Engenharia Gramatical (1º ano de MEI)

Grafos na análise e interpretação de código fonte

Relatório de Desenvolvimento

André Martins
PG47009

José Ferreira
PG47375

30 de maio de 2022

Resumo

Este relatório é relativo ao terceiro Trabalho Prático da Unidade Curricular de Engenharia Gramatical.

Conteúdo

Capítulo 1

Introdução

Supervisor: Pedro Rangel Henriques

1.1 Grafos na análise e interpretação de código fonte

Área: Engenharia Gramatical

Neste segundo trabalho prático da unidade curricular de Engenharia Gramatical foi-nos proposto o desenvolvimento de uma ferramenta capaz de gerar grafos de controlo de fluxo - doravante chamados de CFG - e grafos de dependências do sistema "lite- doravante chamados de SDG -, (não consideram as dependências de dados). Como requisito opcional, a ferramenta deve ser capaz de detetar zonas de código inalcansável e calcular a Complexidade de McCabe para um excerto de código, a partir dos grafos gerados para esses mesmo código. Os excertos de código analisados são escritos na linguagem definida aquando da realização do segundo trabalho prático da UC (gramática definida adiante), que suporta a declaração de variáveis atómicas (int, float e string) e estruturais (set, list, tuple e dict), a atribuição de valores a variáveis, operações de leitura e escrita, estruturas condicionais (if/else) e estruturas cíclicas (while, repeat e for).

Reutilizamos os ficheiros do segundo trabalho prático da UC uma vez que, desse modo, temos a gramática definida e todas as suas funcionalidades implementadas, uma classe que, a partir do módulo de geração de processadores de linguagens **Lark.Interpreter**, gera como resultado a informação necessária para construir um relatório sobre o código analisado, assim como sugestões, avisos e erros encontrados durante esta primeira análise.
Adicionamos, de seguida, uma classe que recorre novamente ao **Lark.Interpreter**, mas desta vez com o objetivo de gerar, a partir do código analisado, os grafos para análise do código fonte (CFG e SDG), assim como as informações acima referidas, a retirar dos mesmos.

Cumpridos todos estes pressupostos, o programa foi capaz de analisar todos os elementos da linguagem e de gerar toda a informação pretendida, sendo apresentado, de seguida, vários exemplos e a própria gramática da linguagem utilizada.

Capítulo 2

Definição da gramática

Uma vez que o objetivo deste trabalho prático era a geração de grafos e de informação a partir dos mesmos, e dado que a gramática definida é idêntica à utilizada no segundo trabalho prático, apresentamos neste segmento a gramática completa, juntamente com um exemplo de código escrito na linguagem definida.

Gramática Completa

```
start: BEGIN program END
program: instruction+
instruction: declaration | comment | operation
declaration: atomic | structure
operation: atrib | print | read | cond | cicle
print: "print" PE (VARNAME | ESCAPED_STRING) PD PV
read: "read" PE VARNAME PD PV
cond: IF PE op PD body (ELSE body)?
cicle: ciclewhile | ciclefor | ciclerepeat
ciclewhile: WHILE PE op PD body
WHILE: "while"
ciclefor: FOR PE (initcicle (VIR initcicle)*)? PV op PV ((inc | dec) (VIR (inc | dec))*)? PD body
initcicle: VARNAME EQUAL op
FOR: "for"
ciclerepeat: REPEAT PE (SIGNED_INT | VARNAME) PD body
REPEAT: "repeat"
body: open program close
atrib: VARNAME EQUAL op PV
inc: VARNAME INC
INC: "++"
dec: VARNAME DEC
DEC: "--"
op: NOT op | op (AND | OR) factcond | factcond
NOT: "!"
AND: "&"
OR: "#"
factcond: factcond BINSREL expcond | expcond
BINSREL: LESSEQ | LESS | MOREEQ | MORE | EQ | DIFF
LESSEQ: "<="
```

```
LESS: "<"
MOREEQ: ">="
MORE: ">"
EQ: "=="
DIFF: "!="
expcond: expcond (PLUS | MINUS) termocond | termocond
PLUS: "+"
MINUS: "-"
termocond: termocond (MUL|DIV|MOD) factor | factor
MUL: "*"
DIV: "/"
MOD: "%"
factor: PE op PD | SIGNED_INT | VARNAME | DECIMAL
atomic: TYPEATOMIC VARNAME (EQUAL elem)? PV
structure: (set | list | dict | tuple) PV
set: "set" VARNAME (EQUAL OPENBRACKET (elem (VIR elem)*)? CLOSEBRACKET)?
dict: "dict" VARNAME (EQUAL OPENBRACKET (elem DD elem (VIR elem DD elem)*)? CLOSEBRACKET)?
list: "list" VARNAME (EQUAL OPENSQR (elem (VIR elem)*)? CLOSESQR)?
tuple: "tuple" VARNAME (EQUAL PE (elem (VIR elem)*)? PD)?
elem: ESCAPED_STRING | SIGNED_INT | DECIMAL | op
TYPEATOMIC: "int" | "float" | "string"
VARNAME: WORD
comment: C_COMMENT
BEGIN: "-{"
END: "}-"
PV: ";"
VIR: ","
OPENBRACKET: "{"
CLOSEBRACKET: "}"
OPENSQR: "["
CLOSESQR: "]"
DD: ":"
PE: "("
PD: ")"
EQUAL: "="
open: OPEN
OPEN: "{"
close: CLOSE
CLOSE: "}"
IF: "if"
ELSE: "else"


%import common.WORD
%import common.SIGNED_INT
%import common.DECIMAL
%import common.WS
%import common.ESCAPED_STRING
```

```
%import common.C_COMMENT
%ignore WS
```

Exemplo de código

```
-{
    /*int a;
    int b = 2;
    float c;
    float d = 3.4;
    string e;
    string f = "ola";

    set g;
    set h = {};
    set i = {2, 3.4, "ola"};

    list j;
    list k = [];
    list l = [2, 3.4, "ola"];

    tuple m;
    tuple n = ();
    tuple o = (2, 3.4, "ola");

    dict p;
    dict q = {};
    dict r = {1:"ola", 3.2:"mundo"};

    a = 3 + 1;
    e = "mundo";
    c = 4.3 + 3.14;

    print("oi");
    read(a);

    if(a==3){
        a = 5;
        a = 2;
    }

    if(a == 4){
        a = 5;
    } else {
        a = 4;
    }

    while(a < 5){
        a = a + 1;
```

```
        a = 3;
    }


    repeat(5){
        a = 2;
    }
    a = 5;*/

    for(a = 0, b = 3; a < 5; a++,b--){
        b = 4;
        c = 0;
    }
}-
```

Capítulo 3

Interpretador Lark

Recorrendo ao módulo de geração de processadores de linguagens Lark do Python, definimos a nossa ferramenta para analisar a linguagem e gramática previamente apresentada e exemplificada. Utilizando *Lark.Visitors* foi possível analisar toda a nossa gramática e gerar os grafos. Para melhor explicar o funcionamento de toda a análise do código, iremos explicar a estrutura e composição da classe do nosso interpretador. O interpretador definido está estruturado na seguinte forma:

```
self.cfg = "digraph G {\n\t\"entry\" -> "
self.sdg = "digraph G {\n\t"
self.ifID = -1
self.structureID = -1
self.atomicID = -1
self.atribID = -1
self.printID = -1
self.readID = -1
self.whileID = -1
self.repeatID = -1
self.forID = -1
self.initcicleID = -1
self.incID = -1
self.decID = -1
self.output = {}
self.second = False # SDG só lê uma vez cada instrução, ao contrário do CFG
self.incicle = False # previne leitura dupla do body (OPEN program CLOSE)
self.only = False # Há situações em que só queremos escrever no SDG
self.islands = set() # Ilhas de código - Inalcansáveis
self.mccabe = 0 # Cálculo da Complexidade de McCabe
```

Começamos por explicar o início da geração dos grafos. Para o CFG, apenas vamos utilizar a "entry"uma vez como origem de uma aresta, que marca o início do programa. Para o SDG, vamos utilizar "entry"mais do que uma vez, sendo que todas as instruções ao "nível 0"do código são destinos de arestas com origem em "entry".

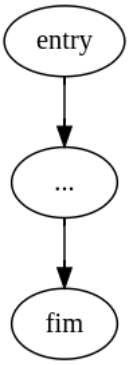


Figura 3.1: Exemplo de CFG

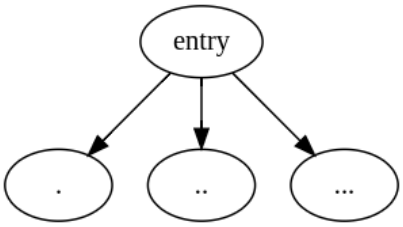


Figura 3.2: Exemplo de SDG

De seguida, definimos um identificador único para cada tipo de operação. Isto porque, no formato dot em que geramos os grafos, se dois nodos tiverem exatamente o mesmo texto, serão tratados como um único nodo. Ao termos este identificador único (que vai incrementando) para cada tipo de operação, garantimos que cada operação é única e cada nodo é, igualmente único.

As variáveis seguintes servem apenas como flags para concatenação de strings aos grafos.

Capítulo 4

Resultados obtidos

Nesta secção serão apresentadas as páginas HTML geradas para o ficheiro de teste em anexo.
Para tornar a interpretação e análise mais organizada e acessível decidimos criar 6 páginas HTML distintas e acessíveis através de si mesmas. A primeira página apresenta todas as estatísticas e informações retiradas acerca do ficheiro analisado. Toda a informação está estruturada em tabelas de maneira clara e objetiva. A segunda página HTML apresenta uma versão do código analisado com destaque para erros e warnings encontrados, aparecendo os mesmos realçados e fáceis de identificar. Assim, qualquer utilizador consegue percorrer o código apresentado e encontrar todas as melhorias e correções que devem ser feitas. A terceira página HTML é focada exclusivamente na identificação e sugestão para o caso particular da existências de *ifs* aninhados. É apresentada uma tabela em que aparece o excerto de código original seguido da respetiva sugestão de alteração. A quarta página apresenta o SDG, a quinta apresenta o CFG, ambas em formato dot. Por último, a sexta página apresenta a complexidade de McCabe, juntamente com a deteção de ilhas de código inalcansável.

4.1 Página HTML de análise de resultados

Variáveis atómicas

Code Analysis | Original Code | Nested If's Suggestions | SDG | CFG | McCabe Complexity and Islands

Variable	Type	Value	Warnings	Errors
a	int	1	No warnings	No errors
b	int	4	No warnings	No errors
c	float	0	No warnings	No errors
d	float	3.4	Variable "d" was never used.	No errors
e	string	mundo	No warnings	No errors
f	string	ola	Variable "f" was never used.	No errors

Variáveis estruturadas e número total de variáveis

Code Analysis | Original Code | Nested If's Suggestions | SDG | CFG | McCabe Complexity and Islands

Tabel with program's structural variables

Variable	Type	Size	Value	Warnings
g	set	0	set()	Variable "g" was never used.
h	set	0	set()	Variable "h" was never used.
i	set	3	{'ola', 2, 3.4}	Variable "i" was never used.
j	list	0	[]	Variable "j" was never used.
k	list	0	[]	Variable "k" was never used.
l	list	3	[2, 3.4, 'ola']	Variable "l" was never used.
m	tuple	0	()	Variable "m" was never used.
n	tuple	0	()	Variable "n" was never used.
o	tuple	3	(2, 3.4, 'ola')	Variable "o" was never used.
p	dict	0	{}	Variable "p" was never used.
q	dict	0	{}	Variable "q" was never used.
r	dict	2	{1: 'ola', 3.2: 'mundo'}	Variable "r" was never used.

Total number of program variables: 18

Tipos de dados estruturados

Structural data types used

Type	Amount
set	3
list	3
tuple	3
dict	3

Número total de instruções

Total amount of instructions

Instruction	Amount
atomic_declaration	6
atrib	18
structure_declaration	12
print	2
read	1
if	4
while	1
repeat	1
for	1
Total	46

Estruturas de controlo

Control Structures

ID	Type	Parents
0	if	No parents
1	if	No parents
2	while	No parents
3	if	No parents
4	if	Parents' IDs: 3 |
5	repeat	No parents
6	for	No parents

4.2 Código original, com destaque de erros

```
-{

        int a;

        int b = 2;

        float c;

        float d = 3.4;

        string e;

        string f = "ola";

        set g;

        set h = {};

        set i = {2, 3.4, "ola"};

        list j;

        list k = [];

        list l = [2, 3.4, "ola"];
```

```
tuple m;

tuple n = ();

tuple o = (2, 3.4, "ola");

dict p;

dict q = {};

dict r = {1 : "ola", 3.2 : "mundo"};

a = 3 + 1;

e = "mundo";

c = 4.3 + 3.14;

print("oi");

read(a);

if(a == 3){
```

```
        a = 5;

        a = 2;

}        if(a == 4){

        a = 5;

} else {


        a = 4;

}        while(a < 5){

        a = a + 1;

        a = 3;

}        if(a == 3){

        if(b == 2){

                print("ola mundo!");
```

```
                print("ola mundo!");

        }        }        repeat(5){

        a = 2;

}        a = 5;

for(a = 0, b = 3; a < 5; a++, b--){

        b = 4;

        c = 0;

}}-
```

4.3 Sugestão de If's

Original	Suggestion

if(a == 3){
  if(b == 2){
  print("ola mundo!");
 	
  }
}
	if((a == 3) & (b == 2)){
	  print("ola mundo!");
	}n

4.4 CFG

Visualizador online

```
digraph G {
"entry" -> "atomic_0 int a"
"atomic_0 int a" -> "atomic_1 int b = 2"
"atomic_1 int b = 2" -> "atomic_2 float c"
"atomic_2 float c" -> "atomic_3 float d = 3.4"
"atomic_3 float d = 3.4" -> "atomic_4 string e"
"atomic_4 string e" -> "atomic_5 string f = 'ola'"
"atomic_5 string f = 'ola'" -> "structure_0 set g"
"structure_0 set g" -> "structure_1 set h = {}"
"structure_1 set h = {}" -> "structure_2 set i = {2, 3.4, 'ola'}"
"structure_2 set i = {2, 3.4, 'ola'}" -> "structure_3 list j"
"structure_3 list j" -> "structure_4 list k = []"
"structure_4 list k = []" -> "structure_5 list l = [2, 3.4, 'ola']"
"structure_5 list l = [2, 3.4, 'ola']" -> "structure_6 tuple m"
"structure_6 tuple m" -> "structure_7 tuple n = ()"
"structure_7 tuple n = ()" -> "structure_8 tuple o = (2, 3.4, 'ola')"
"structure_8 tuple o = (2, 3.4, 'ola')" -> "structure_9 dict p"
"structure_9 dict p" -> "structure_10 dict q = {}"
"structure_10 dict q = {}" -> "structure_11 dict r = {1 : 'ola', 3.2 : 'mundo'}"
"structure_11 dict r = {1 : 'ola', 3.2 : 'mundo'}" -> "atrib_0 a = 3 + 1"
"atrib_0 a = 3 + 1" -> "atrib_1 e = 'mundo'"
"atrib_1 e = 'mundo'" -> "atrib_2 c = 4.3 + 3.14"
"atrib_2 c = 4.3 + 3.14" -> "print_0 print('oi')"
"print_0 print('oi')" -> "read_0 read(a)"
"read_0 read(a)" -> "if_0_start if(a == 3)"
"if_0_start if(a == 3)" -> "atrib_3 a = 5"
"atrib_3 a = 5" -> "atrib_4 a = 2"
"atrib_4 a = 2" -> "if_0_end if(a == 3)"
"if_0_start if(a == 3)" -> "if_0_end if(a == 3)"
"if_0_end if(a == 3)" -> "if_1_start if(a == 4)"
"if_1_start if(a == 4)" -> "atrib_5 a = 5"
"atrib_5 a = 5" -> "if_1_end if(a == 4)"
```

```
"if_1_start if(a == 4)" -> "atrib_6 a = 4"
"atrib_6 a = 4" -> "if_1_end if(a == 4)"
"if_1_end if(a == 4)" -> "while_0_start while(a < 5)"
"while_0_start while(a < 5)" -> "while_0_end while(a < 5)"
"while_0_start while(a < 5)" -> "atrib_7 a = a + 1"
"atrib_7 a = a + 1" -> "atrib_8 a = 3"
"atrib_8 a = 3" -> "while_0_start while(a < 5)"
"while_0_end while(a < 5)" -> "if_2_start if(1)"
"if_2_start if(1)" -> "if_3_start if(2)"
"if_3_start if(2)" -> "print_1 print('ola mundo!')"
"print_1 print('ola mundo!')" -> "if_3_end if(2)"
"if_3_start if(2)" -> "if_3_end if(2)"
"if_3_end if(2)" -> "if_2_end if(1)"
"if_2_start if(1)" -> "if_2_end if(1)"
"if_2_end if(1)" -> "repeat_0_start repeat(5)"
"repeat_0_start repeat(5)" -> "repeat_0_end repeat(5)"
"repeat_0_start repeat(5)" -> "atrib_9 a = 2"
"atrib_9 a = 2" -> "repeat_0_start repeat(5)"
"repeat_0_end repeat(5)" -> "atrib_10 a = 5"
"atrib_10 a = 5" -> "for_0_start"
"for_0_start" -> "initcicle_0 a = 0"
"initcicle_0 a = 0" -> "initcicle_1 b = 3"
"initcicle_1 b = 3" -> "for_0_cond (a < 5)"
"for_0_cond (a < 5)" -> "for_0_end"
"for_0_cond (a < 5)" -> "atrib_11 b = 4"
"atrib_11 b = 4" -> "atrib_12 c = 0"
"atrib_12 c = 0" -> "inc_0 a++"
"inc_0 a++" -> "dec_0 b--"
"dec_0 b--" -> "for_0_cond (a < 5)"
"for_0_end" -> "fim"
"while_0_start while(a < 5)" [shape=diamond]
"if_0_start if(a == 3)" [shape=diamond]
"repeat_0_start repeat(5)" [shape=diamond]
"if_1_start if(a == 4)" [shape=diamond]
"if_3_start if(2)" [shape=diamond]
"for_0_cond (a < 5)" [shape=diamond]
"if_2_start if(1)" [shape=diamond]
}
```

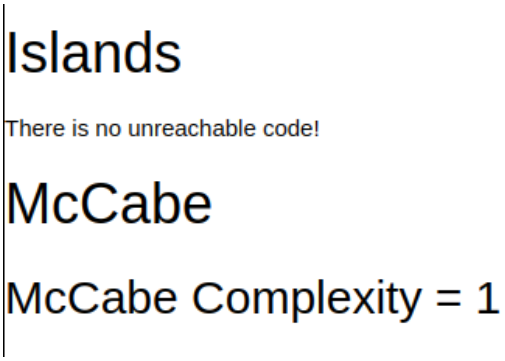
4.5 SDG

Visualizador online

```
digraph G {
"entry" -> "atomic_0 int a"
"entry" -> "atomic_1 int b = 2"
"entry" -> "atomic_2 float c"
"entry" -> "atomic_3 float d = 3.4"
```

```
"entry" -> "atomic_4 string e"
"entry" -> "atomic_5 string f = 'ola'"
"entry" -> "structure_0 set g"
"entry" -> "structure_1 set h = {}"
"entry" -> "structure_2 set i = {2, 3.4, 'ola'}"
"entry" -> "structure_3 list j"
"entry" -> "structure_4 list k = []"
"entry" -> "structure_5 list l = [2, 3.4, 'ola']"
"entry" -> "structure_6 tuple m"
"entry" -> "structure_7 tuple n = ()"
"entry" -> "structure_8 tuple o = (2, 3.4, 'ola')"
"entry" -> "structure_9 dict p"
"entry" -> "structure_10 dict q = {}"
"entry" -> "structure_11 dict r = {1 : 'ola', 3.2 : 'mundo'}"
"entry" -> "atrib_0 a = 3 + 1"
"entry" -> "atrib_1 e = 'mundo'"
"entry" -> "atrib_2 c = 4.3 + 3.14"
"entry" -> "print_0 print('oi')"
"entry" -> "read_0 read(a)"
"entry" -> "if_0 if(a == 3)"
"if_0 if(a == 3)" -> "then0"
"then0" -> "atrib_3 a = 5"
"then0" -> "atrib_4 a = 2"
"entry" -> "if_1 if(a == 4)"
"if_1 if(a == 4)" -> "then1"
"then1" -> "atrib_5 a = 5"
"if_1 if(a == 4)" -> "else1"
"else1" -> "atrib_6 a = 4"
"entry" -> "while_0 while(a < 5)"
"while_0 while(a < 5)" -> "atrib_7 a = a + 1"
"while_0 while(a < 5)" -> "atrib_8 a = 3"
"entry" -> "if_2 if(1)"
"if_2 if(1)" -> "then2"
"then2" -> "if_3 if(2)"
"if_3 if(2)" -> "then3"
"then3" -> "print_1 print('ola mundo!')"
"entry" -> "repeat_0 repeat(5)"
"repeat_0 repeat(5)" -> "atrib_9 a = 2"
"entry" -> "atrib_10 a = 5"
"entry" -> "initcicle_0 a = 0"
"entry" -> "initcicle_1 b = 3"
"entry" -> "for_0 for(a < 5)"
"for_0 for(a < 5)" -> "atrib_11 b = 4"
"for_0 for(a < 5)" -> "atrib_12 c = 0"
"for_0 for(a < 5)" -> "inc_0 a++"
"for_0 for(a < 5)" -> "dec_0 b--"
}
```

4.6 Código inacessível e Complexidade de McCabe



Capítulo 5

Conclusão

Dado por concluído o terceiro trabalho prático da unidade curricular de Engenharia Gramatical, cremos ter alcançados com sucesso todos os objetivos pretendidos e termos obtido todos os resultados que eram esperados. Este trabalho prático permitiu aprofundar e utilizar o módulo Lark num projeto mais extenso e mais complexo, permitindo tirar partido de todas as suas funcionalidades para geração de gráficos para análise de código fonte. A construção do analisador de código foi complexa mas estamos satisfeitos com os resultados finais e com a forma como os mesmos são apresentados.

Apêndice A

Código

```
from dataclasses import InitVar
from doctest import Example
from mimetypes import init
from lark import Discard
from lark import Lark,Token,Tree
from lark.tree import pydot__tree_to_png
from lark.visitors import Interpreter

class MyInterpreter (Interpreter):
    def __init__(self):
        self.output = {}
        self.warnings = {}
        self.errors = {}
        self.correct = True
        self.inCicle = False
        self.if_count = 0
        self.if_depth = {}
        self.nivel_if = 0
        self.instructions = {}
        self.controlID = 0
        self.controlStructs = {}
        self.if_concat = False
        self.ifCat = ""
        self.body_cat = False
        self.bodyCat = ""
        self.sugestoes = {}

        self.if_parts = {}
        self.code = ""
        self.html_body = ""

        self.ident_level = 0

        self.atomic_vars = dict()
        # ATOMIC_VARS = {VARNAME : (TYPE,VALUE,INIT?,USED?)}

        self.struct_vars = dict()
        # STRUCT_VARS = {VARNAME : (TYPE,SIZE,VALUE,USED?)}

```

```
40          self.nrStructs = dict()
41          # NR_STRUCTS = {ID : (TYPE, ACTIVE, PARENT_STRUCTS)}
42
43      def start(self,tree):
44          self.code += "-{\n"
45          self.html_body += "<pre><body>\n<p class=\"code\">\n-{\n</p>\n"
46          self.ident_level += 1
47          self.visit(tree.children[1])
48          self.ident_level -= 1
49          self.html_body += "<p class=\"code\">\n}-\n</p>\n"
50          self.code += "}-\n"
51
52          for var in self.atomic_vars.keys():
53              if var not in self.warnings.keys():
54                      self.warnings[var] = []
55
56              if self.atomic_vars[var][2] == 0 and self.atomic_vars[var][3] == 0:
57                  self.warnings[var].append("Variable \"" + var + "\" was never
                        initialized nor used.")
58
59              elif self.atomic_vars[var][2] == 1 and self.atomic_vars[var][3] == 0:
60                  self.warnings[var].append("Variable \"" + var + "\" was never used.")
61
62          for var in self.struct_vars.keys():
63              if var not in self.warnings.keys():
64                      self.warnings[var] = []
65
66              if self.struct_vars[var][0] not in self.nrStructs.keys():
67                  self.nrStructs[self.struct_vars[var][0]] = 1
68
69              else:
70                  self.nrStructs[self.struct_vars[var][0]] += 1
71
72              if self.struct_vars[var][3] == 0:
73                  self.warnings[var].append("Variable \"" + var + "\" was never used.")
74
75          self.output["atomic_vars"] = dict(self.atomic_vars)
76          self.output["struct_vars"] = dict(self.struct_vars)
77          self.output["correct"] = self.correct
78          erros = dict()
79          for k,v in self.errors.items():
80              erros[k] = []
81              for s in v:
82                  erros[k].append(s)
83
84          warns = dict()
85          for k,v in self.warnings.items():
86              warns[k] = []
87              for s in v:
88                  warns[k].append(s)
89
90          self.output["errors"] = erros
91          self.output["warnings"] = warns
92          self.output["if_count"] = self.if_count
```

```
93          self.output["if_depth"] = self.if_depth
94          self.output["nrStructs"] = self.nrStructs
95          self.output["instructions"] = dict(self.instructions)
96          self.output["controlStructs"] = dict(self.controlStructs)
97          self.output["if_parts"] = self.if_parts
98          self.output["code"] = self.code
99          self.output["html_body"] = self.html_body
100         self.output["sugestoes"] = self.sugestoes
101
102         self.if_strings = {}
103         self.if_bodys = {}
104
105         # IF CONCAT
106
107         for i in self.if_parts.keys():
108             self.if_concat = True
109             self.visit(self.if_parts[i][0])
110             self.if_strings[i] = self.ifCat
111             self.ifCat = ""
112
113         self.if_concat = False
114
115
116         for i in self.if_parts.keys():
117             self.body_cat = True
118             self.ident_level = 0
119             self.visit(self.if_parts[i][1])
120             self.if_bodys[i] = self.bodyCat
121             self.bodyCat = ""
122
123         self.body_cat = False
124
125         aux = {}
126         parentsSet = set()
127
128         for k,v in self.controlStructs.items():
129             l = v[2]
130             flag = True
131             parents = []
132             if len(l) == 0 or v[0] != "if":
133                 flag = False
134             for p in l:
135                 if self.controlStructs[p][0] != "if":
136                     flag = False
137                 if p not in parentsSet:
138                     parentsSet.add(p)
139                     parents.append(p)
140
141             if flag:
142                 aux[k] = tuple([v[0],v[1],parents])
143
144         auxIfs = {}
145         for k,t in aux.items():
146             p = t[2][0]
```

```
147
148             if p not in auxIfs.keys():
149                 auxIfs[p] = list()
150             auxIfs[p].append(k)
151
152         removeKeys = set()
153
154         for k,v in auxIfs.items():
155             for v1 in v:
156                 if v1 in auxIfs.keys():
157                     auxIfs[k].append(auxIfs[v1][0])
158                     auxIfs[v1] = []
159                     removeKeys.add(v1)
160
161         for k in removeKeys:
162             auxIfs.pop(k)
163
164
165         finalDict = {}
166
167         for k,l in auxIfs.items():
168             last = False
169             for v in l:
170                 if len(self.if_parts[v][1].children[1].children) > 1:
171                     if k not in finalDict.keys():
172                         finalDict[k] = []
173                     finalDict[k].append(v)
174                     last = True
175                 elif not last:
176                     if k not in finalDict.keys():
177                         finalDict[k] = []
178                     finalDict[k].append(v)
179
180         for k,l in finalDict.items():
181             condS = self.if_strings[k]
182
183             for v in l:
184                 condS += " & " + self.if_strings[v]
185
186         for k,v in finalDict.items():
187             i = 1
188             keyC = "if(" + self.if_strings[k] + "){\n"
189             for elem in v:
190                 for t in range(i):
191                     keyC += "\t"
192                 keyC += "if(" + self.if_strings[elem] + "){\n"
193                 i += 1
194
195
196             body = self.if_bodys[max(v)][2:len(self.if_bodys[max(v)])-2]
197             bodyLines = body.split("\n")
198
199             for line in bodyLines:
200                 for t in range(i-1):
```

```
201                     keyC += "\t"
202                 keyC += line + "\n"
203 
204             for elem in v:
205                 i -= 1
206                 for t in range(i):
207                     keyC += "\t"
208                 keyC += "}\n"
209             keyC += "}"
210 
211             valueC = "if((" + self.if_strings[k] + ")"
212 
213             for elem in v:
214                 valueC += " & (" + self.if_strings[elem] + ")"
215 
216             valueC += ")" + self.if_bodys[max(v)]
217 
218             self.sugestoes[keyC] = valueC
219 
220         return self.output
221 
222     def program(self, tree):
223         for c in tree.children:
224             self.visit(c)
225         pass
226 
227     def instruction(self,tree):
228         self.visit(tree.children[0])
229 
230         pass
231 
232     def comment(self, tree):
233         comment = tree.children[0].value
234         if self.body_cat:
235             for i in range(self.ident_level):
236                 self.bodyCat += "\t"
237             self.bodyCat += comment + "\n"
238         else:
239             self.code += comment + "\n"
240             self.html_body += "<p class=\"comment\">\n"
241 
242             for i in range(self.ident_level):
243                 self.html_body += "\t"
244 
245             self.html_body += comment + "\n</p>\n"
246 
247         pass
248 
249     def declaration(self, tree):
250         self.visit(tree.children[0])
251 
252         pass
253 
254     def atomic(self, tree):
```

```
255         if "atomic_declaration" not in self.instructions.keys():
256             self.instructions["atomic_declaration"] = 1
257         else:
258             self.instructions["atomic_declaration"] += 1
259 
260         var_type = tree.children[0].value
261 
262         var_name = tree.children[1].value
263 
264         if var_name not in self.errors.keys():
265             self.errors[var_name] = set()
266 
267         if not self.body_cat:
268             self.html_body += "<p class=\"code\">\n"
269             for i in range(self.ident_level):
270                 self.html_body += "\t"
271 
272         flag = False
273 
274         if(var_name in self.atomic_vars.keys() or var_name in self.struct_vars.keys()
                ):
275             self.correct = False
276             self.errors[var_name].add("Variable \"" + var_name + "\" declared more
                    than once!")
277             if not self.body_cat:
278                 self.html_body += "<div class=\"error\">"+var_type + " " + var_name
279             flag = True
280 
281         if self.body_cat:
282             for i in range(self.ident_level):
283                 self.bodyCat += "\t"
284             self.bodyCat += var_type + " " + var_name
285         else:
286             self.code += var_type + " " + var_name
287             if not flag:
288                 self.html_body += var_type + " " + var_name
289 
290         var_value = None
291         if flag == True:
292             var_value = self.atomic_vars[var_name][1]
293         init = 0
294         used = 0
295 
296         if(len(tree.children) > 3):
297             if self.body_cat:
298                 self.bodyCat += " = "
299             else:
300                 self.code += " = "
301                 self.html_body += " = "
302             var_value = self.visit(tree.children[3])
303             #print("RETORNA => " + str(var_value))
304             init = 1
305             if "atrib" not in self.instructions.keys():
306                 self.instructions["atrib"] = 1
```

```
307             else:
308                 self.instructions["atrib"] += 1
309 
310 
311         val = (var_type,var_value,init,used)
312         self.atomic_vars[var_name] = val
313 
314         if flag and not self.body_cat:
315             self.html_body += "<span class=\"errortext\">Variable declared more than
                    once!</span></div>"
316 
317         flag = False
318 
319         if self.body_cat:
320             self.bodyCat += ";\n"
321         else:
322             self.code += ";\n"
323             self.html_body += ";\n</p>\n"
324 
325         pass
326 
327     def elem(self, tree):
328 
329         if(not isinstance(tree.children[0], Tree)):
330             if self.body_cat:
331                 self.bodyCat += str(tree.children[0])
332             else:
333                 self.code += str(tree.children[0])
334                 self.html_body += str(tree.children[0])
335 
336             if(tree.children[0].type == "ESCAPED_STRING"):
337                 return str(tree.children[0].value[1:(len(tree.children[0].value)-1)])
338             elif(tree.children[0].type == "DECIMAL"):
339                 return float(tree.children[0].value)
340             elif(tree.children[0].type == "SIGNED_INT"):
341                 return int(tree.children[0].value)
342         else:
343             r = self.visit(tree.children[0])
344             return r
345 
346     def structure(self, tree):
347         if "structure_declaration" not in self.instructions.keys():
348             self.instructions["structure_declaration"] = 1
349         else:
350             self.instructions["structure_declaration"] += 1
351 
352         self.visit(tree.children[0])
353 
354         pass
355 
356     def set(self, tree):
357         if not self.body_cat:
358             self.html_body += "<p class=\"code\">\n"
359             for i in range(self.ident_level):
```

```
360                 self.html_body += "\t"
361
362         ret = set()
363         childs = len(tree.children)
364         sizeS = 0
365
366         if self.body_cat:
367             for i in range(self.ident_level):
368                 self.bodyCat += "\t"
369             self.bodyCat += "set " + tree.children[0].value
370         else:
371             self.code += "set " + tree.children[0].value
372             self.html_body += "set " + tree.children[0].value
373
374         if childs != 1 and childs != 4:
375             if self.body_cat:
376                 self.bodyCat += " = "
377             else:
378                 self.code += " = "
379                 self.html_body += " = "
380             for c in tree.children[2:]:
381                 if c != "{" and c != "}" and c != ",":
382                     ret.add(self.visit(c))
383                 if c == "{" or c == "}" or c == ",":
384                     if self.body_cat:
385                         self.bodyCat += c.value
386                     else:
387                         self.code += c.value
388                         self.html_body += c.value
389                     if c == ",":
390                         if self.body_cat:
391                             self.bodyCat += " "
392                         else:
393                             self.code += " "
394                             self.html_body += " "
395             sizeS = len(ret)
396         elif childs == 4:
397             if self.body_cat:
398                 self.bodyCat += " = {}"
399             else:
400                 self.code += " = {}"
401                 self.html_body += " = {}"
402
403         if self.body_cat:
404             self.bodyCat += ";\n"
405         else:
406             self.code += ";\n"
407             self.html_body += ";\n</p>\n"
408
409         self.struct_vars[tree.children[0].value] = ("set", sizeS, ret, 0)
410
411
412         pass
413
```

```
414     def list(self, tree):
415         if not self.body_cat:
416             self.html_body += "<p class=\"code\">\n"
417             for i in range(self.ident_level):
418                 self.html_body += "\t"
419 
420         ret = list()
421         childs = len(tree.children)
422         sizeL = 0
423         if self.body_cat:
424             for i in range(self.ident_level):
425                 self.bodyCat += "\t"
426             self.bodyCat += "list " + tree.children[0].value
427         else:
428             self.code += "list " + tree.children[0].value
429             self.html_body += "list " + tree.children[0].value
430 
431         if childs != 1 and childs != 4:
432             if self.body_cat:
433                 self.bodyCat += " = "
434             else:
435                 self.code += " = "
436                 self.html_body += " = "
437             for c in tree.children[2:]:
438                 if c != "[" and c != "]" and c != ",":
439                     ret.append(self.visit(c))
440                 if c == "[" or c == "]" or c == ",":
441                     if self.body_cat:
442                         self.bodyCat += c.value
443                     else:
444                         self.code += c.value
445                         self.html_body += c.value
446                     if c == ",":
447                         if self.body_cat:
448                             self.bodyCat += " "
449                         else:
450                             self.code += " "
451                             self.html_body += " "
452             sizeL = len(ret)
453         elif childs == 4:
454             if self.body_cat:
455                 self.bodyCat += " = []"
456             else:
457                 self.code += " = []"
458                 self.html_body += " = []"
459 
460         if self.body_cat:
461             self.bodyCat += ";\n"
462         else:
463             self.code += ";\n"
464             self.html_body += ";\n</p>\n"
465 
466         self.struct_vars[tree.children[0].value] = ("list", sizeL, ret, 0)
467 
```

```
468         pass
469 
470     def tuple(self, tree):
471         if not self.body_cat:
472             self.html_body += "<p class=\"code\">\n"
473             for i in range(self.ident_level):
474                 self.html_body += "\t"
475 
476         aux = list()
477         ret = tuple()
478         sizeT = 0
479         childs = len(tree.children)
480         if self.body_cat:
481             for i in range(self.ident_level):
482                 self.bodyCat += "\t"
483             self.bodyCat += "tuple " + tree.children[0].value
484         else:
485             self.code += "tuple " + tree.children[0].value
486             self.html_body += "tuple " + tree.children[0].value
487         if childs != 1 and childs != 4:
488             if self.body_cat:
489                 self.bodyCat += " = "
490             else:
491                 self.code += " = "
492                 self.html_body += " = "
493             for c in tree.children[2:]:
494                 if c != "(" and c != ")" and c != ",":
495                     aux.append(self.visit(c))
496                 if c == "(" or c == ")" or c == ",":
497                     if self.body_cat:
498                         self.bodyCat += c.value
499                     else:
500                         self.code += c.value
501                         self.html_body += c.value
502                     if c == ",":
503                         if self.body_cat:
504                             self.bodyCat += " "
505                         else:
506                             self.code += " "
507                             self.html_body += " "
508             ret = tuple(aux)
509             sizeT = len(ret)
510         elif childs == 4:
511             if self.body_cat:
512                 self.bodyCat += " = ()"
513             else:
514                 self.code += " = ()"
515                 self.html_body += " = ()"
516 
517         if self.body_cat:
518             self.bodyCat += ";\n"
519         else:
520             self.code += ";\n"
521             self.html_body += ";\n</p>\n"
```

```
522
523         self.struct_vars[tree.children[0].value] = ("tuple", sizeT, ret, 0)
524
525         pass
526
527     def dict(self, tree):
528         if not self.body_cat:
529             self.html_body += "<p class=\"code\">\n"
530             for i in range(self.ident_level):
531                 self.html_body += "\t"
532
533         ret = dict()
534         childs = len(tree.children)
535         sizeD = 0
536         if self.body_cat:
537             for i in range(self.ident_level):
538                 self.bodyCat += "\t"
539             self.bodyCat += "dict " + tree.children[0].value
540         else:
541             self.code += "dict " + tree.children[0].value
542             self.html_body += "dict " + tree.children[0].value
543         if childs != 1 and childs != 4:
544             if self.body_cat:
545                 self.bodyCat += " = {"
546             else:
547                 self.code += " = {"
548                 self.html_body += " = {"
549             start = 3
550             while start < childs-1:
551                 key = self.visit(tree.children[start])
552                 if self.body_cat:
553                     self.bodyCat += " : "
554                 else:
555                     self.code += " : "
556                     self.html_body += " : "
557                 value = self.visit(tree.children[start+2])
558                 if start + 4 < (childs-1):
559                     if self.body_cat:
560                         self.bodyCat += ", "
561                     else:
562                         self.code += ", "
563                         self.html_body += ", "
564                 ret[key] = value
565                 start += 4
566             if self.body_cat:
567                 self.bodyCat += "}"
568             else:
569                 self.code += "}"
570                 self.html_body += "}"
571             sizeD = len(ret)
572         elif childs == 4:
573             if self.body_cat:
574                 self.bodyCat += " = {}"
575             else:
```

```
576                 self.code += " = {}"
577                 self.html_body += " = {}"
578 
579         if self.body_cat:
580             self.bodyCat += ";\n"
581         else:
582             self.code += ";\n"
583             self.html_body += ";\n</p>\n"
584 
585         self.struct_vars[tree.children[0].value] = ("dict", sizeD, ret, 0)
586 
587         pass
588 
589     def atrib(self,tree):
590 
591         if "atrib" not in self.instructions.keys():
592             self.instructions["atrib"] = 1
593         else:
594             self.instructions["atrib"] += 1
595 
596         if not self.body_cat:
597             self.html_body += "<p class=\"code\">\n"
598             for i in range(self.ident_level):
599                 self.html_body += "\t"
600             self.code += tree.children[0].value + " = "
601         else:
602             for i in range(self.ident_level):
603                 self.bodyCat += "\t"
604             self.bodyCat += tree.children[0].value + " = "
605 
606         if str(tree.children[0]) not in self.errors.keys():
607             self.errors[str(tree.children[0])] = set()
608 
609         if str(tree.children[0]) not in self.atomic_vars.keys():
610             self.errors[str(tree.children[0])].add("Variable \"" + tree.children[0] +
                 "\" was not declared")
611             self.correct = False
612             typeV = "undefined"
613             valueV = None
614             if not self.body_cat:
615                 self.html_body += "<div class=\"error\">" + tree.children[0].value +
                     " = "
616             valueV = self.visit(tree.children[2])
617             if not self.body_cat:
618                 self.html_body += "<span class=\"errortext\">Variable undeclared</
                     span></div>"
619             self.atomic_vars[str(tree.children[0])] = tuple([typeV,valueV,0,1])
620 
621         else:
622             typeV = self.atomic_vars[str(tree.children[0])][0]
623             if not self.body_cat:
624                 self.html_body += tree.children[0].value + " = "
625             valueV = self.visit(tree.children[2])
626             self.atomic_vars[str(tree.children[0])] = tuple([typeV,valueV,1,1])
```

```
627
628         if self.body_cat:
629             self.bodyCat += ";\n"
630         else:
631             self.html_body += ";\n</p>\n"
632             self.code += ";\n"
633
634         pass
635
636     def initcicle(self, tree):
637         if "atrib" not in self.instructions.keys():
638             self.instructions["atrib"] = 1
639         else:
640             self.instructions["atrib"] += 1
641
642         if self.body_cat:
643             self.bodyCat += tree.children[0].value + " = "
644         else:
645             self.code += tree.children[0].value + " = "
646
647         if str(tree.children[0]) not in self.errors.keys():
648             self.errors[str(tree.children[0])] = set()
649         if str(tree.children[0]) not in self.atomic_vars.keys():
650             self.errors[str(tree.children[0])].add("Variable \"" + tree.children[0] +
                 "\" was not declared")
651             if not self.body_cat:
652                 self.html_body += "<div class=\"error\">"+tree.children[0].value + "
                     = "
653             valueV = self.visit(tree.children[2])
654             if not self.body_cat:
655                 self.html_body += "<span class=\"errortext\">Variable was not
                     declared</span></div>"
656             self.correct = False
657         else:
658             typeV = self.atomic_vars[tree.children[0]][0]
659             if not self.body_cat:
660                 self.html_body += tree.children[0].value + " = "
661             valueV = self.visit(tree.children[2])
662             self.atomic_vars[str(tree.children[0])] = tuple([typeV,valueV,1,1])
663
664         pass
665
666     def print(self,tree):
667         if "print" not in self.instructions.keys():
668             self.instructions["print"] = 1
669         else:
670             self.instructions["print"] += 1
671
672         if not self.body_cat:
673             self.html_body += "<p class=\"code\">\n"
674             for i in range(self.ident_level):
675                 self.html_body += "\t"
676             self.html_body += "print("
677             self.code += "print("
```

```
678         else:
679             for i in range(self.ident_level):
680                 self.bodyCat += "\t"
681             self.bodyCat += "print("
682 
683         if tree.children[1].type == "VARNAME":
684             if self.body_cat:
685                 self.bodyCat += tree.children[1].value
686             else:
687                 self.code += tree.children[1].value
688             if str(tree.children[1]) not in self.errors.keys():
689                 self.errors[str(tree.children[1])] = set()
690             if str(tree.children[1]) not in self.atomic_vars.keys():
691                 self.errors[str(tree.children[1])].add("Variable \"" + tree.children
                        [1] + "\" was not declared")
692                 if not self.body_cat:
693                     self.html_body += "<div class=\"error\">" + tree.children[0].
                            value + "<span class=\"errortext\">Variable undeclared</span
                            ></div>"
694                 self.correct = False
695             elif not self.atomic_vars[str(tree.children[1])][2]:
696                 self.errors[str(tree.children[1])].add("Variable \"" + tree.children
                        [1] + "\" declared but not initialized")
697                 if not self.body_cat:
698                     self.html_body += "<div class=\"error\">" + tree.children[0].
                            value + "<span class=\"errortext\">Variable was never
                            initialized</span></div>"
699                 self.correct = False
700             else:
701                 if not self.body_cat:
702                     self.html_body += tree.children[1].value
703 
704         elif tree.children[1].type == "ESCAPED_STRING":
705             if self.body_cat:
706                 self.bodyCat += tree.children[1].value
707             else:
708                 self.code += tree.children[1].value
709                 self.html_body += tree.children[1].value
710             s = tree.children[1]
711             s = s.replace("\"","")
712 
713         if self.body_cat:
714             self.bodyCat += ");\n"
715         else:
716             self.code += ");\n"
717             self.html_body += ");\n</p>\n"
718 
719         pass
720 
721     def read(self,tree):
722         if "read" not in self.instructions.keys():
723             self.instructions["read"] = 1
724         else:
725             self.instructions["read"] += 1
```

```
726
727         if not self.body_cat:
728             self.html_body += "<p class=\"code\">\n"
729             for i in range(self.ident_level):
730                 self.html_body += "\t"
731             self.html_body += "read("
732             self.code += "read(" + tree.children[1].value
733         else:
734             for i in range(self.ident_level):
735                 self.bodyCat += "\t"
736             self.bodyCat += "read(" + tree.children[1].value
737
738         if str(tree.children[1]) not in self.errors.keys():
739             self.errors[str(tree.children[1])] = set()
740
741         if str(tree.children[1]) not in self.atomic_vars.keys():
742             if str(tree.children[1]) in self.struct_vars.keys():
743                 self.errors[str(tree.children[1])].add("Variable \"" + tree.children
                        [1] + "\" cannot be defined by user input.")
744                 if not self.body_cat:
745                     self.html_body += "<div class=\"error\">" + tree.children[0].
                            value + "<span class=\"errortext\">Variable is a structure</
                            span></div>"
746             else:
747                 self.errors[str(tree.children[1])].add("Variable \"" + tree.children
                        [1] + "\" was not declared")
748                 if not self.body_cat:
749                     self.html_body += "<div class=\"error\">" + tree.children[0].
                            value + "<span class=\"errortext\">Variable undeclared</span
                            ></div>"
750             self.correct = False
751
752         else:
753             if not self.body_cat:
754                 self.html_body += tree.children[1].value
755             value = input("> ")
756             typeV = self.atomic_vars[tree.children[1]][0]
757             initV = 1
758             usedV = 1
759             val = int(value)
760             self.atomic_vars[tree.children[1]] = tuple([typeV, val, initV, usedV])
761
762         if self.body_cat:
763             self.bodyCat += ");\n"
764         else:
765             self.code += ");\n"
766             self.html_body += ");\n</p>\n"
767
768         pass
769
770     def cond(self,tree):
771         if "if" not in self.instructions.keys():
772             self.instructions["if"] = 1
773         else:
```

```
774             self.instructions["if"] += 1
775
776         if not self.body_cat:
777             self.html_body += "<p class=\"code\">\n"
778             for i in range(self.ident_level):
779                 self.html_body += "\t"
780
781
782         # Vamos buscar todas as estruturas que est o ativas (ainda nao foram
                fechadas) e consideramos que a estrutura est  aninhada dentro delas
783         parents = []
784         for id in self.controlStructs.keys():
785             if self.controlStructs[id][1] == 1:
786                 parents.append(id)
787
788         if not self.body_cat:
789             self.if_parts[self.controlID] = (tree.children[2],tree.children[4])
790         # Pomos no dict um tuplo com o tipo da estrutura de controlo, uma flag que
                nos diz que est  ativa e a lista das estruturas de hierarquia superior
791             self.controlStructs[(self.controlID)] = tuple(["if",1,parents])
792         # Incrementamos o ID para a proxima estrutura de controlo
793         self.controlID += 1
794
795         # Usamos o contador de ifs para definir os ids das estruturas de controlo
796         self.if_count += 1
797         self.if_depth[self.if_count] = self.nivel_if
798
799         l = len(tree.children)
800
801         if self.body_cat:
802             for i in range(self.ident_level):
803                 self.bodyCat += "\t"
804             self.bodyCat += "if("
805         else:
806             self.code += "if("
807             self.html_body += "if("
808         self.visit(tree.children[2])
809         if self.body_cat:
810             self.bodyCat += ")"
811         else:
812             self.code += ")"
813             self.html_body += ")"
814
815         self.visit(tree.children[4])
816
817         if(tree.children[(l-2)] == "else"):
818             if self.body_cat:
819                 self.bodyCat += " else "
820             else:
821                 self.code += " else "
822                 self.html_body += " else "
823             self.visit(tree.children[(l-1)])
824
825         pass
```

```
826
827     def ciclewhile(self,tree):
828         if "while" not in self.instructions.keys():
829             self.instructions["while"] = 1
830         else:
831             self.instructions["while"] += 1
832
833         if not self.body_cat:
834             self.html_body += "<p class=\"code\">\n"
835             for i in range(self.ident_level):
836                 self.html_body += "\t"
837
838         # Vamos buscar todas as estruturas que est o ativas (ainda nao foram
                fechadas) e consideramos que a estrutura est  aninhada dentro delas
839         parents = []
840         for id in self.controlStructs.keys():
841             if self.controlStructs[id][1] == 1:
842                 parents.append(id)
843
844         # Pomos no dict um tuplo com o tipo da estrutura de controlo, uma flag que
                nos diz que est  ativa e a lista das estruturas de hierarquia superior
845         self.controlStructs[self.controlID] = tuple(["while",1,parents])
846         # Incrementamos o ID para a proxima estrutura de controlo
847         self.controlID += 1
848
849         aux = self.nivel_if
850         self.nivel_if = 0
851         self.inCicle = True
852
853         if self.body_cat:
854             for i in range(self.ident_level):
855                 self.bodyCat += "\t"
856             self.bodyCat +=  "while("
857         else:
858             self.code += "while("
859             self.html_body += "while("
860
861         self.visit(tree.children[2])
862
863         if self.body_cat:
864             self.bodyCat += ")"
865         else:
866             self.code += ")"
867             self.html_body += ")"
868
869         self.visit(tree.children[4])
870
871         self.inCicle = False
872         self.nivel_if = aux
873
874         pass
875
876     def ciclefor(self,tree):
877         if "for" not in self.instructions.keys():
```

```
878             self.instructions["for"] = 1
879         else:
880             self.instructions["for"] += 1
881 
882         # Vamos buscar todas as estruturas que est o ativas (ainda nao foram
                fechadas) e consideramos que a estrutura est  aninhada dentro delas
883         parents = []
884         for id in self.controlStructs.keys():
885             if self.controlStructs[id][1] == 1:
886                 parents.append(id)
887 
888         # Pomos no dict um tuplo com o tipo da estrutura de controlo, uma flag que
                nos diz que est  ativa e a lista das estruturas de hierarquia superior
889         self.controlStructs[self.controlID] = tuple(["for",1,parents])
890         # Incrementamos o ID para a proxima estrutura de controlo
891         self.controlID += 1
892 
893         aux = self.nivel_if
894         self.nivel_if = 0
895         self.inCicle = True
896 
897         if not self.body_cat:
898             self.html_body += "<p class=\"code\">\n"
899             for i in range(self.ident_level):
900                 self.html_body += "\t"
901         else:
902             for i in range(self.ident_level):
903                 self.bodyCat += "\t"
904 
905         for c in tree.children:
906             if c != "for" and c != "(" and c != ")" and c != ";" and c != ",":
907                 self.visit(c)
908             if c == "for" or c == "(" or c == ")" or c == ";" or c == ",":
909                 if self.body_cat:
910                     self.bodyCat += c.value
911                 else:
912                     self.code += c.value
913                     self.html_body += c.value
914                 if(c == ";" or c == ","):
915                     if self.body_cat:
916                         self.bodyCat += " "
917                     else:
918                         self.code += " "
919                         self.html_body += " "
920 
921         self.inCicle = False
922         self.nivel_if = aux
923 
924         pass
925 
926     def inc(self, tree):
927         if self.body_cat:
928             self.bodyCat += tree.children[0] + "++"
929         else:
```

```
930             self.code += tree.children[0] + "++"
931             self.html_body += tree.children[0] + "++"
932         typeV = self.atomic_vars[str(tree.children[0])][0]
933         valueV = self.atomic_vars[str(tree.children[0])][1] + 1
934         self.atomic_vars[str(tree.children[0])] = tuple([typeV,valueV,1,1])
935 
936         pass
937 
938     def dec(self, tree):
939         if self.body_cat:
940             self.bodyCat += tree.children[0] + "--"
941         else:
942             self.code += tree.children[0] + "-"
943             self.html_body += tree.children[0] + "--"
944         typeV = self.atomic_vars[str(tree.children[0])][0]
945         valueV = self.atomic_vars[str(tree.children[0])][1] - 1
946         self.atomic_vars[str(tree.children[0])] = tuple([typeV,valueV,1,1])
947 
948         pass
949 
950     def ciclerepeat(self,tree):
951         if "repeat" not in self.instructions.keys():
952             self.instructions["repeat"] = 1
953         else:
954             self.instructions["repeat"] += 1
955 
956         # Vamos buscar todas as estruturas que est o ativas (ainda nao foram
               fechadas) e consideramos que a estrutura est  aninhada dentro delas
957         parents = []
958         for id in self.controlStructs.keys():
959             if self.controlStructs[id][1] == 1:
960                 parents.append(id)
961 
962         # Pomos no dict um tuplo com o tipo da estrutura de controlo, uma flag que
               nos diz que est  ativa e a lista das estruturas de hierarquia superior
963         self.controlStructs[self.controlID] = tuple(["repeat",1,parents])
964         # Incrementamos o ID para a proxima estrutura de controlo
965         self.controlID += 1
966 
967         aux = self.nivel_if
968         self.nivel_if = 0
969         self.inCicle = True
970 
971         if not self.body_cat:
972             self.html_body += "<p class=\"code\">\n"
973             for i in range(self.ident_level):
974                 self.html_body += "\t"
975 
976             self.code += "repeat(" + tree.children[2].value + ")"
977             self.html_body += "repeat(" + tree.children[2].value + ")"
978         else:
979             for i in range(self.ident_level):
980                 self.bodyCat += "\t"
981             self.bodyCat += "repeat(" + tree.children[2].value + ")"
```

```
982
983         self.visit(tree.children[4])
984
985         self.inCicle = False
986         self.nivel_if = aux
987
988         pass
989
990     def body(self,tree):
991         self.visit_children(tree)
992
993         pass
994
995     def open(self,tree):
996         if not self.inCicle:
997             self.nivel_if += 1
998         if self.body_cat:
999             self.bodyCat += "{\n"
1000            self.ident_level += 1
1001        else:
1002            self.code += "{\n"
1003            self.ident_level += 1
1004            self.html_body += "{\n</p>\n"
1005
1006        pass
1007
1008    def close(self,tree):
1009        self.nivel_if -= 1
1010
1011        newDict = dict(filter(lambda elem: elem[1][1] == 1, self.controlStructs.items
                ()))
1012
1013        if len(newDict.keys()) > 0:
1014            k = max(newDict.keys())
1015            self.controlStructs[k] = (self.controlStructs[k][0],0,self.controlStructs
                    [k][2])
1016
1017        if not self.body_cat:
1018            self.ident_level -= 1
1019            self.code += "}\n"
1020            if not self.body_cat:
1021                self.html_body += "<p class=\"code\">\n"
1022                for i in range(self.ident_level):
1023                    self.html_body += "\t"
1024            self.html_body += "}\n</p>\n"
1025        else:
1026            self.ident_level -= 1
1027            for i in range(self.ident_level):
1028                self.bodyCat += "\t"
1029            self.bodyCat += "}\n"
1030        pass
1031
1032    def op(self,tree):
1033        if(len(tree.children) > 1):
```

```
1034            if(tree.children[0] == "!"):
1035
1036                if self.if_concat:
1037                    self.ifCat += "!"
1038                elif self.body_cat:
1039                    self.bodyCat += "!"
1040                else:
1041                    self.code += "!"
1042                    self.html_body += "!"
1043                r = int(self.visit(tree.children[1]))
1044                if r == 0: r = 1
1045                else: r = 0
1046            elif(tree.children[1] == "&"):
1047                t1 = self.visit(tree.children[0])
1048
1049                if self.if_concat:
1050                    self.ifCat += " & "
1051                elif self.body_cat:
1052                    self.bodyCat += " & "
1053                else:
1054                    self.code += " & "
1055                    self.html_body += " & "
1056
1057                t2 = self.visit(tree.children[2])
1058                if t1 and t2:
1059                    r = 1
1060                else:
1061                    r = 0
1062            elif(tree.children[1] == "#"):
1063                t1 = self.visit(tree.children[0])
1064                if self.if_concat:
1065                    self.ifCat += " # "
1066                elif self.body_cat:
1067                    self.bodyCat += " # "
1068                else:
1069                    self.html_body += " # "
1070                    self.code += " # "
1071
1072                t2 = self.visit(tree.children[2])
1073                if t1 or t2:
1074                    r = 1
1075                else:
1076                    r = 0
1077        else:
1078            r = self.visit(tree.children[0])
1079
1080        return r
1081
1082    def factcond(self,tree):
1083        if len(tree.children) > 1:
1084            t1 = self.visit(tree.children[0])
1085            if self.if_concat:
1086                self.ifCat += " " + tree.children[1].value + " "
1087            elif self.body_cat:
```

```
1088                self.bodyCat += " " + tree.children[1].value + " "
1089            else:
1090                self.code += " " + tree.children[1].value + " "
1091                self.html_body += " " + tree.children[1].value + " "
1092
1093            t2 = self.visit(tree.children[2])
1094            if tree.children[1] == "<=":
1095                if t1 <= t2:
1096                    r = 1
1097                else:
1098                    r = 0
1099            elif tree.children[1] == "<":
1100                if t1 < t2:
1101                    r = 1
1102                else:
1103                    r = 0
1104            elif tree.children[1] == ">=":
1105                if t1 >= t2:
1106                    r = 1
1107                else:
1108                    r = 0
1109            elif tree.children[1] == ">":
1110                if t1 > t2:
1111                    r = 1
1112                else:
1113                    r = 0
1114            elif tree.children[1] == "==":
1115                if t1 == t2:
1116                    r = 1
1117                else:
1118                    r = 0
1119            elif tree.children[1] == "!=":
1120                if t1 != t2:
1121                    r = 1
1122                else:
1123                    r = 0
1124        else:
1125            r = self.visit(tree.children[0])
1126
1127        return r
1128
1129    def expcond(self,tree):
1130        if len(tree.children) > 1:
1131            t1 = self.visit(tree.children[0])
1132            if self.if_concat:
1133                self.ifCat += " " + tree.children[1].value + " "
1134            elif self.body_cat:
1135                self.bodyCat += " " + tree.children[1].value + " "
1136            else:
1137                self.html_body += " " + tree.children[1].value + " "
1138                self.code += " " + tree.children[1].value + " "
1139
1140            t2 = self.visit(tree.children[2])
1141            if(tree.children[1] == "+"):
```

```
1142                r = t1 + t2
1143            elif(tree.children[1] == "-"):
1144                r = t1 - t2
1145        else:
1146            r = self.visit(tree.children[0])
1147
1148        return r
1149
1150    def termocond(self,tree):
1151        if len(tree.children) > 1:
1152            t1 = self.visit(tree.children[0])
1153            if self.if_concat:
1154                self.ifCat += " " + tree.children[1].value + " "
1155            elif self.body_cat:
1156                self.bodyCat += " " + tree.children[1].value + " "
1157            else:
1158                self.code += " " + tree.children[1].value + " "
1159                self.html_body += " " + tree.children[1].value + " "
1160
1161            t2 = self.visit(tree.children[2])
1162            if(tree.children[1] == "*"):
1163                r = t1 * t2
1164            elif(tree.children[1] == "/"):
1165                r = int(t1 / t2)
1166            elif(tree.children[1] == "%"):
1167                r = t1 % t2
1168        else:
1169            r = self.visit(tree.children[0])
1170
1171        return r
1172
1173    def factor(self,tree):
1174        r = None
1175        if tree.children[0].type == 'SIGNED_INT':
1176            r = int(tree.children[0])
1177            if self.if_concat:
1178                self.ifCat += str(r)
1179            elif self.body_cat:
1180                self.bodyCat += str(r)
1181                #print("r => " + str(r))
1182            else:
1183                self.code += str(r)
1184                self.html_body += str(r)
1185
1186        elif tree.children[0].type == 'VARNAME':
1187
1188            if str(tree.children[0]) not in self.errors.keys():
1189                self.erros[str(tree.children[0])] = set()
1190
1191            if str(tree.children[0]) not in self.atomic_vars.keys():
1192                self.errors[str(tree.children[0])].add("Undeclared variable \"" + str
                        (tree.children[0]) + "\"")
1193                self.correct = False
1194                if not self.if_concat and not self.body_cat:
```

```
1195                     self.html_body += "<div class=\"error\">"+tree.children[0].value
                             +"<span class=\"errortext\">Undeclared Variable</span></div>"
1196                 r = -1
1197             elif self.atomic_vars[str(tree.children[0])][2] == 0:
1198                 #print(tree.children[0].value + " -> " + str(self.atomic_vars[str(
                         tree.children[0])]))
1199                 self.errors[str(tree.children[0])].add("Variable \"" + str(tree.
                         children[0]) + "\" was never initialized")
1200                 if not self.if_concat and not self.body_cat:
1201                     self.html_body += "<div class=\"error\">"+tree.children[0].value
                             +"<span class=\"errortext\">Variable was never initialized</
                             span></div>"
1202                 self.correct = False
1203                 r = self.atomic_vars[str(tree.children[0])][1]
1204                 typeV = self.atomic_vars[str(tree.children[0])][0]
1205                 initV = self.atomic_vars[str(tree.children[0])][2]
1206                 self.atomic_vars[str(tree.children[0])] = tuple([typeV,r,initV,1])
1207             else:
1208                 r = self.atomic_vars[str(tree.children[0])][1]
1209                 typeV = self.atomic_vars[str(tree.children[0])][0]
1210                 initV = self.atomic_vars[str(tree.children[0])][2]
1211                 if not self.if_concat and not self.body_cat:
1212                     self.html_body += tree.children[0].value
1213                 self.atomic_vars[str(tree.children[0])] = tuple([typeV,r,initV,1])
1214 
1215             if self.if_concat:
1216                 self.ifCat += tree.children[0].value
1217             elif self.body_cat:
1218                 self.bodyCat += tree.children[0].value
1219             else:
1220                 self.code += tree.children[0].value
1221 
1222 
1223         elif tree.children[0] == "(":
1224             r = self.visit(tree.children[1])
1225 
1226         return r
1227 
1228 class GraphInterpreter (Interpreter):
1229     def __init__(self):
1230         self.cfg = "digraph G {\n\t\"entry\" -> "
1231         self.sdg = "digraph G {\n\t"
1232         self.statements = set()
1233         self.ifID = -1
1234         self.structureID = -1
1235         self.atomicID = -1
1236         self.atribID = -1
1237         self.printID = -1
1238         self.readID = -1
1239         self.whileID = -1
1240         self.repeatID = -1
1241         self.forID = -1
1242         self.initcicleID = -1
1243         self.incID = -1
```

```
1244            self.decID = -1
1245            self.output = {}
1246            self.second = False
1247            self.incicle = False
1248            self.only = False
1249            self.islands = set()
1250            self.mccabe = 0
1251
1252        def start(self,tree):
1253            #print("comecei")
1254            self.visit(tree.children[1])
1255            self.cfg += "\"fim\"\n"
1256
1257            lines = self.cfg.splitlines()
1258            lines2 = self.sdg.splitlines()
1259            #print(lines)
1260            #print(lines2[1:len(lines2)-1])
1261
1262            statements = set()
1263            statements2 = set()
1264            chunks = {}
1265
1266            for line in lines:
1267                aux = line.split(" -> ")
1268                for c in aux:
1269                    c = c.replace("\t","")
1270                    statements.add(c)
1271
1272            for line in lines2[1:len(lines2)-1]:
1273                aux = line.split(" -> ")
1274
1275                aux[0] = aux[0].replace("\t","")
1276                aux[1] = aux[1].replace("\t","")
1277
1278                aux[0] = aux[0].replace("\"","")
1279                aux[1] = aux[1].replace("\"","")
1280
1281                if aux[0] not in chunks.keys():
1282                    chunks[aux[0]] = []
1283                chunks[aux[0]].append(aux[1])
1284
1285                statements2.add(aux[0])
1286                statements2.add(aux[1])
1287
1288
1289
1290            ks = set()
1291            for k in chunks.keys():
1292                ks.add(k)
1293                self.islands.add(k)
1294
1295            for k in ks:
1296                for t,v in chunks.items():
1297                    if k != t:
```

```
                if k in v:
                    self.islands.remove(k)
        self.islands.remove("entry")

        for c in statements:
            i = 0
            while i <= self.ifID:
                s = "if_" + str(i) + "_start"
                if s in c:
                    if c[0] != "\t":
                        c = "\t" + c
                    self.cfg += c + " [shape=diamond]\n"
                i = i + 1

            i = 0
            while i <= self.whileID:
                s = "while_" + str(i) + "_start"
                if s in c:
                    if c[0] != "\t":
                        c = "\t" + c
                    self.cfg += c + " [shape=diamond]\n"
                i = i + 1

            i = 0
            while i <= self.repeatID:
                s = "repeat_" + str(i) + "_start"
                if s in c:
                    if c[0] != "\t":
                        c = "\t" + c
                    self.cfg += c + " [shape=diamond]\n"
                i = i + 1

            i = 0
            while i <= self.forID:
                s = "for_" + str(i) + "_cond"
                if s in c:
                    if c[0] != "\t":
                        c = "\t" + c
                    self.cfg += c + " [shape=diamond]\n"
                i = i + 1

        self.cfg += "}"

        self.sdg += "\n}"
        #print(self.cfg)
        #print(self.sdg)
        #print(self.islands)

        edges = len(lines2[1:len(lines2)-1])
        nodes = len(statements2)

        self.mccabe = edges - nodes + 2

        self.output["cfg"] = self.cfg
```

```
1352         self.output["sdg"] = self.sdg
1353         self.output["islands"] = self.islands
1354         self.output["mccabe"] = self.mccabe
1355 
1356         return self.output
1357 
1358     def program(self,tree):
1359 
1360         for c in tree.children:
1361             if not self.incicle and c.children[0].data != "comment":
1362                 self.sdg += "\"entry\" -> "
1363             if c.children[0].data != "comment":
1364                 self.visit(c)
1365         pass
1366 
1367     def instruction(self,tree):
1368         self.visit(tree.children[0])
1369 
1370         pass
1371 
1372     def declaration(self, tree):
1373         self.visit(tree.children[0])
1374 
1375         pass
1376 
1377     def atomic(self,tree):
1378         if not self.second:
1379             self.atomicID += 1
1380 
1381         var_type = tree.children[0].value
1382 
1383         var_name = tree.children[1].value
1384 
1385         self.cfg += "\"atomic_" + str(self.atomicID) + " " + var_type + " " +
                var_name
1386         self.sdg += "\"atomic_" + str(self.atomicID) + " " + var_type + " " +
                var_name
1387 
1388         if(len(tree.children) > 3):
1389             self.cfg += " = "
1390             self.sdg += " = "
1391             self.visit(tree.children[3])
1392 
1393         self.cfg += "\"\n\t" + "\"atomic_" + str(self.atomicID) + " " + var_type + "
                " + var_name
1394         self.sdg += "\"\n\t"
1395 
1396         if(len(tree.children) > 3):
1397             self.cfg += " = "
1398             self.second = True
1399             self.visit(tree.children[3])
1400             self.second = False
1401 
1402         self.cfg += "\" -> "
```

```
1403
1404    def elem(self, tree):
1405        if(not isinstance(tree.children[0], Tree)):
1406            if(tree.children[0].type == "ESCAPED_STRING"):
1407                self.cfg += tree.children[0].replace('\"','\'')
1408                if not self.second:
1409                    self.sdg += tree.children[0].replace('\"','\'')
1410            else:
1411                self.cfg += str(tree.children[0])
1412                if not self.second:
1413                    self.sdg += str(tree.children[0])
1414        else:
1415            r = self.visit(tree.children[0])
1416            return r
1417
1418    def structure(self, tree):
1419        if not self.second:
1420            self.structureID += 1
1421
1422        self.cfg += "\"structure_" + str(self.structureID) + " "
1423        self.sdg += "\"structure_" + str(self.structureID) + " "
1424
1425        self.visit(tree.children[0])
1426
1427        self.second = True
1428
1429        self.cfg += "\"\n\t\"structure_" + str(self.structureID) + " "
1430        self.sdg += "\"\n\t"
1431
1432        self.visit(tree.children[0])
1433
1434        self.cfg += "\" -> "
1435
1436        self.second = False
1437
1438        pass
1439
1440    def set(self, tree):
1441        childs = len(tree.children)
1442
1443        self.cfg += "set " + tree.children[0].value
1444        if not self.second:
1445            self.sdg += "set " + tree.children[0].value
1446
1447        if childs != 1 and childs != 4:
1448            self.cfg += " = "
1449            if not self.second:
1450                    self.sdg += " = "
1451
1452            for c in tree.children[2:]:
1453                if c != "{" and c != "}" and c != ",":
1454                    self.visit(c)
1455                if c == "{" or c == "}" or c == ",":
1456                    self.cfg += c.value
```

```
1457                    if not self.second:
1458                        self.sdg += c.value
1459                    if c == ",":
1460                        self.cfg += " "
1461                        if not self.second:
1462                            self.sdg += " "
1463        elif childs == 4:
1464            self.cfg += " = {}"
1465            if not self.second:
1466                    self.sdg += " = {}"
1467
1468        pass
1469
1470    def list(self, tree):
1471        childs = len(tree.children)
1472
1473        self.cfg += "list " + tree.children[0].value
1474        if not self.second:
1475            self.sdg += "list " + tree.children[0].value
1476
1477        if childs != 1 and childs != 4:
1478            self.cfg += " = "
1479            if not self.second:
1480                self.sdg += " = "
1481
1482            for c in tree.children[2:]:
1483                if c != "[" and c != "]" and c != ",":
1484                    self.visit(c)
1485                if c == "[" or c == "]" or c == ",":
1486                    self.cfg += c.value
1487                    if not self.second:
1488                        self.sdg += c.value
1489                    if c == ",":
1490                        self.cfg += " "
1491                        if not self.second:
1492                            self.sdg += " "
1493        elif childs == 4:
1494            self.cfg += " = []"
1495            if not self.second:
1496                self.sdg += " = []"
1497        pass
1498
1499    def tuple(self, tree):
1500        childs = len(tree.children)
1501
1502        self.cfg += "tuple " + tree.children[0].value
1503        if not self.second:
1504            self.sdg += "tuple " + tree.children[0].value
1505        if childs != 1 and childs != 4:
1506            self.cfg += " = "
1507            if not self.second:
1508                self.sdg += " = "
1509            for c in tree.children[2:]:
1510                if c != "(" and c != ")" and c != ",":
```

```
1511                     self.visit(c)
1512                 if c == "(" or c == ")" or c == ",":
1513                     self.cfg += c.value
1514                     if not self.second:
1515                         self.sdg += c.value
1516                     if c == ",":
1517                         self.cfg += " "
1518                         if not self.second:
1519                             self.sdg += " "
1520         elif childs == 4:
1521             self.cfg += " = ()"
1522             if not self.second:
1523                 self.sdg += " = ()"
1524         pass
1525 
1526     def dict(self, tree):
1527         childs = len(tree.children)
1528 
1529         self.cfg += "dict " + tree.children[0].value
1530         if not self.second:
1531             self.sdg += "dict " + tree.children[0].value
1532 
1533         if childs != 1 and childs != 4:
1534             self.cfg += " = {"
1535             if not self.second:
1536                 self.sdg += " = {"
1537             start = 3
1538             while start < childs-1:
1539                 key = self.visit(tree.children[start])
1540                 self.cfg += " : "
1541                 if not self.second:
1542                     self.sdg += " : "
1543                 value = self.visit(tree.children[start+2])
1544                 if start + 4 < (childs-1):
1545                     self.cfg += ", "
1546                     if not self.second:
1547                         self.sdg += ", "
1548                 start += 4
1549             self.cfg += "}"
1550             if not self.second:
1551                 self.sdg  += "}"
1552         elif childs == 4:
1553             self.cfg += " = {}"
1554             if not self.second:
1555                 self.sdg += " = {}"
1556         pass
1557 
1558     def atrib(self,tree):
1559         if not self.second:
1560             self.atribID += 1
1561         self.cfg += "\"atrib_" + str(self.atribID) + " " + tree.children[0].value + "
                 = "
1562         self.sdg += "\"atrib_" + str(self.atribID) + " " + tree.children[0].value + "
                 = "
```

```
1563
1564            self.visit(tree.children[2])
1565
1566            self.second = True
1567            self.cfg += "\"\n\t\"atrib_" + str(self.atribID) + " " + tree.children[0].
                    value + " = "
1568            self.sdg += "\"\n\t"
1569            self.visit(tree.children[2])
1570            self.cfg += "\" -> "
1571            self.second = False
1572
1573            pass
1574
1575        def initcicle(self, tree):
1576            if not self.second:
1577                self.initcicleID += 1
1578            self.cfg += "initcicle_" + str(self.initcicleID) + " " + tree.children[0].
                    value + " = "
1579            #self.visit(tree.children[2])
1580            if not self.second:
1581                self.sdg += "initcicle_" + str(self.initcicleID) + " " + tree.children
                        [0].value + " = "
1582            self.visit(tree.children[2])
1583
1584            pass
1585
1586        def print(self,tree):
1587
1588            if not self.second:
1589                self.printID += 1
1590
1591            s = tree.children[1].value.replace('\"','\'')
1592
1593            self.cfg += "\"print_" + str(self.printID) + " print(" + s + ")\"\n\t"
1594            self.sdg += "\"print_" + str(self.printID) + " print(" + s + ")\"\n\t"
1595
1596            self.cfg += "\"print_" + str(self.printID) + " print(" + s + ")\" -> "
1597
1598            pass
1599
1600        def read(self,tree):
1601
1602            if not self.second:
1603                self.readID += 1
1604
1605            self.cfg += "\"read_" + str(self.readID) + " read(" + tree.children[1].value
                    + ")\"\n\t"
1606            self.sdg += "\"read_" + str(self.readID) + " read(" + tree.children[1].value
                    + ")\"\n\t"
1607
1608            self.cfg += "\"read_" + str(self.readID) + " read(" + tree.children[1].value
                    + ")\" -> "
1609            pass
1610
```

```
1611    def cond(self,tree):
1612        self.incicle = True
1613
1614        if not self.second:
1615            self.ifID += 1
1616        self.cfg += "\"if_" + str(self.ifID) + "_start if("
1617        self.sdg += "\"if_" + str(self.ifID) + " if("
1618
1619        l = len(tree.children)
1620        self.visit(tree.children[2])
1621
1622        self.cfg += ")\"\n\t\"if_" + str(self.ifID) + "_start if("
1623        self.sdg += ")\"\n\t\"if_" + str(self.ifID) + " if("
1624
1625        #self.second = True
1626        self.visit(tree.children[2])
1627        #self.second = False
1628
1629        self.cfg += ")\" -> "
1630        self.sdg += ")\" -> \"then" + str(self.ifID) + "\"\n\t"
1631
1632        self.incicle = True
1633        # body
1634        #self.visit(tree.children[4].children[1])
1635        for c in tree.children[4].children[1].children:
1636            #print(c.pretty())
1637            self.sdg += "\"then" + str(self.ifID) + "\" -> "
1638            self.visit(c)
1639        #fim do if
1640        self.incicle = False
1641
1642
1643        self.cfg += "\"if_" + str(self.ifID) + "_end if("
1644        self.second = True
1645        self.visit(tree.children[2])
1646
1647        self.cfg += ")\"\n\t"
1648        self.second = False
1649
1650        if(tree.children[(l-2)] == "else"):
1651            self.cfg += "\"if_" + str(self.ifID) + "_start if("
1652            self.sdg += "\"if_" + str(self.ifID) + " if("
1653            self.visit(tree.children[2])
1654            self.cfg += ")\" -> "
1655            self.sdg += ")\" -> \"else" + str(self.ifID) + "\"\n\t"
1656
1657            self.incicle = True
1658            for c in tree.children[(l-1)].children[1].children:
1659                self.sdg += "\"else" + str(self.ifID) + "\" -> "
1660                self.visit(c)
1661            #self.visit(tree.children[(l-1)])
1662            self.incicle = False
1663
1664            self.second = True
```

```
1665            self.cfg += "\"if_" + str(self.ifID) + "_end if("
1666            self.visit(tree.children[2])
1667            self.cfg += ")\"\n\t"
1668            self.second = False
1669
1670        else:
1671            self.second = True
1672            self.cfg += "\"if_" + str(self.ifID) + "_start if("
1673            self.visit(tree.children[2])
1674            self.cfg += ")\" -> \"if_" + str(self.ifID) + "_end if("
1675            self.visit(tree.children[2])
1676            self.cfg += ")\"\n\t"
1677            self.second = False
1678
1679        self.second = True
1680        self.cfg += "\"if_" + str(self.ifID) + "_end if("
1681
1682        self.visit(tree.children[2])
1683
1684        self.cfg += ")\" -> "
1685        self.second = False
1686
1687        pass
1688
1689    def ciclewhile(self,tree):
1690        self.whileID += 1
1691
1692        self.cfg += "\"while_" + str(self.whileID) + "_start while("
1693        self.sdg += "\"while_" + str(self.whileID) + " while("
1694        self.visit(tree.children[2])
1695        self.cfg += ")\"\n\t\"while_" + str(self.whileID) + "_start while("
1696        self.sdg += ")\"\n\t"
1697        self.second = True
1698        self.visit(tree.children[2])
1699        self.cfg += ")\" -> \"while_" + str(self.whileID) + "_end while("
1700        self.visit(tree.children[2])
1701        self.cfg += ")\"\n\t\"while_" + str(self.whileID) + "_start while("
1702        self.visit(tree.children[2])
1703        self.cfg += ")\" -> "
1704        self.second = False
1705
1706        for c in tree.children[4].children[1].children:
1707            self.sdg += "\"while_" + str(self.whileID) + " while("
1708            self.visit(tree.children[2])
1709            self.sdg += ")\" -> "
1710            self.visit(c)
1711
1712        self.second = True
1713        self.cfg += "\"while_" + str(self.whileID) + "_start while("
1714        self.visit(tree.children[2])
1715        self.cfg += ")\"\n\t\"while_" + str(self.whileID) + "_end while("
1716        self.visit(tree.children[2])
1717        self.cfg += ")\" -> "
1718        self.second = False
```

```
1719
1720
1721        pass
1722
1723    def ciclerepeat(self,tree):
1724        if not self.second:
1725            self.repeatID += 1
1726
1727        self.cfg += "\"repeat_" + str(self.repeatID) + "_start repeat(" + tree.
               children[2].value
1728        self.sdg += "\"repeat_" + str(self.repeatID) + " repeat(" + tree.children[2].
               value + ")\"\n\t"
1729        self.cfg += ")\"\n\t\"repeat_" + str(self.repeatID) + "_start repeat(" + tree
               .children[2].value
1730        self.cfg += ")\" -> \"repeat_" + str(self.repeatID) + "_end repeat(" + tree.
               children[2].value
1731        self.cfg += ")\"\n\t\"repeat_" + str(self.repeatID) + "_start repeat(" + tree
               .children[2].value
1732        self.cfg += ")\" -> "
1733
1734        for c in tree.children[4].children[1].children:
1735            self.sdg += "\"repeat_" + str(self.repeatID) + " repeat(" + tree.children
                   [2] + ")\" -> "
1736            self.visit(c)
1737
1738        self.cfg += "\"repeat_" + str(self.repeatID) + "_start repeat(" + tree.
               children[2].value
1739        self.cfg += ")\"\n\t\"repeat_" + str(self.whileID) + "_end repeat(" + tree.
               children[2].value
1740        self.cfg += ")\" -> "
1741
1742        pass
1743
1744    def ciclefor(self,tree):
1745        if not self.second:
1746            self.forID += 1
1747
1748        self.cfg += "\"for_" + str(self.forID) + "_start\"\n\t\"for_" + str(self.
               forID) + "_start\" -> "
1749
1750        forcicle = {}
1751
1752        forcicle["op"] = []
1753        forcicle["initcicle"] = []
1754        forcicle["body"] = []
1755        forcicle["inc"] = []
1756        forcicle["dec"] = []
1757
1758        for c in tree.children:
1759            if c != "for" and c != "(" and c != ")" and c != ";" and c != ",":
1760                forcicle[c.data].append(c)
1761
1762        for c in forcicle["initcicle"]:
1763            self.cfg += "\""
```

```
            self.sdg += "\""
            self.visit(c)
            self.cfg += "\"\n\t\""
            self.sdg += "\"\n\t\"entry\" -> "
            self.second = True
            self.visit(c)
            self.cfg += "\" -> "
            self.second = False


        self.sdg += "\"for_" + str(self.forID) + " for("
        self.only = True
        self.visit(forcicle["op"][0])
        self.only = False
        self.sdg += ")\"\n\t"

        for c in forcicle["op"]:
            self.second = True
            self.cfg += "\"for_" + str(self.forID) + "_cond ("
            self.visit(c)
            self.cfg += ")\"\n\t\"for_" + str(self.forID) + "_cond ("
            self.visit(c)
            self.cfg += ")\" -> \"for_" + str(self.forID) + "_end\"\n\t"
            self.cfg += "\"for_" + str(self.forID) + "_cond ("
            self.visit(c)
            self.cfg += ")\" -> "
            self.second = False

        for c in forcicle["body"]:
            for t in c.children[1].children:
                self.sdg += "\"for_" + str(self.forID) + " for("
                self.only = True
                self.visit(forcicle["op"][0])
                self.sdg += ")\" -> "
                self.only = False
                self.visit(t)
        i = 0
        for c in forcicle["inc"]:
            if i == 0:
                self.sdg += "\"for_" + str(self.forID) + " for("
            else:
                self.sdg += "\n\t\"for_" + str(self.forID) + " for("
            self.only = True
            self.visit(forcicle["op"][0])
            self.only = False
            self.sdg += ")\" -> \""
            self.cfg += "\""
            self.visit(c)
            self.cfg += "\"\n\t\""
            self.sdg += "\""
            self.second = True
            self.visit(c)
            self.cfg += "\" -> "
            self.second = False
```

```
1818
1819
1820        for c in forcicle["dec"]:
1821            self.sdg += "\n\t\"for_" + str(self.forID) + " for("
1822            self.only = True
1823            self.visit(forcicle["op"][0])
1824            self.only = False
1825            self.sdg += ")\" -> \""
1826            self.cfg += "\""
1827            self.visit(c)
1828            self.cfg += "\"\n\t\""
1829            self.sdg += "\""
1830            self.second = True
1831            self.visit(c)
1832            self.cfg += "\" -> "
1833            self.second = False
1834
1835        for c in forcicle["op"]:
1836            self.second = True
1837            self.cfg += "\"for_" + str(self.forID) + "_cond ("
1838            self.visit(c)
1839            self.cfg += ")\"\n\t\"for_" + str(self.forID) + "_end\" -> "
1840            self.second = False
1841
1842        pass
1843
1844    def inc(self, tree):
1845        if not self.second:
1846            self.incID += 1
1847        self.cfg += "inc_" + str(self.incID) + " " + tree.children[0] + "++"
1848        if not self.second:
1849            self.sdg += "inc_" + str(self.incID) + " " + tree.children[0] + "++"
1850
1851        pass
1852
1853    def dec(self, tree):
1854        if not self.second:
1855            self.decID += 1
1856        self.cfg += "dec_" + str(self.decID) + " " + tree.children[0] + "--"
1857        if not self.second:
1858            self.sdg += "dec_" + str(self.decID) + " " + tree.children[0] + "--"
1859
1860        pass
1861
1862    def op(self,tree):
1863        if(len(tree.children) > 1):
1864            if(tree.children[0] == "!"):
1865                if not self.only:
1866                    self.cfg += "!"
1867                if not self.second:
1868                    self.sdg += "!"
1869                self.visit(tree.children[1])
1870            elif(tree.children[1] == "&"):
1871                self.visit(tree.children[0])
```

```
1872                 if not self.only:
1873                     self.cfg += " & "
1874                 if not self.second:
1875                     self.sdg += " & "
1876                 self.visit(tree.children[2])
1877             elif(tree.children[1] == "#"):
1878                 t1 = self.visit(tree.children[0])
1879                 if not self.only:
1880                     self.cfg += " # "
1881                 if not self.second:
1882                     self.sdg += " # "
1883                 t2 = self.visit(tree.children[2])
1884         else:
1885             self.visit(tree.children[0])
1886 
1887     def factcond(self,tree):
1888         if len(tree.children) > 1:
1889             self.visit(tree.children[0])
1890             if not self.only:
1891                 self.cfg += " " + tree.children[1].value + " "
1892             if not self.second:
1893                     self.sdg += " " + tree.children[1].value + " "
1894             self.visit(tree.children[2])
1895         else:
1896             self.visit(tree.children[0])
1897 
1898     def expcond(self,tree):
1899         if len(tree.children) > 1:
1900             self.visit(tree.children[0])
1901             if not self.only:
1902                 self.cfg += " " + tree.children[1].value + " "
1903             if not self.second:
1904                     self.sdg += " " + tree.children[1].value + " "
1905             self.visit(tree.children[2])
1906         else:
1907             self.visit(tree.children[0])
1908 
1909     def termocond(self,tree):
1910         if len(tree.children) > 1:
1911             self.visit(tree.children[0])
1912             if not self.only:
1913                 self.cfg += " " + tree.children[1].value + " "
1914             if not self.second:
1915                     self.sdg += " " + tree.children[1].value + " "
1916             self.visit(tree.children[2])
1917         else:
1918             self.visit(tree.children[0])
1919 
1920     def factor(self,tree):
1921         r = None
1922         if tree.children[0].type == 'SIGNED_INT':
1923             r = int(tree.children[0])
1924             if not self.only:
1925                 self.cfg += str(r)
```

```
1926             if not self.second:
1927                 self.sdg += str(r)
1928         elif tree.children[0].type == 'VARNAME':
1929             if not self.only:
1930                 self.cfg += tree.children[0].value
1931             if not self.second:
1932                 self.sdg += tree.children[0].value
1933         elif tree.children[0].type == 'DECIMAL':
1934             r = float(tree.children[0])
1935             if not self.only:
1936                 self.cfg += str(r)
1937             if not self.second:
1938                 self.sdg += str(r)
1939         elif tree.children[0] == "(":
1940             self.visit(tree.children[1])
1941
1942 grammar = '''
1943 start: BEGIN program END
1944 program: instruction+
1945 instruction: declaration | comment | operation
1946 declaration: atomic | structure
1947 operation: atrib | print | read | cond | cicle
1948 print: "print" PE (VARNAME | ESCAPED_STRING) PD PV
1949 read: "read" PE VARNAME PD PV
1950 cond: IF PE op PD body (ELSE body)?
1951 cicle: ciclewhile | ciclefor | ciclerepeat
1952 ciclewhile: WHILE PE op PD body
1953 WHILE: "while"
1954 ciclefor: FOR PE (initcicle (VIR initcicle)*)? PV op PV ((inc | dec) (VIR (inc | dec)
        )*)? PD body
1955 initcicle: VARNAME EQUAL op
1956 FOR: "for"
1957 ciclerepeat: REPEAT PE (SIGNED_INT | VARNAME) PD body
1958 REPEAT: "repeat"
1959 body: open program close
1960 atrib: VARNAME EQUAL op PV
1961 inc: VARNAME INC
1962 INC: "++"
1963 dec: VARNAME DEC
1964 DEC: "--"
1965 op: NOT op | op (AND | OR) factcond | factcond
1966 NOT: "!"
1967 AND: "&"
1968 OR: "#"
1969 factcond: factcond BINSREL expcond | expcond
1970 BINSREL: LESSEQ | LESS | MOREEQ | MORE | EQ | DIFF
1971 LESSEQ: "<="
1972 LESS: "<"
1973 MOREEQ: ">="
1974 MORE: ">"
1975 EQ: "=="
1976 DIFF: "!="
1977 expcond: expcond (PLUS | MINUS) termocond | termocond
1978 PLUS: "+"
```

```
1979 MINUS: "-"
1980 termocond: termocond (MUL|DIV|MOD) factor | factor
1981 MUL: "*"
1982 DIV: "/"
1983 MOD: "%"
1984 factor: PE op PD | SIGNED_INT | VARNAME | DECIMAL
1985 atomic: TYPEATOMIC VARNAME (EQUAL elem)? PV
1986 structure: (set | list | dict | tuple) PV
1987 set: "set" VARNAME (EQUAL OPENBRACKET (elem (VIR elem)*)? CLOSEBRACKET)?
1988 dict: "dict" VARNAME (EQUAL OPENBRACKET (elem DD elem (VIR elem DD elem)*)?
         CLOSEBRACKET)?
1989 list: "list" VARNAME (EQUAL OPENSQR (elem (VIR elem)*)? CLOSESQR)?
1990 tuple: "tuple" VARNAME (EQUAL PE (elem (VIR elem)*)? PD)?
1991 elem: ESCAPED_STRING | SIGNED_INT | DECIMAL | op
1992 TYPEATOMIC: "int" | "float" | "string"
1993 VARNAME: WORD
1994 comment: C_COMMENT
1995 BEGIN: "-{"
1996 END: "}-"
1997 PV: ";"
1998 VIR: ","
1999 OPENBRACKET: "{"
2000 CLOSEBRACKET: "}"
2001 OPENSQR: "["
2002 CLOSESQR: "]"
2003 DD: ":"
2004 PE: "("
2005 PD: ")"
2006 EQUAL: "="
2007 open: OPEN
2008 OPEN: "{"
2009 close: CLOSE
2010 CLOSE: "}"
2011 IF: "if"
2012 ELSE: "else"
2013
2014
2015 %import common.WORD
2016 %import common.SIGNED_INT
2017 %import common.DECIMAL
2018 %import common.WS
2019 %import common.ESCAPED_STRING
2020 %import common.C_COMMENT
2021 %ignore WS
2022 '''
2023
2024 parserLark = Lark(grammar)
2025 f = open("teste1.txt")
2026 example = f.read()
2027 parse_tree = parserLark.parse(example)
2028 data = MyInterpreter().visit(parse_tree)
2029
2030 def geraHTML(atomic_vars, struct_vars, warnings, errors, nrStructs, instrucoes,
         output_html, control):
```

```
2031    output_html.write("<!DOCTYPE html>")
2032    output_html.write("<html lang=\"pt\">")
2033    output_html.write("<head>")
2034    output_html.write("<meta charset=\"UTF-8\">")
2035    output_html.write("<link rel=\"stylesheet\" href=\"https://www.w3schools.com/
            w3css/4/w3.css\">")
2036    output_html.write("<title>EG - TP2</title>")
2037    output_html.write("</head>")
2038
2039    output_html.write("<body>")
2040
2041    navbar = '''
2042    <div class="w3-top">
2043            <div class="w3-bar w3-yellow intronav">
2044                <header>
2045                        <a href="output.html" class="w3-bar-item w3-button w3-hover-
                                black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">An lise do C digo</a>
2046                        <a href="codeHTML.html" class="w3-bar-item w3-button w3-hover
                                -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">C digo Original</a>
2047                        <a href="sugestao.html" class="w3-bar-item w3-button w3-hover
                                -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">Sugest o If's</a>
2048                </header>
2049            </div>
2050        </div>
2051    '''
2052
2053    output_html.write(navbar)
2054
2055    output_html.write("<h1> Tabela com todas as vari veis at micas do programa </h1
            >")
2056    output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2057    output_html.write("<tr class=\"w3-yellow\">")
2058    output_html.write("<th>Vari vel</th>")
2059    output_html.write("<th>Tipo</th>")
2060    output_html.write("<th>Valor</th>")
2061    output_html.write("<th>Warnings</th>")
2062    output_html.write("<th>Erros</th>")
2063    output_html.write("</tr>")
2064
2065    for var in atomic_vars.keys():
2066        output_html.write("<tr>")
2067        output_html.write("<td>" + var + "</td>")
2068        output_html.write("<td>" + str(atomic_vars[var][0]) + "</td>")
2069        output_html.write("<td>" + str(atomic_vars[var][1]) + "</td>")
2070        if var in warnings.keys():
2071            if len(warnings[var]) == 0:
2072                output_html.write("<td>Sem warnings associados</td>")
2073            else:
2074                w = ""
2075                for string in warnings[var]:
2076                    w += string + "\n"
```

```
2077                output_html.write("<td>" + w + "</td>")
2078
2079            if var in errors.keys():
2080                if len(errors[var]) == 0:
2081                    output_html.write("<td>Sem erros associados</td>")
2082                else:
2083                    erros = ""
2084                    for erro in errors[var]:
2085                        erros += erro + " "
2086                    output_html.write("<td>" + erros + "</td>")
2087
2088            output_html.write("</tr>")
2089        output_html.write("</table>")
2090
2091        output_html.write("<h1> Tabela com todas as estruturas do programa </h1>")
2092        output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2093        output_html.write("<tr class=\"w3-yellow\">")
2094        output_html.write("<th>Vari vel</th>")
2095        output_html.write("<th>Tipo</th>")
2096        output_html.write("<th>Tamanho</th>")
2097        output_html.write("<th>Valor</th>")
2098        output_html.write("<th>Warnings</th>")
2099        output_html.write("</tr>")
2100
2101        for var in struct_vars.keys():
2102            output_html.write("<tr>")
2103            output_html.write("<td>" + var + "</td>")
2104            output_html.write("<td>" + str(struct_vars[var][0]) + "</td>")
2105            output_html.write("<td>" + str(struct_vars[var][1]) + "</td>")
2106            output_html.write("<td>" + str(struct_vars[var][2]) + "</td>")
2107
2108            if var in warnings.keys():
2109                if len(warnings[var]) == 0:
2110                    output_html.write("<td>Sem warnings associados</td>")
2111                else:
2112                    w = ""
2113                    for string in warnings[var]:
2114                        w += string + "\n"
2115                    output_html.write("<td>" + w + "</td>")
2116
2117            output_html.write("</tr>")
2118
2119        output_html.write("</table>")
2120
2121        output_html.write("<h1> Total de vari veis do programa: " + str(len(atomic_vars.
               keys()) + len(struct_vars.keys())) + "</h1>")
2122
2123        output_html.write("<h1> Tipos de dados estruturados usados </h1>")
2124        output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2125        output_html.write("<tr class=\"w3-yellow\">")
2126        output_html.write("<th>Tipo</th>")
2127        output_html.write("<th>N mero</th>")
2128        output_html.write("</tr>")
2129
```

```
2130    for type in nrStructs.keys():
2131        output_html.write("<tr>")
2132        output_html.write("<td>" + type + "</td>")
2133        output_html.write("<td>" + str(nrStructs[type]) + "</td>")
2134        output_html.write("</tr>")
2135
2136    output_html.write("</table>")
2137
2138    output_html.write("<h1> N mero total de instru   es </h1>")
2139    output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2140    output_html.write("<tr class=\"w3-yellow\">")
2141    output_html.write("<th>Instru   o </th>")
2142    output_html.write("<th>N mero</th>")
2143    output_html.write("</tr>")
2144
2145    total = 0
2146
2147    for instrucao in instrucoes.keys():
2148        output_html.write("<tr>")
2149        output_html.write("<td>" + instrucao + "</td>")
2150        output_html.write("<td>" + str(instrucoes[instrucao]) + "</td>")
2151        output_html.write("</tr>")
2152        total += instrucoes[instrucao]
2153
2154    output_html.write("<td>Total</td>")
2155    output_html.write("<td>" + str(total) + "</td>")
2156    output_html.write("</table>")
2157
2158    ##
2159
2160    output_html.write("<h1> Estruturas de controlo </h1>")
2161    output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2162    output_html.write("<tr class=\"w3-yellow\">")
2163    output_html.write("<th>ID</th>")
2164    output_html.write("<th>Type</th>")
2165    output_html.write("<th>Parents</th>")
2166    output_html.write("</tr>")
2167
2168    total = 0
2169
2170    for c in control.keys():
2171        output_html.write("<tr>")
2172        output_html.write("<td>" + str(c) + "</td>")
2173        output_html.write("<td>" + str(control[c][0]) + "</td>")
2174        if len(control[c][2]) == 0:
2175            output_html.write("<td>Sem parents associados</td>")
2176        else:
2177            id = ""
2178            for ids in control[c][2]:
2179                id += str(ids) + " | "
2180            output_html.write("<td>ID's dos ciclos associados: " + id + "</td>")
2181        output_html.write("</tr>")
2182        total += 1
2183
```

```
2184     output_html.write("</body>")
2185     output_html.write("</html>")
2186
2187 output_html = open("output.html", "w")
2188
2189 #1 e 2 e 3
2190 geraHTML(data["atomic_vars"],data["struct_vars"], data["warnings"], data["errors"],
       data["nrStructs"],
2191 data["instructions"] ,output_html, data["controlStructs"])
2192
2193 html_header = '''<!DOCTYPE html>
2194 <html>
2195     <style>
2196         .error {
2197             position: relative;
2198             display: inline-block;
2199             border-bottom: 1px dotted black;
2200             color: red;
2201         }
2202
2203         .code {
2204             position: relative;
2205             display: inline-block;
2206         }
2207
2208         .comment {
2209             position: relative;
2210             display: inline-block;
2211             color: grey;
2212         }
2213
2214         .error .errortext {
2215             visibility: hidden;
2216             width: 200px;
2217             background-color: #555;
2218             color: #fff;
2219             text-align: center;
2220             border-radius: 6px;
2221             padding: 5px 0;
2222             position: absolute;
2223             z-index: 1;
2224             bottom: 125%;
2225             left: 50%;
2226             margin-left: -40px;
2227             opacity: 0;
2228             transition: opacity 0.3s;
2229         }
2230
2231         .error .errortext::after {
2232             content: "";
2233             position: absolute;
2234             top: 100%;
2235             left: 20%;
2236             margin-left: -5px;
```

```
2237             border-width: 5px;
2238             border-style: solid;
2239             border-color: #555 transparent transparent transparent;
2240         }
2241 
2242         .error:hover .errortext {
2243             visibility: visible;
2244             opacity: 1;
2245         }
2246     </style>
2247     <head>
2248         <link rel=\"stylesheet\" href=\"https://www.w3schools.com/w3css/4/w3.css\">
2249         <title>EG - TP2</title>
2250     </head>
2251     '''
2252 
2253 navbar = '''
2254     <div class="w3-top">
2255             <div class="w3-bar w3-yellow intronav">
2256                 <header>
2257                         <a href="output.html" class="w3-bar-item w3-button w3-hover-
                                black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">An lise do C digo</a>
2258                         <a href="codeHTML.html" class="w3-bar-item w3-button w3-hover
                                -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">C digo Original</a>
2259                         <a href="sugestao.html" class="w3-bar-item w3-button w3-hover
                                -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                xlarge">Sugest o If's</a>
2260                 </header>
2261             </div>
2262         </div>
2263     '''
2264 
2265 html = html_header + "<body>\n" + navbar +data["html_body"] + "\n</body></html>"
2266 
2267 with open("codeHTML.html","w") as out:
2268     out.write(html)
2269 
2270 def geraSugestao(sugestoes, output_html):
2271     output_html.write("<!DOCTYPE html>")
2272     output_html.write("<html lang=\"pt\">")
2273     output_html.write("<head>")
2274     output_html.write("<meta charset=\"UTF-8\">")
2275     output_html.write("<link rel=\"stylesheet\" href=\"https://www.w3schools.com/
            w3css/4/w3.css\">")
2276     output_html.write("<title>EG - TP2</title>")
2277     output_html.write("</head>")
2278 
2279     output_html.write("<body>")
2280 
2281     navbar = '''
2282     <div class="w3-top">
2283             <div class="w3-bar w3-yellow intronav">
```

```
2284                 <header>
2285                         <a href="output.html" class="w3-bar-item w3-button w3-hover-
                                 black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                 xlarge">An lise do C digo</a>
2286                         <a href="codeHTML.html" class="w3-bar-item w3-button w3-hover
                                 -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                 xlarge">C digo Original</a>
2287                         <a href="sugestao.html" class="w3-bar-item w3-button w3-hover
                                 -black w3-padding-16 w3-text-black w3-hover-text-white w3-
                                 xlarge">Sugest o If's</a>
2288                 </header>
2289             </div>
2290         </div>
2291     '''
2292
2293     output_html.write(navbar)
2294
2295     output_html.write("<h1> Sugest es para If's aninhados</h1>")
2296     output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
2297     output_html.write("<tr class=\"w3-yellow\">")
2298     output_html.write("<th>Original</th>")
2299     output_html.write("<th>Sugest o</th>")
2300     output_html.write("</tr>")
2301
2302     for sugestao in sugestoes.keys():
2303         sug = sugestao.replace("\t","\" + \t \"")
2304         output_html.write("<tr>")
2305         output_html.write("<td><span style=\"white-space: pre-wrap\">" + sugestao +
                 "</span></td>")
2306         output_html.write("<td><span style=\"white-space: pre-wrap\">" + sugestoes[
                 sugestao] + "</span></td>")
2307         output_html.write("</tr>")
2308
2309     output_html.write("</table>")
2310     output_html.write("</body>")
2311     output_html.write("</html>")
2312
2313 output_html = open("sugestao.html", "w")
2314 geraSugestao(data["sugestoes"], output_html)
```

Apêndice B

Ficheiro de teste

```
1  -{
2      int a;
3      int b = 2;
4      float c;
5      float d = 3.4;
6      string e;
7      string f = "ola";
8
9      set g;
10     set h = {};
11     set i = {2, 3.4, "ola"};
12
13     list j;
14     list k = [];
15     list l = [2, 3.4, "ola"];
16
17     tuple m;
18     tuple n = ();
19     tuple o = (2, 3.4, "ola");
20
21     dict p;
22     dict q = {};
23     dict r = {1:"ola", 3.2:"mundo"};
24
25     a = 3 + 1;
26     e = "mundo";
27     c = 4.3 + 3.14;
28
29     print("oi");
30     read(a);
31
32     if(a==3){
33         a = 5;
34         a = 2;
35     }
36
37     if(a == 4){
38         a = 5;
39     } else {
```

```
40          a = 4;
41      }
42
43      while(a < 5){
44          a = a + 1;
45          a = 3;
46      }
47
48      if(a == 3){
49          if(b == 2){
50              print("ola mundo!");
51          }
52      }
53
54
55      repeat(5){
56          a = 2;
57      }
58      a = 5;
59
60      for(a = 0, b = 3; a < 5; a++,b--){
61          b = 4;
62          c = 0;
63      }
64  }-
```