



PLD Compilateur

Structure du code

Extrait de la grammaire:

```
grammar ifcc;

axiom: prog;

prog: 'int' 'main' '(' ')' '{' expr+ '}';

expr: ret | decl | assign | inlineArithmetic;

ret: 'return' rval ';';

decl: type sdecl (',' sdecl)* ';';
sdecl: ID ('=' rval)?;

assign: sassign (',' sassign)* ';';
sassign: ID '=' rval;
rval: arithmetic | ID | CONST;

type: 'int';

inlineArithmetic: arithmetic ';';
arithmetic: arithmetic ('*' | '/') arithmetic #muldiv
           | arithmetic ('+' | '-') arithmetic #addminus
```

```

| '(' arithmetic ')' #par
| CONST #const
| '-' arithmetic #moinsunaire
;

RETURN: 'return';
ID: [a-zA-Z_][a-zA-Z0-9_]*;
CONST: '-'? [0-9]+;
COMMENT: '/*' .*? '*/' -> skip;
COMMENT_LINE: '//' .*? '\n' -> skip;
DIRECTIVE: '#' .*? '\n' -> skip;
WS: [ \t\r\n] -> channel(HIDDEN);

```

Explications de la grammaire

- Toujours un axiom
- Programme

```
prog: 'int' 'main' '(' ')' '{' expr+ '}' ;
```

Car programme de la forme:

```
int main(){
    ...
    return 0;
}
```

- expressions:

```
expr: ret | decl | assign | inlineArithmetic;
```

Car un expr est: soit un return, soit une declaration, soit un assignment, soit de l'arithmetique (pour le moment)

- Apres pour le reste ca decrit juste de plus en plus en détail ce qui se passe jusqu'à aller à des symbole terminaux (en Majuscule)

Code C++

Pour le moment on a que deux classes importantes: (en plus on a commenté)

1. CodeGenVisitor: c'est la ou on va mettre nos visiteurs pour y générer le code assembleur
2. SymboleTable: c'est comme un utilitaire pour l'instant qui permet de stocker les positions, utilisations, et les offset en mémoire de chaque variables