PROJET GL

RENDU INTERMÉDIAIRE

Grenoble INP-Ensimag

Amine ANGAR
Dy EL ALEM
Mouad TARBOUI
Oussama KADDAMI
Youness LEHDILI

Ce document constitue un manuel utilisateur pour le rendu intermédiaire. Nous avons jugé qu'il était pertinent de le déposer afin de pouvoir expliquer le contenu du rendu et l'utilisation des tests implémentés.

CONTENU DU RENDU:

Le rendu englobe les fonctionnalités que stipule le cahier de charge. Nous avons essayé d'implémenter un compilateur fonctionnel pour la partie sans objet avec la même qualité et les mêmes options que celles du compilateur final. Pour ce faire, nous avons implémenté des tests tout au long de la phase d'implémentation pour détecter les défauts potentiels du compilateur.

TESTS:

Depuis le début de projet, nous avons essayé d'implémenter des tests unitaires pour assurer le fonctionnement correct de chaque partie codée. Nous avons défini des incréments et des sous incréments afin de pouvoir avancer de manière parallèle. À la fin de chaque incrément, nous testions le compilateur en rassemblant les différentes parties afin d'assurer une cohérence globale.

Partie A:

Pour la partie analyse lexicale et syntaxique
- $lex-test.sh$: $effectuanttest\_lexsurlestestsvalides$
- $synt-test.sh$: $effectuanttest\_syntsurlestestsvalides$
- $lex-test-inv.sh$: $effectuanttest\_lexsurlestestsinvalides$
- $synt-test-inv.sh$: $effectuanttest\_syntsurlestestsinvalides$

Pour lancer un test unitaire sur la partie lexicale, il suffit de lancer à partir de la racine du projet la commande $test\_lex$ (si test_lex est ajouté au PATH) sinon faire:
src/test/script/launchers/test_lex Pour lancer un test unitaire sur la partie syntaxique il suffit de lancer à partir de la racine du projet la commande
test_synt (si test_synt est ajouté au PATH), sinon faire :
src/test/script/launchers/test_synt
Pour lancer les scripts de tests automatiques faire depuis la racine du projet: src/test/script/

Partie B:

Pour la partie B, on a implémenté plusieurs tests pour tester les différentes fonctionnalités du langage sans objets, ces tests sont organisés dans les fichiers $valid/valid\_sans\_objet$ pour les tests valides et $invalid/invalid\_sans\_objet$ pour les tests invalides. On peut tester ces différents tests d'une manière unitaire par la commande test_context $src/test/deca/contex/valid/valid\_sans\_objet/nom\_du\_test$ ou lancer les script automatiques qui testent tous les fichiers $./src/test/script/context-test.sh$ pour les tests valides et $./src/test/script/context-test-inv.sh$ pour les tests invalides.
Rq: les tests sont à lancer depuis le répertoire $Projet\_GL$

Partie C:

Afin de tester la partie de génération du code de manière indépendante, nous avons implémenté différents tests dans le répertoire $src/test/java/fr.ensimag.deca.tree$ en plus du test $ManualTestInialGenCode.java$ fourni. Ensuite, après la fin de chaque incrément, nous implémentions des tests ".deca". Vous trouverez les tests associés à la partie génération de code dans le répertoire $src/test/deca/codegen/valid/$. Ce répertoire contient des sous-répertoires classifiant les différents tests. Vous pouvez lancer le script des tests depuis le répertoire $Projet\_GL$ à l'aide de la commande $./src/test/script/gencode-test.sh$. Ce script permet de créer les fichiers assembleurs associés aux fichiers sources ainsi que d'exécuter ces dernier avec une redirection de sortie vers des fichiers ".res". Après la génération de ces derniers, vous pouvez les supprimer en lançant la commande $./src/test/script/gencode-clean.sh$. Nous avons aussi généré des tests pour vérifier le bon fonctionnement des exceptions dictées dans la partie **Semantique**. Vous trouverez ces tests dans le répertoire $src/test/deca/codegen/invalid/$. Concernant les tests associés aux fonctions "read", ils sont présents dans le sous-répertoire $src/test/deca/codegen/invalid/read$, puisque leurs sorties dépendent des paramètres d'entrée renseignés par l'utilisateur.

Nous avons intégré la base des tests dans le fichier pom.xml mais cela reste pour l'instant imparfait.

EXTENSION

L'implémentation de la partie TRIGO nécessite une recherche approfondie des algorithmes usuels, cette recherche prend beaucoup de temps afin de bien comprendre des méthodes qui sont parfois vaguement décrites. La première idée qui nous est venu à l'esprit est l'utilisation

des séries entières (Notamment cosinus et sinus grâce à la convergence rapide des deux séries) . Cet algorithme donne des résultats compatibles à 2 ULPs prés en moyenne et 2 Ulp au maximum(sur 1 635 779 tests) , concernant la fonction ulp(), on a réalisé une multitude de versions à cause du manque de la documentation, la fonction implémentée donne des résultats compatibles sans erreur(sur 10 240 000 tests) , enfin la fonction arctan que nous avons implémenté est basé sur les polynômes d'Hermite, elle donne des résultats compatibles à 2 ULPs en moyenne et un maximum de 2 ulps(38400 tests).