

Perfil de EL - Engenharia de Linguagens (1º ano do MEI)
Trabalho Prático 2
Relatório de Desenvolvimento

Gonçalo Freitas
(PG50398)

Henrique Alvelos
(PG50414)

1 de maio de 2023

Resumo

Após no 1º Trabalho prático, estudarmos várias ferramentas avançadas de análise de código, passamos agora a desenvolver um Analisador de código para uma evolução da Linguagem de Programação Imperativa (LPI), definida anteriormente por nós.

Conteúdo

1	Introdução	2
1.1	Enquadramento	2
1.2	Contexto	2
1.3	Problema	3
1.4	Objetivo	3
1.5	Estrutura do Relatório	3
2	Concepção/desenho da Resolução	4
2.1	Estruturas de Dados	4
2.2	Algoritmos	4
3	Codificação e Testes	6
3.1	Alternativas de Implementação	6
3.2	Testes realizados e Resultados	6
3.2.1	Teste 1	6
3.2.2	Teste 2	8
3.2.3	Teste 3	9
4	Conclusão	12
4.1	Síntese do Documento	12
4.2	Análise crítica dos resultados	12
4.3	Trabalho futuro	12
A	Gramática Desenvolvida	13
B	Código do Programa	16

Capítulo 1

Introdução

1.1 Enquadramento

Tal como estudado no 1º Trabalho Prático, existem várias ferramentas avançadas de análise de código (ou seja, de programas-fonte em Linguagens de Programação de alto nível) com vista a ajudar em tarefas tais como:

- embelezar textualmente a escrita do programa
- detetar situações que infringem as boas práticas de codificação na linguagem em causa ou que podem ser vulneráveis durante a execução
- sugerir melhores formas de codificar sem alterar o significado do programa-fonte
- avaliar a performance do programa estática ou dinamicamente

1.2 Contexto

Este 2º Trabalho Prático, foi proposto pretende-se que desenvolva um Analisador de Código para a evolução de uma Linguagem de Programação Imperativa (LPI) anteriormente desenvolvida na resolução de um TPC proposto no início da UC de *Engenharia Gramatical*. Esta linguagem permite:

- declarar variáveis e manipular valores dos tipos Inteiro, Booleano, Array, Tuplo, String, Lista
- escrever instruções tais como Atribuição, Leitura, Escrita, Seleção (SE, CASO), Repetição (ENQ-FAZER, REPETIR-ATE, PARA-interv-FAZER)
- escrever expressões para definir valores com base em operadores associados aos tipos tais como operadores aritméticos, operadores lógicos, operadores relacionais, operadores de indexação de arrays, seleção de elementos de tuplos e de listas, inserção em listas, teste de pertença em listas.

1.3 Problema

Foi proposto o desenvolvimento de uma ferramenta em Python (usando o Parser e os Visitors do módulo para geração de processadores de linguagens Lark.Interpreter) que possua as seguintes funcionalidades:

1. analise programas escritos na sua linguagem LPI
2. gere em HTML um relatório com os resultados dessa análise
 - Lista de todas as variáveis do programa indicando os casos de: redeclaração ou não-declaração; variáveis usadas mas não inicializadas; variáveis declaradas e nunca mencionadas.
 - Total de variáveis declaradas versus os Tipos de dados estruturados usados.
 - Total de instruções que formam o corpo do programa, indicando o 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.
3. Em situações que envolvam SE aninhados, indicar se há a possibilidade desses aninhamentos poderem ser substituídos por um só SE.

1.4 Objetivo

Com este trabalho pretendemos ambientar-nos com a manipulação e análise de LPI's utilizando as ferramentas da livreria Lark.Interpreter do Python.

1.5 Estrutura do Relatório

No capítulo 2 são expostas as estruturas de dados e algoritmos utilizados para resolver o problema em questão.

No capítulo 3 discutimos as alternativas, decisões e problemas encontrados na implementação da nossa solução além de testes realizados e respetivos resultados.

No capítulo 4 termina-se o relatório com uma síntese do que foi dito, as conclusões e o trabalho futuro.

Capítulo 2

Concepção/desenho da Resolução

2.1 Estruturas de Dados

De forma a guardar e armazenar a informação necessária a realizar a análise proposta, utilizamos uma série de dicionários e variáveis dentro da class Gramática(Interpreter) que recebe um Lark Interpreter e trata um texto de entrada dado a gramática que estruturamos. Entre as estruturas de dados utilizadas destacam-se as seguintes:

vars Um dicionário em que as chaves são os id's das variáveis utilizadas no corpo principal e o valor é um dicionário com os campos *tipo*, *val* (valor), *dec* (declarado) e *init* (inicializado). Os dois últimos campos são contadores para o número de vezes que aquele *id* foi declarado/inicializado.

funcoes Um dicionário em que as chaves são os nomes das funções declaradas e tem com valores o campo *vars* (similar ao dicionário definido anteriormente), *params* (uma lista dos id's dos parametros passados) e *ret* (o valor de retorno da função).

func_id Uma variável que guarda o nome da função atual, no caso do Interpreter estar fora de uma função é definido a None.

instrucoes Um dicionário que serve maioritariamente como contador. Possui os campos *total* (total de instruções), *atrib* (total de atribuições), *escrita* (total de escritas), entre outros.

2.2 Algoritmos

Utilizamos uma travessia *inorder* para percorrer a árvore gerada ao realizar o parse do texto de *input*, devido a realizar este tipo de travessia escolhemos criar o texto de *output* o mais perto possível das folhas de forma à escrita acompanhá-la.

Ao estarmos a realizar uma travessia *inorder* foi necessário a utilização de variáveis de forma a manter um estado, por exemplo *func_id*. Consideramos isto um fator importante pois o tratamento de cada uma das regras definidas na gramática difere no caso de estarmos dentro de uma função ou não. Um exemplo desta diferença são as instruções de atribuição nas quais é necessário verificar se as variáveis utilizadas são parâmetros ou variáveis anteriormente declaradas na função, enquanto que fora das funções apenas é necessário verificar se são variáveis anteriormente declaradas.

Para calcular a possibilidade dos aninhamentos dos **SE** poderem ser substituídos por apenas um, definimos uma variável *SE_simple* que, inicialmente, é nula e só altera o seu valor, para verdadeiro, quando atingir a última condição desse ninho. A partir daí, nos restantes, verifica-se se o número de filhos é igual a 2 (contendo basicamente um **SE**). Quando é superior a 2, já não pode ser aninhado, pelo que a variável *SE_simple* fica com valor falso. Quando chegar ao último **SE**, esta variável volta a ser nula.

Capítulo 3

Codificação e Testes

3.1 Alternativas de Implementação

Uma das alternativas à nossa implementação teria sido, por exemplo, apenas criar o texto de saída no final de processar toda a árvore isto por um lado simplificaria a identificação de casos tais como a identificação de variáveis declaradas e nunca utilizadas contudo tornaria necessário a criação de uma estrutura de dados muito mais complexa.

3.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (ficheiros de *input*) e os respectivos resultados obtidos:

3.2.1 Teste 1

```
_func { a }  
  Int b 3  
  ATRIB a 3  
  SE b > 3  
    ATRIB a 4  
  ES  
RET a
```

```
ESCREVER a+1
```

```
Int a  
Int c  
LER a
```

```
SE b > 3
```



```

SE a > 4
  SE a > 4
    ATRIB b 3
    ATRIB a 9
  ES
ES
ES

CASO a
  5 : LER a
  6 : ESCREVER c
OSAC

```

Output:

Total	Valor	Total	Valor
Variáveis declaradas	3	Instruções	13
Tipos de dados	1	Atribuições	4
Funções	1	Escrita	2
Funções com parâmetros	1	Leitura	2
Funções com variáveis locais	1	Seleção	5
		Repetição	0
		Controlo aninhado	2
		Substituição por um SE	2

Figura 3.1: Resultado 1

```

Variável não declarada nem inicializada
SE b > 3
    SE a > 4
        SE a > 4
            ATRIB b 3
            ATRIB a 9
        ES
    ES
ES

```

Figura 3.2: Resultado 1

3.2.2 Teste 2

Input:

```

_id {a}
ATRIB a 5*a
RET a

```

Int a

Output:

Total	Valor	Total	Valor
Variáveis declaradas	2	Instruções	1
Típos de dados	1	Atribuições	1
Funções	1	Escrita	0
Funções com parâmetros	1	Leitura	0
Funções com variáveis locais	0	Seleção	0
		Repetição	0
		Controlo aninhado	0
		Substituição por um SE	0

Figura 3.3: Resultado 2

3.2.3 Teste 3

Input:

```
_id { a }
Int a
ATRIB a 5 * a
RET a
Int a 4
Boolean b
String c
Array d [2]
Tuplo e [3]
Lista f [4]
ATRIB a 5
ATRIB a a
ATRIB b True
ATRIB c "BOAS"
ATRIB d [2, 1]
ATRIB e (2, 1)
ATRIB f [2, 1]
LER a
ESCREVER a
SE b
ATRIB a 4
ES
CASO a
5 :
ESCREVER a
6 :
ESCREVER b
OSAC
ENQ a = 5 FAZER {
ESCREVER a
}
REPETIR
ESCREVER a
ATE b = 2
PARA 5 FAZER
{
ESCREVER a
}
ATRIB a 5 + 3
ESCREVER f
ESCREVER a
SE b = 3
```

```

ESCREVER a
ESCREVER b
ES
SE c = "BOAS"
SE d = [2, 1]
SE e = (2, 1)
ATRIB d False
ES
ES
ES

```

Total	Valor	Total	Valor
Variáveis declaradas	7	Instruções	31
Tipos de dados	6	Atribuições	11
Funções	1	Escrita	10
Funções com parâmetros	1	Leitura	1
Funções com variáveis locais	0	Seleção	6
		Repetição	3
		Controlo aninhado	2
		Substituição por um SE	2

Figura 3.4: Resultado 3

```
Variável já declarada mas não inicializada
Int a
ATRIB a 5 * a
RET a
Int a 4
Boolean b
String c
Array d [2]
Tuplo e [3]
Lista f [4]
ATRIB a 5
ATRIB a a
ATRIB b True
ATRIB c "BOAS"
ATRIB d [2, 1]
ATRIB e (2, 1)
ATRIB f [2, 1]
LER a
ESCREVER a
SE b
    ATRIB a 4
ES
CASO a
    5 :
        ESCREVER a
    6 :
        ESCREVER b
OSAC
```

Figura 3.5: Único Erro

Capítulo 4

Conclusão

4.1 Síntese do Documento

Ao longo deste trabalho foi desenvolvido um analisador de código capaz de extrair um conjunto de informações de uma LPI por nós definida utilizando as ferramentas da biblioteca Lark.Interpreter do Python.

4.2 Análise crítica dos resultados

O coletivo tem a noção de que poderia fazer melhor no tratamento das fases aninhadas com mais cuidado e mais opções; poderíamos melhorar na estrutura de dados usada para guardar as funções e as variáveis. Contudo, achamos que foi um trabalho bem conseguido, com uma página HTML organizada com várias cores e indentação que permite perceber com facilidade o código usado. Já na gramática, com as ajudas dadas da apresentação na aula e com uma melhor análise do que tínhamos anteriormente, sabemos que melhorou

4.3 Trabalho futuro

O grupo tem a opinião de que este projeto pode ser melhorado em várias vertentes. Poderá suportar erros de atribuição de valores a variáveis com tipos diferentes, guardar valores das variáveis e até verificar se determinadas estruturas de controlo são realmente necessárias, como é o caso de algumas condições. Para além disso, em termos de apresentação, temos a ideia de que a página HTML apresentada poderá ser melhorada tanto em termos de apresentação das estatísticas numa barra lateral, como realçar diretamente no código algumas das estatísticas.

Apêndice A

Gramática Desenvolvida

programa: funcao* item+

item: declaracao
| instrucao

declaracao: TIPO ID val?

instrucao: atrib
| leitura
| escrita
| selecao
| repeticao

atrib: "ATRIB" exp2 exp

leitura: "LER" exp2

escrita: "ESCREVER" exp

selecao: "SE" exp item+ "ES"
| "CASO" exp2 (val ":" item)+ "OSAC"

repeticao: "ENQ" exp "FAZER" "{" item+ "}"
| "REPETIR" item+ "ATE" exp
| "PARA" NUM "FAZER" "{" item+ "}"

funcao: FUNCAO_ID params item+ "RET" exp

exp: abs (op abs)*

exp2: ID ("." ID)*

abs: NEG? valor

NEG: "!"

op: OP
| RELACIONAL
| LOGICO

OP: "+"
| "-"
| "*"
| "/"
| "%"
| "^"
| "cons"
| "snoc"
| "in"

RELACIONAL : "="
| ">"
| ">="
| "<"
| "<="
| "!="

LOGICO : "&"
| "|"

valor: exp2
| val
| funcao_chamada
| ID "[" exp "]"
| ELEM exp2

ELEM: "head"
| "tail"

val: NUM
| STRING
| conjunto
| BOOL

conjunto: tuplo
| lista

tuplo: "(" val ("," val)* ")"


```

    | "(" ")"

lista: "[" val ("," val)* "]"
    | "[" "]"

BOOL: "True"
    | "False"

ID: /[a-z]+/
NUM: /[0-9]|([1-9][0-9]*)/
STRING: ESCAPED_STRING

TIPO: "Int"
    | "Boolean"
    | "String"
    | "Array"
    | "Tuplo"
    | "Lista"

funcao_chamada: FUNCAO_ID args

FUNCAO_ID: /_[a-z]+/
params: "{" ID ("," ID)* "}"
    | "{" "}"

args: "(" exp2 ("," exp2)* ")"
    | "(" ")"

%import common.WS
%import common.ESCAPED_STRING
%ignore WS

```

Apêndice B

Código do Programa

Listing B.1: CSS

```
1 :root {
2   --vscode-bg: #1E1E1E;
3   --vscode-fg: #D4D4D4;
4   --vscode-grey: #6A6A6A;
5   --vscode-blue: #569CD6;
6   --vscode-yellow: #DCDCAA;
7   --vscode-green: #B5CEA8;
8   --vscode-red: #D16969;
9   --vscode-orange: #CE9178;
10  --vscode-purple: #C586C0;
11  --vscode-cyan: #4EC9B0;
12 }
13
14 body {
15   background-color: var(--vscode-bg);
16   color: var(--vscode-fg);
17 }
18
19 a {
20   color: var(--vscode-blue);
21 }
22
23 a:hover {
24   color: var(--vscode-yellow);
25 }
26
27 button,
28 input[type="submit"] {
29   background-color: var(--vscode-grey);
30   border-color: var(--vscode-grey);
31   color: var(--vscode-fg);
32 }
```

```

33
34 button:hover ,
35 input [type="submit"]:hover {
36     background-color: var(--vscode-blue);
37     border-color: var(--vscode-blue);
38 }
39
40 code ,
41 pre {
42     background-color: var(--vscode-grey);
43     color: var(--vscode-fg);
44 }
45
46 code {
47     padding: 2px;
48 }
49
50 pre {
51     padding: 10px;
52 }
53
54 .highlight {
55     background-color: var(--vscode-yellow);
56     color: var(--vscode-bg);
57 }
58
59 .variable {
60     color: var(--vscode-purple);
61 }
62
63 .value {
64     color: var(--vscode-green);
65 }
66
67 .function {
68     color: var(--vscode-blue);
69 }
70
71 .other {
72     color: var(--vscode-orange);
73 }

```

Listing B.2: Tratamento de HTML

```

1 def initHTML():
2
3     return """
4 <!DOCTYPE html>

```

```

5 <html>
6   <head>
7     <meta charset="utf-8">
8     <title>Análise do ficheiro</title>
9   </head>
10  <style>
11    html {
12      background-color: #000000;
13    }
14    .error {
15      position: relative;
16      display: inline-block;
17      border-bottom: 1px dotted black;
18      color: red;
19    }
20    .code {
21      position: relative;
22      display: inline-block;
23    }
24    .error .errortext {
25      visibility: hidden;
26      width: 500px;
27      background-color: #555;
28      color: #fff;
29      text-align: center;
30      border-radius: 6px;
31      padding: 5px 0;
32      position: absolute;
33      z-index: 1;
34      bottom: 125%;
35      left: 50%;
36      margin-left: -100px;
37      opacity: 0;
38      transition: opacity 0.3s;
39    }
40    .error .errortext::after {
41      content: " ";
42      position: absolute;
43      top: 100%;
44      left: 20%;
45      margin-left: -5px;
46      border-width: 5px;
47      border-style: solid;
48      border-color: #555 transparent transparent transparent;
49    }
50    .error:hover .errortext {
51      visibility: visible;
52      opacity: 1;

```

```

53 }
54 pre {
55     background-color: #2D2D2D;
56     color: #F8F8F2;
57     padding: 20px;
58     font-family: "Consolas", "Courier New", monospace;
59     font-size: 14px;
60     line-height: 1.5;
61     overflow-x: auto;
62     border-radius: 5px;
63 }
64 .keyword {
65     color: #C586C0;
66     font-weight: bold;
67 }
68 .value {
69     color: #B5CEA8;
70 }
71 .function-name {
72     color: #DCDCAA;
73 }
74
75 .type {
76     color: #4EC9B0;
77     font-style: italic;
78 }
79 thead {
80     background-color: #2c313c;
81     color: #fff; /* para definir a cor do texto dentro do cabeçalho
82                 como branco */
83 }
84 .variable-name {
85     color: #9CDCFE;
86 }
87 .highlight {
88     background-color: #3B3B3B;
89     color: #D4D4D4;
90 }
91 .table {
92     font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
93     color: #abb2bf;
94     border-collapse: collapse;
95 }
96 .table th {
97     color: #dcdfe4;
98     text-align: left;
99     font-weight: 600;
100    border-bottom: 2px solid #3f4451;

```

```

100         padding: 12px 15px;
101     }
102     .table td {
103         border-bottom: 1px solid #2c313c;
104         padding: 10px 15px;
105     }
106     .table tr:last-child td {
107         border-bottom: none;
108     }
109     .table tr:nth-child(even) td {
110         background-color: #2c313c;
111     }
112     .table tr:hover td {
113         background-color: #3e4452;
114     }
115 </style>
116 <body>
117     <pre><code>"""
118
119 def endHTML():
120     return """
121         </code></pre>
122         </body>
123     </html>"""
124
125 def error(id, txt, errortxt):
126     return f"""<span id="{id}" class="error">{txt}<span class="errortext">{
127         errortxt}</span></span> """
128
129 def function_name(txt, indent):
130     return "<span class='function-name'>" + "\t"*indent + f"""{txt} </span
131         >"""
132
133 def variable_name(txt):
134     return "<span class='variable-name'>" + f"""{txt} </span>"""
135
136 def type(txt, indent):
137     return "<span class='type'>" + "\t"*indent + f"""{txt} </span>"""
138
139 def keyword(txt, indent):
140     return "<span class='keyword'>" + "\t"*indent + f"""{txt} </span>"""
141
142 def value(txt):
143     return "<span class='value'>" + f"""{txt} </span>"""
144
145 def highlight(txt):
146     return f"""<span class="highlight">{txt}</span>"""

```

```

146 def listToTxt(l : list):
147     txt = ""
148     if len(l) > 0:
149         return str(l[0])
150     for i in l[1:]:
151         txt += ", " + str(i)
152     return txt
153
154
155 def stats(instrucoes, vars, funcoes, nmerge):
156     not_repetidas = [funcoes[key]["vars"] for _, key in enumerate(funcoes)]
157     not_repetidas = list(filter(lambda x: "extra" not in x, not_repetidas))
158     n_vars = len(vars) + len(not_repetidas)
159
160     tipos = set()
161
162     for _, key1 in enumerate(funcoes):
163         for _, key2 in enumerate(funcoes[key1]["vars"]):
164             if funcoes[key1]["vars"][key2]["dec"] != 0:
165                 tipos.add(funcoes[key1]["vars"][key2]["tipo"])
166
167                 if "extra" in funcoes[key1]["vars"][key2] and funcoes[key1]["vars"]
168                     ["key2"]["dec"] == 1:
169                     n_vars -= 1
170
171     for _, key in enumerate(vars):
172         tipos.add(vars[key]["tipo"])
173
174     n_Func_varLocais = 0
175     for _, key1 in enumerate(funcoes):
176         for _, key2 in enumerate(funcoes[key1]["vars"]):
177             if "extra" not in funcoes[key1]["vars"][key2]:
178                 n_Func_varLocais += 1
179     n_tipos = len(tipos)
180
181     return f"""
182     <div style="display: flex; justify-content: center; padding: 0 30px;">
183         <div style="justify-content: center; padding: 0 30px;">
184             <table class="table">
185                 <thead>
186                     <tr>
187                         <th>Total</th>
188                         <th>Valor</th>
189                     </tr>
190                 </thead>
191                 <tbody>
192                     <tr>
193                         <td>Variáveis declaradas</td>

```

```

193         <td>{n_vars}</td>
194     </tr>
195     <tr>
196         <td>Tipos de dados</td>
197         <td>{n_tipos}</td>
198     </tr>
199     <tr>
200         <td>Funções</td>
201         <td>{len(funcoes)}</td>
202     </tr>
203     <tr>
204         <td>Funções com parâmetros</td>
205         <td>{len([key for _, key in enumerate(funcoes) if len
                (funcoes[key][”params”]) > 0])}</td>
206     </tr>
207     <tr>
208         <td>Funções com variáveis locais</td>
209         <td>{n_Func_varLocais}</td>
210     </tr>
211 </tbody>
212 </table>
213 </div>
214 <div style=”justify-content: center; padding: 0 30px;”>
215     <table class=”table”>
216         <thead>
217             <tr>
218                 <th>Total</th>
219                 <th>Valor</th>
220             </tr>
221         </thead>
222         <tbody>
223             <tr>
224                 <td>Instruções</td>
225                 <td>{instrucoes[”total”]}</td>
226             </tr>
227             <tr>
228                 <td>Atribuições</td>
229                 <td>{instrucoes[”atrib”]}</td>
230             </tr>
231             <tr>
232                 <td>Escrita</td>
233                 <td>{instrucoes[”escrita”]}</td>
234             </tr>
235             <tr>
236                 <td>Leitura</td>
237                 <td>{instrucoes[”leitura”]}</td>
238             </tr>
239             <tr>

```



```

240         <td>Seleção</td>
241         <td>{instrucoes ["selecao"]}</td>
242     </tr>
243     <tr>
244         <td>Repetição</td>
245         <td>{instrucoes ["repeticao"]}</td>
246     </tr>
247     <tr>
248         <td>Controlo aninhado</td>
249         <td>{instrucoes ["aninhado"]}</td>
250     </tr>
251     <tr>
252         <td>Substituição por um SE</td>
253         <td>{nmerge}</td>
254     </tr>
255 </tbody>
256 </table>
257 </div>
258 </div>
259 """

```

Listing B.3: Analisador

```

1 from lark import Discard
2 from lark import Lark,Token,Tree
3 from lark.visitors import Interpreter
4 import htmlOut
5
6 gramatica = open("data/gic.txt", "r").read()
7
8 class Gramatica(Interpreter):
9     vars = {} # corpo principal
10    #self.vars[id] = {
11    #    "tipo": None,
12    #    "val": val,
13    #    "dec": 0,
14    #    "init": 1
15    #    }
16
17    #self.funcoes[self.func_id]["vars"][id] = {
18    #    "tipo": None,
19    #    "val": val,
20    #    "dec": 0,
21    #    "init": 1
22    #    }
23    # self.funcoes[self.func_id] ={
24    #    "params" : {},
25    #    "ret" : val,

```

```

26 #         "vars" : {},
27 #     }
28
29 html = htmlOut.initHTML()
30 funcoes = {}
31 merge = 0
32 indent = 0
33 se_aninhado = 0
34 SE_simples = None
35 aninhado = 0
36 func_id = None
37 instrucoes = {
38     "total" : 0,
39     "atrib" : 0,
40     "escrita": 0,
41     "leitura": 0,
42     "selecao": 0,
43     "repeticao": 0,
44     "aninhado": 0,
45     "merge": 0,
46 }
47
48 def programa(self, args):
49     self.visit_children(args)
50     self.html += htmlOut.endHTML()
51     self.stats()
52     return {"vars": self.vars,
53           "funcs": self.funcoes,
54           "html": self.html,
55           "instrucoes": self.instrucoes,
56           }
57
58 def funcao(self, args):
59     id = str(args.children[0])
60     params = self.visit(args.children[1])
61     self.new_func(id, params)
62
63     paramsHTML = ""
64     for _, key in enumerate(params):
65         paramsHTML += htmlOut.variable_name(key) + ", "
66     paramsHTML = paramsHTML[:-2]
67     self.html += htmlOut.keyword(id, self.indent) + " { " + paramsHTML +
68         "}"
69     self.html += "<br>"
70     self.indent += 1
71
72     for child in args.children[1:-1]:
73         self.visit(child)

```

```

73
74     self.indent -= 1
75     self.html += htmlOut.keyword("RET", self.indent)
76     for _, html in self.visit(args.children[-1]):
77         self.html += html
78
79     self.html += "<br>"
80     self.func_id = None
81
82     def item(self, args):
83         self.visit_children(args)
84
85     def declaracao(self, args):
86         tipo = str(args.children[0])
87         id = str(args.children[1])
88         valor = None
89         dec = 1
90         init = 0
91         if len(args.children) > 2:
92             valor = self.visit(args.children[2])
93             init = 1
94
95         self.html += htmlOut.keyword(tipo, self.indent)
96
97         erro, html = self.checkID(id)
98
99         if erro == -1: #Não declarada
100             self.new_var(id, tipo, valor, dec, init, self.func_id)
101             self.html += htmlOut.variable_name(id)
102         else: #Variavel já declarada
103             self.html += html
104
105         if valor:
106             self.html += htmlOut.value(str(valor))
107
108         self.html += "<br>"
109
110
111     def instrucao(self, args):
112         self.instrucoes["total"] += 1
113         self.visit_children(args)
114
115     def atrib(self, args):
116         self.instrucoes["atrib"] += 1
117         self.html += htmlOut.keyword("ATRIB", self.indent)
118
119         id = self.visit(args.children[0])
120         main_id = id[0] if type(id) == list else id

```

```

121     ans = self.visit(args.children[1])
122     valor = "".join([str(x[0]) for x in ans])
123     valHTML = "".join([x[1] for x in ans])
124
125     err, errHTML = self.checkID(id)
126     if err == -1: #Não declarada
127         self.html += errHTML + ("".join(id[1:]) if type(id) == list else
128             "")
129     else:
130         if err == -2 or err == 2 or err == 0:
131             #Pode ou não ter sido inicializada (não importa). Pode ser
132             #variável local ou global
133             if self.func_id == None: # Variável global
134                 self.vars[main_id]["init"] += 1
135                 self.vars[main_id]["val"] = valor
136             else: # Variável local
137                 self.funcoes[self.func_id]["vars"][main_id]["init"] += 1
138                 self.funcoes[self.func_id]["vars"][main_id]["val"] =
139                     valor
140             self.html += htmlOut.variable_name(main_id + "".join(id[1:]))
141
142         else: # Variável de um parâmetro. Ver se esse parâmetro existe
143             if main_id in self.funcoes[self.func_id]["params"]:
144                 self.new_var(main_id, None, valor, 0, 1, self.func_id)
145                 self.funcoes[self.func_id]["vars"][main_id]["extra"] =
146                     True
147                 self.funcoes[self.func_id]["vars"][main_id]["init"] += 1
148                 self.funcoes[self.func_id]["vars"][main_id]["val"] =
149                     valor
150                 self.html += htmlOut.variable_name(main_id + "".join(id
151                     [1:]))
152             else:
153                 self.html += htmlOut.error("notDecl", main_id, "Variável
154                     nula")
155
156     self.html += valHTML
157     self.html += "<br>"
158
159 def leitura(self, args):
160     self.instrucoes["leitura"] += 1
161
162     self.html += htmlOut.keyword("LER", self.indent)
163
164     id = self.visit(args.children[0])
165     main_id = id[0] if type(id) == list else id
166     err, errHTML = self.checkID(id)

```

```

161
162 if err == -1: #Não declarada
163     self.html += errHTML + (".".join(id[1:])) if type(id) == list else
164         """)
165 else:
166     if err == -2 or err == 2 or err == 0:
167         #Pode ou não ter sido inicializada (não importa). Pode ser
168         #variável local ou global
169         if self.func_id == None: # Variável global
170             self.vars[main_id]["init"] += 1
171             self.vars[main_id]["val"] = "read_value"
172         else: # Variável local
173             self.funcoes[self.func_id]["vars"][main_id]["init"] += 1
174             self.funcoes[self.func_id]["vars"][main_id]["val"] = "
175                 read_value"
176         self.html += htmlOut.variable_name(main_id + ".".join(id[1:]))
177         )
178
179     else: # Variável de um parâmetro. Ver se esse parâmetro existe
180         if self.funcoes[self.func_id]["params"][main_id] != None:
181             self.funcoes[self.func_id]["params"][main_id]["init"] +=
182                 1
183             self.funcoes[self.func_id]["params"][main_id]["val"] = "
184                 read_value"
185             self.html += htmlOut.variable_name(main_id + ".".join(id
186                 [1:]))
187
188         else:
189             self.html += htmlOut.error("notDecl", main_id, "Variável
190                 nula")
191
192     self.html += "<br>"
193
194 def escrita(self, args):
195     self.instrucoes["escrita"] += 1
196     self.html += htmlOut.keyword("ESCREVER", self.indent)
197     val, valHTML = self.visit(args.children[0])[0]
198     if val == None:
199         self.html += valHTML
200     else :
201         self.html += htmlOut.variable_name(str(val))
202     self.html += "<br>"
203
204 def exp2(self, args):
205     if isinstance(args.children[0], Tree):
206         return [str(child) for child in args.children]
207     return str(args.children[0])

```

```

201
202 def exp(self, args):
203     return self.visit_children(args)
204
205 def abs(self, args):
206     if isinstance(args.children[0], Tree):
207         return self.visit(args.children[0])
208     else:
209         self.html += "!"
210         return self.visit(args.children[1])
211
212 def op(self, args):
213     return None, str(args.children[0]) + " "
214
215 # verifica ID quando faz parte de uma expressão
216 def checkID(self, child):
217     id = str(child)
218     if self.func_id != None:
219         if id in self.funcoes[self.func_id]["vars"]:
220             if self.funcoes[self.func_id]["vars"][id]["init"] == 0:
221                 return -2, htmlOut.error("notInit", id, "Variável já
222                     declarada mas não inicializada")
223             else:
224                 return 2, htmlOut.error("Init", id, "Variável já
225                     declarada e inicializada")
226         elif id in self.funcoes[self.func_id]["params"]:
227             return 1, htmlOut.error("Decl", id, "Variável já declarada
228                 mas não inicializada")
229         else:
230             return -1, htmlOut.error("notDecl", id, "Variável não
231                 declarada nem inicializada")
232     if id in self.vars:
233         if self.vars[id]["init"] == 0:
234             return -2, htmlOut.error("notInit", id, "Variável já
235                 declarada mas não inicializada")
236         else:
237             return 0, htmlOut.error("Init", id, "Variável já declarada e
238                 inicializada")
239     return -1, htmlOut.error("notDecl", id, "Variável não declarada nem
240         inicializada")
241
242
243
244
245
246 def valor(self, args):
247     fst_children = args.children[0]
248     typedata = fst_children.data if isinstance(fst_children, Tree) else
249         fst_children.type
250     if typedata == "ID":
251         err, errHTML = self.checkID(fst_children)

```

```

241         if err < 0:
242             return None, errHTML
243         else:
244             id = str(fst_children)
245             return id, htmlOut.variable_name(id)
246
247     elif typedata == "ELEM":
248         exp2 = self.visit(fst_children)
249         return str(fst_children) + exp2, htmlOut.keyword(str(fst_children
250             )) + htmlOut.variable_name(exp2)
251
252     elif typedata == "funcao_chamada":
253         funcHTML = self.visit(fst_children)
254         return [-6, funcHTML]
255
256     elif typedata == "val":
257         value = self.visit(fst_children)
258         return [value, htmlOut.value(str(value))]
259
260     else:
261         id = self.visit(fst_children)
262         cID = self.checkID(id)
263         if cID[0] < 0:
264             return None, cID[1] + (self.visit(args.children[1:]) if len(
265                 args.children) > 1 else "")
266         else:
267             return id, htmlOut.variable_name(id) #+ self.visit(args.
268                 children[1:]) TODO
269
270 def params(self, args):
271     params = {}
272     for child in args.children:
273         params[str(child)] = self.vars[str(child)] if str(child) in self.
274             vars else {}
275     return params
276
277 def selecaoSE(self, args):
278     self.html += htmlOut.keyword("SE", self.indent)
279
280     expressaoHTML = ""
281     exp = self.visit(args.children[0])
282     for _, html in exp:
283         expressaoHTML += html
284
285     self.html += expressaoHTML + "\n"
286
287     self.indent += 1

```

```

285
286     for child in args.children[1:]:
287         self.visit(child)
288
289     self.indent -= 1
290
291     if self.SE_simples == None:
292         # Último SE
293         self.SE_simples = True
294     elif len(args.children) == 2 and self.SE_simples == True:
295         # Restante dos SEs. Só podem ter um filho que é um SE
296         self.merge += 1
297         self.SE_simples = True
298     elif len(args.children) > 2 and self.SE_simples == True:
299         # SE com mais de um filho, já não pode ser aninhado
300         self.SE_simples = False
301
302
303     self.html += htmlOut.keyword("ES", self.indent)
304     self.html += "<br>"
305
306
307 def selecaoCASO(self, args):
308     # fora de funcoes
309     id = self.visit(args.children[0])
310
311     self.html += htmlOut.keyword("CASO", self.indent)
312     err, errHTML = self.checkID(id)
313     if err < 0:
314         self.html += errHTML
315     else:
316         self.html += htmlOut.variable_name(id)
317
318     self.html += "<br>"
319     self.indent += 1
320
321     exps = iter(args.children[1:])
322     for child in exps:
323         # val
324         self.html += "\t"*self.indent + str(self.visit(child)) + " :\n"
325         # item
326         self.indent += 1
327         self.visit(next(exps))
328         self.indent -= 1
329
330     self.indent -= 1
331
332     self.html += htmlOut.keyword("OSAC", self.indent)

```



```

333     self.html += "<br>"
334
335
336 def selecao(self, args):
337     if self.aninhado > 0:
338         self.instrucoes["aninhado"] += 1
339     self.aninhado += 1
340     self.se_aninhado += 1
341     self.instrucoes["selecao"] += 1
342     if args.children[0].data == "exp":
343         self.selecaoSE(args)
344     else:
345         self.selecaoCASO(args)
346
347     self.aninhado -= 1
348     self.se_aninhado -= 1
349     if self.se_aninhado == 0:
350         self.SE_simples = None
351
352 def repeticao(self, args):
353     if self.aninhado > 0:
354         self.instrucoes["aninhado"] += 1
355     self.aninhado += 1
356
357     if isinstance(args.children[0], Tree):
358         if args.children[0].data == "exp":
359             self.html += htmlOut.keyword("ENQ", self.indent)
360             a = self.visit(args.children[0])
361             for _, html in a:
362                 self.html += html
363
364             self.html += htmlOut.keyword("FAZER", self.indent)
365             self.html += "{"
366             self.html += "<br>"
367             self.indent += 1
368
369             for child in args.children[1:]:
370                 self.visit(child)
371
372             self.indent -= 1
373             self.html += "}"
374             self.html += "<br>"
375         elif args.children[0].data == "item":
376             self.html += htmlOut.keyword("REPETIR", self.indent)
377             self.indent += 1
378             self.html += "<br>"
379             for child in args.children[:-1]:
380                 self.visit(child)

```

```

381
382     self.indent -= 1
383     self.html += htmlOut.keyword("ATE", self.indent)
384     a = self.visit(args.children[-1])
385     for _, html in a:
386         self.html += html
387     self.html += "<br>"
388 else:
389     self.html += htmlOut.keyword("PARA", self.indent)
390     self.html += htmlOut.value(str(args.children[0]))
391     self.html += htmlOut.keyword("FAZER", self.indent)
392     self.html += "<br>"
393     self.html += "{"
394     self.html += "<br>"
395     self.indent += 1
396
397     for child in args.children[1:]:
398         self.visit(child)
399
400     self.indent -= 1
401     self.html += "}"
402     self.html += "<br>"
403
404     self.instrucoes["repeticao"] += 1
405     self.aninhado -= 1
406
407 def funcao_chamada(self, args):
408     func_id = str(args.children[0])
409     func_html = ""
410     if func_id in self.funcoes:
411         func_html += htmlOut.keyword(str(func_id), self.indent)
412     else:
413         func_html += htmlOut.error(func_id, "Função não declarada")
414
415     params = ""
416
417     for param in self.visit(args.children[1]):
418         err, errHTML = self.checkID(param)
419         if err < 0:
420             params += errHTML
421         else:
422             params += htmlOut.variable_name(str(param))
423         params += ", "
424
425     return func_html + " { " + params + " } "
426
427
428 ##### TIPOS DE DADOS #####

```

```

429
430 def val(self, args):
431     args = args.children[0]
432     val = None
433     # conjunto (lista | tuplo)
434     if isinstance(args, Tree):
435         val = self.visit(args)
436     else:
437         if args.type == "NUM":
438             val = int(args)
439
440         elif args.type == "BOOL":
441             if (args == "True"):
442                 val = True
443             else:
444                 val = False
445
446         elif args.type == "STRING":
447             val = str(args)
448
449     return val
450
451 def conjunto(self, args):
452     return self.visit(args.children[0])
453
454 def tuplo(self, args):
455     val = ()
456     for child in args.children:
457         val += (self.visit(child),)
458
459     return val
460
461 def lista(self, args):
462     val = []
463     for child in args.children:
464         val.append(self.visit(child))
465     return val
466
467 def args(self, args):
468     params = []
469     for child in args.children:
470         params.append(str(child))
471     return params
472
473 def new_func(self, id, params):
474     self.func_id = id
475     self.funcoes[self.func_id] = {
476         "params" : params,

```

```

477         "vars" : {},
478     }
479
480     def new_var(self, id, tipo, valor, dec, init, func_id):
481         if func_id != None: #Variável dentro de função
482             self.funcoes[func_id]["vars"][id] = {
483                 "tipo" : tipo,
484                 "valor" : valor,
485                 "dec" : dec,
486                 "init" : init,
487             }
488         else: #Variável fora de função
489             self.vars[id] = {
490                 "tipo" : tipo,
491                 "valor" : valor,
492                 "dec" : dec,
493                 "init" : init,
494             }
495
496     def stats(self):
497         self.html += htmlOut.stats(self.instrucoes, self.vars, self.funcoes,
498                                     self.merge)
499
500
501 p = Lark(gramatica, start="programa")
502
503 tree = p.parse(open("data/testes.txt", "r").read())
504
505 data = Gramatica().visit(tree) # chamar o transformer para obter
506
507 with open("out/output.html", "w") as f:
508     f.write(data["html"])
509     f.close()

```
