Projet de Compilation
Compilateur de code écrit en TPC

Table des matières

GHAMRI Samy
OSAJ Xhavit

Description du projet

Ce projet a consisté à créer un compilateur pour un langage appelé ***tpc***. Nous avions comme tâche de créer une grammaire pour analyser syntaxiquement le code source écrit en tpc, pour cela nous avons utiliser des langages comme ***Flex*** et ***Bison.*** Nous devions ensuite générer l'arbre abstrait et la tables des symboles. Ensuite, nous devions nous pencher sur les erreurs syntaxiques qui peuvent se glisser dans le code source. Enfin si le code passait l'analyse syntaxique et sémantique, alors on devait le traduire en assembleur afin de créer le fichier cible pour au final créer l'exécutable permettant d'exécuter notre code source.

Mode d'emploi

Compilation du projet

Pour compiler le projet, veuillez ouvrir un terminal et vous placer dans le répertoire du projet qui contient un *makefile* ainsi que les différents fichiers et dossiers nécessaires.

Tapez ensuite la commande ***make*** dans le terminal afin de compiler le projet.

Un exécutable avec le nom ***tpcc*** devrait alors être crée dans le répertoire ***bin***.

Pour supprimer les différents fichiers objet crées, il suffit d'entrer la commande ***make clean*** dans le terminal.

Pour supprimer l'exécutable et les fichiers objets, il suffit d'entrer la commande *make vacuum* dans le terminal.

Exécution du projet

Pour exécuter le projet, tapez la commande ***./bin/tpcc.*** Vous pourrez alors écrire du code tpc dans le terminal.

Pour exécuter le projet avec un de vos code tpc écrit dans un fichier, tapez la commande **./bin/tpcc < NomDeFichier.tpc**

Options du programme

Le programme permet de lui entrer des paramètres pour l'affichage de l'arbre abstrait et/ou des tables des symboles et/ou d'une aide.

Arbre abstrait

Pour afficher l'arbre abstrait, ajouter l'option **-t** ou bien **--tree** lors de l'exécution, par exemple

./bin/tpcc -t < NomDeFichier.tpc ou bien .**/bin/tpcc --tree < NomDeFichier.tpc**

Tables des symboles

Pour afficher les tables des symboles, ajouter l'option **-s** ou bien **--symtabs** lors de l'exécution, par exemple

./bin/tpcc -s < NomDeFichier.tpc ou bien .**/bin/tpcc --symtabs < NomDeFichier.tpc**

Aide

Pour afficher l'aide, ajouter l'option **-h** ou bien **--help** lors de l'exécution, par exemple

./bin/tpcc -h < NomDeFichier.tpc ou bien .**/bin/tpcc --help < NomDeFichier.tpc**

Exécution du script Bash

Ce projet dispose aussi un script ***Bash*** permettant de réaliser des tests automatiquement sur une larges séries des codes source tpc correct ou non.

Pour lancer ces tests automatique, veuillez vous placer à la racine du projet, là où est le fichier contenant le script, et entrez la commande ***./script.sh***
Un fichier ***rapport_tests.txt*** contenant un récapitulatif des résultats des tests est alors crée.

Rajout de fichiers test personnels

Si vous voulez rajouter vos propres fichier test contenant du code tpc, vous devez le placer dans le dossier correspondant à votre fichier dans le dossier test. **Votre code doit obligatoirement avoir comme extension .tpc**. Par exemple, si votre fichier ne contient pas d'erreur syntaxiques ou sémantiques, placez le dans ***test/good***. Si votre fichier provoque un warning, placez le dans ***test/warn***. Si votre fichier test provoque une erreur syntaxique (respectivement une erreur sémantique) placez le dans ***test/syn-err*** (respectivement ***test/sem-error***).

Architecture

Organisation du projet

Le projet est composé de plusieurs sous-dossiers et fichiers.

Il y a le dossier **bin** qui va contenir le futur exécutable.

Il y a le dossier **doc** qui contient ce rapport.

Il y a le dossier **obj** qui va contenir tous les fichiers objet qui seront générés lors de la compilation.

Il y a le dossier **src** qui contient les fichiers sources de notre compilateur.

Il y a le dossier **test** qui contient quatre dossiers contenant respectivement des fichiers test correct, avec des erreurs syntaxiques, avec des erreurs sémantiques et des fichiers test générant des warnings.

Il y a le **Makefile** qui permet de compiler les différents fichiers composants le projet.

Il y a un **script Bash** permettant de lancer des tests automatiques au compilateur.

Modularité

Notre projet se décompose en 5 modules.

Il y a le module **abstract-tree** dans lequel on s'occupe de la création de l'arbre abstrait ainsi que des fonctions liées à l'arbre abstrait.

Il y a le module **code_nasm** dans lequel on s'occupe de la création du fichier cible ainsi que des fonctions liées au fichier cible.

Il y a le module **parser** dans lequel on s'occupe du parcours de l'arbre abstrait et des tables des symboles.

Il y a le module SymbolTable dans lequel on s'occupe de la création des tables des symboles ainsi que des fonctions liées aux tables des symboles.

Il y a le fichier écrit en **Flex** permettant de créer un analyseur lexical.

Il y a le fichier écrit en Bison permettant de créer la grammaire du projet.

Nous avons choisi de modulariser le projet de cette manière afin de le rendre assez épurée et donc facile à lire, à modifier et donc à mettre à jour.

Difficultés rencontrées

Au cours de la réalisation du projet, nous avons rencontrés un nombre important de problèmes et de difficultés. Ces problèmes nous ont ralenti mais pour la plupart nous avons réussi à les surmonter même si nous n'avons pas pu totalement finir notre compilateur.

La première grosse difficulté que nous avons rencontrée était un problème avec les noms des fonctions et des structures dans l'arbre abstrait. En effet, si une fonction avait des arguments, son nom dans l'arbre abstrait était écrasé par le nom de son dernier argument, c'est-à-dire si une fonction ***test(int a, int b) {…}*** avait comme nom, dans l'arbre abstrait, **b** au lieu de **test**. Pour résoudre ce problème, nous nous sommes penchés sur sa cause et cela était causé par un appel du token ***IDENT*** qui écrasait du coup le nom de la fonction et le remplaçait par le nom de son dernier argument. Pour régler ça, nous avons sauvegarder le nom de la fonction avant que le token soit appelé sur ses arguments ce qui a permis d'avoir le bon nom de fonction dans l'arbre abstrait et du coup de pouvoir continuer le projet dans de bonnes conditions. Malheureusement ce problème nous a fait perdre des points dans la note du rendu intermédiaire et cela aurait pu être évitée.

Nous avons aussi eu un problème avec la partie du projet permettant de créer le code cible. Nous n'étions clairement pas les meilleurs dans la création du code en assembleur. Les connaissances sur cet aspect du codage nous ont fait défaut ce qui a rendu les choses clairement plus difficiles. Notre création de code cible n'est pas terminée ce qui ne permet pas d'avoir un compilateur totalement fonctionnel. Nous avons donc préféré bien finir toutes les autres parties du projet bien comme il faut avant de mettre toutes nos forces restantes dans la génération de code cible. On est déçu de ne pas avoir pu s'attaquer totalement à l'assembleur qui nous paraissait tellement intéressant pour la compréhension l'informatique en général. Il nous reste donc un goût amer car nous n'avons pas totalement fini ce compilateur.

Vérifier toutes les erreurs sémantiques était quelque chose de long aussi car il y a beaucoup de choses à regarder et il ne faut pas oublier des cas.

La longueur du projet était aussi une difficulté à prendre en compte. Nous avons travailler au moins 2 heures presque chaque jour pour pouvoir avancer le projet sans être dans une situation insurmontable. Notre chargé de TD nous a aussi beaucoup aidé en répondant à nos questions et en nous aiguillant dans la bonne direction et nous le remercions pour ça.

Expérience d’apprentissage

Le contexte sanitaire actuel ne nous a pas permis de nous réunir dans un lieu physique pour avancer le projet, cela a un peu compliqué les choses mais nous avons appris à gérer la répartition du travail, ainsi que la communication entre nous. Pour cela nous avons beaucoup utilisé Discord, qui nous permettait à la fois de communiquer par écrit (transfert de fichiers, questions, remarques, etc...) et par vocal.

Nous avons pu avoir un aperçu de comment un langage fonctionne, de comment les cas d’erreur sont gérés et de comment un exécutable est crée.

Ce projet nous a beaucoup intéressé car on a vu les étapes à suivre lors de la création d’un nouveau langage de programmation et nous avons une idée assez claire de la difficulté que cela représente.