Université Jean Monnet
de Saint-Etienne

Département d'Informatique

Unité d'Enseignement
Compilation

Réalisation d'un compilateur et d'une machine virtuelle pour le langage CPYRR

François Jacquenet
Faculté des Sciences et Techniques
Laboratoire Hubert Curien
18 rue Benoît Lauras
42000 Saint-Etienne
e-mail : Francois.Jacquenet@univ-st-etienne.fr

Table des matières

1 Grammaire du langage CPYRR

```
programme             -> PROG corps

corps                 -> liste_declarations liste_instructions
corps                 -> liste_instructions

liste_declarations    -> declaration POINT_VIRGULE
liste_declarations    -> liste_declarations declaration POINT_VIRGULE

liste_instructions    -> DEBUT suite_liste_inst FIN

suite_liste_inst      -> instruction POINT_VIRGULE
suite_liste_inst      -> suite_liste_inst instruction

declaration           -> declaration_type
declaration           -> declaration_variable
declaration           -> declaration_procedure
declaration           -> declaration_fonction

declaration_type      -> TYPE IDF DEUX_POINTS suite_declaration_type

suite_declaration_type -> STRUCT liste_champs FSTRUCT
suite_declaration_type -> TABLEAU dimension DE nom_type

dimension             -> CROCHET_OUVRANT liste_dimensions CROCHET_FERMANT

liste_dimensions      -> une_dimension
liste_dimensions      -> liste_dimensions VIRGULE une_dimension

une_dimension         -> expression POINT POINT expression

liste_champs          -> un_champ
liste_champs          -> liste_champs POINT_VIRGULE un_champ

un_champ              -> IDF DEUX_POINTS nom_type

nom_type              -> type_simple
nom_type              -> IDF

type_simple           -> ENTIER
type_simple           -> REEL
type_simple           -> BOOLEEN
type_simple           -> CARACTERE
type_simple           -> CHAINE CROCHET_OUVRANT CSTE_ENTIERE CROCHET_FERMANT

declaration_variable  -> VARIABLE IDF DEUX_POINTS nom_type

declaration_procedure -> PROCEDURE IDF liste_parametres corps

declaration_fonction  -> FONCTION IDF liste_parametres RETOURNE type_simple corps
```

```
liste_parametres      ->
liste_parametres      -> PARENTHESE_OUVRANTE liste_param PARENTHESE_FERMANTE

liste_param           -> un_param
liste_param           -> liste_param POINT_VIRGULE un_param

un_param              -> IDF DEUX_POINTS type_simple

instruction           -> affectation
instruction           -> condition
instruction           -> tant_que
instruction           -> appel
instruction           -> VIDE
instruction           -> RETOURNE resultat_retourne

resultat_retourne     ->
resultat_retourne     -> expression

appel                 -> IDF liste_arguments

liste_arguments       ->
liste_arguments       ->  PARENTHESE_OUVRANTE liste_args PARENTHESE_FERMANTE

liste_args            -> un_arg
liste_args            -> liste_args VIRGULE un_arg

un_arg                -> expression

condition             -> SI expression
                         ALORS liste_instructions
                         SINON liste_instructions

tant_que              -> TANT_QUE expression FAIRE liste_instructions

affectation           -> variable OPAFF expression

variable              -> description des formes possibles des variables

expression            -> description des formes possibles des expressions
```

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.

2 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. Ainsi, dans une même région, on doit pouvoir déclarer plusieurs objets de même nom, mais de nature diffrente (par exemple un type t, une variable t, une procédure t, une fonction t).

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

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

L'affectation n'est autorisée qu'entre objets 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.

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.

3 Table lexicographique

Cette table permet de stocker tous les lexèmes utiles et de leur associer 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 lexème suivant de même hash-code.

4 Table des déclarations

La table des déclarations, construite lors de la compilation, sert à l'identification des noms (lors de la compilation) et à l'interprétation (lors de l'exécution du programme compilé par la machine virtuelle). 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 déclarations.
Comme nous l'avons vu en cours, 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 (la table primaire a donc la même taille que la table lexicographique).
Chaque enregistrement de la table contient 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 correspond à une déclaration de type structure.
  - NATURE=2 : l'enregistrement correspond à une déclaration de type tableau.
  - NATURE=3 : l'enregistrement correspond à une déclaration de variable.
  - NATURE=4 : l'enregistrement correspond à une déclaration de paramètre.
  - NATURE=5 : l'enregistrement correspond à une déclaration de procédure.
  - NATURE=6 : l'enregistrement correspond 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 (déclaration d'un type structure ou tableau) : index dans la table contenant la description du type (table de représentation des types et des entêtes de sous-programmes).
  - si NATURE=3 ou NATURE=4 (déclaration d'une variable ou d'un paramètre) : index, dans la table des déclarations, de l'enregistrement associé à la déclaration du type de la variable dont on mémorise la déclaration. Si cette valeur vaut 0, 1, 2 ou 3, il s'agit d'un type de base (entier, réel, booléen ou caractère)
  - si NATURE=5 ou NATURE=6 (déclaration d'une procédure ou d'une fonction) : index dans la table contenant la description de l'entête de la procédure ou de la fonction (table de représentation des types et des entêtes de sous-programmes).
- Le cinquième champ a la signification suivante (c'est le champ appelé "exec" et laissé vide en cours/td pour l'instant) :
  - 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)
  - si NATURE=3 ou NATURE=4 : déplacement à l'exécution, de l'emplacement associé à la variable ou au paramètre dans la zone de données corrrespondante.
  - si NATURE=5 ou NATURE=6 : numéro de la région associée à la procédure ou à la fonction.

Il ne faut pas oublier que les paramètres de sous-programmes sont considérés comme des variables locales, ils doivent donc possèder des enregistrements dans la table des déclarations.

5 Table de représentation des types et des entêtes de sous-programmes

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

Concernant les types, il s'agit obligatoirement de structures ou de tableaux. La description est la suivante :

- **Pour les structures :**
  + Nombre de champs de la structure.
  + Pour chaque champ :
    * L'index, dans la table des déclarations, de l'enregistrement associé à la déclaration de son type. Si cette valeur vaut 0, 1, 2 ou 3, il s'agit d'un type de base.
    * Le numéro lexicographique du nom du champ.
    * Le déplacement, à l'exécution de l'emplacement du champ à l'intérieur de la structure (appel champ exec et laiss vide en cours/td pour l'instant).
- **Pour les tableaux :**
  + Le type des éléments.
  + 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 0, 1, 2 ou 3 pour indiquer le type du résultat de la fonction ; ce champ n'existe pas pour une procédure.
- Le nombre de paramètres.
- Pour chaque paramètre :
  * Son numéro lexicographique.
  * La valeur 0, 1, 2 ou 3 précisant son type (le type d'un paramètre ne peut être qu'un type de base).

6 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 rgions :

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

7 Arbre absstrait - Interprétation

L'arbre abstrait sera codé par la structure d'arbre binaire fils/frère comme vu en cours/td. Vous devrez prévoir une procédure pour afficher un texte intermédiaire sous une forme lisible simple.

Après une compilation d'un programme CPYRR, vous devez prévoir de sauvegarder en fichier(s) toutes les données nécessaires en vue d'une interprétation ultérieure 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'exécution d'un programme CPYRR est le suivant : la machine virtuelle doit parcourir l'arbre abstrait et l'interpréter au fur et à mesure, exécutant ainsi le programme CPYRR. La gestion de la pile (cf cours sur la pile à l'exécution) 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. La récursivité devra toutefois bien sûr être utilisée pour le parcours de l'arbre abstrait.

Pour gérer la pile à l'exécution, on se contentera de déclarer un grand tableau que l'on gèrera 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 ;
} u ;

u pile[5000] ;
```

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 chainages dans la pile sont représentés par des valeurs entières servant à indexer le tableau.