

Análise Léxica da linguagem TPP

Henrique S. Marcuzzo¹

¹Departamento de Ciência da Computação
– Universidade Tecnológica Federal do Paraná (UTFPR)

henriquemarcuzzo@alunos.utfpr.edu.br

Abstract. *This article describes the steps taken to perform the lexical analysis work, with a brief explanation of the fictitious language TPP, how the regular expressions were generated, details of how the code was produced, and examples of running the produced code.*

Resumo. *Este artigo descreve os passos tomadas para a execução do trabalho de análise léxica, com um breve explicação sobre a linguagem fictícia TPP, como foram gerados as expressões regulares, detalhes de como foi produzido o código, além de exemplos de execução do código produzido.*

1. Introdução

Este trabalho foi realizado com a linguagem de programação *Python* juntamente com a biblioteca *PLY*, para a Análise léxica dos códigos `.tpp`, durante o processo foram criadas e aprimoradas algumas expressões regulares para que atendessem as necessidade da linguagem de programação fictícia da disciplina **TPP**.

2. Especificação da linguagem de programação T++

A linguagem *T++*, foi construída na disciplina de compiladores, com intuito didático, para proporcionar aos alunos uma noção de como funciona a construção de um compilador, portanto a linguagem estudada oferece apenas as estruturas mais simples de uma linguagem convencional, que são:

- Número em notação científica
- Número flutuante
- Número inteiro
- Identificadores (variáveis ou nome de funções)
- Operadores binários
 - Mais
 - Menos
 - Multiplicação
 - Divisão
 - E lógico
 - OU lógico
 - Diferente
 - Menor Igual
 - Menor
 - Maior Igual
 - Maior

- Igual
- Operadores unitários
 - Negação
- Símbolos
 - Abre Parenteses
 - Fecha Parenteses
 - Abre Colchetes
 - Fecha Colchetes
 - Virgula
 - Dois Pontos
 - Atribuição
- Palavras Reservadas
 - se
 - então
 - senão
 - fim
 - repita
 - até
 - flutuante
 - inteiro
 - retorna
 - leia
 - escreva

Além disso, também é necessário que os arquivos tenha extensão .tpp.

3. Especificação formal dos autômatos para a formação de cada classe de token da linguagem

Para especificar os autômatos, haverá uma separação em dois grupos, os tokens simples, que são aqueles que necessitam apenas da identificação de símbolos, operadores lógicos e relacionais, e os tokens mais complexos, que são os identificadores, comentários, quebra de linha, e números, seja inteiro, flutuante ou notação científica.

Vale lembrar também que todas as Expressões Regulares estão no formato aceito pelo *Regex*.

3.1. Tokens Simples

Nesta seção a geração de cada token consiste em identificar símbolos, em sequência ou isolados, portanto não se faz necessário uma explicação individual.

Assim os tokens e suas expressões regulares são:

- Símbolos
 - Mais
TOKEN: MAIS
Expressão Regular: \+

- Menos
TOKEN: MENOS
Expressão Regular: -
- Multiplicação
TOKEN: VEZES
Expressão Regular: *
- Divisão
TOKEN: DIVIDE
Expressão Regular: /
- Abre Parenteses
TOKEN: ABRE_PARENTESE
Expressão Regular: \(
- Fecha Parenteses
TOKEN: FECHA_PARENTESE
Expressão Regular: \)
- Abre Colchetes
TOKEN: ABRE_COLCHETE
Expressão Regular: \[
- Fecha Colchetes
TOKEN: FECHA_COLCHETE
Expressão Regular: \]
- Virgula
TOKEN: VIRGULA
Expressão Regular: ,
- Dois Pontos
TOKEN: DOIS_PONTOS
Expressão Regular: :
- Atribuição
TOKEN: ATRIBUICAO
Expressão Regular: :=

- Operadores Lógicos

- E lógico
TOKEN: E
Expressão Regular: &&
- OU lógico
TOKEN: OU

Expressão Regular: \\|

- Negação

TOKEN: NAO

Expressão Regular: !

- Operadores Relacionais

- Diferente

TOKEN: DIFERENTE

Expressão Regular: <>

- Menor Igual

TOKEN: MENOR_IGUAL

Expressão Regular: <=

- Menor

TOKEN: MENOR

Expressão Regular: <

- Maior Igual

TOKEN: MAIOR_IGUAL

Expressão Regular: >=

- Maior

TOKEN: MAIOR

Expressão Regular: >

- Igual

TOKEN: IGUAL

Expressão Regular: =

3.2. Tokens Complexos

Nesta seção, como cada token requer uma abordagem diferente haverá uma explicação individual das expressões regulares, a fim de deixar claro o motivo de cada abordagem.

- Comentário

TOKEN: (Nenhum token é gerado, pois comentários são ignorados na hora de compilação)

Expressão Regular: (\((.|\\n)*?) \\)

Explicação: A partir do momento que for identificado um abre chave '{', tudo é aceito e considerado como comentário, inclusive quebras de linhas, até o momento que identifica-se o primeiro fecha chaves '}'

- Identificador

TOKEN: ID

Expressão Regular: ((([a-zA-ZáÁãÃàÀéÉíÍóÓõÕ]))((([a-zA-ZáÁãÃàÀéÉíÍóÓõÕ])|_([0-9]))*))

Explicação: A partir do momento que for identificado uma letra (([a-zA-ZáÂãÄÀÉÉÍóÓõÖ])), é aceite 0 ou quantidades de ilimitadas de letras, dígitos ([0-9]) ou *underline*.

- Número flutuante

TOKEN: NUM_PONTO_FLUTUANTE

Expressão Regular: [\d]+\.[\d]*

Explicação: Para ser um número flutuante, deve ter no mínimo um dígito (entre 0 e 9), após o dígito é necessário ter o símbolo ponto "." e por fim é necessário ser seguido de zero ou mais dígitos.

- Número inteiro

TOKEN: NUM_INTEIRO

Expressão Regular: [\d]+

Explicação: Para ser um número inteiro, deve haver no mínimo um dígito (entre 0 e 9).

- Número em notação científica

TOKEN: NUM_NOTACAO_CIENTIFICA

Expressão Regular: ([\d][\.]?[\d]*[eE][+-]?[\d]+)

Explicação: São aceites números inteiro ou flutuante (entre 0 e 9) seguido de *e* ou *E*, e após isso pode haver ou não outro sinal seguido de um número inteiro (entre 0 e 9).

- Nova linha

TOKEN: (Nenhum token é gerado, pois quebras de linha são ignorados na hora de compilação)

Expressão Regular: \n+

Explicação: Sempre que encontrar um ou mais \n, é identificado como quebra de linha.

- Erro

TOKEN: (Nenhum token é gerado)

Expressão Regular: (Nenhuma)

Explicação: Sempre que encontrar algum carácter ou sequências de caracteres que não atendem nenhuma das especificações é gerado um erro léxico e detectado por essa função.

4. Detalhes da implementação escolhidas pelo projetista

Para a execução deste trabalho foi escolhido a linguagem de programação *Python* e a biblioteca *PLY Lex* (Python Lex-Yacc).

O código gerado, se fundamentou principalmente no código de início disponibilizado pelo professor, as mudanças que ocorreram foram nas melhorias das expressões regulares de números inteiros, flutuantes e notação científica, além de tornar o código orientado a objetos, tornando-o mais estruturado e separado por classes.

5. Exemplos de saída do sistema de varredura

Ao executar o código é possível passar um parâmetro opcional chamado *detailed*, que altera a saída do programa, onde sem a presença deste parâmetro será impresso apenas a lista de tokens do código e com a presença deste parâmetro a saída, será a padrão do *Analísadores Léxico PLY*.

Portanto, para executar o código é necessário digitar no diretório `BCC__BCC36B__P[1]__HenriqueMarcuzzo_2046334/impl/` o seguinte comando `python3 lex_submit.py code.tpp [detailed]`, note que *detailed* está em colchetes pois é opcional, e o `code.tpp` pode ser qualquer código com a extensão `.tpp`, sendo possível encontrar alguns no diretório `BCC__BCC36B__P[1]__HenriqueMarcuzzo_2046334/impl/testes_lexico/`.

Sendo assim, a imagem 1 está sendo executado sem o parâmetro opcional e a imagem 2 com o parâmetro opcional.

```
hmarcuzzo@hmarcuzzo-63-3590:~/Documentos/GitHub/compiler_project/BCC__BCC36B__P[1]__HenriqueMarcuzzo_2046334/impl$ python3 lex_submit.py testes_lexico/teste.tpp
INTEIRO
DOIS_PONTOS
ID
ABRE_COL
NUM_INTEIRO
FECHA_COL
FLUTUANTE
DOIS_PONTOS
ID
INTEIRO
ID
ABRE_PAR
INTEIRO
DOIS_PONTOS
ID
VIRGULA
FLUTUANTE
DOIS_PONTOS
ID
FECHA_PAR
INTEIRO
DOIS_PONTOS
ID
SE
ABRE_PAR
ID
MAIOR
ID
FECHA_PAR
ENTAO
ID
ATRIBUICAO
ID
ADICAO
ID
SENAO
ID
ATRIBUICAO
ID
MULTIPLICACAO
ID
FIM
RETORNA
ABRE_PAR
ID
FECHA_PAR
FIM
ID
ABRE_PAR
INTEIRO
DOIS_PONTOS
ID
VIRGULA
FLUTUANTE
```

Figura 1. Código sendo executado sem o parâmetro opcional

6. Referências

Moodle. Análise Léxica - Expressões Regulares. 2021. Disponível em: https://moodle.utfpr.edu.br/pluginfile.php/141983/mod_resource/content/7/aula-03-analise-lexica-re.md.slides.pdf. Acesso em: 07 mar.2021.

Regex101. Regular Expressions 101. 2021. Disponível em: <https://regex101.com/>. Acesso em: 07 mar.2021.

```

hmarcuzzo@hmarcuzzo-63-3590:~/Documentos/GitHub/compiler_project/BCC__BCC368__P[1]_HenriqueMarcuzzo_2046334/impl$ python3 lex_submit.py testes_lexico/teste.tpp detailed
LexToken(INTEIRO,'inteiro',1,0)
LexToken(DOIS_PONTOS,':',1,7)
LexToken(ID,'a',1,9)
LexToken(ABRE_COL,'{',1,10)
LexToken(NUM_INTEIRO,'10',1,11)
LexToken(FECHA_COL,'}',4,13)
LexToken(FLUTUANTE,'flutuante',2,15)
LexToken(DOIS_PONTOS,':',2,24)
LexToken(ID,'b',2,26)
LexToken(INTEIRO,'inteiro',4,29)
LexToken(func1,'func1',4,37)
LexToken(ABRE_PAR,'(',4,42)
LexToken(INTEIRO,'inteiro',4,43)
LexToken(DOIS_PONTOS,':',4,50)
LexToken(ID,'x',4,51)
LexToken(VIRGULA,',',4,52)
LexToken(FLUTUANTE,'flutuante',4,54)
LexToken(DOIS_PONTOS,':',4,63)
LexToken(ID,'y',4,64)
LexToken(FECHA_PAR,')',4,65)
LexToken(INTEIRO,'inteiro',5,69)
LexToken(DOIS_PONTOS,':',5,76)
LexToken(ID,'res',5,78)
LexToken(SE,'se',6,84)
LexToken(ABRE_PAR,'(',6,87)
LexToken(ID,'x',6,88)
LexToken(MAIOR,'>',6,90)
LexToken(ID,'y',6,92)
LexToken(FECHA_PAR,')',6,93)
LexToken(ENTAO,'entao',6,95)
LexToken(ID,'res',7,105)
LexToken(ATRIBUICAO,':=',7,109)
LexToken(ID,'x',7,112)
LexToken(ADICAO,'+',7,114)
LexToken(ID,'y',7,116)
LexToken(ENAO,'senao',8,120)
LexToken(ID,'res',9,130)
LexToken(ATRIBUICAO,':=',9,134)
LexToken(ID,'x',9,137)
LexToken(MULTIPLICACAO,'*',9,139)
LexToken(ID,'y',9,141)
LexToken(FIM,'fim',10,145)
LexToken(RETORNA,'retorna',11,151)
LexToken(ABRE_PAR,'(',11,158)
LexToken(ID,'res',11,159)
LexToken(FECHA_PAR,')',11,162)
LexToken(FIM,'fim',12,164)
LexToken(ID,'func2',14,169)
LexToken(ABRE_PAR,'(',14,174)
LexToken(INTEIRO,'inteiro',14,175)
LexToken(DOIS_PONTOS,':',14,182)
LexToken(ID,'z',14,183)
LexToken(VIRGULA,',',14,184)
LexToken(FLUTUANTE,'flutuante',14,186)

```

Figura 2. Código sendo executado com o parâmetro opcional