Engenharia Gramatical (1º ano de MEI)

Analisador de Código Fonte

Relatório de Desenvolvimento

André Martins
PG47009

José Ferreira
PG47375

25 de abril de 2022

Resumo

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

Conteúdo

Capítulo 1

Introdução

Supervisor: Pedro Rangel Henriques

1.1 Analisador 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 analisar código de uma linguagem imperativa definida pelo grupo de trabalho. Essa mesma linguagem teria que permitir declarar diversas variáveis atómicas e estruturadas (tais como conjuntos, listas, tuplos e dicionário), instruções condicionais e 3 variantes de ciclos, sendo elas os **ciclos for, while e repeat**.
Após a definição da gramática para a implementação da nossa linguagem e de todas as suas funcionalidades, recorremos ao módulo de geração de processadores de linguagens **Lark.Interpreter**, gerando como resultado final 3 ficheiros HTML com todas as informações pertinentes acerca do código analisado, bem como sugestões, avisos e erros encontrados durante essa mesma análise. Como resultados principais da análise realizada podemos salientar os seguintes:

- Lista de todas as variaveis do programa indicando os casos de: redeclaração, não-declaração, variáveis usadas mas não inicializadas e variáveis declaradas mas nunca mencionadas
- Total de variaveis declaradas versus os Tipos de dados estruturados usados.
- Total de instruções que formam o corpo do programa, indicandoo número de instruções de cada tipo (atribuições, leitura e escrita, condicionais e cíclicas).
- Total de situações em que estruturas de controlo surgem aninhadas em outras estruturas de controlo do mesmo ou de tipos diferentes.
- Informações acerca da presença de **ifs** aninhados indicando os casos em **ifs** aninhados possam ser substituídos por um só **if**.

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

Devido à grande dimensão da gramática em questão, iremos apresentar a mesma de uma forma mais resumida recorrendo a excertos da mesma.

```
start: BEGIN program END
program: instruction+
instruction: declaration | comment | operation
declaration: atomic | structure
operation: atrib | print | read | cond | cicle
cond: IF PE op PD body (ELSE body)?
cicle: ciclewhile | ciclefor | ciclerepeat
structure: (set | list | dict | tuple) PV
TYPEATOMIC: "int" | "float" | "string"
```

Uma vez que o principal objetivo do trabalho prático era a definição de uma linguagem imperativa simples, podemos defini-la como sendo um agregado de 1 ou mais instruções.
Como referenciado no excerto acima, cada instrução pode ser uma declaração, um comentário presente no código ou uma operação. As declarações podem ser de variáveis atómicas ou de variáveis estruturadas, tais como listas, tuplos, conjuntos e dicionários.
Por fim, as operações assumem-se como podendo ser uma atribuição, um print de valores ou strings, read de valores para variáveis declaradas, operações condicionais e 3 tipos de ciclo, sendo eles o for, while e repeat.

```
body: open program close
```

Para ser possível realizar as operações cíclicas e condicionais, foi definida a regra *body*, que representa um conjuntos de quaisquer instruções de código delimitadas por chavetas. Assim, é possível analisar todo o tipo de ciclos e operações condicionais, permitindo, também, realizar as sugestões de aninhamentos de *ifs* que estejam presentes no código.
De seguida será colocada a definição completa da gramática definida, contudo cremos ser possível com as regras apresentadas mostrar o "cerne"da nossa linguagem e perceber todas as operações que são possíveis realizar. Apesar de ser uma gramática extensa, rapidamente se percebe a logística e a forma de articulação da mesma, de como é possível construir um programa complexo e como é possível "navegar"dentro do mesmo, sendo possível realizar todo o tipo de análise estática aos elementos presentes.
Por fim, é possível apresentar um excerto de código sintaticamente correto e que a nossa ferramenta é capaz de analisar, e, de seguida, iremos apresentar a análise do mesmo bem como as sugestões e melhorias que podem ser feitas.

```
-{
/*atoms*/
int b = 1 + 1;
list l = [1,"ola", 3.2];
dict r = {1:"ola", 3.2:"mundo"};
while(a < 0){
    print("while");
}
for(a = 0; a < 20; a++){
print("oi");
}
repeat(5){
    print("ola");
}
if(a == 0){
    if(c == 3){
        print("olaMundo");
        for(a = 3; a == 3; a++){
            print("reset");
            if(c != 3){
                print(b);
            }
        }
    }
}
if(b==3){
    if(b==3){
        if(b==3){
            int x = 3 + 1;
            set teste = {};
            x = 5;
            print("3");
            if(x == 3){
                print("deu?");
            }
            for(x = 0; x < 5; x++){
                print("5");
            }
            repeat(6){
                print("repeat");
            }
        }
    }
}
}-
```

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
```

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 obter todos os resultados pretendidos. 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.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?)}
```

```
self.nrStructs = dict()
# NR_STRUCTS = {ID : (TYPE, ACTIVE, PARENT_STRUCTS)}
```

Iremos começar a explicar como eram guardadas todas as variáveis, e com que informação eram guardadas. Começando pelas variáveis atómicas, estas eram guardadas num dicionário cuja *key* seria o nome da variável e o *value* seria um tuplo com o seu tipo, valor associado e duas flags para controlo de erros e warnings, que indicavam se essa mesma variável havia sido inicializada e se era, em algum momento, utilizada.
De modo análogo, o armazenamento das variáveis estruturas era realizado dessa forma, tendo um campo adicional que indicada o tamanho da variável em questão.
De seguida, foi criado outro dicionário para interpretar as estruturas de controlo (operações condiconais e cíclicas) e analisar se estas estavam contidas dentro de outras estruturas de controlo.
Focando agora no tratamento de warnings e de erros, também foram criados 2 dicionários que a cada variável associavam os respetivos erros e warnings encontrados durante a análise do código. Assim, é fácil aceder a toda a informação de uma dada variável, uma vez que através do seu nome (parâmetro uníco e irrepetível durante o programa) é possível obter todo o tipo de informações que eram pretendidas.
Para termos de análise também existe um dicionário que guarda todos os tipos diferentes de instruções existentes bem como o número de ocorrências das mesmas.
Para a identificação e posterior sugestão da presença de ifs aninhados foi necessário recorrer a várias estruturas e variáveis auxiliares. Contudo, achamos ser de maior importância apresentar o raciocínio que guiou essa mesma prática. Primeiro, era necessário identificar os casos em que se poderia realmente sugerir o aninhamento das operações condicionais, uma vez que não era condição suficiente a existência de 2 ou mais operações do género. Assim, apenas era passível de sugestão uma sequência de 2 ou mais ifs estritamente consecutivos (sem a presença de outras instruções entre eles), sendo agrupadas todas as condições dos mesmos. Atingida a última operação condicional, era guardado o body da mesma, uma vez que seria este que iria ser executada após o cumprimento de todas as condições em questão. Após todo este processo de análise, era possível realizar o aninhamento dos ifs.
Apresentação de dois exemplos em que é possível aninhamento e em que não é possível aninhamento, respetivamente. Neste primeiro exemplo é possível aninhar os 3 primeiros ifs presentes no excerto de código.

```
if(b==3){
    if(b==3){
        if(b==3){
            int x = 3 + 1;
            set teste = {};
            x = 5;
            print("3");
            if(x == 3){
                print("deu?");
            }
            for(x = 0; x < 5; x++){
                print("5");
            }
            repeat(6){
                print("repeat");
            }
        }
    }
}
```

Já neste exemplo, o aninhamento de ifs não é realizado, uma vez que estes não são estritamente consecutivos.

```
if(c == 3){
        print("olaMundo");
            if(c != 3){
                print(b);
            }
    }
```

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 3 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. Por fim, 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.

4.1 Página HTML de análise de resultados

Variáveis atómicas

Variável	Tipo	Valor	Warnings	Erros
a	int	4	Sem warnings associados	Sem erros associados
b	int	2	Sem warnings associados	Sem erros associados
c	float	None	Sem warnings associados	Variable "c" was never initialized
d	float	3.4	Variable "d" was never used.	Sem erros associados
e	string	None	Variable "e" was never initialized nor used.	Sem erros associados
f	string	ola	Variable "f" was never used.	Sem erros associados
z	undefined	3	Sem warnings associados	Variable "z" was not declared
y	int	5	Sem warnings associados	Variable "y" declared more than once!
x	int	1	Sem warnings associados	Sem erros associados

Variáveis estruturadas e número total de variáveis

Tabela com todas as estruturas do programa

Variável	Tipo	Tamanho	Valor	Warnings
g	set	0	set()	Variable "g" was never used.
h	set	0	set()	Variable "h" was never used.
i	set	3	{1, 3.2, 'ola'}	Variable "i" was never used.
j	list	0	[]	Variable "j" was never used.
k	list	0	[]	Variable "k" was never used.
l	list	3	[1, 'ola', 3.2]	Variable "l" was never used.
m	tuple	0	()	Variable "m" was never used.
n	tuple	0	()	Variable "n" was never used.
o	tuple	3	(1, 'ola', 3.2)	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.
teste	set	0	set()	Variable "teste" was never used.

Total de variáveis do programa: 22

Tipos de dados estruturados

Tipos de dados estruturados usados

Tipo	Número
set	4
list	3
tuple	3
dict	3

Número total de instruções

Número total de instruções

Instrução	Número
atomic_declaration	9
atrib	13
structure_declaration	13
while	2
print	11
for	3
repeat	2
if	9
Total	62

Estruturas de controlo

Estruturas de controlo

ID	Type	Parents
0	while	Sem parents associados
1	for	Sem parents associados
2	repeat	Sem parents associados
3	if	Sem parents associados
4	if	ID's dos ciclos associados: 3 |
5	for	ID's dos ciclos associados: 3 | 4 |
6	if	ID's dos ciclos associados: 3 | 4 | 5 |
7	if	Sem parents associados
8	if	ID's dos ciclos associados: 7 |
9	while	ID's dos ciclos associados: 7 | 8 |
10	if	Sem parents associados
11	if	ID's dos ciclos associados: 10 |
12	if	ID's dos ciclos associados: 10 | 11 |
13	if	ID's dos ciclos associados: 10 | 11 | 12 |
14	for	ID's dos ciclos associados: 10 | 11 | 12 |
15	repeat	ID's dos ciclos associados: 10 | 11 | 12 |

4.2 Código original, com destaque de erros

Análise do Código Código Original Sugestão If's

```
-{

        /*atoms*/

        int a;

        a = 3;

        /*read(a);*/

        int b = 1 + 1;

        float c;

        float d = 3.4;

        string e;

        string f = "ola";

        z = 3;
```

```
if(a == 0){

        if(c == 3){


                print("olaMundo");

                for(a = 3; a == 3; a++){

                        print("reset");
                        Variable was never ini
                        if(c != 3){


                                print(b);

                        }

                }
```

4.3 Sugestão de If's

Análise do Código Código Original Sugestão If's

Original	Sugestão

if(a == 0){
  if(c == 3){
  print("olaMundo");
  for(a = 3; a == 3; a++){
  print("reset");
  if(c != 3){
  print(b);
  }
  }
 	
  }
}
	if((a == 0) & (c == 3)){
	  print("olaMundo");
	  for(a = 3; a == 3; a++){
	  print("reset");
	  if(c != 3){
	  print(b);
	  }
	  }
	}

if(a == 1){
  if(a == 0){
  print("coco");
  int y = 3 + 1;
  while(y < 5){
  y = y + 1;
  }
 	
  }
}
	if((a == 1) & (a == 0)){
	  print("coco");
	  int y = 3 + 1;
	  while(y < 5){
	  y = y + 1;
	  }
	}

Capítulo 5

Conclusão

Dado por concluído o segundo 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. 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 grammar = '''
1229 start: BEGIN program END
1230 program: instruction+
1231 instruction: declaration | comment | operation
1232 declaration: atomic | structure
1233 operation: atrib | print | read | cond | cicle
1234 print: "print" PE (VARNAME | ESCAPED_STRING) PD PV
1235 read: "read" PE VARNAME PD PV
1236 cond: IF PE op PD body (ELSE body)?
1237 cicle: ciclewhile | ciclefor | ciclerepeat
1238 ciclewhile: WHILE PE op PD body
1239 WHILE: "while"
1240 ciclefor: FOR PE (initcicle (VIR initcicle)*)? PV op PV (inc | dec (VIR (inc | dec))
        *)? PD body
1241 initcicle: VARNAME EQUAL op
1242 FOR: "for"
```

```
1243 ciclerepeat: REPEAT PE (SIGNED_INT | VARNAME) PD body
1244 REPEAT: "repeat"
1245 body: open program close
1246 atrib: VARNAME EQUAL op PV
1247 inc: VARNAME INC
1248 INC: "++"
1249 dec: VARNAME DEC
1250 DEC: "--"
1251 op: NOT op | op (AND | OR) factcond | factcond
1252 NOT: "!"
1253 AND: "&"
1254 OR: "#"
1255 factcond: factcond BINSREL expcond | expcond
1256 BINSREL: LESSEQ | LESS | MOREEQ | MORE | EQ | DIFF
1257 LESSEQ: "<="
1258 LESS: "<"
1259 MOREEQ: ">="
1260 MORE: ">"
1261 EQ: "=="
1262 DIFF: "!="
1263 expcond: expcond (PLUS | MINUS) termocond | termocond
1264 PLUS: "+"
1265 MINUS: "-"
1266 termocond: termocond (MUL|DIV|MOD) factor | factor
1267 MUL: "*"
1268 DIV: "/"
1269 MOD: "%"
1270 factor: PE op PD | SIGNED_INT | VARNAME | DECIMAL
1271 atomic: TYPEATOMIC VARNAME (EQUAL elem)? PV
1272 structure: (set | list | dict | tuple) PV
1273 set: "set" VARNAME (EQUAL OPENBRACKET (elem (VIR elem)*)? CLOSEBRACKET)?
1274 dict: "dict" VARNAME (EQUAL OPENBRACKET (elem DD elem (VIR elem DD elem)*)?
          CLOSEBRACKET)?
1275 list: "list" VARNAME (EQUAL OPENSQR (elem (VIR elem)*)? CLOSESQR)?
1276 tuple: "tuple" VARNAME (EQUAL PE (elem (VIR elem)*)? PD)?
1277 elem: ESCAPED_STRING | SIGNED_INT | DECIMAL | op
1278 TYPEATOMIC: "int" | "float" | "string"
1279 VARNAME: WORD
1280 comment: C_COMMENT
1281 BEGIN: "-{"
1282 END: "}-"
1283 PV: ";"
1284 VIR: ","
1285 OPENBRACKET: "{"
1286 CLOSEBRACKET: "}"
1287 OPENSQR: "["
1288 CLOSESQR: "]"
1289 DD: ":"
1290 PE: "("
1291 PD: ")"
1292 EQUAL: "="
1293 open: OPEN
1294 OPEN: "{"
1295 close: CLOSE
```

```
1296 CLOSE: "}"
1297 IF: "if"
1298 ELSE: "else"
1299
1300
1301 %import common.WORD
1302 %import common.SIGNED_INT
1303 %import common.DECIMAL
1304 %import common.WS
1305 %import common.ESCAPED_STRING
1306 %import common.C_COMMENT
1307 %ignore WS
1308 '''
1309
1310 parserLark = Lark(grammar)
1311 f = open("teste1.txt")
1312 example = f.read()
1313 parse_tree = parserLark.parse(example)
1314 data = MyInterpreter().visit(parse_tree)
1315
1316 def geraHTML(atomic_vars, struct_vars, warnings, errors, nrStructs, instrucoes,
        output_html, control):
1317     output_html.write("<!DOCTYPE html>")
1318     output_html.write("<html lang=\"pt\">")
1319     output_html.write("<head>")
1320     output_html.write("<meta charset=\"UTF-8\">")
1321     output_html.write("<link rel=\"stylesheet\" href=\"https://www.w3schools.com/
            w3css/4/w3.css\">")
1322     output_html.write("<title>EG - TP2</title>")
1323     output_html.write("</head>")
1324
1325     output_html.write("<body>")
1326
1327     navbar = '''
1328     <div class="w3-top">
1329             <div class="w3-bar w3-yellow intronav">
1330                 <header>
1331                         <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>
1332                         <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>
1333                         <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>
1334                 </header>
1335             </div>
1336         </div>
1337     '''
1338
1339     output_html.write(navbar)
1340
```

```
1341    output_html.write("<h1> Tabela com todas as vari veis at micas do programa </h1
            >")
1342    output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1343    output_html.write("<tr class=\"w3-yellow\">")
1344    output_html.write("<th>Vari vel</th>")
1345    output_html.write("<th>Tipo</th>")
1346    output_html.write("<th>Valor</th>")
1347    output_html.write("<th>Warnings</th>")
1348    output_html.write("<th>Erros</th>")
1349    output_html.write("</tr>")
1350
1351    for var in atomic_vars.keys():
1352        output_html.write("<tr>")
1353        output_html.write("<td>" + var + "</td>")
1354        output_html.write("<td>" + str(atomic_vars[var][0]) + "</td>")
1355        output_html.write("<td>" + str(atomic_vars[var][1]) + "</td>")
1356        if var in warnings.keys():
1357            if len(warnings[var]) == 0:
1358                output_html.write("<td>Sem warnings associados</td>")
1359            else:
1360                w = ""
1361                for string in warnings[var]:
1362                    w += string + "\n"
1363                output_html.write("<td>" + w + "</td>")
1364
1365        if var in errors.keys():
1366            if len(errors[var]) == 0:
1367                output_html.write("<td>Sem erros associados</td>")
1368            else:
1369                erros = ""
1370                for erro in errors[var]:
1371                    erros += erro + " "
1372                    output_html.write("<td>" + erros + "</td>")
1373
1374        output_html.write("</tr>")
1375    output_html.write("</table>")
1376
1377    output_html.write("<h1> Tabela com todas as estruturas do programa </h1>")
1378    output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1379    output_html.write("<tr class=\"w3-yellow\">")
1380    output_html.write("<th>Vari vel</th>")
1381    output_html.write("<th>Tipo</th>")
1382    output_html.write("<th>Tamanho</th>")
1383    output_html.write("<th>Valor</th>")
1384    output_html.write("<th>Warnings</th>")
1385    output_html.write("</tr>")
1386
1387    for var in struct_vars.keys():
1388        output_html.write("<tr>")
1389        output_html.write("<td>" + var + "</td>")
1390        output_html.write("<td>" + str(struct_vars[var][0]) + "</td>")
1391        output_html.write("<td>" + str(struct_vars[var][1]) + "</td>")
1392        output_html.write("<td>" + str(struct_vars[var][2]) + "</td>")
1393
```

```
1394            if var in warnings.keys():
1395                if len(warnings[var]) == 0:
1396                    output_html.write("<td>Sem warnings associados</td>")
1397                else:
1398                    w = ""
1399                    for string in warnings[var]:
1400                        w += string + "\n"
1401                    output_html.write("<td>" + w + "</td>")
1402
1403            output_html.write("</tr>")
1404
1405        output_html.write("</table>")
1406
1407        output_html.write("<h1> Total de vari veis do programa: " + str(len(atomic_vars.
               keys()) + len(struct_vars.keys())) + "</h1>")
1408
1409        output_html.write("<h1> Tipos de dados estruturados usados </h1>")
1410        output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1411        output_html.write("<tr class=\"w3-yellow\">")
1412        output_html.write("<th>Tipo</th>")
1413        output_html.write("<th>N mero</th>")
1414        output_html.write("</tr>")
1415
1416        for type in nrStructs.keys():
1417            output_html.write("<tr>")
1418            output_html.write("<td>" + type + "</td>")
1419            output_html.write("<td>" + str(nrStructs[type]) + "</td>")
1420            output_html.write("</tr>")
1421
1422        output_html.write("</table>")
1423
1424        output_html.write("<h1> N mero total de instru   es </h1>")
1425        output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1426        output_html.write("<tr class=\"w3-yellow\">")
1427        output_html.write("<th>Instru   o </th>")
1428        output_html.write("<th>N mero</th>")
1429        output_html.write("</tr>")
1430
1431        total = 0
1432
1433        for instrucao in instrucoes.keys():
1434            output_html.write("<tr>")
1435            output_html.write("<td>" + instrucao + "</td>")
1436            output_html.write("<td>" + str(instrucoes[instrucao]) + "</td>")
1437            output_html.write("</tr>")
1438            total += instrucoes[instrucao]
1439
1440        output_html.write("<td>Total</td>")
1441        output_html.write("<td>" + str(total) + "</td>")
1442        output_html.write("</table>")
1443
1444        ##
1445
1446        output_html.write("<h1> Estruturas de controlo </h1>")
```

```
1447     output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1448     output_html.write("<tr class=\"w3-yellow\">")
1449     output_html.write("<th>ID</th>")
1450     output_html.write("<th>Type</th>")
1451     output_html.write("<th>Parents</th>")
1452     output_html.write("</tr>")
1453 
1454     total = 0
1455 
1456     for c in control.keys():
1457         output_html.write("<tr>")
1458         output_html.write("<td>" + str(c) + "</td>")
1459         output_html.write("<td>" + str(control[c][0]) + "</td>")
1460         if len(control[c][2]) == 0:
1461             output_html.write("<td>Sem parents associados</td>")
1462         else:
1463             id = ""
1464             for ids in control[c][2]:
1465                 id += str(ids) + " | "
1466             output_html.write("<td>ID's dos ciclos associados: " + id + "</td>")
1467         output_html.write("</tr>")
1468         total += 1
1469 
1470     output_html.write("</body>")
1471     output_html.write("</html>")
1472 
1473 output_html = open("output.html", "w")
1474 
1475 #1 e 2 e 3
1476 geraHTML(data["atomic_vars"],data["struct_vars"], data["warnings"], data["errors"],
        data["nrStructs"],
1477 data["instructions"] ,output_html, data["controlStructs"])
1478 
1479 html_header = '''<!DOCTYPE html>
1480 <html>
1481     <style>
1482         .error {
1483             position: relative;
1484             display: inline-block;
1485             border-bottom: 1px dotted black;
1486             color: red;
1487         }
1488 
1489         .code {
1490             position: relative;
1491             display: inline-block;
1492         }
1493 
1494         .comment {
1495             position: relative;
1496             display: inline-block;
1497             color: grey;
1498         }
1499 
```

```
1500         .error .errortext {
1501             visibility: hidden;
1502             width: 200px;
1503             background-color: #555;
1504             color: #fff;
1505             text-align: center;
1506             border-radius: 6px;
1507             padding: 5px 0;
1508             position: absolute;
1509             z-index: 1;
1510             bottom: 125%;
1511             left: 50%;
1512             margin-left: -40px;
1513             opacity: 0;
1514             transition: opacity 0.3s;
1515         }
1516 
1517         .error .errortext::after {
1518             content: "";
1519             position: absolute;
1520             top: 100%;
1521             left: 20%;
1522             margin-left: -5px;
1523             border-width: 5px;
1524             border-style: solid;
1525             border-color: #555 transparent transparent transparent;
1526         }
1527 
1528         .error:hover .errortext {
1529             visibility: visible;
1530             opacity: 1;
1531         }
1532     </style>
1533     <head>
1534         <link rel=\"stylesheet\" href=\"https://www.w3schools.com/w3css/4/w3.css\">
1535         <title>EG - TP2</title>
1536     </head>
1537     '''
1538 
1539 navbar = '''
1540     <div class="w3-top">
1541             <div class="w3-bar w3-yellow intronav">
1542                 <header>
1543                         <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>
1544                         <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>
1545                         <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>
1546                 </header>
1547             </div>
```

```
1548         </div>
1549     '''
1550
1551 html = html_header + "<body>\n" + navbar +data["html_body"] + "\n</body></html>"
1552
1553 with open("codeHTML.html","w") as out:
1554     out.write(html)
1555
1556 def geraSugestao(sugestoes, output_html):
1557     output_html.write("<!DOCTYPE html>")
1558     output_html.write("<html lang=\"pt\">")
1559     output_html.write("<head>")
1560     output_html.write("<meta charset=\"UTF-8\">")
1561     output_html.write("<link rel=\"stylesheet\" href=\"https://www.w3schools.com/
            w3css/4/w3.css\">")
1562     output_html.write("<title>EG - TP2</title>")
1563     output_html.write("</head>")
1564
1565     output_html.write("<body>")
1566
1567     navbar = '''
1568     <div class="w3-top">
1569             <div class="w3-bar w3-yellow intronav">
1570                 <header>
1571                         <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>
1572                         <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>
1573                         <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>
1574                 </header>
1575             </div>
1576         </div>
1577     '''
1578
1579     output_html.write(navbar)
1580
1581     output_html.write("<h1> Sugest es para If's aninhados</h1>")
1582     output_html.write("<table class=\"w3-table w3-table-all w3-hoverable\">")
1583     output_html.write("<tr class=\"w3-yellow\">")
1584     output_html.write("<th>Original</th>")
1585     output_html.write("<th>Sugest o</th>")
1586     output_html.write("</tr>")
1587
1588     for sugestao in sugestoes.keys():
1589         sug = sugestao.replace("\t","\" + \t \"")
1590         output_html.write("<tr>")
1591         output_html.write("<td><span style=\"white-space: pre-wrap\">" + sugestao +
                "</span></td>")
1592         output_html.write("<td><span style=\"white-space: pre-wrap\">" + sugestoes[
                sugestao] + "</span></td>")
```

```
1593             output_html.write("</tr>")
1594
1595         output_html.write("</table>")
1596         output_html.write("</body>")
1597         output_html.write("</html>")
1598
1599 output_html = open("sugestao.html", "w")
1600 geraSugestao(data["sugestoes"], output_html)
```

Apêndice B

Ficheiro de teste

```
1  -{
2  /*atoms*/
3  int a;
4  a = 3;
5  /*read(a);*/
6  int b = 1 + 1;
7  float c;
8  float d = 3.4;
9  string e;
10 string f = "ola";
11
12 z = 3;
13
14 /*structures*/
15 set g;
16 set h = {};
17 set i = {1,"ola", 3.2};
18
19 list j;
20 list k = [];
21 list l = [1,"ola", 3.2];
22
23 tuple m;
24 tuple n = ();
25 tuple o = (1,"ola", 3.2);
26
27 int y = 3;
28
29 dict p;
30 dict q = {};
31 dict r = {1:"ola", 3.2:"mundo"};
32
33 while(a < 0){
34     print("while");
35 }
36
37 for(a = 0; a < 20; a++){
38 print("oi");
39 }
```

```
40 repeat(5){
41     print("ola");
42 }
43
44 if(a == 0){
45     if(c == 3){
46         print("olaMundo");
47         for(a = 3; a == 3; a++){
48             print("reset");
49             if(c != 3){
50                 print(b);
51             }
52         }
53     }
54 }
55 if(a == 1){
56     if(a == 0){
57         print("coco");
58         int y = 3 + 1;
59         while(y < 5){
60             y = y + 1;
61         }
62     }
63 }
64
65 if(b==3){
66     if(b==3){
67         if(b==3){
68             int x = 3 + 1;
69             set teste = {};
70             x = 5;
71             print("3");
72             if(x == 3){
73                 print("deu?");
74             }
75             for(x = 0; x < 5; x++){
76                 print("5");
77             }
78             repeat(6){
79                 print("repeat");
80             }
81         }
82     }
83 }
84 }-
```