

Documentation utilisateur







ANDRIANARISOLO Elie

FARHAT Widad

SARR Seynabou

TONG An Jun

Sommaire

	1. Introduction	3
	2. Prérequis	3
	3. Installation du compilateur ifcc	4
	4. Utilisation du compilateur ifcc	5
	5. Fonctionnalités du compilateur ifcc	5
	6. Tests	6



1. Introduction

Ce document présente le projet de développement d'un compilateur de langage C, nommé `ifcc`, réalisé dans le cadre d'un projet de longue durée en quatrième année à l'INSA Lyon. Ce compilateur, écrit en C++ avec ANTLR4, se concentre sur la génération de code assembleur x86 pour un sous-ensemble du langage C. Il fournira des instructions détaillées pour l'installation et l'utilisation du compilateur.



2. Prérequis

Pour installer le compilateur `ifcc`, il est essentiel de préparer correctement votre environnement de travail. D'une part, assurez-vous d'avoir Java et C++ installés sur votre système. Ces deux langages sont nécessaires pour exécuter et compiler le code du compilateur `ifcc`. Vous pouvez télécharger et installer les versions appropriées depuis les sites officiels respectifs de Java et de g++.

D'autre part, ANTLR4 est crucial pour la génération d'analyseurs syntaxiques dans le projet `ifcc`. Il est donc indispensable d'avoir ANTLR4 correctement installé et configuré selon les spécifications de votre système. Voici différents cas de figure pour l'installation d'ANTLR4 (qui n'est en principe à faire qu'une seule fois sur chaque machine) :

- Machines linux du département (salle 208) : utilisez le script `./install-antlr.sh` (fourni dans le dossier initial du PLD Compilateur sur Moodle) pour installer ANTLR4 dans le répertoire de votre choix. Cela marche uniquement dans un sous-répertoire de votre home linux, et ce script ne supporte pas les espaces dans les noms de répertoire.
- Ubuntu ≥ 20 : ANTLR est pré-emballé, il faut installer trois paquets :

```
$ sudo apt install antlr4 libantlr4-runtime-dev default-jdk
```

- Ubuntu < 20 : Upgrader à un ubuntu ≥ 20 , la sécurité de ces versions n'est plus assurée.
- Windows : Utiliser WSL, version 2 minimum, et à jour (dans le doute, tapez `wsl --update` dans un powershell). Ensuite ouvrir un terminal linux et procéder comme pour ubuntu ci-dessus.
- Machines Mac OS : utilisez le script `./install-antlr.sh` (fourni dans le dossier initial du PLD Compilateur sur Moodle) comme pour une machine du département.

Si vous devez exécuter `./install-antlr.sh` : après de longs écrans de messages divers et variés (avec même de la couleur pour les chanceux), si vous voyez finalement une ligne disant PLD COMP : ANTLR OK alors c'est que votre installation s'est passée correctement.

Une fois ces prérequis pris en compte, vous pourrez passer à l'installation proprement dite du compilateur `ifcc`, dont les étapes détaillées seront abordées dans la section suivante.



3. Installation du compilateur ifcc

Nous utilisons un Makefile pour générer l'exécutable ifcc. Pour se faire, il faut se placer dans le répertoire "compiler" puisque le Makefile s'y trouve :

```
$ cd pld-comp/compiler
```

NB : Il est intéressant de noter que l'un des intérêts de l'inclure au début du Makefile fourni est d'isoler du Makefile partagé la partie qui va différer d'un OS à l'autre. Ceci permet ainsi d'avoir un makefile partagé par un projet dont les membres ont différents systèmes d'exploitation.

Une fois dans le répertoire compiler, exécutez la commande make. Cette action déclenche la séquence d'opérations suivante :

- Tout d'abord, l'outil ANTLR sera invoqué pour traduire notre grammaire ifcc.g4 en plusieurs classes C++ dans le répertoire generated.
- Ensuite, plusieurs appels à G++ seront effectués pour compiler l'ensemble du code généré.
- Enfin, un dernier appel à G++ sera effectué pour lier toutes les parties ensemble et produire l'exécutable ifcc pouvant être trouvé dans le répertoire courant.

Voici l'ensemble des commandes associé à notre Makefile :

- `make` : Compile le projet pour produire l'exécutable ifcc.
- `make clean` : Supprime les fichiers binaires et les fichiers objets générés, ainsi que tous les fichiers générés lors de la compilation.
- `make re` : Supprime tous les fichiers binaires, objets et fichiers générés précédemment, puis reconstruit entièrement le projet à partir de zéro.
- `make gui FILE=CHEMIN_FICHER` : Affiche l'arbre de dérivation dans une fenêtre graphique du fichier donné situé au CHEMIN_FICHER. Il est intéressant de souligner que cette visualisation nécessite le compilateur javac. Assurez-vous donc d'avoir installé les paquets nécessaires, par exemple en tapant `sudo apt install default-jdk`.
- `make test TEST_FILES=CHEMIN_TEST` : Compare la compilation et l'exécution des fichiers tests situés à CHEMIN_TEST entre notre compilateur ifcc et gcc pour valider le bon fonctionnement de ifcc.



4. Utilisation du compilateur ifcc

Après avoir généré l'exécutable ifcc, il suffit maintenant de se placer dans le dossier compiler et d'exécuter la commande qui suit :

```
$ ./ifcc [nom de votre fichier C]
```

L'exécution de cette commande produira du code source assembleur x86 sur la sortie standard.

Pour créer un exécutable, assemblez le code généré et effectuez l'édition des liens avec les commandes suivantes (en considérant ici un fichier C nommé file.c et l'exécutable exe qui lui est associé) :

```
$ ./ifcc file.c > file.s  
$ as -o file.o file.s  
$ gcc file.o -o exe  
$ ./exe
```



5. Fonctionnalités du compilateur ifcc

ifcc compile des fichiers en langage C, mais certaines fonctionnalités du langage ne sont pas prises en charge. L'utilisation de ces fonctionnalités non implémentées peut entraîner des erreurs de syntaxe à la compilation ou des résultats inattendus lors de l'exécution.

La liste suivante donne toutes les fonctionnalités disponibles à la compilation avec ifcc :

- Un seul fichier source sans pré-processing. Les directives du préprocesseur sont autorisées par la grammaire, mais ignorées, ce afin de garantir que la compilation par un autre compilateur soit possible (exemple : inclusion de `stdio.h`).
- Les commentaires sont ignorés.
- Types de données : `int` et `char`.
- Déclaration de variables n'importe où .
- Affectation d'une valeur à une variable (qui, en C, retourne aussi une valeur).
- Opérations arithmétiques (+, -, *, /, %), opérations bit-à-bit (|, &, ^), opérations unaires (!, -), Opérations de comparaison (==, !=, <, >, <=, >=), et expressions composées de constantes entières et de variables.
- Opérateurs d'affectation (+=, -=, *=, /=)

- Utilisation des fonctions standard putchar et getchar pour les entrées-sorties.
- Définition de fonctions avec paramètres, et type de retour int ou void
- Structure de blocs grâce à { et }
- Support des portées de variables et du shadowing
- Les structures de contrôle if, else, while
- Support du return expression n'importe où
- Vérification qu'une variable utilisée a été déclarée
- Vérification qu'une variable n'est pas déclarée plusieurs fois
- Vérification qu'une variable déclarée est utilisée



6. Tests

Un environnement de tests vous est fourni dans le répertoire tests pour automatiser les tests fonctionnels. Assurez-vous d'avoir Python installé sur votre machine.

Pour exécuter les tests, vous avez deux façons possibles de faire :

1. Si vous êtes dans le répertoire compiler, vous pouvez faire appel à la commande Makefile suivante :

```
$ make test TEST_FILES=CHEMIN_TEST
```

avec CHEMIN_TEST correspondant au chemin menant vers le(s) fichier(s) C que vous souhaitez tester. Il est à noter que si vous faites appel seulement à la commande make test, vous exécutez tous les tests fournis par défaut.

2. Si vous êtes dans le répertoire tests, il faudra s'appuyer sur la commande qui suit :

```
$ python3 ./ifcc-test.py CHEMIN_TEST
```

avec CHEMIN_TEST correspondant au chemin menant vers le(s) fichier(s) C que vous souhaitez tester.

Dans les deux cas, chaque test génère quatre résultats possibles :

- TEST OK : le test est réussi.
- TEST FAIL (votre compilateur accepte un programme invalide).
- TEST FAIL (votre compilateur rejette un programme valide).
- TEST FAIL (votre compilateur produit un assemblage incorrect).

Par ailleurs, les résultats détaillés de chaque test sont disponibles dans le sous-répertoire `ifcc-test-output`, avec un dossier distinct pour chaque test.

NB : Pour en savoir plus sur les options dispensées par `ifcc-test.py`, n'hésitez pas à employer la commande suivante dans le répertoire `tests` :

```
$ python3 ./ifcc-test.py --help
```