

# Projet de Compilation

## Licence d'informatique

—2023-2024—

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 de vos jeux d'essais en exécutant le code obtenu.

Le projet est à faire en binôme ou seul. Vous pouvez vous choisir un binôme librement dans n'importe quel groupe. Si vous n'en avez pas trouvé, vous pouvez contacter Eric Laporte. Chaque binôme réutilisera son projet d'analyse syntaxique du premier semestre en **flex** et **bison** et pourra le modifier <sup>(1)</sup>.

La date limite de rendu est le 24 mai 2024 à 18 h.

## 1 Définition du langage source TPC

La syntaxe du TPC est celle du projet d'analyse syntaxique de 2023-2024, y compris l'extension autorisant les tableaux.

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.

### 1.1 Identificateurs

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.

### 1.2 Fonctions

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

L'utilisation des instructions **return** doit être conforme au type de retour de la fonction dans laquelle elles apparaissent. Un appel de fonction ne peut pas être utilisé comme expression si le type de retour de la fonction appelée est **void**.

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. On ne demande pas

---

(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).

que `getint()` saute les caractères blancs : si le premier caractère lu n'est pas un chiffre ni un signe moins, `getint()` doit terminer l'exécution du programme et le programme doit renvoyer la valeur de retour 5, qui code une erreur d'entrées-sorties. 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. Le compilateur écrira ces fonctions directement en `nasm`, et le code `tpc` ne pourra pas les redéfinir.

## 1.3 Types

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.

## 1.4 Tableaux

La sémantique pour les tableaux est celle du langage C, sauf indication contraire ci-dessous.

Dans une déclaration de variable de type tableau, la taille du tableau devra être strictement positive. Un tableau de taille nulle est une erreur sémantique.

- Si la grammaire de votre analyseur syntaxique impose que la taille du tableau soit une constante numérique simple (exemple : `int duration[12];`), un tableau de taille strictement négative devra être une erreur syntaxique.
- Si au contraire la grammaire de votre analyseur syntaxique permet que la taille du tableau soit une expression constante (exemple : `int duration[365/30];`), un tableau de taille strictement négative devra être une erreur sémantique.

Dans une instruction, les seuls opérateurs applicables à un tableau sont les crochets, comme dans `duration[2]`, et le fait de passer le tableau en argument dans un appel de fonction, comme dans `bissextile(duration,12);`. Appliquer un autre opérateur à un tableau, ou l'utiliser comme booléen, comme dans `if (duration) return;`, est une erreur sémantique.

# 2 Spécifications

## 2.1 Travail demandé

On demande :

- un compilateur qui détecte les erreurs lexicales et syntaxiques, et émet des messages d'erreur lexicale ou syntaxique qui donneront le numéro de ligne, et le numéro dans la ligne d'un caractère proche de l'erreur,
- les erreurs sémantiques, avec des messages d'erreur sémantique qui donneront 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 tableaux, 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.

## 2.2 Organisation

Nous vous demandons de respecter l'organisation suivante. *Pour évaluer vos projets, nous supposons 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,
- `rep` pour votre rapport,
- `test` pour les jeux d'essais, avec des sous-répertoires :
  - `good`
  - `syn-err`
  - `sem-err`
  - `warn`

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 bonne 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<sup>(2)</sup> : 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
  - `-s, --syntabs` qui affiche toutes les tables des symboles sur la sortie standard ;
- 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<sup>(3)</sup>
  - 2 s'il contient une erreur sémantique<sup>(4)</sup>
  - 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 :

---

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

(3). 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.

(4). 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.

- si la ligne de commande ne donne pas de nom de fichier d’entrée (`./tpcc [OPTIONS] < FILE.tpc`), 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` (exemple : `./tpcc [OPTIONS] FILE.tpc`), par défaut le nom du fichier cible est `FILE.asm`.

## 2.3 Modifications de la grammaire

En plus d’autoriser les tableaux, 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 Nommage et 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 Évaluation

Comme pour le projet d’analyse syntaxique, l’évaluation du projet de compilation comporte 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, votre rapport et votre banc de tests (4 points)

Il est essentiel que l’interface de votre programme respecte les consignes données ci-dessus dans la partie 2, et notamment qu’il fonctionne par défaut en analysant son entrée standard. Ainsi doivent par exemple fonctionner :

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

Bon travail !