

PLD-COMPILATEUR - DOCUMENTATION POUR UTILISATEURS

Hexanôme 4244
ALAMI Meryem
AL ZAHABI Hanaa
BELAHCEN Basma
CHELLAOUI Adam
GREVAUD Paul
GUILLEVIC Marie
M'BARECK Aichetou
PREVOT Jade
WAQIF Salma

10 avril 2022



Table des matières

1	Utilisation du compilateur	3
2	Fonctionnalités implémentées	4
2.1	Déclarations et types	4
2.2	Expressions	4
2.2.1	Expressions arithmétiques	4
2.2.2	Autres expressions	4
2.3	Tableaux	5
2.4	Fonctions	5
2.4.1	Fonctions c	5
2.5	Les conditions	6
2.6	Les returns	6
2.7	Gestion des erreurs et warnings	7
3	Fonctionnalités non implémentées	7

Table des figures

1	Exemple d'une affectation avec expression arithmétique ($c = 0$)	4
2	Exemple d'une déclaration et utilisation d'une fonction "test"	5
3	Exemple de deux codes avec if else et while	6
4	Exemple d'un code avec plusieurs returns	6

1 Utilisation du compilateur

Pour utiliser notre compilateur ifcc, vous aurez besoin d'un environnement Linux et un processeur Intel comme nous utilisons le langage assembleur x86.

Il faut également avoir une installation Antlr4 valide sur votre ordinateur qui dépend de l'environnement sur lequel vous travaillez :

- Sur VMWare Horizon avec Linux-2D : vous n'avez pas besoin de faire grand chose, tout est installé. Pour utiliser le compilateur, exécuter la commande `make` ou le script suivant : `./compiler/runmake-fedora.sh`.
- Sur MacOS : utilisez le script `./install-antlr.sh`. Pour compiler, utilisez la commande `make` dans le dossier `/compiler`.
- Sur Ubuntu ou WSL/Ubuntu : installez les paquets indiqués dans le début du fichier nommé `./compiler/runmake-ubuntu.sh`. Utilisez la commande `./compiler/runmake-ubuntu.sh` pour compiler. Attention sous WSL, pour pouvoir utiliser la fonctionnalité d'affichage de l'arbre de dérivation, il faudra installer le paquet GWSL de Microsoft Store.
- Autre distribution Linux : regardez si la distribution offre des paquets pour Antlr4 et Antlr4-runtime, si c'est le cas vous éviterez de tout recompiler. Par exemple sous Fedora, il s'agit des paquets `antlr4`, `antlr4-cpp-runtime` et `antlr4-cpp-runtime-devel`. Sinon lancez le script `./install-antlr.sh`.

Pour les tests, rendez vous dans le dossier `/testfiles`. Vous pourrez exécuter les tests avec les commandes suivantes :

- Sur VMWare Horizon avec Linux-2D : `./ifcc-test.py testfiles/`
- Sur WSL/Ubuntu : `python3 ifcc-test.py testfiles/`

Si les tests ne sont pas corrects, essayez la commande suivante : `"dos2unix ifcc-wrapper.sh"`. Vous devrez peut-être installer `dos2unix` avec la commande `"sudo apt dos2unix"`.

Nos tests sont répartis en 4 dossiers différents :

- `test-code-correct` : Ce sont les tests qui génèrent l'assembleur pour des programmes valides.
- `test-comportement-compileur` : ce sont les tests dont le comportement est indéfini.
- `test-erreurs` : ce sont les tests qui produisent des erreurs du fait que le résultat de l'assembleur généré par le compilateur ne correspond pas au résultat de l'assembleur généré par `gcc`.
- `test-pas-implemente` : Ce sont des classes qui testent des fonctionnalités que nous n'avons pas implémenté essentiellement par manque de temps.

2 Fonctionnalités implémentées

Nous avons implémenté plusieurs fonctionnalités.

2.1 Déclarations et types

Pour la gestion des variables, vous pourrez utiliser les types : **int** et **char**. De plus, notre grammaire nous permet de déclarer en une ligne plusieurs variables les unes à la suite des autres (ex `int a, b;`) et nous gérons l'inférence de types entre **int** et **char** : par exemple si nous additionons deux **char** cela revient à additionner leur code ASCII.

Vous pouvez également déclarer et affecter des valeurs à vos variables sur la même ligne : `int a = 4;`

2.2 Expressions

Vous pouvez affecter à une variable une expression qui peut prendre plusieurs formes (une constante, une autre variable etc...).

2.2.1 Expressions arithmétiques

Les expressions implémentées concernent les additions, soustractions, multiplications et divisions tout en faisant attention à la priorité des opérations. Elles incluent aussi la priorité des parenthèses.

```
int main(){
    int a=5, b=7;
    int c = (a+b*2)/(a*5+3);
    return c;
}
```

FIGURE 1 – Exemple d'une affectation avec expression arithmétique ($c = 0$)

2.2.2 Autres expressions

Les opérations logiques bit à bit, de comparaison et opérations unaires sont également implémentées dans les expressions avec respect des priorités.

- Logique bit à bit (`&`, `|` et `^`) : `b = a | 0`.
- Comparaisons (`==`, `!=`, `<`, `>`) : `b = a==5`.
- Unaires (`-` et `!`) : `a = -b`.

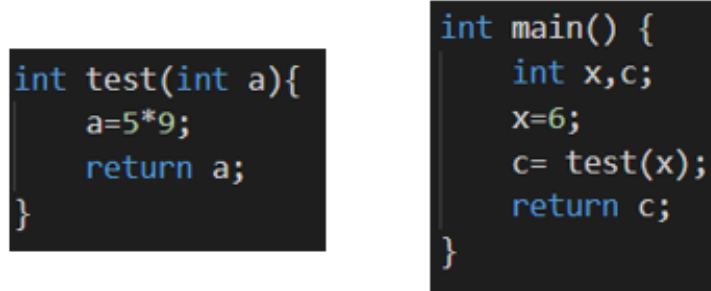
2.3 Tableaux

La compilation de tableau dans le code est aussi possible :

- Déclaration : `int b [5]`.
- Affectations : `b[i] = 8`.
- Expressions : `c = b[2]`.

2.4 Fonctions

Vous pourrez également effectuer des appels et des déclarations de fonctions ayant au plus 6 arguments.



```
int test(int a){  
    a=5*9;  
    return a;  
}  
  
int main() {  
    int x,c;  
    x=6;  
    c= test(x);  
    return c;  
}
```

FIGURE 2 – Exemple d’une déclaration et utilisation d’une fonction "test"

Les types de retour d’une fonction que vous pourrez utiliser sont les `int`, les `char` et le `void`.

Vous pouvez seulement passer des `int` et des `char` en paramètre de votre fonction. Passer une expression arithmétique, logique ou de comparaison est possible : `a = test(5*5+6)`, `b=test(5==4)`.

2.4.1 Fonctions c

Vous pouvez utiliser dans votre code les fonctions `putchar` et `getchar` (ex : `putchar('a')`, `getchar()`).

2.5 Les conditions

Notre compilateur gère également les boucles if..else et les boucles while. Ci dessous deux exemples de codes compilables :

```
int main() {  
    int a =3,b=0;  
    if (a == 2) {  
        b = 1;  
    }else {  
        b = 2;  
    }  
    return b;  
}
```

```
int main(){  
    int a = 5;  
    while(a>60){  
        a = a*2;  
    }  
    return a;  
}
```

FIGURE 3 – Exemple de deux codes avec if else et while

Notre compilateur ne permet pas d'implémenter un if sans else.

2.6 Les returns

Vous pourrez effectuer vos return où bon vous semble.

```
int main(){  
    int a = 6;  
    int c = a*2 + 5;  
    if (c<7){  
        return c;  
    }else{  
        return c+1;  
    }  
    return c;  
}
```

FIGURE 4 – Exemple d'un code avec plusieurs returns

2.7 Gestion des erreurs et warnings

Nous avons également inclus la gestion de plusieurs erreurs et de warning. Quelques exemples sont cités ci-dessous :

- Si une variable est déclarée 2 fois ou plus : il y a une erreur.
- Si une variable est utilisée sans être déclarée : il y a une erreur.
- Si une variable est initialisée à une valeur trop grande pour 64 bits : il y a un warning.
- Si une variable est déclarée mais jamais utilisée : il y a un warning.

Le principe s'applique également à la déclaration de fonctions.

3 Fonctionnalités non implémentées

Malheureusement, vous ne pourrez pas implémenter toutes sortes de code c comme notre compilateur ne propose pas certaines fonctionnalités.

- Les opérateurs d'affectation `+=`, `-=` etc., d'incrémentation `++` et décrémentation `--`
- Le modulo
- Les pointeurs
- Les `break` et `continue`
- Les `switch...case`
- Les opérateurs logiques paresseux `||`, `&&`
- Le Recyclage vers plusieurs architectures : x86, MSP430, ARM
- La propagation de constantes simple et de variables constantes (avec analyse du data-flow)...