Universidade do Vale do Itajaí
Campus Kobrasol

Trabalho Analisador Léxico

Compiladores

Katya Balvedi e Pedro André

Professor: Alessandro Mueller

Arquivo JavaCC

```
options {
   STATIC = false; // define se as classes e métodos gerados pelo
JavaCC serão static ou não.
   LOOKAHEAD = 1; // número de tokens que o parser deve olhar à
frente para tomar decisões durante o parsing.
   DEBUG_LOOKAHEAD = true; // Habilita a depuração para o
processo de lookahead.
}

PARSER_BEGIN(LanguageParser)

package classes;

public class LanguageParser {
   private StringBuilder resultado = new StringBuilder();

   public void leituraToken()  {
       try {
           Token token = null;
           token = getNextToken();
           System.out.println("KIND DO TOKEN LIDO: " +
token.kind);
           if (token.kind == EOF) return;
           System.out.println("PALAVRA_RESERVADA: " +
PALAVRA_RESERVADA);
           if (token.kind == PALAVRA_RESERVADA){
               resultado.append("Token: '" + token.image + "' -
Tipo: 'PALAVRA_RESERVADA'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
               System.out.println("Token: ['" + token.image + "']
- Tipo: 'PALAVRA_RESERVADA' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
           }else if (token.kind == IDENTIFICADOR){
               resultado.append("Token: '" + token.image + "' -
Tipo: 'IDENTIFICADOR'  -  " + "ID: " +  token.kind + " - Linha:
'" + token.beginLine + "'  -  Coluna: '" + token.beginColumn +
"'\n");
               System.out.println("Token: ['" + token.image + "']
- Tipo: 'IDENTIFICADOR' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
           }else if(token.kind == CONSTANTE_LITERAL){
               resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_LITERAL'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
               System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_LITERAL' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
```

```
token.beginColumn + "'\n");
            }else if(token.kind == CONSTANTE_INTEIRO){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_INTEIRA'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_INTEIRA' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == CONSTANTE_REAL){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_REAL'  -  " + "ID: " +  token.kind + " - Linha:
'" + token.beginLine + "'  -  Coluna: '" + token.beginColumn +
"'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_REAL' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == SIMBOLO_ESPECIAL){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'SIMBOLO_ESPECIAL'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'SIMBOLO_ESPECIAL' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == OPERADOR_ARITMETICO){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'OPERADOR_ARITMETICO'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'OPERADOR_ARITMETICO' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == OPERADOR_RELACIONAL){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'OPERADOR_RELACIONAL'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'OPERADOR_RELACIONAL' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == OPERADOR_LOGICO){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'OPERADOR_LOGICO'  -  " + "ID: " +  token.kind + " - Linha:
'" + token.beginLine + "'  -  Coluna: '" + token.beginColumn +
"'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'OPERADOR_LOGICO' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
```

```
            }else if(token.kind == SIMBOLO_INVALIDO){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'SIMBOLO_INVALIDO'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'SIMBOLO_INVALIDO' -  " + "Id: '" +  token.kind + "' -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == CONSTANTE_LITERAL_INVALIDA){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_LITERAL_INVALIDA'  -  " + "ID: " +  token.kind +
" - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_LITERAL_INVALIDA' -  " + "Id: '" +  token.kind
+ "' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == CONSTANTE_INTEIRA_INVALIDA){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_INTEIRA_INVALIDA'  -  " + "ID: " +  token.kind +
" - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_INTEIRA_INVALIDA' -  " + "Id: '" +  token.kind
+ "' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == CONSTANTE_REAL_INVALIDA){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'CONSTANTE_REAL_INVALIDA'  -  " + "ID: " +  token.kind + "
- Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_REAL_INVALIDA' -  " + "Id: '" +  token.kind +
"' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }else if(token.kind == IDENTIFICADOR_INVALIDO){
                resultado.append("Token: '" + token.image + "' -
Tipo: 'IDENTIFICADOR_INVALIDO'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'IDENTIFICADOR_INVALIDO' -  " + "Id: '" +  token.kind +
"' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
            }
            leituraToken();
        }catch(Error erro){
            resultado.append("Erro - " + erro.getMessage() +
"\n");
            System.out.println(erro.toString());
            leituraToken();
        }
    }
    public String getTokens(String args[], String
```

```
textoParaAnalisar) {
            LanguageParser languageParser;
            if(args.length == 0){

System.out.println("#####################################");
                System.out.println("### Iniciando leitura dos
tokens ...");

System.out.println("#####################################");
                java.io.InputStream targetStream = new
java.io.ByteArrayInputStream(textoParaAnalisar.getBytes());
                languageParser = new LanguageParser(targetStream);
            }else if(args.length == 1){
                try{
                    languageParser = new LanguageParser(new
java.io.FileInputStream(args[0]));
                }catch(java.io.FileNotFoundException e){
                    System.err.println(args[0] + " não foi
encontrado." );
                    System.err.println(e);
                    return args[0] + " não foi encontrado.";
                }
            }else{
                System.out.println("Use:\njava Analisador Lexico <
inputFile");
                System.out.println("ou java Analisador Lexico
inputFile");
                return "Use:java Analisador Lexico < inputFile";
            }
            this.leituraToken();
            resultado.append("<EOF>");
            return resultado.toString();
        }

        public String getResultado(){
            return resultado.toString();
        }
}


PARSER_END(LanguageParser)
//TOKENS
//CARACTERES ESPECIAIS IGNORADOS
SKIP : {
   " " | "\t" | "\n" | "\r" | "\f" | "\r\n"  |  "\n\r"
}

//COMENTARIO DE LINHA Tudo será ignorado após o // e quando
chegar no /n o que estiver escrito não será mais ignorado
SKIP:
{
   "//": COMENTARIO_LINHA
}
<COMENTARIO_LINHA> SKIP:
{
```

```
       <["\n" , "\r"]> : DEFAULT
   |   <~[]>
}
//COMENTARIOS DE BLOCO - Tudo será ignorado após o /* e quando
chegar no */ o que estiver escrito não será mais ignorado
SKIP:
{
   "/*" : COMENTARIO_BLOCO
}
<COMENTARIO_BLOCO> SKIP:
{
       "*/" : DEFAULT
   |   <~[]>
}

//PALAVRA RESERVADA
TOKEN :
{
   <PALAVRA_RESERVADA:
         "do"
       | "make"
       | "end"
       | "const"
       | "var"
       | "int"
       | "real"
       | "char"
       | "bool"
       | "get"
       | "put"
       | "if"
       | "then"
       | "else"
       | "true"
       | "false"
       | "while">
}

//Simbolo Especial
TOKEN :
{
   <SIMBOLO_ESPECIAL:
         "(" //ABRE_PARENTESES
       | ")" //FECHA_PARENTESES
       | "." //PONTO
       | "," //VIRGULA
       | ":" //DOISPONTOS
       | "->" //ATRIBUICAO
       | ";"> //PONTOVIRGULA
}

//Simbolo Especial Operadores Aritiméticos
TOKEN :
{
     <OPERADOR_ARITMETICO:
```

```
          "+" // ADICAO
        | "-" //SUBTRACAO
        | "*" //MULTIPLICACAO
        | "/" //DIVISAO
        | "**" //POTENCIA
        | "%" //DIVISAO INTEIRA
        | "%%"> //RESTO
}

//Simbolo Especial Operadores Relacionais
TOKEN :
{
     <OPERADOR_RELACIONAL:
         "=" //IGUAL
       | "<>" //DIFERENTE
       | "<" //MENOR
       | ">" //MAIOR
       | "<=" //MENOR_IGUAL
       | ">="> //MAIOR_IGUAL
}

//Simbolo Especial Operadores Lógicos
TOKEN :
{
     <OPERADOR_LOGICO:
         "&" //E
       | "|" //OU
       | "!"> //NAO
}

//CONSTANTES NUMÉRICAS
TOKEN :
{
     <#DIGITO: ["0"-"9"]>
   | <#NUMERO_DECIMAL: "." <DIGITO>((<DIGITO>)*)>
   | <CONSTANTE_INTEIRO:
<DIGITO>((<DIGITO>)?)((<DIGITO>)?)((<DIGITO>)?)>
   | <CONSTANTE_REAL: <DIGITO>((<DIGITO>)?)<NUMERO_DECIMAL>>
}

//CONSTANTE LITERAIS 'PALAVRAS' OU "FRASES"
TOKEN :
{
   <CONSTANTE_LITERAL: "\"" (~["\n","\r","\"","\'"])*"\"" |
"'"(~["\n","\r","\"","\'"])*"'">
}

//IDENTIFICADORES EXEMPLOS VÁLIDOS: vAr_3 | vvv | v2 | vAv
TOKEN :
{
     <#LETRA: ["A"-"Z","a"-"z"]>
   | <#LETRA_MAIUSCULA: ["A"-"Z"]>
   | <#LETRA_MINUSCULA: ["a"-"z"]>
   | <IDENTIFICADOR: <LETRA_MINUSCULA> (<LETRA> | <DIGITO> |
("_"(<LETRA> | <DIGITO>) ))* | <DIGITO>>
```

```
}

//ERROS O ~FAZ A NEGAÇÃO DOS SIMBOLOS VÁLIDOS, ENTÃO QUALQUER
SIMBOLO DIFERENTE DESSES SERÃO INVÁLIDADOS
TOKEN :
{
    <SIMBOLO_INVALIDO:
        (~["a"-"z","A"-"Z","0"-"9",
        "\n","\r","\t","\f"," ",
        "(", ")", ";", ".", ",", ":", "+", "-", "*", "/", "*",
"%", "=", "<", ">", "&", "|", "!" ])+>
   | <CONSTANTE_INTEIRA_INVALIDA:
(<DIGITO>)+(<DIGITO>)+(<DIGITO>)+(<DIGITO>)+(<DIGITO>)*  >
   | <CONSTANTE_REAL_INVALIDA:
          <DIGITO>((<LETRA>)+)<NUMERO_DECIMAL>
        | <DIGITO>((<DIGITO>)*)"."."
        |
<DIGITO>((<DIGITO>)*)<NUMERO_DECIMAL>((<DIGITO>)*)(<LETRA>)+
        | "." (<DIGITO>)+
        | <DIGITO>((<DIGITO>)*)"." (<LETRA>)+>
   | <CONSTANTE_LITERAL_INVALIDA:
"\""(~["\n","\r","\"","\'"])*~["\""] |
"'"(~["\n","\r","\"","\'"])*~["\'"]>
   | <IDENTIFICADOR_INVALIDO: ( (<LETRA_MAIUSCULA>)+ | ("_")+)
((<LETRA>)*|(<DIGITO>)*)>
)
```

Arquivo JavaCC - Definição da linguagem

```
//TOKENS
//CARACTERES ESPECIAIS IGNORADOS
SKIP : {
   " " | "\t" | "\n" | "\r" | "\f" | "\r\n"  |  "\n\r"
}

//COMENTARIO DE LINHA Tudo será ignorado após o // e quando
chegar no /n o que estiver escrito não será mais ignorado
SKIP:
{
   "//": COMENTARIO_LINHA
}
<COMENTARIO_LINHA> SKIP:
{
       <["\n" , "\r"]> : DEFAULT
   |   <~[]>
}
//COMENTARIOS DE BLOCO - Tudo será ignorado após o /* e quando
chegar no */ o que estiver escrito não será mais ignorado
SKIP:
```

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

//PALAVRA RESERVADA
TOKEN :
{
   <PALAVRA_RESERVADA:
         "do"
       | "make"
       | "end"
       | "const"
       | "var"
       | "int"
       | "real"
       | "char"
       | "bool"
       | "get"
       | "put"
       | "if"
       | "then"
       | "else"
       | "true"
       | "false"
       | "while">
}

//Simbolo Especial
TOKEN :
{
   <SIMBOLO_ESPECIAL:
         "(" //ABRE_PARENTESES
       | ")" //FECHA_PARENTESES
       | "." //PONTO
       | "," //VIRGULA
       | ":" //DOISPONTOS
       | "->" //ATRIBUICAO
       | ";"> //PONTOVIRGULA
}

//Simbolo Especial Operadores Aritiméticos
TOKEN :
{
     <OPERADOR_ARITMETICO:
         "+" // ADICAO
       | "-" //SUBTRACAO
       | "*" //MULTIPLICACAO
       | "/" //DIVISAO
       | "**" //POTENCIA
       | "%" //DIVISAO INTEIRA
```

```
       | "%%"> //RESTO
}

//Simbolo Especial Operadores Relacionais
TOKEN :
{
     <OPERADOR_RELACIONAL:
         "=" //IGUAL
       | "<>" //DIFERENTE
       | "<" //MENOR
       | ">" //MAIOR
       | "<=" //MENOR_IGUAL
       | ">="> //MAIOR_IGUAL
}

//Simbolo Especial Operadores Lógicos
TOKEN :
{
     <OPERADOR_LOGICO:
         "&" //E
       | "|" //OU
       | "!"> //NAO
}

//CONSTANTES NUMÉRICAS
TOKEN :
{
     <#DIGITO: ["0"-"9"]>
   | <#NUMERO_DECIMAL: "." <DIGITO>((<DIGITO>)*)>
   | <CONSTANTE_INTEIRO:
<DIGITO>((<DIGITO>)?)((<DIGITO>)?)((<DIGITO>)?)>
   | <CONSTANTE_REAL: <DIGITO>((<DIGITO>)?)<NUMERO_DECIMAL>>
}

//CONSTANTE LITERAIS 'PALAVRAS' OU "FRASES"
TOKEN :
{
   <CONSTANTE_LITERAL: "\"" (~["\n","\r","\"","\'"])*"\"" |
"'"(~["\n","\r","\"","\'"])*"'">
}

//IDENTIFICADORES EXEMPLOS VÁLIDOS: vAr_3 | vvv | v2 | vAv
TOKEN :
{
     <#LETRA: ["A"-"Z","a"-"z"]>
   | <#LETRA_MAIUSCULA: ["A"-"Z"]>
   | <#LETRA_MINUSCULA: ["a"-"z"]>
   | <IDENTIFICADOR: <LETRA_MINUSCULA> (<LETRA> | <DIGITO> |
("_"(<LETRA> | <DIGITO>) ))* | <DIGITO>>
}

//ERROS O ~FAZ A NEGAÇÃO DOS SIMBOLOS VÁLIDOS, ENTÃO QUALQUER
SIMBOLO DIFERENTE DESSES SERÃO INVÁLIDADOS
TOKEN :
{
```

```
    <SIMBOLO_INVALIDO:
        (~["a"-"z","A"-"Z","0"-"9",
        "\n","\r","\t","\f"," ",
        "(", ")", ";", ".", ",", ":", "+", "-", "*", "/", "*",
"%", "=", "<", ">", "&", "|", "!" ])+>
   | <CONSTANTE_INTEIRA_INVALIDA:
(<DIGITO>)+(<DIGITO>)+(<DIGITO>)+(<DIGITO>)+(<DIGITO>)*  >
   | <CONSTANTE_REAL_INVALIDA:
         <DIGITO>((<LETRA>)+)<NUMERO_DECIMAL>
        | <DIGITO>((<DIGITO>)*)"."."
        |
<DIGITO>((<DIGITO>)*)<NUMERO_DECIMAL>((<DIGITO>)*)(<LETRA>)+
        | "." (<DIGITO>)+
        | <DIGITO>((<DIGITO>)*)"." (<LETRA>)+>
   | <CONSTANTE_LITERAL_INVALIDA:
"\""(~["\n","\r","\"","\'"])*~["\""] |
"'"(~["\n","\r","\"","\'"])*~["\'"]>
   | <IDENTIFICADOR_INVALIDO: ( (<LETRA_MAIUSCULA>)+ | ("_")+)
((<LETRA>)*|(<DIGITO>)*)>
}
```

Estrutura de comentário

Comentário de linha

// Qualquer simbolo/palavra

Comentário de bloco

/* qualquer simbolo/palavra */

Tabela de símbolos terminais

(token, código do token, descrição do token)

```
int EOF = 0;
 /** RegularExpression Id. */
 int PALAVRA_RESERVADA = 14; //palavras reservadas
 /** RegularExpression Id. */
 int SIMBOLO_ESPECIAL = 15;  //operadores especiais
 /** RegularExpression Id. */
 int OPERADOR_ARITMETICO = 16; //operadores aritmeticos
 /** RegularExpression Id. */
 int OPERADOR_RELACIONAL = 17; //operadores relacionais
 /** RegularExpression Id. */
```

```
 int OPERADOR_LOGICO = 18;    //operadores lógicos
 /** RegularExpression Id. */
 int DIGITO = 19; //digitos
 /** RegularExpression Id. */
 int NUMERO_DECIMAL = 20; //numeros decimais depois da
virgula
 /** RegularExpression Id. */
 int CONSTANTE_INTEIRO = 21; //constantes numericas
inteiras
 /** RegularExpression Id. */
 int CONSTANTE_REAL = 22; //constantes numericas reais
 /** RegularExpression Id. */
 int CONSTANTE_LITERAL = 23; //constantes literais
 /** RegularExpression Id. */
 int LETRA = 24; //Letras tanto maiúscula quanto
minuscula
 /** RegularExpression Id. */
 int LETRA_MAIUSCULA = 25; //linguagem com tabela ASCII
em maiuscula
 /** RegularExpression Id. */
 int LETRA_MINUSCULA = 26; //linguagem com tabela ASCII
em minuscula
 /** RegularExpression Id. */
 int IDENTIFICADOR = 27; //identifica um identificador
 /** RegularExpression Id. */
 int SIMBOLO_INVALIDO = 28; //Simbolos invalidos
 /** RegularExpression Id. */
 int CONSTANTE_INTEIRA_INVALIDA = 29; //Constantes
numericas invalidas, neste caso constantes inteiras
 /** RegularExpression Id. */
 int CONSTANTE_REAL_INVALIDA = 30; //Constantes numericas
invalidas, neste caso são constantes reais
 /** RegularExpression Id. */
 int CONSTANTE_LITERAL_INVALIDA = 31; // Constantes
literais invalidas
```

```
 /** RegularExpression Id. */
 int IDENTIFICADOR_INVALIDO = 32; //Identificadores
invalidos

 /** Lexical state. */
 int DEFAULT = 0; //define como estado default
 /** Lexical state. */
 int COMENTARIO_LINHA = 1; //Comentario de linha
 /** Lexical state. */
 int COMENTARIO_BLOCO = 2; //Comentário em bloco
```

Mensagem de erro

(código do erro e descrição do erro)

Tokens

```
 int SIMBOLO_INVALIDO = 28;
 /** RegularExpression Id. */
 int CONSTANTE_INTEIRA_INVALIDA = 29;
 /** RegularExpression Id. */
 int CONSTANTE_REAL_INVALIDA = 30;
 /** RegularExpression Id. */
 int CONSTANTE_LITERAL_INVALIDA = 31;
 /** RegularExpression Id. */
 int IDENTIFICADOR_INVALIDO = 32;
```

Descrição do erro *(esta na linguagem.jj)*

```
CONSTANTE_LITERAL_INVALIDA
resultado.append("Token: '" + token.image + "' - Tipo:
'CONSTANTE_LITERAL_INVALIDA'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_LITERAL_INVALIDA' -  " + "Id: '" +  token.kind
+ "' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");

CONSTANTE_INTEIRA_INVALIDA
resultado.append("Token: '" + token.image + "' - Tipo:
'CONSTANTE_INTEIRA_INVALIDA'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
```

```
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_INTEIRA_INVALIDA' -  " + "Id: '" +  token.kind
+ "' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");

CONSTANTE_REAL_INVALIDA
resultado.append("Token: '" + token.image + "' - Tipo:
'CONSTANTE_REAL_INVALIDA'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'CONSTANTE_REAL_INVALIDA' -  " + "Id: '" +  token.kind +
"' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");

IDENTIFICADOR_INVALIDO
resultado.append("Token: '" + token.image + "' - Tipo:
'IDENTIFICADOR_INVALIDO'  -  " + "ID: " +  token.kind + " -
Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
                System.out.println("Token: ['" + token.image + "']
- Tipo: 'IDENTIFICADOR_INVALIDO' -  " + "Id: '" +  token.kind +
"' - Linha: '" + token.beginLine + "'  -  Coluna: '" +
token.beginColumn + "'\n");
```