Université Gustave Eiffel

Projet de Compilation

Licence d'informatique

—2021-2022—

Le but du projet est d'écrire un compilateur en langage C.

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. Vous pouvez vous mettre en binôme avec quelqu'un d'un autre groupe. 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 en `flex` et `bison` et pourra le modifier [(1)]. Les dates limite de rendu sont :

- le 18 avril 2022 à 23h59 pour une première version du compilateur ; pour vous situer dans l'avancement de votre projet, nous vous donnerons un score que nous obtiendrons en lançant automatiquement la compilation de votre projet et l'exécution sur nos jeux d'essais ; ce score n'entrera pas dans votre note de projet ;
- le 6 juin 2022 à 23h59 pour le compilateur complet.

1 Définition du langage source TPC

La syntaxe du TPC est celle du projet d'analyse syntaxique de 2021-2022, y compris l'extension autorisant les `switch`. La syntaxe pour les `switch` est celle du langage C, sauf pour deux exceptions, qui sont les deux consignes données pour le projet d'analyse syntaxique, répétées ici :

- après un `switch`, les parenthèses puis les accolades seront obligatoires ;
- il pourra y avoir des `case`, des `default` et des `break` à l'intérieur des accolades reliées directement à un `switch`, mais pas à l'intérieur des autres accolades.

Attention, la syntaxe du langage C autorise des choses bizarres dans les `switch`, et donc celle du TPC aussi, par exemple il peut y avoir [(2)]

- des instructions avant le premier `case`,
- un `default` avant un `case`,
- plusieurs `default` dans le même `switch`... [(3)]

Pour savoir quelles bizarreries sont des erreurs, et quelles erreurs doivent être traitées comme syntaxiques ou sémantiques, voir le document "Switch en TPC : erreurs syntaxiques ou sémantiques" sur la plateforme elearning dans la section Projet.

En cas d'erreur lexicale ou syntaxique, on ne demande pas que le compilateur redémarre après l'erreur.

Votre compilateur doit aussi détecter les erreurs sémantiques. La sémantique des expressions et instructions du langage est la sémantique habituelle en langage C, sauf indication contraire. Dans le cas d'une erreur sémantique, votre compilateur doit émettre un message d'erreur, puis s'arrêter ou continuer la compilation, ce choix est libre.

(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).

(2). Nous vous recommandons de ne pas écrire ce genre de chose en tant que développeur quand vous écrivez vous-même un switch, mais nous vous demandons de les autoriser en tant qu'auteur d'un compilateur.

(3). Le fait qu'il y ait plusieurs default rattachés au même switch est une erreur sémantique. Une grammaire qui spécifie la syntaxe du C ou du TPC l'autorise, parce que c'est une contrainte difficile à spécifier dans une grammaire algébrique. Mais un compilateur doit signaler l'erreur.

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

Une variable globale et une fonction ne peuvent pas avoir le même nom.

Une variable locale et un paramètre d'une même fonction ne peuvent pas avoir le même nom.

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.

Les fonctions `getchar()` et `getint()` permettent de lire un caractère et un entier en notation décimale saisis au clavier par l'utilisateur. Elles renvoient le caractère ou l'entier comme valeur de retour. Les fonctions `putchar()` et `putint()` permettent d'afficher sur la sortie standard un caractère et un entier passés en paramètre. L'entier est affiché en notation décimale. Les fonctions `putchar()` et `putint()` ne renvoient pas de valeur. En TPC, ces quatre fonctions peuvent être appelées, mais pas définies. Votre compilateur doit les définir en `nasm` et les incorporer au code cible.

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

- Le type `int` doit pouvoir représenter les entiers signés codés sur 4 octets.
- Toute expression de type `char` à laquelle on applique une opération est implicitement convertie en `int`. Par exemple, si `count` est de type `int`, `count='a'` est correct et ne doit pas provoquer d'avertissement.
- 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.
- 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. Par exemple, si `my_char` est de type `char`, `my_char=97` doit provoquer un avertissement.
- 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.

2 Rendu intermédiaire

2.1 Travail demandé

On demande une première version du compilateur qui doit au moins :

- détecter les erreurs lexicales et syntaxiques, et émettre des messages d'erreur lexicale ou syntaxique qui donneront le numéro de ligne,
- détecter les erreurs sémantiques correspondant
  - aux variables ou paramètres redéclarés comme variables ou paramètres,
  - et aux variables ou paramètres utilisés sans avoir été déclarés,

  et émettre des messages d'erreur sémantique qui donneront le numéro de ligne.

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 3 sous-répertoires suivants :

- `src` pour les fichiers sources écrits par les humains,
- `bin` pour le fichier binaire (votre compilateur doit être nommé `tpcc`),

— `obj` pour les fichiers intermédiaires entre les sources et le binaire.

Si vous avez des jeux d'essais, placez-les dans un 4e sous-répertoire `test`, frère des trois ci-dessus, avec au moins trois sous-répertoires :

— `good`
— `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]` (dans ce cas, la seule façon de donner le fichier d'entrée TPC en ligne de commande est de faire une redirection par `<`) ; *pour évaluer vos projets, nous lançons des tests automatiques qui utilisent cette commande* ;
  — vous pouvez aussi implémenter, en plus, la ligne de commande `./tpcc [OPTIONS] FILE.tpc`
— Options [4] : au moins
  `-t`, `--tree` affiche l'arbre abstrait sur la sortie standard
  `-h`, `--help` affiche une description de l'interface utilisateur et termine l'exécution
— 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 [5]
  — 2 s'il contient une erreur sémantique [6]
  — 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.

2.3 Modifications de la grammaire

En plus d'autoriser les `switch`, vous pouvez modifier la grammaire fournie, par exemple pour lever des conflits d'analyse ou pour faciliter la traduction, mais vos modifications ne doivent pas changer le langage engendré.

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 les mêmes consignes sur les modifications éventuelles de la gramaire et le dépôt du projet.

3.1 Travail demandé

En plus du rendu intermédiaire :

(4). Pour analyser la ligne de commande vous pouvez utiliser la fonction `getopt()`.

(5). Si votre compilateur redémarre après une ou plusieurs erreurs et qu'ensuite l'exécution se termine normalement, il devra renvoyer un code 1 dans le cas d'erreurs lexicales ou syntaxiques.

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

— la détection des autres erreurs sémantiques, avec le numéro de ligne ;
— les avertissements, avec le numéro de ligne ;
— quatre jeux d'essais :
  — un pour les programmes TPC corrects sans avertissements,
  — un pour les programmes avec des erreurs lexicales ou syntaxiques,
  — un pour les programmes avec des erreurs sémantiques,
  — un pour les programmes TPC corrects avec des avertissements ;
— un script de déploiement des tests, qui produit un rapport unique donnant les résultats de tous les tests ;
— la génération du code cible en `nasm`, y compris si le programme contient des `switch`, 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.

3.2 Organisation

Le répertoire `ProjetCompilationL3_NOM1_NOM2` doit contenir les sous-répertoires suivants, comme frères de ceux déjà demandés pour le rendu intermédiaire :
— `doc` pour le rapport,
— `test` pour les jeux d'essais, avec des sous-répertoires :
  — `good`
  — `syn-err`
  — `sem-err`
  — `warn`

Votre compilateur doit avoir l'interface utilisateur demandée pour le rendu intermédiaire, avec en plus :
— Options : en plus des précédentes, au moins `-s`, `--symtabs` qui affiche toutes les tables des symboles sur la sortie standard ;
— 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`.

3.3 Évaluation

Comme pour le projet d'analyse syntaxique, l'évaluation du projet de compilation se décompose en une évaluation automatique et une évaluation manuelle. Le barème est donné ici à titre indicatif.

L'évaluation automatique (8 points) reprend les tests du projet d'analyse syntaxique et y ajoute des tests sur la détection d'erreurs sémantiques. Ces tests sont orthogonaux autant que possible afin qu'une erreur ne soit pénalisée qu'à sa juste valeur. Sauf exception, les résultats représentent votre avancée dans l'implantation des fonctionnalités demandés.

Seront évalués manuellement :
— la production de code assembleur (3 points)
— l'arbre abstrait et les tables de symboles (3 points)
— les messages d'avertissement et d'erreur (2 points)
— l'organisation, la documentation et votre banc de tests (4 points)

Il est essentiel que l'interface de votre programme respecte les consignes données ci-dessus dans les parties 2.2 et 2.3, et notamment qu'il fonctionne par défaut en analysant son entrée standard. Ainsi doivent par exemple fonctionner :

```
$ ./tpcc < ../tests/good/09-switch-simple.tpc
$ cat ./tests/sem-err/12-double-declaration.tpc | ./bin/tpcc
```

Bon travail !