

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DISCIPLINA DE COMPILADORES
KESSIA MARQUES

BRUNA GOMES CAMILO
DIÊGO SIMÕES MARIA
MARINA BERNARDES

COMPILADOR PARTE 1
IMPLEMENTAÇÃO LÉXICA

BELO HORIZONTE
2022

SUMÁRIO

| | |
|----------------------------|----------|
| Analizador léxico | 3 |
| Tag | 3 |
| Token | 3 |
| NumFloat e Num | 3 |
| Lexeme | 3 |
| Léxico | 3 |
| Tabelas de Símbolos | 3 |
| Execução inicial | 4 |
| Teste 1 | 4 |
| Teste 2 | 7 |
| Teste 3 | 10 |
| Teste 4 | 14 |
| Teste 5 | 17 |
| Teste 6 | 20 |

Código

Analizador léxico

1. Tag

Foi criada uma classe Tag. Essa classe define as constantes para os tokens, ela contém um enumerado das palavras reservadas na gramática.

START, EXIT, INT, STRING, FLOAT, IF, THEN, END, ELSE, DO, WHILE, SCAN, PRINT, NOT

Seguido dos demais operandos e tokens.

PV, VRG, AP, FP, AC, FC, EQ, GT, GE, LT, LE, NE, SUM, MIN, MUL, DIV, OR, AND, DOT, PPV, NUM, ID, LIT, VOID, EOF, UNEXPECTED_EOF, INVALID_TOKEN, END_OF_FILE

2. Token

Representa um token genérico. Contém a constante que representa o token.

3. NumFloat e Num

Classes criadas para tratar tokens numéricos: float e int.

4. Lexeme

A classe lexeme representa um token de palavras reservadas, identificadores e tokens compostos como

&&, ==, ||, <=, =>

5. Léxico

Classe que implementa o analisador léxico. Seu construtor insere as palavras reservadas na tabela de símbolos. Possui um método scan que devolve um Token.

Tabelas de Símbolos

Classe que implementa uma estrutura de dados para conter informações sobre os identificadores do programa fonte. Usou-se um HashMap para criar um dicionário dos identificadores usados pelo programador. Sua chave é o Token e o valor a Id.

Suas principais operações são inserir chave e valor na tabela (através do método put), buscar id associada ao token (por meio do método get) e verificar se tabela já possui token (pelo método has).

A classe Id (não sei o que faz a classe id).

Testes

Execução inicial

1. Teste 1

O código apresentado é:

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

    a = 2;
    x = .1;
    scan (b);
    scan (y)
    result = (a*b ++ 1) / 2;
    print "Resultado: ";
    print (result);
    print ("Total: ");
    total = y / x;
    print ("Total: ";
    print (total);
exit
```

Gerando uma saída do console:

| | | |
|--|---|---|
| <pre><start,START> <int,INT> <a,ID> <VRG,VRG> <b,ID> <PV,PV> <int,INT> <result,ID> <PV,PV> <float,FLOAT> <a,ID> <VRG,VRG> <x,ID> <VRG,VRG> <total,ID> <PV,PV> <a,ID> <PPV,PPV> <2,NUM> <PV,PV> <x,ID> <PPV,PPV> Analizador Lexico: Token invalido na linha 7 <1,NUM> <PV,PV> <scan,SCAN> <AP,AP></pre> | <pre><b,ID> <FP,FP> <PV,PV> <scan,SCAN> <AP,AP> <y,ID> <FP,FP> <result,ID> <PPV,PPV> <AP,AP> <a,ID> <MUL,MUL> <b,ID> <SUM,SUM> <SUM,SUM> <1,NUM> <FP,FP> <DIV,DIV> <2,NUM> <PV,PV> <print,PRINT> <Resultado: ,STRING> <PV,PV> <print,PRINT> <AP,AP> <result,ID> <FP,FP> <PV,PV> <print,PRINT> <AP,AP></pre> | <pre><total,ID> <FP,FP> <PV,PV> <exit,EXIT> <EOF,EOF> Total de linhas: 17</pre> |
|--|---|---|

E a saída da tabela de símbolos:

```
Tabela de símbolos:
total -> { lexeme: total }
y -> { lexeme: y }
b -> { lexeme: b }
a -> { lexeme: a }
result -> { lexeme: result }
x -> { lexeme: x }
```

Corrigindo o erro na linha 7, apresentado pela falta de um '0' para a atribuição de float, temos:

```
<start, START>
<int, INT>
<a, ID>
<VRG, VRG>
<b, ID>
<PV, PV>
<int, INT>
<result, ID>
<PV, PV>
<float, FLOAT>
<a, ID>
<VRG, VRG>
<x, ID>
<VRG, VRG>
<total, ID>
<PV, PV>
<a, ID>
<PPV, PPV>
<2, NUM>
<PV, PV>
<x, ID>
<PPV, PPV>
<0.1, NUM>
<PV, PV>
<scan, SCAN>
<AP, AP>
<b, ID>
<FP, FP>
<PV, PV>
```

2. Teste 2

O código apresentado é:

```
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 + d)/2;
```

Gerando uma saída do console:

```
<start,START>
<int,INT>
Analizador Lexico: Token invalido na linha 2
<a,ID>
<VRG,VRG>
<c_,ID>
<PV,PV>
<float,FLOAT>
<d,ID>
<VRG,VRG>
<_e,ID>
<PV,PV>
<a,ID>
<PPV,PPV>
<0,NUM>
<PV,PV>
<d,ID>
<PPV,PPV>
<0.5,NUM>
<c,ID>
<PPV,PPV>
<d,ID>
<DIV,DIV>
<0.2,NUM>
<PV,PV>
<Scan,ID>
<AP,AP>
<a,ID>
```

```
<FP,FP>
<PV,PV>
<Scan,ID>
<AP,AP>
<c,ID>
<FP,FP>
<PV,PV>
<b,ID>
<PPV,PPV>
<a,ID>
<MUL,MUL>
<a,ID>
<PV,PV>
<c,ID>
<PPV,PPV>
<b,ID>
<SUM,SUM>
<a,ID>
<MUL,MUL>
<AP,AP>
<1,NUM>
<SUM,SUM>
<a,ID>
<MUL,MUL>
<c,ID>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<Resultado: ,STRING>
<FP,FP>
<PV,PV>
```



```

<print,PRINT>
<c,ID>
<PV,PV>
<d,ID>
<PPV,PPV>
<0.2,NUM>
<e,ID>
<PPV,PPV>
<val,ID>
<SUM,SUM>
<0.2,NUM>
<PV,PV>
<print,PRINT>
<AP,AP>
<E: ,STRING>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<e,ID>
<FP,FP>
<PV,PV>
<a,ID>
<PPV,PPV>
<b,ID>
<SUM,SUM>
<c,ID>
<SUM,SUM>
<d,ID>
<FP,FP>
<DIV,DIV>
<2,NUM>
<PV,PV>
<EOF,EOF>
Total de linhas: 18

```

E a saída da tabela de símbolos:

```

Tabela de símbolos:
Scan -> { lexeme: Scan }
e -> { lexeme: e }
c_ -> { lexeme: c_ }
a -> { lexeme: a }
d -> { lexeme: d }
_e -> { lexeme: _e }
c -> { lexeme: c }
val -> { lexeme: val }
b -> { lexeme: b }

```

Corrigindo o erro na linha 2, apresentado pelo uso do ':' para a atribuição de um inteiro, temos:

```
<start,START>
<int,INT>
<a,ID>
<VRG,VRG>
<c_,ID>
<PV,PV>
<float,FLOAT>
<d,ID>
<VRG,VRG>
<_e,ID>
<PV,PV>
<a,ID>
<PPV,PPV>
<0,NUM>
```

3. Teste 3

O código apresentado é:

```
int pontuacao, pontuacaoMaxima, 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) and (disponibilidade=="Sim") then
        out("Candidato aprovado");
    else
        out("Candidato reprovado")
    end
while (pontuação >= 0)end
exit
```

Gerando uma saída do console:

```
<int,INT>
<pontuacao,ID>
<VRG,VRG>
<pontuacaoMaxima,ID>
<VRG,VRG>
<disponibilidade,ID>
<PV,PV>
<string,STRING>
<pontuacaoMinima,ID>
<PV,PV>
<disponibilidade,ID>
<PPV,PPV>
<Sim,STRING>
<PV,PV>
<pontuacaoMinima,ID>
<PPV,PPV>
<50,NUM>
<PV,PV>
<pontuacaoMaxima,ID>
<PPV,PPV>
<100,NUM>
<PV,PV>
Exception in thread "main" java.lang.Exception: Um comentário não foi fechado
    at Lexico.scan(Lexico.java:86)
    at Compilador.main(Compilador.java:23)
```

Uma exceção que é desencadeada quando um comentário não é fechado no código, impossibilitando o reconhecimento de novos tokens após este.

Corrigindo o erro de código, apresentado pela falta de fechamento do comentário, temos:

```
<int,INT>
<pontuacao,ID>
<VRG,VRG>
<pontuacaoMaxima,ID>
<VRG,VRG>
<disponibilidade,ID>
<PV,PV>
<string,STRING>
<pontuacaoMinima,ID>
<PV,PV>
<disponibilidade,ID>
<PPV,PPV>
<Sim,STRING>
<PV,PV>
<pontuacaoMinima,ID>
<PPV,PPV>
<50,NUM>
<PV,PV>
<pontuacaoMaxima,ID>
<PPV,PPV>
<100,NUM>
<PV,PV>
<do,DO>
<print,PRINT>
<AP,AP>
<Pontuacao Candidato: ,STRING>
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<pontuacao,ID>
<FP,FP>
```

```
<PV,PV>
<print,PRINT>
<AP,AP>
<Disponibilidade Candidato: ,STRING>
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<disponibilidade,ID>
<FP,FP>
<PV,PV>
<if,IF>
<AP,AP>
<AP,AP>
<pontuaÃ, ID>
Analizador Lexico: Token invalido na linha 17
<Ã,ID>
Analizador Lexico: Token invalido na linha 17
<o,ID>
<GT,GT>
<pontuacaoMinima,ID>
<FP,FP>
<and,ID>
<AP,AP>
<disponibilidade,ID>
<==,EQ>
<Sim,STRING>
<FP,FP>
<then,THEN>
<out,ID>
<AP,AP>
```

```

<Candidato aprovado,STRING>
<FP,FP>
<PV,PV>
<else,ELSE>
<out,ID>
<AP,AP>
<Candidato reprovado,STRING>
<FP,FP>
<end,END>
<while,WHILE>
<AP,AP>
<pontuaÃ,ID>
Analizador Lexico: Token invalido na linha 22
<Ã,ID>
Analizador Lexico: Token invalido na linha 22
<o,ID>
<>=,GE>
<0,NUM>
<FP,FP>
<end,END>
<exit,EXIT>
<EOF,EOF>
Total de linhas: 23

```

E com a saída incorreta da tabela de símbolos:

```

Tabela de símbolos:
o -> { lexeme: o }
pontuaÃ -> { lexeme: pontuaÃ }
pontuacaoMaxina -> { lexeme: pontuacaoMaxina }
pontuacaoMaxima -> { lexeme: pontuacaoMaxima }
pontuacao -> { lexeme: pontuacao }
disponibilidade -> { lexeme: disponibilidade }
pontuacaoMinima -> { lexeme: pontuacaoMinima }
Ã -> { lexeme: Ã }

```

Corrigindo os erros nas linhas 17 e 22, apresentado pelo uso de caracteres especiais, fora do padrão ASCII usado para a atribuição de um char, temos:

| | |
|----------------------|------------------------------|
| <PV,PV> | <Candidato aprovado,STRING> |
| <if,IF> | <FP,FP> |
| <AP,AP> | <PV,PV> |
| <AP,AP> | <else,ELSE> |
| <pontuacao,ID> | <out,ID> |
| <GT,GT> | <AP,AP> |
| <pontuacaoMinima,ID> | <Candidato reprovado,STRING> |
| <FP,FP> | <FP,FP> |
| <and,ID> | <end,END> |
| <AP,AP> | <while,WHILE> |
| <disponibilidade,ID> | <AP,AP> |
| <==,EQ> | <pontuacao,ID> |
| <Sim,STRING> | <>=,GE> |
| <FP,FP> | <0,NUM> |
| <then,THEN> | <FP,FP> |
| <out,ID> | <end,END> |
| <AP,AP> | |

E a saída da tabela de símbolos correta de:

```
Tabela de símbolos:
pontuacaoMinima -> { lexeme: pontuacaoMinima }
out -> { lexeme: out }
pontuacaoMaxima -> { lexeme: pontuacaoMaxima }
pontuacaoMaxina -> { lexeme: pontuacaoMaxina }
disponibilidade -> { lexeme: disponibilidade }
and -> { lexeme: and }
pontuacao -> { lexeme: pontuacao }
```

4. Teste 4

O código apresentado é:

```
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
```

Gerando uma saída do console:

| | |
|--|--|
| <pre> <start,START> <Int,ID> <a,ID> <VRG,VRG> <aux,ID> Analizador Lexico: Token invalido na linha 3 <VRG,VRG> <b,ID> <PV,PV> <string,STRING> <nome,ID> <VRG,VRG> <sobrenome,ID> <VRG,VRG> <msg,ID> <PV,PV> <print,PRINT> <AP,AP> <Nome,ID> Analizador Lexico: Token invalido na linha 6 <FP,FP> <PV,PV> <scan,SCAN> <AP,AP> <nome,ID> <FP,FP> <PV,PV> <print,PRINT> <AP,AP> </pre> | <pre> <Sobrenome: ,STRING> <FP,FP> <PV,PV> <scan,SCAN> <AP,AP> <sobrenome,ID> <FP,FP> <PV,PV> <msg,ID> <PPV,PPV> <Ola, ,STRING> <SUM,SUM> <nome,ID> <SUM,SUM> < ,STRING> <SUM,SUM> <sobrenome,ID> <SUM,SUM> <!,STRING> <PV,PV> <msg,ID> <PPV,PPV> <msg,ID> <SUM,SUM> <1,NUM> <PV,PV> <print,PRINT> <AP,AP> <msg,ID> </pre> |
|--|--|

| | |
|-------------|-------------------------|
| <FP,FP> | |
| <PV,PV> | |
| <scan,SCAN> | |
| <AP,AP> | |
| <a,ID> | |
| <FP,FP> | |
| <PV,PV> | |
| <scan,SCAN> | |
| <AP,AP> | |
| <b,ID> | |
| <FP,FP> | <PV,PV> |
| <PV,PV> | <end,END> |
| <if,IF> | <PV,PV> |
| <AP,AP> | <print,PRINT> |
| <a,ID> | <AP,AP> |
| <GT,GT> | <Apos a troca: ,STRING> |
| <b,ID> | <FP,FP> |
| <FP,FP> | <PV,PV> |
| <then,THEN> | <out,ID> |
| <aux,ID> | <AP,AP> |
| <PPV,PPV> | <a,ID> |
| <b,ID> | <FP,FP> |
| <PV,PV> | <PV,PV> |
| <b,ID> | <out,ID> |
| <PPV,PPV> | <AP,AP> |
| <a,ID> | <b,ID> |
| <PV,PV> | <FP,FP> |
| <a,ID> | <exit,EXIT> |
| <PPV,PPV> | <EOF,EOF> |
| <aux,ID> | Total de linhas: 24 |

E a saída da tabela de símbolos:

```
Tabela de símbolos:
a -> { lexeme: a }
Int -> { lexeme: Int }
nome -> { lexeme: nome }
Nome -> { lexeme: Nome }
out -> { lexeme: out }
aux -> { lexeme: aux }
sobrenome -> { lexeme: sobrenome }
msg -> { lexeme: msg }
b -> { lexeme: b }
```

Corrigindo o erro na linha 3, apresentado pelo uso do '\$' para a iniciação de uma variável do tipo inteiro, temos:


```
<start,START>
<Int,ID>
<a,ID>
<VRG,VRG>
<aux,ID>
<VRG,VRG>
<b,ID>
<PV,PV>
<string,STRING>
<nome,ID>
<VRG,VRG>
<sobrenome,ID>
<VRG,VRG>
<msg,ID>
<PV,PV>
```

Corrigindo o erro na linha 6, apresentado pelo uso do ':' o que, na verdade, é a falta de aspas no string dentro do print que só pode ser visto no analisador sintático, temos:

```
<sobrenome,ID>
<VRG,VRG>
<msg,ID>
<PV,PV>
<print,PRINT>
<AP,AP>
<Nome: ,STRING>
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<nome,ID>
```

5. Teste 5

O código apresentado é:

```
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) and (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
```

Gerando uma saída do console:

```
<start,START>
<int,INT>
<a,ID>
<VRG,VRG>
<b,ID>
<VRG,VRG>
<c,ID>
<VRG,VRG>
<maior,ID>
<VRG,VRG>
<outro,ID>
<PV,PV>
<do,DO>
<print,PRINT>
<AP,AP>
<A,STRING>
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<a,ID>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<B,STRING>
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<b,ID>
<FP,FP>
<and,ID>
<AP,AP>
<a,ID>
<GT,GT>
<c,ID>
<FP,FP>
<FP,FP>
<maior,ID>
```

```
<PPV,PPV>
<a,ID>
<else,ELSE>
<if,IF>
<AP,AP>
<b,ID>
<GT,GT>
<c,ID>
<FP,FP>
<then,THEN>
<maior,ID>
<PPV,PPV>
<b,ID>
<PV,PV>
<else,ELSE>
<maior,ID>
<PPV,PPV>
<c,ID>
<PV,PV>
<end,END>
<end,END>
<print,PRINT>
<AP,AP>
<Maior valor:,STRING>
<);
    print (maior);
    print (,STRING>
<Outro,ID>
Analizador Lexico: Token invalido na linha 24
Exception in thread "main" java.lang.Exception: Um string não foi fechado
```

Uma exceção que é desencadeada quando um string não é fechado no código, impossibilitando o reconhecimento de novos tokens após este.

Corrigindo o erro de código, apresentado pela falta de fechamento do comentário, temos:

| | | | |
|---------------|---------------|-------------|-----------------------|
| <start,START> | <B,STRING> | <GT,GT> | <end,END> |
| <int,INT> | <FP,FP> | <c,ID> | <print,PRINT> |
| <a,ID> | <PV,PV> | <FP,FP> | <AP,AP> |
| <VRG,VRG> | <scan,SCAN> | <FP,FP> | <Maior valor:,STRING> |
| <b,ID> | <AP,AP> | <FP,FP> | <FP,FP> |
| <VRG,VRG> | <b,ID> | <maior,ID> | <PV,PV> |
| <c,ID> | <FP,FP> | <PPV,PPV> | <print,PRINT> |
| <VRG,VRG> | <PV,PV> | <a,ID> | <AP,AP> |
| <maior,ID> | <print,PRINT> | <else,ELSE> | <maior,ID> |
| <VRG,VRG> | <AP,AP> | <if,IF> | <FP,FP> |
| <outro,ID> | <C,STRING> | <AP,AP> | <PV,PV> |
| <PV,PV> | <FP,FP> | <b,ID> | <print,PRINT> |
| <do,DO> | <PV,PV> | <GT,GT> | <AP,AP> |
| <print,PRINT> | <scan,SCAN> | <c,ID> | <Outro? ,STRING> |
| <AP,AP> | <AP,AP> | <FP,FP> | <FP,FP> |
| <A,STRING> | <c,ID> | <then,THEN> | <PV,PV> |
| <FP,FP> | <FP,FP> | <maior,ID> | <scan,SCAN> |
| <PV,PV> | <PV,PV> | <PPV,PPV> | <AP,AP> |
| <scan,SCAN> | <if,IF> | <b,ID> | <outro,ID> |
| <AP,AP> | <AP,AP> | <PV,PV> | <FP,FP> |
| <a,ID> | <AP,AP> | <else,ELSE> | <PV,PV> |
| <FP,FP> | <a,ID> | <maior,ID> | <while,WHILE> |
| <PV,PV> | <GT,GT> | <PPV,PPV> | <AP,AP> |
| <print,PRINT> | <b,ID> | <c,ID> | <outro,ID> |
| <AP,AP> | <FP,FP> | <PV,PV> | <=>,GE> |
| | <and,ID> | <end,END> | <0,NUM> |
| | <AP,AP> | | <FP,FP> |
| | <a,ID> | | <PV,PV> |
| | | | <exit,EXIT> |
| | | | <EOF,EOF> |
| | | | Total de linhas: 29 |

Com a tabela de símbolos:

```
Tabela de símbolos:
c -> { lexeme: c }
outro -> { lexeme: outro }
a -> { lexeme: a }
maior -> { lexeme: maior }
b -> { lexeme: b }
and -> { lexeme: and }
```

6. Teste 6

O código apresentado é:

```
//Comentario
start
    int a, b;
    int result;
    string c;

    print(a);
    scan (a);
    print(b);
```

```

scan (b);

print(a);
print(b);

result = a / b;

if(result>1) then
    c = "A é maior que B";
else
    c = "B é maior que A";
end;

print(c);

exit

```

Gerando uma saída do console:

```

<start,START>
<int,INT>
<a,ID>
<VRG,VRG>
<b,ID>
<PV,PV>
<int,INT>
<result,ID>
<PV,PV>
<string,STRING>
<c,ID>
<PV,PV>
<print,PRINT>
<AP,AP>
<a,ID>
Analizador Lexico: Token invalido na linha 7
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<a,ID>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<b,ID>

```

```

Analizador Lexico: Token invalido na linha 9
<FP,FP>
<PV,PV>
<scan,SCAN>
<AP,AP>
<b,ID>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<a,ID>
<FP,FP>
<PV,PV>
<print,PRINT>
<AP,AP>
<b,ID>
<FP,FP>
<PV,PV>
<result,ID>
<PPV,PPV>
<a,ID>
<DIV,DIV>
<b,ID>
<PV,PV>
<if,IF>
<AP,AP>
<result,ID>
<GT,GT>
<1,NUM>
<FP,FP>

```

```

<then, THEN>
<c, ID>
<PPV, PPV>
<A > maior que B, STRING>
<PV, PV>
<else, ELSE>
<c, ID>
<PPV, PPV>
<B > maior que A, STRING>
<PV, PV>
<end, END>
<PV, PV>
<print, PRINT>
<AP, AP>
<c, ID>
<FP, FP>
<PV, PV>
<exit, EXIT>
<EOF, EOF>
Total de linhas: 25

```

E a saída da tabela de símbolos:

```

Tabela de símbolos:
c -> { lexeme: c }
b -> { lexeme: b }
result -> { lexeme: result }
a -> { lexeme: a }

```

Corrigindo o erro na linha 7 e 9, apresentado pelo uso do ‘:’ o que, na verdade, é a falta de aspas no string dentro do print que só pode ser visto no analisador sintático, temos:

```

<PV, PV>
<print, PRINT>
<AP, AP>
<a:, STRING>
<FP, FP>
<PV, PV>
<scan, SCAN>
<AP, AP>
<a, ID>
<FP, FP>
<PV, PV>
<print, PRINT>
<AP, AP>
<b:, STRING>
<FP, FP>
<PV, PV>
<scan, SCAN>
<AP, AP>
<b, ID>
<FP, FP>
<PV, PV>

```