

Instructions for Authors of SBC Conferences Papers and Abstracts

Felipe Archanjo da Cunha Mendes¹

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão– PR – Brazil

felipecunhamendes@gmail.com

Abstract. *This article deals with the resolution of project01 of the Formal Languages, Automata and Computability subject, taught by Professor Rogério Gonçalves in relation to the creation of a lexical analysis with a Moore machine that recognizes c- language source code.*

Resumo. *Este artigo se trata da resolução do projeto01 da disciplina de Linguagens Formais, Automatos e Computabilidade, ministrada pelo professor Rogério Gonçalves em relação a criação de uma análise lexica com maquina de moore que reconhece um codigo fonte de linguagem c-.*

1. Informações Gerais

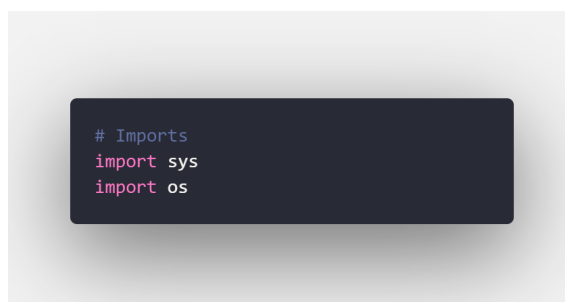
As ferramentas para a criação do algoritmo de analise lexica usando uma maquina de moore são:

- Python 3.9.2
- Pycharm 2021.1.2 (Professional Edition)

2. Desenvolvimento

2.1. Imports

Para o desenvolvimento deste projeto, utilizamos apenas as bibliotecas padrões do python. Importamos a biblioteca sys para verificar a entrada de parametros no terminal e para encerrar o programa caso ocorra algum erro de manipulação de arquivos. Não só isso, mas importamos tambem a biblioteca os para verificar a existencia de arquivos no diretorio.

A screenshot of a code editor with a dark background. It shows three lines of Python code: a comment line '# Imports' followed by two import statements, 'import sys' and 'import os', each on a new line. The text is in a light color, likely white or light blue, for contrast against the dark background.

```
# Imports
import sys
import os
```

Figure 1. Imports

2.2. Variaveis

Algumas variaveis serão necessarias para o desenvolvimento desse projeto.

1. Buffer: Uma lista que utilizaremos para analisar e encontrar os tokens ID no código fonte.
2. tokens_operadores: Um dicionario onde as chaves são os simbolos de operação e os valores são os seus respectivos tokens.
3. tokens_delimitadores: Um dicionario onde as chaves são os simbolos dos delimitadores e os valores são seus respectivos tokens.
4. tokens_palavras_reservadas: Um dicionario onde as chaves são palavras reservadas e os valores são seus respectivos tokens.
5. abc: Uma variavel que contenha as letras do abcdario latino.
6. numeros: uma variavel que contenha os numeros de 0 a 9.

```
# Variaveis
buffer = []

tokens_operadores = {
    '+': 'PLUS',
    '-': 'MINUS',
    '*': 'TIMES',
    '/': 'DIVIDE',
    '=': 'ATTRIBUTION',
    '==': 'EQUALS',
    '!=': 'DIFFERENT',
    '>': 'GREATER',
    '<': 'LESS',
    '>=': 'GREATER_EQUAL',
    '<=': 'LESS_EQUAL',
}

tokens_delimitadores = {
    '(': 'LPAREN',
    ')': 'RPAREN',
    '[': 'LBRACK',
    ']': 'RBRACK',
    '{': 'LBRACE',
    '}': 'RBRACE',
    ';': 'SEMICOLON',
    ',': 'COMMA',
}

tokens_palavras_reservadas = {
    'if': 'IF',
    'else': 'ELSE',
    'while': 'WHILE',
    'int': 'INT',
    'return': 'RETURN',
    'void': 'VOID',
}

abc = "abcdefghijklmnopqrstuvwxyz"
numeros = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Figure 2. Variaveis

2.3. Função Principal

Em primeiro lugar é necessário receber o nome do arquivo de entrada onde se encontra o código em c-. Para tal, o nome do arquivo pode ser recebido tanto por argv ou manualmente digitado quando a função input do programa for executada. Em seguida, foi necessário utilizar o comando `os.path.isfile(filename)` para verificar se o arquivo dado pelo usuário realmente existe. Por fim, o arquivo é aberto para leitura.

```
# Funções
def main():
    # Definindo o arquivo de entrada
    if len(sys.argv) > 1:
        f_in = sys.argv[1]
    else:
        print("Digite o nome do arquivo de entrada:")
        f_in = input()

    # Verificando se o arquivo existe
    if not os.path.isfile(f_in):
        print("Arquivo não encontrado!")
        sys.exit()

    # Abrindo o arquivo de entrada
    try:
        file = open(f_in, "r")
    except:
        print("Erro ao abrir o arquivo de entrada!")
        sys.exit()
```

Figure 3. Manipulação do arquivo de entrada

Da mesma forma que abrimos o arquivo de entrada, devemos agora criar um arquivo de saída, mas com o parâmetro de escrita.

```
# Definindo o arquivo de saída
if len(sys.argv) > 2:
    f_out = sys.argv[2]
else:
    f_out = "saida.out"

# Abrindo o arquivo de saída
try:
    outfile = open(f_out, "w")
except:
    print("Erro ao abrir o arquivo de saída!")
    sys.exit()
```

Figure 4. Manipulação do arquivo de saída

A partir de agora devemos ler o arquivo de entrada e analisar caractere por caractere do código em c-. Para isso usamos dois laços for: o primeiro para iterar nas linhas e o segundo para iterar os caracteres de cada uma dessas linhas.

Uma das primeiras verificações que deve ser feita é a existência de espaços em branco. Caso isso ocorra nós os ignoramos e avançamos para a próxima iteração.

Uma segunda verificação que fazemos é a existência de valores numéricos. Para

```

# Lendo o arquivo de entrada
for line in file:
    for i in range(len(line)):

        # Verificando se é espaço em branco
        if line[i] == ' ' or line[i] == '\n' or line[i] == '\t':
            continue

```

Figure 5. Verificação de espaços e numeros

isso, nós analisamos o caractere e verificamos se ele está presente no vetor de numeros. Em caso afirmativo, o token NUM é escrito no arquivo de saída.

Ademais, precisamos verificar se o determinado caractere analisado é um operador ou delimitador. Para isso criamos dois ifs parecidos no qual buscamos o caractere nos dicionários de tokens_operadores e tokens_delimitadores e escrevemos o token representativo para cada símbolo no arquivo.

Uma análise considerável que devemos fazer é em relação à detecção de ID's. O vetor buffer, inicialmente vazio, foi criado para ser utilizado nesta etapa. Cada vez que for lido um caractere alfanumérico, esse caractere é adicionado dentro do buffer e, assim, é lido o próximo caractere na próxima iteração. Quando for verificado que no buffer existe uma palavra reservada, é buscado o token dela no dicionário tokens_palavras_reservadas, escrevemos ele no arquivo de saída e limpamos o buffer. Por outro lado, se caracteres alfanuméricos forem sendo adicionados ao buffer e, em algum momento, for inserido um símbolo qualquer (delimitadores, operadores ou numeros), podemos afirmar que os caracteres no buffer, de forma concatenada, formam um ID.

```

# Verificando se o caractere é um numero
if line[i] in numeros:
    if buffer:
        buffer.append(line[i])
    else:
        outfile.write("NUMBER\n")

# Verificando se o caractere é um operador
elif line[i] in tokens_operadores:
    if buffer:
        outfile.write("ID\n")
        buffer.clear()

    outfile.write(tokens_operadores[line[i]] + "\n")

# Verificando se o caractere é um delimitador
elif line[i] in tokens_delimitadores:
    if buffer:
        outfile.write("ID\n")
        buffer.clear()

    outfile.write(tokens_delimitadores[line[i]] + "\n")

# Verificando se é um identificador ou palavra reservada
elif line[i] in abc:
    buffer.append(line[i])

    if "".join(buffer) in tokens_palavras_reservadas:
        outfile.write(tokens_palavras_reservadas["".join(buffer)] + "\n")
        buffer.clear()

```

Figure 6. Verificação operadores, delimitadores, IDs e palavras reservadas

Por fim, chamamos a função main

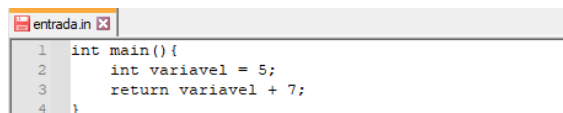


```
# Função principal
if __name__ == "__main__":
    main()
    print("Análise finalizada!")
```

Figure 7. Função main

3. Executando o programa

Para executar esse código, é necessário criar um arquivo com qualquer nome e inserir um código C- em seu conteúdo. Vamos tomar como exemplo o arquivo com o seguinte conteúdo:

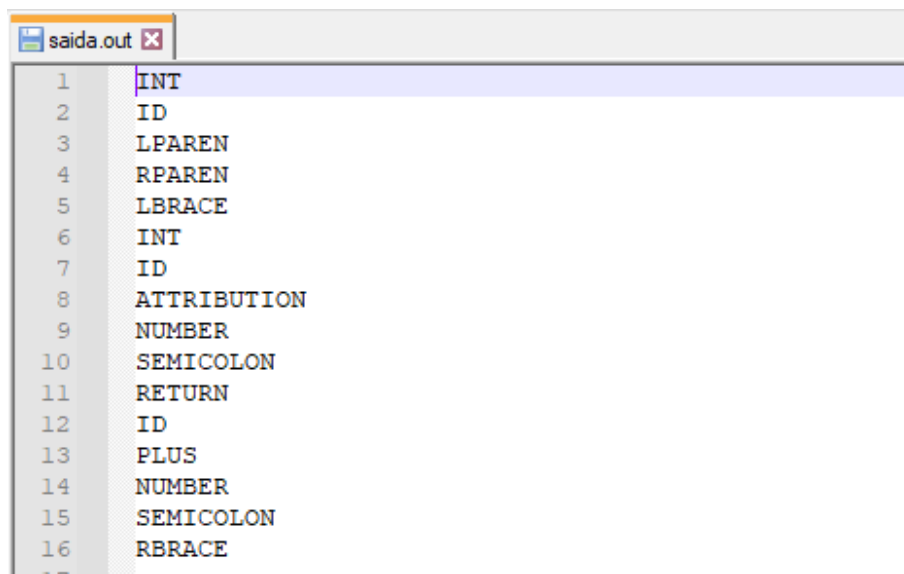


```
1 int main(){
2     int variavel = 5;
3     return variavel + 7;
4 }
```

Figure 8. arquivo de entrada

Para executar o programa basta abrir o terminal no diretório do projeto e digitar *python automato.py*. Aparecerá uma mensagem no terminal pedindo o nome do arquivo de entrada, então deve ser digitado o nome do arquivo que foi criado anteriormente.

Ao término da execução, um arquivo *arquivo.out* será gerado no mesmo diretório, com o seguinte conteúdo:



```
saida.out
1 INT
2 ID
3 LPAREN
4 RPAREN
5 LBRACE
6 INT
7 ID
8 ASSIGNMENT
9 NUMBER
10 SEMICOLON
11 RETURN
12 ID
13 PLUS
14 NUMBER
15 SEMICOLON
16 RBRACE
```

Figure 9. arquivo.out

Portanto, percebe-se que o algoritmo funcionou corretamente para o trecho de código analisado no arquivo de entrada.