Compiladores
Relatório do trabalho prático – Analisador Léxico

Alline Santos Ferreira
Arthur Miranda Moreira

1. Introdução

Quando utilizamos um software não pensamos no que acontece em um computador para que o resultado final seja o programa e suas funções. Na disciplina de Compiladores, estamos aprendendo que existe um processo por trás do software que a partir de um código fonte, construído por um programador, há um programa que faz a tradução desse código, em um formato que o computador seja capaz de executar e, como efeito final dessa execução é exibido o programa para que o usuário final possa utilizar.

Na disciplina de Compiladores, aprendemos mais detalhadamente como essa tradução ocorre. Sabemos que a estrutura deste software é dividida em análise e síntese. Na primeira parte, o programa fonte passa por uma fase de análise da estrutura gramatical. Se ela estiver incorreta, conforme é definido pela linguagem de programação utilizada para sua construção, o compilador exibe mensagem de erro para que o programador possa identificar e corrigir. Uma vez que essa fase de análise é feita com sucesso, então, na segunda etapa, denominada síntese, o compilador constrói o programa a partir do reconhecimento das informações analisadas e, em seguida, gera o executável que é o programa utilizado pelo usuário final.

Este trabalho visa, entendermos na prática, o funcionamento do compilador. Para isso, o trabalho proposto na disciplina é dividido em três etapas: Analisador Léxico, Analisador Sintático, Analisador Semântico e Tabelas de Símbolos.

Neste relatório, iremos apresentar a construção da primeira etapa de um compilador que é o Analisador Léxico. É a etapa que identifica se a estrutura gramatical do software está lexicalmente correta. Sua função é ler um fluxo de caracteres que compõem o código fonte e os agrupar em sequências significativas denominadas por lexemas. Cada lexema, produzirá uma saída chamada de token que é utilizada na segunda etapa do trabalho que é a análise sintática. O token, tem o formato <nome-token, valor-atributo>. O nome-token é um símbolo abstrato usado durante a análise sintática, e o valor-atributo, aponta para uma tabela de símbolos referente a esse token. Essa tabela de símbolos é necessária para a análise semântica (3a. etapa do trabalho). E, ao final da construção das três etapas deste trabalho, compreenderemos na prática o funcionamento de um compilador.

2. Desenvolvimento

2.2 Descrição da implementação

Para a implementação do Analisador Léxico, utilizamos a linguagem de programação Java. E para execução do programa é necessário que seja informado o caminho onde se encontra o arquivo de teste. Ao digitar o caminho válido, o arquivo é aberto pelo analisador léxico, que analisa os caracteres do arquivo fonte e retorna o token formado, a cada chamada da função nextToken(). Caso haja algum erro durante a formação dos tokens, o analisador lança uma exceção léxica. Para entender melhor, como executar o programa, a seguir iremos explicar em detalhes a construção de cada classe e sua função.

2.2.1 Classes do programa

Para a construção deste analisador, foram desenvolvidas as seguintes classes: App, Lexer, Word, Num, Token e Tag.

A classe **Tag** define as constantes que representam os diferentes tipos de tokens. Já a classe **Token** é responsável por armazenar essas representações de forma padronizada.

A classe **Word** gerencia os lexemas relacionados a palavras reservadas, identificadores e operadores compostos como: &&, conforme definidos pela gramática do analisador. A classe **Num**, por sua vez, trata dos tokens numéricos reconhecidos durante a análise.

A lógica do analisador léxico está implementada na classe **Lexer**. É nessa classe que ocorre a leitura do código-fonte de teste, bem como a verificação de sua conformidade sintática e semântica com a estrutura gramatical definida. Caso seja detectado algum erro, o programa exibirá uma mensagem com a descrição do problema e a linha onde ele ocorreu, interrompendo a execução para que o programador possa corrigi-lo antes de tentar compilar novamente. Ou seja, nesta implementação não há recuperação de erro (erros são considerados fatais).

Por fim, a classe **App** contém o método main. Sua função é verificar se o arquivo de teste foi localizado. Se encontrado, a classe Lexer é chamada para iniciar a análise léxica; caso contrário, uma mensagem será exibida informando que o arquivo não foi encontrado. Essa classe contém um loop para ler todo o arquivo, e a cada iteração é impresso na tela o token lido. Foi utilizado um switch case para ler os os Tokens de operadores e pontuação (vírgula, ponto e vírgula, sinal de adição, entre outros).

Os códigos construídos nas classes: Tag, Word, Token, Num, foram inspirados no trabalho dos autores Alfred Aho, Monica Lam, Ravi Sethi e Jeffrey Ullman.

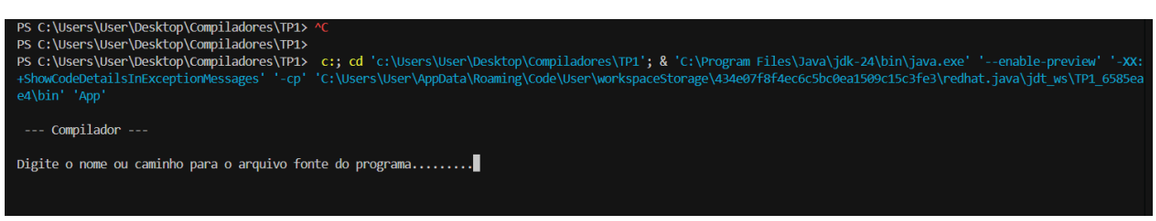
3. Resultados

Para executarmos o programa a fim de avaliarmos a criação do Analisador Léxico, foi disponibilizado pela professora 5 arquivos testes. O objetivo era verificar se o compilador construído, reconheceria os erros, léxicos, semânticos e sintáticos existentes, identificando através de uma mensagem em qual linha houve o erro e abortando a execução. Em seguida, os erros seriam corrigidos pelo usuário e ao repetir a execução, ele deveria avaliar novamente e com sucesso. Nessa etapa, os únicos erros identificados devem ser os erros léxicos, relacionados à formação dos tokens, destacados de vermelho nos programas de teste.

A seguir mostraremos os resultados de execução de todos os arquivos testes. E outro teste feito pelos autores do trabalho.

3.1 Início da execução

Informar o caminho onde está salvo o arquivo teste.



3.2 Programas de entrada e saída do compilador

Teste do Programa 1:

```
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
```

```
 --- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.....C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa1.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: int , linha: 2
Token lido: : , linha: 2
Token lido: a , linha: 2
Token lido: , , linha: 2
Token lido: b , linha: 2
Token lido: , , linha: 2
Token lido: c , linha: 2
Token lido: ; , linha: 2
Token lido: float , linha: 3
Token lido: : , linha: 3
Token lido: result , linha: 3
Token lido: ; , linha: 3
Token lido: char , linha: 4
Token lido: : , linha: 4
Token lido: ch , linha: 4
Token lido: ; , linha: 4
Token lido: begin , linha: 6
Token lido: out , linha: 7
Token lido: ( , linha: 7
Token lido: "Digite o valor de a:" , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: in , linha: 8
Token lido: ( , linha: 8
Token lido: a , linha: 8
Token lido: ) , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Digite o valor de c:" , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
```

```
Token lido: "Digite o valor de a:" , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: in , linha: 8
Token lido: ( , linha: 8
Token lido: a , linha: 8
Token lido: ) , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Digite o valor de c:" , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: read , linha: 10
Token lido: ( , linha: 10
Token lido: ch , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: b , linha: 11
Token lido: = , linha: 11
Token lido: 10 , linha: 11
Token lido: ; , linha: 11
Token lido: result , linha: 12
Token lido: = , linha: 12
Token lido: ( , linha: 12
Token lido: a , linha: 12
Token lido: * , linha: 12
Token lido: ch , linha: 12
Token lido: ) , linha: 12
Token lido: / , linha: 12
Token lido: ( , linha: 12
Token lido: b , linha: 12
Token lido: 5 , linha: 12
Exception in thread "main" LexicalException: Caractere inválido na linha 12
        at Lexer.nextToken(Lexer.java:266)
        at App.main(App.java:16)
PS C:\Users\User\Desktop\Compiladores\TP1>
```

O programa é lido até chegar na linha 12, onde ocorre um erro devido um caractere inválido (‘–’), no lugar de um sinal de subtração. Substituindo ‘–’ por ‘-’ no programa1.txt e executando novamente:

```
--- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa1.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: int , linha: 2
Token lido: : , linha: 2
Token lido: a , linha: 2
Token lido: , , linha: 2
Token lido: b , linha: 2
Token lido: , , linha: 2
Token lido: c , linha: 2
Token lido: ; , linha: 2
Token lido: float , linha: 3
Token lido: : , linha: 3
Token lido: result , linha: 3
Token lido: ; , linha: 3
Token lido: char , linha: 4
Token lido: : , linha: 4
Token lido: ch , linha: 4
Token lido: ; , linha: 4
Token lido: begin , linha: 6
Token lido: out , linha: 7
Token lido: ( , linha: 7
Token lido: "Digite o valor de a:" , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: in , linha: 8
Token lido: ( , linha: 8
Token lido: a , linha: 8
Token lido: ) , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Digite o valor de c:" , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: read , linha: 10
Token lido: ( , linha: 10
Token lido: ch , linha: 10
Token lido: ) , linha: 10
```

```
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: b , linha: 11
Token lido: = , linha: 11
Token lido: 10 , linha: 11
Token lido: ; , linha: 11
Token lido: result , linha: 12
Token lido: = , linha: 12
Token lido: ( , linha: 12
Token lido: a , linha: 12
Token lido: * , linha: 12
Token lido: ch , linha: 12
Token lido: ) , linha: 12
Token lido: / , linha: 12
Token lido: ( , linha: 12
Token lido: b , linha: 12
Token lido: 5 , linha: 12
Token lido: - , linha: 12
Token lido: 345.27 , linha: 12
Token lido: ) , linha: 12
Token lido: ; , linha: 12
Token lido: out , linha: 13
Token lido: ( , linha: 13
Token lido: "O resultado e: " , linha: 13
Token lido: ) , linha: 13
Token lido: ; , linha: 13
Token lido: out , linha: 14
Token lido: ( , linha: 14
Token lido: result , linha: 14
Token lido: ) , linha: 14
Token lido: ; , linha: 14
Token lido: result , linha: 15
Token lido: = , linha: 15
Token lido: result , linha: 15
Token lido: + , linha: 15
Token lido: ch , linha: 15
Token lido: ; , linha: 15
Token lido: end , linha: 16
Token lido: null - Fim de arquivo
```

O arquivo é processado até o final.

```
Tabela de símbolos:
{a=a, b=b, c=c, read=read, in=in, ch=ch, program=program, then=then, do=do, while=while, float=float, int=int, out=out, result=result, else=else, repeat=repeat, ch
ar=char, end=end, until=until, begin=begin, if=if}
PS C:\Users\User\Desktop\Compiladores\TP1>
```

Teste do Programa 2:

```
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
```

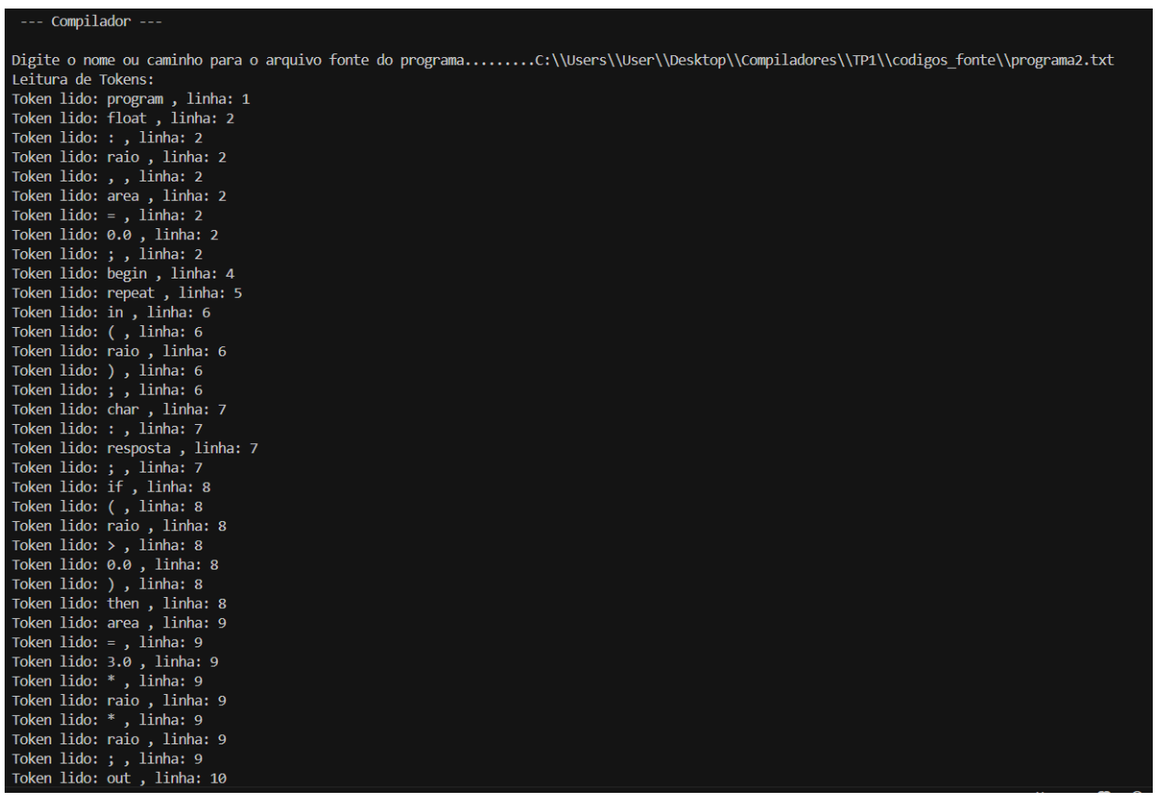
```
--- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa2.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: float , linha: 2
Token lido: : , linha: 2
Token lido: raio , linha: 2
Token lido: , , linha: 2
Token lido: area , linha: 2
Exception in thread "main" LexicalException: Caractere inválido na linha 2
        at Lexer.nextToken(Lexer.java:266)
        at App.main(App.java:16)
PS C:\Users\User\Desktop\Compiladores\TP1>
```

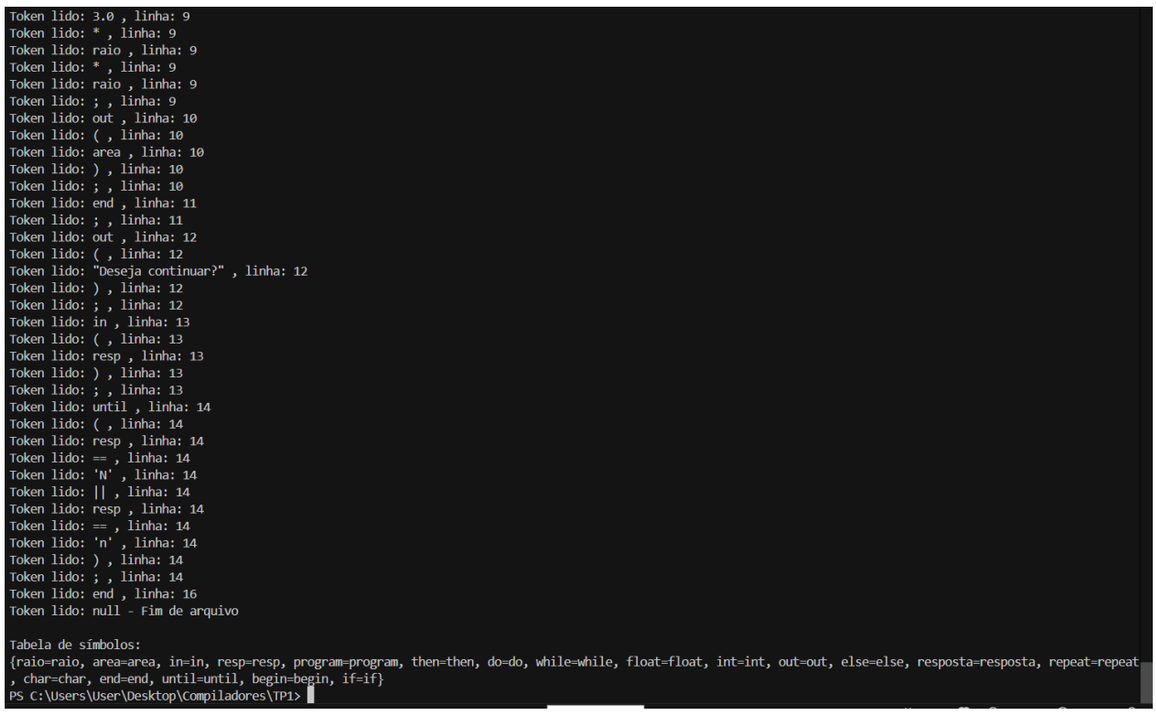
A execução ocorre até a linha 2, onde é lido um caractere inválido (‘$’). Removendo ‘$’ no programa2.txt e executando novamente:

```
--- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa2.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: float , linha: 2
Token lido: : , linha: 2
Token lido: raio , linha: 2
Token lido: , , linha: 2
Token lido: area , linha: 2
Token lido: = , linha: 2
Token lido: 0.0 , linha: 2
Token lido: ; , linha: 2
Token lido: begin , linha: 4
Token lido: repeat , linha: 5
Token lido: in , linha: 6
Token lido: ( , linha: 6
Token lido: raio , linha: 6
Token lido: ) , linha: 6
Token lido: ; , linha: 6
Token lido: char , linha: 7
Token lido: : , linha: 7
Token lido: resposta , linha: 7
Token lido: ; , linha: 7
Token lido: if , linha: 8
Token lido: ( , linha: 8
Token lido: raio , linha: 8
Token lido: > , linha: 8
Token lido: 0.0 , linha: 8
Token lido: ) , linha: 8
Token lido: then , linha: 8
Token lido: area , linha: 9
Token lido: = , linha: 9
Exception in thread "main" LexicalException: Float mal formado (esperado dígito, encontrado  ), na linha 9
        at Lexer.nextToken(Lexer.java:196)
        at App.main(App.java:16)
PS C:\Users\User\Desktop\Compiladores\TP1>
```

A execução ocorre até a linha 9, onde é lido um erro por float mal formado (esperado um dígito após “3.). Removendo acrescentando um 0, de forma que fique “3.0” no programa2.txt e executando novamente:

A execução ocorre até a linha 12, onde é lido um literal contendo quebra de linha, gerando um erro. Fechando aspas no programa2.txt, de forma que a linha 12 se torne: out ("Deseja continuar?");, e executando novamente:





O arquivo é processado até o final.

Teste do Programa 3:

```
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
```

```
 --- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.....C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa3.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: int , linha: 3
Token lido: : , linha: 3
Token lido: a , linha: 3
Token lido: , , linha: 3
Token lido: b , linha: 3
Token lido: , , linha: 3
Token lido: aux , linha: 3
Token lido: ; , linha: 3
Token lido: begin , linha: 5
Token lido: in , linha: 6
Token lido: ( , linha: 6
Token lido: a , linha: 6
Token lido: ) , linha: 6
Token lido: ; , linha: 6
Token lido: in , linha: 7
Token lido: ( , linha: 7
Token lido: b , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: if , linha: 8
Token lido: ( , linha: 8
Token lido: a , linha: 8
Token lido: > , linha: 8
Token lido: b , linha: 8
Token lido: ) , linha: 8
Token lido: then , linha: 8
Token lido: int , linha: 9
Token lido: aux , linha: 9
Token lido: ; , linha: 9
Token lido: aux , linha: 10
Token lido: = , linha: 10
Token lido: b , linha: 10
Token lido: ; , linha: 10
Token lido: b , linha: 11
Token lido: = , linha: 11
```

```
Token lido: b , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: if , linha: 8
Token lido: ( , linha: 8
Token lido: a , linha: 8
Token lido: > , linha: 8
Token lido: b , linha: 8
Token lido: ) , linha: 8
Token lido: then , linha: 8
Token lido: int , linha: 9
Token lido: aux , linha: 9
Token lido: ; , linha: 9
Token lido: aux , linha: 10
Token lido: = , linha: 10
Token lido: b , linha: 10
Token lido: ; , linha: 10
Token lido: b , linha: 11
Token lido: = , linha: 11
Token lido: a , linha: 11
Token lido: ; , linha: 11
Token lido: a , linha: 12
Token lido: = , linha: 12
Token lido: aux , linha: 12
Token lido: end , linha: 13
Token lido: ; , linha: 13
Token lido: out , linha: 14
Token lido: ( , linha: 14
Token lido: a , linha: 14
Token lido: ; , linha: 14
Token lido: out , linha: 15
Token lido: ( , linha: 15
Token lido: b , linha: 15
Token lido: ) , linha: 15
Token lido: end , linha: 16
Token lido: null - Fim de arquivo

Tabela de símbolos:
{a=a, b=b, in=in, aux=aux, program=program, then=then, do=do, while=while, float=float, int=int, out=out, else=else, repeat=repeat, char=char, end=end, until=until
, begin=begin, if=if}
PS C:\Users\User\Desktop\Compiladores\TP1>
```

O arquivo é processado até o final. O programa até possui erros, como a ausência de ponto e vírgula e parênteses não fechado, mas trata-se de erro sintático, não léxico, de forma que o analisador léxico não percebe erro algum.

Teste do Programa 4:

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

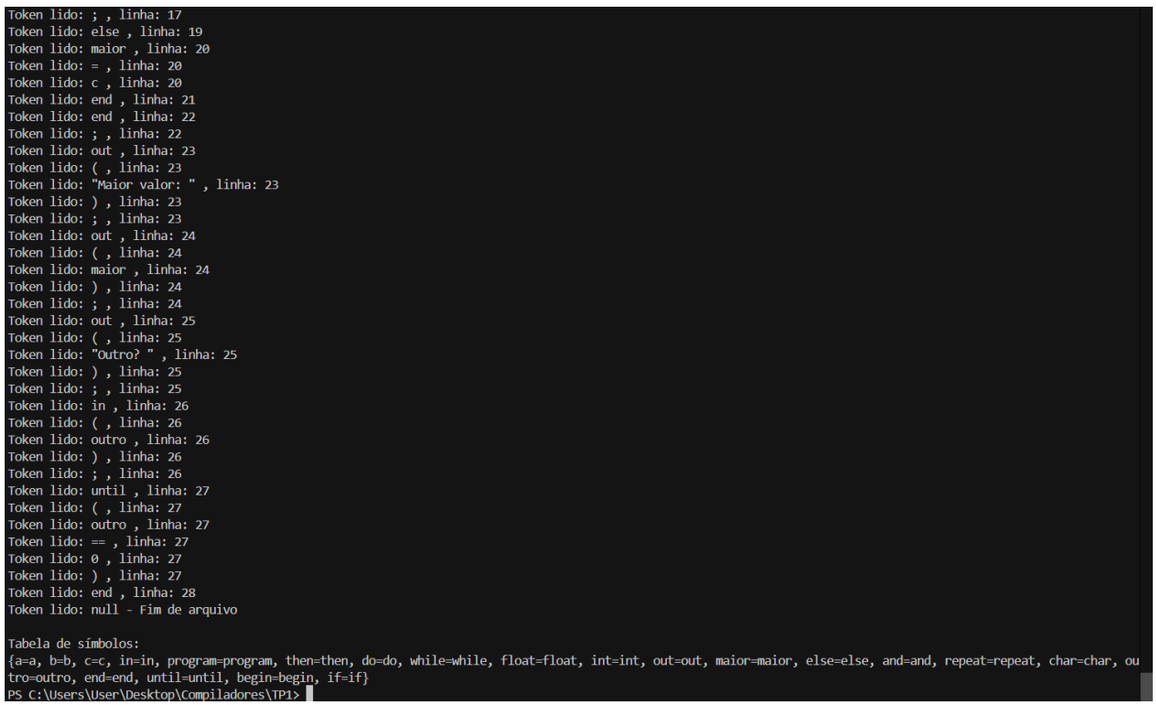
```
    in(c);
    %Verifica o maior
    if ( (a>b) and (a>c) ) end
        maior = a

    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
```

```
 --- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa4.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: a , linha: 2
Token lido: , , linha: 2
Token lido: b , linha: 2
Token lido: , , linha: 2
Token lido: c , linha: 2
Token lido: , , linha: 2
Token lido: maior , linha: 2
Token lido: , , linha: 2
Token lido: outro , linha: 2
Token lido: : , linha: 2
Token lido: int , linha: 2
Token lido: ; , linha: 2
Token lido: begin , linha: 3
Token lido: repeat , linha: 4
Token lido: out , linha: 5
Token lido: ( , linha: 5
Token lido: "A" , linha: 5
Token lido: ) , linha: 5
Token lido: ; , linha: 5
Token lido: in , linha: 6
Token lido: ( , linha: 6
Token lido: a , linha: 6
Token lido: ) , linha: 6
Token lido: ; , linha: 6
Token lido: out , linha: 7
Token lido: ( , linha: 7
Token lido: "B" , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: in , linha: 8
Token lido: ( , linha: 8
Token lido: b , linha: 8
Token lido: ) , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
```

A execução ocorre até a linha 9, onde é lido um literal contendo quebra de linha, gerando um erro. Removendo a aspas a mais no programa4.txt, de forma que a linha 9 se torne: out("Maior valor: ");, e executando novamente:

```
 --- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa4.txt
Leitura de Tokens:
Token lido: program , linha: 1
Token lido: a , linha: 2
Token lido: , , linha: 2
Token lido: b , linha: 2
Token lido: , , linha: 2
Token lido: c , linha: 2
Token lido: , , linha: 2
Token lido: maior , linha: 2
Token lido: , , linha: 2
Token lido: outro , linha: 2
Token lido: : , linha: 2
Token lido: int , linha: 2
Token lido: ; , linha: 2
Token lido: begin , linha: 3
Token lido: repeat , linha: 4
Token lido: out , linha: 5
Token lido: ( , linha: 5
Token lido: "A" , linha: 5
Token lido: ) , linha: 5
Token lido: ; , linha: 5
Token lido: in , linha: 6
Token lido: ( , linha: 6
Token lido: a , linha: 6
Token lido: ) , linha: 6
Token lido: ; , linha: 6
Token lido: out , linha: 7
Token lido: ( , linha: 7
Token lido: "B" , linha: 7
Token lido: ) , linha: 7
Token lido: ; , linha: 7
Token lido: in , linha: 8
Token lido: ( , linha: 8
Token lido: b , linha: 8
Token lido: ) , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
```

```
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "C" , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: in , linha: 10
Token lido: ( , linha: 10
Token lido: c , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: if , linha: 12
Token lido: ( , linha: 12
Token lido: ( , linha: 12
Token lido: a , linha: 12
Token lido: > , linha: 12
Token lido: b , linha: 12
Token lido: ) , linha: 12
Token lido: and , linha: 12
Token lido: ( , linha: 12
Token lido: a , linha: 12
Token lido: > , linha: 12
Token lido: c , linha: 12
Token lido: ) , linha: 12
Token lido: ) , linha: 12
Token lido: end , linha: 12
Token lido: maior , linha: 13
Token lido: = , linha: 13
Token lido: a , linha: 13
Token lido: else , linha: 15
Token lido: if , linha: 16
Token lido: ( , linha: 16
Token lido: b , linha: 16
Token lido: > , linha: 16
Token lido: c , linha: 16
Token lido: ) , linha: 16
Token lido: then , linha: 16
Token lido: maior , linha: 17
Token lido: = , linha: 17
Token lido: b , linha: 17
Token lido: ; , linha: 17
Token lido: else , linha: 19
```



O arquivo é processado até o final. Novamente, existem outros erros, mas não são erros léxicos.

Teste do Programa 5:

```
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

```
--- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.........C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa5.txt
Leitura de Tokens:
Token lido: programa , linha: 1
Token lido: declare , linha: 2
Token lido: inteiro , linha: 3
Token lido: : , linha: 3
Token lido: pontuacao , linha: 3
Token lido: , , linha: 3
Token lido: pontuacaomaxina , linha: 3
Token lido: , , linha: 3
Token lido: disponibilidade , linha: 3
Token lido: ; , linha: 3
Token lido: char , linha: 4
Token lido: : , linha: 4
Token lido: pontuacaominima , linha: 4
Token lido: begin , linha: 5
Token lido: disponibilidade , linha: 6
Token lido: = , linha: 6
Token lido: 'S' , linha: 6
Token lido: ; , linha: 6
Token lido: pontuacaominima , linha: 7
Token lido: = , linha: 7
Token lido: 50 , linha: 7
Token lido: ; , linha: 7
Token lido: pontuacaomaxima , linha: 8
Token lido: = , linha: 8
Token lido: 100 , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Pontuacao Candidato: " , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: in , linha: 10
Token lido: ( , linha: 10
Token lido: pontuacao , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: out , linha: 11
```

```
Token lido: ; , linha: 3
Token lido: char , linha: 4
Token lido: : , linha: 4
Token lido: pontuacaominima , linha: 4
Token lido: begin , linha: 5
Token lido: disponibilidade , linha: 6
Token lido: = , linha: 6
Token lido: 'S' , linha: 6
Token lido: ; , linha: 6
Token lido: pontuacaominima , linha: 7
Token lido: = , linha: 7
Token lido: 50 , linha: 7
Token lido: ; , linha: 7
Token lido: pontuacaomaxima , linha: 8
Token lido: = , linha: 8
Token lido: 100 , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Pontuacao Candidato: " , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: in , linha: 10
Token lido: ( , linha: 10
Token lido: pontuacao , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: out , linha: 11
Token lido: ( , linha: 11
Token lido: "Disponibilidade Candidato: " , linha: 11
Token lido: ) , linha: 11
Token lido: ; , linha: 11
Token lido: in , linha: 12
Token lido: ( , linha: 12
Token lido: disponibilidade , linha: 12
Token lido: ) , linha: 12
Token lido: ; , linha: 12
Exception in thread "main" LexicalException: Comentário não fechado, na linha 34
        at Lexer.nextToken(Lexer.java:90)
        at App.main(App.java:16)
PS C:\Users\User\Desktop\Compiladores\TP1>
```

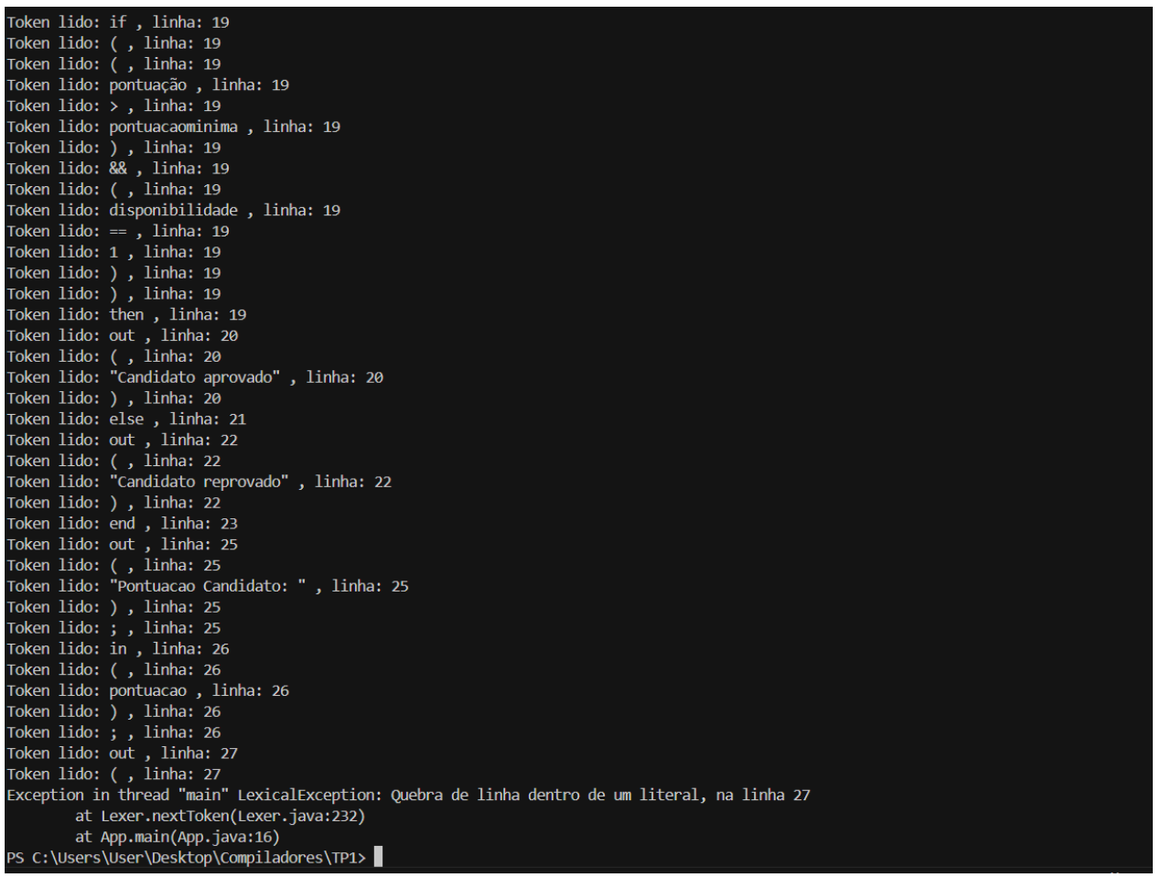
O compilador lê um ‘{‘, que inicia um comentário de mais de uma linha, de forma que todos os caracteres lidos a seguir são ignorados. Porém, o comentário não é fechado, causando um erro. Fechando o comentário:

{ Comentario
grande}

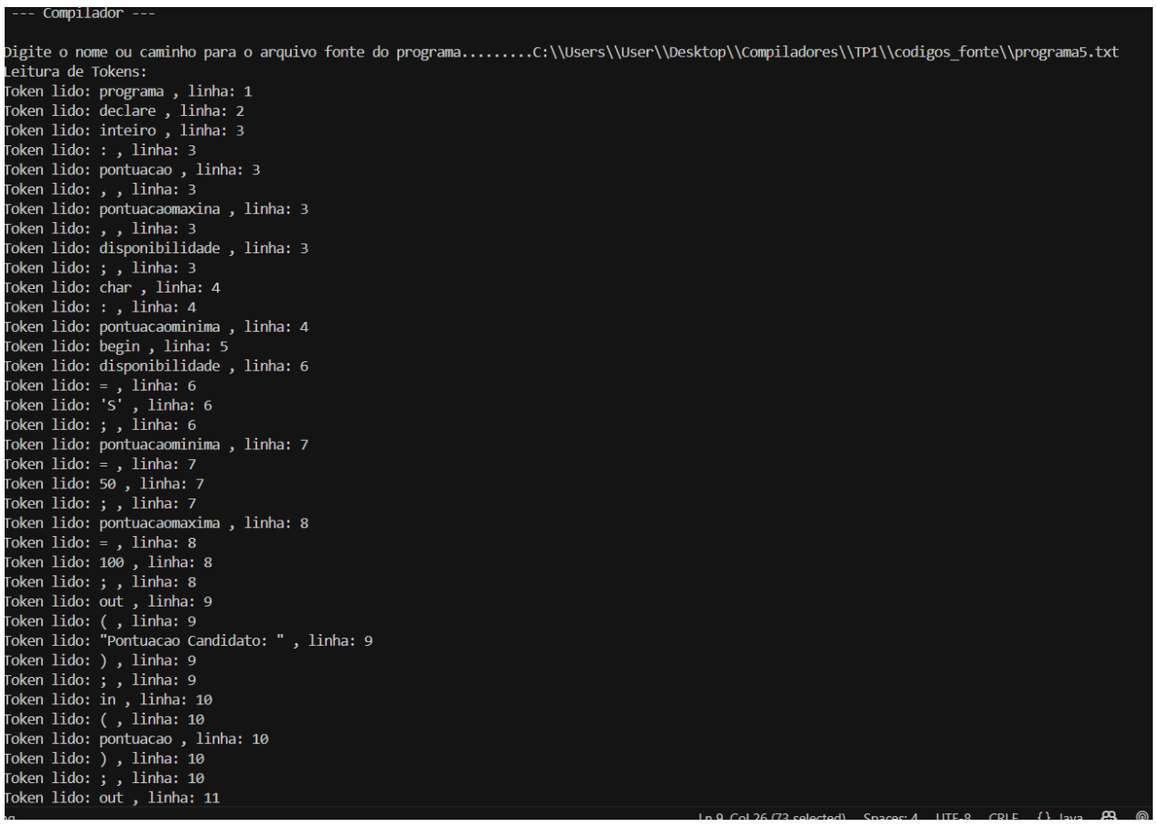
E executando novamente:

```
--- Compilador ---

Digite o nome ou caminho para o arquivo fonte do programa.....C:\\Users\\User\\Desktop\\Compiladores\\TP1\\codigos_fonte\\programa5.txt
Leitura de Tokens:
Token lido: programa , linha: 1
Token lido: declare , linha: 2
Token lido: inteiro , linha: 3
Token lido: : , linha: 3
Token lido: pontuacao , linha: 3
Token lido: , , linha: 3
Token lido: pontuacaomaxina , linha: 3
Token lido: , , linha: 3
Token lido: disponibilidade , linha: 3
Token lido: ; , linha: 3
Token lido: char , linha: 4
Token lido: : , linha: 4
Token lido: pontuacaominima , linha: 4
Token lido: begin , linha: 5
Token lido: disponibilidade , linha: 6
Token lido: = , linha: 6
Token lido: 'S' , linha: 6
Token lido: ; , linha: 6
Token lido: pontuacaominima , linha: 7
Token lido: = , linha: 7
Token lido: 50 , linha: 7
Token lido: ; , linha: 7
Token lido: pontuacaomaxima , linha: 8
Token lido: = , linha: 8
Token lido: 100 , linha: 8
Token lido: ; , linha: 8
Token lido: out , linha: 9
Token lido: ( , linha: 9
Token lido: "Pontuacao Candidato: " , linha: 9
Token lido: ) , linha: 9
Token lido: ; , linha: 9
Token lido: in , linha: 10
Token lido: ( , linha: 10
Token lido: pontuacao , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
```

```
Token lido: ; , linha: 9
Token lido: in , linha: 10
Token lido: ( , linha: 10
Token lido: pontuacao , linha: 10
Token lido: ) , linha: 10
Token lido: ; , linha: 10
Token lido: out , linha: 11
Token lido: ( , linha: 11
Token lido: "Disponibilidade Candidato: " , linha: 11
Token lido: ) , linha: 11
Token lido: ; , linha: 11
Token lido: in , linha: 12
Token lido: ( , linha: 12
Token lido: disponibilidade , linha: 12
Token lido: ) , linha: 12
Token lido: ; , linha: 12
Token lido: while , linha: 16
Token lido: ( , linha: 16
Token lido: pontuacao , linha: 16
Token lido: > , linha: 16
Token lido: 0 , linha: 16
Token lido: && , linha: 16
Token lido: ( , linha: 16
Token lido: pontuação , linha: 16
Token lido: <= , linha: 16
Token lido: pontuacaomaxima , linha: 16
Token lido: ) , linha: 16
Token lido: do , linha: 16
Token lido: int , linha: 17
Token lido: : , linha: 17
Token lido: cont , linha: 17
Token lido: ; , linha: 17
Token lido: cont , linha: 18
Token lido: = , linha: 18
Token lido: cont , linha: 18
Token lido: + , linha: 18
Token lido: 1 , linha: 18
Token lido: ; , linha: 18
Token lido: if , linha: 19
Token lido: ( , linha: 19
Token lido: ( , linha: 19
```

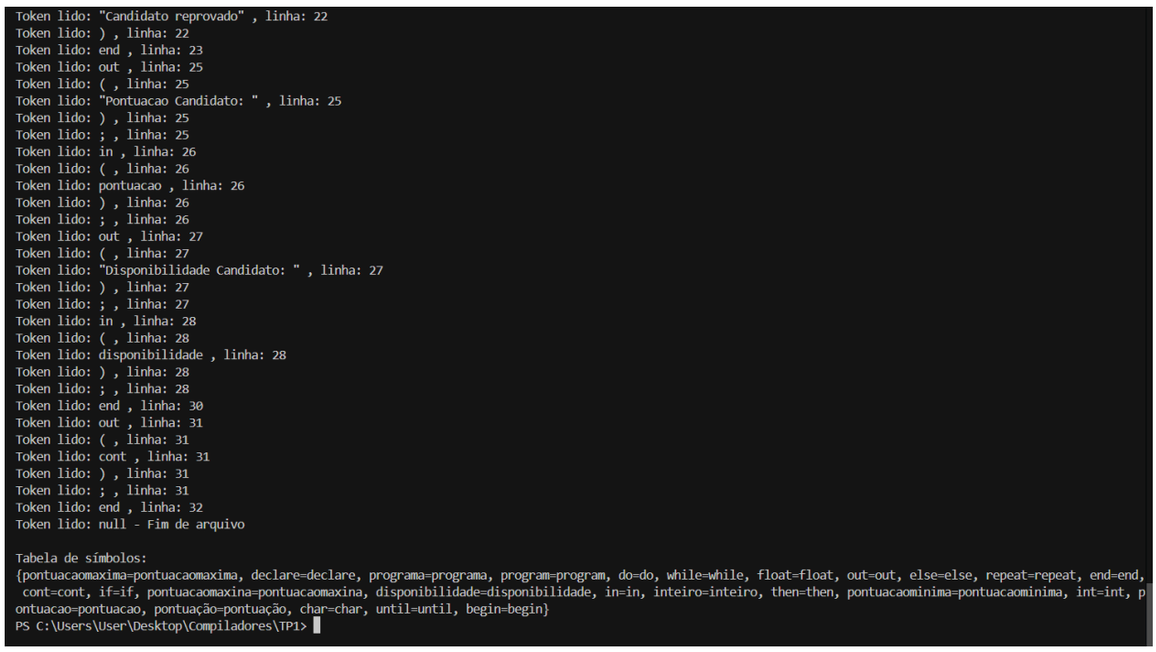


A execução ocorre até a linha 27, onde é lido um literal contendo quebra de linha. Removendo a quebra de linha, de forma que a linha se torne: out("Disponibilidade Candidato: "); , e executando novamente:



```
Token lido: ; , linha: 10
Token lido: out , linha: 11
Token lido: ( , linha: 11
Token lido: "Disponibilidade Candidato: " , linha: 11
Token lido: ) , linha: 11
Token lido: ; , linha: 11
Token lido: in , linha: 12
Token lido: ( , linha: 12
Token lido: disponibilidade , linha: 12
Token lido: ) , linha: 12
Token lido: ; , linha: 12
Token lido: while , linha: 16
Token lido: ( , linha: 16
Token lido: pontuacao , linha: 16
Token lido: > , linha: 16
Token lido: 0 , linha: 16
Token lido: && , linha: 16
Token lido: ( , linha: 16
Token lido: pontuação , linha: 16
Token lido: <= , linha: 16
Token lido: pontuacaomaxima , linha: 16
Token lido: ) , linha: 16
Token lido: do , linha: 16
Token lido: int , linha: 17
Token lido: : , linha: 17
Token lido: cont , linha: 17
Token lido: ; , linha: 17
Token lido: cont , linha: 18
Token lido: = , linha: 18
Token lido: cont , linha: 18
Token lido: + , linha: 18
Token lido: 1 , linha: 18
Token lido: ; , linha: 18
Token lido: if , linha: 19
Token lido: ( , linha: 19
Token lido: ( , linha: 19
Token lido: pontuação , linha: 19
Token lido: > , linha: 19
Token lido: pontuacaominima , linha: 19
Token lido: ) , linha: 19
Token lido: && , linha: 19
```

```
Token lido: ) , linha: 19
Token lido: && , linha: 19
Token lido: ( , linha: 19
Token lido: disponibilidade , linha: 19
Token lido: == , linha: 19
Token lido: 1 , linha: 19
Token lido: ) , linha: 19
Token lido: ) , linha: 19
Token lido: then , linha: 19
Token lido: out , linha: 20
Token lido: ( , linha: 20
Token lido: "Candidato aprovado" , linha: 20
Token lido: ) , linha: 20
Token lido: else , linha: 21
Token lido: out , linha: 22
Token lido: ( , linha: 22
Token lido: "Candidato reprovado" , linha: 22
Token lido: ) , linha: 22
Token lido: end , linha: 23
Token lido: out , linha: 25
Token lido: ( , linha: 25
Token lido: "Pontuacao Candidato: " , linha: 25
Token lido: ) , linha: 25
Token lido: ; , linha: 25
Token lido: in , linha: 26
Token lido: ( , linha: 26
Token lido: pontuacao , linha: 26
Token lido: ) , linha: 26
Token lido: ; , linha: 26
Token lido: out , linha: 27
Token lido: ( , linha: 27
Token lido: "Disponibilidade Candidato: " , linha: 27
Token lido: ) , linha: 27
Token lido: ; , linha: 27
Token lido: in , linha: 28
Token lido: ( , linha: 28
Token lido: disponibilidade , linha: 28
Token lido: ) , linha: 28
Token lido: ; , linha: 28
Token lido: end , linha: 30
Token lido: out , linha: 31
```



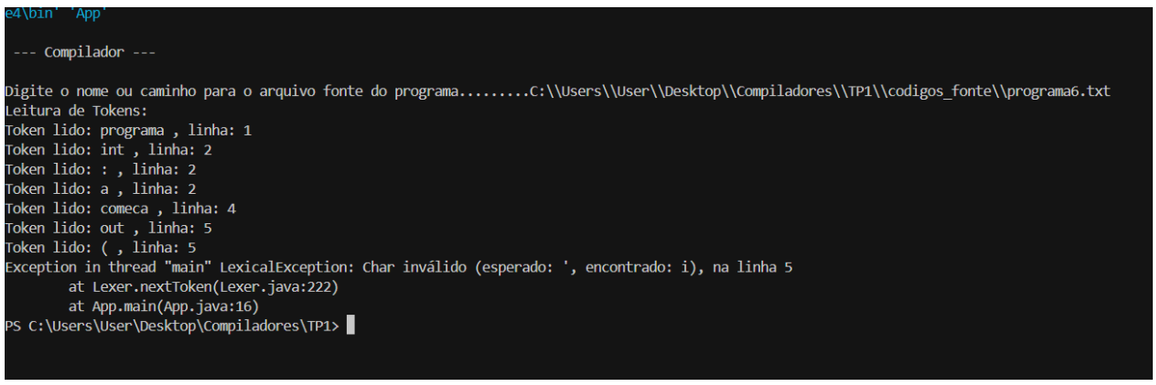
O arquivo é processado até o final.

<u>Teste do Programa 6:</u>

programa
    int: a

comeca
    out('Digite o valor de a:);
termina

A motivação deste programa é testar os erros envolvendo variáveis do tipo char.

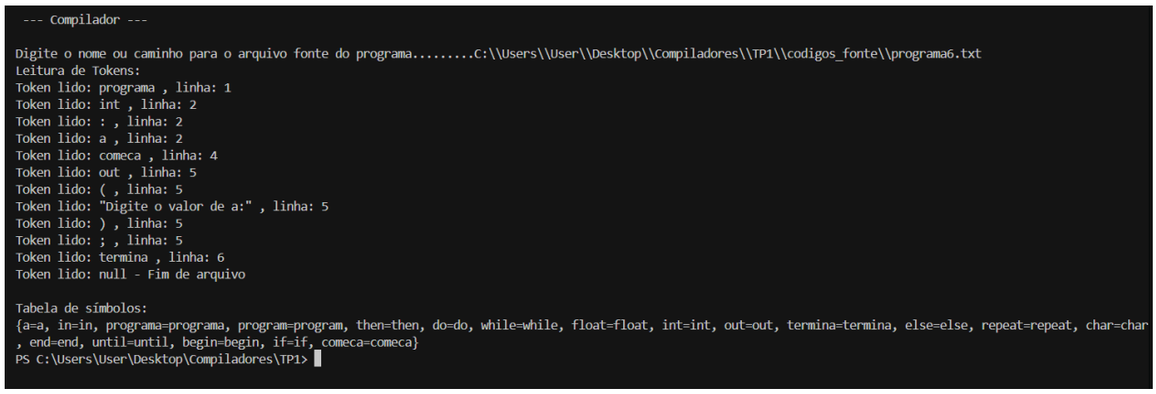


Ocorreu um erro após a leitura de ‘D , pois é esperado um fecha aspas após a leitura de um único caractere, para variáveis do tipo char. Substituindo as aspas simples por aspas duplas, de forma que o programa fique:

programa
    int: a

comeca
    out("Digite o valor de a:");
termina

E executando novamente:



A execução ocorre sem erros.

4. Conclusão

Entender o funcionamento de um compilador é importante para os alunos que pretendem atuar na área da computação. Saber que, ao escrever um programa é necessário entender que cada linguagem de programação tem uma gramática própria e durante a compilação do programa, será avaliado se a estrutura do seu programa está coerente com análise sintática e semântica dessa gramática, é fundamental para compreender até mesmo a origem dos erros reportados, durante a fase de compilação de um programa que está sendo construído.

Na construção deste trabalho, o aluno pode vivenciar na prática, como é feito a Análise Léxica que é a primeira fase para o funcionamento de um compilador.

5. Referência Bibliográfica

1. AHO, A. V. Jeffrey; Sethi, R.; Lam, MS (2008). **Compiladores. Princípios, técnicas e ferramentas.** São Paulo: Addison-Wesley, Pearson.