GIOVANNANGELI Loann
RIOU Maxence
MOUHOUB Noureddine
CHEMOUNE Aleaddine

INF601 A1

Rapport de projet terminal

Compilation

L3 Informatique
Université de BORDEAUX
2016/2017

Table des matières

Introduction

Ce projet consiste à mettre en oeuvre les connaissances acquises pendant le cours de compilation de la L3 Informatique de l'Université de Bordeaux.

Il consiste à analyser le langage Pseudo-Pascal (PP), à l'interpréter, le traduire en C3A et écrire un interprète de C3A.

Travail à réaliser :

Partie obligatoire :

1- Un analyseur syntaxique du langage PP.
2- Un analyseur sémantique du langage PP.
3- Un interpréteur du langage PP.
4- Un traducteur (ou compilateur) de PP vers C3A.
5- Un interpréteur du langage C3A.

Extensions(facultatives) :

6- Un traducteur (ou compilateur) de C3A vers Y86.
7- Un "ramasse-miettes" pour l'interprète de la question 3).

I - Partie réalisée

1) Analyseur syntaxique du langage PP

La grammaire du langage PP est définie syntaxiquement dans pppascal.y au format Bison.

Notre fichier pppascal.l (au format Flex) analyse le programme PP en entrée. Il fait appelle au Bison qui réalise l'analyse syntaxique. On obtient ainsi liste des variables globales. Une erreur est retournée si la syntaxe du programme est incorrecte.

2) Analyseur sémantique du langage PP

L'analyse sémantique permet d'analyser le sens du programme PP donné en entrée. On vérifie notamment que les types des différentes variables utilisées dans le programme sont corrects.

L'analyse sémantique peut retourner trois types de messages en sortie :

- erreur de typage dans le cas d'une erreur sémantique (c-à-d : affectation de deux variables qui n'ont pas le même type, une case du tableau qui n'existe pas…).
- programme bien typé dans le cas où aucune erreur de sémantique n'a été détecté.
- typage incomplet.

3) Interpréteur du langage PP

Présent dans interpPP.c. Cette fonctionnalité prend en entrée du code en PP et renvoie dans le terminal son "interprétation" : le contenu de son environnement (essentiellement composé des variables et de leurs valeurs).

4) Un traducteur (ou compilateur) de C3A

Notre compilateur génère du code C3A à partir d'un programme PP en entrée. Pour cela on lit le PP ligne par ligne et on génère des QUAD, c'est-à-dire des lignes de la forme :

Etiquette : Opérateur : Argument 1 : Argument 2 : Destination

Pour créer un QUAD on doit récupérer d'abord l'opérateur de la ligne en cours dans le PP. En fonction de cet opérateur, notre compilateur va s'occuper de remplir les valeurs de Argument 1, Argument 2 et Destination de manière récursive. L'ensemble de ces QUAD est appelé le BILQUAD. C'est ce BILQUAD qui est retourné à la fin de la traduction.

II - Partie non réalisée

Certaines parties n'ont pas été implémentées dans notre projet.

Les extensions facultatives :
- Un traducteur (ou compilateur) de C3A vers Y86.
- Un "ramasse-miettes" pour l'interprète de la question 3).

L'interpréteur PP, le compilateur C3A et l'interpréteur C3A ne gèrent pas les fonctions.

L'interpréteur C3A ne différencie pas les types pour les tableaux :
En effet : ne sachant pas comment faire apparaître un type dansla déclaration C3A d'un tableau, notre interpréteur ne sait pas récupérer le type lié aux tableaux et toutes les valeurs stockées et utilisés par ceux-ci sont donc des entiers.

III - Utilisation

En fonction de ce que vous souhaitez réaliser (analyser, interpréter ou compiler) il faut décommenter une partie du main qui se trouve dans le fichier pppascal.y

Il faut ensuite créer l'exécutable en tapant tout simplement dans le terminal la commande : make.

Pour tester le programme tapez :
```
./pppascal < inputFile
```

par exemple :
```
./pppascal < EXEMPLES/pex1.pp
```

IV - Difficultés rencontrées

Nous avions commencé le projet par réaliser l'analyse syntaxique et sémantique en se basant plus ou moins sur ce que nous avions fait dans le mini projet.
Nous avons cependant eu du mal à comprendre et implémenter le projet terminal à partir du mini projet. Or, nous avons suivi le TD7 de compilation sur la sémantique du PP et avons choisi de recommencer le projet à partir de l'environnement vu au cours de ce TD. Nous avons donc perdu du temps au début du projet et avons en plus dû prendre le temps de nous approprier le contenu du TD7.

Notre principale difficulté a été la gestion des tableaux. Du côté de compilateur nous avons mis du temps à faire des QUAD avec la bonne syntaxe. Un de nos problèmes était que le programme ne pouvait pas connaître à tout moment le nom d'un tableau précédemment déclaré.
Pour résoudre ce problème nous avons rajouté un tableau matching avec une fonction NameToId.

char *matching[TAILLEADR] sert à stocker le nom du tableau en fonction de son indice, ce dernier étant incrémenté de la même manière que dans le tableau ADR.
exemple :
ADR[x] donne la première case "i" du tableau dans le tas.
matching[x] donne le nom du tableau qui a pour première case dans le tas "i".

Nous utilisons la fonction NameToId(char *name) pour rechercher l'index du tableau de nom "name".

Avec cette méthode dès qu'un tableau est appelé par son nom de variable, on peut facilement accéder à son adresse dans le tas.

V - Organisation

Versioning :

Ce projet ayant été réalisé en groupe de 4, nous avons utilisé l'outil de versioning Git (Github) pour mettre en commun nos avancées.
(https://github.com/Didine24/Compilation)

Répartition :

Etant donné le petit nombre de fichiers à modifier, nous travaillions au maximum sur 2 ordinateurs à la fois, nous séparant en deux groupes de deux par poste.

Méthode :

Nous n'avons pas réellement mis en place de méthode de développement de projet. A posteriori, on peut dire que nous nous sommes rapprochés d'un développement par incrément en parallèle : nous avions deux groupes travaillant sur deux fonctionnalités différentes en même temps, et chaque groupe complétait (c-à-d développait et testait) sa fonctionnalité avant de passer à la suivante.

Gestion :

Aucune gestion ou suivi de projet n'a été mis en place pour ce projet.

Conclusion

Ce projet était l'occasion pour nous de mettre en œuvre les concepts élémentaires de compilation de langage de programmation moderne.

Pour cela, nous nous sommes familiarisées avec un outil d'analyse lexical (FLEX) et d'analyse syntaxique (YACC/BISON).

Le compilateur développé, qui suit les spécifications imposées, peut toutefois être amélioré (par exemple ajouter une gestion des fonctions ou améliorer la gestion des tableaux)