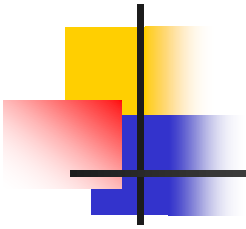


Parte 1 do Trabalho 2

- Criar um Analisador Léxico que tabula a saída código/token para PS com as extensões do grupo

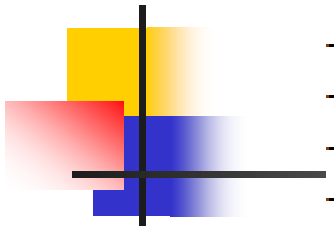
- AnaLex.jj

1. Declaração da classe e de opções

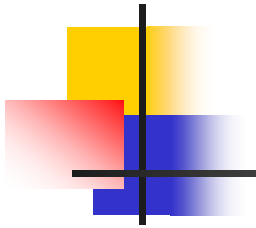
```
PARSER_BEGIN (AnaLex)
        public class AnaLex{
                public static void main(String[] args){
                        AnaLex parser = new AnaLex(System.in);
                        while(true){
                                try {
                                        parser.parser();
                                } catch (ParseException ex) {
                                        System.out.println(ex.getMessage());
                                        System.exit(-1);
                                } catch (TokenMgrError ex) {
                                        System.out.println(ex.getMessage());
                                        System.exit(-1);
                                }
                        }
                }
        }
PARSER_END(AnaLex)
```

2. Declarações da Parte Léxica

```
/**
 * Eliminando comentarios e espacos
 */
SKIP: { " " | "\r" | "\t" | "\n" }
SKIP: { <BLOCKCOMMENT: "{" (~["}"])* "}"> }
SKIP: { <LINECOMMENT: "//" (~["\n"])* ("\n"|"\r"|"\r\n")> }

/**
 * Palavras reservados
 */
TOKEN: { <AND: "and"> }
TOKEN: { <BEGIN: "begin"> }
TOKEN: { <DIV: "div"> }
TOKEN: { <DO: "do"> }
TOKEN: { <ELSE: "else"> }
TOKEN: { <END: "end"> }
TOKEN: { <IF: "if"> }
TOKEN: { <NOT: "not"> }
TOKEN: { <OR: "or"> }
TOKEN: { <PROCEDURE: "procedure"> }
TOKEN: { <PROGRAM: "program"> }
TOKEN: { <THEN: "then"> }
TOKEN: { <VAR: "var"> }
TOKEN: { <WHILE: "while"> }
```



```
/**
 * Simbolos
 */
TOKEN: { <GREATER: "<"> }
TOKEN: { <GREATEREQ: "<="> }
TOKEN: { <NOTEQUAL: "<>"> }
TOKEN: { <EQUAL: "="> }
TOKEN: { <LESS: ">"> }
TOKEN: { <LESSEQ: ">="> }
TOKEN: { <MINUS: "-"> }
TOKEN: { <COMMA: ","> }
TOKEN: { <SEMICOLON: ";"> }
TOKEN: { <COLON: ":"> }
TOKEN: { <ASSIGNMENT: ":="> }
TOKEN: { <DOT: "."> }
TOKEN: { <LPAREN: "("> }
TOKEN: { <RPAREN: ")"> }
TOKEN: { <STAR: "*"> }
TOKEN: { <PLUS: "+"> }


/**
 * Tokens auxiliares
 */
TOKEN: { <#LETTER: ["a"-"z"] | ["A"-"Z"]>}
TOKEN: { <#DIGIT: ["0"-"9"]> }
TOKEN: { <#UNDERLINE: "_"> }
```



```
/**
 * Inteiros
 */
TOKEN: { <NUMBER: (<DIGIT>)+> }

/**
 * Identificadores
 */
TOKEN: { <ID: (<LETTER> | <UNDERLINE>) (<LETTER> | <DIGIT> | <UNDERLINE>)* > }
```

3. Especificação do parser em EBNF

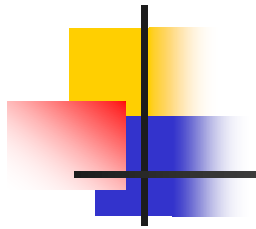
Como nossa tarefa é criar um Analisador Léxico que tabula a saída código/token para PS com as extensões do grupo,

a parte do parser será composta somente das várias opções de tokens do PS com a impressão.

- Token is a class representing tokens. Each Token object has an integer field kind that represents the kind of the token (PLUS, NUMBER, or EOF) and a String field image, which represents the sequence of characters from the input file that the token represents.

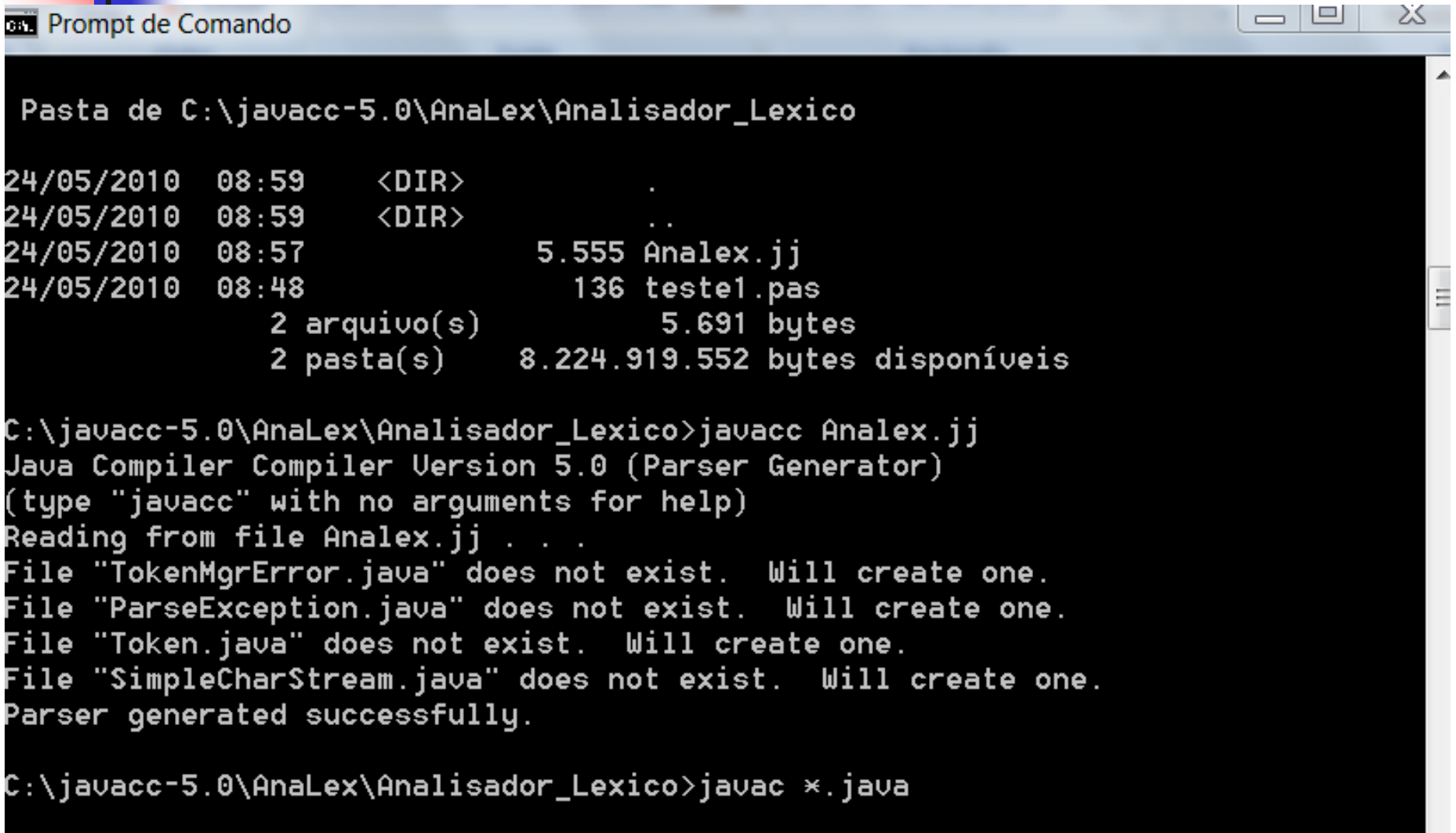
```
void parser() throws TokenMgrError, ParseException :
{
        Token t;
}
{
                        t=<AND>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tAND
"+t.toString() ); }
                      | t=<BEGIN>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tBEGIN
"+t.toString() ); }
                      | t=<DIV>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tDIV
"+t.toString() ); }
                      | t=<DO>                  { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tDO
"+t.toString() ); }
                      | t=<ELSE>                { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tELSE
"+t.toString() ); }
                      | t=<END>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tEND
"+t.toString() ); }
                      | t=<IF>                  { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tIF
"+t.toString() ); }
                      | t=<NOT>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tNOT
"+t.toString() ); }
                      | t=<OR>                  { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tOR
"+t.toString() ); }
                      | t=<PROCEDURE>    { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tPROCEDURE    -
"+t.toString() ); }
                      | t=<PROGRAM>      { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tPROGRAM      -
"+t.toString() ); }
                      | t=<THEN>                { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tTHEN
"+t.toString() ); }
                      | t=<VAR>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tVAR
"+t.toString() ); }
                      | t=<WHILE>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tWHILE
"+t.toString() ); }
```

```
                /**
                 * Simbolos
                 */
                |       t=<GREATER>     { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tGREATER     -
"+t.toString() ); }
                |       t=<GREATEREQ>   { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tGREATEREQ   -
"+t.toString() ); }
                |       t=<NOTEQUAL>    { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tNOTEQUAL    -
"+t.toString() ); }
                |       t=<EQUAL>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tEQUAL       -
"+t.toString() ); }
                |       t=<LESS>                { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tLESS       -
"+t.toString() ); }
                |       t=<LESSEQ>              { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tLESSEQ     -
"+t.toString() ); }
                |       t=<MINUS>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tMINUS      -
"+t.toString() ); }
                |       t=<COMMA>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tCOMMA      -
"+t.toString() ); }
                |       t=<SEMICOLON>   { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tSEMICOLON   -
"+t.toString() ); }
                |       t=<COLON>               { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tCOLON      -
"+t.toString() ); }
                |       t=<ASSIGNMENT>  { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tASSIGNMENT  -
"+t.toString() ); }
                |       t=<DOT>                 { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tDOT       -
"+t.toString() ); }|
                |       t=<LPAREN>              { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tLPAREN     -
"+t.toString() ); }
                |       t=<RPAREN>              { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tRPAREN     -
"+t.toString() ); }
                |       t=<STAR>                { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tSTAR       -
"+t.toString() ); }
                |       t=<PLUS>                { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tPLUS       -
"+t.toString() ); }
```



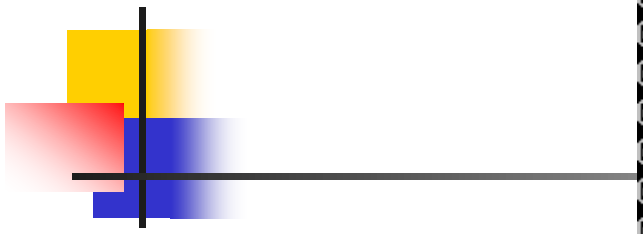
```
                /**
                 * Inteiros
                 */
                |       t=<NUMBER>              { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tNUMBER
"+t.toString() ); }
                /**
                 * Identificadores
                 */
                |       t=<ID>                  { System.out.println("@("+t.beginLine+","+t.beginColumn+")\tID
"+t.toString() ); }

                |       <EOF>                   { System.exit(0); }
}
```

Javacc e compilar os arq resultantes



Programa Teste – correto

```
program teste1;
var x: integer;
begin
        x := 3;
        if x > 5 then
    begin
                x := 5
        end else
    begin
                x := 0
        end
end.
```



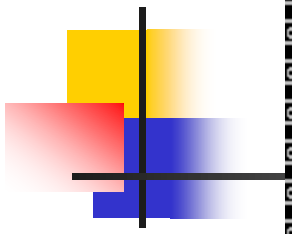
Saída AnaLex, programa correto

Prompt de Comando

```
C:\javacc-5.0\AnaLex\Analisador_Lexico>java AnaLex < teste1.pas
@(1,1)  PROGRAM     - program
@(1,9)  ID          - teste1
@(1,15) SEMICOLON   - ;
@(2,1)  VAR         - var
@(2,5)  ID          - x
@(2,6)  COLON       - :
@(2,8)  ID          - integer
@(2,15) SEMICOLON   - ;
@(3,1)  BEGIN       - begin
@(4,9)  ID          - x
@(4,11) ASSIGNMENT  - :=
@(4,14) NUMBER      - 3
@(4,15) SEMICOLON   - ;
@(5,9)  IF          - if
@(5,12) ID          - x
@(5,14) LESS        - >
@(5,16) NUMBER      - 5
@(5,18) THEN        - then
@(6,6)  BEGIN       - begin
@(7,17) ID          - x
@(7,19) ASSIGNMENT  - :=
@(7,22) NUMBER      - 5
@(8,9)  END         - end
@(8,13) ELSE        - else
@(9,6)  BEGIN       - begin
@(10,17)        ID          - x
@(10,19)        ASSIGNMENT  - :=
@(10,22)        NUMBER      - 0
@(11,9) END         - end
@(12,1) END         - end
@(12,4) DOT         - .

C:\javacc-5.0\AnaLex\Analisador_Lexico>
```

Programa teste – abriu comentário e não fechou

```
program teste1;
var x: integer;
begin
      x := 3;
      if x > 5 then
     begin
           x := 5
      end else
     begin
           x := 0
      end
{     |
end.
```



Saída AnaLex, programa com erro

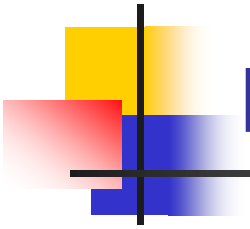
```
C:\javacc-5.0\AnaLex\Analisador_Lexico>java AnaLex < teste2.pas
@(1,1)  PROGRAM     - program
@(1,9)  ID          - teste1
@(1,15) SEMICOLON   - ;
@(2,1)  VAR         - var
@(2,5)  ID          - x
@(2,6)  COLON       - :
@(2,8)  ID          - integer
@(2,15) SEMICOLON   - ;
@(3,1)  BEGIN       - begin
@(4,9)  ID          - x
@(4,11) ASSIGNMENT  - :=
@(4,14) NUMBER      - 3
@(4,15) SEMICOLON   - ;
@(5,9)  IF          - if
@(5,12) ID          - x
@(5,14) LESS        - >
@(5,16) NUMBER      - 5
@(5,18) THEN        - then
@(6,6)  BEGIN       - begin
@(7,17) ID          - x
@(7,19) ASSIGNMENT  - :=
@(7,22) NUMBER      - 5
@(8,9)  END         - end
@(8,13) ELSE        - else
@(9,6)  BEGIN       - begin
@(10,17)        ID          - x
@(10,19)        ASSIGNMENT  - :=
@(10,22)        NUMBER      - 0
@(11,9) END         - end
Lexical error at line 14, column 0.  Encountered: <EOF> after : "{    \r\nend.\r
\n"

C:\javacc-5.0\AnaLex\Analisador_Lexico>
```

Outro programa com erro

```
C:\javacc-5.0\AnaLex\Analisador_Lexico>java AnaLex < teste2.pas
@(1,1)  PROGRAM     - program
@(1,9)  ID          - teste1
@(1,15) SEMICOLON   - ;
@(2,1)  VAR         - var
@(2,5)  ID          - x
@(2,6)  COLON       - :
@(2,8)  ID          - integer
@(2,15) SEMICOLON   - ;
@(3,1)  BEGIN       - begin
Lexical error at line 4, column 1.  Encountered: "$" (36), after : ""

C:\javacc-5.0\AnaLex\Analisador_Lexico>
```

```
program teste1;
var x: integer;
begin
$
      x := 3;
      if x > 5 then
     begin
           x := 5
      end else
     begin
           x := 0
      end
end.
```

O ideal seria customizar as mensagens de erros



- Traduzir seria uma opção