Licence 3 Informatique

Rapport en Compilation

Projet Compilation

GUILBAULT Maxime, BARRIERE Antoine, FLEURY Thomas, MAYOLINI Maxime

10 avril 2017

Table des matières

Chapitre 1

Introduction

Nous sommes un groupe de 4 étudiants en Licence 3 d'Informatique de l'Université de Bordeaux composé de MAYOLINI Maxime, FLEURY Thomas, BARRIERE Antoine et GUILBAULT Maxime. Ce document correspond au compte-rendu du projet de Compilation que nous avons dû réaliser dans le courant de notre sixième semestre. Le projet consiste à analyser le langage Pseudo-Pascal, à l'interpréter, le traduire en C3A et à écrire un interprète de C3A.

Pour cela, nous utilisons le Flex et le Bison comme langage afin de pouvoir analyser la grammaire et ainsi traduire afin de pouvoir faire un compilateur de Pseudo-Pascal en C3A.

Chapitre 2

Outils utilisés

Pour notre projet, nous avons utilisé différents outils tels que :

- **Atom :** l'éditeur principal pour coder. Il permet, avec les plugins configurés, d'avoir un code lisible et l'auto-complétion. De plus, cela permet de gagner en temps.
- **Github :** utilisé afin de versionner le code et de pouvoir développer à plusieurs.

Chapitre 3

Architecture du code

3.1 Description des structures

Nous utilisons diverses structures dans notre code :

- **Type :** il est composé :
  - d'un entier "dim" représentant la dimension d'un tableau dans le cas où le type est un array.
  - d'un entier "def" définissant le type où l'entier 1 représente un int, 2 un boolean, et 3 un array
- **Node :** il est composé d'une chaîne de caractères "value" représentant sa valeur, d'un type "m_type" représentant son type, de deux noeuds "rightChild" et "leftChild" représentant respectivement ses fils gauche et droit.

3.2 Description générale des fichiers

Le code est implémenté sur deux fichiers :

- **ppascal.y** : vérifie que la syntaxe du langage est correcte ;
- **ppascal.l** : vérifie que le lexique du langage (c'est-à-dire les chaînes de caractères utilisées) est correct.

Par ailleurs, il peut appeler différentes fonctions implémentées sur d'autres fichiers C :

- **PPutils.c** : permet de générer des Arbres de Syntaxe Abstraite (AST) ;
- **environ.c** : permet de générer des environnements de variables ;
- **bilquad.c** : permet de générer le code en C3A ;
- **util.c** : contient les structures et les fonctions que nous ont donné les enseignants pour la réalisation de ce projet.

3.3 Description détaillée des fichiers

Nous avons mis les fonctions que nous avons créées dans notre fichier PPutils. Comme les autres fichiers nous ont été donnés par les enseignants, nous n'en voyons pas l'utilité de les décrire à nouveau.

3.3.1 PPutils.c

Création de noeud

- **Nom de la fonction :** create_Node() ;
- **Paramètres :** value (char *), mtype (Type), rightChild (Node), leftChild (Node) ;
- **Principe :** permet de créer un noeud ;
- **Fonctionnement :** on déclare un noeud, auquel on attribue comme valeur "value", comme type "mtype", comme fils gauche "leftChild", comme fils droit "rightChild", puis on le retourne.

Fusion de noeud

- **Nom de la fonction :** fusionNode() ;
- **Paramètres :** filename (char *), type (Type), leftChild (Node), rightChild (Node) ;
- **Principe :** elle permet la fusion de deux noeuds entre eux ;
- **Fonctionnement :** elle retourne tout simplement la fusion de deux noeuds dans un autre noeud. Nous avons un nouveau noeud composé de 2 fils qui sont les noeuds "leftChild" et "rightChild".

Création de type

- **Nom de la fonction :** create_Type() ;
- **Paramètres :** dim (int), mtype (int) ;
- **Principe :** elle permet donner un type particulier à nos noeuds. Cela nous est utile pour l'analyse sémantique ;
- **Fonctionnement :** on déclare un type, auquel on attribue comme dimension "dim" et comme type "mtype" puis on le retourne .

Recherche d'une variable

- **Nom de la fonction :** searchVar() ;
- **Paramètres :** searched (Node), def (Node) ;
- **Principe :** elle permet de vérifier si une variable est bien définie. Elle est utilisée pour l'analyse sémantique ;
- **Fonctionnement :** tout d'abord on vérifie que le noeud que l'on donne en définition existe, puis nous comparons la valeur du noeud que l'on recherche

avec celui défini. La fonction est récursive, elle remonte l'arbre jusqu'au dernier noeud défini. Si le noeud recherché est défini, alors il renvoie la définition, Sinon la fonction renvoie NULL.

Affichage de l'arbre

- **Nom de la fonction :** printNode() ;
- **Paramètres :** n¡ud (Node) ;
- **Principe :** elle permet d'afficher l'arbre ;
- **Fonctionnement :** tout d'abord, on vérifie que le noeud que l'on donne n'est pas NULL. Puis il va afficher le noeud ainsi que ses fils qui sont le fils droit et fils gauche. Cette fonction est récursive, ce qui permet d'afficher l'arbre entier (utile pour répondre à la question 2).

Récupérer le type

- **Nom de la fonction :** getVarType() ;
- **Paramètres :** child (Node), def(Node) ;
- **Principe :** récupérer le type ;
- **Fonctionnement :** cherche le type grâce à la fonction searchvar().

Chapitre 4

Développement

4.1 Création d'un noeud

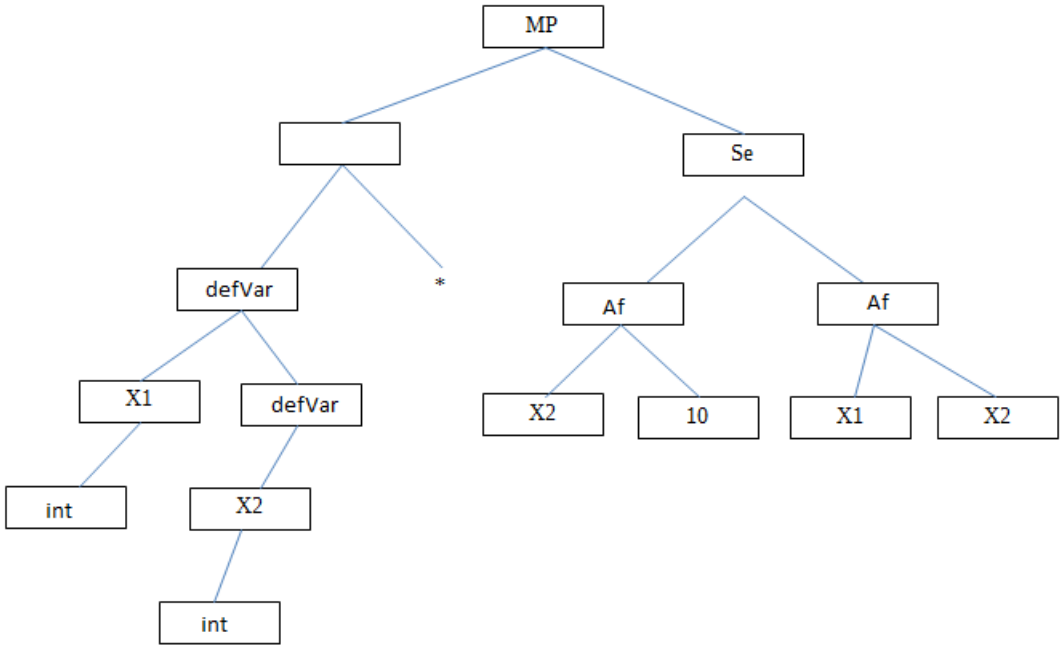
La création de noeud permet par la suite de créer l'arbre qui nous permettra de faire l'interpréteur. Un noeud est constitué d'une value, d'un type, d'un fils gauche et d'un fils droit, avec la possibilité de n'avoir que un seul fils ou aucun.

```
Node create_Node(char *value, Type mtype, Node leftChild, Node rightChild){
Node node = (Node)malloc(sizeof(struct noeud));
node - >value = strdup(value);
node - >m_type = mtype;
node - >rightChild = rightChild;
node - >leftChild = leftChild;
return node;
}
```

4.2 Création de l'AST

Pour le premier exemple on obtient l'arbre suivant

- var X1 : integer,
- var X2 : integer
- X2 := 10;
- X1 := X2



* représente l'emplacement des éventuelles définitions de fonctions et procédures, mais ici comme il y en a pas, le fils droit est vide

Chapitre 5

Problèmes rencontrés et solutions envisagées

5.1 Problèmes rencontrés

Dans ce projet nous avons été confrontés à plusieurs problèmes :
— Problème sur la compréhension de la grammaire (PPascal inconnu) ;
— Problème sur la construction de l'arbre ;
— Problème de segmentation fault.

5.2 Solutions envisagées

Pour l'interpréteur : à l'aide d'une fonction récursive, on regarde la valeur du noeud. S'il s'agit d'une variable, alors on la stocke dans un tableau de variables (si elle y est déjà, alors on met sa valeur à jour). S'il s'agit d'une opération, alors on l'exécute sur les deux noeuds fils. S'il s'agit d'une condition (if, while, etc), alors selon si la condition dans le noeud fils gauche est vraie ou non, on exécute ou non le noeud fils droit.