

# Documentação Linguagem L--

Lucas Montenegro e Letícia Medeiros

05/04/2021

Universidade Federal de Alagoas - Instituto de Computação  
Compiladores  
Professor Alcino Dall Igna Junior

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Estrutura Geral do Programa</b>	<b>3</b>
2.1	Ponto Inicial do Programa . . . . .	3
2.2	Definição de funções . . . . .	3
2.3	Definição de instruções . . . . .	3
2.4	Estrutura da nomeação de variáveis e funções . . . . .	3
2.5	Símbolos especiais . . . . .	4
<b>3</b>	<b>Conjunto de tipos de dados e nomes</b>	<b>4</b>
3.1	Palavras Reservadas . . . . .	4
3.2	Inteiro . . . . .	4
3.3	Ponto flutuante . . . . .	4
3.4	Booleano . . . . .	5
3.5	Char . . . . .	5
3.6	String . . . . .	5
3.7	Arranjos unidimensionais . . . . .	6
3.8	Variáveis . . . . .	7
3.9	Operações de cada tipo . . . . .	7
3.10	Valores padrão . . . . .	7
<b>4</b>	<b>Conjunto de Operadores</b>	<b>7</b>
4.1	Tipos das Operações . . . . .	7
4.1.1	Operadores Aritméticos . . . . .	7
4.1.2	Operadores Relacionais . . . . .	8
4.1.3	Operadores Lógicos . . . . .	8
4.1.4	Operadores de Concatenação . . . . .	8
4.2	Ordem de precedência e associatividade . . . . .	8
<b>5</b>	<b>Instruções</b>	<b>8</b>
5.1	Atribuição . . . . .	8
5.2	Estrutura Condicional de uma e duas vias . . . . .	9
5.3	Estrutura condicional de controle lógico . . . . .	9
5.4	Estrutura iterativa controlada por contador . . . . .	10
5.4.1	Entrada . . . . .	10
5.4.2	Saída . . . . .	10
5.4.3	Comentários . . . . .	11
5.5	Funções . . . . .	11
<b>6</b>	<b>Programas de Exemplo</b>	<b>12</b>
6.0.1	Hello World . . . . .	12
6.0.2	Shell sort . . . . .	12
6.0.3	Série de Fibbonaci . . . . .	13

# 1 Introdução

## 2 Estrutura Geral do Programa

### 2.1 Ponto Inicial do Programa

O ponto inicial do programa é a função **main** que é obrigatória e possui algumas particularidades como:

- A função **main** deve ser do tipo **int**;
- Não pode ter parâmetros;
- Só pode existir uma função **main**;
- Ela deve sempre retornar 1.

```
1  function int main (){
2      .
3      .
4      .
5      return 1;
6  }
```

Figura 1: Estrutura da função inicial do programa

### 2.2 Definição de funções

As funções podem ser declaradas em qualquer lugar do código, entretanto não podem ser declaradas dentro de outras funções e o modo de passagem de parâmetros é por cópia. Além disso, o escopo de cada função é delimitado pela abertura e fechamento de chaves.

- A declaração de uma função deve ser iniciada pela palavra reservada **function**, seguida de: tipo de retorno (**int**, **bool**, **float**, **char**, **string**), identificador, lista de parâmetros entre ' ( ' e ' ) ' e seus itens separados por ' , ' , seu corpo entre ' { ' e ' } '.

Exemplo:

```
1  function bool id_function (int a, int b, float c){
2      .
3      .
4      .
5      return true;
6  }
```

Figura 2: Estrutura de uma função

### 2.3 Definição de instruções

As instruções só podem ser declaradas dentro do escopo de funções, além de que cada instrução deve ser finalizada com ' ; '.

### 2.4 Estrutura da nomeação de variáveis e funções

Os nomes das funções e variáveis devem ser iniciados por letras [a - zA- Z] e depois podem conter letras [a - zA- Z], dígitos [0 - 9] e o *underline* ('\_'). A linguagem é *case sensitive*.

```
1  int nome_da_variavel;
2
3  bool minhaVariavel_3;
4
```

Figura 3: Nomeação de variáveis

## 2.5 Símbolos especiais

Os símbolos especiais aceitos na linguagem são (no formato flex):

- ‘‘ ‘ ; ‘ , ‘ . ‘ : ‘ ? ‘ ! ‘ + ‘ - ‘ \* ‘ \ \ ‘ / ‘ \_ ‘ % ‘ @ ‘ & ‘ # ‘ \# ‘ \$ ‘ < ‘ > ‘ = ‘ ( ‘ ( ‘ ) ‘ [ ‘ [ ‘ ] ‘ { ‘ { ‘ } ‘ | ‘ | ‘ \ ‘ \ “ ‘ ^ ‘ \n ;

### 3 Conjunto de tipos de dados e nomes

A especificação de tipos é estática.

### 3.1 Palavras Reservadas

As palavras reservadas da linguagem são:

main, function, return, if, elsif, else, for, while, null, break, void, void, int, float, char, bool, string, array, read, write, true, false.

### 3.2 Inteiro

O tipo inteiro é definido pela palavra reservada **int**, representando um número inteiro de tamanho 32 bits. Pode receber uma Constante inteira ou outra variável do mesmo tipo. Seus literais são expressos da seguinte forma:

[0-9]+

Possui a seguinte declaração e atribuição:

```
1  int var_name;  
2  
3  var_name = 4;
```

### 3.3 Ponto flutuante

O tipo ponto flutuante é definido pela palavra reservada **float**, representando um número decimal de tamanho 64 bits. Pode receber uma Constante de ponto flutuante ou outra variável do mesmo tipo. Seus literais são expressos da seguinte forma:

[0-9]+ '\.' [0-9]+

Possui a seguinte declaração e atribuição:

```
1 float var_name;
2
3 var_name = 32.2;
4
```

### 3.4 Booleano

O tipo booleano é definido pela palavra reservada **bool**, e tem tamanho de 1 byte por ter apenas dois valores possíveis ('true', 'false'). Pode receber uma Constante Booleana ou outra variável do mesmo tipo. Seus literais são expressos da seguinte forma:

`('true' | 'false')`

Possui a seguinte declaração e atribuição:

```
1 bool var_name, var_name2;  
2  
3 var_name = true;  
4 var_name2 = false;
```

### 3.5 Char

O tipo caractere é definido pela palavra reservada **char**, tem tamanho de 1 byte e guarda um código referente ao seu símbolo na tabela ASCII. Pode receber uma Constante Char ou outra variável do mesmo tipo. Seus literais são expressos da seguinte forma:

`('\'') (( '{letra}' | '{símbolo}' | '{dígito}') ('^'#')) ('\'')`

Em que letra

- letra [a-zA-Z];
- dígito [0-9];
- símbolos [ ' | ' | ' ; | ' | ' , | ' | ' . | ' | ' : | ' | ' ? | ' | ' ~ | ' | ' ! | ' | ' + | ' | ' - | ' | ' # | ' | ' \* | ' | ' \ | ' | ' / | ' | ' \_ | ' | ' % | ' | ' @ | ' | ' & | ' | ' \# | ' | ' # | ' | ' \$ | ' | ' < | ' | ' > | ' | ' = | ' | ' ( | ' | ' ) | ' | ' [ | ' | ' ] | ' | ' { | ' | ' } | ' | ' | ' \ | ' | ' \ " | ' | ' ^ | ' | ' \n ];

Possui a seguinte declaração e atribuição:

```
1 char var_name;  
2  
3 var_name = 'b';  
4
```

### 3.6 String

O tipo string é responsável por uma cadeia de caracteres cujos códigos estão na tabela ASCII e é definido pela palavra reservada **string**. Pode receber uma Constante String ou outra variável do mesmo tipo e o seu tamanho varia de acordo com a última cadeia de caracteres atribuída. Seus literais são expressados da seguinte forma:

`('\"') (( '{letra}' | '{símbolo}' | '{dígito}')*) ('\"')`

Em que letra

- letra [a-zA-Z];
- dígito [0-9];
- símbolos [ ' | ' | ' ; | ' | ' , | ' | ' . | ' | ' : | ' | ' ? | ' | ' ~ | ' | ' ! | ' | ' + | ' | ' - | ' | ' # | ' | ' \* | ' | ' \ | ' | ' / | ' | ' \_ | ' | ' % | ' | ' @ | ' | ' & | ' | ' \# | ' | ' # | ' | ' \$ | ' | ' < | ' | ' > | ' | ' = | ' | ' ( | ' | ' ) | ' | ' [ | ' | ' ] | ' | ' { | ' | ' } | ' | ' | ' \ | ' | ' \ " | ' | ' ^ | ' | ' \n ];

Possui a seguinte declaração e atribuição:

```

1  string var_name;
2
3  var_name = 'testando string';
4

```

### 3.7 Arranjos unidimensionais

O tipo arranjo unidimensional é responsável por um array de um tipo e é definido pela palavra reservada **array**. O seu tamanho deve ser definido durante a declaração da variável, é estático e cada posição do array recebe o valor **null** durante a declaração.

Para a atribuição, poderá tanto atribuir todos os itens de uma única vez entre '[' e ']', ou atribuir um valor para cada posição do *array*, acessado com o número do índice entre '[' e ']'. O índice de cada *array* começa com 0 e último índice acessível é equivalente ao tamanho do *array* menos um.

Declaração:

```

1  array int array_teste [tamanho];

```

Exemplo de declaração, atribuição e acesso na linguagem:

```

3  array int lista[3];
4
5  lista[0] = 1;
6
7  lista[1] = 2;
8
9  lista[2] = 3;
10
11 write("\ni", lista[0]);

```

Exemplo de passagem por parâmetro

```

8  function int main(){
9      array int list[4];
10     list[4] = [2,5,78,3];
11     a_func(4, list);
12     return 1;
13 }

```

Exemplo de definição em parâmetro de função:

```

2  function void a_func(int size, array int aux[size]){
3      .
4      .
5      .
6  }

```

Como visto na imagem acima, ao definir um array no parâmetro de uma função no qual seu tamanho vai ser passado por parâmetro, deve-se passar primeiro a variável com o tamanho para que em seguida o array seja declarado com tamanho.

### 3.8 Variáveis

Em uma declaração de variável, é possível definir mais de uma variável na mesma instrução utilizando o separador vírgula, desde que as variáveis sejam do mesmo tipo. Entretanto, a linguagem não suporta a inicialização das variáveis durante a declaração, sendo atribuído o valor **null** por default para cada variável declarada.

Exemplo de declaração na mesma linha:

```
2  int a,b;
3  a = 3;
4  b = 234;
```

### 3.9 Operações de cada tipo

As operações não realizam coerção. Segue a tabela com todas as operações suportadas pela linguagem:

Operador	Tipos que realizam a operação
'^' '%'	int
'*' '/' '-' '+'	int, float
'<' '<=' '>' '>='	int, float
'==' '!='	int, float, bool, char, string
'&&' '  ' '!'	bool
'@'	string

### 3.10 Valores padrão

Tipo	Valor
int	0
float	0.0
char	null
string	null
bool	false

## 4 Conjunto de Operadores

### 4.1 Tipos das Operações

Os operandos de uma operação devem obrigatoriamente ser do mesmo tipo, logo o retorno de uma operação será do mesmo tipo dos operandos, (válido para todas as operações).

#### 4.1.1 Operadores Aritméticos

Operação	símbolo	Exemplo
Adição	+	var_a + var_b
Subtração	-	var_a - var_b
Unário negativo	-	-var_a
Multiplicação	*	var_a * var_b
Divisão	/	var_a / var_b
Exponenciação	^	var_a ^ var_b
Módulo	%	var_a % var_b

#### 4.1.2 Operadores Relacionais

Operação	símbolo	Exemplo
Igual	==	var_a == var_b
Diferente	!=	var_a != var_b
Menor que	<	var_a < var_b
Maior que	>	var_a > var_b
Menor ou igual	≤	var_a ≤ var_b
Maior ou igual	≥	var_a ≥ var_b

#### 4.1.3 Operadores Lógicos

Operação	símbolo	Exemplo
conjunção	&&	var_a && var_b
Disjunção		var_a    var_b
Negação	!	!var_a

#### 4.1.4 Operadores de Concatenação

Operação	símbolo	Exemplo
concatenação*	@	var_a @ var_b

\*Apenas para cadeia de caracteres.

### 4.2 Ordem de precedência e associatividade

A ordem de precedência vai de cima para baixo e tem a seguinte tabela:

Ordem de precedência	Associatividade
'!' (Negação)	Direita para esquerda
'-' (Operador unário negativo)	Direita para esquerda
'^' (Exponenciação)	Direita para esquerda
'*', '/' e '%' (Operadores aritméticos)	Esquerda para direita
'+' e '-' (Operadores aritméticos)	Esquerda para direita
'<', '<=', '>' e '>=' (Operadores relacionais)	Não Associativo
'==', '!=' (Operadores relacionais)	Não Associativo
'&&' (Conjunção)	Esquerda para direita
'  ' (Disjunção)	Esquerda para direita
'@' (concatenação)	Nenhuma

## 5 Instruções

### 5.1 Atribuição

A instrução de atribuição é definida pelo operador '@', sendo o lado esquerdo o identificador a receber o valor e o lado direito o valor ou expressão a ser atribuído. Ambos devem possuir o mesmo tipo.

Exemplo da instrução de atribuição:

```
2 int var_a = 4;
```



## 5.2 Estrutura Condicional de uma e duas vias

Cada estrutura condicional deverá possuir obrigatoriamente no início um **if** que é a condição inicial, seguido da expressão lógica entre '(' e ')'.

Exemplo da estrutura condicional **if**:

```
1  if(expressao_logica){
2      ...
3  }
```

Há também a estrutura **elsif** que deverá estar entre **if** e **else** e também é precedida pela expressão lógica entre '(' e ')', podendo haver mais de um **elsif** entre eles.

Exemplo da estrutura condicional **elsif**:

```
15  if(expressão_logica){
16      ...
17  }
18  elsif(expressão_logica){
19      ...
20  }
21  else{
22      ...
23  }
```

Por fim, a ultima estrutura é o **else** que abrange todas as condições não abordadas pelas estruturas anteriores.

Exemplo da estrutura condicional **else**:

```
7   if(expressao_logica){
8       ...
9   }
10  else{
11      ...
12  }
```

Todas as estruturas condicionais (if, elsif e else) devem possuir chaves para a abertura e fechamento do seu escopo.

## 5.3 Estrutura condicional de controle lógico

Definida pela palavra reservada **while**, seguida pela expressão lógica entre '(' e ')'. Enquanto essa expressão lógica for 'true', a estrutura irá iterar.

Exemplo da estrutura condicional com controle lógico:

```
1  while(expressao_logica){
2      ...
3  }
```

A estrutura deve possuir chaves para a abertura e fechamento do seu escopo.

## 5.4 Estrutura iterativa controlada por contador

Definida pela palavra reservada **for**. Essa é uma estrutura é dividido em 4 partes:

1. Declaração da variável contador (apenas tipo **int**;
2. Valor inicial do contador (**int**);
3. Valor limitante que o contador pode alcançar, ultrapassand ele a estrutura é encerrada;
4. valor do incremento que acontecerá em cada repetição.

Além disso, cada parte é separada por um ','.

Na primeira e segunda parte a linguagem aceita apenas variáveis inteiras e na terceira parte deverá apenas ter o valor inteiro do passo, caso seja negativo deverá utilizar o operador unário '-'. Por fim, a estrutura **for** encerra quando o contador alcançar o valor limitante.

Exemplo da estrutura iterativa controlada por contador:

```
1  for(int i, a, b, c){
2      ...
3  }
```

A estrutura deve possuir chaves para a abertura e fechamento do seu escopo.

### 5.4.1 Entrada

A instrução de entrada, definida pela palavra reservada **read**, poderá receber um ou mais parâmetros e não é necessário especificar o seus tipos, apenas realizar uma separação entre cada variável pelo símbolo especial ','.

Exemplo da instrução de entrada:

```
1  read(a,b,c);
```

### 5.4.2 Saída

A instrução de saída, definida pela palavra reservada **write**, deverá estar entre aspas duplas e poderá receber um ou mais parâmetros. Ao inserir variáveis é necessário inserir '#' seguido do código responsável pela formatação de cada tipo, além disso cada variável deve ser separada por ',' , após o fechamento das aspas duplas.

Exemplo da instrução de saída:

```
5  a = "Lucas";
6  b = 'L';
7
8  write ("Ola \#s tudo bem? Minha letra e \#c", a, b);
```

Segue abaixo a tabela com os códigos para a utilização de variáveis na instrução de saída:

Formatação	
#i	Para inteiros
#f	Para números flutuantes com 2 casas decimais
#f.d	Para números flutuantes com d casas decimais
#c	Para caracteres
#s	Para string
#b	Para boolean

Para utilizar algum símbolo especial que não é possível normalmente, deverá utilizar o caractere de escape '\ '. Por exemplo: `write("Esse e o \\ (backslash) e esse e o \# (jogo da velha)");`

### 5.4.3 Comentários

O comentário inicia com o símbolo '\$' e vai até o final da linha.

Exemplo de um comentário:

```
var_a = "Lucas"; $variavel a recebe lucas
```

## 5.5 Funções

As funções podem ser declaradas em qualquer lugar do código (menos dentro de outras funções) e seu escopo é delimitado pelas chaves '{ }' que são obrigatórias.

Cada função deve ser iniciada pela palavra reservada 'function', seguido pelo seu tipo por parâmetros. Ela pode ter ou não parâmetros e caso tenha ela deve especificar entre '(' )' cada parâmetro e seu tipo, além de separá-los por ','.

Estrutura da declaração da função:

```
1  function tipo nome_funcao (tipo param1, tipo param2){
2      ...
3      return x;
4
5  }
```

Exemplo de função:

```
1  function bool id_function (int a, int b, float c){
2      .
3      .
4      .
5      return true;
6  }
```

A função pode ter os seguintes tipos: **int**, **bool**, **float**, **char** e **void**. Ela deverá utilizar a palavra reservada **return** para especificar a expressão de retorno (obrigatório) o qual deverá obedecer o tipo da função.

Para usar uma função, basta chamá-la em qualquer função e colocando os nomes das variáveis (ou os valores) dentro dos parâmetros (se existirem).

Exemplo:

```

8  function int main(){
9      array int list[4];
10     list4[4] = [2,5,78,3];
11     a_func(4, list);
12     return 1;
13 }

```

Por fim, para especificar um array em um parâmetro deverá ser utilizada a mesma formatação da declaração do array, com a palavra reservada 'array', o seu tipo, o nome da variável e o tamanho entre colchetes.

## 6 Programas de Exemplo

### 6.0.1 Hello World

```

function int main () {
    write ("Hello World!");
    return 1;
}

```

### 6.0.2 Shell sort

```

function void shellSort () {

    int gap, i, j , aux, size;

    size = 5;

    array int lista[size];

    lista[size] = {85,56,7,23,4};

    gap = 1;

    while(gap < size){
        gap = 3*gap+1;
    }

    while(gap > 0){
        for(int i, gap, size,1){
            aux = lista[i];
            j = i;

            while(j > gap-1 && aux <= lista[j-gap]){
                lista[j] = lista[j - gap];
                j = j - gap;
            }

            lista[j]=aux;
        }

        gap = gap/3;
    }

    return;
}

```

```

function int main(){
    shellSort();
    return 1;
}

```

### 6.0.3 Série de Fibbonaci

```

function void fibonacci(int n) {
    int a, b, c;
    string space;
    a = 0;
    b = 1;
    if(n == 0) {
        write("Digite um valor maior que zero!\n");
    }
    elseif(n == 1) {
        write("0\n");
    }
    else {
        write("0, 1");
        c = a + b;
        while(c < n) {
            if(c < n) {
                write(", #d", c);
            }
            else {
                write("\n");
                break;
            }
            a = b;
            b = c;
        }
    }
    return;
}

function int main () {
    write("Hello World!");
    return 1;
}

```