Projeto de Implementação de um Compilador para a Linguagem T++
Análise Léxica (Trabalho – 1ª parte)

Gabriel Negrão Silva[1]
[1] Universidade Tecnológica Federal do Paraná - Campo Mourão (UTFPR-CM)
Campo Mourão – PR – Brasil, 12 de Setembro de 2017

itsg_negrao@hotmail.com

Resumo. *Este artigo descreve como ocorre o processo de análise léxica de um compilador construindo um analisador para a linguagem* ***T++*** *com o auxílio da ferramento PLY, tal como procedimentos, alterações, instruções, exemplos e análise dos resultados.*

1. Introdução

O trabalho proposto foi aplicado de forma gradual, sendo constituído de 4 partes principais, neste artigo discorreremos sobre a primeira parte deste trabalho proposto. Este relatório tem como objetivo mostrar como foi projetado e construído o algoritmo com o auxílio de uma ferramenta (PLY) que discorreremos sobre posteriormente junto com uma conclusão final. O código está identado e comentado, instruções auxiliares são dadas pelo no decorrer do artigo anexado.

2. Descrição e Procedimentos

2.1. Descrição da Linguagem (T++/TPP)

A linguagem T++ ou Tpp (T Plus Plus) é uma linguagem baseada em C++ que será utilizada com o propósito de ser uma linguagem didática na disciplina de compiladores para que haja entendimento sobre a matéria e sua base, uma vez que a linguagem é escrita em Português-BR a mesma é de fácil aprendizagem e análise de certa forma, por não ser uma linguagem conhecida não há material na internet sobre a mesma, somente a documentação fornecida pelos criadores e professores da matéria.

É necessário para implementar o scanner da linguagem ter conhecimento de suas palavras reservadas e símbolos fornecidas, que são (Tabela 1).

palavras reservadas	símbolos
se	+ soma
então	- subtração
senão	* multiplicação
fim	/ divisão
repita	= igualdade
flutuante	, vírgula
retorna	:= atribuição
até	< menor
leia	> maior
escreve	<= menor-igual
inteiro	>= maior-igual
	(abre-par
	(fecha-par
	: dois-pontos
	[abre-col
	] fecha-col
	&& e-logico
	! negação

Tabela 1. Tokens da linguagem T++.

2.2. Descrição da Ferramenta Auxiliar (PLY)

A ferramenta PLY (Python Lex-Yacc) que contém analisador léxico e sintático é programada na linguagem Python e é a ferramenta que dara suporte a toda base do projeto e pelo qual o programa criado será amparado e auxiliado. A ferramenta é como uma API embutida, e sua documentação até que é clara e objetiva auxiliando no aprendizado e utilização da mesma.

A versão original do PLY foi desenvolvida em 2001 para uso em um curso de Introdução ao Compilador, onde os alunos usaram para criar um compilador para uma linguagem simples semelhante a Pascal. O PLY-3.0 adiciona suporte para o Python 3.0 e fornece aos internos do PLY uma revisão muito necessária. O código deve ser executado com **Python 3**, para o bom funcionamento do mesmo.

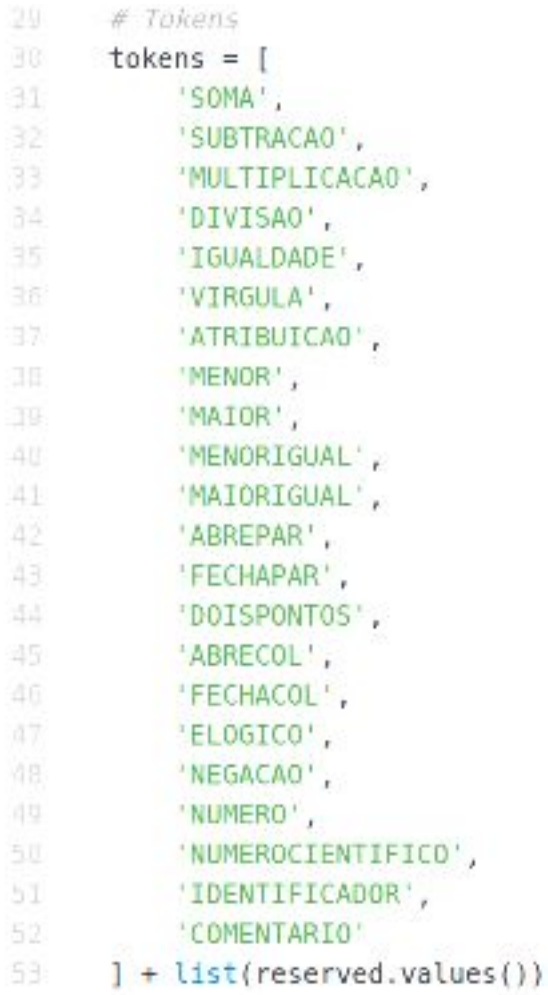
2.3. Descrição da Implementação

Começaremos a dissertar sobre projeto de scanner léxico da linguagem neste trecho do código com a importação da ferramenta PLY.LEX (analisador léxico) denominado dentro do projeto como *'lexerNegrao.py'* e a seguir o trecho do programa que descreve como a ferramenta interpreta as palavras que serão as palavras 'reservadas' da linguagem a ser analisada. A palavras são inseridas em um array juntamente com seus respectivos tokens (Figura 1).

```
8   # Importação ply lexer
9   import ply.lex as lex
10  import sys
11
12  #Palavras reservadas
13  reserved = {
14      'se' : 'SE',
15      'então' : 'ENTAO',
16      'senão' : 'SENAO',
17      'fim' : 'FIM',
18      'repita' : 'REPITA',
19      'flutuante' : 'FLUTUANTE',
20      'retorna' : 'RETORNA',
21      'leia' : 'LEIA',
22      'até' : 'ATE',
23      'vazio' : 'VAZIO',
24      'escreva' : 'ESCREVA',
25      'inteiro' : 'INTEIRO',
26      'principal' : 'PRINCIPAL'
27  }
```

Figura 1. Trecho do Programa Comentado. 1ª parte

A seguir temos o trecho do código responsável por concatenar todos os tokens a serem utilizados para quebra do código com as palavras reservadas previamente estabelecidas na imagem anterior (Figura 2).



```
29    # Tokens
30    tokens = [
31        'SOMA',
32        'SUBTRACAO',
33        'MULTIPLICACAO',
34        'DIVISAO',
35        'IGUALDADE',
36        'VIRGULA',
37        'ATRIBUICAO',
38        'MENOR',
39        'MAIOR',
40        'MENORIGUAL',
41        'MAIORIGUAL',
42        'ABREPAR',
43        'FECHAPAR',
44        'DOISPONTOS',
45        'ABRECOL',
46        'FECHACOL',
47        'ELOGICO',
48        'NEGACAO',
49        'NUMERO',
50        'NUMEROCIENTIFICO',
51        'IDENTIFICADOR',
52        'COMENTARIO'
53    ] + list(reserved.values())
```

Figura 2. Trecho do Programa Comentado. 2ª parte.

Definimos nesta parte a declaração das expressões regulares para associar aos respectivos tokens sendo ela por atribuição explícita ou por meio de uma função (mais indicado para casos em que a expressão regular é extensa ou envolvem operações/tratamentos complexos) tal como a função de reconhecimento de números científicos (t_NUMEROCIENTIFICO) como por exemplo *'2e+10'*. Também definimos duas funções padrões da ferramenta PLY nesta parte, a de contagem de linhas (t_newline) presente na linha 97 do código a seguir e a de erro (t_erro) presente na linha 101 do código a seguir responsável por emitir um erro com o caractere encontrado (Figura 3 e 4).

```
55  #Declaração das expressões regulares
56  t_SOMA = r'\+'
57  t_MULTIPLICACAO = r'\*'
58  t_DIVISAO = r'/'
59  t_IGUALDADE = r'\='
60  t_VIRGULA = r'\,'
61  t_ATRIBUICAO = r'\:\='
62  t_MENOR = r'\<'
63  t_MAIOR = r'\>'
64  t_MENORIGUAL = r'\<\='
65  t_MAIORIGUAL = r'\>\='
66  t_ABREPAR = r'\('
67  t_FECHAPAR = r'\)'
68  t_DOISPONTOS = r'\:'
69  t_ABRECOL = r'\['
70  t_FECHACOL = r'\]'
71  t_ELOGICO = r'\&\&'
72  t_NEGACAO = r'\!'
73  t_ignore = " \t"
74
75  #Trata Numeros cientificos
76  def t_NUMEROCIENTIFICO(t):
77      r'[\d]+\.?[\d]*[Ee](?:[-+]?[\d]+)?'
78      return t
79
80  def t_NUMERO(t):
81      r'\d+(\.\d+)?'
82      return t
```

Figura 3. Trecho do Programa Comentado. 3ª Parte

```
84    def t_SUBTRACAO(t):
85        r'\-'
86        return t
87
88    def t_IDENTIFICADOR(t):
89        r'[a-zA-Z-à-ú][ a-zA-Z 0-9-à-ú]*'
90        t.type = reserved.get(t.value,'IDENTIFICADOR')
91        return t
92
93    def t_COMENTARIO(t):
94        r'\{[^}]*[^{]*\}'
95        return t
96
97    def t_newline(t):
98        r'\n+'
99        t.lexer.lineno += len(t.value)
100
101   def t_error(t):
102       print("Caracter Ilegal '%s'" % t.value[0])
103       t.lexer.skip(1)
104
```

Figura 4. Trecho do Programa Comentado. 4ª Parte

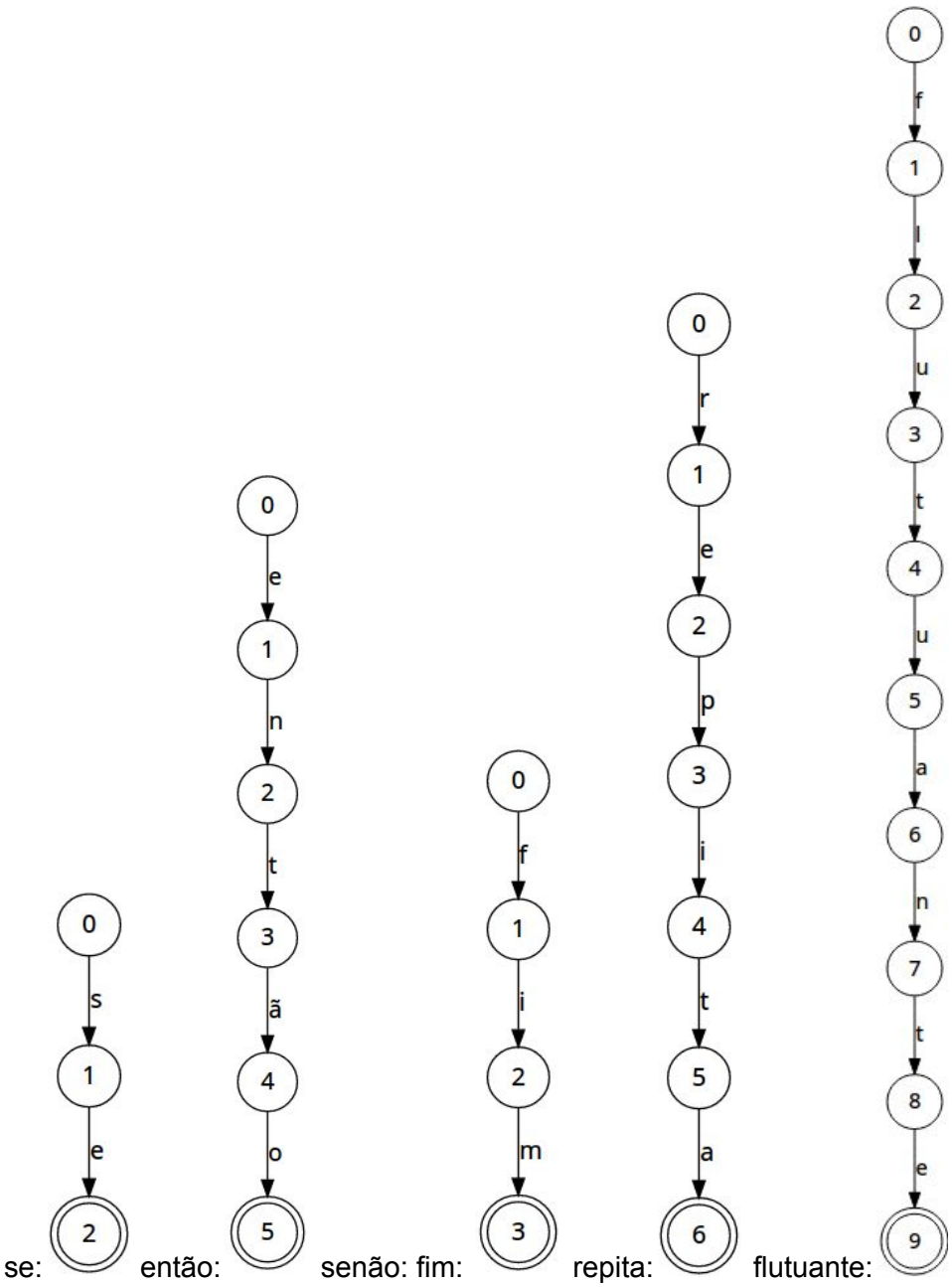
Neste último trecho é definido a função main que é responsável por receber e atribuir o arquivo recebido por parâmetro pelo terminal de comando para o analisador léxico aqui chamado de *'lexer'* e o resultado salvo em um arquivo de saída com o nome do arquivo a ser analisado sem a extensão e com o final para indicar o arquivo de saída *"_Tokens.out"* (Figura 5).

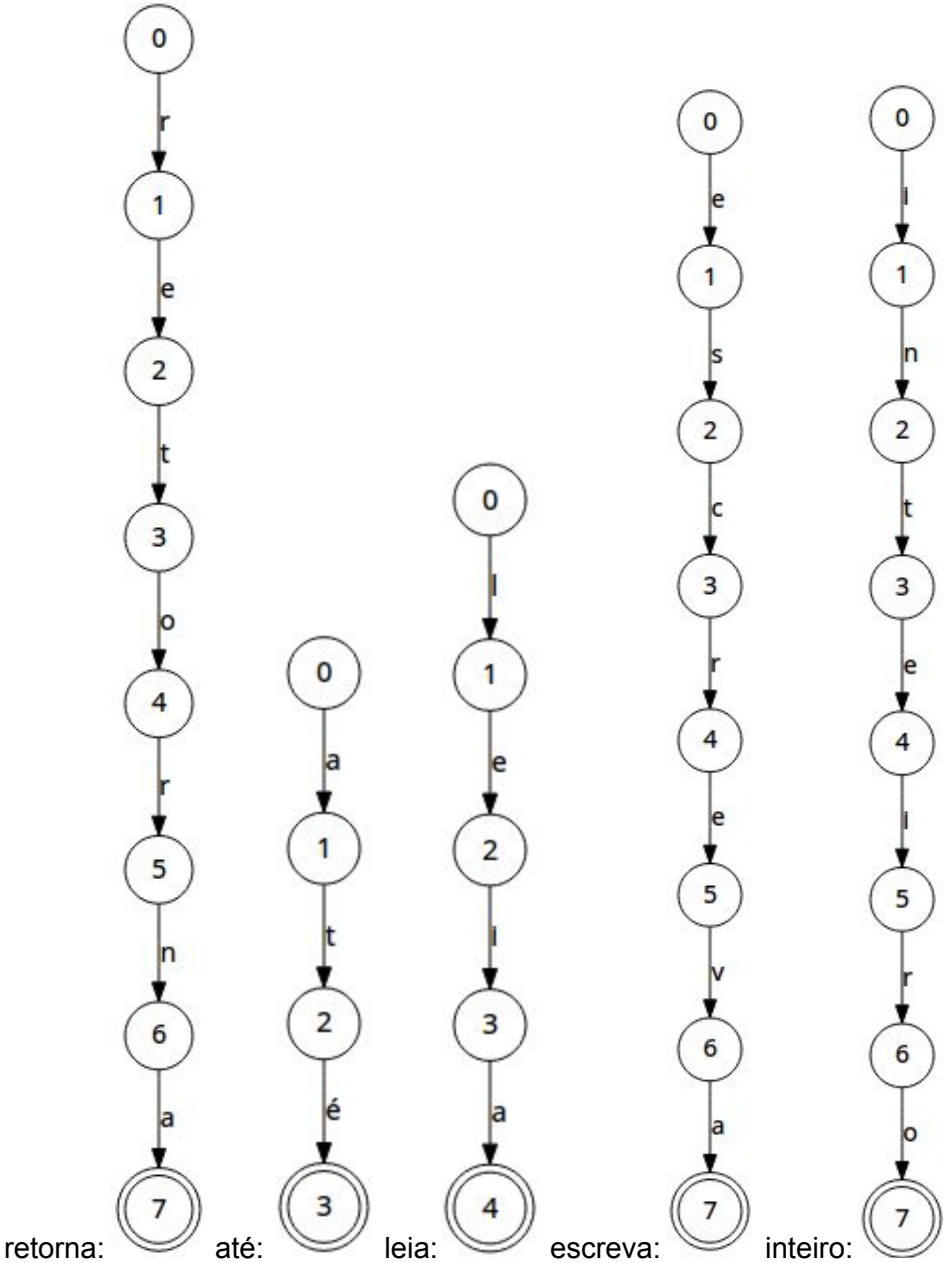
```
105  def main():
106      #abre o arquivo
107      arq = open(sys.argv[1],'r')
108      saida = open(sys.argv[1].replace(".tpp","")+"_Tokens.out", 'w')
109      data = arq.read()
110
111      lexer = lex.lex()
112
113      #Atribui ao lexer o arquivo para tokenizar
114      lexer.input(data)
115
116      while 1:
117          result = lexer.token()  #Tokeniza as strings de entrada
118          if not result:
119              break      # caso não houver entradas
120          saida.write(result.type+" "+result.value+"\n")
121      print("O Analisador Finalizou.")
122      saida.close()
123      arq.close()
124
125  #Define a execução pelo terminal
126  if __name__ == "__main__":
127      main()
```

Figura 5. Trecho do Programa Comentado. Ultima Parte

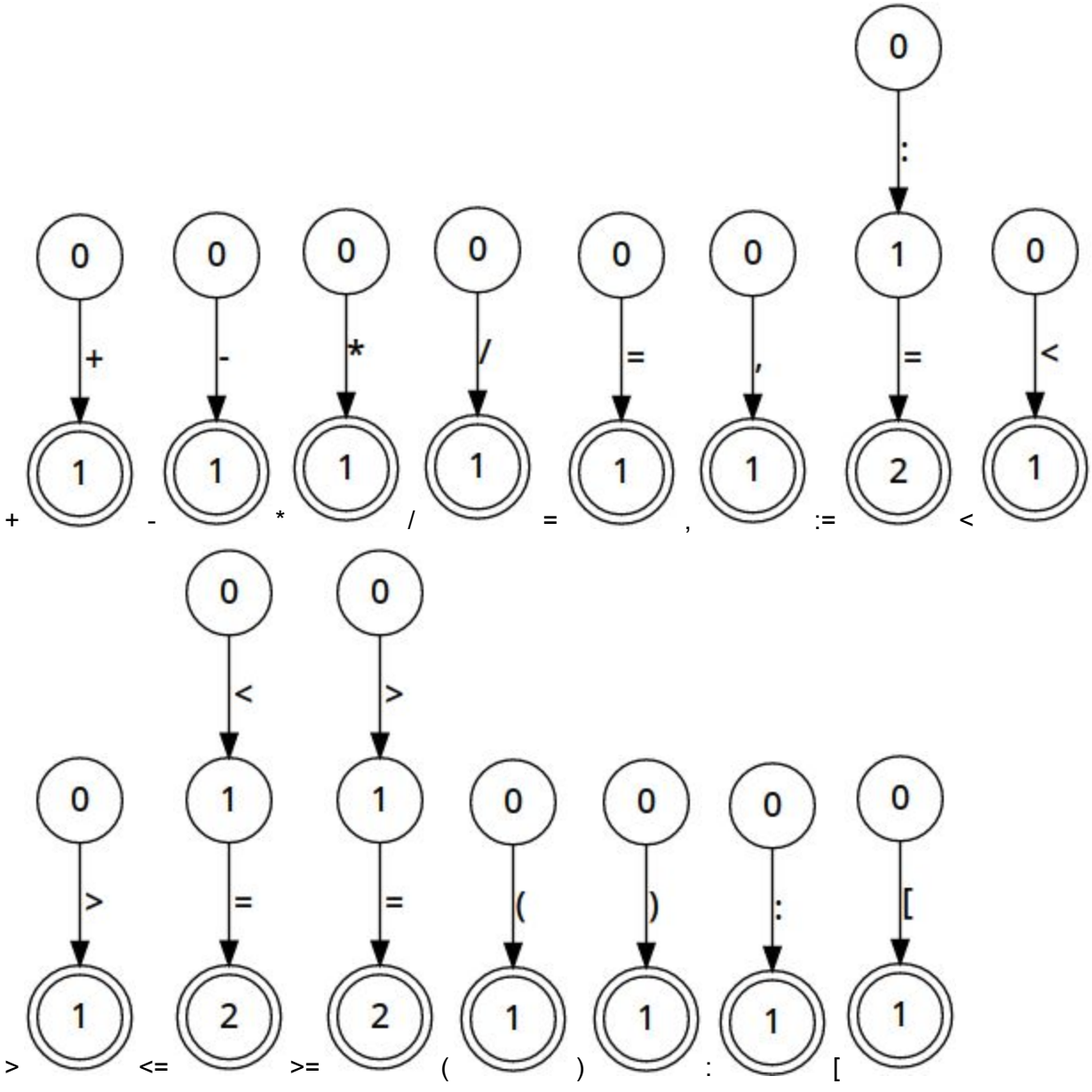
2.4. Automatos da Linguagem

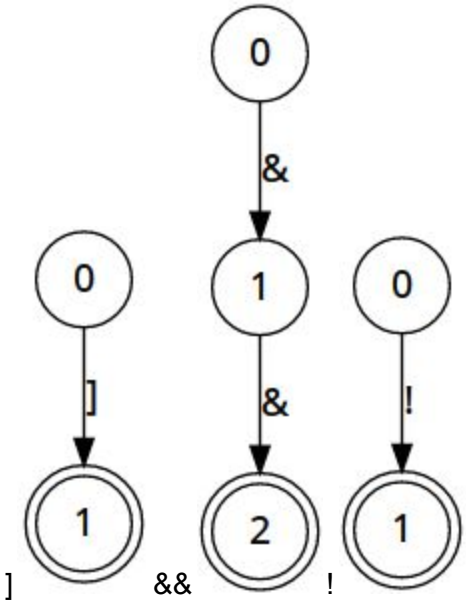
2.4.1. Palavras Reservadas




retorna:
0
r
1
e
2
t
3
o
4
r
5
n
6
a
7
até:
0
a
1
t
2
é
3
leia:
0
l
1
e
2
i
3
a
4
escreva:
0
e
1
s
2
c
3
r
4
e
5
v
6
a
7
inteiro:
0
i
1
n
2
t
3
e
4
i
5
r
6
o
7

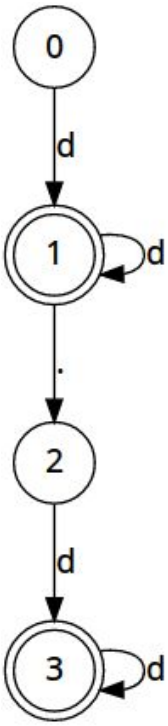
2.4.2. Símbolos



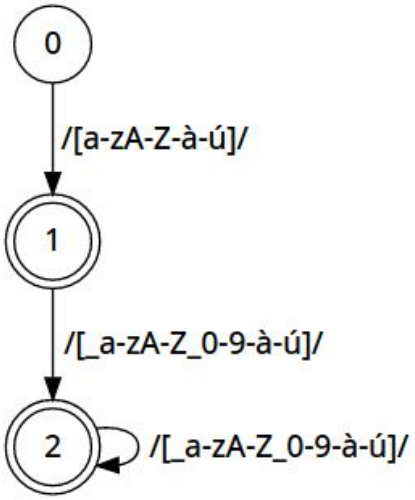


] && !

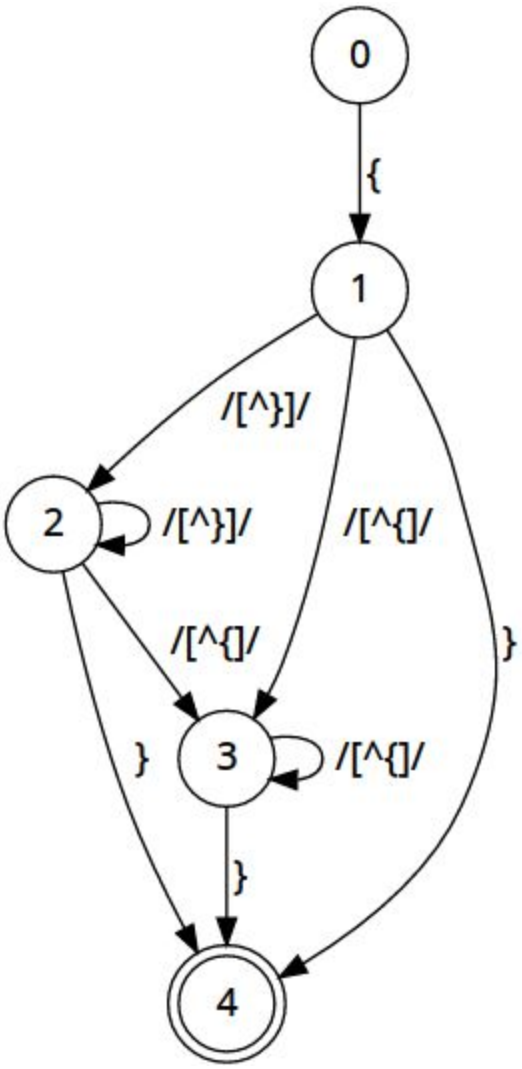
2.4.3. Outras Expressões

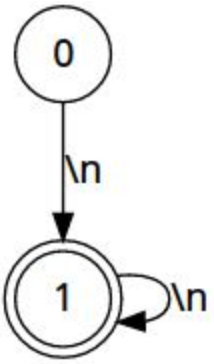


Número:



Identificador:





Comentário: \n

3. Resultados e Análise

Dado o algoritmo de teste escrito na linguagem T++ o qual recebe um número inteiro e escreve seu fatorial na tela como entrada para o analisador léxico programando em Python para a linguagem. (Figura 6).

O algoritmo a seguir (Figura 6) escrito na linguagem T++ anexado ao projeto, o qual escreve na tela o fatorial de um número inteiro n, é a entrada de teste para o analisador léxico programado em Python com auxílio da ferramenta PLY executado via terminal com **Python 3**.

```
inteiro: n
inteiro: b

inteiro fat( flutuante: a, inteiro: b)
    a:= 10.5
fim

inteiro fatorial(inteiro: n)
    inteiro: fat
    se n > 0 então {não calcula se n > 0}
        fat := 1
        repita
            fat := fat * n
            n := n - 1
        até n = 0
        retorna(fat) {retorna o valor do fatorial de n}
    senão
        retorna(0)
    fim
fim

inteiro principal()
    leia(n)
    escreva(fatorial(n))
    escreva(fat(1,1))
fim
```

Figura 6. Algoritmo de calculo fatorial em t++.

A saída do programa criado na linguagem Python para a análise léxica da linguagem T++ usando o programa de teste escrito na linguagem T++ incorporado ao projeto é dada no seguinte formato, podendo ser customizado, o identificador respectivo ao tipo do token, seguido pelo valor do token, tal como exemplo, “NUMERO 1024” que corresponde a um número contido no programa seja ele do tipo inteiro ou ponto flutuante , a saída final executando o programa *‘lexerNegrao.py’* via terminal de comando utilizando **Python 3** e passando o programa **teste-1.tpp** como parâmetro foi a seguinte (Figuras 7, 8, 9, 10 e 11).

Exemplo de execução via terminal:

---

root@root-PC:~/Lex$ python3 lexerNegrao.py teste-1.tpp
O Analisador Finalizou.

```
1   INTEIRO inteiro
2   DOISPONTOS :
3   IDENTIFICADOR n
4   INTEIRO inteiro
5   DOISPONTOS :
6   IDENTIFICADOR b
7   INTEIRO inteiro
8   IDENTIFICADOR fat
9   ABREPAR (
10  FLUTUANTE flutuante
11  DOISPONTOS :
12  IDENTIFICADOR a
13  VIRGULA ,
14  INTEIRO inteiro
15  DOISPONTOS :
16  IDENTIFICADOR b
17  FECHAPAR )
```

Figura 7. Trecho da Saída “teste-1_Tokens.out”. 1ª Parte.

```
18  IDENTIFICADOR a
19  ATRIBUICAO :=
20  NUMERO 10.5
21  FIM fim
22  INTEIRO inteiro
23  IDENTIFICADOR fatorial
24  ABREPAR (
25  INTEIRO inteiro
26  DOISPONTOS :
27  IDENTIFICADOR n
28  FECHAPAR )
29  INTEIRO inteiro
30  DOISPONTOS :
31  IDENTIFICADOR fat
32  SE se
33  IDENTIFICADOR n
34  MAIOR >
```

Figura 8. Trecho da Saída "teste-1_Tokens.out". 2ª Parte.

```
35  NUMERO 0
36  ENTAO então
37  COMENTARIO {não calcula se n > 0}
38  IDENTIFICADOR fat
39  ATRIBUICAO :=
40  NUMERO 1
41  REPITA repita
42  IDENTIFICADOR fat
43  ATRIBUICAO :=
44  IDENTIFICADOR fat
45  MULTIPLICACAO *
46  IDENTIFICADOR n
47  IDENTIFICADOR n
48  ATRIBUICAO :=
49  IDENTIFICADOR n
50  SUBTRACAO -
51  NUMERO 1
52  ATE até
```

Figura 9. Trecho da Saída "teste-1_Tokens.out". 3ª Parte.

```
53  IDENTIFICADOR n
54  IGUALDADE =
55  NUMERO 0
56  RETORNA retorna
57  ABREPAR (
58  IDENTIFICADOR fat
59  FECHAPAR )
60  COMENTARIO {retorna o valor do fatorial de n}
61  SENAO senão
62  RETORNA retorna
63  ABREPAR (
64  NUMERO 0
65  FECHAPAR )
66  FIM fim
67  FIM fim
68  INTEIRO inteiro
```

Figura 10. Trecho da Saída “teste-1_Tokens.out”. 4ª Parte.

```
69  PRINCIPAL principal
70  ABREPAR (
71  FECHAPAR )
72  LEIA leia
73  ABREPAR (
74  IDENTIFICADOR n
75  FECHAPAR )
76  ESCREVA escreva
77  ABREPAR (
78  IDENTIFICADOR fatorial
79  ABREPAR (
80  IDENTIFICADOR n
81  FECHAPAR )
82  FECHAPAR )
83  ESCREVA escreva
84  ABREPAR (
85  IDENTIFICADOR fat
86  ABREPAR (
87  NUMERO 1
88  VIRGULA ,
89  NUMERO 1
90  FECHAPAR )
91  FECHAPAR )
92  FIM fim
```

Figura11. Trecho da Saída “teste-1_Tokens.out”. Ultima Parte.

4. Discussão

A saída dos teste executados foram satisfatórias, tal como este demonstrado anteriormente. Há mais algoritmos programados em T++ para testes do analisador e o resultado obtido com a execução dos mesmos foram satisfatoriamente boas.

Os experimentos foram realizados em computador pessoal móvel (Notebook) próprio com processador intel quad core e seu tempo de execução foi baixo e olho nú aparentemente de término instantâneo dado tamanho do teste de entrada.

A ferramenta de análise léxica PLY foi de grande ajuda e serviu como base para inicio deste projeto, encapsulando certas funções autônomas das quais o programador não precisa se preocupar como funcionam internamente, não necessitando implementação própria do programador também.

5. Conclusão

A parte complicada, se é que houve, é a de integração do conteúdo com o projeto implementado, tal como, expressões regulares e autômatos vistos em aula e implementados com a ferramenta PLY, após ocorrido o entendimento do mesmo, a programação se torna mais fluida. Este projeto teve como contribuição o entendimento e auxílio inicial a disciplina de Compiladores, das funcionalidades e construção de um compilador e das ferramentas de suporte ao projeto, mais especificamente a parte de um analisador léxico, parte integrante de um compilador.