CafezInScript

Integrantes: André Ulhôa Werneck, Eduardo Cezar Carvalho, Mateus Augusto, Pedro Carvalho

Link do repositório no github: https://github.com/Eduardo-Cezar/Trabalho-Compiladores

Definições da linguagem

Comentários:
Comentários de linha iniciam com "//" e vão até o final da linha.
Exemplo: // Isso é um comentário de linha.

Parâmetros:
(valores, variáveis)
Delimitador:
;
Operadores aritméticos:
Soma: var + var
Subtração: var - var
Multiplicação var * var
Divisão: var / var

Atribuição:
var = (var, valor, operação aritmética)

Operadores relacionais:
Maior que: (var, valor, operação aritmética) > (var, valor, operação aritmética)

Maior ou igual: (var, valor, operação aritmética) >= (var, valor, operação aritmética)

Menor que: (var, valor, operação aritmética) < (var, valor, operação aritmética)

Menor ou igual: (var, valor, operação aritmética) <= (var, valor, operação aritmética)

Diferente: (var, valor, operação aritmética) != (var, valor, operação aritmética)

Igual: (var, valor, operação aritmética) == (var, valor, operação aritmética)

Palavras-chave: palavras reservadas da linguagem que possuem significado especial.
Exemplos: func, while, if, then, else, return, in, out, int, float, boolean, str, var, vars, call

Imprimir valor no console: out((var, valor, operação aritmética))

Tipos:

int
float
boolean
str

Funções:

```
func tipo nome da função(Parâmetros){
        …
        return (var, valor, operação aritmética)
}
```

Se, então, senão:

```
if (Operação relacional) then {
        …
}

ou

if (Operação relacional) then {
        …
}else{
        …
}
```

Estrutura de repetição:

```
while (Operação relacional){
        …
}
```

Declaração de variáveis:

```
vars{
        tipo nome;
}
```

Chamada de função:

call nome da função(Parâmetros)

Retorno de função:

return (call, variável, valor)

Padrão	**Tipo de lexema**	**Sigla**
Os próprios lexemas.	Operadores aritméticos: +, -, *, /	OpArit
Os próprios lexemas.	Operadores relacionais: >, >=, <, <=, !=, ==	OpRel
Os próprios lexemas	Atribuição: =	atr
Os próprios lexemas.	Delimitador: ;	Del
Os próprios lexemas.	Parênteses: (,)	AP / FP
Os próprios lexemas	Chaves: {, }	AC / FC
Os próprios lexemas	Aspas: ‘ “ ’	asp
Sequência de letras e números que começam com letras.	Variável	id
Sequência dos caracteres ‘w’,’h’,’i’,’l’,’e’, respeitando a ordem apresentada.	Estrutura de repetição: while	whi
Sequência dos caracteres ‘i’,’f’ respeitando a ordem apresentada.	Estrutura condicional: if	if
Sequência dos caracteres ‘t’,’h’, ‘e’, ‘n’ respeitando a ordem apresentada.	Parte da estrutura condicional	then
Sequência dos caracteres ‘e’,’l’,’s’, ‘e’ respeitando a ordem apresentada.	Parte da estrutura condiciona	else
Sequência dos caracteres ‘c’,’a’,’l’, ‘l’ respeitando a ordem apresentada.	Chamada de função	call
Os próprios lexemas.	Tipos: int, float, boolean, str	tipo
Sequência dos caracteres ‘r’,’e’,’t’, ‘u’, ‘r’, ‘n’ respeitando a ordem apresentada.	Retorno de função	ret

Dígitos de 0-9 com ou sem ponto flutuante.	Constantes numéricas	num
Conjunto de caracteres dentro de “ ,”.	Constante de caracteres	cstr
Sequência dos caracteres ‘v’,’a’,’r’, ‘s’ respeitando a ordem apresentada.	Bloco de declaração de variáveis	var
Sequência dos caracteres ‘f’,’u’,’n’, ‘c’ respeitando a ordem apresentada.	Funções	fun

Exemplos de programa:

Exemplo 1 (Fatorial):

```
    //função fatorial
func int fat(int n){
            if (n == 1) then {
        return 1;
            }else{
                    return n * call fat (n-1);
            }
    }
//função principal
    func int main(){
            vars{
                    int n;
            }

            in(n);
            out(call fat(n));

            return 0;
    }
```

Exemplo 2 (Média aritmética de dois valores):

```
func int main(){
        vars{
                int valor1;
                int valor2;
        }

        in(valor1);
        in(valor2);

        resultado = (valor1 + valor2)  /  2;

        out(valor);

        return 0;
}
```

Exemplos de programas que não compilam devido a erros léxicos:

Exemplo 1(fatorial):

```
        //função fatorial
    func int fat(int n){
                if (n == 1) then {
            return 1;
                }else{
                        return n * call fat (n-1);
                }
        }
    //função principal
        func int main(){
                int n;  //aqui há um erro na declaração da variável, que deveria ser
declarada dentro de um bloco de comando “var{}”

                in(n);
                out(fat(n)); //aqui há um erro na forma de chamar a função, devido a falta
do comando “call”

                return 0;
        }
```

Exemplo 2(Média aritmética de dois valores):

```
func int main(){
   vars{
```

```
    int valor1;
        int valor2;
  }

  cin (valor1);
  cin(valor2);//Aqui há um erro nas entradas dos valores das variáveis, a qual deve ser feita pelo operador “in” e não “cin”

  resultado = (valor1 + valor2)  /  2;

  out(valor);

  return 0;
}
```

Prints, Arquivos Gerados pelo ANTLR

Arquivo: *GramaticaLexer.java*

O código é a implementação do lexer para a gramática "Gramatica" e fornece todas as informações necessárias para analisar e reconhecer os tokens da linguagem definida pela gramática.

```
29 @    private static String[] makeRuleNames() {
30          return new String[] {
31              "AP", "FP", "AC", "FC", "DEL", "VAR", "ID", "NUM", "TIPO", "FUN", "CALL",
32              "RET", "OPARIT", "OPREL", "WHI", "IF", "THEN", "ELSE", "WS", "LETRA",
33              "DIGITO"
34          };
35      }
```

Esse método retorna um array de strings contendo os nomes das regras definidas na gramática.

```
38 @    private static String[] makeLiteralNames() {
39          return new String[] {
40              null, "'('", "')'", "'{'", "'}'", "';'", "'vars'", null, null, null,
41              "'func'", "'call'", "'return'", null, null, "'while'", "'if'", "'then'",
42              "'else'"
43          };
44      }
```

Esse método retorna um array de strings contendo os literais definidos na gramática.

```
46 @    private static String[] makeSymbolicNames() {
47          return new String[] {
48              null, "AP", "FP", "AC", "FC", "DEL", "VAR", "ID", "NUM", "TIPO", "FUN",
49              "CALL", "RET", "OPARIT", "OPREL", "WHI", "IF", "THEN", "ELSE", "WS"
50          };
51      }
```

O método retorna os nomes simbólicos definidos na gramática.

```
157     public static final ATN _ATN =
158         new ATNDeserializer().deserialize(_serializedATN.toCharArray());
159     static {
160         _decisionToDFA = new DFA[_ATN.getNumberOfDecisions()];
161         for (int i = 0; i < _ATN.getNumberOfDecisions(); i++) {
162             _decisionToDFA[i] = new DFA(_ATN.getDecisionState(i), i);
163         }
164     }
```

Esse trecho de código está relacionado à configuração da Análise Sintática do ANTLR.

Arquivo: *GramaticaLexer.interp*

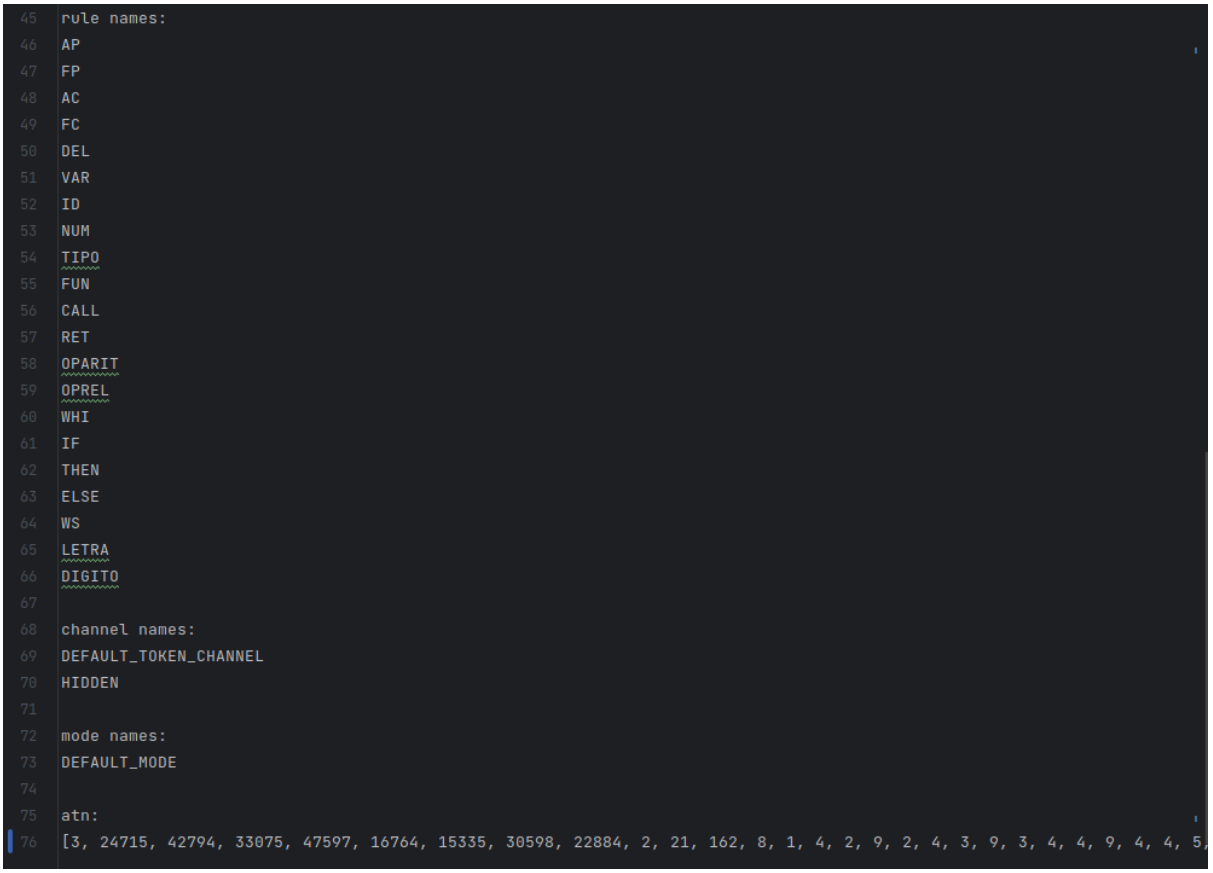
src/GramaticaLexer.interp × src/GramaticaLexer.tokens TrabalhoCompiladores/GramaticaLexer.interp TrabalhoCompilad

```
token literal names:
null
'('
')'
'{'
'}'
';'
'vars'
null
null
null
'func'
'call'
'return'
null
null
'while'
'if'
'then'
'else'
null

```

```
token symbolic names:
null
AP
FP
AC
FC
DEL
VAR
ID
NUM
TIPO
FUN
CALL
RET
OPARIT
OPREL
WHI
IF
THEN
ELSE
WS

```



O arquivo .interp é uma representação intermediária da gramática e dos dados relacionados, que são utilizados durante a execução do analisador gerado pelo ANTLR.

Arquivo: GramaticaLexer.token

```
AP=1
FP=2
AC=3
FC=4
DEL=5
VAR=6
ID=7
NUM=8
TIPO=9
FUN=10
CALL=11
RET=12
OPARIT=13
OPREL=14
WHI=15
IF=16
THEN=17
ELSE=18
WS=19
'('=1
')'=2
'{'=3
'}'=4
';'=5
'vars'=6
'func'=10
'call'=11
'return'=12
'while'=15
'if'=16
'then'=17
'else'=18
```

Cada token definido na gramática possui um nome único e um valor associado, que pode ser uma string literal ou um identificador único. O arquivo .tokens lista esses nomes e valores correspondentes para cada token definido.

Arquivos gerados pela equipe

Gramatica.g4

```
grammar Gramatica;

AP: '(';
FP: ')';
AC: '{';
FC: '}';
DEL: ';';
VAR: 'vars';
ID: LETRA(DIGITO | LETRA)*;
NUM: DIGITO+('.'DIGITO+)?;
TIPO: 'int' | 'float' | 'boolean' | 'str';
FUN: 'func';
CALL: 'call';
RET: 'return';
OPARIT: '+' | '-' | '*' | '/';
OPREL: '>' | '<' | '>=' | '<=' | '!=' | '==';
WHI: 'while';
IF: 'if';
THEN: 'then';
ELSE: 'else';
WS: [ \r\t\n]*->skip;
fragment LETRA: [a-zA-Z];
fragment DIGITO: [0-9];
```

O arquivo .g4 é o arquivo de gramática utilizado no ANTLR . Ele contém a definição da gramática formal de uma linguagem.

AppLexer.java

```
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.Token;

import java.io.IOException;

Eduardo-Cezar *
public class AppLexer {
    Eduardo-Cezar *
    public static void  main (String[] args){
        // !!!! Mudar o caminho abaixo !!!!!
        String filename = "/home/eduardo/Documentos/Materias/Compiladores/Trabalhos/TrabalhoCompiladores/codigo1.txt";
        try{
            CharStream input = CharStreams.fromFileName(filename);
            GramaticaLexer lexer = new GramaticaLexer(input);
            Token token;
            while (!lexer._hitEOF){
                token = lexer.nextToken();
                System.out.println("Token: "+ token.toString());
                System.out.println("    Lexema: "+ token.getText());
                System.out.println("    Classe: "+lexer.getVocabulary().getDisplayName(token.getType()));
            }

        } catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Este arquivo tem como intuito testar um código em nossa análise léxica e obter as saídas (Tokens) geradas pelo ANTLR.