### **Notations**

Les symboles en majuscule sont des symboles terminaux.

Les symboles en minuscule sont des symboles non-terminaux.

L'axiome de la grammaire est le symbole ``programme''.

A noter que cette grammaire est incomplète et doit être complétée par vos soins.

### **Remarques concernant le langage**

\* CPYRR est un langage procédural à structure de blocs.

\* La portée d'une déclaration est celle du sous-programme la contenant.

\* La surcharge est autorisée et doit être prise en compte par le compilateur.

\* Le masquage est autorisé. Un objet est masqué (non accessible) lorsqu'une déclaration du même identificateur apparait dans une région interne.

\* Toute déclaration introduit un nouveau type.

\* Dans une structure, les champs apparaissant à un même niveau doivent avoir des noms différents.

\* L'affectation n'est autorisée qu'entre valeurs de types simples. Les types de la source et de la destination doivent être identiques.

\* Les opérateurs ont leurs sens habituels :

* entier + entier  entier ( idem pour - \* et / )
* réel + réel  réel ( idem pour - \* et / )
* chaine + chaine  chaine ( concaténation )
* entier = entier  booléen ( idem pour <>, <, <=, >=, > )
* réel = réel  booléen ( idem pour <>, <, <=, >=, > )
* chaine = chaine  booléen ( idem pour <>, <, <=, >=, > )
* booléen et booléen  booléen (idem pour ou)

\* Les passages de paramètres se font par valeur. Les paramètres formels sont des variables locales initialisées, à l'entrée de la procédure appelante, avec les valeurs des paramètres effectifs.

\* Les fonctions ne peuvent retourner que l'un des 4 types simples (entier, réel, booléen, caractère)

\* On sort d'une procédure par l'instruction RETOURNE ou par la fin de la procédure.

\* On sort d'une fonction par l'instruction RETOURNE expression.

\* Entrées/sorties : il existe deux procédures prédéfinies :

* LIRE PARENTHESE\_OUVRANTE liste\_variables PARENTHESE\_FERMANTE
* qui lit des valeurs de variables de type simple
* ECRIRE PARENTHESE\_OUVRANTE format suite\_ecriture PARENTHESE\_FERMANTE
* suite\_ecriture ->
* suite\_ecriture -> VIRGULE variable suite\_ecriture
* qui a les mêmes fonctionnalités que le printf du C.
* La correspondance en nombre et en type des variables avec les % du format devra être réalisée.

## **Table lexicographique**

Cette table permet de stocker tous les lexèmes utiles et de leur associer de façon unique un code (valeur entière) appelé numéro lexicographique.

On utilisera la structure classique :

* Table de hash-code.
* Table d'enregistrements contenant pour chaque lexème :
  + La longueur du lexème.
  + Le lexème (ou un pointeur sur le lexème).
  + Un pointeur sur le lex&egrave;me suivant de même hash-code dans la table.

## **Table des déclarations**

La table des déclarations, construite lors de la compilation, sert à l'association de noms lors de ce processus, mais aussi à l'interprétation du programme lors de l'exécution de celui-ci. Elle sert à stocker toutes les déclarations de types, de variables, de procédures et de fonctions. Il est nécessaire de différencier, dans la table, les différentes catégories de déclaration.

La table des déclarations sera décomposée en une table primaire et une zone de débordement, la table primaire étant indicée par les numéros lexicographiques des identificateurs.

Chaque enregistrement de la table contient, à la fin de la compilation, cinq champs dont la signification varie, pour certains, suivant la valeur de l'un d'entre eux : le champ NATURE.

Ces champs sont les suivants :

* Champ NATURE. Ce champ peut prendre l'une des 4 valeurs :
  + NATURE=1 : l'enregistrement corrrespond à une déclaration de type structure.
  + NATURE=2 : l'enregistrement corrrespond à une déclaration de type tableau.
  + NATURE=3 : l'enregistrement corrrespond à une déclaration de variable.
  + NATURE=4 : l'enregistrement corrrespond à une déclaration de procédure.
  + NATURE=5 : l'enregistrement corrrespond à une déclaration de fonction.
* Chainage sur les autres déclarations de même nom, dans la zone de débordement.
* Numéro de région contenant la déclaration.
* Le quatrième champ a la signification suivante :
  + si NATURE=1 ou NATURE=2 : index dans la table contenant la description du type.
  + si NATURE=3 : index, dans la table des déclarations, de l'enregistrement associé à la déclaration du type. Si cette valeur vaut 1, 2, 3, ou 4, il s'agit d'un type de base (entier, réel, booléen ou caractère).
  + si NATURE=4 ou NATURE=5 : index dans une table contenant la description de l'ent\^ete de la procédure ou de la fonction.
* Le cinquième champ a la signification suivante :
  + si NATURE=1 ou NATURE=2 : taille à l'exécution d'une valeur de ce type (en tenant compte qu'il s'agit d'une machine C, cf ci-dessous).
  + si NATURE=3 : déplacement à l'exécution, de l'emplacement associé à la variable dans la zone de données corrrespondante.
  + si NATURE=4 ou NATURE=5 : numéro de la région associée à la procédure ou à la fonction.

## **Table de représentation des types et des entêtes de sous-programmes**

Cette table d'entiers contient la description des types, des procédures et des fonctions déclaré(e)s dans le programme.

Concernant les types, il s'agit obligatoirement de types structures ou tableaux. La description est définie ci-dessous.

* Pour les structures :
  + Nombre de champs de la structure.
  + Pour chaque champ :
    - Son type (index, dans la table des déclarations, de l'enregistrement associé à la déclaration de ce type). Si cette valeur vaut 1, 2, 3 ou 4, il s'agit d'un type de base.
    - Son numéro lexicographique.
    - Le déplacement, à l'exécution, pour atteindre l'emplacement du champ à l'intérieur de la structure.
* Pour les tableaux :
  + Le type des éléments (index, dans la table des déclarations, de l'enregistrement associé à la déclaration de ce type).
  + Le nombre de dimensions du tableau.
  + Pour chaque dimension, la borne inférieure et la borne supérieure.

Concernant les fonctions et procédures, La description est la suivante :

* La valeur 1, 2, 3, 4 pour indiquer le type du résultat dans le cas d'une fonction ; ce champ n'existe pas pour une procédure.
* Le nombre de paramètres.
* Pour chaque paramètre :
  + Son numéro lexicographique.
  + La valeur 1, 2, 3 ou 4 précisant son type.

## **Table des régions**

Un programme CPYRR comporte un certain nombre de régions : le programme principal, les procédures et les fonctions. Le numéro de région sert à indexer une table globale appelée *table des régions*.

Chaque enregistrement de cette table contient trois champs fournissant les informations suivantes sur chacune des régions :

* La taille de la zone de données associée dans la pile à l'exécution (cette information sert notamment pour la mise à jour de la base courante).
* Le niveau d'imbrication statique du corps de cette région. Le corps du programme principal possède un niveau d'imbrication statique égal à 0.
* Le pointeur vers l'arbre abstrait des instructions de cette région.

## **Texte intermédiaire - Interprétation**

La structure de données utilisée pour implanter le texte intermédiaire sera un arbre abstrait représenté par la structure fils/frere comme vu en cours/td.

Vous devez prévoir une procédure pour afficher un texte intermédiaire sous une forme lisible simple.

ATTENTION !!! : Après une compilation d'un programme CPYRR, vous devez prévoir de sauvegarder en fichier(s) toutes les données nécessaires en vue d'une interprétation utltérieure de ce programme (cette interprétation pouvant avoir lieu à un instant différent de la compilation).

*Remarque* : lorsqu'un texte intermédiaire peut être totalement généré, c'est une version syntaxiquement correcte, vis-à-vis d'une grammaire que vous pouvez définir, d'un programme CPYRR syntaxiquement et sémantiquement correct. On peut donc facilement écrire, lorsqu'on a la grammaire du texte intermédiaire, un analyseur syntaxique, avec les outils LEX et YACC, pour parcourir ce texte et le recharger en mémoire.

Le principe général de l'interprétation est le suivant : l'interpréteur doit parcourir le texte intermédiaire et l'interpréter au fur et à mesure, exécutant ainsi le programme CPYRR. La gestion de la pile doit être faite de façon explicite.

ATTENTION !!! : En aucun cas vous n'utiliserez la récursivité du langage d'implantation de l'interpréteur pour gérer la pile de façon implicite. Vous devrez gérer explicitement le chaînage dynamique et le chaînage statique. Le non respect de cette contrainte conduire à la nullité de cette partie du projet.

Pour gérer la pile à l'exécution, on réservera un grand tableau que l'on gèrera dynamiquement comme s'il s'agissait d'une pile. Les valeurs que l'on devra manipuler seront de quatre types : entier, réel, booléen, caractères. La déclaration, en C, du tableau, pourra être de la forme :

union{  
 int entier ;  
 float reel ;  
 char boleen ;  
 char caractere ;  
 } cellule ;  
   
 cellule pile[500] ;

La cellule de mémoire est ainsi un élément du tableau. On a donc alors égalité entre les tailles d'une valeur de type entier, réel, booléen ou caractères, il n'y a ainsi aucun problème d'alignement. Les chaînages dans la pile sont représentés par des valeurs entières servant à indexer le tableau.