BIAUD Alexandre – EL MIR Raoul – HASAN TAWFIQ Amar –
KAROUIA Lilia – MAILLARD Swan – PEIGUE Alix

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Projet Compilateur – Documentation Utilisateur

1. Introduction

Nous avons développé IFCC, un compilateur pour un sous-ensemble de C. Ce compilateur traduit le code source écrit dans ce sous-ensemble en une représentation intermédiaire (IR), permettant de générer un code compilé en assembleur Intel x86 ou en RISC-V.

2. Utilisation d'IFCC

Pour compiler un programme écrit dans le sous-ensemble de C pris en charge par IFCC, suivez ces étapes :

1. Lancez la commande `$ make` à la racine du dossier pour compiler IFCC.
2. Assurez-vous que l'exécutable **"./ifcc"** est présent à la racine du dossier.
3. Écrivez votre programme dans un fichier avec l'extension **'.c'**.
4. Utilisez la commande suivante pour compiler votre programme :

   `$ ifcc <nom_fichier_source>.c [-t target] [-o <nom_fichier_cible>.s]`

   Cela compile le fichier source en code assembleur. L'option `-t target` vous permet de spécifier l'architecture cible pour la génération de l'assembleur. Vous pouvez choisir entre `"x86-64"` pour Intel x86 et `"rv64"` pour RISC-V. Par défaut, le code est compilé en assembleur Intel x86. L'option `-o <nom_fichier_cible>.s` permet de spécifier le nom du fichier de sortie pour le code assembleur généré. Par défaut le nom du fichier de sortie est **"./out.s"**

5. Assemblez et linkez le code assembleur en fonction de l'architecture cible avec l'outil de votre choix. Vous pouvez par exemple utiliser :

- $ gcc <nom_fichier_source>.s [-o <nom_executable>] si le programme est compilé en assembleur Intel x86. Par défaut le nom de l'exécutable généré est **"./a.out"**
- $ riscv64-linux-gnu-gcc <nom_fichier_source>.s [-o <nom_executable>] si le programme est compilé en assembleur RISC-V. Par défaut le nom de l'exécutable généré est **"./a.out"**

6. Exécutez le programme avec la commande $ ./a.out , ou utilisez un émulateur si vous avez compilé en RISC-V.

3. Syntaxe supportée

IFCC prend en charge un sous-ensemble de la syntaxe C. Voici une description des fonctionnalités et des constructions de langage supportées par notre compilateur :

Déclarations et affectations

- **Déclarations de variables :** Les variables peuvent être déclarées avec le type **int** ou **void**. Elles peuvent être déclarées sur une ligne (séparées par des virgules) et peuvent être initialisées ou non.
- **Affectations :** Les variables peuvent être affectées avec des valeurs ou des expressions arithmétiques.

Expressions

- **Constantes :** Constantes entières positives et les constantes de caractères.
- **Variables :** Les variables peuvent être utilisées dans les expressions. Leur nom commence par une lettre ou un underscore, suivi d'une combinaison de lettres, chiffres et underscores.
- **Opérateurs arithmétiques :** Opérateurs arithmétiques **+**, **-**, *****, **/** et **%**.
- **Opérateurs unaires :** Opérateurs unaire **+** et **-**.
- **Opérateurs de comparaison :** Opérateurs de comparaison **<**, **<=**, **>**, **>=**, **==** et **!=**.
- **Opérateurs logiques :** Opérateurs logiques **&&**, **||** et **!**.

- **Opérateurs de bits :** Opérateurs de bits **&**, **|**, **^**, **<<** et **>>**.
- **Opérateurs d'incrémentation/décrémentation :** Opérateurs d'incrémentation (**++**) et de décrémentation (**--**) pré et postfixé.

Instructions de contrôle de flux

- **Conditions :** Les instructions conditionnelles **if** peuvent être utilisées pour exécuter des blocs de code en fonction d'une condition. L'option **else** est facultative.
- **Switch :** Les instructions **switch** permettent une sélection conditionnelle parmi un ensemble de cas.
- **Boucles :** Les boucles **while** et **do-while** sont prises en charge pour l'itération.
- **Break et Continue :** Les instructions **break** et **continue** peuvent être utilisées pour sortir d'une boucle ou sauter à l'itération suivante, respectivement.

Fonctions

- **Définition de fonctions :** Les fonctions peuvent être définies avec ou sans paramètres et avec ou sans type de retour.
- **Appels de fonctions :** Les fonctions peuvent être appelées avec ou sans arguments.

Instructions de retour

- **Return :** L'instruction **return** permet de renvoyer une valeur depuis une fonction ou de n'importe quel endroit du programme.

Commentaires et directives

- **Commentaires :** Les commentaires (**/*** ... ***/**) sont ignorés par le compilateur.
- **Directives :** Les directives de préprocesseur (commençant par **#**) sont ignorées par le compilateur.

Exemple

Voici un exemple simple d'un programme écrit dans le sous-ensemble de C pris en charge par IFCC :

```
int factorial(int n) {
  if (n <= 1) return 1;
  else return n * factorial(n - 1);
}

int main() {
  int result = factorial(5);
  return result;
}
```

Ce programme calcule le factoriel de 5 en utilisant une fonction récursive factorial et renvoie le résultat depuis la fonction main.

4. Gestion des erreurs

Le compilateur IFCC intègre un mécanisme de gestion des erreurs pour détecter et signaler les problèmes dans le code source soumis à la compilation.

Rapports d'Erreurs

Lorsqu'une erreur est détectée, le compilateur IFCC arrête le processus de compilation et signale l'erreur à l'utilisateur. Un message explicatif est affiché sur la sortie d'erreur standard, indiquant l'emplacement de l'erreur (ligne et position) ainsi qu'une description du problème. De plus, une représentation visuelle de l'emplacement exact de l'erreur est affichée en soulignant le symbole concerné dans le code source.

Types d'Erreurs

IFCC peut détecter divers types d'erreurs, notamment :

- Utilisation de variables non déclarées ou initialisées
- Re-déclaration de variables
- Tentatives d'affectation de valeurs de type incorrect
- Utilisation incorrecte des opérateurs
- Déclarations de variables invalides
- Appels de fonctions incorrects

Un warning est également levé lorsqu'une variable est déclarée mais jamais utilisée.