Compiladores - Trabalho M1

Alunos: Benjamin Mezger & Pedro Prado

Estrutura dos comentários

```
/*
Comentário de bloco
/*

/* Comentário de bloco /*

# Comentário de linha
# Outro comentário de linha
```

Mensagens de erro

Código do Erro	Descrição do Erro
61	Símbolo Inválido
62	Identificador Inválido
5	Comentário de bloco não encerrado

Tokens e identificadores da linguagem 2020.1

Abaixo se encontra a tabela que especifica quais tokens existem na linguagem e quais seus identificadores.

23	*read*	Palavra Reservada
25	*all*	Palavra Reservada
22	*designate*	Palavra Reservada
8	*integer*	Palavra Reservada
13	*do*	Palavra Reservada

17	*is*	Palavra Reservada
54	*logic*	Palavra Reservada
10	*string*	Palavra Reservada
27	*result*	Palavra Reservada
18	*as*	Palavra Reservada
9	*real*	Palavra Reservada
14	*this*	Palavra Reservada
28	*repeat*	Palavra Reservada
11	*true*	Palavra Reservada
29	*declaration*	Palavra Reservada
15	*description*	Palavra Reservada
16	*body*	Palavra Reservada
20	*constant*	Palavra Reservada
26	*avaliate*	Palavra Reservada
19	*and*	Palavra Reservada
30	*type*	Palavra Reservada
21	*variable*	Palavra Reservada
12	*untrue*	Palavra Reservada
Identificador	**Token**	**Descrição**
24	*write*	Palavra Reservada

Identificador	Token	Descrição
32	-	Operador Aritmético
50	,	Simbolo especial
40	!=	Operador Lógico
49	.	Simbolo especial
47	(	Simbolo especial
48	)	Simbolo especial
45	[	Simbolo especial
46	]	Simbolo especial
33	*	Operador Aritmético

35	**	Operador Aritmético
34	/	Operador Aritmético
36	%	Operador Aritmético
37	%%	Operador Aritmético
31	+	Operador Aritmético
41	<<	Operador Relacional
43	<<=	Operador Relacional
39	=	Operador Relacional
38	==	Operador Relacional
42	>>	Operador Relacional
44	>>=	Operador Relacional
51	&	Operador Lógico
53	!	Operador Lógico
52	|	Operador Lógico

Código JavaCC

```
options {
   IGNORE_CASE = true;
   JAVA_UNICODE_ESCAPE = true;
   STATIC = false;
   DEBUG_PARSER = true;
}

PARSER_BEGIN(LanguageParser)
package compiler.parser;

import javax.swing.text.html.parser.Parser;import java.util.List;
import java.util.ArrayList;
import java.io.InputStream;
import java.io.ByteArrayInputStream;
import java.lang.StringBuilder;

public class LanguageParser {

   public static List<Token> getTokens(String stream){
      InputStream target =  new ByteArrayInputStream(stream.getBytes());
      LanguageParser parser = new LanguageParser(target);
```

```
        return tokenize(parser);
    }

    public static void main(String args[]) throws TokenMgrError, ParseException {
        LanguageParser parser = null;

        if (args.length == 0) {
            parser = new LanguageParser(System.in);
        }
        else if (args.length == 1) {
            try {
                parser = new LanguageParser(new java.io.FileInputStream(args[0]));
            }
            catch (java.io.FileNotFoundException e) {
                System.out.println("LanguageParser: file " + args[0] + " was not found.");
                return;
            }
        }

        for (Token token: tokenize(parser)){
            String name = LanguageParserConstants.tokenImage[token.kind];
            System.out.println("Line " + token.beginLine + " | Column " + token.beginColumn
+ " | " + token + "\n");
        }
    }

    public static List<Token> tokenize(LanguageParser parser){
        List<Token> tokens = new ArrayList<>();

        Token token = parser.getNextToken();
        while (token.kind != LanguageParserConstants.EOF){
            tokens.add(token);
            token = parser.getNextToken();
        }

        return tokens;
    }

}
PARSER_END(LanguageParser)

SKIP: {
    " "
    | "\n"
    | "\t"
    | < "#" (~["\n"])* >
    | < "/*" > : BLOCK_COMMENT_STATE
}

<BLOCK_COMMENT_STATE> SKIP: {
```

```
   <"/*"> : DEFAULT
   | <~[]>
}


/* Keywords */
TOKEN: {
   <INTEGER: "integer">
   | <REAL: "real">
   | <STRING: "string">
   | <TRUE: "true">
   | <UNTRUE: "untrue">
   | <DO: "do">
   | <THIS: "this">
   | <DESCRIPTION: "description">
   | <BODY: "body">
   | <IS: "is">
   | <AS: "as">
   | <AND: "and">
   | <CONSTANT: "constant">
   | <VARIABLE: "variable">
   | <DESIGNATE: "designate">
   | <READ: "read">
   | <WRITE: "write">
   | <ALL: "all">
   | <AVALIATE: "avaliate">
   | <RESULT: "result">
   | <REPEAT: "repeat">
   | <DECLARATION: "declaration">
   | <TYPE: "type">
}

/* Operators, relations, etc*/
TOKEN: {
   <PLUS : "+">
   | <MINUS : "-">
   | <MULTIPLY : "*">
   | <DIV : "/">
   | <POWER : "**">
   | <WHOLE_DIV : "%">
   | <REST_DIV: "%%">
   | <EQUAL_TO: "==">
   | <ASSIGN: "=">
   | <DIFF_THAN: "!=">
   | <LESS_THAN: "<<">
   | <GREATER_THAN: ">>">
   | <LESS_THAN_OR_EQ_TO: "<<=">
   | <GREATER_THAN_OR_EQ_TO: ">>=">
   | <OPEN_BRACKET: "[">
   | <CLOSE_BRACKET: "]">
```

```
   | <OPEN_PARENTHESIS: "(">
   | <CLOSE_PARENTHESIS: ")">
   | <DOT: ".">
   | <COMMA: ",">

}

/* Logical Operators */
TOKEN: {
   <LOGICAL_AND : "&" >
   | <LOGICAL_OR : "|">
   | <LOGICAL_NOT: "!">
}

/* Identifiers */
TOKEN: {
    <IDENTIFIER:
(<LETTER>)+((<LETTER>)*<DIGITS>(<LETTER>|"_"))*(("_")|<LETTER>)*>
   | <#LETTER: ["a"-"z","A"-"Z"]>
   | <#DIGITS: (["0"-"9"])+>
}

/* Numbers  */
TOKEN:  {
   <NUM : (<MINUS>)?(<DIGIT>)+>
   | <DOUBLE : ((<MINUS>)? (<DIGIT>)+ <DOT> (<DIGIT>)+)>
   | <#DIGIT : ["0" - "9"]>
}

TOKEN: {
   <STRING_CONSTANT: "\"" ( ~["\"","\\","\n","\r"]
      | "\\" ( ["n","t","b","r","f","\\","\'","\""]
      | ( ["\n","\r"]
      | "\r\n")))* "\"">
}

/* Catch all for undefined tokens */
TOKEN : {
   <OTHER: ~[]> |
   <INVALID_IDENTIFIER:
      <LETTER>(<LETTER>|<DIGITS>|("_"))*<DIGITS>(<DIGITS>)+(<LETTER>|("_"))* |
      <LETTER>(<LETTER>|<DIGITS>|("_"))*(<DIGITS>)+ |
      <LETTER>(<LETTER>|<DIGITS>)*(("_"))+<DIGITS>(<LETTER>|<DIGITS>|("_"))* |
      (<DIGITS>)+(<LETTER>|<DIGITS>|("_"))* |
      ("_")(<LETTER>|<DIGITS>|("_"))*>
}
```