Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE)
INE5426 - Construção de Compiladores

Freedom-- :
Apresentação da linguagem e Análise Léxica

16104059 - Bruno Izaias Bonotto
16100725 - Fabíola Maria Kretzer
16105151 - João Vicente Souto

Florianópolis
2018

1. Descrição da Linguagem

A linguagem *Freedom--* (*Freedom Less Less* - Liberdade Menos Menos) é inspirada nas linguagens c e c++. No entanto, possui posicionamento pré-estabelecido do código, de modo que force o desenvolvedor seguir as regras de padronização de código. Deve-se a essa característica de "engessamento" que definiu-se o nome da linguagem.

Esta linguagem possui operações aritméticas, chamada de funções, definição de classes, estruturas if then else, while, for e switch case, não podendo definir classes e funções sem a sua respectiva implementação. Como as linguagens base, também é fortemente tipada, possuindo como tipos primitivos: int, double, float, short, char, bool e void.

Freedom-- oferece uma abordagem fraca de orientação a objetos, permitindo apenas a criação de classes (*struct* em c) contendo métodos e atributos públicos e (ou) privados. Entretanto, por motivos de simplificação *Freedom--* não possui polimorfismo de tipo, logo, não oferece herança.

2. Especificação Formal

a. Gramática

```
grammar FreedomLessLess;

file: import_def? (class_def | function_def)* mainFunction ;

import_def: (IMPORT STRING)+ ;

class_def: CLASS ID OPEN_KEY classMembers CLOSE_KEY ;

classMembers:  public_def private_def? | private_def ;

public_def: PUBLIC class_scope_def ;

private_def: PRIVATE class_scope_def ;

class_scope_def: attribute_def* function_def* ;

attribute_def: att_def SEMICOLON ;

att_def:
```

type_def ID (OPEN_BRAK INTEGER CLOSE_BRAK)* (ASSIGN valued_exp_def)? (COMMA ID (OPEN_BRAK INTEGER CLOSE_BRAK)* (ASSIGN valued_exp_def)?)* |
CLASS ID ID (OPEN_BRAK INTEGER CLOSE_BRAK)* (ASSIGN valued_exp_def)? (COMMA ID (OPEN_BRAK INTEGER CLOSE_BRAK)* (ASSIGN valued_exp_def)?)* ;

valued_exp_def:
value_def | funcCall_def | valued_exp_def (logical_op | arithmetic_op) valued_exp_def |
ID (((ASSIGN | auto_assign_op) valued_exp_def) | auto_increm_op | OPEN_BRAK INTEGER CLOSE_BRAK)? ;

funcCall_def: (ID '.')? ID OPEN_PAR arg_def CLOSE_PAR ('.' ID OPEN_PAR arg_def CLOSE_PAR)* ;

arg_def: valued_exp_def (COMMA valued_exp_def)* ;

function_def: type_def ID OPEN_PAR param_def? CLOSE_PAR block_def ;

param_def:
type_def ID (OPEN_BRAK CLOSE_BRAK)* (COMMA param_def)? |
CLASS ID ID (OPEN_BRAK CLOSE_BRAK)* (COMMA param_def)? ;

block_def: OPEN_KEY (valueless_exp_def SEMICOLON | struct_def)* CLOSE_KEY ;

valueless_exp_def:
ID (((ASSIGN | auto_assign_op) valued_exp_def) | auto_increm_op) |
funcCall_def | att_def | RETURN valued_exp_def | BREAK | CONTINUE ;

struct_def: if_def | for_def | while_def | switch_def ;

if_def: IF OPEN_PAR valued_exp_def CLOSE_PAR block_def (ELSE block_def)? ;

for_def:
FOR OPEN_PAR att_valued_def (COMMA att_valued_def)* SEMICOLON valued_exp_def SEMICOLON valued_exp_def* CLOSE_PAR block_def ;

att_valued_def:
type_def ID (OPEN_BRAK INTEGER CLOSE_BRAK)* ASSIGN valued_exp_def |
CLASS ID ID (OPEN_BRAK INTEGER CLOSE_BRAK)* ASSIGN valued_exp_def ;

while_def:
WHILE OPEN_PAR valued_exp_def CLOSE_PAR block_def ;

switch_def:
SWITCH OPEN_PAR valued_exp_def CLOSE_PAR OPEN_KEY switch_case_def* switch_default_def CLOSE_KEY ;

switch_case_def:
CASE value_def TWO POINTS (valueless_exp_def SEMICOLON | struct_def)+ BREAK SEMICOLON;

switch_default_def:
DEFAULT TWOPOINTS (valueless_exp_def SEMICOLON | struct_def)* BREAK

```
        SEMICOLON;

mainFunction:
        VOID_T MAIN OPEN_PAR INT_T ID COMMA CHAR_T OPEN_BRAK CLOSE_BRAK
        OPEN_BRAK CLOSE_BRAK ID CLOSE_PAR block_def ;

type_def: INT_T | UNSIGNED_T | SHORT_T | FLOAT_T | DOUBLE_T | CHAR_T | BOOL_T | VOID_T ;

value_def: STRING | INTEGER | FLOATING | BOOLEAN | NULL ;

logical_op: LESS | BIGGER | LESS_EQ | BIGGER_EQ | EQUALS | NOT_EQUALS | AND | OR ;

arithmetic_op: PLUS | MINUS | MULT | DIV ;

auto_assign_op: AUTOPLUS | AUTOMINUS | AUTOMULT | AUTODIV ;

auto_increm_op: INCREM | DECREM ;
```

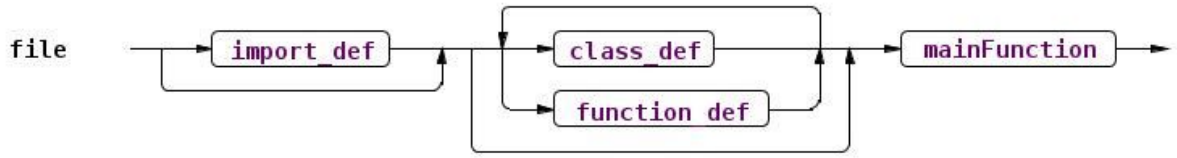
b. Tokens

```
INT_T           : 'int' ;
UNSIGNED_T      : 'unsigned' ;
FLOAT_T         : 'float' ;
DOUBLE_T        : 'double' ;
SHORT_T         : 'short' ;
CHAR_T          : 'char' ;
BOOL_T          : 'bool' ;
VOID_T          : 'void' ;

IMPORT          : 'import' ;
CLASS           : 'class' ;
PUBLIC          : 'public' TWO_POINTS ;
PRIVATE         : 'private' TWO_POINTS ;
MAIN            : 'main' ;

IF              : 'if ';
ELSE            : 'else' ;
FOR             : 'for' ;
WHILE           : 'while' ;
SWITCH          : 'switch' ;
CASE            : 'case' ;
BREAK           : 'break' ;
CONTINUE        : 'continue' ;
DEFAULT         : 'default' ;
RETURN          : 'return' ;

ASSIGN          : '=' ;
PLUS            : '+' ;
MINUS           : '-' ;
MULT            : '*' ;
DIV             : '/' ;
```

```
INCREM          : '++' ;
DECREM          : '--' ;

AUTOPLUS        : '+=' ;
AUTOMINUS       : '-=' ;
AUTOMULT        : '*=' ;
AUTODIV         : '/=' ;

LESS            : '<' ;
BIGGER          : '>' ;
LESS_EQ         : '<=' ;
BIGGER_EQ       : '>=' ;
EQUALS          : '==' ;
NOT_EQUALS      : '!=' ;
AND             : '&&' ;
OR              : '||' ;

OPEN_PAR        : '(' ;
CLOSE_PAR       : ')' ;
OPEN_KEY        : '{' ;
CLOSE_KEY       : '}' ;
OPEN_BRAK       : '[' ;
CLOSE_BRAK      : ']' ;
COMMA           : ',' ;
SEMICOLON       : ';' ;
TWOPOINTS       : ':' ;

NULL            : 'null' ;
BOOLEAN         : 'true' | 'false' ;
STRING          : '"' ( ESC | ~ ["\\])* '"' | '\'' ( ESC | ~ ["\\])* '\'' ;
ID              : [_A-Za-z] [_0-9A-Za-z]* ;

INTEGER         : '-'? INT ;
FLOATING        : '-'? INT '.' [0-9]+ ;

WS              : [ \t\n\r]+ -> channel(HIDDEN) ;
COMMENT         : '/*' .*? '*/' -> channel(HIDDEN) ;
LINE_COMMENT : '//' ~('\r' | '\n')* -> channel(HIDDEN) ;

fragment INT    : '0' | [1-9] [0-9]* ;
fragment ESC    : '\\' (["\\/bfnrt]) ;
```

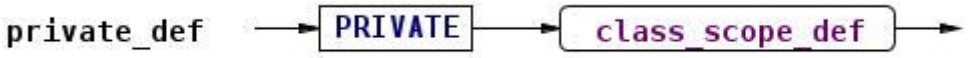
c. Grafos EBNF

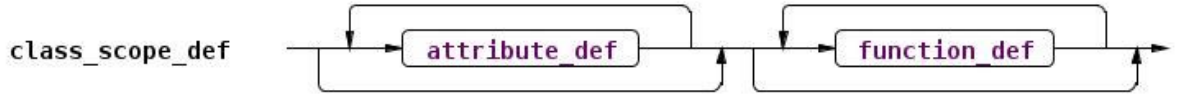
i. Estrutura:



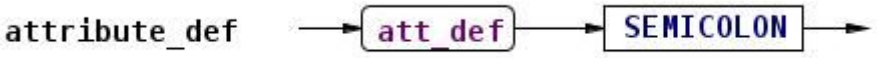
ii. Importação de arquivos externos:

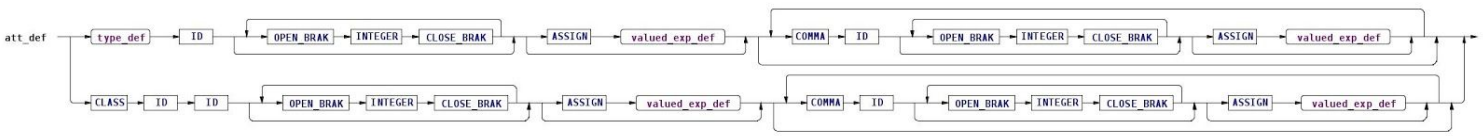
iii. Classes:



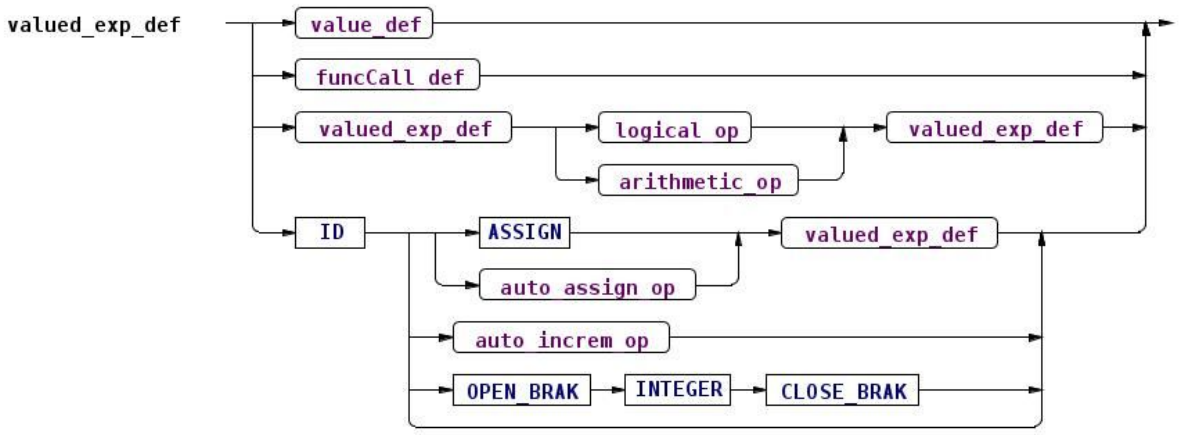


iv. Atributos:

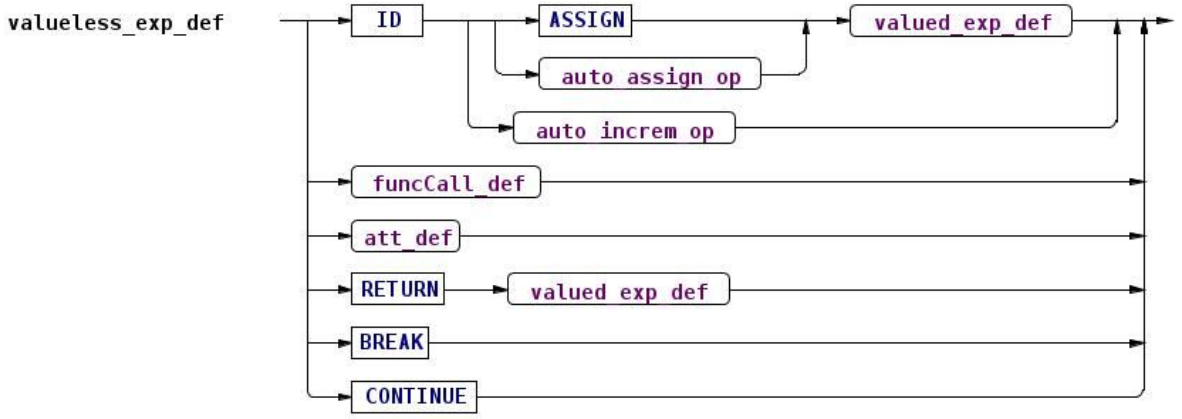




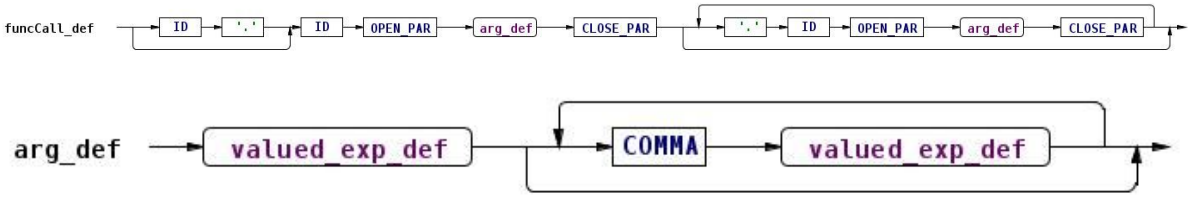
v. Expressão valorada:



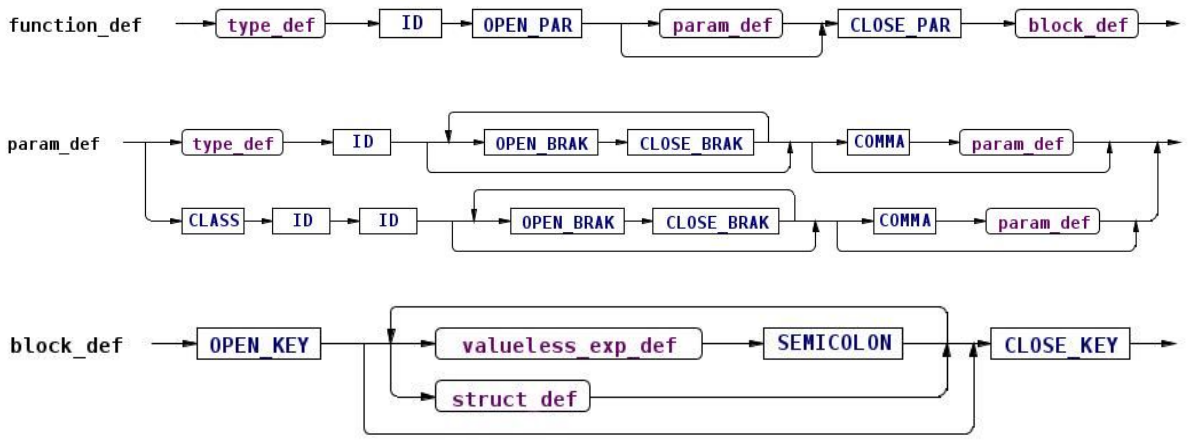
vi. Expressão não valorada:



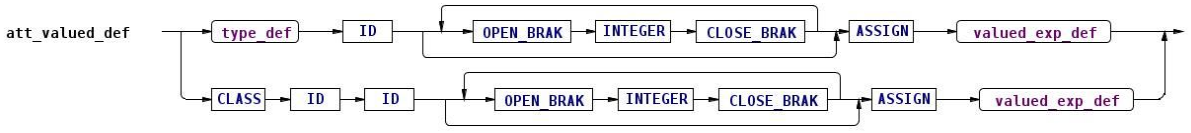
vii. Chamada de função:



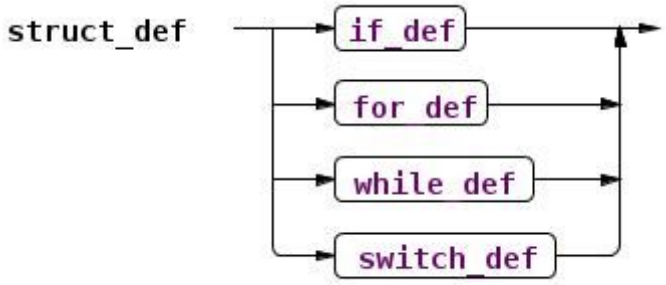
viii. Funções:

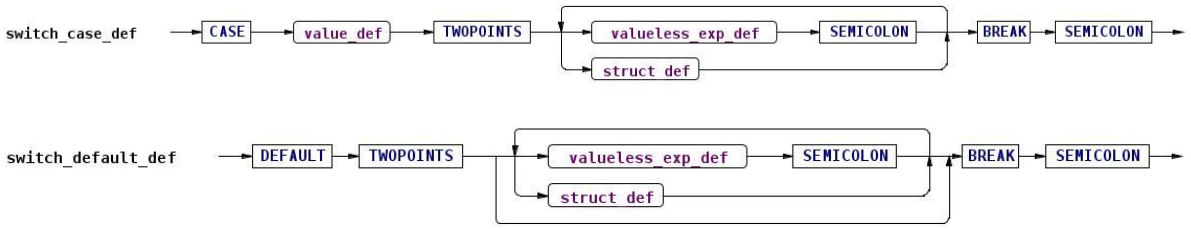


ix. Atributo valorado:



x. Estruturas de seleção e repetição:





xi. Função principal:

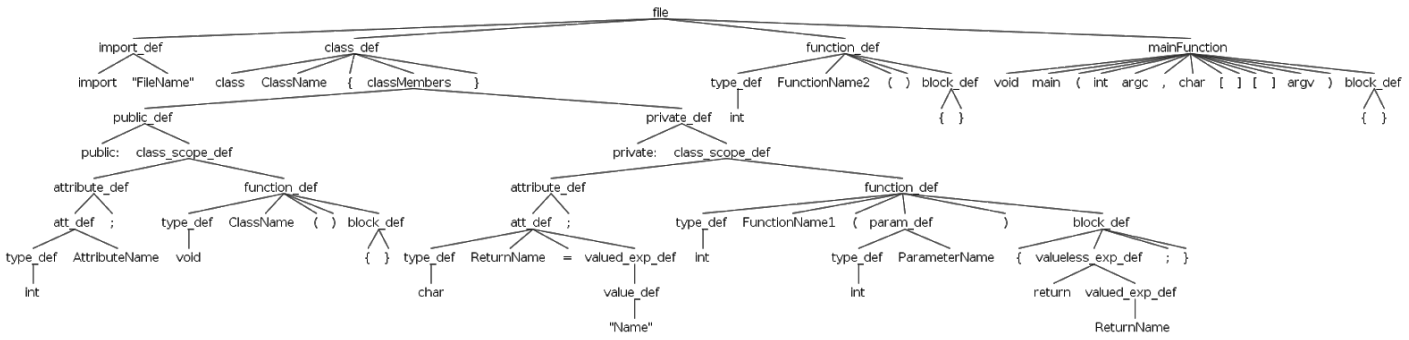
3. Analisador Léxico

Para realizar a análise léxica da linguagem descrita foi utilizada a ferramenta ANTLR 4, uma vez que é possível gerar automaticamente os analisadores léxico e sintático.

4. Exemplos

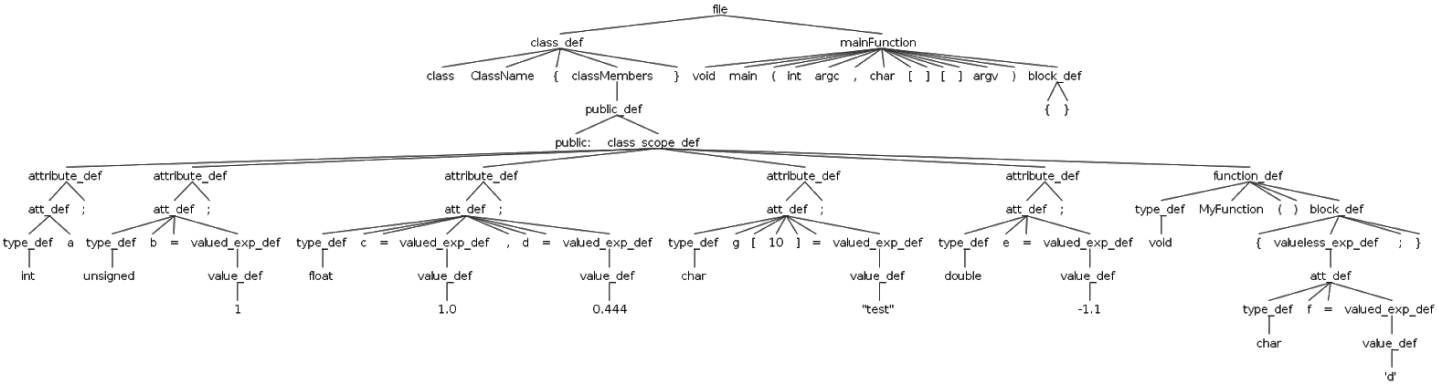
a. Estrutura

```
import "FileName"
class ClassName {
        public:
                int AttributeName;
                void ClassName () {}
        private:
                char ReturnName = "Name";
                int FunctionName1 (int ParameterName) { return ReturnName; }
}
int FunctionName2 () {}
void main (int argc, char[][] argv) {}
```



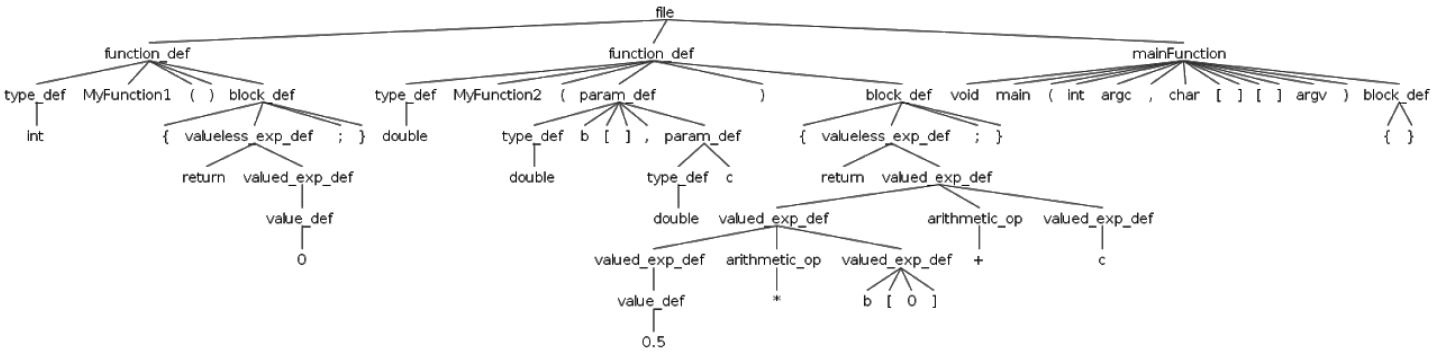
b. Declaração de variáveis e atributos

```
class ClassName {
        public:
                int a;
                unsigned b = 1;
                float c = 1.0, d = 0.444;
                char g[10] = "test";
                double e = -1.1;

                void MyFunction () { char f = 'd'; }
}
void main (int argc, char[][] argv) {}
```



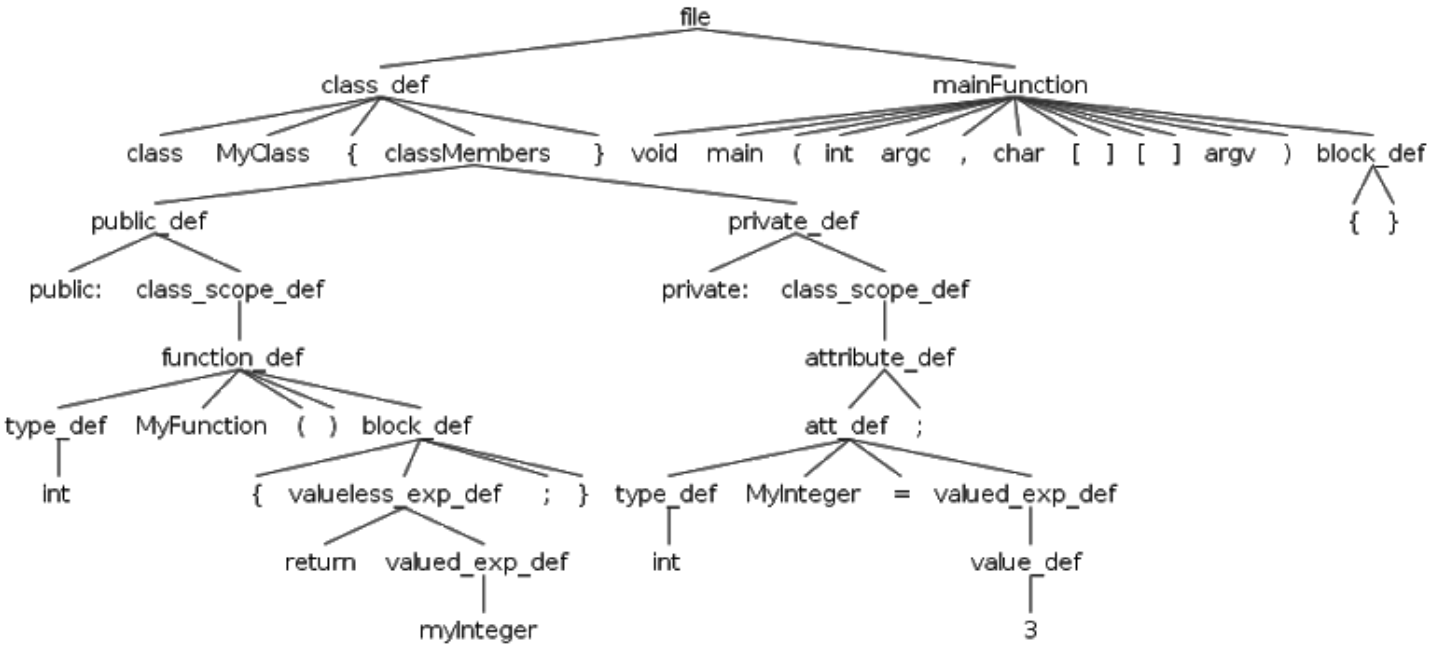
c. Declaração de funções

```
int MyFunction1 () { return 0; }
double MyFunction2 (double b[], double c) { return 0.5 * b[0] + c; }
void main (int argc, char[][] argv) {}
```



d. Definição de classes

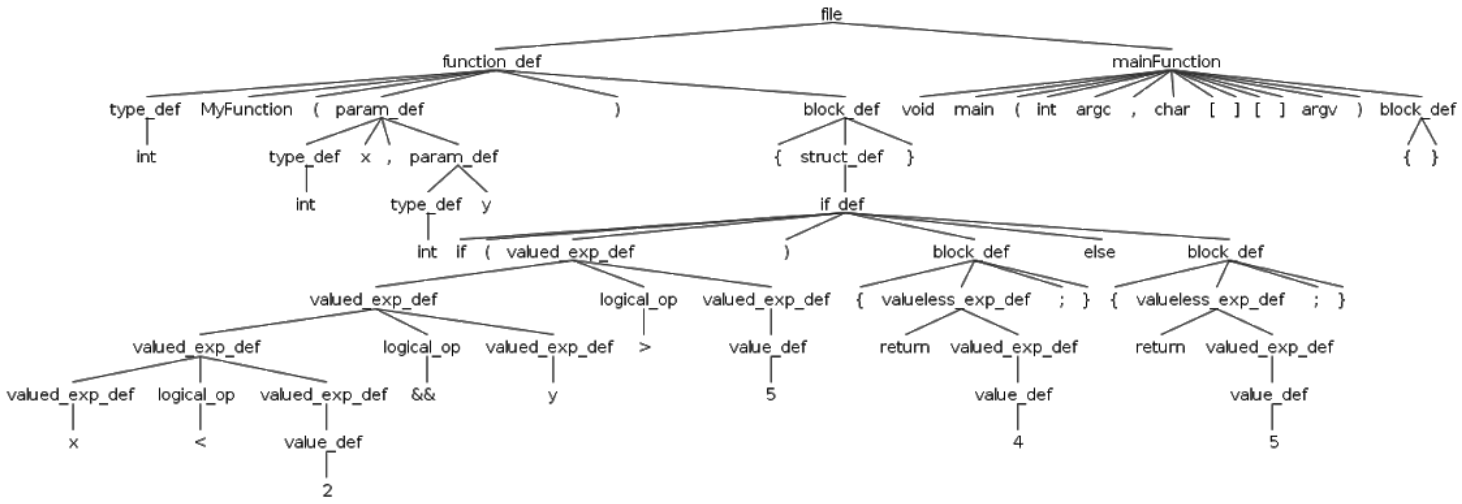
```
class MyClass {
        public:
                int MyFunction () { return myInteger; }
        private:
                int MyInteger = 3;
}
void main (int argc, char[][] argv) {}
```



e. Estrutura de seleção if

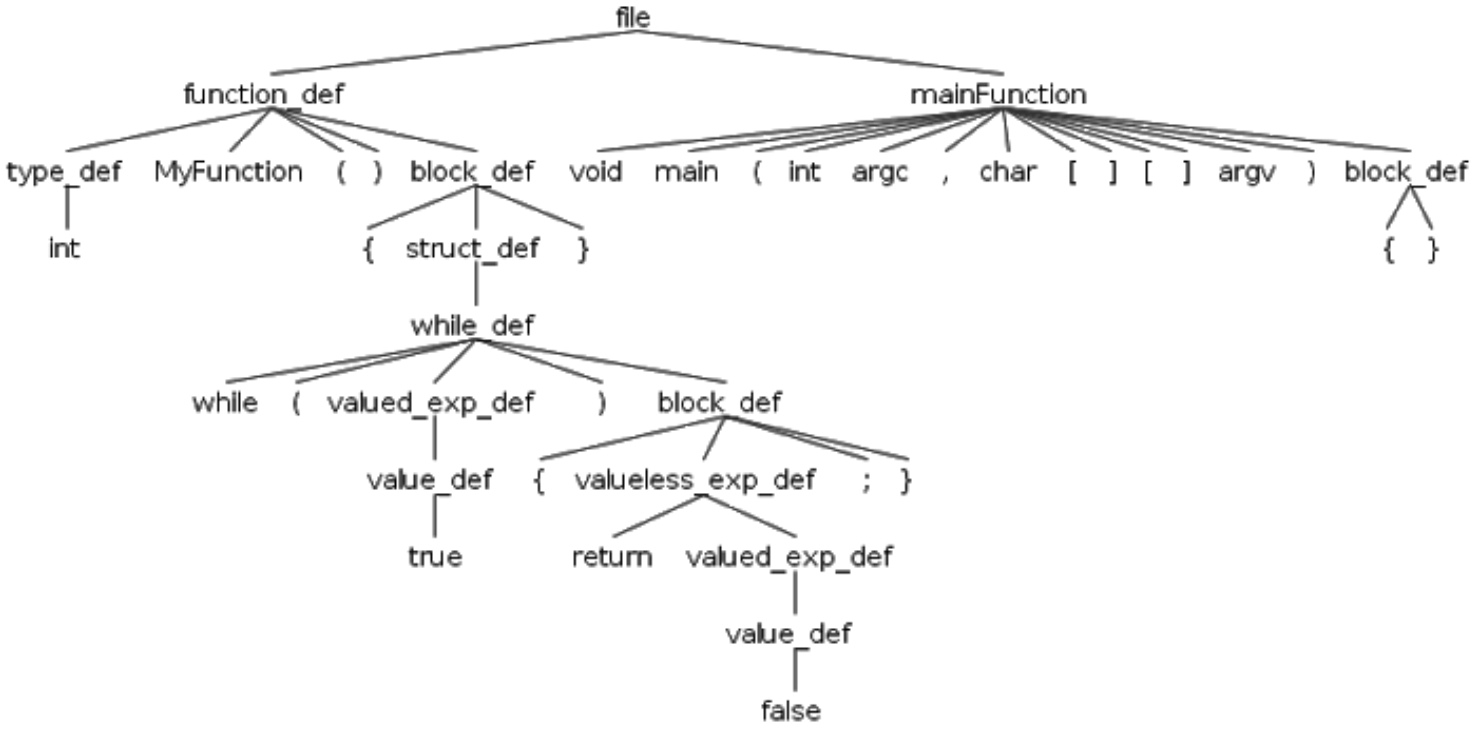
```
int MyFunction (int x, int y) {
        if (x < 2 && y > 5) {
                return 4;
        } else {
                return 5;
```

```
        }
}
void main (int argc, char[][] argv) {}
```



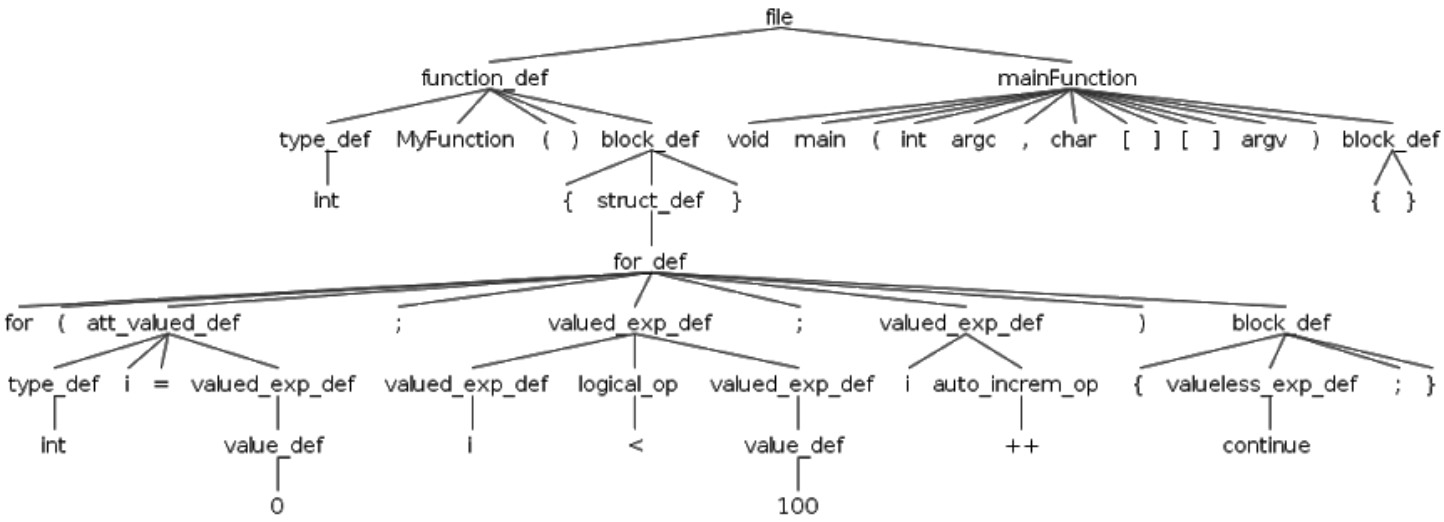
f. Estrutura de seleção while

```
int MyFunction () {
        while (true) { return false; }
}
void main (int argc, char[][] argv) {}
```



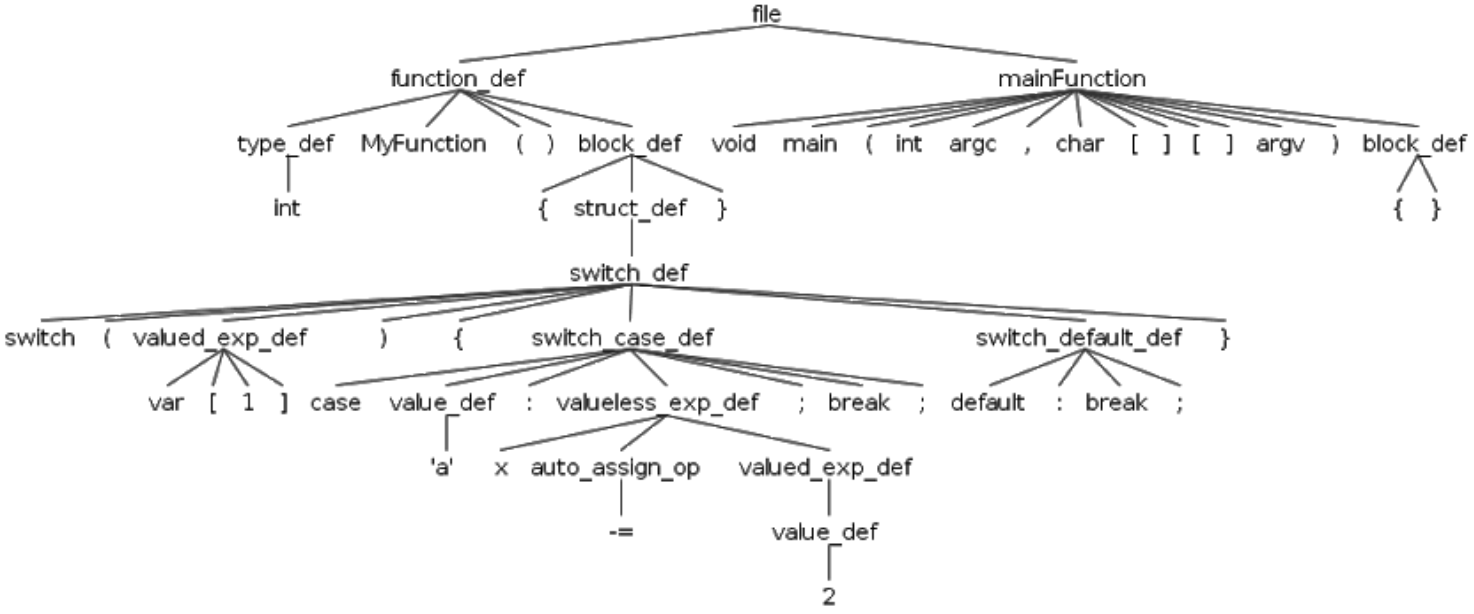
g. Estrutura de seleção for

```
int MyFunction () {
        for (int i = 0; i < 100; i++) { continue; }
}
void main (int argc, char[][] argv) {}
```



h. Estrutura de seleção switch case

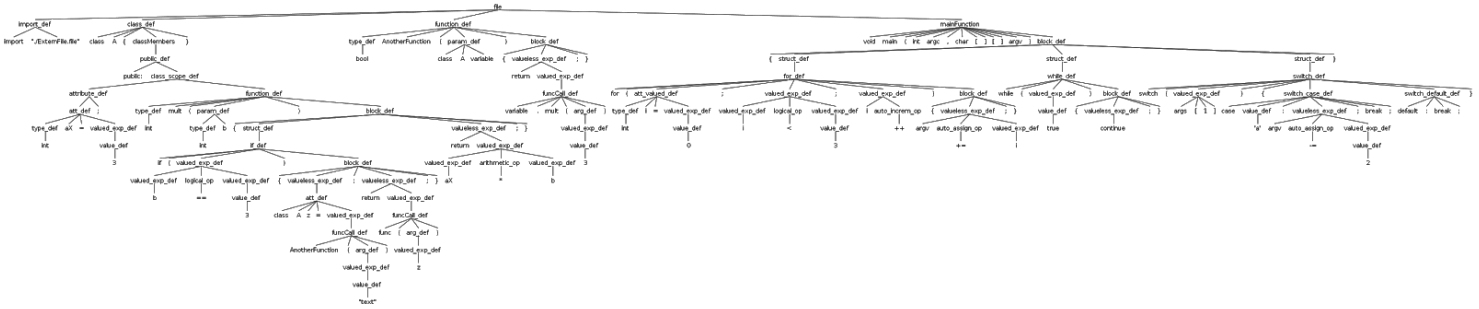
```
int MyFunction () {
        switch (var[1]) {
                case 'a': x -= 2;
                        break;
                default:
                        break;
        }
}
void main (int argc, char[][] argv) {}
```



i. Programa completo

```
import "./ExternFile.file"
class A {
	public:
		int aX = 3;
		int mult (int b) {
			if (b == 3) {
				class A z = AnotherFunction("text");
				return func(z);
			}
			return aX * b;
		}
}
bool AnotherFunction (class A variable) { return variable.mult(3); }
void main(int argc, char[][] argv) {
	for (int i = 0; i < 3; i++) { argv += i; }
	while (true) { continue; }
	switch (args[1]) {
		case 'a': argv -= 2;
			break;
		default:
			break;
	}
```

}


file
import_def
import "./ExternFile.flw"
class_def
class A { classMembers }
public_def
public: class_scope_def
attribute_def
stt_def ;
type_def aX = valued_exp_def
int
value_def
3
function_def
type_def mult (params_def)
int
if_def
if (valued_exp_def)
valued_exp_def logical_op valued_exp_def
b == value_def
3
block_def
{ valueless_exp_def ; valueless_exp_def ; }
stt_def
class A z = valued_exp_def
funcCall_def
funcCall_def func (arg_def)
AnotherFunction (arg_def) valued_exp_def
valued_exp_def
value_def
"text"
function_def
type_def AnotherFunction (params_def)
bool
block_def
class A variable { valueless_exp_def ; }
return valued_exp_def
funcCall_def
variable . mult (arg_def)
type_def b { struct_def
valueless_exp_def ; }
int
return valued_exp_def
valued_exp_def arithmetic_op valued_exp_def 3
aX * b
value_def
3
mainFunction
void main (int argc , char [ ] [ ] argv) block_def
{ struct_def
struct_def
C
struct_def }
for_def
for (stt_valued_def ; valued_exp_def ; valued_exp_def) block_def
type_def I = valued_exp_def
int
value_def
0
valued_exp_def logical_op valued_exp_def
I < value_def
3
auto_increm_op
++ argv
while_def
while (valued_exp_def) block_def
{ valueless_exp_def ; }
auto_assign_op valued_exp_def
+= 1
value_def
true
block_def switch (valued_exp_def) { switch_def }
{ valueless_exp_def ; }
continue
args [1]
switch_def
switch_case_def switch_default_def
case value_def : valueless_exp_def ; break ; default : break ;
'a' argv auto_assign_op valued_exp_def
-= value_def
2