Rapport de projet

COMP - M1 Info GL2

Mullier Antoine

Sadok Romain

Sommaire

Sommaire **1**

Introduction **2**

1 - Méthodologie de travail **2**

1.1 - Technique de développement 2

1.2 - Organisation du code 2

2 - Travail réalisé **3**

2.1 - Ce qui a été fait 3

2.2 - Programme de test 4

3 - Tests **5**

Conclusion **5**

Introduction

Ce rapport présente la réflexion, la conception et la réalisation du projet “Compilateur VSL+” dans le cadre de l’enseignement COMP.

Le but du TP était de réaliser un compilateur du langage VSL+. Notre travail était focalisé sur la compilation du langage VSL+ dans un code intermédiaire trois adresses comme vu en cours. La production de code assembleur MIPS était déjà implémentée et l’exécution des différents programmes est gérée par NACHOS.

1 - Méthodologie de travail

1.1 - Technique de développement

Afin de valider toutes les étapes de notre compilateur, nous nous sommes inspiré de l’esprit du développement agile. Pour chacune des fonctionnalités implémentées, nous avons mis en oeuvre des tests pour identifier et corriger les éventuels problèmes et vérifier que chaque étape ne modifie pas les fonctionnalités déjà implémentées.

En pratique, la première chose que nous avons faite était de reconnaître intégralement l’AST. Cela nous a permis de compléter petit à petit les traitements attendus par les nouvelles fonctionnalités de façon structurée. Avant chaque implémentation de fonctionnalités une recherche dans le cours et dans les TD était nécessaire pour comprendre ce qui était attendu.

Afin de pouvoir travailler à plusieurs l’utilisation d’un gestionnaire de versions (Git dans notre cas) nous a été très utile. Toujours dans le but de développer en “méthode agile” nous nous sommes basé sur les fonctionnalités testées et validées pour développer de nouvelles fonctionnalités en ayant la possibilités de revenir aux versions validées.

1.2 - Organisation du code

Pour développer notre compilateur VSL+ nous avons principalement édité deux fichiers :

- **VSLTreeParser.g :**
  Grammaire attribuée permettant d’arpenter l’AST pour produire le code trois adresses.
  La production de *Code3a* fait appel aux méthodes implémentées dans le fichier Code3aGenerator.java

- **Code3aGenerator.java :**
  Classe statique qui permet d’agréger les différentes méthodes de génération de code trois adresses. Cela nous permet d’alléger le contenu de la grammaire en dissociant les traitements de génération de code et la reconnaissance de l’AST.

- **TypeCheck.java :**
  Le fichier permet d'effectuer les vérifications de type liées au langage VSL+.

- **Makefile :**
  Contient les commandes de compilation pour automatiser la compilation des tests.

- **run_test_nachos.sh :**
  Script shell créé pour automatiser les tests d'exécution à l'aide de NACHOS.

2 - Travail réalisé

2.1 - Ce qui a été fait

Voici la liste des fonctionnalités implémentées :

- **Expressions simples :**
  Opérations binaires, opérations unaires, gestion de la priorité des opérateurs.
- **L'instruction d'affectation**
- **La gestion des blocs**
  Gestion des variables locales à un bloc.
- **Déclaration de variables**
- **Les expressions avec variables**
- **Les instructions de contrôle :**
  Gestion du IF, de la boucle WHILE et des séquences d'instructions.
- **La définition et l'appel de fonctions :**
  La définition d'une fonction, le prototypage d'une fonction, l'appel d'une fonction, l'appel d'une procédure.

  Remarque : L'appel de fonction avec un paramètre de type tableaux n'a pas été géré.
- **Les fonctions de la bibliothèque :**
  Affichage et lecture à l'aide des fonctions `PRINT`, `READ`.
- **La gestion des tableaux :**
  Affectation dans un tableau, lecture et affichage d'une case d'un tableau.

2.2 - Programme de test

Voici notre programme de test reprenant toutes les fonctionnalités de notre compilateur VSL :

```
PROTO INT fibo(n)
FUNC VOID main()
{
    INT n,i,t[11]
    i:=0
    PRINT "Veuillez saisir un entier n<11 :"
    READ n
    WHILE n-i
    DO
    {
       t[i] := fibo(i)
       i:= i+1
    }
    DONE

    i:=0
    WHILE n-i
    DO
    {
       PRINT "t[", i ,"]= ", t[i],"\n"
       i:= i+1
    }
    DONE
}
FUNC INT fibo(n)
{
    INT i
    i:=(n-1)
    IF n
    THEN
    {
       IF i
       THEN
       {
             RETURN fibo(n-1) + fibo(n-2)
       }
       ELSE
       {
             RETURN 1
       }
       FI
    }
    ELSE
    {
       RETURN 0
    }
    FI
}
```

3 - Tests

Nous avons effectué tous les tests disponibles sur le répertoire /share. Tous les tests **compilent et s'exécutent correctement** mis à part les paramètres de fonctions de type tableau comme nous l'avons énoncé précédemment.

Nous avons implémenté nous même quelques tests qui nous ont servis à débugger et à mieux comprendre le compilateur et les problèmes rencontrés au travers de ce TP.

Conclusion

La principale difficulté rencontrée a été la prise en main de tous les outils mis à notre disposition. En effet, comprendre quels sont les rôles des différentes composants, comment et où apporter nos modifications a été une tâche chronophage.

Faire le lien entre le cours et la pratique liée au TP nous a aussi pris du temps mais cela nous a permis à de nombreuses reprises de nous sortir de problèmes dans lesquels nous étions plongés depuis longtemps.

La validation de chacune de nos étapes nous appris de voir notre travail avancer sans devoir revenir en arrière pour revoir les fonctionnalités de base du compilateur. Le travail de reconnaissance de l'AST nous a permis d'avoir d'emblée une structure de grammaire claire sans risquer de compliquer la grammaire à chaque étape.