TP Compiladores - Parte I
Analisador Léxico e Tabela de símbolos
25/09/2023

Guilherme de Assis Lima

Código Fonte e arquivo .jar

O código fonte e arquivo .jar estão disponíveis em:
https://github.com/guicompeng/compilador/tree/lexico

Forma de uso do compilador

Para executar o compilador, basta rodar o comando:

```
java -jar AnalisadorLexico.jar testes/1.txt
```

Observação: “testes/1.txt” é o programa a ser compilado, podendo ser qualquer arquivo texto.

Abordagem

A abordagem utilizada foi baseada no livro texto da disciplina, com algumas mudanças no código para adaptar a especificação do compilador.
As seguintes classes foram criadas:

- Lexer: classe que implementa o analisador léxico. Seu construtor insere as palavras reservadas na tabela de símbolos. Possui um método scan que devolve um Token. Além disso, insere os identificadores na tabela de símbolos.
- Tag: classe que define os enums para os tokens. Foi realizada uma mudança em relação ao livro, pois a classe foi feita com enums ao invés de constantes que mapeiam um número inteiro.
- Token: representa um token genérico. Contém a constante que representa o token. Também foi realizada uma mudança em relação ao livro, pois a classe Token se tornou abstrata contendo o método abstrato *getLexeme()* . Portanto, todas as classes que entenderem de Token precisaram implementar este método. Essa mudança foi feita para facilitar nos testes para imprimir o lexema daquele token. Dessa forma, independentemente de quem estender a classe token, será possível imprimir o lexema.
- Float: representa um token do tipo float. A classe Float se estende da classe Token. Obs: Foi utilizado o tipo primitivo double do java para representar o float nesta linguagem.
- Int: representa um token do tipo int. Classe Int estende a classe Token.

- Word: representa um token de palavras reservadas, identificadores e tokens compostos. A diferença com a do livro, é que foi implementado o método getLexeme().
- Teste: é a classe Main, que é responsável por ler o arquivo texto, instanciar a classe Lexer, e chamar o método scan. Essa classe simula (de forma bem simples e resumida) o comportamento do analisador sintático a ser desenvolvido na próxima etapa do trabalho.

Teste 1

Comando: *java -jar AnalisadorLexico.jar testes/1.txt*

Resultado: sem erro léxico.

Fonte 1

```
class Teste1
  int a,b,c;
  float result;

{
  write("Digite o valor de a:");
  read (a);
  write("Digite o valor de c:");
  read (c);
  b = 10;
  result = (a * c)/(b  5 - 345);
  write("O resultado e: ");
  write(result);
}
```

Resultado tabela de símbolos

```
Tabela de simbolos:
if
read
float
class
while
do
string
int
c
b
a
result
Teste1
write
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

Token	Lexema
CLASS	class
ID	Teste1
INT	int
ID	a
COMMA	,
ID	b
COMMA	,
ID	c
SEMICOLON	;
FLOAT	float
ID	result
SEMICOLON	;
OPEN_CURLY_BRACKET	{
WRITE	write
OPEN_ROUND_BRACKET	(
LITERAL	Digite o valor de a:
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
READ	read
OPEN_ROUND_BRACKET	(
ID	a
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
WRITE	write
OPEN_ROUND_BRACKET	(
LITERAL	Digite o valor de c:
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
READ	read
OPEN_ROUND_BRACKET	(
ID	c
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
ID	b
ASSIGN	=
INT	10
SEMICOLON	;
ID	result
ASSIGN	=
OPEN_ROUND_BRACKET	(
ID	a
OP_MUL	*
ID	c
CLOSE_ROUND_BRACKET	)
OP_DIV	/
OPEN_ROUND_BRACKET	(
ID	b
INT	5
OP_SUB	-
INT	345
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
WRITE	write
OPEN_ROUND_BRACKET	(
LITERAL	O resultado e:
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
WRITE	write
OPEN_ROUND_BRACKET	(
ID	result
CLOSE_ROUND_BRACKET	)
SEMICOLON	;
CLOSE_CURLY_BRACKET	}
END_OF_FILE	

Teste 2

Comando: *java -jar AnalisadorLexico.jar testes/2.txt*

Resultado: existe um erro léxico, pois não foi finalizado o comentário e o programa terminou.

Fonte 2

```
class Teste2
/* Teste de comentário
com mais de uma linha

a, 9valor, b_1, b_2 : int;

{
  write("Entre com o valor de a: ");
  read (a);
  b_1 := a * a;
  write("O valor de b1 e: ");
  write (b_1);
  b_2 = b + a/2 * (a + 5);
  write("O valor de b2 e: ");
  Write (b2);
}
```

```
Token                          Lexema
-------------------------------------------------------------
CLASS                          class
ID                             Teste2
Erro léxico na linha 15. Fim de arquivo não era esperado
UNEXPECTED_EOF
END_OF_FILE
```

Comando: *java -jar AnalisadorLexico.jar testes/2parcial.txt*
Resultado: após corrigir o erro do comentário, o compilador mostrou novos erros léxicos: na linha 4 não foi encontrado o token ":" e na linha 9 ":=". Além disso, a variável "9valor" não foi considerada um erro, e sim duas variáveis "9" e "valor".

Fonte 2 parcialmente corrigido

```
class Teste2
/* Teste de comentário
com mais de uma linha */
a, 9valor, b_1, b_2 : int;

{
    write("Entre com o valor de a: ");
    read (a);
    b_1 := a * a;
    write("O valor de b1 e: ");
    write (b_1);
    b_2 = b + a/2 * (a + 5);
    write("O valor de b2 e: ");
    Write (b2);
}
```

```
Token                          Lexema
-------------------------------------------------------------
CLASS                          class
ID                             Teste2
ID                             a
COMMA                          ,
INT                            9
ID                             valor
COMMA                          ,
ID                             b_1
COMMA                          ,
ID                             b_2
Erro léxico na linha 4. Lexema inválido: :
INVALID_TOKEN                  :
INT                            int
SEMICOLON                      ;
OPEN_CURLY_BRACKET             {
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Entre com o valor de a:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             a
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
ID                             b_1
Erro léxico na linha 9. Lexema inválido: :=
INVALID_TOKEN                  :=
ID                             a
OP_MUL                         *
ID                             a
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        O valor de b1 e:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
ID                             b_1
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
ID                             b_2
ASSIGN                         =
ID                             b
OP_SUM                         +
ID                             a
OP_DIV                         /
INT                            2
OP_MUL                         *
OPEN_ROUND_BRACKET             (
ID                             a
OP_SUM                         +
INT                            5
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        O valor de b2 e:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
ID                             Write
OPEN_ROUND_BRACKET             (
ID                             b2
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
CLOSE_CURLY_BRACKET            }
END_OF_FILE
```

Comando: *java -jar AnalisadorLexico.jar testes/2corrigido.txt*

Resultado: analisador léxico ok

Fonte 2 corrigido

```
class Teste2
/* Teste de comentário
com mais de uma linha */
a, 9, valor, b_1, b_2, int;

{
    write("Entre com o valor de a: ");
    read (a);
    b_1 = a * a;
    write("O valor de b1 e: ");
    write (b_1);
    b_2 = b + a/2 * (a + 5);
    write("O valor de b2 e: ");
    Write (b2);
}
```

Resultado tabela de símbolos

```
Tabela de simbolos:
if
b_2
b_1
read
float
class
valor
while
do
string
int
b2
b
Write
a
Teste2
write
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
Token                   Lexema
-------------------------------------------------------------
CLASS                   class
ID                      Teste2
ID                      a
COMMA                   ,
INT                     9
COMMA                   ,
ID                      valor
COMMA                   ,
ID                      b_1
COMMA                   ,
ID                      b_2
COMMA                   ,
INT                     int
SEMICOLON               ;
OPEN_CURLY_BRACKET      {
WRITE                   write
OPEN_ROUND_BRACKET      (
LITERAL                 Entre com o valor de a:
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
READ                    read
OPEN_ROUND_BRACKET      (
ID                      a
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
ID                      b_1
ASSIGN                  =
ID                      a
OP_MUL                  *
ID                      a
SEMICOLON               ;
WRITE                   write
OPEN_ROUND_BRACKET      (
LITERAL                 O valor de b1 e:
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
WRITE                   write
OPEN_ROUND_BRACKET      (
ID                      b_1
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
ID                      b_2
ASSIGN                  =
ID                      b
OP_SUM                  +
ID                      a
OP_DIV                  /
INT                     2
OP_MUL                  *
OPEN_ROUND_BRACKET      (
ID                      a
OP_SUM                  +
INT                     5
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
WRITE                   write
OPEN_ROUND_BRACKET      (
LITERAL                 O valor de b2 e:
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
ID                      Write
OPEN_ROUND_BRACKET      (
ID                      b2
CLOSE_ROUND_BRACKET     )
SEMICOLON               ;
CLOSE_CURLY_BRACKET     }
END_OF_FILE
```

Teste 3

Comando: *java -jar AnalisadorLexico.jar testes/3.txt*

Resultado: sem erro léxico.

Fonte 3

```
classe Teste3

/** Verificando fluxo de controle
Programa com if e while aninhados **/

 int i;
 int media, soma;

 {
   soma = 0;

   write("Quantos dados deseja informar?" );
   read (qtd);

   IF (qtd>=2){
      i=0;
      do{
        write("Altura: ");
        read (altura);
        soma = soma+altura;
        i = i + 1;
      }while( i < qtd);

      media = soma / qtd;
      write("Media: ");
      write (media);


   }
   else{
     write("Quantidade inválida.");
   }
 }
```

Resultado tabela de símbolos

```
Tabela de simbolos:
string
int
Teste3
altura
media
IF
class
soma
if
while
do
read
i
else
classe
write
qtd
float
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
Token                       Lexema
-------------------------------------------------------------------
ID                          classe
ID                          Teste3
INT                         int
ID                          i
SEMICOLON                   ;
INT                         int
ID                          media
COMMA                       ,
ID                          soma
SEMICOLON                   ;
OPEN_CURLY_BRACKET          {
ID                          soma
ASSIGN                      =
INT                         0
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Quantos dados deseja informar?
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          qtd
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
ID                          IF
OPEN_ROUND_BRACKET          (
ID                          qtd
GREATER_EQUAL               >=
INT                         2
CLOSE_ROUND_BRACKET         )
OPEN_CURLY_BRACKET          {
ID                          i
ASSIGN                      =
INT                         0
SEMICOLON                   ;
DO                          do
OPEN_CURLY_BRACKET          {
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Altura:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          altura
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
ID                          soma
ASSIGN                      =
ID                          soma
OP_SUM                      +
ID                          altura
SEMICOLON                   ;
ID                          i
ASSIGN                      =
ID                          i
OP_SUM                      +
INT                         1
SEMICOLON                   ;
CLOSE_CURLY_BRACKET         }
WHILE                       while
OPEN_ROUND_BRACKET          (
ID                          i
LESS                        <
ID                          qtd
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
ID                          media
ASSIGN                      =
ID                          soma
OP_DIV                      /
ID                          qtd
```

	SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (LITERAL Media: CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID media CLOSE_ROUND_BRACKET) SEMICOLON ; CLOSE_CURLY_BRACKET } ID else OPEN_CURLY_BRACKET { WRITE write OPEN_ROUND_BRACKET (LITERAL Quantidade inválida. CLOSE_ROUND_BRACKET) SEMICOLON ; CLOSE_CURLY_BRACKET } CLOSE_CURLY_BRACKET } END_OF_FILE

Teste 4

Comando: *java -jar AnalisadorLexico.jar testes/4.txt*

Resultado: Erro léxico na linha 4 (variável começa com @), na linha 7 (não foi fechado a “ da string), e na linha 13, 14 e 15 (operador inválido :=)

Fonte 4

```
{
// Outro programa de teste

int idade, j, k, @total;
string nome, texto;

write("Digite o seu nome: );
read(nome);
write("Digite o seu sobrenome");
read(sobrenome);
write("Digite a sua idade: ");
read (idade);
k := i * (5-i * 50 / 10;
j := i * 10;
k := i * j / k;
texto = nome + " " + sobrenome + ", os números gerados sao: ";
write (text);
write(j);
write(k);

}
```

```
Token                          Lexema
-------------------------------------------------------------------
OPEN_CURLY_BRACKET             {
INT                            int
ID                             idade
COMMA                          ,
ID                             j
COMMA                          ,
ID                             k
COMMA                          ,
Erro léxico na linha 4. Lexema inválido: @total;
INVALID_TOKEN                  @total;
STRING                         string
ID                             nome
COMMA                          ,
ID                             texto
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
Erro léxico na linha 7. String não foi fechada: Digite o seu nome:
INVALID_TOKEN                  Digite o seu nome: );
READ                           read
OPEN_ROUND_BRACKET             (
ID                             nome
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite o seu sobrenome
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             sobrenome
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite a sua idade:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             idade
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
ID                             k
Erro léxico na linha 13. Lexema inválido: :=
INVALID_TOKEN                  :=
ID                             i
OP_MUL                         *
OPEN_ROUND_BRACKET             (
INT                            5
OP_SUB                         -
ID                             i
OP_MUL                         *
INT                            50
OP_DIV                         /
INT                            10
SEMICOLON                      ;
ID                             j
Erro léxico na linha 14. Lexema inválido: :=
INVALID_TOKEN                  :=
ID                             i
OP_MUL                         *
INT                            10
SEMICOLON                      ;
ID                             k
Erro léxico na linha 15. Lexema inválido: :=
INVALID_TOKEN                  :=
ID                             i
OP_MUL                         *
ID                             j
OP_DIV                         /
ID                             k
SEMICOLON                      ;
```

	ID texto ASSIGN = ID nome OP_SUM + LITERAL OP_SUM + ID sobrenome OP_SUM + LITERAL , os números gerados sa SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID text CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID j CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID k CLOSE_ROUND_BRACKET) SEMICOLON ; CLOSE_CURLY_BRACKET } END_OF_FILE

Comando: *java -jar AnalisadorLexico.jar testes/4corrigido.txt*

Resultado: analisador léxico ok

Fonte 4 corrigido

```
{
// Outro programa de teste

int idade, j, k, @total;
string nome, texto;

write("Digite o seu nome: ");
read(nome);
write("Digite o seu sobrenome");
read(sobrenome);
write("Digite a sua idade: ");
read (idade);
k := i * (5-i * 50 / 10;
j := i * 10;
k := i * j / k;
texto = nome + " " + sobrenome + ", os
números gerados sao: ";
write (text);
write(j);
write(k);
}
```

Resultado tabela de símbolos

```
Tabela de simbolos:
string
int
texto
idade
sobrenome
text
class
if
while
do
k
j
read
nome
i
write
float
total
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
[-------------------------------------------------------------
OPEN_CURLY_BRACKET          {
INT                         int
ID                          idade
COMMA                       ,
ID                          j
COMMA                       ,
ID                          k
COMMA                       ,
ID                          total
SEMICOLON                   ;
STRING                      string
ID                          nome
COMMA                       ,
ID                          texto
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Digite o seu nome:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          nome
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Digite o seu sobrenome
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          sobrenome
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Digite a sua idade:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          idade
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
ID                          k
ASSIGN                      =
ID                          i
OP_MUL                      *
OPEN_ROUND_BRACKET          (
INT                         5
OP_SUB                      -
ID                          i
OP_MUL                      *
INT                         50
OP_DIV                      /
INT                         10
SEMICOLON                   ;
ID                          j
ASSIGN                      =
ID                          i
OP_MUL                      *
INT                         10
SEMICOLON                   ;
ID                          k
ASSIGN                      =
ID                          i
OP_MUL                      *
ID                          j
OP_DIV                      /
ID                          k
SEMICOLON                   ;
ID                          texto
ASSIGN                      =
ID                          nome
```

	OP_SUM + LITERAL OP_SUM + ID sobrenome OP_SUM + LITERAL , os números gerados sao: SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID text CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID j CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID k CLOSE_ROUND_BRACKET) SEMICOLON ; CLOSE_CURLY_BRACKET } END_OF_FILE

Teste 5

Comando: *java -jar AnalisadorLexico.jar testes/5.txt*

Resultado: Erro léxico na linha 12 operador inválido ":="

Fonte 5

```
class MinhaClasse
{
   float a, b, c;

      write("Digite um número");
      read(a);
      write("Digite outro número: ");
      read(b);
      write("Digite mais um número: ");
      read(c;

      maior := 0;


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



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


     write("O maior número é: ");
     write(maior);
```

```
Token                          Lexema
-------------------------------------------------------------
CLASS                          class
ID                             MinhaClasse
OPEN_CURLY_BRACKET             {
FLOAT                          float
ID                             a
COMMA                          ,
ID                             b
COMMA                          ,
ID                             c
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite um número
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             a
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite outro número:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             b
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite mais um número:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             c
SEMICOLON                      ;
ID                             maior
Erro léxico na linha 12. Lexema inválido: :=
INVALID_TOKEN                  :=
INT                            0
SEMICOLON                      ;
IF                             if
OPEN_ROUND_BRACKET             (
ID                             a
GREATER                        >
ID                             b
AND                            &&
ID                             a
GREATER                        >
ID                             c
CLOSE_ROUND_BRACKET            )
ID                             maior
ASSIGN                         =
ID                             a
SEMICOLON                      ;
ID                             else
IF                             if
OPEN_ROUND_BRACKET             (
ID                             b
GREATER                        >
ID                             c
CLOSE_ROUND_BRACKET            )
ID                             maior
ASSIGN                         =
ID                             b
SEMICOLON                      ;
ID                             else
ID                             maior
ASSIGN                         =
ID                             c
SEMICOLON                      ;
```

	WRITE write OPEN_ROUND_BRACKET (LITERAL O maior número é: CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID maior CLOSE_ROUND_BRACKET) SEMICOLON ; END_OF_FILE

Comando: *java -jar AnalisadorLexico.jar testes/5corrigido.txt*

Resultado: analisador léxico ok

Fonte 5 corrigido

```
class MinhaClasse
{
 float a, b, c;

 write("Digite um número");
 read(a);
 write("Digite outro número: ");
 read(b);
 write("Digite mais um número: ");
 read(c;

 maior = 0;
 if ( a>b && a>c )
 maior = a;
 else
 if (b>c)
 maior = b;
 else
 maior = c;

 write("O maior número é: ");
 write(maior);
```

Resultado tabela de símbolos

```
Tabela de simbolos:
if
read
float
class
while
do
maior
string
int
c
else
b
a
MinhaClasse
write
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
Token                          Lexema
-------------------------------------------------------------
CLASS                          class
ID                             MinhaClasse
OPEN_CURLY_BRACKET             {
FLOAT                          float
ID                             a
COMMA                          ,
ID                             b
COMMA                          ,
ID                             c
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite um número
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             a
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite outro número:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             b
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite mais um número:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             c
SEMICOLON                      ;
ID                             maior
ASSIGN                         =
INT                            0
SEMICOLON                      ;
IF                             if
OPEN_ROUND_BRACKET             (
ID                             a
GREATER                        >
ID                             b
AND                            &&
ID                             a
GREATER                        >
ID                             c
CLOSE_ROUND_BRACKET            )
ID                             maior
ASSIGN                         =
ID                             a
SEMICOLON                      ;
ID                             else
IF                             if
OPEN_ROUND_BRACKET             (
ID                             b
GREATER                        >
ID                             c
CLOSE_ROUND_BRACKET            )
ID                             maior
ASSIGN                         =
ID                             b
SEMICOLON                      ;
ID                             else
ID                             maior
ASSIGN                         =
ID                             c
SEMICOLON                      ;
WRITE                          write
```

	OPEN_ROUND_BRACKET (LITERAL O maior número é: CLOSE_ROUND_BRACKET) SEMICOLON ; WRITE write OPEN_ROUND_BRACKET (ID maior CLOSE_ROUND_BRACKET) SEMICOLON ; END_OF_FILE

Teste 6

Comando: *java -jar AnalisadorLexico.jar testes/6.txt*

Resultado: sem erro léxico.

Fonte 6

```
classe Teste6
int a, b, resultado;
{
    write("Digite o valor de a:");
    read (a);
    write("Digite o valor de b:");
    read (b);
    resultado = a + b;
    write("A soma e: ");
    write(resultado);
}
```

Resultado tabela de símbolos

```
Tabela de simbolos:
if
read
classe
float
class
resultado
while
do
string
int
b
a
Teste6
write
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
Token                       Lexema
-------------------------------------------------------------
ID                          classe
ID                          Teste6
INT                         int
ID                          a
COMMA                       ,
ID                          b
COMMA                       ,
ID                          resultado
SEMICOLON                   ;
OPEN_CURLY_BRACKET          {
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Digite o valor de a:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          a
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     Digite o valor de b:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
READ                        read
OPEN_ROUND_BRACKET          (
ID                          b
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
ID                          resultado
ASSIGN                      =
ID                          a
OP_SUM                      +
ID                          b
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
LITERAL                     A soma e:
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
WRITE                       write
OPEN_ROUND_BRACKET          (
ID                          resultado
CLOSE_ROUND_BRACKET         )
SEMICOLON                   ;
CLOSE_CURLY_BRACKET         }
END_OF_FILE
```

Teste 7

Comando: *java -jar AnalisadorLexico.jar testes/7.txt*

Resultado: sem erro léxico.

Fonte 7

```
classe Teste7
int peso, altura, resultado;
{
    write("Digite o peso:");
    read (peso);
    write("Digite a altura:");
    read (altura);
    resultado = peso/(altura*altura);
    write("O IMC e: ");
    write(resultado);
}
```

Resultado tabela de símbolos

```
Tabela de simbolos:
if
read
classe
float
class
resultado
while
do
string
int
altura
Teste7
peso
write
```

Obs: esta tabela de símbolos contém as palavras reservadas e identificadores. O nível e tipo de cada identificador será obtido no analisador sintático.

```
Token                          Lexema
-------------------------------------------------------------
ID                             classe
ID                             Teste7
INT                            int
ID                             peso
COMMA                          ,
ID                             altura
COMMA                          ,
ID                             resultado
SEMICOLON                      ;
OPEN_CURLY_BRACKET             {
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite o peso:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             peso
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        Digite a altura:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
READ                           read
OPEN_ROUND_BRACKET             (
ID                             altura
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
ID                             resultado
ASSIGN                         =
ID                             peso
OP_DIV                         /
OPEN_ROUND_BRACKET             (
ID                             altura
OP_MUL                         *
ID                             altura
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
LITERAL                        O IMC e:
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
WRITE                          write
OPEN_ROUND_BRACKET             (
ID                             resultado
CLOSE_ROUND_BRACKET            )
SEMICOLON                      ;
CLOSE_CURLY_BRACKET            }
END_OF_FILE
```