INTRODUCTION

Le but des travaux pratiques que nous avons effectués au cours des 6 séances de la Compilation était de développer un compilateur pour le langage VSL+ dont une description informelle nous est fournie au début avec les fichiers de test contenant le code VSL+. Une majeure partie de ce compilateur, i.e. les analyseurs lexicale et syntaxique, et la structure globale du code Java y compris le code commenté dans la plupart des fichiers - nous est aussi fournie. Alors notre objectif techniquement était d'écrire le code pour les classes Code3aGenerator, TypeCheck, Errors et VSLTreeParser en y ajoutant nos propres modifications à chaque étape.

Méthodologie de travail

- Plusieurs courtes itérations d'analyse-développement-
  en adaptant une méthodologie agile. Puisque nous n'étions pas dans le même groupe, nous avons utilisé 'Google Docs' pour partager le code, en privé avec les commentaires, ce qui nous a aussi facilité la gestion des révisions ainsi que la discussion sur des nouvelles modifications à la fin de chaque étape.
- Nous nous sommes réparti le travail principalement en terme des classes à coder, par. ex. si un augmentait l'arpenteur de l'arbre, il communiquait ses besoins des fonctions pour la génération du code 3-adresses à l'autre afin que l'autre pût développer TypeCheck et Code3aGenerator. A la fin de chaque cycle, nous nous sommes expliqué les choix du développement que nous avons fait pour permettre une vérification statique avant d'éxecuter notre compilateur sur les fichiers tests *.vsl du dossier level-N pour le N-ième séance du TP.
- Quant à l'organisation du code, nous avons bien exploité des fonctionnalités natives de l'IDE Eclipse. A part cela, nous avons dû créer une classe supplémentaire, viz. SymbolTableCode.java, pour surmonter la lacune d'ANTLR qui empêche le retour de plusieurs variables par une méthode Java(alors un symbole non-terminal dans le fichier grammaire).

Bilan de réalisation

— Les expressions simples
— L’instruction d’affectation
— La gestion des blocs
— La déclaration des variables
— Les expressions avec variables
— Les instructions de contrôle if, while et la séquence
— La définition et l’appel de fonctions (avec les prototypes)
— Les fonctions de la bibliothèque : PRINT et READ
— La gestion des tableaux (déclaration, expression, affectation et lecture)
Notre compilateur comprend tout le langage VSL+ à l'exception du passage des tableaux comme arguments pour les fonctions.

Rapport des tests

Nous avons inclus tout les fichiers produits par le machine NachOS à partir du code MIPS compilé par notre compilateur.
Tous les fichiers VSL ont produit le résultat attendu selon la logique du code sauf le cas des tableaux mentionné ci-dessus.

Conclusion

Pendant la progression du TP nous avons rencontré des difficultés variés. Par exemple dans la déclaration de variables on a investi du temps en trouvant la façon correcte d’insérer les identifiants dans la table de symboles. Aussi nous avons du créer un nouveau objet qui encapsulait la table de symboles et le code généré parce que JAVA ne permet pas de retourner deux variables dans un méthode. Pour l’exécution de nachos nous avons enlevé les guillemets de l'attribut data de l'objet Data3a et nous avons ajouté une fonction setter setText. Finalement nous avons reussi a generé le code vsl pour plusieurs cas de tests et nous les avons compilés et executé sur la machine MIPS.