UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO

Construção de Compiladores

Documentação Interna - T3

Áquila Oliveira, 759313

Carlos Eduardo Fontaneli , 769949

Ingrid Lira dos Santos, 790888

São Carlos - SP

11 de julho de 2023

1 Documentação Interna

1.1 Documentação interna da main.py

```python
# Importacao de bibliotecas
import sys
from antlr4 import *
from LALexer import LALexer
from LAParser import LAParser
from LASemantico import LASemantico
from Analisador import Analisador

sys.setrecursionlimit(10000)

# Leitura dos nomes e abertura dos arquivos
input_file_name = sys.argv[1]
output_file_name = sys.argv[2]
input_stream = FileStream(input_file_name, encoding="utf-8")
output = open(output_file_name, "w")

# Criando o analisador lexico
lexer = LALexer(input_stream)

# Criando o fluxo de tokens
stream = CommonTokenStream(lexer)

# Criando o analisador sintatico
parser = LAParser(stream)

arvore = parser.programa()
listener = LASemantico()
# parser.addParseListener(listener)

# parser.programa()
listener.visitPrograma(arvore)

for error in Analisador.erros:
    output.write(error + "\n")
    print(error)
output.write("Fim da compilacao" + "\n")
```

```
output.close()
```

1.2 Documentação interna do Analisador.py

```
from LAParser import LAParser
from Estruturas import Escopo, TabelaDeSimbolos


class Analisador:
    erros = []

    def adicionar_erro_semantico(token, mensagem):
        Analisador.erros.append(f"Linha {token.line}: {mensagem}")

    def verificar_tipo_parcela_unario(escopos, contexto):
        if contexto.NUM_INT() is not None:
            return TabelaDeSimbolos.TipoLA.INTEIRO
        if contexto.NUM_REAL() is not None:
            return TabelaDeSimbolos.TipoLA.REAL
        if contexto.identificador() is not None:
            return Analisador.verificar_tipo_identificador(
                escopos, contexto.identificador()
            )
        if contexto.IDENT() is not None:
            resultado = Analisador.verificar_tipo_nome_var(
                escopos, contexto.IDENT().getText()
            )
            for expressaoContext in contexto.expressao():
                auxiliar = Analisador.verificar_tipo_expressao(
                    escopos, expressaoContext
                )
                if resultado is None:
                    resultado = auxiliar
                elif (
                    resultado != auxiliar
                    and auxiliar != TabelaDeSimbolos.TipoLA.INVALIDO
                ):
                    resultado = TabelaDeSimbolos.TipoLA.INVALIDO
            return resultado
        else:
```

```
            resultado = None
            for expressaoContext in contexto.expressao():
                auxiliar = Analisador.verificar_tipo_expressao(
                    escopos, expressaoContext
                )
                if resultado is None:
                    resultado = auxiliar
                elif (
                    resultado != auxiliar
                    and auxiliar != TabelaDeSimbolos.TipoLA.INVALIDO
                ):
                    resultado = TabelaDeSimbolos.TipoLA.INVALIDO
            return resultado

    def verificar_tipo_termo_logico(escopos, contexto):
        resultado = None
        for token in contexto.fator_logico():
            auxiliar = Analisador.verificar_tipo_fator_logico(escopos,
    token)
            if resultado is None:
                resultado = auxiliar
            elif resultado != auxiliar and auxiliar != TabelaDeSimbolos.
    TipoLA.INVALIDO:
                resultado = TabelaDeSimbolos.TipoLA.INVALIDO
        return resultado

    def verificar_tipo_fator_logico(escopos, contexto):
        return Analisador.verificar_tipo_parcela_logica(
            escopos, contexto.parcela_logica()
        )

    def verificar_tipo_parcela_logica(escopos, contexto):
        if contexto.exp_relacional() is not None:
            resultado = Analisador.verificar_tipo_exp_relacional(
                escopos, contexto.exp_relacional()
            )
        else:
            resultado = TabelaDeSimbolos.TipoLA.LOGICO
        return resultado
```

```
74
75      def verificar_tipo_exp_relacional(escopos, contexto):
76          resultado = None
77          if contexto.op_relacional() is not None:
78              for token in contexto.exp_aritmetica():
79                  auxiliar = Analisador.verificar_tipo_exp_aritmetica(
    escopos, token)
80                  auxiliar_numeric0 = (
81                      auxiliar == TabelaDeSimbolos.TipoLA.INTEIRO
82                      or auxiliar == TabelaDeSimbolos.TipoLA.REAL
83                  )
84                  resultado_numerico = (
85                      resultado == TabelaDeSimbolos.TipoLA.INTEIRO
86                      or resultado == TabelaDeSimbolos.TipoLA.REAL
87                  )
88                  if resultado is None:
89                      resultado = auxiliar
90                  elif (
91                      not (auxiliar_numeric0 and resultado_numerico)
92                      and auxiliar != resultado
93                  ):
94                      resultado = TabelaDeSimbolos.TipoLA.INVALIDO
95              if resultado != TabelaDeSimbolos.TipoLA.INVALIDO:
96                  resultado = TabelaDeSimbolos.TipoLA.LOGICO
97          else:
98              resultado = Analisador.verificar_tipo_exp_aritmetica(
99                  escopos, contexto.exp_aritmetica(0)
100             )
101         return resultado
102
103     def verificar_tipo_identificador(escopos, contexto):
104         nome_var = ""
105         resultado = TabelaDeSimbolos.TipoLA.INVALIDO
106         for i in range(len(contexto.IDENT())):
107             nome_var += contexto.IDENT(i).getText()
108             if i != len(contexto.IDENT()) - 1:
109                 nome_var += "."
110         for tabela in escopos.obter_pilha():
111             if tabela.contem(nome_var):
```

```
112                 resultado = Analisador.verificar_tipo_nome_var(escopos,
    nome_var)
113         return resultado
114 
115     def verificar_tipo_termo(escopos, contexto):
116         resultado = None
117         for token in contexto.fator():
118             auxiliar = Analisador.verificar_tipo_fator(escopos, token)
119             auxiliar_numeric0 = (
120                 auxiliar == TabelaDeSimbolos.TipoLA.INTEIRO
121                 or auxiliar == TabelaDeSimbolos.TipoLA.REAL
122             )
123             resultado_numerico = (
124                 resultado == TabelaDeSimbolos.TipoLA.INTEIRO
125                 or resultado == TabelaDeSimbolos.TipoLA.REAL
126             )
127             if resultado is None:
128                 resultado = auxiliar
129             elif (
130                 not (auxiliar_numeric0 and resultado_numerico) and
    auxiliar != resultado
131             ):
132                 resultado = TabelaDeSimbolos.TipoLA.INVALIDO
133         return resultado
134 
135     def verificar_tipo_fator(escopos, contexto):
136         resultado = None
137         for token in contexto.parcela():
138             auxiliar = Analisador.verificar_tipo_parcela(escopos, token)
139             if resultado is None:
140                 resultado = auxiliar
141             elif resultado != auxiliar and auxiliar != TabelaDeSimbolos.
    TipoLA.INVALIDO:
142                 resultado = TabelaDeSimbolos.TipoLA.INVALIDO
143         return resultado
144 
145     def verificar_tipo_parcela(escopos, contexto):
146         resultado = TabelaDeSimbolos.TipoLA.INVALIDO
147         if contexto.parcela_nao_unario() is not None:
```

```
148             resultado = Analisador.verificar_tipo_parcela_nao_unario(
149                 escopos, contexto.parcela_nao_unario()
150             )
151         else:
152             resultado = Analisador.verificar_tipo_parcela_unario(
153                 escopos, contexto.parcela_unario()
154             )
155         return resultado
156 
157     def verificar_tipo_parcela_nao_unario(escopos, contexto):
158         if contexto.identificador() is not None:
159             return Analisador.verificar_tipo_identificador(
160                 escopos, contexto.identificador()
161             )
162         return TabelaDeSimbolos.TipoLA.LITERAL
163 
164     def verificar_tipo_nome_var(escopos, nome_var):
165         tipo = None
166         for tabela in escopos.obter_pilha():
167             tipo = tabela.verificar(nome_var)
168         return tipo
169 
170     def verificar_tipo_expressao(escopos, contexto):
171         resultado = None
172         for token in contexto.termo_logico():
173             auxiliar = Analisador.verificar_tipo_termo_logico(escopos,
    token)
174             if resultado is None:
175                 resultado = auxiliar
176             elif resultado != auxiliar and auxiliar != TabelaDeSimbolos.
    TipoLA.INVALIDO:
177                 resultado = TabelaDeSimbolos.TipoLA.INVALIDO
178         return resultado
179 
180     def verificar_tipo_exp_aritmetica(escopos, contexto):
181         resultado = None
182         for token in contexto.termo():
183             auxiliar = Analisador.verificar_tipo_termo(escopos, token)
184             if resultado is None:
```

```
185                 resultado = auxiliar
186             elif resultado != auxiliar and auxiliar != TabelaDeSimbolos.
    TipoLA.INVALIDO:
187                 resultado = TabelaDeSimbolos.TipoLA.INVALIDO
188         return resultado
```

1.3 Documentação interna do LASemantico.py

```
1  from LAParser import LAParser
2  from LAVisitor import LAVisitor
3  from Estruturas import TabelaDeSimbolos, Escopo
4  from Analisador import Analisador
5
6
7  class LASemantico(LAVisitor):
8      escopos = Escopo()
9
10     def visitPrograma(self, contexto):
11         return super().visitPrograma(contexto)
12
13     def visitCmdAtribuicao(self, contexto):
14         tipoExpressao = Analisador.verificar_tipo_expressao(
15             LASemantico.escopos, contexto.expressao()
16         )
17         error = False
18         nome_var = contexto.identificador().getText()
19         if tipoExpressao != TabelaDeSimbolos.TipoLA.INVALIDO:
20             for escopo in LASemantico.escopos.obter_pilha():
21                 if escopo.contem(nome_var):
22                     tipoVar = Analisador.verificar_tipo_nome_var(
23                         LASemantico.escopos, nome_var=nome_var
24                     )
25                     varNumeric = (
26                         tipoVar == TabelaDeSimbolos.TipoLA.INTEIRO
27                         or tipoVar == TabelaDeSimbolos.TipoLA.REAL
28                     )
29                     expNumeric = (
30                         tipoExpressao == TabelaDeSimbolos.TipoLA.INTEIRO
31                         or tipoExpressao == TabelaDeSimbolos.TipoLA.REAL
32                     )
```

```
33                  if (
34                      not (varNumeric and expNumeric)
35                      and tipoVar != tipoExpressao
36                      and tipoExpressao != TabelaDeSimbolos.TipoLA.
    INVALIDO
37                  ):
38                      error = True
39          else:
40              error = True
41
42          if error:
43              Analisador.adicionar_erro_semantico(
44                  contexto.identificador().start,
45                  f"atribuicao nao compativel para {nome_var}",
46              )
47
48          return super().visitCmdAtribuicao(contexto)
49
50      def visitIdentificador(self, contexto):
51          for escopo in LASemantico.escopos.obter_pilha():
52              if not escopo.contem(contexto.IDENT(0).getText()):
53                  Analisador.adicionar_erro_semantico(
54                      contexto.start,
55                      f"identificador {contexto.IDENT(0).getText()} nao
    declarado",
56                  )
57                  break
58          return super().visitIdentificador(contexto)
59
60      def visitDeclaracao_tipo(self, contexto):
61          escopo_atual = LASemantico.escopos.obter_escopo_atual()
62
63          if escopo_atual.contem(contexto.IDENT().getText()):
64              Analisador.adicionar_erro_semantico(
65                  contexto.start,
66                  f"tipo {contexto.IDENT().getText()} ja declarado duas
    vezes no mesmo escopo",
67              )
68          else:
```

```
69              escopo_atual.adicionar(
70                  contexto.IDENT().getText(), TabelaDeSimbolos.TipoLA.TIPO
71              )
72
73          return super().visitDeclaracao_tipo(contexto)
74
75      def visitDeclaracao_variavel(self, contexto):
76          escopo_atual = LASemantico.escopos.obter_escopo_atual()
77
78          for identificador in contexto.variavel().identificador():
79              if escopo_atual.contem(identificador.getText()):
80                  Analisador.adicionar_erro_semantico(
81                      identificador.start,
82                      f"identificador {identificador.getText()} ja
    declarado anteriormente",
83                  )
84              else:
85                  tipo = TabelaDeSimbolos.TipoLA.INTEIRO
86                  if contexto.variavel().tipo().getText() == "literal":
87                      tipo = TabelaDeSimbolos.TipoLA.LITERAL
88                  elif contexto.variavel().tipo().getText() == "real":
89                      tipo = TabelaDeSimbolos.TipoLA.REAL
90                  elif contexto.variavel().tipo().getText() == "logico":
91                      tipo = TabelaDeSimbolos.TipoLA.LOGICO
92
93                  escopo_atual.adicionar(identificador.getText(), tipo)
94
95          return super().visitDeclaracao_variavel(contexto)
96
97      def visitDeclaracao_global(self, contexto):
98          escopo_atual = LASemantico.escopos.obter_escopo_atual()
99
100         if escopo_atual.contem(contexto.IDENT().getText()):
101             Analisador.adicionar_erro_semantico(
102                 contexto.start,
103                 f"{contexto.IDENT().getText()} ja declarado
    anteriormente",
104             )
105         else:
```

```
106             escopo_atual.adicionar(
107                 contexto.IDENT().getText(), TabelaDeSimbolos.TipoLA.TIPO
108             )
109 
110         return super().visitDeclaracao_global(contexto)
111 
112     def visitTipo_basico_ident(self, contexto):
113         if contexto.IDENT() is not None:
114             for escopo in LASemantico.escopos.obter_pilha():
115                 if not escopo.contem(contexto.IDENT().getText()):
116                     Analisador.adicionar_erro_semantico(
117                         contexto.start,
118                         f"tipo {contexto.IDENT().getText()} nao
    declarado",
119                     )
120                     break
121 
122         return super().visitTipo_basico_ident(contexto)
123 
124     def visitDeclaracao_constante(self, contexto: LAParser.
    Declaracao_constanteContext):
125         escopo_atual = LASemantico.escopos.obter_escopo_atual()
126 
127         if escopo_atual.contem(contexto.IDENT().getText()):
128             Analisador.adicionar_erro_semantico(
129                 contexto.start,
130                 f"constante {contexto.IDENT().getText()} ja declarada
    anteriormente",
131             )
132         else:
133             tipo = TabelaDeSimbolos.TipoLA.INTEIRO
134             if contexto.tipo_basico().getText() == "literal":
135                 tipo = TabelaDeSimbolos.TipoLA.LITERAL
136             elif contexto.tipo_basico().getText() == "real":
137                 tipo = TabelaDeSimbolos.TipoLA.REAL
138             elif contexto.tipo_basico().getText() == "logico":
139                 tipo = TabelaDeSimbolos.TipoLA.LOGICO
140 
141             escopo_atual.adicionar(contexto.IDENT().getText(), tipo)
```

```
142
143         return super().visitDeclaracao_constante(contexto)
```

1.4 Documentação interna do Estruturas.py

```
1  from enum import Enum
2
3  class TabelaDeSimbolos:
4      class TipoLA(Enum):
5          INTEIRO = 1
6          REAL = 2
7          LITERAL = 3
8          LOGICO = 4
9          INVALIDO = 5
10         TIPO = 6
11         IDENT = 7
12
13     class EntradaTabelaDeSimbolos:
14         def __init__(self, nome, tipo):
15             self.nome = nome
16             self.tipo = tipo
17
18     def __init__(self):
19         self.tabelaDeSimbolos = {}
20
21     def adicionar(self, nome, tipo):
22         etds = TabelaDeSimbolos.EntradaTabelaDeSimbolos(nome, tipo)
23         self.tabelaDeSimbolos[nome] = etds
24
25     def contem(self, nome):
26         return nome in self.tabelaDeSimbolos
27
28     def verificar(self, nome):
29         return self.tabelaDeSimbolos.get(nome).tipo
30
31
32 class Escopo:
33     def __init__(self):
34         self.pilhaDeTabelas = []
35         self.criar_novo_escopo()
```

```
36
37      def criar_novo_escopo(self):
38          self.pilhaDeTabelas.append(TabelaDeSimbolos())
39
40      def obter_escopo_atual(self):
41          return self.pilhaDeTabelas[-1]
42
43      def obter_pilha(self):
44          return self.pilhaDeTabelas
45
46      def abandonar_escopo(self):
47          self.pilhaDeTabelas.pop()
```