# Algoritmo de Análise Léxica da Linguagem TPP

**Juliano Cesar Petini**

Universidade Tecnológica Federal do Paraná (UTFPR)

Campo - Mourão– Brazil

[1]julianopetini@alunos.utfpr.edu.br

***Abstract.*** *The following work aims to make a compiler starting with the lexical analysis in an algorithm to check if it is lexically correct,where in its entry is a file of type 'Tpp' and the algorithm will generate related tokens for each predefined type. To use the algorithm it was necessary to use regular expressions (Regex) the python library (Ply).*

***Resumo.*** *O trabalho a seguir tem com o objetivo conseguir fazer um compilador começando com a análise léxica em um algoritmo para verificar se está lexicalmente correto, onde na sua entrada é um arquivo do tipo 'Tpp' e o algoritmo irá gerar tokens relacionados para cada tipo predefinido. Para utilizar o algoritmo foi necessário utilizar expressões regulares (Regex) a biblioteca de python (Ply).*

# 1. Introdução

O Trabalho tem como objetivo de conseguir fazer um compilador começando com a análise léxica em cima de um algoritmo da linguagem *'Tpp'*, após efetuar a análise o algoritmo retorna uma lista de tokens para a entrada de um algoritmo que possa fazer uma análise sintática. para realizar o trabalho foi utilizado a linguagem python com uso do *'Ply'* em conjunto com as expressões regulares *'Regex'*. O trabalho está divido em partes, iremos apresentar cada parte do projeto em sequência.

Na primeira parte iremos apresentar o Python *'Ply'* iremos dar uma ideia de como funciona e como está sendo utilizado no algoritmo, logo em seguida será explicado o'que é expressões regulares *'Regex'* e também como está sendo utilizado em nosso projeto. Já a seguir iremos falar sobre o'que se trata a análise léxica qual a sua ideia e como ela funciona dando alguns exemplos e mostrando como nosso algoritmo tenta reproduzir essa ideia. E no próximo capítulo iremos falar sobre o *'Tpp'* a linguagem que foi implementada, eo compilador para reconhecimento, juntamente com a implementação do algoritmo, como funciona os tokens, como ele identifica cada um dos token, e também o'que ele retorna após executar o programa e por fim algumas conclusões falando sobre o desempenho do algoritmo com base em vários testes executados.

O Projeto teve início com a disciplina de compiladores da Universidade Tecnológica Federal do Paraná - Campo Mourão onde foi feito a primeira parte do projeto que consiste em fazer o desenvolvimento de um compilador com somente a primeira parte implementada a análise léxica de um algoritmo.

# 2. Python Ply

O ‘*Ply’* e uma implementação de ferramentas para análise léxica e sintática no python onde basicamente ele faz um parsing com ajuda do *Regex* para identificar tokens, que são palavras ou símbolos que são reservadas na linguagem. A versão original da biblioteca foi desenvolvida em 2001 por David Beazley para uso em um curso de Introdução aos Compiladores, em que os alunos a usavam para construir um compilador para uma linguagem simples do tipo Pascal. Devido ao seu uso em um ambiente instrucional, muito trabalho foi feito para fornecer uma extensa verificação de erros. Além disso, essa experiência foi usada para resolver problemas comuns de usabilidade. Desde então, uma variedade de melhorias incrementais foram feitas na biblioteca.

# 3. Expressões regulares

Expressões regulares ‘*Regex’* e um notação para poder identificar um padrão em uma string, serve para validar entradas de dados ou fazer busca e extração de informações em textos. Expressões regulares são escritas numa linguagem formal que pode ser interpretada por um processador de expressão regular, um programa que serve como um examinador de texto e identifica as partes que combine com a especificação dada.

No algoritmo criado são utilizadas várias expressões regulares para identificar padrões, iremos apresentar algumas delas como autômatos, na Figura-1 que irá mostrar um autômato que reconhece a palavra *‘inteiro’,* na Figura-2 apresenta um autômato que reconhece a palavra *‘repita’*, e por fim na Figura-3 iremos apresentar um autômato que reconhece a palavra *‘leia’*, esse são apenas alguns de vários outros que foram implementados no projeto.

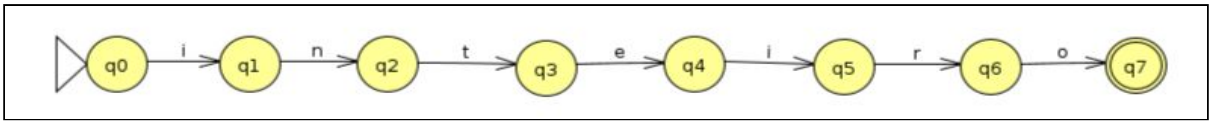


Figura-1 Apresenta o autômato que identifica o token *inteiro.*

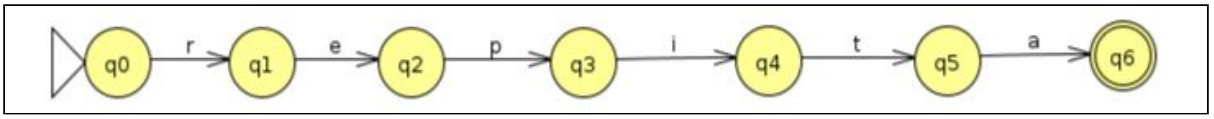


Figura-2 Apresenta o autômato que identifica o token *repita.*

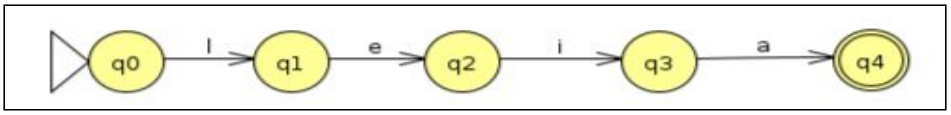


Figura-3 Apresenta o autômato que identifica o token de *leia.*

# 4. Análise Léxica

Léxico formalmente é o conjunto de palavras existente em um determinado linguagem que as pessoas têm à disposição para expressar-se, oralmente ou por escrito em seu contexto, são basicamente palavras predefinidas de uma linguagem a serem seguidas, no projeto criado a nossa linguagem léxica tem várias palavras e símbolos reservados algumas já foram apresentada aqui como a palavra *inteiro*,*leia* ou a *repita*, porém existem várias outras, cada uma dessas palavra é capturadas pelo nosso analisador léxico e caso esteja escrita correto ela vira um token de saída.

# 5. Algoritmo

Para apresentar o algoritmo iremos separar em duas parte, a primeira iremos falar um pouco da ideia do tpp e como surgiu a suas especificações, já na segunda parte iremos falar da implementação do algoritmo e como foi feita cada parte e o'que cada parte é responsável em fazer.

## 5.1. TPP

A ideia do '*Tpp'* e ser uma linguagem de fácil entendimento permitindo criar algoritmos sem dificuldades, com semelhanças de linguagem de alto nível e também ter aspectos bem intuitivos, a descrição da linguagem tem origem na disciplina de compiladores com intuito de criarmos uma nova linguagem para nosso compilador.

## 5.1. IMPLEMENTAÇÃO

A implementação do algoritmo  foi feito utilizando a linguagem python com a biblioteca '*Ply'* e o '*Regex',* para começar o algoritmo precisamos primeiro definir uma lista de tokens que serão nossas palavras e símbolos reservados, na Figura-4 iremos apresentar como está feito a lista de tokens no compilador.

```
def Toke():
    tokens = (
    'MAIS','MENOS','MULTIPLICACAO','DIVISAO','DOIS_PONTOS','VIRGULA','MENOR','MAIOR',
    'IGUAL','DIFERENTE','MENOR_IGUAL','MAIOR_IGUAL','E_LOGICO','OU_LOGICO','NEGACAO','ABRE_PARENTESE',
    'FECHA_PARENTESE','ABRE_COLCHETE','FECHA_COLCHETE','SE','ENTAO','SENAO','FIM','REPITA','ATE',
    'ATRIBUICAO','LEIA','ESCREVA','RETORNA','INTEIRO','FLUTUANTE','NUM_INTEIRO','NUM_PONTO_FLUTUANTE',
    'NUM_NOTACAO_CIENTIFICA','ID','COMENTARIO',
    )

    return tokens
```

Figura-4 Apresenta a lista de tokens definido na linguagem *Tpp.*

A segunda parte da implementação e a criação de expressões regulares ‘*Regex’* para ler linha a linha fazendo parsing e capturar os tokens, assim gerando uma lista de tokens no final, para cada token é preciso criar uma expressão específica que basicamente funciona como um autômato. Na Figura-5 iremos apresentar algumas expressões regulares simples para capturar palavras reservadas e gerar os tokens. Já na Figura-6 apresenta algumas das expressões regulares complexas para identificar palavras reservadas.

```
t_FIM = r'fim'
t_SENAO = r'senão'
t_SE = r'(se)'
t_ATE = r'até'
t_LEIA = r'leia'
t_MAIS = r'\+'
t_MENOS = r'-'
```

Figura-5 Apresenta expressões regulares simples para gerar tokens de palavras reservadas.

```
def t_NUM_PONTO_FLUTUANTE(t):
    r'(-)?\d+\.\d*'
    t.value = float(t.value)
    return t

def t_NUM_INTEIRO(t):

    r'((?<=\D)[+-]\d+)|(^(?<=\D)[+-]\d+)|\d+'

    t.value = int(t.value)
    return t
```

Figura-6 APresenta expressões regulares complexas para gerar tokens de palavras reservadas.

A entrada do programa é um arquivo do tipo tpp, que contenha um código, para então a partir desse código o programa começar a fazer a análise e retirar o tokens. Na Figura-7 iremos ilustrar um código simples na linguagem, logo em seguida na Figura-8 iremos mostrar a lista de tokens gerada pelo programa.

```
inteiro: n
flutuante: a[10]

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))
    retorna(0)
fim
```

Figura-7 Ilustra um código Fatorial em *Tpp.*


Figura-8 Ilustra uma parte da lista de tokens gerada.

# 6. Conclusão

Como conclusão, podemos afirma que o analisador léxico criado e funcional por fato de funciona como deveria em diversos teste e algoritmos diferentes, em todos os casos funcionou perfeitamente sem apresentar qualquer tipo de falha ou classificar um token incorretamente, então a partir disso afirmamos que foi concluído o objetivo do projeto que era a princípio criar uma parte de um compilador, a análise léxica foi feita com êxito e apresenta ótimos resultados.

# 7. References

Wikipedia Léxico Disponível em: <https://pt.wikipedia.org/wiki/L%C3%A9xico>.Acesso em: 20 mar. 2020.
Dabeaz PythonPly Disponível em: <https://www.dabeaz.com/ply/>.Acesso em: 20 mar. 2020.
Wikipedia Regex Disponível em: <https://en.wikipedia.org/wiki/Regular_expression>.Acesso em: 20 mar. 2020.
JFLAP Autômato Disponível em: <http://www.jflap.org/>.Acesso em: 20 mar. 2020.