés sont :

ETAF	composante d'une rangée à gauche d'un signe d'affectation
ETPA	composante dans une déclaration locale. La séquence générée est identique à ETAF.
ETEX	composante dans une expression.

<indice> (cf. II. 2. 7. 5-a)



<identificateur> (cf. II. 2. 7. 5-b)



L'adresse ml, où se trouve la valeur ou l'adresse de la composante, est rangée sur la pile opra. Cette opération est effectuée dans le module expression.

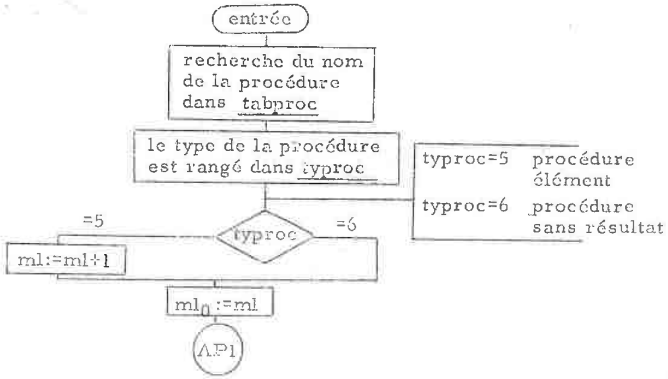
C. 6. 2. Appel de procédure (module appelproc)

Un appel de procédure est détecté par la parenthèse ouvrante qui suit un identificateur. Le nom de la procédure est rangé dans ident et le contrôle est donné à appelproc. Les appels de macroinstructions générées sont :

PELT	passage d'un paramètre élément	
PREP	idem	repère
PRAN	idem	rangée de repères
APRC	appel effectif de la procédure	

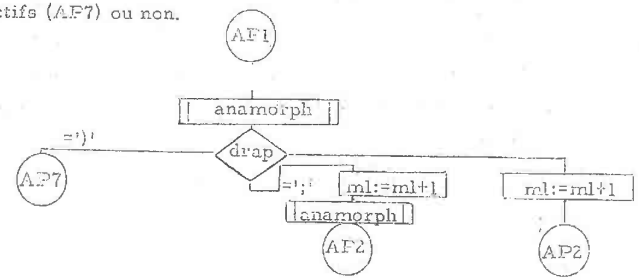
<identificateur> (cf. II. 2. 7. 5-d)

Dans le cas d'une procédure élément, le compteur ml est incrémenté de 1 pour réserver un mot pour le résultat.



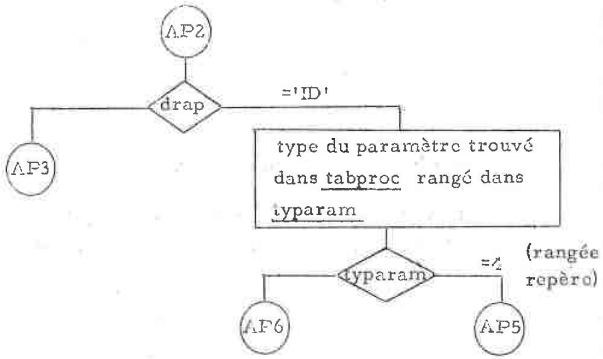
<liste de paramètres effectifs> (cf. II. 2. 7. 5-e)

Cette partie de programme vérifie si on est en fin des paramètres effectifs (AP7) ou non.

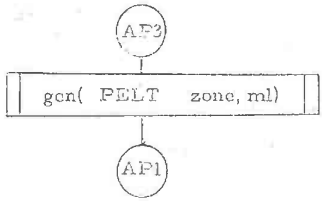


<paramètre effectif> (cf. II. 2. 7. 5-f)

Un paramètre effectif peut être un élément, une rangée de repères, un repère, une procédure ou une rangée de procédures. Les trois derniers cas sont traités de la même façon. On recherche le type du paramètre dans tabproc.

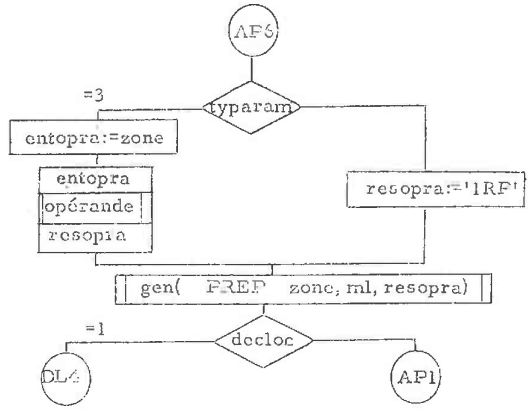


<élément> (cf. II. 2. 7. 5-d)

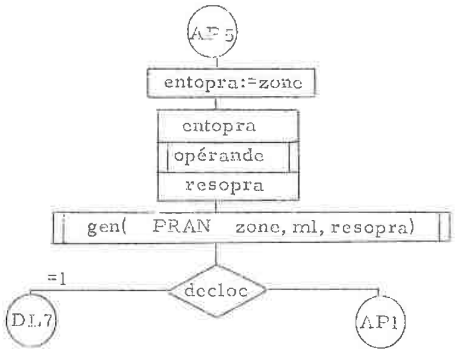


<identificateur> (cf. II. 2. 7. 5-f)

Paramètre effectif repère, procédure, rangée de procédures :
Un test supplémentaire différencie les repères des procédures et des rangées de procédures. Ceci pour permettre en Face$_0$ l'utilisation d'identificateurs de procédures et de rangées de procédures commençant avec une lettre 'F'.

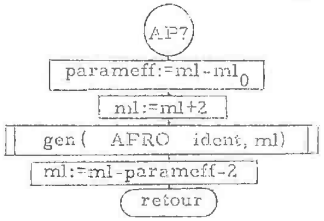


Paramètre effectif rangée de repères :



<appel de procédure> (cf. II. 2. 7. 5-d)

Après la mise en place des paramètres, il faut incrémenter le compteur ml de 2. Ces deux emplacements servent à la sauvegarde de l'ancien index et de l'adresse retour. La différence ml-ml$_0$ donne le nombre de paramètres effectifs dans l'appel.



CHAPITRE V

LES PROCEDURES STANDARD

Pour compiler les programmes source, le compilateur Face, écrit en $Face_0$, utilise plusieurs procédures standard :

1. préan — le préanalyseur (cf. II. 2. 3) [12]
2. analyseur — l'analyseur (cf. II. 2. 3) [12]
3. appl — le programme de commande des rangées de procédures ;
4. entrée — permet de faire des entrées en table ;
5. recherche — recherche les informations entrées en table.

1 - LE PROGRAMME DE COMMANDE DE $FACE_0$ (appl) (cf. III. 4. 3)

Il s'agit d'activer les procédures d'une rangée en fonction des numéros de règles transférés par l'analyseur syntaxique dans une rangée de repères. appl donne le contrôle à une procédure et en fin d'exécution de celle-ci il y a branchement à @RPL, adresse de retour dans appl.

C'est une procédure sans résultat dont les paramètres sont :

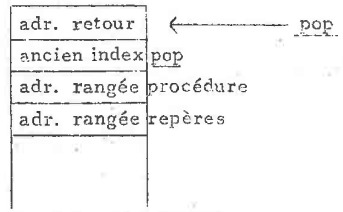
1. une rangée de repères ;
2. une rangée de procédures (= table d'adresse des procédures de la rangée).

Sa fonction regroupe trois actions :

1. charger la valeur de l'élément suivant la rangée de repères ;
2. chercher l'adresse de la procédure ;
3. branchement à cette adresse.

Description de la procédure appl : (1)

Soient r_1 et r_2 deux registres distincts des registres lop, i_1, i_2, pop

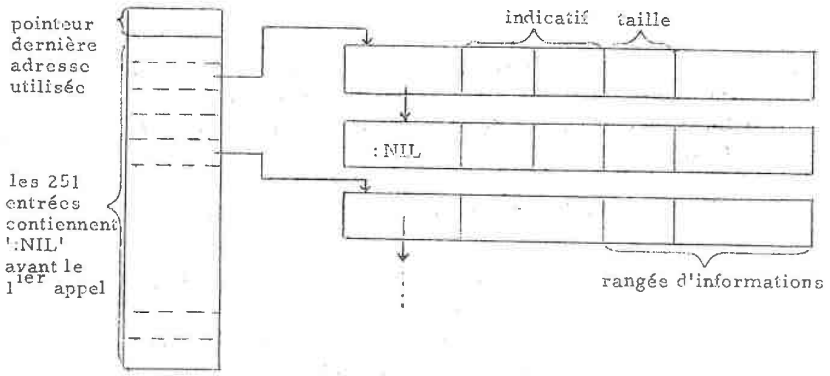



```
APPL    LW, r1      -3, pop     passage rangée rep.
        LW, r2      -2, pop     "      rangée proc.
        AI, r2      -1
        LW, r3      r1
        AW, r3      *r1         dernière adr. rangée rep.
        STW, r3     adernelt
@APPL   AI, r1      1
        CW, r1      adernelt    fin rangée repère
        BG          finappl
        LW, r3      *r1         contenu rangée rep.
        LW, r3      *r2, r3     adr. procédure
        B           0, r3
finappl LW, r3      0, pop
        B           0, r3
```

2 - LES PROCEDURES DE GESTION DES TABLES

La compilation d'un programme nécéssite l'emploi d'une ou plusieurs tables. Celles-ci renferment les informations des identificateurs déclarés dans le programme. En Face les informations sont des rangées de repères. Elles sont identifiées par une clé (ou indicatif) qui est une rangée de repères à deux éléments. Les tables sont gérées par la méthode de "partage de table" ("hashing" avec chaînage des synonymes). Leur structure est la suivante

(1) Pour être compatible avec son utilisation en Face, la table d'adresse des procédures est précédée d'un mot vide et l'adresse zéro de cette table contient l'adresse de la première procédure de la rangée, si elle existe.



L'algorithme d'accès aux données est le suivant :

1. Un OU exclusif entre les deux moitiés de l'indicatif (2) = (1) ⊕ (2)
2. Un décalage de 10 bits à gauche afin de garantir la plus grande répartition possible parmi les premiers caractères rencontrés dans un identificateur.
3. Division du résultat par le nombre d'entrées retenues (251). Le reste de la division donne le numéro de l'entrée.

2. 1. La procédure entrée

Cette procédure entre de nouvelles informations avec leur indicatif dans la table. Les paramètres de cette procédure sont :

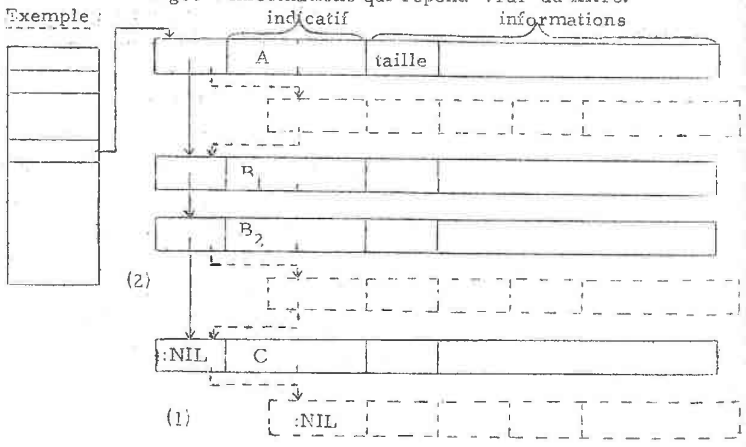
1. l'identificateur de la table
2. l'indicatif (rangée de repères)
3. les informations (rangée de repères)
4. le filtre (procédure résultat)

La procédure filtre a pour paramètre une rangée de repères. Le filtrage se fait sur le contenu d'une rangée d'informations précédemment introduite en table. Le résultat de cette procédure vaut 0 (faux) ou 1 (vrai).

L'endroit où sont introduit les nouvelles rangées d'informations dépend des entrées antérieures :

- si l'indicatif n'existe pas encore dans la table, l'entrée a lieu en fin des indicatifs existants.
- si l'indicatif existe mais le résultat de la procédure filtre vaut toujours zéro, son entrée a lieu derrière les indicatifs identiques.
- si l'indicatif existe et le résultat de la procédure filtre ne vaut pas toujours 0, l'entrée a lieu devant la première rangée d'informations qui répond 'vrai' au filtre.

Exemple :



Remarque : Les indicatifs B_1 et B_2 sont identiques.

Cas 1 : entrée d'un indicatif D qui a le même numéro d'entrée que A, B, C

Cas 2 : entrée d'un indicatif B ; le résultat de la procédure filtre vaut toujours 0

Cas 3 : entrée d'un indicatif B ; le résultat de la procédure filtre vaut 1 en testant B_1.

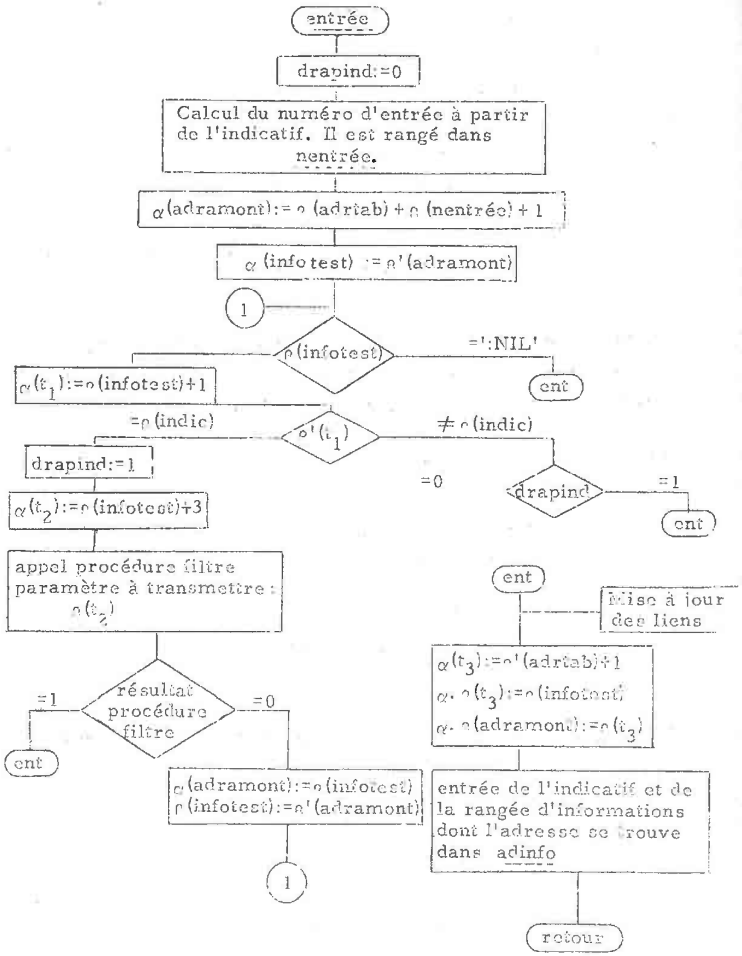
Il existe deux versions de cette procédure standard, la première étant une procédure sans résultat (entrée), la deuxième ayant pour résultat l'adresse de la rangée d'informations ajoutée (entrée 2).

Dans la suite de l'étude, nous emploierons les notations suivantes :

$\alpha(a)$ = mot d'adresse a
$\beta(a)$ = contenu de ce mot
$\rho'(a) = \rho_0\,\rho(a)$

ρ (adrtab) = adresse de la table
ρ (adinfo) = adresse de la rangée d'informations
ρ (indic) = indicatif

drapind permet de tester si on a déjà trouvé l'indicatif ou non.

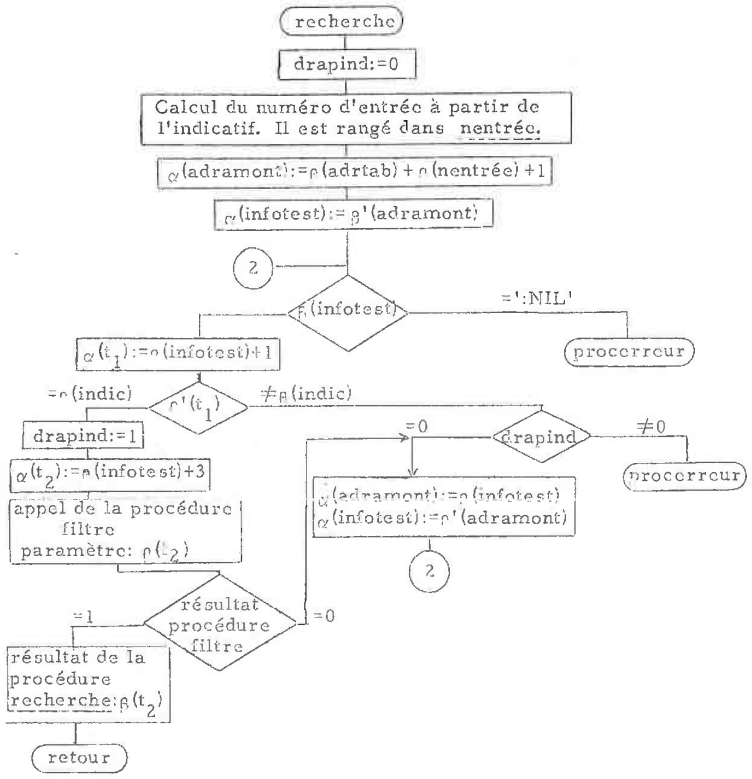


2. 2. La procédure recherche.

Cette procédure recherche la première rangée d'informations qui a le même indicatif que celui donné en paramètre et pour qui le résultat de la procédure filtre vaut 1. Le résultat est une rangée de repères qui est la rangée demandée. Les paramètres sont les suivants :

1. l'identificateur de la table
2. l'indicatif (rangée de repères)
3. le filtre (procédure résultat)
4. erreur (procédure sans paramètre et sans résultat)

Dans le cas où l'on ne trouve pas l'indicatif demandé ou si la procédure filtre répond toujours 'faux', il y a appel de la procédure erreur.



ANNEXE A

Le système de métaassemblage $Sface_0$

```
                SYSTEM    SIG7
                OPEN      A,B,C,D,E,A1,B1,C1
                OPEN      D1,E1,I1,I2,P0P,ML
P0P             EQU       5
I1              EQU       6
I2              EQU       7
IBP             EQU       8
*
IRP             EQU       1
IPAR            EQU       2
IML             EQU       3
ICTE            EQU       4
*
*               RANR      DIMENSION
RANR            CNAME
                PROC
LF              DATA      $+1
                DATA      AF(1)
                RES       AF(1)
                PEND
*
*               DECLARATION PROCEDURE
*
*               EQPA      ETIQ,RANG PARAM
EQPA            CNAME
                PROC
LF              SET       -AF(1)+AF(2)+2
                PEND
*
*               SAUV      NB. DE PARAM.
SAUV            CNAME
                PROC
                STW,IBP   -AF(1)+2,P0P
                PEND
*
*               RESL      NB. DE PARAM.
RESL            CNAME
                PROC
                XW,IBP    -AF(1)+2,P0P
                PEND
*
*               RET
RET             CNAME
                PROC
                LW,I1     0,P0P
                GEN.8,4,3,17 X'68',0,I1,0
                PEND
*
*               DECLARATION RANGEE DE PROCEDURES
*
*               PARE      NB. PARAM. ENTREE
PARE            CNAME
                PROC
                AI,P0P    -AF(1)
                PEND
*
*               PARS      NB. PARAM. SORTIE
PARS            CNAME
                PROC
```

```
                        AI,P8P      AF(1)
                        PEND
                *
                *       EXPRESSION FERMEE
                *
                *       INIT
INIT            CNAME
                PROC
                        REF         PREAN,ANALYSE
                        REF         APPL,ENTREE,ENTREE2,RECHE
                        LI,P8P      ML-1
                        PEND
                *
                *       ARRT
ARRT            CNAME
                PROC
                        CAL1,9      1
ERREX           CAL1,9      2
ML              RES         500
                        PEND
                *
                *       DECLARATIONS LOCALES
                *
                *       DLPR        ML
DLPR            CNAME
                PROC
                        LI,I1       AF(1)+1
                        AW,I1       P8P
                        STW,I1      AF(1),P8P
                        PEND
                *
                *       AFFECTATION
                *
                *       AFFC        OPERANDE,TYPE
AFFC            CNAME
                PROC
                        GOTO,AF(2) A,B,B,C
LF,A            STA,1BP     AF(1)
                        GOTO        E
LF,B            LW,I1       AF(1),P8P
                        STW,1BP     0,I1
                        GOTO        E
LF,C            STW,1BP     AF(1),P8P
E               PEND
                *
                *       FCBND       ETIQUETTE
FCBND           CNAME
                PROC
                        CI,1BP      0
                        BE          AF(1)
                        PEND
                *
                *       EXPRESSION
                *
                *       CHAR        REG,OPERANDE,TYPE
CHAR            CNAME
                PROC
                        GOTO,AF(2) A,B,C,D
A               LW,AF(1)    AF(2)
                        GOTO        E
B               LW,I1       AF(2),P8P
                        LW,AF(1)    0,I1
                        GOTO        E
C               LW,AF(1)    AF(2),P8P
                        GOTO        E
D               LW,AF(1)    =AF(2)
E               PEND
                *
                *       RANG        REG,ML
RANG            CNAME
                PROC
                        STW,AF(1) AF(2),P8P
                        PEND
                *
                *       COMP        REG,OPERANDE,TYPE
PG              CNAME       X'692'
PE              CNAME       X'681'
EG              CNAME       X'683'
DE              CNAME       X'693'
PE              CNAME       X'682'
PP              CNAME       X'691'
                PROC
                        GOTO,AF(3) A,B,C,D
A               CW,AF(1)    AF(2)
                        GOTO        E
B               LW,I1       AF(2),P8P
                        CW,AF(1)    0,I1
                        GOTO        E
C               CW,AF(1)    AF(2),P8P
                        GOTO        E
D               CW,AF(1)    =AF(2)
E               GEN,12,20   NAME(1),$+3
                        LI,1BP      0
                        GEN,8,24    X'68',$+2
                        LI,1BP      1
                        PEND
                *
                *       ADDI        REG,OPERANDE,TYPE
                *       SOUS        REG,OPERANDE,TYPE
ADDI            CNAME       X'30'
SOUS            CNAME       X'38'
                PROC
                        GOTO,AF(3) A,B,C,D
A               GEN,8,4,20  NAME(1),AF(1),AF(2)
                        GOTO        E
B               LW,I1       AF(2),P8P
                        GEN,1,7,4,20 1,NAME(1),AF(1),IND1
                        GOTO        E
C               GEN,2,6,3,17 NAME(1),AF(1),P8P,AF(2
                        GOTO        E
D               GEN,8,4,20  NAME(1),AF(1),=AF(2)
E               PEND
                *
                *       IDENTIFICATEUR INDICE
                *
                *       ETEX        IDENT,RANGEE,TYPE RGEE,
                *       ETAF        IND,INDICE,TYPE INDICE
                *       ETFA        1
```

```
178                     *
179                     *       ETEX
180     ETEX    CNAME
181                     PROC
182                     GOTO,AF(5) A1,B1,C1,D1
183     A1              LW,I1       AF(4)
184                     GOTO        E1
185     B1              LW,I1       AF(4),P0P
186                     LW,I1       0,I1
187                     GOTO        E1
188     C1              LW,I1       AF(4),P0P
189                     GOTO        E1
190     D1              LW,I1       *AF(4)
191     E1              GOTO,AF(2)  A,B
192     A               BLEZ        1ERREX
193                     CW,I1       *AF(1)
194                     BG          1ERREX
195                     LW,I1       *AF(1),I1
196                     STW,I1      AF(3),P0P
197                     GOTO        E
198     B               BLEZ        1ERREX
199                     LW,I2       AF(1),P0P
200                     CW,I1       *I2
201                     BG          1ERREX
202                     LW,I1       *I2,I1
203                     STW,I1      AF(3),P0P
204     E               PEND
205                     *
206                     *       ETAF,ETPA
207     ETAF    CNAME
208     ETPA    CNAME
209                     PROC
210                     GOTO,AF(5) A1,B1,C1,D1
211     A1              LW,I2       AF(4)
212                     GOTO        E1
213     B1              LW,I1       AF(4),P0P
214                     LW,I1       0,I1
215                     GOTO        E1
216     C1              LW,I1       AF(4),P0P
217                     GOTO        E1
218     D1              LW,I1       *AF(4)
219     E1              GOTO,AF(2)  A,B
220     A               BLEZ        1ERREX
221                     CW,I1       *AF(1)
222                     BG          1ERREX
223                     AW,I1       AF(1)
224                     STW,I1      AF(3),P0P
225                     GOTO        E
226     B               BLEZ        1ERREX
227                     LW,I2       AF(1),P0P
228                     CW,I1       *I2
229                     BG          1ERREX
230                     AW,I2       I1
231                     STW,I2      AF(3),P0P
232     E               PEND
233                     *
234                     *       APPEL DE PROCEDURE
235                     *
236                     *       PELT        ELEMENT,ML
```

```
237     PELT    CNAME
238                     PROC
239                     LW,I1       *AF(1)
240                     STW,I1      AF(2),P0P
241                     PEND
242                     *
243                     *       PREP        PARAM.,ML,TYPE
244     PREP    CNAME
245                     PROC
246                     GOTO,AF(3) A,B,C
247     A               LI,I1       AF(1)
248                     STW,I1      AF(2),P0P
249                     GOTO        E
250     B               LW,I1       AF(1),P0P
251                     STW,I1      AF(2),P0P
252                     GOTO        E
253     C               LI,I1       AF(1)
254                     AW,I1       P0P
255                     STW,I1      AF(2),P0P
256     E               PEND
257                     *
258                     *       PRAN        PARAM.,ML,TYPE
259     PRAN    CNAME
260                     PROC
261                     GOTO,AF(3) A,B
262     A               LW,I1       AF(1)
263                     STW,I1      AF(2),P0P
264                     GOTO        E
265     B               LW,I1       AF(1),P0P
266                     STW,I1      AF(2),P0P
267     E               PEND
268                     *
269                     *       APRA        NOM PROCEDURE,ML
270     APRA    CNAME
271                     PROC
272                     STW,P0P     AF(2)-1,P0P
273                     LI,I1       *+4
274                     STW,I1      AF(2),P0P
275                     A1,P0P      AF(2)
276                     SEN,8,24    X'68',AF(1)
277                     LW,P0P      -1,P0P
278                     PEND
279                     *
280                     *
281                     CLOSE       A,B,C,D,E,A1,B1,C1
282                     CLOSE       D1,E1,I1,I2,P0P,ML
283     E
284                     END
```

ANNEXE B

Règles de présentation d'un programme $Face_0$.

1. Les informations sur cartes, utiles pour le compilateur, vont de la colonne 1 à 72. Les colonnes 73 à 80 servent aux commentaires et numérotages éventuels.

2. Les mots soulignés dans la syntaxe de $Face_0$ sont représentés précédés et suivis d'un point.
   Exemple : rangée(20) rep a ; ⟹ . rangée. (20). rep. a ;

3. Nomenclature des opérateurs et signes utilisés en $Face_0$:

Définition $Face_0$	Caractères implémentés
+	+
-	-
<	<
≤	=<
=	=
≠	=|
≥	>=
>	>
:=	:=
:	:
,	,
;	;
(	(
)	)
[	(/
]	/)

REFERENCES

1 - Mme F. BELLEGARDE
Définition et implémentation de Face.
Thèse de spécialité, NANCY (1972)

2 - R. A. BROOKER, O. MORRIS
A general translation program for phrase structure languages
Journal of the ACM 9 (1962) p1-16.

3 - J. FELDMANN, F. GRIES
Translator writing systems
Communication ACM vol. 11 Num. 2 (Février 1968) p77-113.

4 - P. GILBERT, W. G. MC LELLAN
Compiler generation using formal specification of procedure-oriented and machine languages.
Proc. AFIPS (1967) Vol. 30 p447-455.

5 - S. GINSBURG
The mathematical theory of context-free languages.
MC Graw-Hill, New York (1966).

6 - R. KHALIL
Essai de formalisation de la description des compilateurs.
Application à Algol 60.
Thèse Docteur Ingénieur, NANCY (1970).

7 - H. H. METCALFE
A parametrized compiler based on mechanical linguistics.
Annual review in automatic programming (1964) vol. 4 p125-165.

8 - C. PAIR et A. QUERE
Définition et études des bilangages réguliers
Information and Control, 13 (1968), p565-593.

9 - C. PAIR
Sur des notions algébriques liées à l'analyse syntaxique
Exposé fait au Centre d'Automatique de l'Ecole des Mines, Fontainebleau (1969) soumis à la Revue Française d'Informatique et de Recherche Opérationnelle.

10 - C. PAIR, Mme F. BELLEGARDE, J. MAROLDT
FACE, langage pour l'écriture des compilateurs
Congrés AFCET, PARIS sept. 1970

11 - C. PAIR
Cours de l'Ecole d'Eté d'Informatique
Compilation, NEUCHATEL (1972)

12 - H. PISTRE
Thèse de Spécialité, NANCY - à paraître

13 - J.C. REYNOLDS
Cogent Manuel Argonne National Laboratory
Argonne, Illinois (Mars 1965) ANL - 70 22

14 - F. W. SCHNEIDER et G.D. JOHNSON
META 3 : A syntax directed compiler writing compiler to generate efficient code. Proc ACM 19ème Nat. Conf (1964) pD1. 6-1.

15 - D.V. SCHORSE
META II : A syntax oriented compiler writing language
Proc ACM 19 th Natl. Conf (1964). P D 1-3

16 - A. VAN WIJNGAARDEN (ed)
Report on the algorithmic language Algol 68 - Mathematisch Centrum Amsterdam (1969).

17 - W.A. WULF, D.B. RUSSEL, A.N. HABERMANN
BLISS : A language for Systems Programming
CACM, Déc. 1971, Vol. 14, p. 780.

NOM DE L'ETUDIANT : MAROLDT Jean Robert

Nature de la thèse : DOCTEUR INGENIEUR

Vu, Approuvé

et permis d'imprimer

NANCY, le 7 Novembre 1972

Le Président du Conseil de l'Université de NANCY I

J.R. HELLUY