PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
ESCOLA POLITÉCNICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

PEDRO HENRIQUE CAVALHIERI CONTESSOTO
JEFFERSON SOBRINHO ABBIN

PROJETO COMPILADOR (FASE 1)

CURITIBA
2023

PEDRO HENRIQUE CAVALHIERI CONTESSOTO

JEFFERSON SOBRINHO ABBIN

PROJETO COMPILADOR (FASE 1)

Atividade avaliativa de Curso de Linguagens Formais e Compiladores da Pontifícia Universidade Católica do Paraná

Orientador: Me. Eng. Frank Coelho de Alcantara

CURITIBA

2023

1 Fase 1 – Definição da Linguagem E Análise Léxica

Para definição da linguagem de programação foi definida a seguinte bloco de declarações pela lista abaixo, com declarações terminais estão em **negrito**:

program → block
block → {decls stmts}
decls → decls decl | **e**
decl → type id
type → type [**num**] | **basic**
stmts → stmts stmt | **e**
bool → bool || join | join
join → join && equality | equality
equality → equality == rel | equality != rel | rel
rel → expr < expr | expr <= expr | expr >= expr | expr > expr | expr
expr → expr + term | expr - term | term
term → term * unary | term / unary | unary
unary → !unary | unary | factor
factor → (bool) | num | real | **true** | **false |** string
stmt → **if (**bool**)** stmt
stmt → **if (**bool**)** stmt **else** stmt
stmt → **while (** bool **)** stmt
stmt → **do** stmt **while (**bool**)**
stmt → **break**
stmt → **WritePort (**string**, true** | **false)**
stmt → **ReadPort (**string**,** string**)**
stmt → **ReadSerial (**string**,** expr**)**
stmt → **WriteSerial (**string**,** string **)**
stmt → **Writeanalog (**string**,** expr **)**
stmt → **Readanalog (**string, string**)**
stmt → **sleep (**expr**)**
num → {**0 - 65.536**}
char → {**0 - 9**} || {**a - z**} || **{A - Z**} || {**!@#$%¨&*()_+=**}

Para a seguinte linguagem segue 3 (Três) exemplos código com suas funcionalidades de possíveis:

EXEMPLOS

1) IF (X > 12){
   WRITESERIAL(S3, 2301)
   }

2) while (x < 12){
   buffer = !buffer
   WRITEPORT(D1, BUFFER)
   SLEEP (10)
   X = X +1
   }

3) X = READANALOG (A2, BUFFER)
   IF (BUFFER > 3000){
   WRITESERIAL(S3, "PRESSÃO ELEVADA")
   }

Para a etapa da análise Léxica foi desenvolvido o código para realizar a mesmo, conforme solicitado o codigo é capaz de validar todos os possíveis lexemas da linguagem, usando máquinas de estado finito, implementado em Python.

Temos a primeira etapa do codigo onde inicializado os estados inicial

```
def __init__(self):
    self.state = 'START'
    self.lexeme = ''
    self.tokens = []
    self.position = 0
```

Abaixo esta definição dos estados de transição:

```
def transition(self, char):
        if self.state == 'START':
            if char in [' ', '\n', '\t']:  # Skip whitespace
                self.position += 1
                return
            elif char.isalpha() or char == '_':
                self.state = 'IDENTIFIER'
            elif char.isdigit():
                self.state = 'NUMBER'
            elif char in ['=', '!', '<', '>', '+', '-', '*', '/']:
                self.state = 'OPERATOR'
            elif char in ['(', ')', '{', '}', '[', ']', ',', ';']:
                self.state = 'MARKERS'
            else:
                print(f"Invalid token: {char}")
                return

            self.lexeme += char
        elif self.state == 'IDENTIFIER':
            if char.isalpha() or char.isdigit() or char == '_':
                self.lexeme += char
             elif self.lexeme in ["if", "while", "break", "do", "sleep", "WritePort", "ReadPort",
"ReadSerial", "WriteSerial", "Writeanalog", "Readanalog"]:
                self.emit('KEYWORD')
                self.transition(char)
            else:
                self.emit('IDENTIFIER')
                self.transition(char)  # reprocess the current character
        elif self.state == 'NUMBER':
            if char.isdigit():
                self.lexeme += char
            elif char == '.':
                self.state = 'REAL'
```

```
                self.lexeme += char
            else:
                self.emit('NUMBER')
                self.transition(char)  # reprocess the current character
        elif self.state == 'REAL':
            if char.isdigit():
                self.lexeme += char
            else:
                self.emit('REAL')
                self.transition(char)  # reprocess the current character
        elif self.state == 'OPERATOR':
            self.lexeme += char
            if not self.lexeme in ['==', '!=', '<=', '>=']:
                self.lexeme = self.lexeme[:-1]
                self.emit('OPERATOR')
                self.transition(char)  # reprocess the current character

        elif self.state == 'MARKERS':
            self.lexeme += char
            self.lexeme = self.lexeme[:-1]
            self.emit('MARKERS')
            self.transition(char)  # reprocess the current character
```

Abaixo as funções de auxiliares para definição dos tokens reconhecidos:

```
 emit(self, token_type):
        self.tokens.append((token_type, self.lexeme, self.position))
        self.position += len(self.lexeme)
        self.lexeme = ''
        self.state = 'START'

    def tokenize(self, code):
        for char in code:
            self.transition(char)
        if self.lexeme:  # handle the last token
```

```
            self.emit(self.state)
        return self.tokensxiliares
def emit(self, token_type):
        self.tokens.append((token_type, self.lexeme, self.position))
        self.position += len(self.lexeme)
        self.lexeme = ''
        self.state = 'START'

    def tokenize(self, code):
        for char in code:
            self.transition(char)
        if self.lexeme:  # handle the last token
            self.emit(self.state)
        return self.tokens
```

Abaixo as funções para ler o arquivo e cria as tabela de tokens:

```
def read_code_from_file(file_name):
    with open(file_name, 'r') as file:
        code = file.read()
    return code

def create_token_table(tokens, file_name):
    with open(file_name, 'w') as file:
        for token_type, token, position in tokens:
            file.write(f"Token: {token}, Classe: {token_type}, Posição: {position}\n")
```

E por último a função principal para rodar o código:

```
if __name__ == "__main__":
      if sys.argv[1] != "code1.txt" and sys.argv[1] != "code2.txt" and sys.argv[1] !=
"code3.txt":
     print("Arquivo nao existe, escolha: code1.txt, code2.txt ou code3.txt")
     exit(1)
```

```
code = read_code_from_file(sys.argv[1])
analyzer = LexicalAnalyzerFSM()
tokens = analyzer.tokenize(code)
create_token_table(tokens, "token_table.txt")
```

Como Solicitado analisador léxico deve recebe o arquivo de textos contendo codigo, na linha de comando conforme Figura 01

```
~/Analisador-Lexico$ python main.py code1.txt
~/Analisador-Lexico$
```

Figura 1 - Print da linha de comando.

Fonte: autores, 2023.

e retornando o token identificado a sua classe e sua posição no arquivo exemplificado na figura 02

token_table.txt

```
1  Token: if, Classe: KEYWORD, Posição: 0
2  Token: (, Classe: MARKERS, Posição: 3
3  Token: X, Classe: IDENTIFIER, Posição: 4
4  Token: >, Classe: OPERATOR, Posição: 6
5  Token: 12, Classe: NUMBER, Posição: 8
6  Token: ), Classe: MARKERS, Posição: 10
7  Token: {, Classe: MARKERS, Posição: 11
8  Token: WriteSerial, Classe: KEYWORD, Posição: 14
9  Token: (, Classe: MARKERS, Posição: 25
10 Token: s3, Classe: IDENTIFIER, Posição: 26
11 Token: ,, Classe: MARKERS, Posição: 28
12 Token: 2301, Classe: NUMBER, Posição: 30
13 Token: ), Classe: MARKERS, Posição: 34
14 Token: }, Classe: MARKERS, Posição: 37
15
```

Figura 1 - Arquivo de saída Analisador Léxico.

Fonte: autores, 2023.

Link para repositorio do GitHub: https://github.com/JeffAbbin/Projeto_Compilador-

Link para rodar o código no replt:

https://replit.com/@JeffersonAbbin/Analisador-Lexico#main.py

Link2 para rodar o código no replt: https://replit.com/join/mjgkqebsex-jeffersonabbin

REFERÊNCIAS

LINGUAGEM de programação e Compiladores. [*S. l.*], 1 jan. 2021. Disponível em: https://frankalcantara.com/Ling-programacao-compiladores/. Acesso em: 16 abr. 2023.

REGEX refence. [*S. l.*], 16 abr. 2023. Disponível em: https://regexr.com/. Acesso em: 16 abr. 2023.

WHAT is ANTLR?. [S. l.], 25 maio 2023. Disponível em: https://www.antlr.org. Acesso em: 25 maio 2023.

ANALISE Léxica. [S. l.], 25 maio 2023. Disponível em: https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/lexical-analysis.html. Acesso em: 25 maio 2023.