Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG

Compilador

Analisador Léxico

Erick H. D. de Souza

Lucas C. Dornelas

Násser R. F. Kilesse

Professora Orientadora: Kecia Aline Marques Ferreira

Abril de 2023

Sumário

Resumo

Um compilador é um programa de software que converte código fonte de uma linguagem de programação para código objeto que pode ser executado em um computador. Dentre suas atividades, temos que a primeira etapa é crucial para o seu funcionamento, caracterizada como análise sintática, onde é verificado se o código fonte segue a gramática da linguagem de programação utilizada.

Palavras-chave: compilador, gramática, analisador, sintático, software, linguagem, programação, código, fonte, objeto.

Introdução

Um compilador é um software que, a partir de um código-fonte escrito em uma linguagem de programação, gera outro código em uma linguagem alvo ou uma linguagem objeto, que pode ser executado por uma máquina ou uma máquina virtual. O processo de compilação do código-fonte é realizado em partes, denominadas de análise e síntese.

A análise é responsável por verificar se o programa fonte está de acordo com as regras da linguagem de programação fonte, e dentro dela, temos outras subdivisões: análise léxica, análise sintática e análise semântica. A atividade de síntese é responsável por construir o programa alvo ou programa objeto, a partir dos dados obtidos pelo compilador durante a fase de análise. Com base nisso, fica claro que o processo de compilação é iniciado com a realização da análise léxica do código fonte.

A análise léxica realiza a leitura dos arquivos do programa fonte, a cada caractere por vez, agrupando-os em sequências significativas, que identificam um atributo e o seu valor. Um atributo é chamado de lexema, e a combinação de um lexema e o seu valor é denominado de token. Sendo assim, é necessário que construímos um leitor de tokens do código fonte, para que estes dados sejam utilizados futuramente dentro do analisador sintático.

Desenvolvimento

Forma de uso do compilador

Para utilizar o compilador, siga os passos abaixo:

1. Extraia o arquivo **`Compilers.zip`**.
2. Abra o terminal e navegue até o diretório onde foi criada a pasta extraída.
3. Execute o compilador utilizando o comando:

   `java -jar Compilers.jar <file-path>`.

   3.1. Para executar um dos programas presentes dentro da pasta **`programs`**, onde **`<file>`** deve ser substituído pelo nome do arquivo fonte:

   `java -jar Compilers.jar ./programs/<file>.`

   3.2. Para executar programas que não estão dentro da pasta **`programs`** é necessário que o caminho para o arquivo fonte seja informado corretamente em relação a pasta extraída.
4. Para a execução do compilador, é necessário que a versão java instalada na máquina seja a **19.0.1** ou superior.

Descrição da abordagem utilizada na implementação

A seguir, temos as principais classes criadas dentro do compilador que se referem a execução do analisador léxico:

Class Tag

Um enumerador que contém as constantes que representam as tags para cada um dos elementos léxicos. Por exemplo, a tag para a palavra-chave "if" pode ser representada pela constante "IF".

Class Token

Uma classe que representa os tokens que o analisador léxico reconhece durante a execução. Um token é formado por um elemento de classe Tag.

Class Word

Uma classe que extende a classe Token que representa os identificadores e palavras-chave do código-fonte.

Class Number

Uma classe que extende a classe Token que representa os números encontrados no código-fonte, contendo o Token e o seu valor.

Class Lexer

A classe principal do analisador léxico, que é responsável por ler o código-fonte e produzir uma sequência de Tokens correspondente.

Class Table

Uma classe que mantém a tabela de símbolos do programa, contendo informações sobre os identificadores já declarados e seus tipos.

Class Compiler

Uma classe que recebe o arquivo-fonte a ser compilado e utiliza o Lexer para gerar uma sequência de Tokens a serem utilizados posteriormente na análise sintática.

Programas

program teste1

```
 1   programa teste1
 2
 3       a, b is int;
 4       result is int;
 5       a,x is float;
 6
 7   begin
 8
 9       a = 12a;
10       x = 12.;
11       read (a);
12       read (b);
13       read (c)
14       result = (a*b + 1) / (c+2);
15       write {Resultado: };
16       write (result);
17
18   end.
```

program teste2

```
 1   program teste2
 2
 3           a, b, c:int;
 4           d, _var: float;
 5
 6       teste2 = 1;
 7       Read (a);
 8       b = a * a;
 9       c = b + a/2 * (35/b);
10       write c;
11       val := 34.2
12       c = val + 2.2 + a;
13       write (val)
14   end.
```

program UNDEFINED

```
 1   program
 2       a, aux is int;
 3       b is float
 4
 5   begin
 6       b = 0;
 7       in (a);
 8       in(b);
 9       if (a>b) then //troca variaveis
10           aux = b;
11           b = a;
12           a = aux
13       end;
14       write(a;
15       write(b)
```

program teste4

```
 1   program teste4
 2
 3   /* Teste4 do meu compilador
 4
 5       pontuacao, pontuacaoMaxina, disponibilidade is inteiro;
 6       pontuacaoMinima is char;
 7
 8   begin
 9       pontuacaoMinima = 50;
10       pontuacaoMaxima = 100;
11       write({Pontuacao do candidato: });
12       read(pontuacao);
13       write({Disponibilidade do candidato: });
14       read(disponibilidade);
15
16       while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
17           if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
18               write({Candidato aprovado.})
19           else
20               write({Candidato reprovado.})
21           end
22
23           write({Pontuacao do candidato: });
24           read(pontuacao);
25           write({Disponibilidade do candidato: });
26           read(disponibilidade);
27       end
28   end
```

program teste5

```
 1   /* Teste do meu compilador */
 2
 3   program teste5
 4       a, b, c, maior is int;
 5       outro is char;
 6
 7   begin
 8       repeat
 9           write({A});
10           read(a);
11           write({B});
12           read(b);
13           write({C});
14           read(c);
15
16
17
18           if ( (a>b) && (a>c) ) end
19               maior = a
20
21           else
22               if (b>c) then
23                   maior = b;
24
25               else
26                   maior = c
27               end
28           end;
29           write({Maior valor:}});
30           write (maior);
31           write ({Outro? (S/N)});
32           read(outro);
33       until (outro == 'N' || outro == 'n)
34   end
```

program order

```
 1   program order
 2
 3     num1, num2, temp is int;
 4
 5     read(num1);
 6     read(num2);
 7
 8     if num1 > num2 then
 9       temp is num1;
10       num1 is num2;
11       num2 is temp;
12     end;
13
14     write({Menor: });
15     write(num1);
16     write({Maior: });
17     write(num2);
18   end.
```

program factorial

```
 1   program factorial
 2
 3     num, fact is int;
 4
 5     read(num);
 6
 7     fact is 1;
 8     i is 1;
 9
10     while i <= num do
11       fact is fact * i;
12       i is i + 1;
13     end;
14
15     write(fact);
16   end.
```

Resultados

Os programas mencionados acima, foram processados pelo analisador léxico implementado, e podemos verificar a sequência de tokens identificados:

program teste1

```
 1   Lexeme: programa | Tag:ID
 2   Lexeme: teste1 | Tag:ID
 3   Lexeme: a | Tag:ID
 4   COMMA
 5   Lexeme: b | Tag:ID
 6   Lexeme: is | Tag:IS
 7   Lexeme: int | Tag:INT
 8   SEMI_COLON
 9   Lexeme: result | Tag:ID
10   Lexeme: is | Tag:IS
11   Lexeme: int | Tag:INT
12   SEMI_COLON
13   Lexeme: a | Tag:ID
14   COMMA
15   Lexeme: x | Tag:ID
16   Lexeme: is | Tag:IS
17   Lexeme: float | Tag:FLOAT
18   SEMI_COLON
19   Lexeme: begin | Tag:BEGIN
20   Lexeme: a | Tag:ID
21   ASSIGN
22   Value: 12 | Tag: CONST_INT
23   Lexeme: a | Tag:ID
24   SEMI_COLON
25   Lexeme: x | Tag:ID
26   ASSIGN
27   INVALID_TOKEN
28   SEMI_COLON
29   Lexeme: read | Tag:READ
30   OPEN_PAR
31   Lexeme: a | Tag:ID
32   CLOSE_PAR
33   SEMI_COLON
34   Lexeme: read | Tag:READ
35   OPEN_PAR
36   Lexeme: b | Tag:ID
37   CLOSE_PAR
38   SEMI_COLON
39   Lexeme: read | Tag:READ
40   OPEN_PAR
41   Lexeme: c | Tag:ID
42   CLOSE_PAR
```

```
 43   Lexeme: result | Tag:ID
 44   ASSIGN
 45   OPEN_PAR
 46   Lexeme: a | Tag:ID
 47   MUL
 48   Lexeme: b | Tag:ID
 49   ADD
 50   Value: 1 | Tag: CONST_INT
 51   CLOSE_PAR
 52   DIV
 53   OPEN_PAR
 54   Lexeme: c | Tag:ID
 55   ADD
 56   Value: 2 | Tag: CONST_INT
 57   CLOSE_PAR
 58   SEMI_COLON
 59   Lexeme: write | Tag:WRITE
 60   Lexeme: Resultado:  | Tag:LITERAL
 61   SEMI_COLON
 62   Lexeme: write | Tag:WRITE
 63   OPEN_PAR
 64   Lexeme: result | Tag:ID
 65   CLOSE_PAR
 66   SEMI_COLON
 67   Lexeme: end | Tag:END
 68   DOT
 69   END_OF_FILE
```

Conforme identificado na linha 27, foi encontrado um token não esperado pela linguagem. Sendo assim, a linha 10 do código fonte apresenta a seguinte correção:

```
x = 12.1;
```

A nova saída:

```
 1    Lexeme: programa | Tag:ID
 2    Lexeme: teste1 | Tag:ID
 3    Lexeme: a | Tag:ID
 4    COMMA
 5    Lexeme: b | Tag:ID
 6    Lexeme: is | Tag:IS
 7    Lexeme: int | Tag:INT
 8    SEMI_COLON
 9    Lexeme: result | Tag:ID
10    Lexeme: is | Tag:IS
11    Lexeme: int | Tag:INT
12    SEMI_COLON
```

```
13    Lexeme: a | Tag:ID
14    COMMA
15    Lexeme: x | Tag:ID
16    Lexeme: is | Tag:IS
17    Lexeme: float | Tag:FLOAT
18    SEMI_COLON
19    Lexeme: begin | Tag:BEGIN
20    Lexeme: a | Tag:ID
21    ASSIGN
22    Value: 12 | Tag: CONST_INT
23    Lexeme: a | Tag:ID
24    SEMI_COLON
25    Lexeme: x | Tag:ID
26    ASSIGN
27    Value: 12.1 | Tag: CONST_FLOAT
28    SEMI_COLON
29    Lexeme: read | Tag:READ
30    OPEN_PAR
31    Lexeme: a | Tag:ID
32    CLOSE_PAR
33    SEMI_COLON
34    Lexeme: read | Tag:READ
35    OPEN_PAR
36    Lexeme: b | Tag:ID
37    CLOSE_PAR
38    SEMI_COLON
39    Lexeme: read | Tag:READ
40    OPEN_PAR
41    Lexeme: c | Tag:ID
42    CLOSE_PAR
43    Lexeme: result | Tag:ID
44    ASSIGN
45    OPEN_PAR
46    Lexeme: a | Tag:ID
47    MUL
48    Lexeme: b | Tag:ID
49    ADD
50    Value: 1 | Tag: CONST_INT
51    CLOSE_PAR
52    DIV
53    OPEN_PAR
54    Lexeme: c | Tag:ID
55    ADD
56    Value: 2 | Tag: CONST_INT
57    CLOSE_PAR
58    SEMI_COLON
59    Lexeme: write | Tag:WRITE
60    Lexeme: Resultado:  | Tag:LITERAL
61    SEMI_COLON
62    Lexeme: write | Tag:WRITE
63    OPEN_PAR
64    Lexeme: result | Tag:ID
```

```
65    CLOSE_PAR
66    SEMI_COLON
67    Lexeme: end | Tag:END
68    DOT
69    END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou mais nenhum erro.

program teste2

```
 1    Lexeme: program | Tag:PROGRAM
 2    Lexeme: teste2 | Tag:ID
 3    Lexeme: a | Tag:ID
 4    COMMA
 5    Lexeme: b | Tag:ID
 6    COMMA
 7    Lexeme: c | Tag:ID
 8    COLON
 9    Lexeme: int | Tag:INT
10    SEMI_COLON
11    Lexeme: d | Tag:ID
12    COMMA
13    NOT_EXPECTED
14    Lexeme: var | Tag:ID
15    COLON
16    Lexeme: float | Tag:FLOAT
17    SEMI_COLON
18    Lexeme: teste2 | Tag:ID
19    ASSIGN
20    Value: 1 | Tag: CONST_INT
21    SEMI_COLON
22    Lexeme: Read | Tag:ID
23    OPEN_PAR
24    Lexeme: a | Tag:ID
25    CLOSE_PAR
26    SEMI_COLON
27    Lexeme: b | Tag:ID
28    ASSIGN
29    Lexeme: a | Tag:ID
30    MUL
31    Lexeme: a | Tag:ID
32    SEMI_COLON
33    Lexeme: c | Tag:ID
34    ASSIGN
35    Lexeme: b | Tag:ID
36    ADD
37    Lexeme: a | Tag:ID
38    DIV
39    Value: 2 | Tag: CONST_INT
40    MUL
```

```
41   OPEN_PAR
42   Value: 35 | Tag: CONST_INT
43   DIV
44   Lexeme: b | Tag:ID
45   CLOSE_PAR
46   SEMI_COLON
47   Lexeme: write | Tag:WRITE
48   Lexeme: c | Tag:ID
49   SEMI_COLON
50   Lexeme: val | Tag:ID
51   COLON
52   ASSIGN
53   Value: 34.2 | Tag: CONST_FLOAT
54   Lexeme: c | Tag:ID
55   ASSIGN
56   Lexeme: val | Tag:ID
57   ADD
58   Value: 2.2 | Tag: CONST_FLOAT
59   ADD
60   Lexeme: a | Tag:ID
61   SEMI_COLON
62   Lexeme: write | Tag:WRITE
63   OPEN_PAR
64   Lexeme: val | Tag:ID
65   CLOSE_PAR
66   Lexeme: end | Tag:END
67   DOT
68   END_OF_FILE
```

Conforme identificado na linha 13, foi encontrado um token não esperado pela linguagem. Sendo assim, a linha 4 do código fonte apresenta a seguinte correção:

```
d, var_: float;
```

A nova saída:

```
 1   Lexeme: program | Tag:PROGRAM
 2   Lexeme: teste2 | Tag:ID
 3   Lexeme: a | Tag:ID
 4   COMMA
 5   Lexeme: b | Tag:ID
 6   COMMA
 7   Lexeme: c | Tag:ID
 8   COLON
 9   Lexeme: int | Tag:INT
10   SEMI_COLON
11   Lexeme: d | Tag:ID
```

```
12   COMMA
13   Lexeme: var_ | Tag:ID
14   COLON
15   Lexeme: float | Tag:FLOAT
16   SEMI_COLON
17   Lexeme: teste2 | Tag:ID
18   ASSIGN
19   Value: 1 | Tag: CONST_INT
20   SEMI_COLON
21   Lexeme: Read | Tag:ID
22   OPEN_PAR
23   Lexeme: a | Tag:ID
24   CLOSE_PAR
25   SEMI_COLON
26   Lexeme: b | Tag:ID
27   ASSIGN
28   Lexeme: a | Tag:ID
29   MUL
30   Lexeme: a | Tag:ID
31   SEMI_COLON
32   Lexeme: c | Tag:ID
33   ASSIGN
34   Lexeme: b | Tag:ID
35   ADD
36   Lexeme: a | Tag:ID
37   DIV
38   Value: 2 | Tag: CONST_INT
39   MUL
40   OPEN_PAR
41   Value: 35 | Tag: CONST_INT
42   DIV
43   Lexeme: b | Tag:ID
44   CLOSE_PAR
45   SEMI_COLON
46   Lexeme: write | Tag:WRITE
47   Lexeme: c | Tag:ID
48   SEMI_COLON
49   Lexeme: val | Tag:ID
50   COLON
51   ASSIGN
52   Value: 34.2 | Tag: CONST_FLOAT
53   Lexeme: c | Tag:ID
54   ASSIGN
55   Lexeme: val | Tag:ID
56   ADD
57   Value: 2.2 | Tag: CONST_FLOAT
58   ADD
59   Lexeme: a | Tag:ID
60   SEMI_COLON
61   Lexeme: write | Tag:WRITE
62   OPEN_PAR
63   Lexeme: val | Tag:ID
```

```
 64   CLOSE_PAR
 65   Lexeme: end | Tag:END
 66   DOT
 67   END_OF_FILE
```

Ademais, podemos também ter a declaração variável do seguinte modo, onde o caractere '_' está presente no meio da declaração da variável:

```
d, v_ar: float;
```

A nova saída:

```
  1   Lexeme: program | Tag:PROGRAM
  2   Lexeme: teste2 | Tag:ID
  3   Lexeme: a | Tag:ID
  4   COMMA
  5   Lexeme: b | Tag:ID
  6   COMMA
  7   Lexeme: c | Tag:ID
  8   COLON
  9   Lexeme: int | Tag:INT
 10   SEMI_COLON
 11   Lexeme: d | Tag:ID
 12   COMMA
 13   Lexeme: v_ar | Tag:ID
 14   COLON
 15   Lexeme: float | Tag:FLOAT
 16   SEMI_COLON
 17   Lexeme: teste2 | Tag:ID
 18   ASSIGN
 19   Value: 1 | Tag: CONST_INT
 20   SEMI_COLON
 21   Lexeme: Read | Tag:ID
 22   OPEN_PAR
 23   Lexeme: a | Tag:ID
 24   CLOSE_PAR
 25   SEMI_COLON
 26   Lexeme: b | Tag:ID
 27   ASSIGN
 28   Lexeme: a | Tag:ID
 29   MUL
 30   Lexeme: a | Tag:ID
 31   SEMI_COLON
 32   Lexeme: c | Tag:ID
 33   ASSIGN
 34   Lexeme: b | Tag:ID
 35   ADD
```

```
36   Lexeme: a | Tag:ID
37   DIV
38   Value: 2 | Tag: CONST_INT
39   MUL
40   OPEN_PAR
41   Value: 35 | Tag: CONST_INT
42   DIV
43   Lexeme: b | Tag:ID
44   CLOSE_PAR
45   SEMI_COLON
46   Lexeme: write | Tag:WRITE
47   Lexeme: c | Tag:ID
48   SEMI_COLON
49   Lexeme: val | Tag:ID
50   COLON
51   ASSIGN
52   Value: 34.2 | Tag: CONST_FLOAT
53   Lexeme: c | Tag:ID
54   ASSIGN
55   Lexeme: val | Tag:ID
56   ADD
57   Value: 2.2 | Tag: CONST_FLOAT
58   ADD
59   Lexeme: a | Tag:ID
60   SEMI_COLON
61   Lexeme: write | Tag:WRITE
62   OPEN_PAR
63   Lexeme: val | Tag:ID
64   CLOSE_PAR
65   Lexeme: end | Tag:END
66   DOT
67   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou mais nenhum erro.

program UNDEFINED

```
 1   Lexeme: program | Tag:PROGRAM
 2   Lexeme: a | Tag:ID
 3   COMMA
 4   Lexeme: aux | Tag:ID
 5   Lexeme: is | Tag:IS
 6   Lexeme: int | Tag:INT
 7   SEMI_COLON
 8   Lexeme: b | Tag:ID
 9   Lexeme: is | Tag:IS
10   Lexeme: float | Tag:FLOAT
11   Lexeme: begin | Tag:BEGIN
12   Lexeme: b | Tag:ID
13   ASSIGN
```

```
14   Value: 0 | Tag: CONST_INT
15   SEMI_COLON
16   Lexeme: in | Tag:ID
17   OPEN_PAR
18   Lexeme: a | Tag:ID
19   CLOSE_PAR
20   SEMI_COLON
21   Lexeme: in | Tag:ID
22   OPEN_PAR
23   Lexeme: b | Tag:ID
24   CLOSE_PAR
25   SEMI_COLON
26   Lexeme: if | Tag:IF
27   OPEN_PAR
28   Lexeme: a | Tag:ID
29   GREATER
30   Lexeme: b | Tag:ID
31   CLOSE_PAR
32   Lexeme: then | Tag:THEN
33   Lexeme: aux | Tag:ID
34   ASSIGN
35   Lexeme: b | Tag:ID
36   SEMI_COLON
37   Lexeme: b | Tag:ID
38   ASSIGN
39   Lexeme: a | Tag:ID
40   SEMI_COLON
41   Lexeme: a | Tag:ID
42   ASSIGN
43   Lexeme: aux | Tag:ID
44   Lexeme: end | Tag:END
45   SEMI_COLON
46   Lexeme: write | Tag:WRITE
47   OPEN_PAR
48   Lexeme: a | Tag:ID
49   SEMI_COLON
50   Lexeme: write | Tag:WRITE
51   OPEN_PAR
52   Lexeme: b | Tag:ID
53   CLOSE_PAR
54   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou nenhum erro.

program teste4

```
1   Lexeme: programa | Tag:ID
2   Lexeme: teste4 | Tag:ID
3   utils.CompilerException: Linha: 28
4   Erro:
5   Comentário aberto.
```

Conforme identificado na linha 3, foi encontrado um erro léxico. Sendo assim, a linha 3 do código fonte apresenta a seguinte correção:

```
/* Teste4 do meu compilador */
```

A nova saída:

```
 1   Lexeme: programa | Tag:ID
 2   Lexeme: teste4 | Tag:ID
 3   Lexeme: pontuacao | Tag:ID
 4   COMMA
 5   Lexeme: pontuacaoMaxina | Tag:ID
 6   COMMA
 7   Lexeme: disponibilidade | Tag:ID
 8   Lexeme: is | Tag:IS
 9   Lexeme: inteiro | Tag:ID
10   SEMI_COLON
11   Lexeme: pontuacaoMinima | Tag:ID
12   Lexeme: is | Tag:IS
13   Lexeme: char | Tag:CHAR
14   SEMI_COLON
15   Lexeme: begin | Tag:BEGIN
16   Lexeme: pontuacaoMinima | Tag:ID
17   ASSIGN
18   Value: 50 | Tag: CONST_INT
19   SEMI_COLON
20   Lexeme: pontuacaoMaxima | Tag:ID
21   ASSIGN
22   Value: 100 | Tag: CONST_INT
23   SEMI_COLON
24   Lexeme: write | Tag:WRITE
25   OPEN_PAR
26   Lexeme: Pontuacao do candidato:  | Tag:LITERAL
27   CLOSE_PAR
28   SEMI_COLON
29   Lexeme: read | Tag:READ
30   OPEN_PAR
31   Lexeme: pontuacao | Tag:ID
32   CLOSE_PAR
```

```
33   SEMI_COLON
34   Lexeme: write | Tag:WRITE
35   OPEN_PAR
36   Lexeme: Disponibilidade do candidato:  | Tag:LITERAL
37   CLOSE_PAR
38   SEMI_COLON
39   Lexeme: read | Tag:READ
40   OPEN_PAR
41   Lexeme: disponibilidade | Tag:ID
42   CLOSE_PAR
43   SEMI_COLON
44   Lexeme: while | Tag:WHILE
45   OPEN_PAR
46   Lexeme: pontuacao | Tag:ID
47   GREATER
48   Value: 0 | Tag: CONST_INT
49   INVALID_TOKEN
50   OPEN_PAR
51   Lexeme: pontuacao | Tag:ID
52   LOWER_EQ
53   Lexeme: pontuacaoMaxima | Tag:ID
54   CLOSE_PAR
55   Lexeme: do | Tag:DO
56   Lexeme: if | Tag:IF
57   OPEN_PAR
58   OPEN_PAR
59   Lexeme: pontuação | Tag:ID
60   GREATER
61   Lexeme: pontuacaoMinima | Tag:ID
62   CLOSE_PAR
63   AND
64   OPEN_PAR
65   Lexeme: disponibilidade | Tag:ID
66   EQUALS
67   Value: 1 | Tag: CONST_INT
68   CLOSE_PAR
69   CLOSE_PAR
70   Lexeme: then | Tag:THEN
71   Lexeme: write | Tag:WRITE
72   OPEN_PAR
73   Lexeme: Candidato aprovado. | Tag:LITERAL
74   CLOSE_PAR
75   Lexeme: else | Tag:ELSE
76   Lexeme: write | Tag:WRITE
77   OPEN_PAR
78   Lexeme: Candidato reprovado. | Tag:LITERAL
79   CLOSE_PAR
80   Lexeme: end | Tag:END
81   Lexeme: write | Tag:WRITE
82   OPEN_PAR
83   Lexeme: Pontuacao do candidato:  | Tag:LITERAL
84   CLOSE_PAR
```

```
 85   SEMI_COLON
 86   Lexeme: read | Tag:READ
 87   OPEN_PAR
 88   Lexeme: pontuacao | Tag:ID
 89   CLOSE_PAR
 90   SEMI_COLON
 91   Lexeme: write | Tag:WRITE
 92   OPEN_PAR
 93   Lexeme: Disponibilidade do candidato:  | Tag:LITERAL
 94   CLOSE_PAR
 95   SEMI_COLON
 96   Lexeme: read | Tag:READ
 97   OPEN_PAR
 98   Lexeme: disponibilidade | Tag:ID
 99   CLOSE_PAR
100   SEMI_COLON
101   Lexeme: end | Tag:END
102   Lexeme: end | Tag:END
103   END_OF_FILE
```

Conforme identificado na linha 49, foi encontrado um token não esperado pela linguagem. Sendo assim, a linha 16 do código fonte apresenta a seguinte correção:

```
while (pontuacao>0 && (pontuacao<=pontuacaoMaxima) do
```

A nova saída:

```
 1   Lexeme: programa | Tag:ID
 2   Lexeme: teste4 | Tag:ID
 3   Lexeme: pontuacao | Tag:ID
 4   COMMA
 5   Lexeme: pontuacaoMaxina | Tag:ID
 6   COMMA
 7   Lexeme: disponibilidade | Tag:ID
 8   Lexeme: is | Tag:IS
 9   Lexeme: inteiro | Tag:ID
10   SEMI_COLON
11   Lexeme: pontuacaoMinima | Tag:ID
12   Lexeme: is | Tag:IS
13   Lexeme: char | Tag:CHAR
14   SEMI_COLON
15   Lexeme: begin | Tag:BEGIN
16   Lexeme: pontuacaoMinima | Tag:ID
17   ASSIGN
18   Value: 50 | Tag: CONST_INT
19   SEMI_COLON
20   Lexeme: pontuacaoMaxima | Tag:ID
```

```
21   ASSIGN
22   Value: 100 | Tag: CONST_INT
23   SEMI_COLON
24   Lexeme: write | Tag:WRITE
25   OPEN_PAR
26   Lexeme: Pontuacao do candidato:  | Tag:LITERAL
27   CLOSE_PAR
28   SEMI_COLON
29   Lexeme: read | Tag:READ
30   OPEN_PAR
31   Lexeme: pontuacao | Tag:ID
32   CLOSE_PAR
33   SEMI_COLON
34   Lexeme: write | Tag:WRITE
35   OPEN_PAR
36   Lexeme: Disponibilidade do candidato:  | Tag:LITERAL
37   CLOSE_PAR
38   SEMI_COLON
39   Lexeme: read | Tag:READ
40   OPEN_PAR
41   Lexeme: disponibilidade | Tag:ID
42   CLOSE_PAR
43   SEMI_COLON
44   Lexeme: while | Tag:WHILE
45   OPEN_PAR
46   Lexeme: pontuacao | Tag:ID
47   GREATER
48   Value: 0 | Tag: CONST_INT
49   AND
50   OPEN_PAR
51   Lexeme: pontuacao | Tag:ID
52   LOWER_EQ
53   Lexeme: pontuacaoMaxima | Tag:ID
54   CLOSE_PAR
55   Lexeme: do | Tag:DO
56   Lexeme: if | Tag:IF
57   OPEN_PAR
58   OPEN_PAR
59   Lexeme: pontuação | Tag:ID
60   GREATER
61   Lexeme: pontuacaoMinima | Tag:ID
62   CLOSE_PAR
63   AND
64   OPEN_PAR
65   Lexeme: disponibilidade | Tag:ID
66   EQUALS
67   Value: 1 | Tag: CONST_INT
68   CLOSE_PAR
69   CLOSE_PAR
70   Lexeme: then | Tag:THEN
71   Lexeme: write | Tag:WRITE
72   OPEN_PAR
```

```
 73   Lexeme: Candidato aprovado. | Tag:LITERAL
 74   CLOSE_PAR
 75   Lexeme: else | Tag:ELSE
 76   Lexeme: write | Tag:WRITE
 77   OPEN_PAR
 78   Lexeme: Candidato reprovado. | Tag:LITERAL
 79   CLOSE_PAR
 80   Lexeme: end | Tag:END
 81   Lexeme: write | Tag:WRITE
 82   OPEN_PAR
 83   Lexeme: Pontuacao do candidato:  | Tag:LITERAL
 84   CLOSE_PAR
 85   SEMI_COLON
 86   Lexeme: read | Tag:READ
 87   OPEN_PAR
 88   Lexeme: pontuacao | Tag:ID
 89   CLOSE_PAR
 90   SEMI_COLON
 91   Lexeme: write | Tag:WRITE
 92   OPEN_PAR
 93   Lexeme: Disponibilidade do candidato:  | Tag:LITERAL
 94   CLOSE_PAR
 95   SEMI_COLON
 96   Lexeme: read | Tag:READ
 97   OPEN_PAR
 98   Lexeme: disponibilidade | Tag:ID
 99   CLOSE_PAR
100   SEMI_COLON
101   Lexeme: end | Tag:END
102   Lexeme: end | Tag:END
103   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou mais nenhum erro.

program teste5

```
 1   Lexeme: program | Tag:PROGRAM
 2   Lexeme: teste5 | Tag:ID
 3   Lexeme: a | Tag:ID
 4   COMMA
 5   Lexeme: b | Tag:ID
 6   COMMA
 7   Lexeme: c | Tag:ID
 8   COMMA
 9   Lexeme: maior | Tag:ID
10   Lexeme: is | Tag:IS
11   Lexeme: int | Tag:INT
12   SEMI_COLON
13   Lexeme: outro | Tag:ID
14   Lexeme: is | Tag:IS
```

```
15   Lexeme: char | Tag:CHAR
16   SEMI_COLON
17   Lexeme: begin | Tag:BEGIN
18   Lexeme: repeat | Tag:REPEAT
19   Lexeme: write | Tag:WRITE
20   OPEN_PAR
21   Lexeme: A | Tag:LITERAL
22   CLOSE_PAR
23   SEMI_COLON
24   Lexeme: read | Tag:READ
25   OPEN_PAR
26   Lexeme: a | Tag:ID
27   CLOSE_PAR
28   SEMI_COLON
29   Lexeme: write | Tag:WRITE
30   OPEN_PAR
31   Lexeme: B | Tag:LITERAL
32   CLOSE_PAR
33   SEMI_COLON
34   Lexeme: read | Tag:READ
35   OPEN_PAR
36   Lexeme: b | Tag:ID
37   CLOSE_PAR
38   SEMI_COLON
39   Lexeme: write | Tag:WRITE
40   OPEN_PAR
41   Lexeme: C | Tag:LITERAL
42   CLOSE_PAR
43   SEMI_COLON
44   Lexeme: read | Tag:READ
45   OPEN_PAR
46   Lexeme: c | Tag:ID
47   CLOSE_PAR
48   SEMI_COLON
49   Lexeme: if | Tag:IF
50   OPEN_PAR
51   OPEN_PAR
52   Lexeme: a | Tag:ID
53   GREATER
54   Lexeme: b | Tag:ID
55   CLOSE_PAR
56   AND
57   OPEN_PAR
58   Lexeme: a | Tag:ID
59   GREATER
60   Lexeme: c | Tag:ID
61   CLOSE_PAR
62   CLOSE_PAR
63   Lexeme: end | Tag:END
64   Lexeme: maior | Tag:ID
65   ASSIGN
66   Lexeme: a | Tag:ID
```

```
 67   Lexeme: else | Tag:ELSE
 68   Lexeme: if | Tag:IF
 69   OPEN_PAR
 70   Lexeme: b | Tag:ID
 71   GREATER
 72   Lexeme: c | Tag:ID
 73   CLOSE_PAR
 74   Lexeme: then | Tag:THEN
 75   Lexeme: maior | Tag:ID
 76   ASSIGN
 77   Lexeme: b | Tag:ID
 78   SEMI_COLON
 79   Lexeme: else | Tag:ELSE
 80   Lexeme: maior | Tag:ID
 81   ASSIGN
 82   Lexeme: c | Tag:ID
 83   Lexeme: end | Tag:END
 84   Lexeme: end | Tag:END
 85   SEMI_COLON
 86   Lexeme: write | Tag:WRITE
 87   OPEN_PAR
 88   Lexeme: Maior valor: | Tag:LITERAL
 89   NOT_EXPECTED
 90   CLOSE_PAR
 91   SEMI_COLON
 92   Lexeme: write | Tag:WRITE
 93   OPEN_PAR
 94   Lexeme: maior | Tag:ID
 95   CLOSE_PAR
 96   SEMI_COLON
 97   Lexeme: write | Tag:WRITE
 98   OPEN_PAR
 99   Lexeme: Outro? (S/N) | Tag:LITERAL
100   CLOSE_PAR
101   SEMI_COLON
102   Lexeme: read | Tag:READ
103   OPEN_PAR
104   Lexeme: outro | Tag:ID
105   CLOSE_PAR
106   SEMI_COLON
107   Lexeme: until | Tag:UNTIL
108   OPEN_PAR
109   Lexeme: outro | Tag:ID
110   EQUALS
111   Lexeme: N | Tag:CONST_CHAR
112   OR
113   Lexeme: outro | Tag:ID
114   EQUALS
115   NOT_EXPECTED
116   CLOSE_PAR
117   Lexeme: end | Tag:END
118   END_OF_FILE
```

Conforme identificado na linha 89, foi encontrado um token não esperado pela linguagem. Sendo assim, a linha 29 do código fonte apresenta a seguinte correção:

```
write({Maior valor:});
```

A nova saída:

```
1     Lexeme: program | Tag:PROGRAM
2     Lexeme: teste5 | Tag:ID
3     Lexeme: a | Tag:ID
4     COMMA
5     Lexeme: b | Tag:ID
6     COMMA
7     Lexeme: c | Tag:ID
8     COMMA
9     Lexeme: maior | Tag:ID
10    Lexeme: is | Tag:IS
11    Lexeme: int | Tag:INT
12    SEMI_COLON
13    Lexeme: outro | Tag:ID
14    Lexeme: is | Tag:IS
15    Lexeme: char | Tag:CHAR
16    SEMI_COLON
17    Lexeme: begin | Tag:BEGIN
18    Lexeme: repeat | Tag:REPEAT
19    Lexeme: write | Tag:WRITE
20    OPEN_PAR
21    Lexeme: A | Tag:LITERAL
22    CLOSE_PAR
23    SEMI_COLON
24    Lexeme: read | Tag:READ
25    OPEN_PAR
26    Lexeme: a | Tag:ID
27    CLOSE_PAR
28    SEMI_COLON
29    Lexeme: write | Tag:WRITE
30    OPEN_PAR
31    Lexeme: B | Tag:LITERAL
32    CLOSE_PAR
33    SEMI_COLON
34    Lexeme: read | Tag:READ
35    OPEN_PAR
36    Lexeme: b | Tag:ID
37    CLOSE_PAR
38    SEMI_COLON
39    Lexeme: write | Tag:WRITE
40    OPEN_PAR
```

```
41    Lexeme: C | Tag:LITERAL
42    CLOSE_PAR
43    SEMI_COLON
44    Lexeme: read | Tag:READ
45    OPEN_PAR
46    Lexeme: c | Tag:ID
47    CLOSE_PAR
48    SEMI_COLON
49    Lexeme: if | Tag:IF
50    OPEN_PAR
51    OPEN_PAR
52    Lexeme: a | Tag:ID
53    GREATER
54    Lexeme: b | Tag:ID
55    CLOSE_PAR
56    AND
57    OPEN_PAR
58    Lexeme: a | Tag:ID
59    GREATER
60    Lexeme: c | Tag:ID
61    CLOSE_PAR
62    CLOSE_PAR
63    Lexeme: end | Tag:END
64    Lexeme: maior | Tag:ID
65    ASSIGN
66    Lexeme: a | Tag:ID
67    Lexeme: else | Tag:ELSE
68    Lexeme: if | Tag:IF
69    OPEN_PAR
70    Lexeme: b | Tag:ID
71    GREATER
72    Lexeme: c | Tag:ID
73    CLOSE_PAR
74    Lexeme: then | Tag:THEN
75    Lexeme: maior | Tag:ID
76    ASSIGN
77    Lexeme: b | Tag:ID
78    SEMI_COLON
79    Lexeme: else | Tag:ELSE
80    Lexeme: maior | Tag:ID
81    ASSIGN
82    Lexeme: c | Tag:ID
83    Lexeme: end | Tag:END
84    Lexeme: end | Tag:END
85    SEMI_COLON
86    Lexeme: write | Tag:WRITE
87    OPEN_PAR
88    Lexeme: Maior valor: | Tag:LITERAL
89    CLOSE_PAR
90    SEMI_COLON
91    Lexeme: write | Tag:WRITE
92    OPEN_PAR
```

```
 93    Lexeme: maior | Tag:ID
 94    CLOSE_PAR
 95    SEMI_COLON
 96    Lexeme: write | Tag:WRITE
 97    OPEN_PAR
 98    Lexeme: Outro? (S/N) | Tag:LITERAL
 99    CLOSE_PAR
100    SEMI_COLON
101    Lexeme: read | Tag:READ
102    OPEN_PAR
103    Lexeme: outro | Tag:ID
104    CLOSE_PAR
105    SEMI_COLON
106    Lexeme: until | Tag:UNTIL
107    OPEN_PAR
108    Lexeme: outro | Tag:ID
109    EQUALS
110    Lexeme: N | Tag:CONST_CHAR
111    OR
112    Lexeme: outro | Tag:ID
113    EQUALS
114    NOT_EXPECTED
115    CLOSE_PAR
116    Lexeme: end | Tag:END
117    END_OF_FILE
```

Conforme identificado na linha 114, foi encontrado um token não esperado pela linguagem. Sendo assim, a linha 33 do código fonte apresenta a seguinte correção:

```
until (outro == 'N' || outro == 'n')
```

A nova saída:

```
 1    Lexeme: program | Tag:PROGRAM
 2    Lexeme: teste5 | Tag:ID
 3    Lexeme: a | Tag:ID
 4    COMMA
 5    Lexeme: b | Tag:ID
 6    COMMA
 7    Lexeme: c | Tag:ID
 8    COMMA
 9    Lexeme: maior | Tag:ID
10    Lexeme: is | Tag:IS
11    Lexeme: int | Tag:INT
12    SEMI_COLON
13    Lexeme: outro | Tag:ID
```

```
14   Lexeme: is | Tag:IS
15   Lexeme: char | Tag:CHAR
16   SEMI_COLON
17   Lexeme: begin | Tag:BEGIN
18   Lexeme: repeat | Tag:REPEAT
19   Lexeme: write | Tag:WRITE
20   OPEN_PAR
21   Lexeme: A | Tag:LITERAL
22   CLOSE_PAR
23   SEMI_COLON
24   Lexeme: read | Tag:READ
25   OPEN_PAR
26   Lexeme: a | Tag:ID
27   CLOSE_PAR
28   SEMI_COLON
29   Lexeme: write | Tag:WRITE
30   OPEN_PAR
31   Lexeme: B | Tag:LITERAL
32   CLOSE_PAR
33   SEMI_COLON
34   Lexeme: read | Tag:READ
35   OPEN_PAR
36   Lexeme: b | Tag:ID
37   CLOSE_PAR
38   SEMI_COLON
39   Lexeme: write | Tag:WRITE
40   OPEN_PAR
41   Lexeme: C | Tag:LITERAL
42   CLOSE_PAR
43   SEMI_COLON
44   Lexeme: read | Tag:READ
45   OPEN_PAR
46   Lexeme: c | Tag:ID
47   CLOSE_PAR
48   SEMI_COLON
49   Lexeme: if | Tag:IF
50   OPEN_PAR
51   OPEN_PAR
52   Lexeme: a | Tag:ID
53   GREATER
54   Lexeme: b | Tag:ID
55   CLOSE_PAR
56   AND
57   OPEN_PAR
58   Lexeme: a | Tag:ID
59   GREATER
60   Lexeme: c | Tag:ID
61   CLOSE_PAR
62   CLOSE_PAR
63   Lexeme: end | Tag:END
64   Lexeme: maior | Tag:ID
65   ASSIGN
```

```
 66   Lexeme: a | Tag:ID
 67   Lexeme: else | Tag:ELSE
 68   Lexeme: if | Tag:IF
 69   OPEN_PAR
 70   Lexeme: b | Tag:ID
 71   GREATER
 72   Lexeme: c | Tag:ID
 73   CLOSE_PAR
 74   Lexeme: then | Tag:THEN
 75   Lexeme: maior | Tag:ID
 76   ASSIGN
 77   Lexeme: b | Tag:ID
 78   SEMI_COLON
 79   Lexeme: else | Tag:ELSE
 80   Lexeme: maior | Tag:ID
 81   ASSIGN
 82   Lexeme: c | Tag:ID
 83   Lexeme: end | Tag:END
 84   Lexeme: end | Tag:END
 85   SEMI_COLON
 86   Lexeme: write | Tag:WRITE
 87   OPEN_PAR
 88   Lexeme: Maior valor: | Tag:LITERAL
 89   CLOSE_PAR
 90   SEMI_COLON
 91   Lexeme: write | Tag:WRITE
 92   OPEN_PAR
 93   Lexeme: maior | Tag:ID
 94   CLOSE_PAR
 95   SEMI_COLON
 96   Lexeme: write | Tag:WRITE
 97   OPEN_PAR
 98   Lexeme: Outro? (S/N) | Tag:LITERAL
 99   CLOSE_PAR
100   SEMI_COLON
101   Lexeme: read | Tag:READ
102   OPEN_PAR
103   Lexeme: outro | Tag:ID
104   CLOSE_PAR
105   SEMI_COLON
106   Lexeme: until | Tag:UNTIL
107   OPEN_PAR
108   Lexeme: outro | Tag:ID
109   EQUALS
110   Lexeme: N | Tag:CONST_CHAR
111   OR
112   Lexeme: outro | Tag:ID
113   EQUALS
114   Lexeme: n | Tag:CONST_CHAR
115   CLOSE_PAR
116   Lexeme: end | Tag:END
117   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou mais nenhum erro.

program order

```
 1   Lexeme: program | Tag:PROGRAM
 2   Lexeme: order | Tag:ID
 3   Lexeme: num1 | Tag:ID
 4   COMMA
 5   Lexeme: num2 | Tag:ID
 6   COMMA
 7   Lexeme: temp | Tag:ID
 8   Lexeme: is | Tag:IS
 9   Lexeme: int | Tag:INT
10   SEMI_COLON
11   Lexeme: read | Tag:READ
12   OPEN_PAR
13   Lexeme: num1 | Tag:ID
14   CLOSE_PAR
15   SEMI_COLON
16   Lexeme: read | Tag:READ
17   OPEN_PAR
18   Lexeme: num2 | Tag:ID
19   CLOSE_PAR
20   SEMI_COLON
21   Lexeme: if | Tag:IF
22   Lexeme: num1 | Tag:ID
23   GREATER
24   Lexeme: num2 | Tag:ID
25   Lexeme: then | Tag:THEN
26   Lexeme: temp | Tag:ID
27   Lexeme: is | Tag:IS
28   Lexeme: num1 | Tag:ID
29   SEMI_COLON
30   Lexeme: num1 | Tag:ID
31   Lexeme: is | Tag:IS
32   Lexeme: num2 | Tag:ID
33   SEMI_COLON
34   Lexeme: num2 | Tag:ID
35   Lexeme: is | Tag:IS
36   Lexeme: temp | Tag:ID
37   SEMI_COLON
38   Lexeme: end | Tag:END
39   SEMI_COLON
40   Lexeme: write | Tag:WRITE
41   OPEN_PAR
42   Lexeme: Menor:  | Tag:LITERAL
43   CLOSE_PAR
44   SEMI_COLON
45   Lexeme: write | Tag:WRITE
46   OPEN_PAR
47   Lexeme: num1 | Tag:ID
```

```
48   CLOSE_PAR
49   SEMI_COLON
50   Lexeme: write | Tag:WRITE
51   OPEN_PAR
52   Lexeme: Maior:  | Tag:LITERAL
53   CLOSE_PAR
54   SEMI_COLON
55   Lexeme: write | Tag:WRITE
56   OPEN_PAR
57   Lexeme: num2 | Tag:ID
58   CLOSE_PAR
59   SEMI_COLON
60   Lexeme: end | Tag:END
61   DOT
62   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou nenhum erro.

program factorial

```
 1   Lexeme: program | Tag:PROGRAM
 2   Lexeme: factorial | Tag:ID
 3   Lexeme: num | Tag:ID
 4   COMMA
 5   Lexeme: fact | Tag:ID
 6   Lexeme: is | Tag:IS
 7   Lexeme: int | Tag:INT
 8   SEMI_COLON
 9   Lexeme: read | Tag:READ
10   OPEN_PAR
11   Lexeme: num | Tag:ID
12   CLOSE_PAR
13   SEMI_COLON
14   Lexeme: fact | Tag:ID
15   Lexeme: is | Tag:IS
16   Value: 1 | Tag: CONST_INT
17   SEMI_COLON
18   Lexeme: i | Tag:ID
19   Lexeme: is | Tag:IS
20   Value: 1 | Tag: CONST_INT
21   SEMI_COLON
22   Lexeme: while | Tag:WHILE
23   Lexeme: i | Tag:ID
24   LOWER_EQ
25   Lexeme: num | Tag:ID
26   Lexeme: do | Tag:DO
27   Lexeme: fact | Tag:ID
28   Lexeme: is | Tag:IS
29   Lexeme: fact | Tag:ID
30   MUL
```

```
31   Lexeme: i | Tag:ID
32   SEMI_COLON
33   Lexeme: i | Tag:ID
34   Lexeme: is | Tag:IS
35   Lexeme: i | Tag:ID
36   ADD
37   Value: 1 | Tag: CONST_INT
38   SEMI_COLON
39   Lexeme: end | Tag:END
40   SEMI_COLON
41   Lexeme: write | Tag:WRITE
42   OPEN_PAR
43   Lexeme: fact | Tag:ID
44   CLOSE_PAR
45   SEMI_COLON
46   Lexeme: end | Tag:END
47   DOT
48   END_OF_FILE
```

Temos assim, que a execução da leitura de tokens não apresentou nenhum erro.

Conclusão

O trabalho de implementação do analisador léxico foi realizado com sucesso, permitindo a análise de casos de teste e a impressão da lista de tokens encontrados e da tabela de símbolos correspondente. Além disso, observamos que alguns códigos fontes apresentam erros não léxicos, e sendo assim, para o analisador léxico não foi encontrado nenhum erro, mas é esperado ser identificado nos próximos passos da execução do compilador.

O analisador léxico é uma parte fundamental do processo de compilação de programas, pois é responsável por identificar e separar os elementos da linguagem de programação em unidades significativas, facilitando a posterior análise semântica e a geração de código. A implementação de um analisador léxico eficiente e preciso é crucial para o desenvolvimento de sistemas de software robustos e confiáveis.

Com esse trabalho, foi possível adquirir conhecimentos e habilidades importantes na área de compiladores e linguagens de programação, que certamente serão úteis em projetos futuros.