CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Engenharia de Computação

Trabalho Prático - Compiladores

Filipe da Silva Rocha (20193003809)
Matheus Freire Henrique Fonseca (20203002786)
Vitor Laguardia Xavier (201712060554)

Belo Horizonte
Novembro de 2024

1. Sumário

2. Forma de uso do compilador

Para executar o compilador, é necessário compilar o projeto usando o comando “javac Compiler.java” e após isso executá-lo passando algum arquivo de teste como entrada “java Compiler tests/test1.txt”. Como exposto no exemplo, os testes estão na pasta “tests”.

3. Abordagem utilizada e principais classes

Criamos a classe Compiler que funcionará como main e receberá o arquivo .txt de teste para instanciar o analisador léxico e printar os tokens, tabela de símbolos e saída do compilador. Além disso, o analisador léxico é estruturado nas seguintes classes:

Token - irá armazenar um atributo do tipo Tag e irá especificar os tokens a partir das classes filhas listadas abaixo.

StringConst, IntegerConst, FloatConst e Word - estendem a classe token e cada um possui atributos que armazenam valores de suas determinadas categorias.

Tag - estruturado como uma classe enum que irá designar valores para cada token gerado.

SymbolTable - armazena informações sobre os identificadores lidos. Além disso, é nessa classe que armazenamos as palavras reservadas da linguagem.

Error - Ela herda a classe Token, portanto é um Token de Erro. Essa classe é chamada quando há algum erro encontrado no código e nela é estruturada a mensagem que especifica o erro e o local em que ele acontece.

Position - guarda o número de linhas do código e é utilizado no retorno da linha de erro.

Lexer - essa é a classe mais importante do projeto, em que é implementado a estrutura do analisador léxico em si. Nela se abre o arquivo fonte e lê-se caractere a caractere, faz-se o reconhecimento dos tokens e reporta quando existem erros léxicos, tokens inválidos ou caracteres inesperados. Além disso, se implementa toda a lógica de leitura de comentários e as chamadas da SymbolTable para armazenar ou recuperar as palavras reservadas e identificadores.

4. Execução dos testes

4.1. Teste 1

Entrada:

```
start
  float a;
  int b, result;
  float a,x,total;

  a = 2.0;
  x = .1;
  scan (b);
  scan (y)
  result = (a*b ++ 1) % a;
  print {Resultado: };
  print (result);
  print ({Total: });
  total = y / x;
  print ({Total: };
  print (total);
exit
```

Saída:

```
Token: START | Lexeme: start
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: b
Token: COMMA
Token: IDENTIFIER | Lexeme: result
Token: SEMICOLON
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: x
Token: COMMA
Token: IDENTIFIER | Lexeme: total
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: FLOAT_CONST | Value: 2.0
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: x
Token: ASSIGN
Error: Unexpected token: '.' at line 7
Token: INT_CONST | Value: 1
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: y
Token: CLOSE_PAR
Token: IDENTIFIER | Lexeme: result
Token: ASSIGN
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: b
Token: PLUS
Token: PLUS
```

```
Token: INT_CONST | Value: 1
Token: CLOSE_PAR
Token: MOD
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: STRING_CONST | Content: Resultado:
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: result
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Total:
Token: CLOSE_PAR
Token: IDENTIFIER | Lexeme: y
Token: DIV
Token: IDENTIFIER | Lexeme: x
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Total:
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: total
Token: CLOSE_PAR
Token: SEMICOLON
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: y | VALUE: (Token: IDENTIFIER | Lexeme: y)
KEY: x | VALUE: (Token: IDENTIFIER | Lexeme: x)
KEY: result | VALUE: (Token: IDENTIFIER | Lexeme: result)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)
KEY: total | VALUE: (Token: IDENTIFIER | Lexeme: total)

---------- Compilation Result -----------
 Compilation ERROR
-----------------------------------------
```

4.2. Teste 2

Entrada:

```
start
  int: a, c_;
  float d, _e;

  a = 0; d = 3.5
  c = d / 1.2;

  Scan (a);
  Scan (c);
  b = a * a;
  c = b + a * (1 + a*c);
  print ({Resultado: });
  print c;
  d = 34.2
  e = val + 2.2;
  print ({E: });
  print (e);
  a = b + c + a)/2;
```

Saída:

```
Token: START | Lexeme: start
Token: INT | Lexeme: int
Error: Unexpected token: ':' at line 3
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: c_
Token: SEMICOLON
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: d
Token: COMMA
Token: IDENTIFIER | Lexeme: _e
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: INT_CONST | Value: 0
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: d
Token: ASSIGN
Token: FLOAT_CONST | Value: 3.5
Token: IDENTIFIER | Lexeme: c
Token: ASSIGN
Token: IDENTIFIER | Lexeme: d
Token: DIV
Token: FLOAT_CONST | Value: 1.2
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: b
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: c
Token: ASSIGN
```

```
Token: IDENTIFIER | Lexeme: b
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: OPEN_PAR
Token: INT_CONST | Value: 1
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Resultado:
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: IDENTIFIER | Lexeme: c
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: d
Token: ASSIGN
Token: FLOAT_CONST | Value: 34.2
Token: IDENTIFIER | Lexeme: e
Token: ASSIGN
Token: IDENTIFIER | Lexeme: val
Token: PLUS
Token: FLOAT_CONST | Value: 2.2
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: E:
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: e
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: PLUS
```

```
Token: IDENTIFIER | Lexeme: c
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: DIV
Token: INT_CONST | Value: 2
Token: SEMICOLON
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: _e | VALUE: (Token: IDENTIFIER | Lexeme: _e)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: c_ | VALUE: (Token: IDENTIFIER | Lexeme: c_)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: Scan | VALUE: (Token: IDENTIFIER | Lexeme: Scan)
KEY: e | VALUE: (Token: IDENTIFIER | Lexeme: e)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: d | VALUE: (Token: IDENTIFIER | Lexeme: d)
KEY: c | VALUE: (Token: IDENTIFIER | Lexeme: c)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)
KEY: val | VALUE: (Token: IDENTIFIER | Lexeme: val)

---------- Compilation Result -----------
 Compilation ERROR
-----------------------------------------
```

4.3. Teste 3

Entrada:

```
  int pontuacao, pontuacaoMaxina, disponibilidade;
  string pontuacaoMinima;

  disponibilidade = "Sim";
  pontuacaoMinima = 50;
  pontuacaoMaxima = 100;

  /* Entrada de dados
     Verifica aprovação de candidatos
  do
     print({Pontuacao Candidato: });
     scan(pontuacao);
     print({Disponibilidade Candidato: });
     scan(disponibilidade);

     if ((pontuação > pontuacaoMinima) & (disponibilidade=={Sim}) then
      out("Candidato aprovado");
     else
        out({Candidato reprovado})
     end
  while (pontuação >= 0)end
exit
```

Saída:

```
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: pontuacao
Token: COMMA
Token: IDENTIFIER | Lexeme: pontuacaoMaxina
Token: COMMA
Token: IDENTIFIER | Lexeme: disponibilidade
Token: SEMICOLON
Token: STRING | Lexeme: string
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: disponibilidade
Token: ASSIGN
Error: Unexpected token: '"' at line 4
Token: IDENTIFIER | Lexeme: Sim
Error: Unexpected token: '"' at line 4
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: ASSIGN
Token: INT_CONST | Value: 50
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMaxima
Token: ASSIGN
Token: INT_CONST | Value: 100
Token: SEMICOLON
Error: Comment not closed at line 22
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: disponibilidade | VALUE: (Token: IDENTIFIER | Lexeme: disponibilidade)
KEY: pontuacaoMaxina | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMaxina)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: Sim | VALUE: (Token: IDENTIFIER | Lexeme: Sim)
KEY: pontuacao | VALUE: (Token: IDENTIFIER | Lexeme: pontuacao)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: pontuacaoMinima | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMinima)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: pontuacaoMaxima | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMaxima)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)

---------- Compilation Result -----------
 Compilation ERROR
-----------------------------------------
```

4.4. Teste 4

Entrada:

```
start
    Int a, aux$, b;
    string nome, sobrenome, msg;

    print(Nome: );
    scan (nome);
    print({Sobrenome: });
    scan (sobrenome);
    msg = {Ola, } + nome + { } +
sobrenome + {!};
    msg = msg + 1;
    print (msg);

    scan (a);
    scan(b);
    if (a>b) then
      aux = b;
      b = a;
      a = aux;
    end;
    print ({Apos a troca: });
    out(a);
```

```
    out(b)
exit
```

Saída:

```
Token: START | Lexeme: start
Token: IDENTIFIER | Lexeme: Int
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: aux
Error: Unexpected token: '$' at line 2
Token: COMMA
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: STRING | Lexeme: string
Token: IDENTIFIER | Lexeme: nome
Token: COMMA
Token: IDENTIFIER | Lexeme: sobrenome
Token: COMMA
Token: IDENTIFIER | Lexeme: msg
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: Nome
Error: Unexpected token: ':' at line 5
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: nome
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Sobrenome:
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: sobrenome
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: msg
Token: ASSIGN
Token: STRING_CONST | Content: Ola,
Token: PLUS
```

```
Token: IDENTIFIER | Lexeme: nome
Token: PLUS
Token: STRING_CONST | Content:
Token: PLUS
Token: IDENTIFIER | Lexeme: sobrenome
Token: PLUS
Token: STRING_CONST | Content: !
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: msg
Token: ASSIGN
Token: IDENTIFIER | Lexeme: msg
Token: PLUS
Token: INT_CONST | Value: 1
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: msg
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: aux
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: b
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
```

```
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: IDENTIFIER | Lexeme: aux
Token: SEMICOLON
Token: END | Lexeme: end
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Apos a troca:
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table ----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: msg | VALUE: (Token: IDENTIFIER | Lexeme: msg)
KEY: Int | VALUE: (Token: IDENTIFIER | Lexeme: Int)
KEY: sobrenome | VALUE: (Token: IDENTIFIER | Lexeme: sobrenome)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: nome | VALUE: (Token: IDENTIFIER | Lexeme: nome)
KEY: out | VALUE: (Token: IDENTIFIER | Lexeme: out)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: aux | VALUE: (Token: IDENTIFIER | Lexeme: aux)
KEY: Nome | VALUE: (Token: IDENTIFIER | Lexeme: Nome)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)

---------- Compilation Result ----------
 Compilation ERROR
----------------------------------------
```

4.5. Teste 5

Entrada:

```
start
  int a, b, c, maior, outro;

  do
    print({A});
    scan(a);
    print({B});
    scan(b);
    print({C});
    scan(c);
    //Realizacao do teste

    if ( (a>b) && (a>c)  )
      maior = a

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

      else
          maior = c;
      end
    end
    print({Maior valor:}});
    print (maior);
    print ({Outro? });
    scan(outro);
  while (outro >= 0);
exit
```

Saída:

```
Token: START | Lexeme: start
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: b
Token: COMMA
Token: IDENTIFIER | Lexeme: c
Token: COMMA
Token: IDENTIFIER | Lexeme: maior
Token: COMMA
Token: IDENTIFIER | Lexeme: outro
Token: SEMICOLON
Token: DO | Lexeme: do
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: A
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: B
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: C
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
```

```
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: AND
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: CLOSE_PAR
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
Token: IDENTIFIER | Lexeme: else
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: GREATER
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: else
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: c
Token: SEMICOLON
Token: END | Lexeme: end
Token: END | Lexeme: end
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Maior valor:
Error: Unexpected token: '}' at line 24
Token: CLOSE_PAR
Token: SEMICOLON
```

```
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Outro?
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: outro
Token: CLOSE_PAR
Token: SEMICOLON
Token: WHILE | Lexeme: while
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: outro
Token: GREATER_EQ
Token: INT_CONST | Value: 0
Token: CLOSE_PAR
Token: SEMICOLON
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: maior | VALUE: (Token: IDENTIFIER | Lexeme: maior)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: outro | VALUE: (Token: IDENTIFIER | Lexeme: outro)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: else | VALUE: (Token: IDENTIFIER | Lexeme: else)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: c | VALUE: (Token: IDENTIFIER | Lexeme: c)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)

---------- Compilation Result -----------
 Compilation ERROR
-----------------------------------------
```

4.6. Teste 6

Entrada:

```
start
   int par, impar;

   par = 20;
   impar = 5;

   int restoPar = par % 2;
   int restoImpar = impar % 2;

   if(restoPar == 0) then
      print({Numero par!});
   end

    if(restoImpar != 0) then
      print({Numero impar!});
    end

 exit
```

Saída:

```
Token: START | Lexeme: start
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: par
Token: COMMA
Token: IDENTIFIER | Lexeme: impar
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: par
Token: ASSIGN
Token: INT_CONST | Value: 20
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: impar
Token: ASSIGN
Token: INT_CONST | Value: 5
Token: SEMICOLON
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: restoPar
Token: ASSIGN
Token: IDENTIFIER | Lexeme: par
Token: MOD
Token: INT_CONST | Value: 2
Token: SEMICOLON
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: restoImpar
Token: ASSIGN
Token: IDENTIFIER | Lexeme: impar
Token: MOD
Token: INT_CONST | Value: 2
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: restoPar
```

```
Token: NOT_EQ
Token: INT_CONST | Value: 0
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Numero impar!
Token: CLOSE_PAR
Token: SEMICOLON
Token: END | Lexeme: end
Token: EXIT | Lexeme: exit
Token: EOF
```

5. Correção e funcionamento do analisador

5.1. Teste 1 - corrigido

Entrada:

```
start
  float a;
  int b, result;
  float a,x,total;

  a = 2.0;
  x = 0.1;
  scan (b);
  scan (y)
  result = (a*b ++ 1) % a;
  print {Resultado: };
  print (result);
  print ({Total: });
  total = y / x;
  print ({Total: };
  print (total);
exit
```

Saída:

```
Token: START | Lexeme: start
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: b
Token: COMMA
Token: IDENTIFIER | Lexeme: result
Token: SEMICOLON
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: x
Token: COMMA
Token: IDENTIFIER | Lexeme: total
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: FLOAT_CONST | Value: 2.0
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: x
Token: ASSIGN
Token: FLOAT_CONST | Value: 0.1
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
```

```
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: y
Token: CLOSE_PAR
Token: IDENTIFIER | Lexeme: result
Token: ASSIGN
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: b
Token: PLUS
Token: PLUS
Token: INT_CONST | Value: 1
Token: CLOSE_PAR
Token: MOD
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: STRING_CONST | Content: Resultado:
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: result
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Total:
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: total
```

```
Token: ASSIGN
Token: IDENTIFIER | Lexeme: y
Token: DIV
Token: IDENTIFIER | Lexeme: x
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Total:
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: total
Token: CLOSE_PAR
Token: SEMICOLON
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: y | VALUE: (Token: IDENTIFIER | Lexeme: y)
KEY: x | VALUE: (Token: IDENTIFIER | Lexeme: x)
KEY: result | VALUE: (Token: IDENTIFIER | Lexeme: result)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)
KEY: total | VALUE: (Token: IDENTIFIER | Lexeme: total)

---------- Compilation Result -----------
 Compilation SUCCESS!
-----------------------------------------
```

5.2. Teste 2 - corrigido

Entrada:

```
start
  int a, c_;
  float d, _e;

  a = 0; d = 3.5
  c = d / 1.2;

  Scan (a);
  Scan (c);
  b = a * a;
  c = b + a * (1 + a*c);
  print ({Resultado: });
  print c;
  d = 34.2
  e = val + 2.2;
  print ({E: });
  print (e);
  a = b + c + a)/2;
```

Saída:

```
Token: START | Lexeme: start
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: c_
Token: SEMICOLON
Token: FLOAT | Lexeme: float
Token: IDENTIFIER | Lexeme: d
Token: COMMA
Token: IDENTIFIER | Lexeme: _e
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: INT_CONST | Value: 0
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: d
Token: ASSIGN
Token: FLOAT_CONST | Value: 3.5
Token: IDENTIFIER | Lexeme: c
Token: ASSIGN
Token: IDENTIFIER | Lexeme: d
Token: DIV
Token: FLOAT_CONST | Value: 1.2
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Scan
```

```
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: b
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: c
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: OPEN_PAR
Token: INT_CONST | Value: 1
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: MULT
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Resultado:
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: IDENTIFIER | Lexeme: c
```

```
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: d
Token: ASSIGN
Token: FLOAT_CONST | Value: 34.2
Token: IDENTIFIER | Lexeme: e
Token: ASSIGN
Token: IDENTIFIER | Lexeme: val
Token: PLUS
Token: FLOAT_CONST | Value: 2.2
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: E:
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: e
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: PLUS
Token: IDENTIFIER | Lexeme: c
Token: PLUS
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: DIV
Token: INT_CONST | Value: 2
Token: SEMICOLON
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: _e | VALUE: (Token: IDENTIFIER | Lexeme: _e)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: c_ | VALUE: (Token: IDENTIFIER | Lexeme: c_)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: Scan | VALUE: (Token: IDENTIFIER | Lexeme: Scan)
KEY: e | VALUE: (Token: IDENTIFIER | Lexeme: e)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: d | VALUE: (Token: IDENTIFIER | Lexeme: d)
KEY: c | VALUE: (Token: IDENTIFIER | Lexeme: c)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)
KEY: val | VALUE: (Token: IDENTIFIER | Lexeme: val)

---------- Compilation Result -----------
 Compilation SUCCESS!
-----------------------------------------
```

5.3. Teste 3 - corrigido Parte 1

Entrada:

- Correção do comentário não fechado e erros que aparecem no código

```
  int pontuacao, pontuacaoMaxina, disponibilidade;
 string pontuacaoMinima;

 disponibilidade = Sim;
 pontuacaoMinima = 50;
 pontuacaoMaxima = 100;

 /* Entrada de dados */
   Verifica aprovação de candidatos
 do
   print({Pontuacao Candidato: });
   scan(pontuacao);
   print({Disponibilidade Candidato: });
   scan(disponibilidade);

   if ((pontuação > pontuacaoMinima) & (disponibilidade=={Sim}) then
    out("Candidato aprovado");
   else
     out({Candidato reprovado})
   end
 while (pontuação >= 0)end
exit
```

Saída:

```
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: pontuacao
Token: COMMA
Token: IDENTIFIER | Lexeme: pontuacaoMaxina
Token: COMMA
Token: IDENTIFIER | Lexeme: disponibilidade
Token: SEMICOLON
Token: STRING | Lexeme: string
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: disponibilidade
Token: ASSIGN
Token: IDENTIFIER | Lexeme: Sim
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: ASSIGN
Token: INT_CONST | Value: 50
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMaxima
Token: ASSIGN
Token: INT_CONST | Value: 100
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Verifica
Token: IDENTIFIER | Lexeme: aprovação
Token: IDENTIFIER | Lexeme: de
Token: IDENTIFIER | Lexeme: candidatos
Token: DO | Lexeme: do
Token: PRINT | Lexeme: print
Token: OPEN_PAR
```

```
Token: STRING_CONST | Content: Pontuacao Candidato:
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuacao
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Disponibilidade Candidato:
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: disponibilidade
Token: CLOSE_PAR
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuação
Token: GREATER
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: CLOSE_PAR
Error: Invalid character '&' at line 16
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: disponibilidade
Token: EQ
Token: STRING_CONST | Content: Sim
```

```
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: STRING_CONST | Content: Candidato aprovado
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: else
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Error: Unexpected token: '"' at line 19
Token: IDENTIFIER | Lexeme: Candidato
Token: IDENTIFIER | Lexeme: reprovado
Error: Unexpected token: '"' at line 19
Token: CLOSE_PAR
Token: END | Lexeme: end
Token: WHILE | Lexeme: while
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuação
Token: GREATER_EQ
Token: INT_CONST | Value: 0
Token: CLOSE_PAR
Token: END | Lexeme: end
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table -----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: disponibilidade | VALUE: (Token: IDENTIFIER | Lexeme: disponibilidade)
KEY: Verifica | VALUE: (Token: IDENTIFIER | Lexeme: Verifica)
KEY: pontuacaoMaxina | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMaxina)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: Candidato | VALUE: (Token: IDENTIFIER | Lexeme: Candidato)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: aprovação | VALUE: (Token: IDENTIFIER | Lexeme: aprovação)
KEY: Sim | VALUE: (Token: IDENTIFIER | Lexeme: Sim)
KEY: pontuacao | VALUE: (Token: IDENTIFIER | Lexeme: pontuacao)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: candidatos | VALUE: (Token: IDENTIFIER | Lexeme: candidatos)
KEY: pontuacaoMinima | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMinima)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: out | VALUE: (Token: IDENTIFIER | Lexeme: out)
KEY: else | VALUE: (Token: IDENTIFIER | Lexeme: else)
KEY: pontuacaoMaxima | VALUE: (Token: IDENTIFIER | Lexeme: pontuacaoMaxima)
KEY: pontuação | VALUE: (Token: IDENTIFIER | Lexeme: pontuação)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: de | VALUE: (Token: IDENTIFIER | Lexeme: de)
KEY: reprovado | VALUE: (Token: IDENTIFIER | Lexeme: reprovado)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)

---------- Compilation Result -----------
 Compilation ERROR
-----------------------------------------
```

5.4. Teste 3 - corrigido Parte 2

Entrada:

- Correção do comentário não fechado e erros que aparecem no código

```
 int pontuacao, pontuacaoMaxina, disponibilidade;
string pontuacaoMinima;

disponibilidade = Sim;
pontuacaoMinima = 50;
pontuacaoMaxima = 100;

/* Entrada de dados */
  Verifica aprovação de candidatos
do
  print({Pontuacao Candidato: });
  scan(pontuacao);
  print({Disponibilidade Candidato: });
  scan(disponibilidade);

  if ((pontuação > pontuacaoMinima) && (disponibilidade=={Sim}) then
   out({Candidato aprovado});
  else
    out({Candidato reprovado})
  end
```

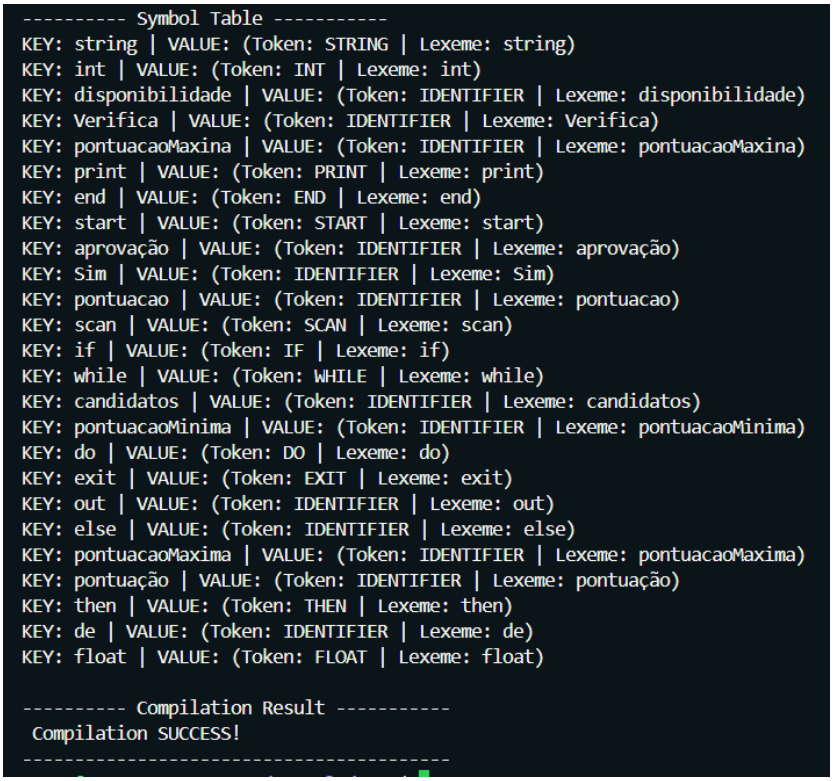
```
  while (pontuação >= 0)end
exit
```

Saída:

```
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: pontuacao
Token: COMMA
Token: IDENTIFIER | Lexeme: pontuacaoMaxina
Token: COMMA
Token: IDENTIFIER | Lexeme: disponibilidade
Token: SEMICOLON
Token: STRING | Lexeme: string
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: disponibilidade
Token: ASSIGN
Token: IDENTIFIER | Lexeme: Sim
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: ASSIGN
Token: INT_CONST | Value: 50
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: pontuacaoMaxima
Token: ASSIGN
Token: INT_CONST | Value: 100
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: Verifica
Token: IDENTIFIER | Lexeme: aprovação
Token: IDENTIFIER | Lexeme: de
Token: IDENTIFIER | Lexeme: candidatos
Token: DO | Lexeme: do
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Pontuacao Candidato:
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuacao
Token: CLOSE_PAR
```

```
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Disponibilidade Candida
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: disponibilidade
Token: CLOSE_PAR
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuação
Token: GREATER
Token: IDENTIFIER | Lexeme: pontuacaoMinima
Token: CLOSE_PAR
Token: AND
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: disponibilidade
Token: EQ
Token: STRING_CONST | Content: Sim
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: STRING_CONST | Content: Candidato aprovado
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: else
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: STRING_CONST | Content: Candidato reprovado
Token: CLOSE_PAR
Token: END | Lexeme: end
Token: WHILE | Lexeme: while
```

```
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: pontuação
Token: GREATER_EQ
Token: INT_CONST | Value: 0
Token: CLOSE_PAR
Token: END | Lexeme: end
Token: EXIT | Lexeme: exit
Token: EOF
```



5.5. Teste 4 - corrigido

Entrada:

```
start
    Int a, aux, b;
    string nome, sobrenome, msg;

    print(Nome);
    scan (nome);
    print({Sobrenome: });
    scan (sobrenome);
    msg = {Ola, } + nome + { } +
sobrenome + {!};
    msg = msg + 1;
    print (msg);

    scan (a);
    scan(b);
    if (a>b) then
      aux = b;
      b = a;
      a = aux;
    end;
    print ({Apos a troca: });
    out(a);
```

```
    out(b)
exit
```

Saída:

```
Token: START | Lexeme: start
Token: IDENTIFIER | Lexeme: Int
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: aux
Token: COMMA
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: STRING | Lexeme: string
Token: IDENTIFIER | Lexeme: nome
Token: COMMA
Token: IDENTIFIER | Lexeme: sobrenome
Token: COMMA
Token: IDENTIFIER | Lexeme: msg
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Nome:
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: nome
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Sobrenome:
Token: CLOSE_PAR
Token: SEMICOLON
```

```
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: sobrenome
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: msg
Token: ASSIGN
Token: STRING_CONST | Content: Ola,
Token: PLUS
Token: IDENTIFIER | Lexeme: nome
Token: PLUS
Token: STRING_CONST | Content:
Token: PLUS
Token: IDENTIFIER | Lexeme: sobrenome
Token: PLUS
Token: STRING_CONST | Content: !
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: msg
Token: ASSIGN
Token: IDENTIFIER | Lexeme: msg
Token: PLUS
Token: INT_CONST | Value: 1
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: msg
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
```

```
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: aux
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: b
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: a
Token: ASSIGN
Token: IDENTIFIER | Lexeme: aux
Token: SEMICOLON
Token: END | Lexeme: end
Token: SEMICOLON
Token: PRINT | Lexeme: print
```

```
Token: OPEN_PAR
Token: STRING_CONST | Content: Apos a troca:
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: out
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: EXIT | Lexeme: exit
Token: EOF
```

```
---------- Symbol Table ----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: msg | VALUE: (Token: IDENTIFIER | Lexeme: msg)
KEY: Int | VALUE: (Token: IDENTIFIER | Lexeme: Int)
KEY: sobrenome | VALUE: (Token: IDENTIFIER | Lexeme: sobrenome)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: nome | VALUE: (Token: IDENTIFIER | Lexeme: nome)
KEY: out | VALUE: (Token: IDENTIFIER | Lexeme: out)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: aux | VALUE: (Token: IDENTIFIER | Lexeme: aux)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)

---------- Compilation Result ----------
 Compilation SUCCESS!
```

5.6. Teste 5 - corrigido

Entrada:

```
start
  int a, b, c, maior, outro;

  do
    print({A});
    scan(a);
    print({B});
    scan(b);
    print({C});
    scan(c);
```

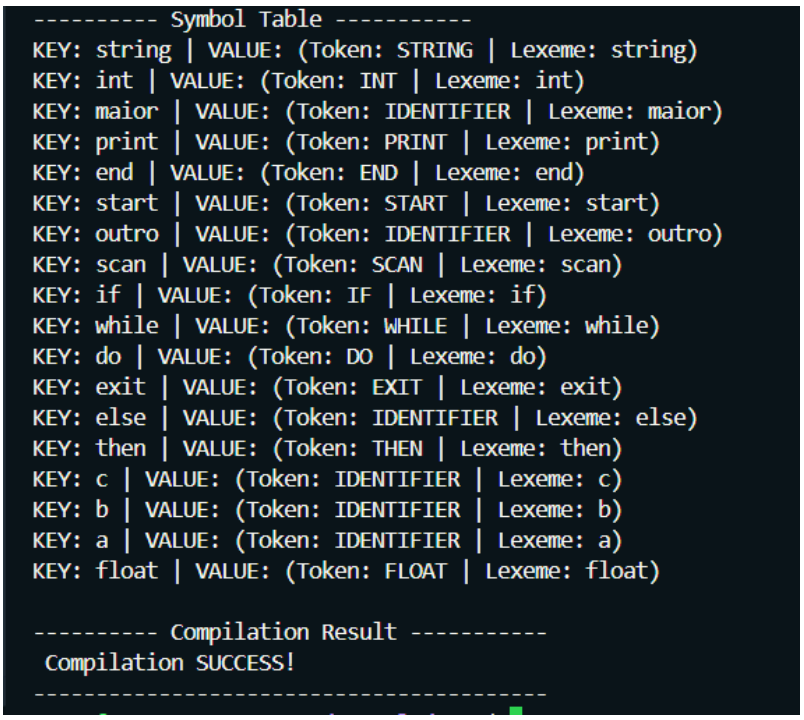
```
    //Realizacao do teste

    if ( (a>b) && (a>c)  )
      maior = a

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

       else
          maior = c;
       end
    end
    print({Maior valor:});
    print (maior);
    print ({Outro? });
    scan(outro);
  while (outro >= 0);
exit
```

```
Token: START | Lexeme: start
Token: INT | Lexeme: int
Token: IDENTIFIER | Lexeme: a
Token: COMMA
Token: IDENTIFIER | Lexeme: b
Token: COMMA
Token: IDENTIFIER | Lexeme: c
Token: COMMA
Token: IDENTIFIER | Lexeme: maior
Token: COMMA
Token: IDENTIFIER | Lexeme: outro
Token: SEMICOLON
Token: DO | Lexeme: do
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: A
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: B
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
```

```
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: C
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: SEMICOLON
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: b
Token: CLOSE_PAR
Token: AND
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: a
Token: GREATER
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: CLOSE_PAR
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: a
Token: IDENTIFIER | Lexeme: else
Token: IF | Lexeme: if
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: b
```

```
Token: GREATER
Token: IDENTIFIER | Lexeme: c
Token: CLOSE_PAR
Token: THEN | Lexeme: then
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: b
Token: SEMICOLON
Token: IDENTIFIER | Lexeme: else
Token: IDENTIFIER | Lexeme: maior
Token: ASSIGN
Token: IDENTIFIER | Lexeme: c
Token: SEMICOLON
Token: END | Lexeme: end
Token: END | Lexeme: end
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Maior valor:
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: maior
Token: CLOSE_PAR
Token: SEMICOLON
Token: PRINT | Lexeme: print
Token: OPEN_PAR
Token: STRING_CONST | Content: Outro?
Token: CLOSE_PAR
Token: SEMICOLON
Token: SCAN | Lexeme: scan
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: outro
```

```
Token: CLOSE_PAR
Token: SEMICOLON
Token: WHILE | Lexeme: while
Token: OPEN_PAR
Token: IDENTIFIER | Lexeme: outro
Token: GREATER_EQ
Token: INT_CONST | Value: 0
Token: CLOSE_PAR
Token: SEMICOLON
Token: EXIT | Lexeme: exit
Token: EOF
```


---------- Symbol Table ----------
KEY: string | VALUE: (Token: STRING | Lexeme: string)
KEY: int | VALUE: (Token: INT | Lexeme: int)
KEY: maior | VALUE: (Token: IDENTIFIER | Lexeme: maior)
KEY: print | VALUE: (Token: PRINT | Lexeme: print)
KEY: end | VALUE: (Token: END | Lexeme: end)
KEY: start | VALUE: (Token: START | Lexeme: start)
KEY: outro | VALUE: (Token: IDENTIFIER | Lexeme: outro)
KEY: scan | VALUE: (Token: SCAN | Lexeme: scan)
KEY: if | VALUE: (Token: IF | Lexeme: if)
KEY: while | VALUE: (Token: WHILE | Lexeme: while)
KEY: do | VALUE: (Token: DO | Lexeme: do)
KEY: exit | VALUE: (Token: EXIT | Lexeme: exit)
KEY: else | VALUE: (Token: IDENTIFIER | Lexeme: else)
KEY: then | VALUE: (Token: THEN | Lexeme: then)
KEY: c | VALUE: (Token: IDENTIFIER | Lexeme: c)
KEY: b | VALUE: (Token: IDENTIFIER | Lexeme: b)
KEY: a | VALUE: (Token: IDENTIFIER | Lexeme: a)
KEY: float | VALUE: (Token: FLOAT | Lexeme: float)
---------- Compilation Result ----------
Compilation SUCCESS!
