CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO

ANA CLARA CUNHA LOPES
GUSTAVO DE ASSIS XAVIER

Compiladores

Trabalho parte 1 - Analisador Léxico

Belo Horizonte - MG
27 de maio de 2025

1. Introdução

Este relatório apresenta o desenvolvimento de um analisador léxico de um compilador para uma linguagem de programação específica, implementado como parte de um trabalho acadêmico. O objetivo é descrever como o analisador léxico funciona, a abordagem utilizada na implementação, destacando as principais classes e seus propósitos, e os resultados dos testes realizados, incluindo os programas-fontes analisados e as saídas do compilador.

2. Forma de Uso do Compilador

O analisador léxico do compilador foi desenvolvido em Java e executado em um ambiente de desenvolvimento integrado (VS-Code). A seguir, descrevemos os passos para executá-lo:

1. **Preparação do programa-fonte:** o usuário deve criar um arquivo de texto (testeX.txt) contendo o código-fonte da linguagem especificada. Este arquivo pode ser salvo na pasta testes, junto aos demais arquivos.
2. **Compilação do projeto:** Com o terminal dentro da pasta **compilador,** faça:

javac analisadorlexico/*.java interpretador/value/*.java *.java

3. **Execução do compilador:** após compilar, para executar o programa basta ir com o terminal para pasta COMPILADORES e executar:

java compilador.exe

Há o passo a passo também no README, nos arquivos.

4. **Interpretação da saída:** a saída do compilador lista os tokens identificados no formato Token(TIPO, "valor", linha=N), onde TIPO é o tipo do token (ex.: PROGRAM, IDENTIFIER), valor é o lexema identificado e N é a linha onde o lexema foi encontrado. Linhas vazias são ignoradas na contagem lógica. Se houver erro, uma mensagem acima do problema.

3. Abordagem Utilizada na Implementação

A implementação do compilador foi realizada em Java, focando inicialmente na etapa de análise léxica, que é responsável por transformar o código-fonte em uma sequência de tokens. A abordagem utilizada é baseada em um analisador léxico determinístico, que processa o código caractere por caractere, identificando tokens com base em regras predefinidas para a linguagem.

4. Principais Classes da Aplicação

A aplicação é composta por três "classes", cada uma com um propósito específico:

- **exe:** esta classe contém o método main, sendo o ponto de entrada da aplicação. É responsável por iniciar o compilador e coordenar a execução do processo de análise léxica. Ela instancia a classe AnalisadorLexico, passa o conteúdo do arquivo de entrada e exibe a saída gerada (tokens ou mensagens de erro) no console.
- **LexicalAnalysis:** esta classe é o núcleo do compilador, responsável pela análise léxica do código-fonte. Ela lê o arquivo de entrada, processa o texto caractere por caractere e gera uma sequência de tokens. Possui métodos para:
    - Pular espaços em branco e quebras de linha, ajustando a contagem de linhas (ignorando linhas vazias na contagem lógica).
    - Identificar palavras-chave, identificadores, constantes, literais, operadores e símbolos.
    - Gerar tokens com informações sobre tipo, valor e linha.
    - Reportar erros léxicos, como caracteres inválidos ou literais mal formados.

A classe utiliza um mapa de palavras reservadas para associar palavras-chave a tipos de tokens específicos.

- **Token:** esta classe representa um token gerado pelo analisador léxico. Cada token possui quatro atributos: o tipo (um enum que define categorias como PROGRAM, IDENTIFIER, SEMICOLON, etc.), o valor (que é representado por uma determinada classe derivada da classe Value, pois isso ajudará nas próximas implementações) o

lexema (o “texto” correspondente no código-fonte) e a linha onde foi encontrado. A classe é usada para estruturar a saída do analisador léxico e facilita a integração com futuras etapas do compilador (como a análise sintática).

- **Value<?> e seus filhos:** Considerando a disciplina de LP, essa classe pode nos ajudar a padronizar valores e fazer conversões, facilitando a implementação dos demais analisadores.

5. Detalhes da Implementação

A análise léxica foi implementada utilizando uma abordagem baseada em máquina de estados implícita, onde o método nextToken() da classe LexicalAnalysis lê caracteres consecutivamente e decide qual token gerar com base em padrões predefinidos:

- Palavras-chave e identificadores são identificados verificando se o caractere inicial é uma letra ou underline, seguido por letras, dígitos ou underline.
- Constantes numéricas são processadas verificando dígitos e, no caso de números reais, a presença de um ponto decimal seguido por mais dígitos.
- Constantes de caracteres e literais são identificados por aspas simples e duplas, respectivamente.
- Símbolos e operadores (como: +, *, - , etc.) são tratados por um switch que verifica o caractere atual e, em alguns casos, o próximo caractere, como por exemplo: ==, >=, etc.).

Os casos no geral são dados pelo diagrama (feito do Draw.io):

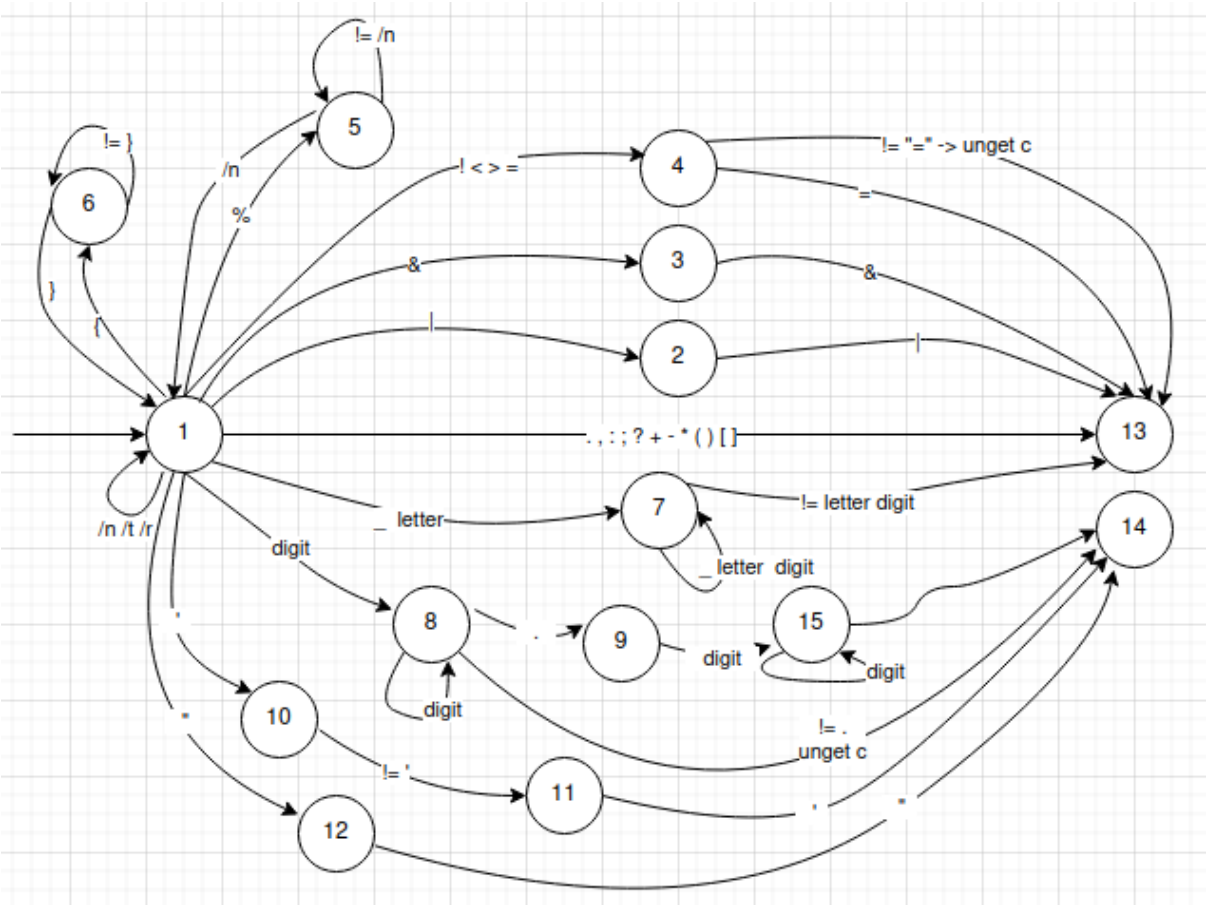

Diagrama de Estados do Analisador Léxico.

6. Resultados dos Testes

Foram realizados testes com diferentes programas-fonte para verificar o funcionamento do compilador. Abaixo, apresentamos o programa-fonte analisado e a saída correspondente. Foi considerado que todos erros são fatais.

6.1. Teste 1:

Programa-Fonte (teste1.txt):

```
program
  int: a,b,c;
  float: result;
  char: ch;

begin
  out("Digite o valor de a:");
  in (a);
  out("Digite o valor de c:");
  read (ch);
  b = 10;
  result = (a * ch)/(b  5 - 345.27);
  out("O resultado e: ");
  out(result);
  result = result + ch;
end
```

Saída do Compilador:

Analisando: teste1.txt

```
Token: PROGRAM          Lexema: program Linha: 1
Token: INTEGER_CONST    Lexema: int     Linha: 2
Token: COLON            Lexema: :       Linha: 2
Token: IDENTIFIER       Lexema: a       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: b       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: c       Linha: 2
Token: SEMICOLON        Lexema: ;       Linha: 2
Token: FLOAT_CONST      Lexema: float   Linha: 3
Token: COLON            Lexema: :       Linha: 3
Token: IDENTIFIER       Lexema: result  Linha: 3
Token: SEMICOLON        Lexema: ;       Linha: 3
Token: CHAR_CONST       Lexema: char    Linha: 4
Token: COLON            Lexema: :       Linha: 4
Token: IDENTIFIER       Lexema: ch      Linha: 4
Token: SEMICOLON        Lexema: ;       Linha: 4
Token: BEGIN            Lexema: begin   Linha: 5
Token: OUT              Lexema: out     Linha: 6
Token: OPEN_PAR         Lexema: (       Linha: 6
Token: LITERAL          Lexema: Digite o valor de a: Linha: 6
Token: CLOSE_PAR        Lexema: )       Linha: 6
Token: SEMICOLON        Lexema: ;       Linha: 6
Token: IN          Lexema: in       Linha: 7
Token: OPEN_PAR         Lexema: (       Linha: 7
Token: IDENTIFIER       Lexema: a       Linha: 7
Token: CLOSE_PAR        Lexema: )       Linha: 7
Token: SEMICOLON        Lexema: ;       Linha: 7
Token: OUT              Lexema: out     Linha: 8
```

```
Token: OPEN_PAR         Lexema: (           Linha: 8
Token: LITERAL          Lexema: Digite o valor de c: Linha: 8
Token: CLOSE_PAR        Lexema: )           Linha: 8
Token: SEMICOLON        Lexema: ;           Linha: 8
Token: READ             Lexema: read        Linha: 9
Token: OPEN_PAR         Lexema: (           Linha: 9
Token: IDENTIFIER       Lexema: ch          Linha: 9
Token: CLOSE_PAR        Lexema: )           Linha: 9
Token: SEMICOLON        Lexema: ;           Linha: 9
Token: IDENTIFIER       Lexema: b           Linha: 10
Token: ASSIGN           Lexema: =           Linha: 10
Token: INTEGER_CONST    Lexema: 10          Linha: 10
Token: SEMICOLON        Lexema: ;           Linha: 10
Token: IDENTIFIER       Lexema: result      Linha: 11
Token: ASSIGN           Lexema: =           Linha: 11
Token: OPEN_PAR         Lexema: (           Linha: 11
Token: IDENTIFIER       Lexema: a           Linha: 11
Token: MUL              Lexema: *           Linha: 11
Token: IDENTIFIER       Lexema: ch          Linha: 11
Token: CLOSE_PAR        Lexema: )           Linha: 11
Token: DIV          Lexema: /           Linha: 11
Token: OPEN_PAR         Lexema: (           Linha: 11
Token: IDENTIFIER       Lexema: b           Linha: 11
Token: INTEGER_CONST    Lexema: 5           Linha: 11
Token: SUB              Lexema: -           Linha: 11
Token: FLOAT_CONST      Lexema: 345.27      Linha: 11
Token: CLOSE_PAR        Lexema: )           Linha: 11
Token: SEMICOLON        Lexema: ;           Linha: 11
Token: OUT              Lexema: out         Linha: 12
Token: OPEN_PAR         Lexema: (           Linha: 12
Token: LITERAL          Lexema: O resultado e:  Linha: 12
Token: CLOSE_PAR        Lexema: )           Linha: 12
Token: SEMICOLON        Lexema: ;           Linha: 12
Token: OUT              Lexema: out         Linha: 13
Token: OPEN_PAR         Lexema: (           Linha: 13
Token: IDENTIFIER       Lexema: result      Linha: 13
Token: CLOSE_PAR        Lexema: )           Linha: 13
Token: SEMICOLON        Lexema: ;           Linha: 13
Token: IDENTIFIER       Lexema: result      Linha: 14
Token: ASSIGN           Lexema: =           Linha: 14
Token: IDENTIFIER       Lexema: result      Linha: 14
Token: ADD              Lexema: +           Linha: 14
Token: IDENTIFIER       Lexema: ch          Linha: 14
Token: SEMICOLON        Lexema: ;           Linha: 14
Token: END              Lexema: end         Linha: 15
Token: END_OF_FILE      Lexema:             Linha: 15
```

Como podemos ver, a execução do código foi um sucesso. O compilador gerou todos os tokens corretamente até o final EOF (final de arquivo).

6.2. Teste 2:

Programa-Fonte (teste2.txt):

```
program

     float: raio, area$ = 0.0;

begin
repeat
   in(raio);
   char: resposta;
   if (raio > 0.0) then
     area =  3. * raio * raio;
     out (area);
   end;
   out ("Deseja continuar?);
   in (resp);

until (resp == 'N' || resp == 'n');

end
```

Saída do Compilador:

```
Analisando: teste2.txt
Token: PROGRAM        Lexema: program Linha: 1
Token: FLOAT_CONST    Lexema: float   Linha: 2
Token: COLON          Lexema: :       Linha: 2
Token: IDENTIFIER     Lexema: raio    Linha: 2
Token: COMMA          Lexema: ,       Linha: 2
Token: IDENTIFIER     Lexema: area    Linha: 2
Erro ao analisar léxico: Na linha 2->Caracter não permitido.
```

Como podemos ver, a execução do código apresentou erro léxico na linha 2 do código fonte (float: raio, area$ = 0.0;). O caractere “$” não foi identificado na lista de tokens da gramática, isso porque esse caractere não faz parte da linguagem especificada. Ajustando esse erro e executando o código novamente, temos a seguinte saída do compilador:

```
Token: PROGRAM        Lexema: program Linha: 1
Token: FLOAT_CONST    Lexema: float   Linha: 2
Token: COLON          Lexema: :       Linha: 2
Token: IDENTIFIER     Lexema: raio    Linha: 2
Token: COMMA          Lexema: ,       Linha: 2
Token: IDENTIFIER     Lexema: area    Linha: 2
Token: ASSIGN         Lexema: =       Linha: 2
```

```
Token: FLOAT_CONST   Lexema: 0.0       Linha: 2
Token: SEMICOLON     Lexema: ;         Linha: 2
Token: BEGIN         Lexema: begin     Linha: 3
Token: REPEAT        Lexema: repeat    Linha: 4
Token: IN        Lexema: in        Linha: 5
Token: OPEN_PAR      Lexema: (         Linha: 5
Token: IDENTIFIER    Lexema: raio      Linha: 5
Token: CLOSE_PAR     Lexema: )         Linha: 5
Token: SEMICOLON     Lexema: ;         Linha: 5
Token: CHAR_CONST    Lexema: char      Linha: 6
Token: COLON         Lexema: :         Linha: 6
Token: IDENTIFIER    Lexema: resposta  Linha: 6
Token: SEMICOLON     Lexema: ;         Linha: 6
Token: IF        Lexema: if        Linha: 7
Token: OPEN_PAR      Lexema: (         Linha: 7
Token: IDENTIFIER    Lexema: raio      Linha: 7
Token: GREATER_THAN  Lexema: >         Linha: 7
Token: FLOAT_CONST   Lexema: 0.0       Linha: 7
Token: CLOSE_PAR     Lexema: )         Linha: 7
Token: THEN          Lexema: then      Linha: 7
Token: IDENTIFIER    Lexema: area      Linha: 8
Token: ASSIGN        Lexema: =         Linha: 8
Erro ao analisar léxico: Na linha 8->Float inválido.
```

Ao executar novamente, podemos perceber que gerou outro erro léxico na linha 8 (float inválido). Isso porque no código de entrada a linha (area = 3. * raio * raio;) contém um float incompleto. Ajustando esse erro e executando novamente, temos:

```
Analisando: teste2.txt
Token: PROGRAM       Lexema: program   Linha: 1
Token: FLOAT_CONST   Lexema: float     Linha: 2
Token: COLON         Lexema: :         Linha: 2
Token: IDENTIFIER    Lexema: raio      Linha: 2
Token: COMMA         Lexema: ,         Linha: 2
Token: IDENTIFIER    Lexema: area      Linha: 2
Token: ASSIGN        Lexema: =         Linha: 2
Token: FLOAT_CONST   Lexema: 0.0       Linha: 2
Token: SEMICOLON     Lexema: ;         Linha: 2
Token: BEGIN         Lexema: begin     Linha: 3
Token: REPEAT        Lexema: repeat    Linha: 4
Token: IN        Lexema: in        Linha: 5
Token: OPEN_PAR      Lexema: (         Linha: 5
Token: IDENTIFIER    Lexema: raio      Linha: 5
Token: CLOSE_PAR     Lexema: )         Linha: 5
Token: SEMICOLON     Lexema: ;         Linha: 5
Token: CHAR_CONST    Lexema: char      Linha: 6
Token: COLON         Lexema: :         Linha: 6
Token: IDENTIFIER    Lexema: resposta  Linha: 6
Token: SEMICOLON     Lexema: ;         Linha: 6
Token: IF        Lexema: if        Linha: 7
Token: OPEN_PAR      Lexema: (         Linha: 7
Token: IDENTIFIER    Lexema: raio      Linha: 7
Token: GREATER_THAN  Lexema: >         Linha: 7
Token: FLOAT_CONST   Lexema: 0.0       Linha: 7
Token: CLOSE_PAR     Lexema: )         Linha: 7
Token: THEN          Lexema: then      Linha: 7
Token: IDENTIFIER    Lexema: area      Linha: 8
Token: ASSIGN        Lexema: =         Linha: 8
```

```
Token: FLOAT_CONST   Lexema: 3.0       Linha: 8
Token: MUL           Lexema: *         Linha: 8
Token: IDENTIFIER    Lexema: raio      Linha: 8
Token: MUL           Lexema: *         Linha: 8
Token: IDENTIFIER    Lexema: raio      Linha: 8
Token: SEMICOLON     Lexema: ;         Linha: 8
Token: OUT           Lexema: out       Linha: 9
Token: OPEN_PAR      Lexema: (         Linha: 9
Token: IDENTIFIER    Lexema: area      Linha: 9
Token: CLOSE_PAR     Lexema: )         Linha: 9
Token: SEMICOLON     Lexema: ;         Linha: 9
Token: END           Lexema: end       Linha: 10
Token: SEMICOLON     Lexema: ;         Linha: 10
Token: OUT           Lexema: out       Linha: 11
Token: OPEN_PAR      Lexema: (         Linha: 11
Erro ao analisar léxico: Na linha 11->Erro Lexico: aspas duplas esperadas na linha
```

Podemos perceber que gerou mais um erro. Esse erro foi devido a falta de fechamento das aspas duplas na linha 11 (out ("Deseja continuar?);). Ajustando esse erro e executando novamente o programa fonte, obtemos uma compilação sem erros, como podemos ver a seguir.

Saída do Compilador:

```
Analisando: teste2.txt
Token: PROGRAM       Lexema: program   Linha: 1
Token: FLOAT_CONST   Lexema: float     Linha: 2
Token: COLON         Lexema: :         Linha: 2
Token: IDENTIFIER    Lexema: raio      Linha: 2
Token: COMMA         Lexema: ,         Linha: 2
Token: IDENTIFIER    Lexema: area      Linha: 2
Token: ASSIGN        Lexema: =         Linha: 2
Token: FLOAT_CONST   Lexema: 0.0       Linha: 2
Token: SEMICOLON     Lexema: ;         Linha: 2
Token: BEGIN         Lexema: begin     Linha: 3
Token: REPEAT        Lexema: repeat    Linha: 4
Token: IN          Lexema: in        Linha: 5
Token: OPEN_PAR      Lexema: (         Linha: 5
Token: IDENTIFIER    Lexema: raio      Linha: 5
Token: CLOSE_PAR     Lexema: )         Linha: 5
Token: SEMICOLON     Lexema: ;         Linha: 5
Token: CHAR_CONST    Lexema: char      Linha: 6
Token: COLON         Lexema: :         Linha: 6
Token: IDENTIFIER    Lexema: resposta  Linha: 6
Token: SEMICOLON     Lexema: ;         Linha: 6
Token: IF          Lexema: if        Linha: 7
Token: OPEN_PAR      Lexema: (         Linha: 7
Token: IDENTIFIER    Lexema: raio      Linha: 7
Token: GREATER_THAN  Lexema: >         Linha: 7
Token: FLOAT_CONST   Lexema: 0.0       Linha: 7
Token: CLOSE_PAR     Lexema: )         Linha: 7
Token: THEN          Lexema: then      Linha: 7
Token: IDENTIFIER    Lexema: area      Linha: 8
Token: ASSIGN        Lexema: =         Linha: 8
Token: FLOAT_CONST   Lexema: 3.0       Linha: 8
Token: MUL           Lexema: *         Linha: 8
```

Token: IDENTIFIER Lexema: raio Linha: 8
Token: MUL Lexema: * Linha: 8
Token: IDENTIFIER Lexema: raio Linha: 8
Token: SEMICOLON Lexema: ; Linha: 8
Token: OUT Lexema: out Linha: 9
Token: OPEN_PAR Lexema: (Linha: 9
Token: IDENTIFIER Lexema: area Linha: 9
Token: CLOSE_PAR Lexema:) Linha: 9
Token: SEMICOLON Lexema: ; Linha: 9
Token: END Lexema: end Linha: 10
Token: SEMICOLON Lexema: ; Linha: 10
Token: OUT Lexema: out Linha: 11
Token: OPEN_PAR Lexema: (Linha: 11
Token: LITERAL Lexema: Deseja continuar? Linha: 11
Token: CLOSE_PAR Lexema:) Linha: 11
Token: SEMICOLON Lexema: ; Linha: 11
Token: IN Lexema: in Linha: 12
Token: OPEN_PAR Lexema: (Linha: 12
Token: IDENTIFIER Lexema: resp Linha: 12
Token: CLOSE_PAR Lexema:) Linha: 12
Token: SEMICOLON Lexema: ; Linha: 12
Token: UNTIL Lexema: until Linha: 13
Token: OPEN_PAR Lexema: (Linha: 13
Token: IDENTIFIER Lexema: resp Linha: 13
Token: EQUALS Lexema: == Linha: 13
Token: CHAR_CONST Lexema: N Linha: 13
Token: OR Lexema: || Linha: 13
Token: IDENTIFIER Lexema: resp Linha: 13
Token: EQUALS Lexema: == Linha: 13
Token: CHAR_CONST Lexema: n Linha: 13
Token: CLOSE_PAR Lexema:) Linha: 13
Token: SEMICOLON Lexema: ; Linha: 13
Token: END Lexema: end Linha: 14
Token: END_OF_FILE Lexema: Linha: 14

6.3. Teste 3:

Programa-Fonte (teste3.txt):

```
program

   int: a, b, aux;

   begin
      in (a);
      in(b);
      if (a>b) then
       int aux;
       aux = b;
       b = a;
       a = aux
      end;
      out(a;
      out(b)
   end
```

Saída do Compilador:

```
Analisando: teste3.txt
Token: PROGRAM          Lexema: program Linha: 1
Token: INTEGER_CONST    Lexema: int     Linha: 2
Token: COLON            Lexema: :       Linha: 2
Token: IDENTIFIER       Lexema: a       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: b       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: aux     Linha: 2
Token: SEMICOLON        Lexema: ;       Linha: 2
Token: BEGIN            Lexema: begin   Linha: 3
Token: IN          Lexema: in        Linha: 4
Token: OPEN_PAR         Lexema: (       Linha: 4
Token: IDENTIFIER       Lexema: a       Linha: 4
Token: CLOSE_PAR        Lexema: )       Linha: 4
Token: SEMICOLON        Lexema: ;       Linha: 4
Token: IN          Lexema: in        Linha: 5
Token: OPEN_PAR         Lexema: (       Linha: 5
Token: IDENTIFIER       Lexema: b       Linha: 5
Token: CLOSE_PAR        Lexema: )       Linha: 5
Token: SEMICOLON        Lexema: ;       Linha: 5
Token: IF          Lexema: if        Linha: 6
Token: OPEN_PAR         Lexema: (       Linha: 6
Token: IDENTIFIER       Lexema: a       Linha: 6
Token: GREATER_THAN     Lexema: >       Linha: 6
Token: IDENTIFIER       Lexema: b       Linha: 6
Token: CLOSE_PAR        Lexema: )       Linha: 6
Token: THEN             Lexema: then    Linha: 6
Token: INTEGER_CONST    Lexema: int     Linha: 7
Token: IDENTIFIER       Lexema: aux     Linha: 7
Token: SEMICOLON        Lexema: ;       Linha: 7
Token: IDENTIFIER       Lexema: aux     Linha: 8
```

```
Token: ASSIGN          Lexema: =       Linha: 8
Token: IDENTIFIER      Lexema: b       Linha: 8
Token: SEMICOLON       Lexema: ;       Linha: 8
Token: IDENTIFIER      Lexema: b       Linha: 9
Token: ASSIGN          Lexema: =       Linha: 9
Token: IDENTIFIER      Lexema: a       Linha: 9
Token: SEMICOLON       Lexema: ;       Linha: 9
Token: IDENTIFIER      Lexema: a       Linha: 10
Token: ASSIGN          Lexema: =       Linha: 10
Token: IDENTIFIER      Lexema: aux     Linha: 10
Token: END             Lexema: end     Linha: 11
Token: SEMICOLON       Lexema: ;       Linha: 11
Token: OUT             Lexema: out     Linha: 12
Token: OPEN_PAR        Lexema: (       Linha: 12
Token: IDENTIFIER      Lexema: a       Linha: 12
Token: SEMICOLON       Lexema: ;       Linha: 12
Token: OUT             Lexema: out     Linha: 13
Token: OPEN_PAR        Lexema: (       Linha: 13
Token: IDENTIFIER      Lexema: b       Linha: 13
Token: CLOSE_PAR       Lexema: )       Linha: 13
Token: END             Lexema: end     Linha: 14
Token: END_OF_FILE     Lexema:         Linha: 14
```

Como podemos perceber, o código compilou sem erros léxicos. No programa fonte, a linha 12 não possui o fecha parênteses, porque é responsabilidade do sintático analisar a estrutura gramatical do programa após a geração de tokens e identificar esses problemas. O mesmo vale para a falta de ponto e vírgula na linha 10 e 13.

6.4. Teste 4:

Programa-Fonte (teste4.txt):

```
program
   a, b, c, maior, outro: int;
   begin
   repeat
      out("A");
      in(a);
      out("B");
      in(b);
      out("C");
      in(c);

      %Verifica o maior

      if ( (a>b) and (a>c)  ) end
        maior = a

      else
         if (b>c) then
             maior = b;


         else
             maior = c
         end
      end;
      out("Maior valor: "");
      out (maior);
      out ("Outro? ");
      in(outro);
   until (outro == 0)
   end
```

Saída do Compilador:

```
Analisando: teste4.txt
Token: PROGRAM          Lexema: program Linha: 1
Token: IDENTIFIER       Lexema: a       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: b       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: c       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: maior   Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: outro   Linha: 2
Token: COLON            Lexema: :       Linha: 2
Token: INTEGER_CONST    Lexema: int     Linha: 2
```

Token: SEMICOLON Lexema: ; Linha: 2
Token: BEGIN Lexema: begin Linha: 3
Token: REPEAT Lexema: repeat Linha: 4
Token: OUT Lexema: out Linha: 5
Token: OPEN_PAR Lexema: (Linha: 5
Token: LITERAL Lexema: A Linha: 5
Token: CLOSE_PAR Lexema:) Linha: 5
Token: SEMICOLON Lexema: ; Linha: 5
Token: IN Lexema: in Linha: 6
Token: OPEN_PAR Lexema: (Linha: 6
Token: IDENTIFIER Lexema: a Linha: 6
Token: CLOSE_PAR Lexema:) Linha: 6
Token: SEMICOLON Lexema: ; Linha: 6
Token: OUT Lexema: out Linha: 7
Token: OPEN_PAR Lexema: (Linha: 7
Token: LITERAL Lexema: B Linha: 7
Token: CLOSE_PAR Lexema:) Linha: 7
Token: SEMICOLON Lexema: ; Linha: 7
Token: IN Lexema: in Linha: 8
Token: OPEN_PAR Lexema: (Linha: 8
Token: IDENTIFIER Lexema: b Linha: 8
Token: CLOSE_PAR Lexema:) Linha: 8
Token: SEMICOLON Lexema: ; Linha: 8
Token: OUT Lexema: out Linha: 9
Token: OPEN_PAR Lexema: (Linha: 9
Token: LITERAL Lexema: C Linha: 9
Token: CLOSE_PAR Lexema:) Linha: 9
Token: SEMICOLON Lexema: ; Linha: 9
Token: IN Lexema: in Linha: 10
Token: OPEN_PAR Lexema: (Linha: 10
Token: IDENTIFIER Lexema: c Linha: 10
Token: CLOSE_PAR Lexema:) Linha: 10
Token: SEMICOLON Lexema: ; Linha: 10
Token: IF Lexema: if Linha: 12
Token: OPEN_PAR Lexema: (Linha: 12
Token: OPEN_PAR Lexema: (Linha: 12
Token: IDENTIFIER Lexema: a Linha: 12
Token: GREATER_THAN Lexema: > Linha: 12
Token: IDENTIFIER Lexema: b Linha: 12
Token: CLOSE_PAR Lexema:) Linha: 12
Token: IDENTIFIER Lexema: and Linha: 12
Token: OPEN_PAR Lexema: (Linha: 12
Token: IDENTIFIER Lexema: a Linha: 12
Token: GREATER_THAN Lexema: > Linha: 12
Token: IDENTIFIER Lexema: c Linha: 12
Token: CLOSE_PAR Lexema:) Linha: 12
Token: CLOSE_PAR Lexema:) Linha: 12
Token: END Lexema: end Linha: 12
Token: IDENTIFIER Lexema: maior Linha: 13
Token: ASSIGN Lexema: = Linha: 13
Token: IDENTIFIER Lexema: a Linha: 13
Token: ELSE Lexema: else Linha: 14
Token: IF Lexema: if Linha: 15
Token: OPEN_PAR Lexema: (Linha: 15
Token: IDENTIFIER Lexema: b Linha: 15
Token: GREATER_THAN Lexema: > Linha: 15
Token: IDENTIFIER Lexema: c Linha: 15
Token: CLOSE_PAR Lexema:) Linha: 15
Token: THEN Lexema: then Linha: 15
Token: IDENTIFIER Lexema: maior Linha: 16

```
Token: ASSIGN           Lexema: =          Linha: 16
Token: IDENTIFIER       Lexema: b          Linha: 16
Token: SEMICOLON        Lexema: ;          Linha: 16
Token: ELSE             Lexema: else       Linha: 17
Token: IDENTIFIER       Lexema: maior      Linha: 18
Token: ASSIGN           Lexema: =          Linha: 18
Token: IDENTIFIER       Lexema: c          Linha: 18
Token: END              Lexema: end        Linha: 19
Token: END              Lexema: end        Linha: 20
Token: SEMICOLON        Lexema: ;          Linha: 20
Token: OUT              Lexema: out        Linha: 21
Token: OPEN_PAR         Lexema: (          Linha: 21
Token: LITERAL          Lexema: Maior valor:  Linha: 21
Token: LITERAL          Lexema: );
out (maior);
out ( Linha: 21
Token: IDENTIFIER       Lexema: Outro      Linha: 21
Erro ao analisar léxico: Na linha 21->Caracter não permitido.
```

Como podemos verificar, a saída gerou um erro léxico. Caractere não permitido, isso porque devido as aspas erradas, foi tentada a criação de um token dado por “?”, que não pertence a nossa linguagem, dessa forma, ao apagá-lo, temos:

```
Token: PROGRAM          Lexema: program    Linha: 1
Token: IDENTIFIER       Lexema: a          Linha: 2
Token: COMMA            Lexema: ,          Linha: 2
Token: IDENTIFIER       Lexema: b          Linha: 2
Token: COMMA            Lexema: ,          Linha: 2
Token: IDENTIFIER       Lexema: c          Linha: 2
Token: COMMA            Lexema: ,          Linha: 2
Token: IDENTIFIER       Lexema: maior      Linha: 2
Token: COMMA            Lexema: ,          Linha: 2
Token: IDENTIFIER       Lexema: outro      Linha: 2
Token: COLON            Lexema: :          Linha: 2
Token: INTEGER_CONST    Lexema: int        Linha: 2
Token: SEMICOLON        Lexema: ;          Linha: 2
Token: BEGIN            Lexema: begin      Linha: 3
Token: REPEAT           Lexema: repeat     Linha: 4
Token: OUT              Lexema: out        Linha: 5
Token: OPEN_PAR         Lexema: (          Linha: 5
Token: LITERAL          Lexema: A          Linha: 5
Token: CLOSE_PAR        Lexema: )          Linha: 5
Token: SEMICOLON        Lexema: ;          Linha: 5
Token: IN          Lexema: in         Linha: 6
Token: OPEN_PAR         Lexema: (          Linha: 6
Token: IDENTIFIER       Lexema: a          Linha: 6
Token: CLOSE_PAR        Lexema: )          Linha: 6
Token: SEMICOLON        Lexema: ;          Linha: 6
Token: OUT              Lexema: out        Linha: 7
Token: OPEN_PAR         Lexema: (          Linha: 7
Token: LITERAL          Lexema: B          Linha: 7
Token: CLOSE_PAR        Lexema: )          Linha: 7
Token: SEMICOLON        Lexema: ;          Linha: 7
Token: IN          Lexema: in         Linha: 8
Token: OPEN_PAR         Lexema: (          Linha: 8
Token: IDENTIFIER       Lexema: b          Linha: 8
Token: CLOSE_PAR        Lexema: )          Linha: 8
```

```
Token: SEMICOLON        Lexema: ;          Linha: 8
Token: OUT              Lexema: out        Linha: 9
Token: OPEN_PAR         Lexema: (          Linha: 9
Token: LITERAL          Lexema: C          Linha: 9
Token: CLOSE_PAR        Lexema: )          Linha: 9
Token: SEMICOLON        Lexema: ;          Linha: 9
Token: IN          Lexema: in         Linha: 10
Token: OPEN_PAR         Lexema: (          Linha: 10
Token: IDENTIFIER       Lexema: c          Linha: 10
Token: CLOSE_PAR        Lexema: )          Linha: 10
Token: SEMICOLON        Lexema: ;          Linha: 10
Token: IF          Lexema: if         Linha: 12
Token: OPEN_PAR         Lexema: (          Linha: 12
Token: OPEN_PAR         Lexema: (          Linha: 12
Token: IDENTIFIER       Lexema: a          Linha: 12
Token: GREATER_THANLexema: >          Linha: 12
Token: IDENTIFIER       Lexema: b          Linha: 12
Token: CLOSE_PAR        Lexema: )          Linha: 12
Token: IDENTIFIER       Lexema: and        Linha: 12
Token: OPEN_PAR         Lexema: (          Linha: 12
Token: IDENTIFIER       Lexema: a          Linha: 12
Token: GREATER_THANLexema: >          Linha: 12
Token: IDENTIFIER       Lexema: c          Linha: 12
Token: CLOSE_PAR        Lexema: )          Linha: 12
Token: CLOSE_PAR        Lexema: )          Linha: 12
Token: END              Lexema: end        Linha: 12
Token: IDENTIFIER       Lexema: maior      Linha: 13
Token: ASSIGN           Lexema: =          Linha: 13
Token: IDENTIFIER       Lexema: a          Linha: 13
Token: ELSE             Lexema: else       Linha: 14
Token: IF          Lexema: if         Linha: 15
Token: OPEN_PAR         Lexema: (          Linha: 15
Token: IDENTIFIER       Lexema: b          Linha: 15
Token: GREATER_THANLexema: >          Linha: 15
Token: IDENTIFIER       Lexema: c          Linha: 15
Token: CLOSE_PAR        Lexema: )          Linha: 15
Token: THEN             Lexema: then       Linha: 15
Token: IDENTIFIER       Lexema: maior      Linha: 16
Token: ASSIGN           Lexema: =          Linha: 16
Token: IDENTIFIER       Lexema: b          Linha: 16
Token: SEMICOLON        Lexema: ;          Linha: 16
Token: ELSE             Lexema: else       Linha: 17
Token: IDENTIFIER       Lexema: maior      Linha: 18
Token: ASSIGN           Lexema: =          Linha: 18
Token: IDENTIFIER       Lexema: c          Linha: 18
Token: END              Lexema: end        Linha: 19
Token: END              Lexema: end        Linha: 20
Token: SEMICOLON        Lexema: ;          Linha: 20
Token: OUT              Lexema: out        Linha: 21
Token: OPEN_PAR         Lexema: (          Linha: 21
Token: LITERAL          Lexema: Maior valor:  Linha: 21
Token: LITERAL          Lexema: );
out (maior);
out ( Linha: 21
Token: IDENTIFIER       Lexema: Outro      Linha: 21
Erro ao analisar léxico: Na linha 21->Erro Lexico: aspas duplas esperadas na linha
```

Agora o erro foi devido ao não fechamento das aspas. Colocando todas as aspas do código em seu devido lugar, obtemos:

Analisando: teste4.txt

```
Token: PROGRAM          Lexema: program Linha: 1
Token: IDENTIFIER       Lexema: a       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: b       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: c       Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: maior   Linha: 2
Token: COMMA            Lexema: ,       Linha: 2
Token: IDENTIFIER       Lexema: outro   Linha: 2
Token: COLON            Lexema: :       Linha: 2
Token: INTEGER_CONST    Lexema: int     Linha: 2
Token: SEMICOLON        Lexema: ;       Linha: 2
Token: BEGIN            Lexema: begin   Linha: 3
Token: REPEAT           Lexema: repeat  Linha: 4
Token: OUT              Lexema: out     Linha: 5
Token: OPEN_PAR         Lexema: (       Linha: 5
Token: LITERAL          Lexema: A       Linha: 5
Token: CLOSE_PAR        Lexema: )       Linha: 5
Token: SEMICOLON        Lexema: ;       Linha: 5
Token: IN       Lexema: in      Linha: 6
Token: OPEN_PAR         Lexema: (       Linha: 6
Token: IDENTIFIER       Lexema: a       Linha: 6
Token: CLOSE_PAR        Lexema: )       Linha: 6
Token: SEMICOLON        Lexema: ;       Linha: 6
Token: OUT              Lexema: out     Linha: 7
Token: OPEN_PAR         Lexema: (       Linha: 7
Token: LITERAL          Lexema: B       Linha: 7
Token: CLOSE_PAR        Lexema: )       Linha: 7
Token: SEMICOLON        Lexema: ;       Linha: 7
Token: IN       Lexema: in      Linha: 8
Token: OPEN_PAR         Lexema: (       Linha: 8
Token: IDENTIFIER       Lexema: b       Linha: 8
Token: CLOSE_PAR        Lexema: )       Linha: 8
Token: SEMICOLON        Lexema: ;       Linha: 8
Token: OUT              Lexema: out     Linha: 9
Token: OPEN_PAR         Lexema: (       Linha: 9
Token: LITERAL          Lexema: C       Linha: 9
Token: CLOSE_PAR        Lexema: )       Linha: 9
Token: SEMICOLON        Lexema: ;       Linha: 9
Token: IN       Lexema: in      Linha: 10
Token: OPEN_PAR         Lexema: (       Linha: 10
Token: IDENTIFIER       Lexema: c       Linha: 10
Token: CLOSE_PAR        Lexema: )       Linha: 10
Token: SEMICOLON        Lexema: ;       Linha: 10
Token: IF       Lexema: if      Linha: 12
Token: OPEN_PAR         Lexema: (       Linha: 12
Token: OPEN_PAR         Lexema: (       Linha: 12
Token: IDENTIFIER       Lexema: a       Linha: 12
Token: GREATER_THAN     Lexema: >       Linha: 12
Token: IDENTIFIER       Lexema: b       Linha: 12
Token: CLOSE_PAR        Lexema: )       Linha: 12
Token: IDENTIFIER       Lexema: and     Linha: 12
Token: OPEN_PAR         Lexema: (       Linha: 12
Token: IDENTIFIER       Lexema: a       Linha: 12
Token: GREATER_THAN     Lexema: >       Linha: 12
Token: IDENTIFIER       Lexema: c       Linha: 12
Token: CLOSE_PAR        Lexema: )       Linha: 12
Token: CLOSE_PAR        Lexema: )       Linha: 12
```

```
Token: END                Lexema: end      Linha: 12
Token: IDENTIFIER         Lexema: maior    Linha: 13
Token: ASSIGN             Lexema: =        Linha: 13
Token: IDENTIFIER         Lexema: a        Linha: 13
Token: ELSE               Lexema: else     Linha: 14
Token: IF          Lexema: if         Linha: 15
Token: OPEN_PAR           Lexema: (        Linha: 15
Token: IDENTIFIER         Lexema: b        Linha: 15
Token: GREATER_THANLexema: >        Linha: 15
Token: IDENTIFIER         Lexema: c        Linha: 15
Token: CLOSE_PAR          Lexema: )        Linha: 15
Token: THEN               Lexema: then     Linha: 15
Token: IDENTIFIER         Lexema: maior    Linha: 16
Token: ASSIGN             Lexema: =        Linha: 16
Token: IDENTIFIER         Lexema: b        Linha: 16
Token: SEMICOLON          Lexema: ;        Linha: 16
Token: ELSE               Lexema: else     Linha: 17
Token: IDENTIFIER         Lexema: maior    Linha: 18
Token: ASSIGN             Lexema: =        Linha: 18
Token: IDENTIFIER         Lexema: c        Linha: 18
Token: END                Lexema: end      Linha: 19
Token: END                Lexema: end      Linha: 20
Token: SEMICOLON          Lexema: ;        Linha: 20
Token: OUT                Lexema: out      Linha: 21
Token: OPEN_PAR           Lexema: (        Linha: 21
Token: LITERAL            Lexema: Maior valor: Linha: 21
Token: CLOSE_PAR          Lexema: )        Linha: 21
Token: SEMICOLON          Lexema: ;        Linha: 21
Token: OUT                Lexema: out      Linha: 22
Token: OPEN_PAR           Lexema: (        Linha: 22
Token: IDENTIFIER         Lexema: maior    Linha: 22
Token: CLOSE_PAR          Lexema: )        Linha: 22
Token: SEMICOLON          Lexema: ;        Linha: 22
Token: OUT                Lexema: out      Linha: 23
Token: OPEN_PAR           Lexema: (        Linha: 23
Token: LITERAL            Lexema: Outro    Linha: 23
Token: CLOSE_PAR          Lexema: )        Linha: 23
Token: SEMICOLON          Lexema: ;        Linha: 23
Token: IN          Lexema: in         Linha: 24
Token: OPEN_PAR           Lexema: (        Linha: 24
Token: IDENTIFIER         Lexema: outro    Linha: 24
Token: CLOSE_PAR          Lexema: )        Linha: 24
Token: SEMICOLON          Lexema: ;        Linha: 24
Token: UNTIL              Lexema: until    Linha: 25
Token: OPEN_PAR           Lexema: (        Linha: 25
Token: IDENTIFIER         Lexema: outro    Linha: 25
Token: EQUALS             Lexema: ==       Linha: 25
Token: INTEGER_CONST   Lexema: 0        Linha: 25
Token: CLOSE_PAR          Lexema: )        Linha: 25
Token: END                Lexema: end      Linha: 26
Token: END_OF_FILE        Lexema:          Linha: 26
```

6.5. Teste 5:

Programa-Fonte (teste5.txt):

```
programa

declare
    inteiro: pontuacao, pontuacaoMaxina, disponibilidade;
    char: pontuacaoMinima
begin
  disponibilidade = 'S';
  pontuacaoMinima = 50;
  pontuacaoMaxima = 100;
  out("Pontuacao Candidato: ");
  in(pontuacao);
  out("Disponibilidade Candidato: ");
  in(disponibilidade);

  { Comentario
  grande


  while (pontuacao>0 && (pontuação<=pontuacaoMaxima) do
     int: cont;
     cont = cont + 1;
     if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
        out("Candidato aprovado")
     else
        out("Candidato reprovado")
     end

     out("Pontuacao Candidato: ");
     in(pontuacao);
     out("Disponibilidade
Candidato: ");
     in(disponibilidade);

  end
  out (cont);
end
```

Saída do código:

```
Analisando: teste5.txt
Token: IDENTIFIER       Lexema: programa   Linha: 1
Token: IDENTIFIER       Lexema: declare  Linha: 2
Token: IDENTIFIER       Lexema: inteiro   Linha: 3
Token: COLON            Lexema: :         Linha: 3
Token: IDENTIFIER       Lexema: pontuacao  Linha: 3
Token: COMMA            Lexema: ,         Linha: 3
Token: IDENTIFIER       Lexema: pontuacaoMaxina Linha: 3
Token: COMMA            Lexema: ,         Linha: 3
Token: IDENTIFIER       Lexema: disponibilidade Linha: 3
Token: SEMICOLON        Lexema: ;         Linha: 3
Token: CHAR_CONST       Lexema: char      Linha: 4
Token: COLON            Lexema: :         Linha: 4
Token: IDENTIFIER       Lexema: pontuacaoMinima Linha: 4
Token: BEGIN            Lexema: begin     Linha: 5
Token: IDENTIFIER       Lexema: disponibilidade Linha: 6
```

```
Token: ASSIGN           Lexema: =          Linha: 6
Token: CHAR_CONST       Lexema: S          Linha: 6
Token: SEMICOLON        Lexema: ;          Linha: 6
Token: IDENTIFIER       Lexema: pontuacaoMinima Linha: 7
Token: ASSIGN           Lexema: =          Linha: 7
Token: INTEGER_CONST    Lexema: 50     Linha: 7
Token: SEMICOLON        Lexema: ;          Linha: 7
Token: IDENTIFIER       Lexema: pontuacaoMaxima Linha: 8
Token: ASSIGN           Lexema: =          Linha: 8
Token: INTEGER_CONST    Lexema: 100            Linha: 8
Token: SEMICOLON        Lexema: ;          Linha: 8
Token: OUT              Lexema: out        Linha: 9
Token: OPEN_PAR         Lexema: (          Linha: 9
Token: LITERAL          Lexema: Pontuacao Candidato:  Linha: 9
Token: CLOSE_PAR        Lexema: )          Linha: 9
Token: SEMICOLON        Lexema: ;          Linha: 9
Token: IN           Lexema: in         Linha: 10
Token: OPEN_PAR         Lexema: (          Linha: 10
Token: IDENTIFIER       Lexema: pontuacao  Linha: 10
Token: CLOSE_PAR        Lexema: )          Linha: 10
Token: SEMICOLON        Lexema: ;          Linha: 10
Token: OUT              Lexema: out        Linha: 11
Token: OPEN_PAR         Lexema: (          Linha: 11
Token: LITERAL          Lexema: Disponibilidade Candidato:  Linha: 11
Token: CLOSE_PAR        Lexema: )          Linha: 11
Token: SEMICOLON        Lexema: ;          Linha: 11
Token: IN           Lexema: in         Linha: 12
Token: OPEN_PAR         Lexema: (          Linha: 12
Token: IDENTIFIER       Lexema: disponibilidade Linha: 12
Token: CLOSE_PAR        Lexema: )          Linha: 12
Token: SEMICOLON        Lexema: ;          Linha: 12
Erro ao analisar léxico: Na linha 13->Fim de arquivo inesperado.
```

A saída apresentou um erro léxico, porque houve um abre chaves mas não teve um fecha chaves (referente a comentário em bloco), logo, o analisador léxico gerou o seguinte erro: (Erro Lexico: comentario em bloco nao fechado antes do fim do arquivo). Inserindo um fecha chaves no final do código, obtemos uma saída sem erros:

```
Analisando: teste5.txt
Token: IDENTIFIER       Lexema: programa   Linha: 1
Token: IDENTIFIER       Lexema: declare  Linha: 2
Token: IDENTIFIER       Lexema: inteiro   Linha: 3
Token: COLON            Lexema: :          Linha: 3
Token: IDENTIFIER       Lexema: pontuacao  Linha: 3
Token: COMMA            Lexema: ,          Linha: 3
Token: IDENTIFIER       Lexema: pontuacaoMaxina Linha: 3
Token: COMMA            Lexema: ,          Linha: 3
Token: IDENTIFIER       Lexema: disponibilidade Linha: 3
Token: SEMICOLON        Lexema: ;          Linha: 3
Token: CHAR_CONST       Lexema: char       Linha: 4
Token: COLON            Lexema: :          Linha: 4
Token: IDENTIFIER       Lexema: pontuacaoMinima Linha: 4
Token: BEGIN            Lexema: begin      Linha: 5
Token: IDENTIFIER       Lexema: disponibilidade Linha: 6
Token: ASSIGN           Lexema: =          Linha: 6
Token: CHAR_CONST       Lexema: S          Linha: 6
```

```
Token: SEMICOLON       Lexema: ;           Linha: 6
Token: IDENTIFIER      Lexema: pontuacaoMinima Linha: 7
Token: ASSIGN          Lexema: =           Linha: 7
Token: INTEGER_CONST   Lexema: 50      Linha: 7
Token: SEMICOLON       Lexema: ;           Linha: 7
Token: IDENTIFIER      Lexema: pontuacaoMaxima Linha: 8
Token: ASSIGN          Lexema: =           Linha: 8
Token: INTEGER_CONST   Lexema: 100             Linha: 8
Token: SEMICOLON       Lexema: ;           Linha: 8
Token: OUT             Lexema: out         Linha: 9
Token: OPEN_PAR        Lexema: (           Linha: 9
Token: LITERAL         Lexema: Pontuacao Candidato:  Linha: 9
Token: CLOSE_PAR       Lexema: )           Linha: 9
Token: SEMICOLON       Lexema: ;           Linha: 9
Token: IN          Lexema: in          Linha: 10
Token: OPEN_PAR        Lexema: (           Linha: 10
Token: IDENTIFIER      Lexema: pontuacao  Linha: 10
Token: CLOSE_PAR       Lexema: )           Linha: 10
Token: SEMICOLON       Lexema: ;           Linha: 10
Token: OUT             Lexema: out         Linha: 11
Token: OPEN_PAR        Lexema: (           Linha: 11
Token: LITERAL         Lexema: Disponibilidade Candidato:  Linha: 11
Token: CLOSE_PAR       Lexema: )           Linha: 11
Token: SEMICOLON       Lexema: ;           Linha: 11
Token: IN          Lexema: in          Linha: 12
Token: OPEN_PAR        Lexema: (           Linha: 12
Token: IDENTIFIER      Lexema: disponibilidade Linha: 12
Token: CLOSE_PAR       Lexema: )           Linha: 12
Token: SEMICOLON       Lexema: ;           Linha: 12
Token: END_OF_FILE     Lexema:             Linha: 30
```

6.6. Teste 6:

Programa-Fonte (teste6.txt):

```
program
int: x, y;
float: media;
char: opcao;

% Teste de comentario de linha inicial
begin
  { Teste de comentario em bloco
    sobre o programa
  }
  out("Digite o valor de x: ");
  in(x);
  out("Digite o valor de y: ");
  in(y);

  media := (x + y) / 2.0;
  out("Media dos valores: ");
  out(media);

  if (media >= 5.0) then
    out("Media suficiente!");
  else
    out("Media insuficiente!");
  end;

  out("Digite a opcao (S/N): ");
  in(opcao);
  while (opcao != 'N' & opcao != 'n') do % Erro intencional: uso de & em vez de &&
    out("Digite novos valores para x e y: ");
    in(x);
    in(y);
    media := (x + y) / 2.0;
    out("Nova media: ");
    out(media);
    out("Digite a opcao (S/N): ");
    in(opcao);
  end;
end
```

Saída do Compilador:

Analisando: teste6.txt
Token: PROGRAM Lexema: program Linha: 1
Token: INTEGER_CONST Lexema: int Linha: 2
Token: COLON Lexema: : Linha: 2
Token: IDENTIFIER Lexema: x Linha: 2
Token: COMMA Lexema: , Linha: 2
Token: IDENTIFIER Lexema: y Linha: 2
Token: SEMICOLON Lexema: ; Linha: 2
Token: FLOAT_CONST Lexema: float Linha: 3
Token: COLON Lexema: : Linha: 3
Token: IDENTIFIER Lexema: media Linha: 3
Token: SEMICOLON Lexema: ; Linha: 3
Token: CHAR_CONST Lexema: char Linha: 4
Token: COLON Lexema: : Linha: 4
Token: IDENTIFIER Lexema: opcao Linha: 4
Token: SEMICOLON Lexema: ; Linha: 4
Token: BEGIN Lexema: begin Linha: 7
Token: OUT Lexema: out Linha: 11
Token: OPEN_PAR Lexema: (Linha: 11
Token: LITERAL Lexema: Digite o valor de x: Linha: 11
Token: CLOSE_PAR Lexema:) Linha: 11
Token: SEMICOLON Lexema: ; Linha: 11
Token: IN Lexema: in Linha: 12
Token: OPEN_PAR Lexema: (Linha: 12

```
Token: IDENTIFIER       Lexema: x           Linha: 12
Token: CLOSE_PAR        Lexema: )           Linha: 12
Token: SEMICOLON        Lexema: ;           Linha: 12
Token: OUT              Lexema: out         Linha: 13
Token: OPEN_PAR         Lexema: (           Linha: 13
Token: LITERAL          Lexema: Digite o valor de y:  Linha: 13
Token: CLOSE_PAR        Lexema: )           Linha: 13
Token: SEMICOLON        Lexema: ;           Linha: 13
Token: IN       Lexema: in      Linha: 14
Token: OPEN_PAR         Lexema: (           Linha: 14
Token: IDENTIFIER       Lexema: y           Linha: 14
Token: CLOSE_PAR        Lexema: )           Linha: 14
Token: SEMICOLON        Lexema: ;           Linha: 14
Token: IDENTIFIER       Lexema: media       Linha: 16
Token: COLON            Lexema: :           Linha: 16
Token: ASSIGN           Lexema: =           Linha: 16
Token: OPEN_PAR         Lexema: (           Linha: 16
Token: IDENTIFIER       Lexema: x           Linha: 16
Token: ADD              Lexema: +           Linha: 16
Token: IDENTIFIER       Lexema: y           Linha: 16
Token: CLOSE_PAR        Lexema: )           Linha: 16
Token: DIV      Lexema: /       Linha: 16
Token: FLOAT_CONST      Lexema: 2.0         Linha: 16
Token: SEMICOLON        Lexema: ;           Linha: 16
Token: OUT              Lexema: out         Linha: 17
Token: OPEN_PAR         Lexema: (           Linha: 17
Token: LITERAL          Lexema: Media dos valores:  Linha: 17
Token: CLOSE_PAR        Lexema: )           Linha: 17
Token: SEMICOLON        Lexema: ;           Linha: 17
Token: OUT              Lexema: out         Linha: 18
Token: OPEN_PAR         Lexema: (           Linha: 18
Token: IDENTIFIER       Lexema: media       Linha: 18
Token: CLOSE_PAR        Lexema: )           Linha: 18
Token: SEMICOLON        Lexema: ;           Linha: 18
Token: IF       Lexema: if      Linha: 20
Token: OPEN_PAR         Lexema: (           Linha: 20
Token: IDENTIFIER       Lexema: media       Linha: 20
Token: GREATER_EQUAL    Lexema: >=          Linha: 20
Token: FLOAT_CONST      Lexema: 5.0         Linha: 20
Token: CLOSE_PAR        Lexema: )           Linha: 20
Token: THEN             Lexema: then        Linha: 20
Token: OUT              Lexema: out         Linha: 21
Token: OPEN_PAR         Lexema: (           Linha: 21
Token: LITERAL          Lexema: Media suficiente! Linha: 21
Token: CLOSE_PAR        Lexema: )           Linha: 21
Token: SEMICOLON        Lexema: ;           Linha: 21
Token: ELSE             Lexema: else        Linha: 22
Token: OUT              Lexema: out         Linha: 23
Token: OPEN_PAR         Lexema: (           Linha: 23
Token: LITERAL          Lexema: Media insuficiente! Linha: 23
Token: CLOSE_PAR        Lexema: )           Linha: 23
Token: SEMICOLON        Lexema: ;           Linha: 23
Token: END              Lexema: end         Linha: 24
Token: SEMICOLON        Lexema: ;           Linha: 24
Token: OUT              Lexema: out         Linha: 26
Token: OPEN_PAR         Lexema: (           Linha: 26
Token: LITERAL          Lexema: Digite a opcao (S/N):  Linha: 26
Token: CLOSE_PAR        Lexema: )           Linha: 26
Token: SEMICOLON        Lexema: ;           Linha: 26
Token: IN       Lexema: in      Linha: 27
```

```
Token: OPEN_PAR         Lexema: (          Linha: 27
Token: IDENTIFIER       Lexema: opcao      Linha: 27
Token: CLOSE_PAR        Lexema: )          Linha: 27
Token: SEMICOLON        Lexema: ;          Linha: 27
Token: WHILE            Lexema: while      Linha: 28
Token: OPEN_PAR         Lexema: (          Linha: 28
Token: IDENTIFIER       Lexema: opcao      Linha: 28
Token: NOT_EQUALS       Lexema: !=         Linha: 28
Token: CHAR_CONST       Lexema: N          Linha: 28
Erro ao analisar léxico: Na linha 28->Erro Lexico: operador de E deve ser '&&', mas foi encontrado somente '&'
```

Como é possível ver, foi gerado um erro pois tínhamos o operador & sozinho, consertando o erro:

```
Analisando: teste6.txt
Token: PROGRAM          Lexema: program    Linha: 1
Token: INTEGER_CONST    Lexema: int        Linha: 2
Token: COLON            Lexema: :          Linha: 2
Token: IDENTIFIER       Lexema: x          Linha: 2
Token: COMMA            Lexema: ,          Linha: 2
Token: IDENTIFIER       Lexema: y          Linha: 2
Token: SEMICOLON        Lexema: ;          Linha: 2
Token: FLOAT_CONST      Lexema: float      Linha: 3
Token: COLON            Lexema: :          Linha: 3
Token: IDENTIFIER       Lexema: media      Linha: 3
Token: SEMICOLON        Lexema: ;          Linha: 3
Token: CHAR_CONST       Lexema: char       Linha: 4
Token: COLON            Lexema: :          Linha: 4
Token: IDENTIFIER       Lexema: opcao      Linha: 4
Token: SEMICOLON        Lexema: ;          Linha: 4
Token: BEGIN            Lexema: begin      Linha: 7
Token: OUT              Lexema: out        Linha: 11
Token: OPEN_PAR         Lexema: (          Linha: 11
Token: LITERAL          Lexema: Digite o valor de x:  Linha: 11
Token: CLOSE_PAR        Lexema: )          Linha: 11
Token: SEMICOLON        Lexema: ;          Linha: 11
Token: IN           Lexema: in         Linha: 12
Token: OPEN_PAR         Lexema: (          Linha: 12
Token: IDENTIFIER       Lexema: x          Linha: 12
Token: CLOSE_PAR        Lexema: )          Linha: 12
Token: SEMICOLON        Lexema: ;          Linha: 12
Token: OUT              Lexema: out        Linha: 13
Token: OPEN_PAR         Lexema: (          Linha: 13
Token: LITERAL          Lexema: Digite o valor de y:  Linha: 13
Token: CLOSE_PAR        Lexema: )          Linha: 13
Token: SEMICOLON        Lexema: ;          Linha: 13
Token: IN           Lexema: in         Linha: 14
Token: OPEN_PAR         Lexema: (          Linha: 14
Token: IDENTIFIER       Lexema: y          Linha: 14
Token: CLOSE_PAR        Lexema: )          Linha: 14
Token: SEMICOLON        Lexema: ;          Linha: 14
Token: IDENTIFIER       Lexema: media      Linha: 16
Token: COLON            Lexema: :          Linha: 16
Token: ASSIGN           Lexema: =          Linha: 16
Token: OPEN_PAR         Lexema: (          Linha: 16
Token: IDENTIFIER       Lexema: x          Linha: 16
Token: ADD              Lexema: +          Linha: 16
Token: IDENTIFIER       Lexema: y          Linha: 16
Token: CLOSE_PAR        Lexema: )          Linha: 16
Token: DIV          Lexema: /          Linha: 16
```

```
Token: FLOAT_CONST   Lexema: 2.0        Linha: 16
Token: SEMICOLON     Lexema: ;          Linha: 16
Token: OUT           Lexema: out        Linha: 17
Token: OPEN_PAR      Lexema: (          Linha: 17
Token: LITERAL       Lexema: Media dos valores:  Linha: 17
Token: CLOSE_PAR     Lexema: )          Linha: 17
Token: SEMICOLON     Lexema: ;          Linha: 17
Token: OUT           Lexema: out        Linha: 18
Token: OPEN_PAR      Lexema: (          Linha: 18
Token: IDENTIFIER    Lexema: media      Linha: 18
Token: CLOSE_PAR     Lexema: )          Linha: 18
Token: SEMICOLON     Lexema: ;          Linha: 18
Token: IF            Lexema: if         Linha: 20
Token: OPEN_PAR      Lexema: (          Linha: 20
Token: IDENTIFIER    Lexema: media      Linha: 20
Token: GREATER_EQUAL   Lexema: >=               Linha: 20
Token: FLOAT_CONST   Lexema: 5.0        Linha: 20
Token: CLOSE_PAR     Lexema: )          Linha: 20
Token: THEN          Lexema: then       Linha: 20
Token: OUT           Lexema: out        Linha: 21
Token: OPEN_PAR      Lexema: (          Linha: 21
Token: LITERAL       Lexema: Media suficiente! Linha: 21
Token: CLOSE_PAR     Lexema: )          Linha: 21
Token: SEMICOLON     Lexema: ;          Linha: 21
Token: ELSE          Lexema: else       Linha: 22
Token: OUT           Lexema: out        Linha: 23
Token: OPEN_PAR      Lexema: (          Linha: 23
Token: LITERAL       Lexema: Media insuficiente! Linha: 23
Token: CLOSE_PAR     Lexema: )          Linha: 23
Token: SEMICOLON     Lexema: ;          Linha: 23
Token: END           Lexema: end        Linha: 24
Token: SEMICOLON     Lexema: ;          Linha: 24
Token: OUT           Lexema: out        Linha: 26
Token: OPEN_PAR      Lexema: (          Linha: 26
Token: LITERAL       Lexema: Digite a opcao (S/N):  Linha: 26
Token: CLOSE_PAR     Lexema: )          Linha: 26
Token: SEMICOLON     Lexema: ;          Linha: 26
Token: IN            Lexema: in         Linha: 27
Token: OPEN_PAR      Lexema: (          Linha: 27
Token: IDENTIFIER    Lexema: opcao      Linha: 27
Token: CLOSE_PAR     Lexema: )          Linha: 27
Token: SEMICOLON     Lexema: ;          Linha: 27
Token: WHILE         Lexema: while      Linha: 28
Token: OPEN_PAR      Lexema: (          Linha: 28
Token: IDENTIFIER    Lexema: opcao      Linha: 28
Token: NOT_EQUALS    Lexema: !=         Linha: 28
Token: CHAR_CONST    Lexema: N          Linha: 28
Token: AND           Lexema: &&         Linha: 28
Token: IDENTIFIER    Lexema: opcao      Linha: 28
Token: NOT_EQUALS    Lexema: !=         Linha: 28
Token: CHAR_CONST    Lexema: n          Linha: 28
Token: CLOSE_PAR     Lexema: )          Linha: 28
Token: DO            Lexema: do         Linha: 28
Token: OUT           Lexema: out        Linha: 29
Token: OPEN_PAR      Lexema: (          Linha: 29
Token: LITERAL       Lexema: Digite novos valores para x e y:  Linha: 29
Token: CLOSE_PAR     Lexema: )          Linha: 29
Token: SEMICOLON     Lexema: ;          Linha: 29
Token: IN            Lexema: in         Linha: 30
Token: OPEN_PAR      Lexema: (          Linha: 30
```

Token: IDENTIFIER Lexema: x Linha: 30
Token: CLOSE_PAR Lexema:) Linha: 30
Token: SEMICOLON Lexema: ; Linha: 30
Token: IN Lexema: in Linha: 31
Token: OPEN_PAR Lexema: (Linha: 31
Token: IDENTIFIER Lexema: y Linha: 31
Token: CLOSE_PAR Lexema:) Linha: 31
Token: SEMICOLON Lexema: ; Linha: 31
Token: IDENTIFIER Lexema: media Linha: 32
Token: COLON Lexema: : Linha: 32
Token: ASSIGN Lexema: = Linha: 32
Token: OPEN_PAR Lexema: (Linha: 32
Token: IDENTIFIER Lexema: x Linha: 32
Token: ADD Lexema: + Linha: 32
Token: IDENTIFIER Lexema: y Linha: 32
Token: CLOSE_PAR Lexema:) Linha: 32
Token: DIV Lexema: / Linha: 32
Token: FLOAT_CONST Lexema: 2.0 Linha: 32
Token: SEMICOLON Lexema: ; Linha: 32
Token: OUT Lexema: out Linha: 33
Token: OPEN_PAR Lexema: (Linha: 33
Token: LITERAL Lexema: Nova media: Linha: 33
Token: CLOSE_PAR Lexema:) Linha: 33
Token: SEMICOLON Lexema: ; Linha: 33
Token: OUT Lexema: out Linha: 34
Token: OPEN_PAR Lexema: (Linha: 34
Token: IDENTIFIER Lexema: media Linha: 34
Token: CLOSE_PAR Lexema:) Linha: 34
Token: SEMICOLON Lexema: ; Linha: 34
Token: OUT Lexema: out Linha: 35
Token: OPEN_PAR Lexema: (Linha: 35
Token: LITERAL Lexema: Digite a opcao (S/N): Linha: 35
Token: CLOSE_PAR Lexema:) Linha: 35
Token: SEMICOLON Lexema: ; Linha: 35
Token: IN Lexema: in Linha: 36
Token: OPEN_PAR Lexema: (Linha: 36
Token: IDENTIFIER Lexema: opcao Linha: 36
Token: CLOSE_PAR Lexema:) Linha: 36
Token: SEMICOLON Lexema: ; Linha: 36
Token: END Lexema: end Linha: 37
Token: SEMICOLON Lexema: ; Linha: 37
Token: END Lexema: end Linha: 38
Token: END_OF_FILE Lexema: Linha: 38