Rapport du projet : compilateur

Polytech Paris Saclay – APP4 Informatique
Cours de compilation

Année universitaire 2023 – 2024

Elèves :
KALALA KABAMBI David
MARTY Julien

Enseignant :
Lavergne Thomas

Table des matières

Introduction 3
I) Bilan du projet 3
a. Phase de tests 3
b. Répartition du travail 3
c. Difficultés rencontrées 4
Bilan global et conclusion 4

Introduction

L'objectif du cours est d'implémenter un compilateur pour le langage C. Le compilateur doit fournir un code exécutable en assembleur à partir du programme C récupéré en entrée. Le code assembleur généré sera exécuté sur un ancien modèle de processeur afin de rendre la génération du code plus aisée. Le but du projet est de comprendre comment est conçu un compilateur et quelles sont les grandes étapes d'analyse du code afin de pouvoir générer un code assembleur. Nous avons choisi d'écrire notre compilateur en Java.

I) Bilan du projet

Tous les points clé vu en cours ne sont pas opérationnels (expressions, variables, conditionnelles, boucles, fonctions, pointeurs, tableaux) et ont été testés. Il nous manque la portée des variables et les boucles imbriquées.
Cependant nous n'avons pas implémenté la fonction « print_num() ». Nous n'avons pas trouvé de solution pour terminer la fonction proprement sans afficher de 0 supplémentaire.
Le Java étant un langage orienté objet, les différentes étapes d'analyse du code sont implémentées par trois classes différentes, chacune représentant une étape de l'analyse. La partie génération du code est implémentée dans une quatrième classe. L'ordonnancement des étapes de l'analyse du code est effectué dans une classe dite de gestion nommée « Compile », c'est cette dernière qui est appelée dans le « main » pour lancer le programme. Nous avons utilisé des énumérations Java pour lister les types de nœuds, tokens et symboles. En plus de cela, les nœuds, opérateurs, symboles et tokens constituent chacun une classe à part entière. Il était ainsi facile de les modifier au gré de l'avancement de notre projet.

a. Phase de tests

Nous avons découpé nos phases de tests en reprenant les grandes parties du cours (expressions, variables, conditionnelles, boucles, fonctions, pointeurs, tableaux). Dans un premier temps, nous testons les cas basiques sensés fonctionner, dans un second temps les cas n'étant pas sensé fonctionner. Enfin ce que l'on appelle les cas limites, qui sont des cas plus complexes qui sont eux aussi sensé fonctionner. Nous avons listé tous nos cas dans un tableur en notant le résultat attendu, celui obtenu et si le test est validé ou non. Lorsqu'il n'était pas validé, nous mettons un commentaire expliquant le problème rencontré. Plus tard nous confrontons nos tests échoués afin de vérifier qu'il n'y ai pas eu d'erreur de code ou d'interprétation et sinon de localiser d'où provient l'erreur.

b. Répartition du travail

Julien ayant une bonne connaissance en Java, c'est lui qui s'est occupé de l'architecture du projet. Une fois celle-ci défini, nous avons chacun avancé sur une partie du code à la fin de chaque cours magistral. David c'est majoritairement occupé de la partie génération du code assembleur et

analyseur lexicale tandis que Julien s'est occupé de la partie analyseur syntaxique et sémantique. Nous avons tous deux réalisés des tests du code assembleur en nous répartissant les fonctions du compilateur à tester. En cas de difficulté majeure ou d'incompréhension, nous nous réunissions afin de résoudre notre problème.

c. Difficultés rencontrées

Les premières étapes du cours étaient assez faciles à aborder. Néanmoins, nous avons rencontré des problèmes avec la portée des variables, notamment en raison d'une erreur que nous avons commise en créant deux nœuds, Type_noeud.block et Type_noeud.bloc. En réalité, il aurait suffi de créer un seul nœud pour gérer cette notion.

De plus, nous avons eu des difficultés avec l'implémentation des tableaux. Il était nécessaire de créer un nœud d'indirection ayant un nœud d'addition comme enfant, auquel étaient liés les nœuds a et e. Au départ, nous avions directement attribué a et e comme enfants du nœud d'indirection, ce qui était une erreur.

Notre plus grande difficulté a été l'implémentation de la boucle for. Cette tâche nous a demandé beaucoup de temps. Comprendre en détail la structure de la boucle while était la première étape. Ensuite, nous avons examiné les différences entre les boucles while et for, notamment le nombre de paramètres dans la boucle et l'utilisation du nœud target. Bien que notre programme compilait sans erreur, notre code assembleur généré n'était pas correct. Nous avons eu du mal à identifier la source de l'erreur, qui provenait de l'ordre d'ajout des nœuds enfants aux nœuds parents, ce qui résultait en un code assembleur incorrect.

De plus, nous avons constaté que les boucles imbriquées ne fonctionnaient pas correctement en plus de la portée des variables, ce qui a ajouté une couche de complexité à notre projet

Bilan global et conclusion

Ce projet fut très enrichissant, il nous aura permis de mieux comprendre le fonctionnement d'un compilateur et les contraintes qui entourent ces derniers. Nous avons à présent une idée assez précise de la manière avec laquelle sont créés les langages de programmation et du cadre nécessaire pour les créer.
Grâce à la segmentation du cours, nous connaissons les quatre piliers sur lesquels un compilateur repose (analyse lexicale, analyse syntaxique, analyse sémantique et enfin génération du code).

D'autre part, nous avons aussi une meilleure appréhension des messages d'erreur et de la logique derrière leur génération. Ceci qui nous permet d'analyser plus rapidement d'où provient les erreurs détectées par le compilateur lorsque nous programmons.