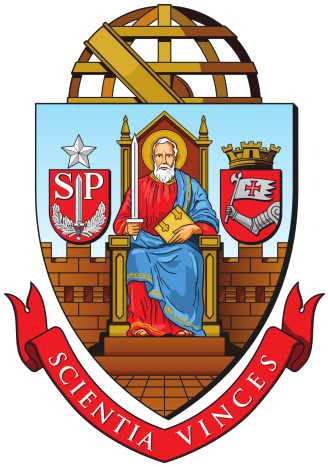
Escola de Engenharia de São Carlos
Instituto de Ciências Matemáticas e de Computação



SCC0605 - Teoria da Computação e Compiladores

1º Trabalho: Analisador Léxico

Bruner Eduardo Augusto - nºUSP: 9435846
Carlos Santos Junior - nºUSP: 9435102
Clayton Miccas Junior - nºUSP: 9292441

Docente: Dr. Thiago A. S. Pardo

23 de abril de 2020

Conteúdo

Lista de Figuras

Lista de Códigos

"O insucesso é apenas uma oportunidade para recomeçar de novo com mais inteligência."
(Henry Ford)

1 Decisões de Projeto

Com base na gramática da linguagem P–, disponível no Repositório do Tidia, enriquecida com o comando "for" (como discutido em aula), desenvolver o analisador léxico para esta linguagem.

```
<programa> ::= program ident ; <corpo> .
<corpo> ::= <dc> begin <comandos> end
<dc> ::= <dc_c> <dc_v> <dc_p>
<dc_c> ::= const ident = <numero> ; <dc_c> | λ
<dc_v> ::= var <variaveis> : <tipo_var> ; <dc_v> | λ
<tipo_var> ::= real | integer
<variaveis> ::= ident <mais_var>
<mais_var> ::= , <variaveis> | λ
<dc_p> ::= procedure ident <parametros> ; <corpo_p> <dc_p> | λ
<parametros> ::= ( <lista_par> ) | λ
<lista_par> ::= <variaveis> : <tipo_var> <mais_par>
<mais_par> ::= ; <lista_par> | λ
<corpo_p> ::= <dc_loc> begin <comandos> end ;
<dc_loc> ::= <dc_v>
<lista_arg> ::= ( <argumentos> ) | λ
<argumentos> ::= ident <mais_ident>
<mais_ident> ::= ; <argumentos> | λ
<pfalsa> ::= else <cmd> | λ
<comandos> ::= <cmd> ; <comandos> | λ
<cmd> ::= read ( <variaveis> ) |
          write ( <variaveis> ) |
          while ( <condicao> ) do <cmd> |
          for ident := <expressao> to <expressao> <cmd> |
          if <condicao> then <cmd> <pfalsa> |
          ident := <expressao> |
          ident <lista_arg> |
          begin <comandos> end
<condicao> ::= <expressao> <relacao> <expressao>
<relacao> ::= = | <> | >= | <= | > | <
<expressao> ::= <termo> <outros_termos>
<op_un> ::= + | - | λ
<outros_termos> ::= <op_ad> <termo> <outros_termos> | λ
<op_ad> ::= + | -
<termo> ::= <op_un> <fator> <mais_fatores>
<mais_fatores> ::= <op_mul> <fator> <mais_fatores> | λ
<op_mul> ::= * | /
<fator> ::= ident | <numero> | ( <expressao> )
<numero> ::= numero_int | numero_real
```

Código 1: Linguagem P– Enriquecida com o Comando For[23]

Conforme o código acima apresentado, elaborou-se o projeto de um analisador léxico na linguagem python, o qual está em anexo a este relatório. No decorrer do projeto decidimos estabelecer algumas diretrizes para fazer a tradução dos autômatos para a linguagem de programação python. Os autômatos definidos foram os de identificadores, números, parênteses, chaves (comentários), pontuações e operadores. Os dois últimos se encontram unidos em uma função em python devido a suas ocorrências estarem intimamente ligadas aos demais autômatos. Os demais se tornaram funções específicas, as quais estão facilmente encontradas no código "LexicalAnalyzer".

2 Autômatos Projetados para a Linguagem P–

Para a elaboração dos autômatos a seguir, o grupo utilizou o software JFLAP 7.1-software foi desenvolvido por Susan H. Rodger atual professora da *Duke University*[1]-apresentado em aula. O software possui funcionalidades de desenvolvimento de autômatos finitos, máquina de Mealy, Moore, Turing, etc. No desenvolvimento deste projeto utilizamos máquina de Moore nos autômatos, dado que esta retorna sua saída levando em consideração apenas o estado atual. O grupo julgou esta como a melhor levando em consideração os requisitos do analisador léxico projetado e para futuros incrementos. A seguir especificamos os autômatos desenvolvidos.

2.1 Autômato Identificador

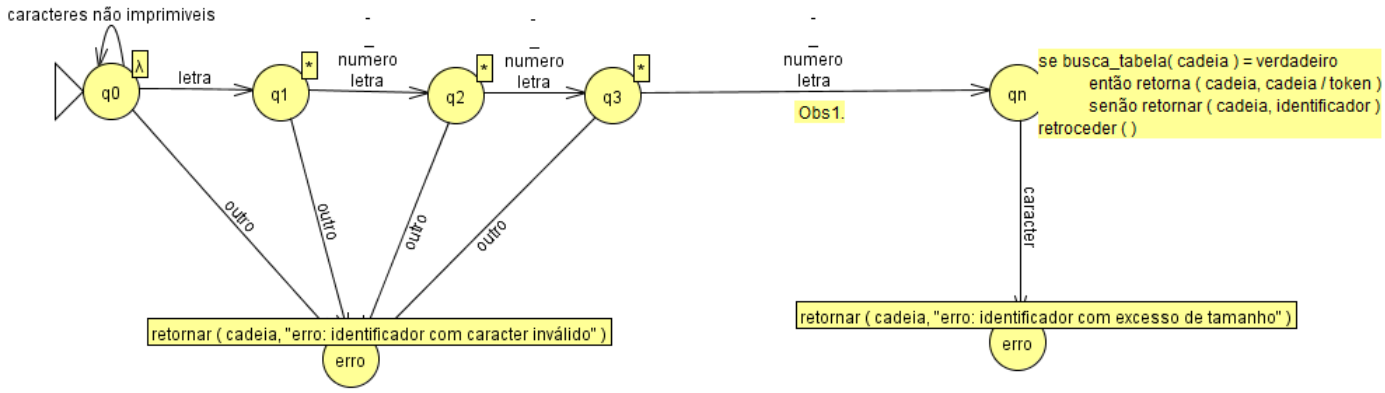


Figura 1: Autômato Identificador

O autômato identificador foi desenvolvido para validar um identificador, sendo que este deve necessariamente iniciar com uma letra, sendo esta seguida por outras letras, números, _ e -. No estado $q0$ temos que o autômato não muda de estado enquanto o mesmo recebe caracteres não imprimíveis, (caracteres de tabulação, espaços em branco, nova linha, etc...). Ainda no mesmo estado, o automato pode receber uma letra e prosseguir para $q1$, ou ir para o estado de erro no qual o mesmo retorna uma mensagem com o erro de entrada. Se a fita possuir mais caracteres, o autômato percorre os estados $q2$ $q3$ ... qn, sempre verificando a validez do carácter.

O autômato foi elaborado com a perspectiva de que entre os estados $q3$ e qn existem estados semelhantes a $q3$. Esta alternativa foi tomada como uma ilustração de recursão, essa aplicação se dá devido a necessidade de contabilizar o tamanho do identificador, sendo que este tamanho é finito e caso seja ultrapassado, há a transição para um estado de erro. Em nosso Analisador léxico utilizamos o tamanho de 32 caracteres, está definição fica a cargo da linguagem.

Por fim foram feitas as seguintes observações: primeira observação(Obs1), indica a alternativa à recursão, onde um identificador pode ter o tamanho de um único carácter, estagio $q1$, até no máximo n, estágio qn.

A saída dos estados $q1$, $q2$, ..., qn retornam tanto com aquela cadeia de caracteres e um token, caso a mesma estiver presente em uma tabela de palavras reservadas, quanto

retornam a cadeia e um identificador, caso contrário. A saída dos estados $q1$, $q2$, ..., $qn-1$ foram descritas como * para uma melhor ilustração.

2.2 Autômato Número

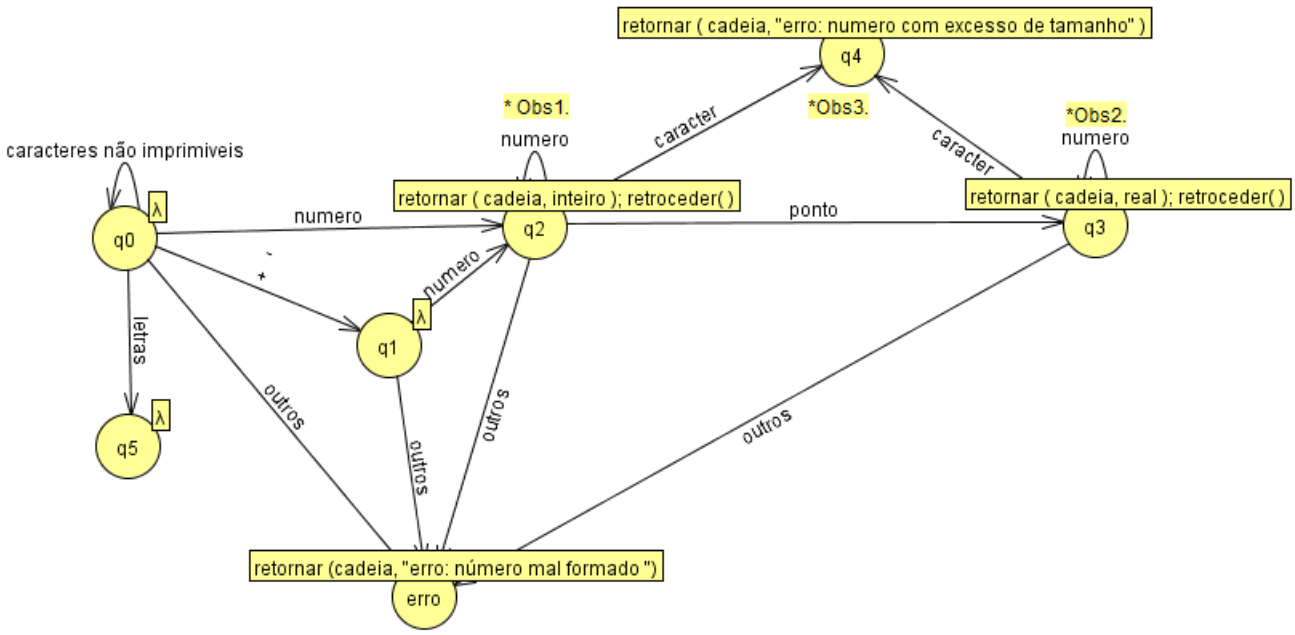


Figura 2: Autômato Números

O autômato números descreve a aceitação de uma entrada como número, e separa esta como número inteiro ou número real. O autômato aceita como entrada inicialmente um digito , + ou -, letras, outros e caracteres não imprimíveis, semelhante ao autômato identificador. Caso receba uma letra, a mesma não pertence a este autômato e vai para um estado vazio, se receber outros dígitos (caracteres especiais) vai para um estado de erro e por fim se receber um dos sinais, de + ou -, ou um numero, o autômato prossegue para o estado $q1$ e $q2$ respectivamente.

Caso o estado seja $q1$ pode-se receber um número e seguir ao estado $q2$,ou receber outros caracteres e relatar um erro que é um número mal formado. Quando estiver em $q2$ seguido de dígitos a partir dessa entrada há a recursão no estado $q2$, a observação 1 (Obs1) é devido ao tamanho máximo que um número pode possuir, semelhante ao explicado na descrição do autômato identificador, e caso ultrapasse este máximo o programa retorna o erro do estado $q4$. Se encerrado no estado $q2$ o autômato retorna que este número é um inteiro, se inserido um carácter do tipo "outros"este é identificado como um número mal formado.

Respeitando este limite de tamanho apresentado, pode-se adicionar um ponto (.) e ir para o estado $q3$ seguido de novos dígitos no mesmo estado. A observação dois (Obs2), retrata novamente o limite de tamanho do número, sendo que se encerrado neste estado o número é classificado como um real, caso haja caractere inválido a cadeia é retornada como um número mal formado. A (Obs3) trata do tamanho do número, como estipulado anteriormente o tamanho máximo de 32 caracteres caso o número possua "n" dígitos no

estado $q2$, este só poderá ter $31 - n$ dígitos no estado $q3$ (31 e não 32 devido ao uso de um destes para o "ponto").

2.3 Autômato Parenteses

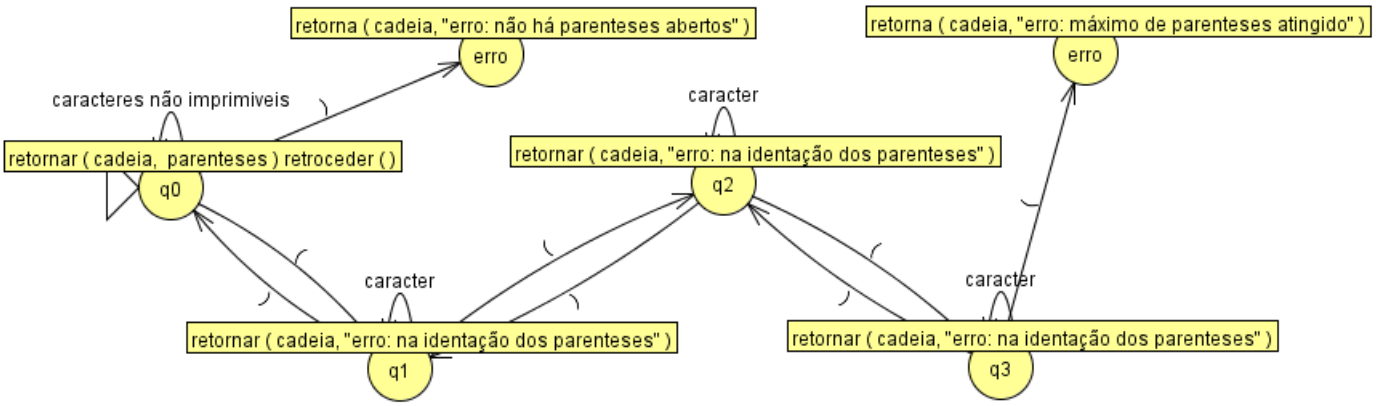


Figura 3: Autômato Parênteses

O autômato parenteses verifica a abertura e fechamento de até 3 parenteses, é fácil a verificação que este pode ser implementado recursivamente para o desenvolvimento de "n" parenteses, mas por ora utilizamos no máximo 3. Este autômato permanece no estado $q0$ enquanto não recebe a abertura de um parenteses, caso haja a leitura de uma letra ou um número, neste estado, os mesmo caracteres podem ser verificado nos autômatos apresentados anteriormente. Quando há a abertura de um parenteses o autômato vai para o estado $q1$ podendo este receber caracteres até a abertura ou fechamento de novos parenteses, podendo avançar ou retroceder entre os estados.

Dado a ilustração, caso o autômato se encerre nos estados $q1$, $q2$ *ou* $q3$ este retorna o erro de indentação dos parenteses, isto é, um ou mais parenteses foram abertos e não fechados. Caso se encerre no estado $q1$ retorna que a cadeia é de comentários, isto é, a abertura e fechamento está correta. Caso o estado $q0$ receba o fechamento de parenteses, esse é indicado como um erro, devido a não existir parenteses abertos e por fim, caso o estado $q3$ receba a abertura de um parenteses esse indica um estado de erro, dado que o máximo de parenteses já foi aberto.

2.4 Autômato Comentário

O autômato comentário descreve a formatação de um comentário, no inicio pode-se ter a abertura de um comentário, para isso, abre-se chaves ({).Caso houver um fecha-se chaves (}), o autômato entra em um estado de erro $q3$, retornando que não há um comentário aberto. No estado $q1$ acontece a inclusão recursiva de caracteres dentro deste comentário até o mesmo fechar chaves (}) e ir ao estado $q2$, onde este retorna que o comentário é válido. Se o estado $q1$ for um estado final, o autômato retorna um erro dizendo que este comentário não foi fechado.

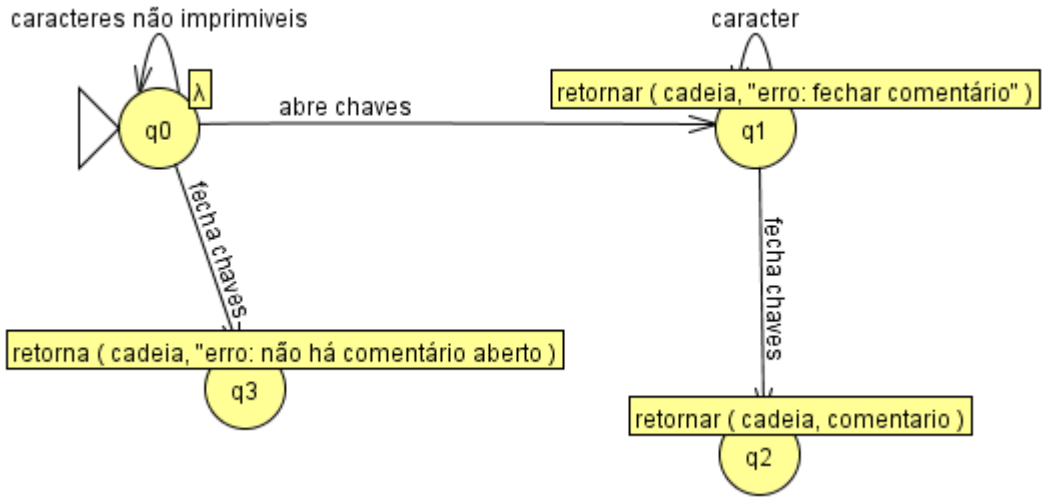


Figura 4: Autômato Comentário

2.5 Autômato Operadores

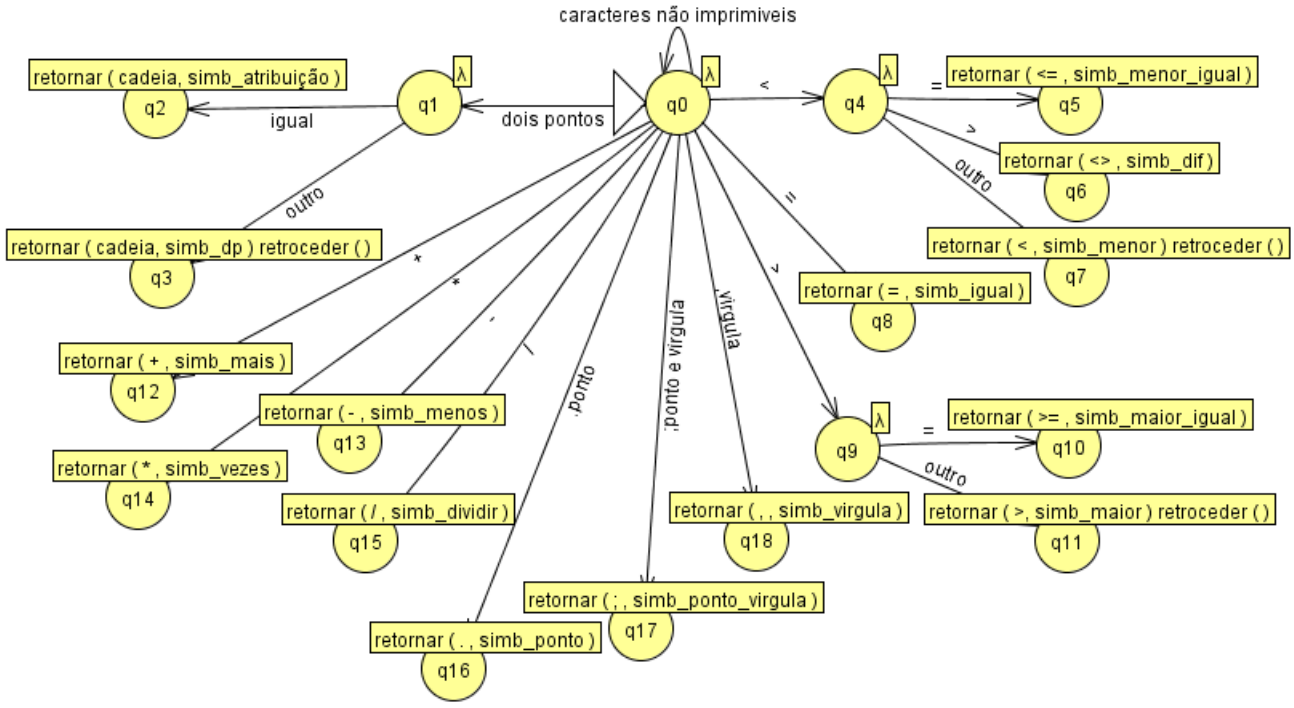


Figura 5: Autômato Operadores

Por último têm-se o autômato operadores, este identifica o operador que não foi tratado nos autômatos apresentados anteriormente. Apesar de seu tamanho maior que os demais, este se apresenta de uma forma simples. Caso ocorra a entrada de caracteres não imprimíveis, ele permanece em seu estado inicial, semelhante aos outros, mas caso receba a entrada de algum de seus operadores $(-, +, *, /, <=, >, etc...)$ o autômato identifica o operador e retorna seu token. Vale ressaltar que nos estados $q1$, $q4$ e $q9$, os operadores dois

pontos (:), menor (<) e maior (>), respectivamente, podem ser complementados por outros operadores, tendo como resultado um novo operador. Por exemplo, a junção dos operadores menor (<) e igual (=) resulta no operador menor e igual (<=) e outros os possíveis pares estão descritos no autômato.

3 Instruções para Compilar o Código-Fonte

Para preparar o terreno para a compilação, respeitando a linguagem utilizada, python, os passos essenciais do código fonte do analisador léxico- LexicalAnalyzer.py -, são basicamente ter a versão básica, python 3 ou superior, para que se possa ter acesso a um interpretador da linguagem utilizada e, assumindo que se tenha acesso a um terminal ou uma IDE para Python, basta rodar o código do compilador - compiler.py -para que o mesmo utilize o analisador léxico na entrada do arquivo txt a ser analisado. Por fim o mesmo gera a saida - saida.txt -com a análise feita pelo léxico, com o intuito de ser utilizado em futuras implementações. Abaixo temos um exemplo de execução, para tal basta utilizar a linha de comando a baixo:

```
1 python3 compiler.py <caminho/nome_do_arquivo_de_entrada.txt>
```

Código 2: Exemplo de Execução Via Terminal

4 Exemplo de Execução

4.1 Exemplo 1

```
program nome1;
{exemplo 1}
    var a, a1, b: integer;
begin
  read(a);
  a1:= a1*2;

  while (a1>0) do
  begin
    write(a1);
    a1:= a1-1;
  end;

  for b:=1 to 10 do
  begin
    b:=b+2;
    a:=a-1;
  end;
  if a<> b then write(a);
end.
```

Código 3: Exemplo Um

```
program, simb_program
nome1, ident
;, simb_ponto_virgula
var, simb_var
a, ident
,, simb_virgula
a1, ident
,, simb_virgula
b, ident
:, simb_dp
integer, simb_tipo
;, simb_ponto_virgula
begin, simb_begin
read, simb_read
(, simb_apar
a, ident
), simb_fpar
;, simb_ponto_virgula
a1, ident
:=, simb_atribuicao
a1, ident
*, simb_vezes
2, num_inteiro
;, simb_ponto_virgula
while, simb_while
(, simb_apar
a1, ident
>, simb_maior
0, num_inteiro
), simb_fpar
do, simb_do
begin, simb_begin
write, simb_write
(, simb_apar
a1, ident
), simb_fpar
;, simb_ponto_virgula
a1, ident
:=, simb_atribuicao
a1, ident
-, simb_menos
1, num_inteiro
;, simb_ponto_virgula
end, simb_end
;, simb_ponto_virgula
for, ident
b, ident
:=, simb_atribuicao
1, num_inteiro
to, simb_to
10, num_inteiro
do, simb_do
begin, simb_begin
b, ident
:=, simb_atribuicao
b, ident
+, simb_mais
2, num_inteiro
;, simb_ponto_virgula
a, ident
:=, simb_atribuicao
a, ident
-, simb_menos
1, num_inteiro
;, simb_ponto_virgula
end, simb_end
;, simb_ponto_virgula
if, simb_if
a, ident
<>, simb_dif
b, ident
then, simb_then
write, simb_write
(, simb_apar
a, ident
), simb_fpar
;, simb_ponto_virgula
end, simb_end
., simb_ponto
```

Código 4: Saída1.tex

4.2 Exemplo 2

```
program nome2;
{exemplo 2}
  var a: real;
  var b: integer;
procedure nomep(x: real);
  var a, c: integer;
begin
  read(c, a);
  if a<x+c then
  begin
    a:= c+x;
    write(a);
  end
  else c:= a+x;
  end;
  begin {programa principal}
    read(b);
    nomep(b);
end.
```

Código 5: Exemplo Dois

```
program, simb_program
nome2, ident
;, simb_ponto_virgula
var, simb_var
a, ident
:, simb_dp
real, simb_tipo
;, simb_ponto_virgula
var, simb_var
b, ident
:, simb_dp
integer, simb_tipo
;, simb_ponto_virgula
procedure,
    simb_procedure
nomep, ident
(, simb_apar
x, ident
:, simb_dp
real, simb_tipo
), simb_fpar
;, simb_ponto_virgula
var, simb_var
a, ident
,, simb_virgula
c, ident
:, simb_dp
integer, simb_tipo
;, simb_ponto_virgula
begin, simb_begin
read, simb_read
(, simb_apar
c, ident
,, simb_virgula
a, ident
), simb_fpar
;, simb_ponto_virgula
if, simb_if
a, ident
<, simb_menor
x, ident
+, simb_mais
c, ident
then, simb_then
begin, simb_begin
a, ident
:=, simb_atribuicao
c, ident
+, simb_mais
x, ident
;, simb_ponto_virgula
write, simb_write
(, simb_apar
a, ident
), simb_fpar
;, simb_ponto_virgula
end, simb_end
else, simb_else
c, ident
:=, simb_atribuicao
a, ident
+, simb_mais
x, ident
;, simb_ponto_virgula
end, simb_end
;, simb_ponto_virgula
begin, simb_begin
read, simb_read
(, simb_apar
b, ident
), simb_fpar
;, simb_ponto_virgula
nomep, ident
(, simb_apar
b, ident
), simb_fpar
;, simb_ponto_virgula
end, simb_end
., simb_ponto
```

Código 6: Saída2.tex

5 Códigos do Analisador Léxico

5.1 Código do Analisador Léxico

```
from pprint import pprint

class LexicalAnalyzer():
    # Tabela de Palavras reservadas da linguagem
    reserved_words_table = {
        'program': 'simb_program',
        'var': 'simb_var',
        'integer': 'simb_tipo',
        'real': 'simb_tipo',
        'begin': 'simb_begin',
        'end': 'simb_end',
        'while': 'simb_while',
        'read': 'simb_read',
        'write': 'simb_write',
        'const': 'simb_const',
        'procedure': 'simb_procedure',
        'else': 'simb_else',
        'then': 'simb_then',
        'if': 'simb_if',
        'do': 'simb_do',
        'to': 'simb_to'
    }

    # Lista de Todos os Tokens
    token_table = []

    def __init__(self, input_file):
        self.input_file = input_file

        line_number = 0

        buffer = ''

        try:
            with open(input_file, 'r') as fp:
                for line in fp:
                    line = line.replace('\t', '')

                    line_number += 1
                    char_position = -1

                    begin_comment, end_comment = self._commentary(line, line_number, char_position)

                    for character in line:
                        char_position += 1

                        if char_position >= begin_comment and char_position <= end_comment:
                            continue
                        elif character == ' ' or character == '\n' or self._is_operator(character):
```

```
50
51                          if buffer != '':
52                              output = self._number(buffer, line_number)
53
54                              if output is None:
55                                  if self._is_keyword(buffer):
56                                      output = (f'{buffer}, {self.
    reserved_words_table[buffer]}')
57                                  else:
58                                      output = self._identifier(buffer,
    line_number)
59
60                              self.token_table.append(output)
61
62                          if self._is_operator(character):
63                              ouput = self._operator(character, line,
    line_number, char_position)
64                              if ouput is not None:
65                                  self.token_table.append(ouput)
66
67                          buffer = ''
68
69                      else:
70                          buffer += character
71
72      except FileNotFoundError:
73          raise FileNotFoundError()
74
75  def _commentary(self, line, line_number, char_position):
76      begin = -1
77      end = -1
78
79      if '{' in line or '}' in line:
80          state = 0
81          for i in range(char_position, len(line)):
82              char_tmp = line[i]
83
84              if state == 0:
85                  if char_tmp == '{':
86                      state = 1
87                      begin = i
88                  elif char_tmp == '}':
89                      state = 3
90
91              elif state == 1:
92                  if i >= len(line) - 1:
93                      self.token_table.append(f'Comentario, ERRO: {
    line_number} - Fechar comentario')
94                      end = begin
95                  elif char_tmp == '}':
96                      state = 2
97
98              elif state == 2:
99                  end = i
100                 break
101
```

```
                elif state == 3:
                    self.token_table.append(f'Comentario, ERRO: {
line_number} - Fechamento de comentario sem abertura')
                    end = begin = i - 1

        return begin, end


    def _operator(self, character, line, line_number, char_position):
        output = None

        if character == ')':    output = '), simb_fpar'
        elif character == '(':
            state = 0

            for i in range(char_position, len(line)):
                char_tmp = line[i]

                if state == 0:
                    if i >= len(line) - 1:
                        output = '(, simb_apar'
                    elif char_tmp == '(':
                        state = 1
                    elif char_tmp == ')':
                        state = 4

                elif state == 1:
                    if i >= len(line) - 1 and char_tmp != ')':
                        line_tmp = line.replace("\n", "")
                        output = f'{line_tmp}, ERRO: {line_number} -
Identacao dos parenteses'
                    elif char_tmp == '(':
                        state = 2
                    elif char_tmp == ')':
                        state = 0

                elif state == 2:
                    if i >= len(line) - 1 and char_tmp != ')':
                        line_tmp = line.replace("\n", "")
                        output = f'{line_tmp}, ERRO: {line_number} -
Identacao dos parenteses'
                    elif char_tmp == '(':
                        state = 3
                    elif char_tmp == ')':
                        state = 1

                elif state == 3:
                    if i >= len(line) - 1 and char_tmp != ')':
                        line_tmp = line.replace("\n", "")
                        output = f'{line_tmp}, ERRO: {line_number} -
Identacao dos parenteses'
                    elif char_tmp == '(':
                        state = 5
                    elif char_tmp == ')':
                        state = 2

```

```
154                 elif state == 4:
155                     line_tmp = line.replace("\n", "")
156                     output = f'{line_tmp}, ERRO: {line_number} - Nao ha
    parenteses abertos'
157 
158                 elif state == 5:
159                     line_tmp = line.replace("\n", "")
160                     output = f'{line_tmp}, ERRO: {line_number} - Maximo de
     parenteses atingido'
161 
162         else:
163             state = 0
164             for i in range(char_position, len(line)):
165                 char_tmp = line[i]
166 
167                 if state == 0:
168                     if char_tmp == ':':
169                         state = 1
170                     elif char_tmp == '<':
171                         state = 4
172                     elif char_tmp == '=':
173                         if line[i - 1] != ':':
174                             state = 8
175                         else:
176                             state = -1
177                     elif char_tmp == '>':
178                         if line[i - 1] != '<':
179                             state = 9
180                         else:
181                             state = -1
182                     elif char_tmp == '+':
183                         state = 12
184                     elif char_tmp == '-':
185                         state = 13
186                     elif char_tmp == '*':
187                         state = 14
188                     elif char_tmp == '/':
189                         state = 15
190                     elif char_tmp == '.':
191                         state = 16
192                     elif char_tmp == ';':
193                         state = 17
194                     elif char_tmp == ',':
195                         state = 18
196 
197                 elif state == 1:
198                     if char_tmp == '=':
199                         state = 2
200                     else:
201                         state = 3
202 
203                 elif state == 2:
204                     output = ':=, simb_atribuicao'
205 
206                 elif state == 3:
207                     output = ':, simb_dp'
```

```
208
209             elif state == 4:
210                 if char_tmp == '=':
211                     state = 5
212                 elif char_tmp == '>':
213                     state = 6
214                 else:
215                     state = 7
216
217             elif state == 5:
218                 output = '<=, simb_menor_igual'
219
220             elif state == 6:
221                 output = '<>, simb_dif'
222
223             elif state == 7:
224                 output = '<, simb_menor'
225
226             elif state == 8:
227                 output = '=, simb_igual'
228
229             elif state == 9:
230                 if char_tmp == '=':
231                     state = 10
232                 else:
233                     state = 11
234
235             elif state == 10:
236                 output = '>=, simb_maior_igual'
237
238             elif state == 11:
239                 output = '>, simb_maior'
240
241             elif state == 12:
242                 output = '+, simb_mais'
243
244             elif state == 13:
245                 output = '-, simb_menos'
246
247             elif state == 14:
248                 output = '*, simb_vezes'
249
250             elif state == 15:
251                 output = '/, simb_dividir'
252
253             elif state == 16:
254                 output = '., simb_ponto'
255
256             elif state == 17:
257                 output = ';, simb_ponto_virgula'
258
259             elif state == 18:
260                 output = ',, simb_virgula'
261
262         return output
263
```

```
264
265     def _is_operator(self, character):
266         is_operator = False
267
268         if character == ';':    is_operator = True
269         elif character == ':':  is_operator = True
270         elif character == '+':  is_operator = True
271         elif character == '-':  is_operator = True
272         elif character == '*':  is_operator = True
273         elif character == '/':  is_operator = True
274         elif character == '(':  is_operator = True
275         elif character == ')':  is_operator = True
276         elif character == '=':  is_operator = True
277         elif character == ',':  is_operator = True
278         elif character == '>':  is_operator = True
279         elif character == '<':  is_operator = True
280         elif character == '.':  is_operator = True
281
282
283         return is_operator
284
285     def _is_keyword(self, buffer):
286         if self.reserved_words_table.get(buffer, None) is None:
287             return False
288         else:
289             return True
290
291
292     def _number(self, buffer, line_number):
293         state = 0
294         output = None
295         size_count = 0
296
297         for i in range(len(buffer)):
298             character = buffer[i]
299
300             if state == 0:
301                 if character >= '0' and character <= '9':
302                     state = 2
303                     size_count += 1
304
305                     if len(buffer) == 1:
306                         output = f'{buffer}, num_inteiro'
307
308                 elif character == '-' or character == '+':
309                     state = 1
310                 elif character > ' ':
311                     state = 5
312
313             elif state == 1:
314                 if character >= '0' and character <= '9':
315                     state = 2
316                     size_count += 1
317                 else:
318                     state = 6
319
```

```
320             elif state == 2:
321                 if size_count >= 32:
322                     state = 4
323                 elif i >= len(buffer) - 1:
324                     if character >= '0' and character <= '9':
325                         output = f'{buffer}, num_inteiro'
326                     else:
327                         output = f'{buffer}, ERRO: {line_number}:{buffer}
    - Numero mal formado'
328                     break
329                 elif character == '.':
330                     state = 3
331                 elif character >= '0' and character <= '9':
332                     size_count += 1
333                 else:
334                     state = 6
335 
336             elif state == 3:
337                 if size_count >= 32:
338                     state = 4
339                 elif i >= len(buffer) - 1:
340                     if character >= '0' and character <= '9':
341                         output = f'{buffer}, num_real'
342                     else:
343                         output = f'{buffer}, ERRO: {line_number}:{buffer}
    - Numero mal formado'
344                     break
345                 elif character >= '0' and character <= '9':
346                     size_count += 1
347                 else:
348                     state = 6
349 
350             elif state == 4:
351                 output = f'{buffer}, ERRO: {line_number}:{buffer} - Numero
     com excesso de tamanho'
352                 break
353 
354             elif state == 6:
355                 output = f'{buffer}, ERRO: {line_number}:{buffer} - Numero
     mal formado'
356                 break
357 
358         return output
359 
360     def _identifier(self, buffer, line_number):
361         state = 0
362         output = None
363         size_count = 0
364 
365         for i in range(len(buffer)):
366             char = buffer[i]
367 
368             if state == 0:
369                 if (char >= 'A' and char <= 'Z') or (char >= 'a' and char
    <= 'z'):
370                     state = 1
```

```
                    size_count += 1

                    if len(buffer) == 1:
                        output = f'{buffer}, ident'

                elif char > ' ':
                    state = 2

            elif state == 1:
                if size_count >= 32:
                    state = 3
                elif i >= len(buffer) - 1:
                    output = f'{buffer}, ident'
                    break
                else:
                    size_count += 1

            elif state == 2:
                output = f'{buffer}, ERRO: {line_number}:{buffer} - Identificador com caracter invalido'
                break

            elif state == 3:
                output = f'{buffer}, ERRO: {line_number}:{buffer} - Identificador com excesso de tamanho'
                break

        return output


    def get_token_table(self):
        '''
            Retorna a tabela contendo todos os Tokens coletados pelo Analisador Lexico
        '''
        return self.token_table

    def get_token(self, position):
        '''
            Retorna Token que esta na posicao passada
        '''
        return self.token_table[position]

    def get_token_table_size(self):
        '''
            Retorna o tamanho da tabela de Tokens
        '''
        return len(self.token_table)
```

Código 7: Analisador Léxico

5.2 Código de Compilação do Analisador Léxico

```
import sys
from pprint import pprint

from LexicalAnalyzer import LexicalAnalyzer

def main():
    try:
        if len(sys.argv) < 2:
            raise Exception('ERRO: Caminho para arquivo codigo fonte nao digitado.')

        lexAnalyzer = LexicalAnalyzer(sys.argv[1])

        with open('saida.txt', 'w') as fp:
            for item in lexAnalyzer.get_token_table():
                fp.write(item + '\n')
    except FileNotFoundError:
        print('ERRO: Arquivo nao existe.')
    except Exception as ex:
        print(ex)

if __name__ == '__main__':
    main()
```

Código 8: Código de Compilação