Université Gustave Eiffel

Projet de Compilation

Licence d'informatique

—2020-2021—

Le but du projet est d'écrire un compilateur en utilisant

- `flex` pour l'analyse lexicale,
- `bison` pour l'analyse syntaxique et la construction de l'arbre abstrait,
- et le langage C pour parcourir l'arbre abstrait et produire le code cible.

Le langage source, TPC, ressemble à un tout petit sous-ensemble du langage C.

Le langage cible est l'assembleur `nasm` dans sa syntaxe pour Unix (option de compilation `-f elf64`) et avec les conventions d'appel AMD 64. Vous vérifierez le résultat de la compilation d'un programme en exécutant le code obtenu.

Le projet est à faire en binôme ou seul. Si vous préférez le faire en binôme mais que vous n'avez pas de partenaire, contactez Eric Laporte. Chaque binôme réutilisera son projet d'analyse syntaxique du premier semestre et pourra le modifier [1]. Les dates limite de rendu sont :

- le lundi 26 avril 2021 à 23h59 pour une première version du compilateur [2] avec les fonctionnalités décrites ci-dessous dans la section Travail demandé, Rendu intermédiaire (1/4 de la note de projet) ;
- le dimanche 6 juin 20201 à 23h59 pour le compilateur complet (3/4 de la note de projet).

1 Définition du langage source TPC

La syntaxe du TPC est celle du projet d'analyse syntaxique de 2020-2021, y compris l'extension autorisant les types structures.

Votre compilateur doit aussi détecter les erreurs sémantiques. La sémantique de la plupart des expressions et instructions du langage est la sémantique habituelle en langage C.

Tout identificateur utilisé dans un programme doit être déclaré avant son utilisation et dans la partie de déclaration appropriée.

Tout programme doit comporter la fonction particulière `main` par laquelle commence l'exécution et qui renvoie obligatoirement un `int`.

Les arguments d'une fonction sont transmis par valeur. Les fonctions récursives directes et indirectes sont autorisées.

L'instruction `readc()` permet de lire un caractère saisi au clavier par l'utilisateur, et `reade()` permet de lire un entier saisi en notation décimale.

Le typage des expressions est à peu près comme en C :

- Le type `int` doit pouvoir représenter les entiers signés codés sur 4 octets.
- L'instruction `print()` doit afficher un entier ou un caractère, en fonction du type de l'expression.
- Toute expression de type `char` à laquelle on applique une opération (sauf `print`) est implicitement convertie en `int`.
- Toute expression peut être interprétée comme booléenne, avec la convention que 0 représente "faux" et tout autre valeur "vrai", et le résultat de toute opération booléenne est de type `int`. En particulier, l'opérateur de négation produit comme résultat 1 quand on l'applique à 0.

(1). Si vous avez déjà validé Analyse syntaxique une autre année, mais pas Compilation, vous devez quand même faire le projet d'analyse syntaxique (on ne peut pas faire de compilateur sans analyseur syntaxique). Un analyseur syntaxique rudimentaire est fourni sur la plate-forme e-learning.

(2). Déposez votre projet sur la plateforme e-learning dans la zone prévue à cet effet.

— Si on affecte une expression de type `int` à une `LValue` de type `char`, le compilateur doit émettre un avertissement (*warning*) mais produire un programme cible fonctionnel.
— Dans les appels de fonctions, on considère que chaque paramètre effectif correspond à une affectation pour ce qui concerne le typage ; l'instruction `return Exp` est aussi considérée comme une affectation. Par exemple, dans une fonction qui renvoie un `int`, `return 'a'` est correct, mais dans une fonction qui renvoie un `char`, `return 97` doit provoquer un avertissement.
— Les instructions `reade()` et `readc()` sont aussi considérées comme des affectations pour ce qui concerne le typage. Par exemple, si `count` est de type `int`, `readc(count)` est correct, mais si `my_char` est de type `char`, `reade(my_char)` doit provoquer un avertissement.

2 Rendu intermédiaire

2.1 Travail demandé

On demande une première version du compilateur qui doit au moins :
— détecter les erreurs lexicales et syntaxiques [3],
— construire l'arbre abstrait,
— construire la ou les tables des symboles et y mettre au moins les variables.
On demande aussi d'écrire deux jeux d'essais, un pour les programmes TPC corrects sans avertissements, et un autre pour les programmes avec des erreurs lexicales ou syntaxiques [4]. Enfin, on demande un script de déploiement des tests, qui produit un rapport unique donnant les résultats de tous les tests.

2.2 Organisation

Nous vous demandons de respecter l'organisation suivante. *Pour évaluer vos projets, nous supposerons que vous l'aurez fait.* Le répertoire que vous déposerez doit s'appeler `ProjetCompilationL3_NOM1_NOM2`, contenir à la racine un makefile nommé `makefile` et au moins les 5 sous-répertoires suivants :
— `src` pour les fichiers sources écrits par les humains,
— `doc` pour le rapport,
— `bin` pour le fichier binaire (votre compilateur doit être nommé `tpcc`),
— `obj` pour les fichiers intermédiaires entre les sources et le binaire,
— `test` pour les jeux d'essais, avec des sous-répertoires :
  — `good`
  — `warn`
  — `syn-err`
  — `sem-err`
Votre compilateur doit avoir l'interface utilisateur suivante.
— Ligne de commande :
  — on doit au moins pouvoir lancer le compilateur par `./tpcc [OPTIONS]` (*pour évaluer vos projets, nous lançons des tests automatiques qui utilisent cette commande*) ;
  — vous pouvez aussi implémenter la ligne de commande `./tpcc [OPTIONS] FILE.tpc`
— Options [5] : au moins
  `-t`, `--tree` affiche l'arbre abstrait sur la sortie standard
  `-s`, `--symtabs` affiche toutes les tables des symboles sur la sortie standard
  `-h`, `--help` affiche une description de l'interface utilisateur et termine l'exécution

(3). Les messages d'erreur donneront le numéro de ligne, le numéro du caractère dans la ligne, puis une reproduction de la ligne et l'indication du caractère par une flèche verticale ou un circonflexe.
(4). Si votre compilateur redémarre après une ou plusieurs erreurs et qu'ensuite l'exécution se termine normalement, il devra renvoyer un code 1.
(5). Pour analyser la ligne de commande vous pouvez utiliser la fonction `getopt()`.

— Valeur de retour :
  — 0 si le programme source ne contient aucune erreur (même s'il y a des avertissements)
  — 1 s'il contient une erreur lexicale ou syntaxique
  — 2 s'il contient une erreur sémantique
  — 3 ou plus pour les autres sortes d'erreurs : ligne de commande, fonctionnalité non implémentée, mémoire insuffisante...

  Quand nous évaluerons votre projet, nos tests automatiques enregistreront chaque valeur de retour, et votre compilateur gagnera des points s'il renvoie les bonnes valeurs.
— Fichier cible :
  — si la ligne de commande ne donne pas de nom de fichier d'entrée, le compilateur lit sur l'entrée standard et par défaut le nom du fichier cible est `_anonymous.asm`
  — si la ligne de commande donne le nom d'un fichier d'entrée `FILE.tpc`, par défaut le nom du fichier cible est `FILE.asm`.

2.3 Modifications de la grammaire

En plus d'autoriser les types structures, vous pouvez modifier la grammaire fournie, par exemple pour lever des conflits d'analyse ou pour faciliter la traduction, mais vos modifications ne peuvent s'écarter de la définition du langage TPC que si cela l'enrichit.

2.4 Dépôt

Déposez votre projet sur la plateforme e-learning dans la zone prévue à cet effet, sous la forme d'une archive tar compressée nommée `ProjetCompilationL3_NOM1_NOM2.tar.gz`, qui, au désarchivage, crée un répertoire contenant le projet.

3 Rendu final

On demande de respecter la même organisation et les mêmes conventions pour le dépôt du projet, avec, en plus du rendu intermédiaire :
— la détection des erreurs sémantiques et les avertissements ;
— deux jeux d'essais supplémentaires, un pour les programmes TPC corrects avec des avertissements et un pour les programmes avec des erreurs sémantiques [(6)] ;
— la génération du code cible en `nasm`, y compris si le programme a des types structures, plusieurs fonctions et des appels de fonctions ;
— un rapport sur les difficultés que vous avez rencontrées et les choix que vous avez faits pour les résoudre.

(6). Si votre compilateur redémarre après une ou plusieurs erreurs et qu'ensuite l'exécution se termine normalement, il devra renvoyer un code 1 ou 2 correspondant à au moins une des erreurs détectées.