Tabela de símbolos terminais (token, código do token, descrição do token)

RPAREN	")"
LPAREN	"("
SEMICOLON	";"
COMMA	","
DOT	"."
null_constant	("n" | "N") ("u" | "U") ("l" | "L") ("l" | "L")
FALSE	("f" | "F") ("a" | "A") ("l" | "L") ("s" | "S") ("e" | "E")
TRUE	("t" | "T") ("r" | "R") ("u" | "U") ("e" | "E")
string_constant	("\"" (~["\"", "\n", "\r"])* "\"" | "'" (~["'", "\n", "\r"])* "'")
float_constant	(["0"-"9"]) (["0"-"9"])? (["0"-"9"])? (["0"-"9"])? "." (["0"-"9"]) (["0"-"9"])?
char_constant	("\"" ~["\"", "\n", "\r", "\f"] "\"" | "'" ~["'", "\n", "\r", "\f"] "'")
int_constant	(["0"-"9"]) (["0"-"9"])? (["0"-"9"])?
DIVISION_REST	"%%"
ENTIRE_DIVISION	"%"
POTENCY	"**"
DIVISION	"/"
MULTIPLICATION	"*"
MINUS	"-"
PLUS	"+"
BIGGER_EQUAL	">="
BIGGER	">"

MINOR_EQUAL	"<="
MINOR	"<"
DIFFERENT	"<>"
EQUAL	"="
ASSIGN	"->"
END_LOGIC OR	"&" "|"
NOT	"!"
DO	("d" | "D") ("o" | "O")
WHILE	("w" | "W") ("h" | "H") ("i" | "I") ("l" | "L") ("e" | "E")
ELSE	("e" | "E") ("l" | "L") ("s" | "S") ("e" | "E")
THEN	("t" | "T") ("h" | "H") ("e" | "E") ("n" | "N")
IF	("i" | "I") ("f" | "F")
PUT	("p" | "P") ("u" | "U") ("t" | "T")
GET	("g" | "G") ("e" | "E") ("t" | "T")
VAR	("v" | "V") ("a" | "A") ("r" | "R")
BOOL	("b" | "B") ("o" | "O") ("o" | "O") ("l" | "L")
CHAR	("c" | "C") ("h" | "H") ("a" | "A") ("r" | "R")
REAL	("r" | "R") ("e" | "E") ("a" | "A") ("l" | "L")
INT	("i" | "I") ("n" | "N") ("t" | "T")
CONST	("c" | "C") ("o" | "O") ("n" | "N") ("s" | "S") ("t" | "T")
END	("e" | "E") ("n" | "N") ("d" | "D")
MAKE	("m" | "M") ("a" | "A") ("k" | "K") ("e" | "E")

```
SKIP : {
   " " | "\t" | "\n" | "\r"| "\f"  // Ignora espaços em branco e
quebras de linha
}
```

Estrutura dos comentários de linha e de bloco; PDF e impresso

Comentário de linha
//

Comentário de bloco

/* -> inicia o comentário de bloco
*/ -> termina o comentário de bloco

Código

```
// Multiline comment
SKIP : {
   "/*" : multilinecomment
}

<multilinecomment> SKIP : {
   "*/" : DEFAULT   // Sai do modo de comentário ao
encontrar */
   | <~[]>
}

// sigle line comment
SKIP : {
   "//" : singlelinecomment
}

<singlelinecomment> SKIP: {
   <["\n","\r"]>:DEFAULT
   | <~[]>
```

}

Lista de mensagens de erro

<INVALID_IDENTIFIER:	Invalid Identifier
INVALID_COMMENT_CLOSE:	Invalid Closing of a Comment:
INVALID_CONST_INT_SIZE:	Invalid Int Size:
INVALID_CONST_FLOAT_SIZE:	Invalid Float Size:
INVALID_CONST_STRING_END:	Invalid String/Character Ending:
INVALID_CONST_STRING_WITH_LINEBREAK:	Invalid String With LineBreak:
INVALID_LEXICAL	Invalid Character Found:

Arquivo do JavaCC

```
options {
   STATIC = false;
}
PARSER_BEGIN(prataLang)
package compilador.regras;
import java.io.*;
import compilador.telas.CompiladorGui;
public class prataLang {

  final static String Version = "PrataLang Compiler - Version 1.0 - 2024";
  boolean Menosshort = false;

  public static void main(String[] args) throws ParseException {
    String filename = ""; // nome do arquivo a ser analizado
    prataLang parser;
```

```
  int i;
  boolean ms = false;

  System.out.println(Version);

  // le os parametros passados para o compilador
  for(i = 0; i < args.length - 1; i++) {
    if ( args[i].equalsIgnoreCase("-short") ) ms = true;
    else{
      System.out.println("Usage is: java PrataLang [-short] inputfile");
      System.exit(0);
    }
  }

  if (args[i].equals("-")){
    // le entrada padrão
     System.out.println(" Reading from standard input . . . ");
     parser = new prataLang(System.in);
  } else {
    // le do arquivo
    filename = args[args.length-1];
    System.out.println("Readig from file " + filename + " . . . ");
    try{
      parser = new prataLang(new java.io.FileInputStream(filename));
    } catch (java.io.FileNotFoundException e ) {
      System.out.println("File "+filename+ " NOT FOUND");
      return;
    }
  }
  parser.Menosshort = ms;
  //parser.program(); //chama o metodo que faz a analise

  // verifica se houver erro lexico
  if (parser.token_source.foundLexError() != 0)
    System.out.println(
      parser.token_source.foundLexError() + " Erro Lexico encontrado!! "
    );
  else System.out.println(" Program successfully analized ");



 } //main

  // Metodo que é chamado no CompiladorGui para compilar pelo botão e pelo menu
 public static void Compile(String file, prataLang parser, CompiladorGui gui) throws
ParseException {
    String filename = file; // nome do arquivo a ser analizado
    boolean ms = false;
```

```
      gui.jTextArea1.append(Version + "\n");

      gui.jTextArea1.append("Readig from file " + filename + " . . .  \n");

      try{
          parser = new prataLang(new java.io.FileInputStream(filename));
      } catch (java.io.FileNotFoundException e ) {
          gui.jTextArea1.append("File "+filename+ " NOT FOUND \n");
          return;
      }

      parser.Menosshort = ms;
      parser.program(gui); //chama o metodo que faz a analise

      if (parser.token_source.foundLexError() != 0)
          gui.jTextArea1.append(
            parser.token_source.foundLexError() + " Lexic Error Found \n"
          );
      else gui.jTextArea1.append(" Successfully Analized Program \n");
  }

  //metodo auxiliar
  static public String im(int x) {
    int  k;
    String s;
    s = tokenImage[x];
    k = s.lastIndexOf("\"");
    try{
      s=s.substring(1,k);
    }
    catch (StringIndexOutOfBoundsException e ){ }
    return s;
  }
}

PARSER_END(prataLang)

TOKEN_MGR_DECLS : {
    int countLexError = 0;

    public int foundLexError(){
        return countLexError;
    }
}

SKIP : {
    " " | "\t" | "\n" | "\r"| "\f"  // Ignora espaços em branco e quebras de linha
```

```
}

// Multiline comment
SKIP : {
   "/*" : multilinecomment
}

<multilinecomment> SKIP : {
   "*/" : DEFAULT   // Sai do modo de comentário ao encontrar */
   | <~[]>
}

// sigle line comment
SKIP : {
   "//" : singlelinecomment
}

<singlelinecomment> SKIP: {
   <["\n","\r"]>:DEFAULT
   | <~[]>
}

// PALAVRAS RESERVADAS
TOKEN : {
     < MAKE : ("m" | "M") ("a" | "A") ("k" | "K") ("e" | "E") >
   {
                  error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");;
   }
   | < END  : ("e" | "E") ("n" | "N") ("d" | "D") >
   {
                  error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   | < CONST  : ("c" | "C") ("o" | "O") ("n" | "N") ("s" | "S") ("t" | "T") >
   {
                  error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   | < INT  : ("i" | "I") ("n" | "N") ("t" | "T") >
   {
                  error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   | < REAL : ("r" | "R") ("e" | "E") ("a" | "A") ("l" | "L") >
   {
                 error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
```

```
  }
  | < CHAR : ("c" | "C") ("h" | "H") ("a" | "A") ("r" | "R") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < BOOL : ("b" | "B") ("o" | "O") ("o" | "O") ("l" | "L") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < VAR : ("v" | "V") ("a" | "A") ("r" | "R") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < GET : ("g" | "G") ("e" | "E") ("t" | "T") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < PUT : ("p" | "P") ("u" | "U") ("t" | "T") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < IF : ("i" | "I") ("f" | "F") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < THEN : ("t" | "T") ("h" | "H") ("e" | "E") ("n" | "N") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < ELSE : ("e" | "E") ("l" | "L") ("s" | "S") ("e" | "E") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < WHILE : ("w" | "W") ("h" | "H") ("i" | "I") ("l" | "L") ("e" | "E") >
  {
        error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
  }
  | < DO : ("d" | "D") ("o" | "O") >
  {
```

```
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
}

// OPERADORES LOGICOS
TOKEN :
{
      < NOT : "!" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < OR : "|" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < END_LOGIC : "&" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
}

// OPERADORES RELACIONAIS
TOKEN :
{
      < ASSIGN : "->" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < EQUAL : "=" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < DIFFERENT : "<>" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < MINOR : "<" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
```

```
    | < MINOR_EQUAL : "<=" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < BIGGER : ">" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < BIGGER_EQUAL : ">=" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
}

// OPERADORES ARITMETRIC
TOKEN :
{
      < PLUS : "+" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < MINUS : "-" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < MULTIPLICATION : "*" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < DIVISION : "/" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < POTENCY : "**" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < ENTIRE_DIVISION : "%" >
    {
```

```
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < DIVISION_REST : "%%" >
    {
                error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
}

// CONSTANTES
TOKEN :
{
      < int_constant  : (["0"-"9"]) (["0"-"9"])? (["0"-"9"])? >
     {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
     }
     | < char_constant : ("\"" ~["\"", "\n", "\r", "\f"] "\"" | "'" ~["'", "\n", "\r", "\f"] "'") >
     {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
     }
     | < float_constant : (["0"-"9"]) (["0"-"9"])? (["0"-"9"])? (["0"-"9"])? "." (["0"-"9"]) (["0"-"9"])? >
     {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
     }
     | < string_constant : // constante string como "abcd bcda" ou 'abcd bcda'
     ("\"" (~["\"", "\n", "\r"])* "\"" | "'" (~["'", "\n", "\r"])* "'")>
     {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
     }
    | < TRUE : ("t" | "T") ("r" | "R") ("u" | "U") ("e" | "E") >
    {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < FALSE   : ("f" | "F") ("a" | "A") ("l" | "L") ("s" | "S") ("e" | "E") >
    {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
    }
    | < null_constant   : ("n" | "N") ("u" | "U") ("l" | "L") ("l" | "L") >
    {
          error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
```

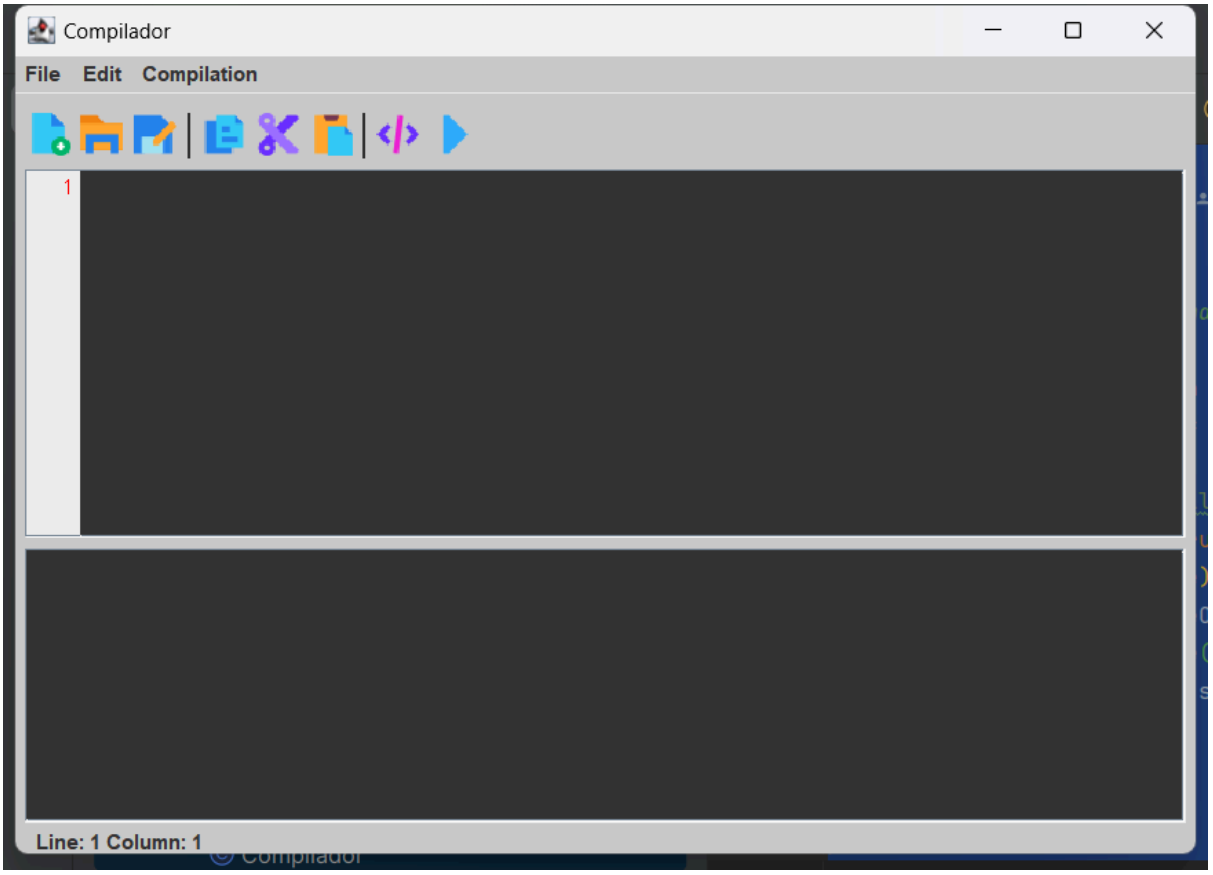
```
    }
}

// IDENTIFICADORES
TOKEN :
{
   < IDENT  : ( ["a"-"z", "A"-"Z"]  | "_" ) ( ["a"-"z", "A"-"Z"]  |"_"| ["0" - "9"] ( ["a"-"z", "A"-"Z"] | "_"
) )* (["a"-"z", "A"-"Z"] | "_") >
   {
            error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
}

// SIMBULOS ESPECIAIS
TOKEN :
{
   < DOT : "." >
   {
      error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   |
   < COMMA : "," >
   {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   |
   < SEMICOLON : ";" >
   {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   |
   < LPAREN : "(" >
   {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
   |
   < RPAREN : ")" >
   {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " Token: " + "\"" + image + "\"" + " Code: ");
   }
}
```

```
// Token para erros léxicos
TOKEN : {

   < INVALID_LEXICAL : (~["a"-"z",
      "A"-"Z",
      "0"-"9",
      "\n", "\r",
      "\t", "\f",
      "\"",
      "'",
      " ",
      "(", ")",
      "[", "]",
      "{", "}",
      ",",
      ".",
      ";",
      "=",
      "<",
      ">",
      "+",
      "-",
      "*",
      "/",
      "&",
      "|",
      "!",
      "%",
      "^",
      "_"]) + >
         {
            error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid Character Found: " + "\"" + image + "\"" + "
Code: ");;
            countLexError++;
         }
  |
      <INVALID_CONST_STRING_WITH_LINEBREAK:
      (("\"" (~ ["\""] | "\n" | "\r" )+ ("\"")) | ("'" (~ ["'"] | "\n" | "\r" )+ ("'")))>
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid String With LineBreak: " + "\"" + image + "\"" + "
Code: ");;
         countLexError++;
      }
  |
      <INVALID_CONST_STRING_END:
      ( ("\"" (~ ["\"", "\n", "\r"])*) | ("'" (~ ["\"", "\n", "\r"])* ))>
```

```
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid String/Character Ending: " + "\"" + image + "\"" +
" Code: ");;
         countLexError++;
      }
  |
      <INVALID_CONST_FLOAT_SIZE:
      ((["0"-"9"]) (["0"-"9"]) (["0"-"9"]) (["0"-"9"]) (["0"-"9"])+ "." (["0"-"9"]) (["0"-"9"])?) |
((["0"-"9"]) (["0"-"9"]) (["0"-"9"]) (["0"-"9"]) (["0"-"9"])+ "." (["0"-"9"]) (["0"-"9"]) (["0"-"9"])+) |
((["0"-"9"]) (["0"-"9"])? (["0"-"9"])? (["0"-"9"])? "." (["0"-"9"]) (["0"-"9"]) (["0"-"9"])+) | "." (["0" -
"9"])* | (["0" - "9"])* ".")>
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid Float Size: " + "\"" + image + "\"" + " Code: ");;
         countLexError++;
      }
  |
      <INVALID_CONST_INT_SIZE:
      (["0"-"9"]) (["0"-"9"]) (["0"-"9"]) (["0"-"9"])+>
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid Int Size: " + "\"" + image + "\"" + " Code: ");;
         countLexError++;
      }
|
      <INVALID_COMMENT_CLOSE:
      "*/" >
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid Closing of a Comment: " + "\"" + image + "\"" + "
Code: ");;
         countLexError++;
      }
|
      <INVALID_IDENTIFIER:
      ((["A"-"Z", "a"-"z"] | "_") (["A"-"Z", "a"-"z"] | "_" | ["0" - "9"] (["A"-"Z", "a"-"z"] ))* ["0" - "9"]) |
((["A"-"Z", "a"-"z"] | "_") (["A"-"Z", "a"-"z"] | "_" | ["0" - "9"] ["0" - "9"] )* (["A"-"Z", "a"-"z"] | "_") ) |
(( ["0" - "9"] ) ( ["a"-"z", "A"-"Z"]  |"_"| ["0" - "9"] ( ["a"-"z", "A"-"Z"] | "_" ) )* (["a"-"z", "A"-"Z"] |
"_")) ) >
      {
         error = ("Line " + input_stream.getBeginLine() + " Column " +
input_stream.getBeginColumn() + " - Invalid Identifier: " + "\"" + image + "\"" + " Code: ");;
         countLexError++;
      }
}

JAVACODE void program(CompiladorGui gui)
```

```
{
Token t;
    do
    {
    t = getNextToken();
    Token st = t;
        while ( st.specialToken != null)
            st = st.specialToken;
            do {
               if ( !Menosshort ) {
                   gui.jTextArea1.append(token_source.error + st.kind + "\n");
                      token_source.error = "";
               }
               else {
                   System.out.println(token_source.error + st.kind + "\n");
                   token_source.error = "";
               }
               st = st.next;
        } while (st != t.next);
    } while (t.kind != prataLangConstants.EOF);
    if(token_source.curLexState == 1){
          gui.jTextArea1.append(" Error to end Block Comment \n");
          token_source.countLexError++;
    }
}
```

Descrição do Funcionamento do Analisador



Na tela terão as opções de abrir um arquivo, salvar, copiar, recortar e colar, no botão de compilar ele irá realizar a análise léxica.

Na tela de cima é onde o usuário irá digitar o que quiser, enquanto a tela de baixo funciona como um prompt, onde irão aparecer os erros léxicos e suas descrições, ou se a análise foi um sucesso.

```
int EOF = 0;
int MAKE = 12;
int END = 13;
int CONST = 14;
int INT = 15;
int REAL = 16;
int CHAR = 17;
int BOOL = 18;
```

```
int VAR = 19;
int GET = 20;
int PUT = 21;
int IF = 22;
int THEN = 23;
int ELSE = 24;
int WHILE = 25
int DO = 26;
int NOT = 27;
int OR = 28;
int END_LOGIC = 29;
int ASSIGN = 30;
int EQUAL = 31;
int DIFFERENT = 32;
int MINOR = 33;
int MINOR_EQUAL = 34;
int BIGGER = 35;
int BIGGER_EQUAL = 36;
int PLUS = 37;
int MINUS = 38;
int MULTIPLICATION = 39;
int DIVISION = 40;
int POTENCY = 41;
int ENTIRE_DIVISION = 42;
int DIVISION_REST = 43;
int int_constant = 44;
int char_constant = 45;
int float_constant = 46;
int string_constant = 47;
int TRUE = 48;
int FALSE = 49;
int null_constant = 50;
int IDENT = 51;
int DOT = 52;
int COMMA = 53;
int SEMICOLON = 54;
int LPAREN = 55;
int RPAREN = 56;
```

```
int INVALID_LEXICAL = 57;
int INVALID_CONST_STRING_WITH_LINEBREAK = 58;
int INVALID_CONST_STRING_END = 59;
int INVALID_CONST_FLOAT_SIZE = 60;
int INVALID_CONST_INT_SIZE = 61;
int INVALID_COMMENT_CLOSE = 62;
int INVALID_IDENTIFIER = 63;
int DEFAULT = 0;
int multilinecomment = 1;
int singlelinecomment = 2;
```