Documentation Utilisateur

Ce document livre une aide à l'installation et l'utilisation du compilateur de langage C, *ifcc*.

Nous distinguerons par la suite deux cas d'installation pour cet outil. L'installation avec Docker est plus simple à réaliser mais implique que les exécutables doivent être exécutés sur le conteneur virtuel Docker. Une exécution sur un autre environnement ne marcherait pas. L'installation avec Antlr peut être plus difficile à mettre en place, mais vous donne accès à la compilation d'exécutables directement sur votre machine.

1 - Avec Docker

Prérequis

Pour une installation du compilateur avec Docker, vous avez besoin d'installer Docker (https://www.docker.com/) sur votre machine. Puis d'importer le conteneur "*eguerin/antlr4cpp*". Cela peut être facilement réalisé en exécutant la commande

$ sh compiler/wrapper.sh

Installation

Après avoir installé Docker et le conteneur, il suffit de compiler les fichier sources du compilateur avec la commande.

$ sh compiler/compile_docker.sh

Utilisation

Pour compiler des fichiers en langage C, il vous suffit maintenant de vous placer dans le dossier *compiler* et d'exécuter la commande

$ sh wrapper.sh ./ifcc [nom de votre fichier C]

L'exécution de cette commande produira du code source assembleur x86 sur la sortie standard. Afin de créer un exécutable, il suffit de mettre le code généré dans un fichier, d'assembler ce code et de réaliser l'édition des liens. Cela se fait avec les commandes suivantes :

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

2 - Avec Antlr

Prérequis

Sous Ubuntu, il faudra télécharger le C++ runtime sur https://www.antlr.org/download.html (C++ target en bas de page)

Ensuite, décompresser puis lancer les commandes suivantes dans le répertoire associé :

```
$ sudo apt install cmake
$ sudo apt install uuid-dev
$ sudo apt install pkg-config
$ mkdir build && mkdir run && cd build
$ cmake ..
$ DESTDIR=../run make install
$ cd ../run/usr/local/include
$ sudo cp -r antlr4-runtime /usr/local/include
$ cd ../lib
$ sudo cp * /usr/local/lib
$ sudo ldconfig
```

Installation

Après avoir installé Antlr, il suffit de compiler les fichier sources avec la commande suivante :

$ sh compiler/compile_ubuntu.sh

Utilisation

Pour compiler des fichiers en langage C, il vous suffit maintenant de vous placer dans le dossier *compiler* et d'exécuter la commande

$ ./ifcc [nom de votre fichier C]

L'exécution de cette commande produira du code source assembleur x86 sur la sortie standard. Afin de créer un exécutable, il suffit de mettre le code généré dans un fichier, d'assembler ce code et de réaliser l'édition des liens. Cela se fait avec les commandes suivantes :

$ ./ifcc file.c > file.s

$ as -o file.o file.s

$ gcc file.o -o exe

$ ./exe

3 - Fonctionnalités du compilateur *ifcc*

Fonctionnalités implémentées

ifcc permet de compiler des fichiers rédigés dans le langage C. Cependant, toutes les fonctionnalités du langage n'ont pas été implémentées. La présence d'une fonctionnalité non implémentée dans un programme donnera, dans la majorité des cas, une erreur de syntaxe à la compilation ou un mauvais résultat lors de l'exécution.

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

- Un seul fichier source sans pré-processing (les directives de pré-processing sont ignorées)
- Types de données : *int* et *char*.
- Déclaration de variables ou de tableaux à une dimension n'importe où dans le programme.
- Affectation d'une valeur à une variable ou un élément de tableau lors de la déclaration ou n'importe où dans le programme.
- Opérations arithmétiques ('+', '-', '*'), opérations bit-à-bit ('|', '&', '^'), opérations unaires ('!', '-') et expressions composées de constantes entières, de variables et d'éléments de tableau.
- Tests utilisant les opérateurs de comparaison ('==', '!=', '<', '>', '<=', '>=').
- Boucles *if*, *else* et *while* sans retour à l'intérieur.
- Déclaration (avec initialisation possible) de variables globales
- Déclaration de fonctions (types de retour : *int*, *char* et *void*).
- Retour de variables, constantes, éléments de tableau et expressions
- Gestion des portées des variables
- Optimisations dans le calcul d'expression en utilisant jusqu'à 8 registres différents pour éviter de faire trop de moves dans les registres.

Choix d'implémentation

A plusieurs reprises, notre compilateur s'écarte du comportement d'autres compilateurs de C comme gcc ou clang. Ces choix font que notre compilateur accepte des programmes que gcc ou clang refuserait et inversement. Cependant, ces choix sont tout à fait justifiés dans le cas de notre implémentation de compilateur C.

Liste des choix d'implémentation de *ifcc* :

- Lancement d'erreurs lors que l'on utilise une variable non définie. En effet, nous ne définissons pas nos variables par défaut, donc pour éviter d'avoir des valeurs aberrants, nous n'autorisons pas l'utilisation d'une variable qui n'a pas été définie par avant.
- Gestion du type char comme un entier. Toutes nos variables déclarées comme des char sont converties en entier dans l'IR. Cela permet de simplifier de nombreuses opérations. Cependant, des adaptations pour gérer les char séparément des int a été implémentée en partie mais n'est pas utilisée.
- Gestion de l'opération -- comme une opération soustraction et d'une opération négation. gcc n'accepte pas cela car il y a un conflit avec l'opérateur '--' de décrémentation. Cependant, dans notre cas, cet opérateur n'est pas géré, donc le conflit n'existe pas et nous n'avons pas de raison de rejeter cette opération valide.

- Lancement d'une erreur lorsque nous tentons d'accéder à un élément en dehors d'un tableau. En effet, l'accès aux éléments d'un tableau se fait uniquement à l'aide d'une constante (l'accès à l'aide d'une variable ou une expression n'a pas été implémentée). Ainsi, nous pouvons vérifier que la constante permettant d'accéder à un élément ne soit pas supérieure ou égale à la taille du tableau ou inférieure à 0 et nous lançons une erreur à la compilation dans ce cas, alors que gcc le fait à l'exécution.
- Lors de l'appel d'une fonction qui n'est pas encore déclarée dans le programme, nous renvoyons une erreur, là où gcc ne renvoie un warning, parce que nous ne faisons pas la déclaration implicite des fonctions.

Limites du compilateur *ifcc*

Nous donnons ici une liste de fonctionnalités non-implémentées. Ce sont des fonctionnalités que nous avions commencé à implémenter, mais que nous n'avons pas terminé, ou des fonctionnalités facilement implémentables dans notre compilateur.

Liste de fonctionnalités non-implémentées :

- Nous ne renvoyons pas de warning lorsqu'une variable n'est pas utilisée dans le programme, mais cette fonctionnalité est simple à implémenter.
- On peut accéder aux éléments d'un tableau avec une constante comme indice, mais l'accès à l'aide d'une variable ou d'une expression comme indice n'est pas implémentée.
- On ne peut pas appeler une fonction avec une expression comme paramètre, uniquement avec des constantes et des variables.
- ifcc, de même que gcc, accepte les programmes avec un main sans return. Cela peut être corrigé très facilement avec notre grammaire.
- Nous ne pouvons pas enchaîner les opérations bit à bit.