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

ISA STOHLER BERTOLACCINI
JEFFERSON SOBRINHO ABBIN

PROJETO COMPILADOR (FASE 2)

CURITIBA
2023

ISA STOHLER BERTOLACCINI
JEFFERSON SOBRINHO ABBIN

PROJETO COMPILADOR (FASE 2)

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

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**} || {**!@#$%¨&*()_+=**}
real → **PonF** (sinal, mantissa,expoente)

sinal → {0,1}
expoente → {0,1}[8]
mantissa → {0,1}[8]
string → char*

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")
}
```

2 Fase 2 – Verificação de Código

Nessa etapa do projeto construímos um analisador lexico, para isso foi utilizado a ferramenta ANTLR4 para a geração do analisador léxico e sintático em Python. Ele permite definir a gramática de uma linguagem usando uma notação formal e, em seguida, gera automaticamente o código do analisador correspondente.

O primeiro passo é definir a gramática da linguagem que você deseja analisar. A gramática é escrita em um arquivo com a extensão .g4. A gramática define as regras de produção, tokens e estrutura da linguagem, se abaixo a definição da linguagem:

```
grammar PythonCode;


// Tokens não terminais
program : statement+ ;
statement : assignmentStatement
        | ifStatement
        | whileStatement
        | functionDeclaration
        | expressionStatement
        | returnStatement
        | printStatement
        | WritePortStatement
        | ReadPortStatement
        | WriteSerialStatement
        | sleepStatement
        | ReadSerialStatement
        | ReadanalogStatement;
```

assignmentStatement : variable '=' expression ;

ifStatement : 'if' '(' equality ')' '{' statement+ '}' ('else' '{' statement+ '}')? ;

whileStatement : 'while' '(' equality ')' '{' statement+ '}' ;

functionDeclaration : 'def' functionName '(' parameterList? ')' '{' statement+ '}' ;

expressionStatement : expression ';' ;

returnStatement : 'return' expression? ';' ;

printStatement : 'print' expressionList ';' ;

WritePortStatement : 'WritePort(' IDENTIFIER ',' BOOL ')\n';

ReadPortStatement : 'ReadPort(' IDENTIFIER ',' IDENTIFIER ')\n';

WriteSerialStatement : 'WriteSerial(' IDENTIFIER ',' IDENTIFIER ')\n';

ReadSerialStatement : 'ReadSerial(' IDENTIFIER ',' INT ')\n';

ReadanalogStatement : 'Readanalog(' IDENTIFIER ',' INT ')\n';

sleepStatement : 'sleep(' expression ')\n';

expressionList : expression (',' expression)* ;

parameterList : parameter (',' parameter)* ;

parameter : variable ;

equality : equality '==' rel | equality '!=' rel | rel | BOOL ;

rel : expression '<' expression | expression '<=' expression | expression '>=' expression | expression '>' expression ;

expression : primaryExpression (binaryOperator primaryExpression)* ;

primaryExpression : variable

| functionCall

| '(' expression ')' ;

functionCall : functionName '(' expressionList? ')' ;

variable : IDENTIFIER ;

functionName : IDENTIFIER ;

binaryOperator : '+' | '-' | '*' | '/' ;

```
IDENTIFIER : [a-zA-Z_][a-zA-Z0-9_]* ;

BOOL : 'true' | 'false' ;

INT : [0-9]+ ;

FLOAT : [0-9]+ '.' [0-9]* ;

WS : [ \t\n\r]+ -> skip ;
```

O ANTLR4 gera automaticamente o código do analisador léxico e sintático com base na gramática especificada. Ele gera várias classes em Python, incluindo um analisador léxico, um analisador sintático e um visitante ou ouvinte (visitor/listener), dependendo da forma de processamento escolhida.

Os códigos gerados estão na repositório do github.

A Partir disso toda nossa estrutura estava estava feita e rodamos o código em main.py fazer toda a análise lexica e os parses, assim demonstrando os possíveis erros e mostrar a árvore formada, segue o código abaixo.

```
from antlr4 import *

from PythonCodeLexer import PythonCodeLexer

from PythonCodeParser import PythonCodeParser

arquivo = open('codigo1.txt', 'r')

codigo = arquivo.read()

# Criar um input stream com a expressão a ser analisada

input_stream = InputStream(codigo)

# Criar um lexer com o input stream

lexer = PythonCodeLexer(input_stream)

# Criar um token stream com o lexer

token_stream = CommonTokenStream(lexer)

# Criar um parser com o token stream

parser = PythonCodeParser(token_stream)
```

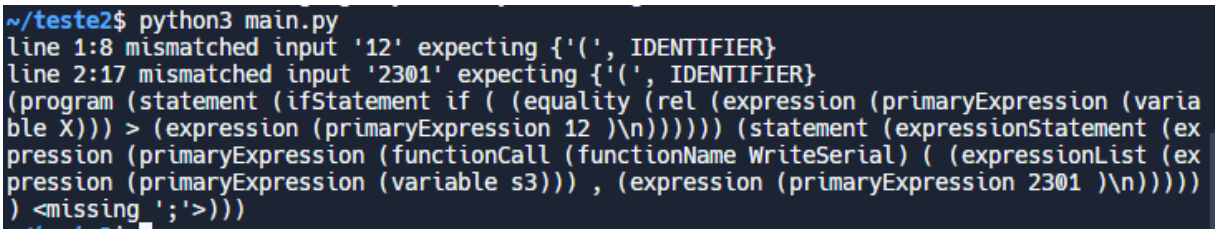
```
# Chamar a regra sintática inicial

tree = parser.program()

# Realizar alguma ação com a árvore sintática resultante

print(tree.toStringTree(recog=parser))
```

resultado do exemplo 1 rodado



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

Link para rodar o código no replt:

https://replit.com/@JeffersonAbbin/Verificacao-de-Codigo-v2#

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.