

# Documentação Linguagem L--

Lucas Montenegro e Letícia Medeiros

2020.1

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

# Conteúdo

<b>1</b>	<b>Estrutura Geral do Programa</b>	<b>3</b>
1.1	Ponto Inicial do Programa . . . . .	3
1.2	Definição de funções . . . . .	3
1.3	Definição de instruções . . . . .	3
1.4	Variáveis . . . . .	3
1.5	Estrutura da nomeação de variáveis e funções . . . . .	4
<b>2</b>	<b>Conjunto de tipos de dados, variáveis e palavras reservadas</b>	<b>4</b>
2.1	Palavras Reservadas . . . . .	4
2.2	Expressões Regulares Auxiliares . . . . .	4
2.3	Inteiro . . . . .	4
2.4	Ponto flutuante . . . . .	4
2.5	Booleano . . . . .	5
2.6	Char . . . . .	5
2.7	String . . . . .	5
2.8	Arranjos unidimensionais . . . . .	5
<b>3</b>	<b>Conjunto de Operadores</b>	<b>6</b>
3.1	Tipos das Operações . . . . .	6
3.1.1	Operadores Aritméticos . . . . .	7
3.1.2	Operadores Relacionais . . . . .	7
3.1.3	Operadores Lógicos . . . . .	7
3.1.4	Operador de Concatenação . . . . .	7
3.2	Operações de cada tipo . . . . .	7
3.3	Valores padrão . . . . .	7
3.4	Ordem de precedência e associatividade . . . . .	7
<b>4</b>	<b>Instruções</b>	<b>8</b>
4.1	Atribuição . . . . .	8
4.2	Palavra reservada break . . . . .	8
4.3	Estrutura Condicional de uma e duas vias . . . . .	8
4.4	Estrutura condicional de controle lógico . . . . .	9
4.5	Estrutura iterativa controlada por contador . . . . .	9
4.5.1	Entrada . . . . .	10
4.5.2	Saída . . . . .	10
4.5.3	Comentários . . . . .	11
4.6	Funções . . . . .	11
<b>5</b>	<b>Programas de Exemplo</b>	<b>12</b>
5.0.1	Hello World . . . . .	12
5.0.2	Shell sort . . . . .	13
5.0.3	Série de Fibbonaci . . . . .	14

# 1 Estrutura Geral do Programa

## 1.1 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**;

```
13  int main(){
14      ...
15      return 0;
16  }
```

Figura 1: Estrutura da função inicial do programa

## 1.2 Definição de funções

As funções podem ser declaradas em qualquer lugar do código, entretanto não podem ser declaradas dentro de outras funções e são visíveis a partir do momento que são declaradas.

O modo de passagem de parâmetros de arrays é de entrada e saída e por referência. Já para os tipos **int**, **float**, **char**, **boolean** e **string**, o modo de passagem de parâmetros é de entrada e por valor.

Além disso, o corpo 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

## 1.3 Definição de instruções

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

## 1.4 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;
```

## 1.5 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 Conjunto de tipos de dados, variáveis e palavras reservadas

A especificação de tipos é estática e todos os tipos e estruturas possuirão compatibilidade por nome.

## 2.1 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.

## 2.2 Expressões Regulares Auxiliares

Segue abaixo as expressões regulares auxiliares:

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

## 2.3 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:

dígito+

Possui a seguinte declaração e atribuição:

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

## 2.4 Ponto flutuante

O tipo ponto flutuante é definido pela palavra reservada **float**, representando um número de ponto flutuante, utilizando 64 bits de tamanho, de acordo com o padrão IEEE 754. Pode receber uma Constante de ponto flutuante ou outra variável do mesmo tipo. Seus literais são expressos da seguinte forma:

dígito+ '\.' dígito+

Possui a seguinte declaração e atribuição:

```

1 float var_name;
2
3 var_name = 32.2;
4

```

## 2.5 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;

```

## 2.6 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}')`

Possui a seguinte declaração e atribuição:

```

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

```

## 2.7 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}')+) ('\"')`

Possui a seguinte declaração e atribuição:

```

8 string var_name;
9
10 var_name = "testando string";

```

## 2.8 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 e o mesmo é armazenado não sendo necessário especificar o valor após a declaração, além disso a estrutura é estática 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:

```
13 array int lista[3];
14
15 lista[0] = 23;
16 lista[1] = 14;
17 write("#i", lista[0]);
```

Para ler um array é necessário utilizar uma das estruturas de repetição para capturar item por item, da mesma forma para imprimir um array na tela.

Exemplo de passagem por parâmetro

```
1 function void a_func(array int aux[]){
2     ...
3 }
4
5 int main(){
6     array int lista[18];
7     a_func(lista);
8     return 0;
9 }
```

Como visto na imagem acima, o array é passado por referência na passagem por parâmetro da função "a\_func".

## 3 Conjunto de Operadores

### 3.1 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 com exceção da concatenação).

A concatenação é a única operação que realiza "coerção". Internamente, é utilizado um método semelhante ao toString da linguagem Java para transformar os tipos para **string**.

### 3.1.1 Operadores Aritméticos

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

### 3.1.2 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
Menor ou igual	≥	var_a ≥ var_b

### 3.1.3 Operadores Lógicos

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

### 3.1.4 Operador de Concatenação

Operação	símbolo	Exemplo
concatenação*	@	var_a @ var_b

## 3.2 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
'@'	int, float, char, string

## 3.3 Valores padrão

## 3.4 Ordem de precedência e associatividade

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

Tipo	Valor
int	null
float	null
char	null
string	null
bool	null

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

## 4 Instruções

### 4.1 Atribuição

A instrução de atribuição é definida pelo símbolo '=', 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;
```

### 4.2 Palavra reservada break

A palavra reservada **break** é utilizada para sair da estrutura de repetição mais interna.

Exemplo do uso do **break**:

```
1 while(true){
2     while(true){
3         break;
4     }
5 }
```

Nesse exemplo acima, a utilização do **break** faz com que o fluxo do programa saia da estrutura de repetição da linha 2.

### 4.3 Estrutura Condicional de uma e duas vias

Cada estrutura condicional deverá possuir obrigatoriamente no início um **if** que é a condição inicial, seguido de uma expressão lógica entre '(' e ')'. Uma expressão lógica é aquela cujo resultado da avaliação é um valor do tipo **bool**.

Caso exista mais de uma condição onde a expressão lógica atenda os requisitos, o fluxo do programa deverá entrar na primeira estrutura que atender o requisito. Ao finalizar o bloco de código daquele condicional, o fluxo do programa sai daquela estrutura condicional.

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 zero ou mais **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, podendo haver 0 ou 1 **else**.

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.

#### 4.4 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.

#### 4.5 Estrutura iterativa controlada por contador

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

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

Na primeira parte, a variável declarada é visível apenas para o controle da estrutura iterativa, não sendo visível dentro do escopo do **for** nem fora dele. As outras partes deverão ser constantes literais do tipo **int** e da mesma forma que na variável da parte 1, elas são visíveis apenas ao controle da estrutura iterativa.

Cada parte dessa estrutura é dividida pelo separador **','** e a estrutura **for** é encerrada quando o contador ultrapassar 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.

#### 4.5.1 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);
```

#### 4.5.2 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)\n");`

O `\n` serve para quebra de linha.

### 4.5.3 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
```

## 4.6 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:

```
1  function void a_func(array int aux[]){
2      ...
3  }
4
5  int main(){
6      array int lista[18];
7      a_func(lista);
8      return 0;
9  }
```

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.

## 5 Programas de Exemplo

### 5.0.1 Hello World

```
1  function int main () {  
2  
3      write ("Hello World!");  
4      return 0;  
5  
6  }
```

## 5.0.2 Shell sort

```
2  function void shellSort (int size, int array lista[]){
3      int gap, i, j , aux;
4      gap = 1;
5
6      while(gap < size){
7          gap = 3*gap+1;
8      }
9      while(gap > 0){
10         for(i = gap, size,1){
11             aux = lista[i];
12             j = i;
13             while(j > gap-1 && aux <= lista[j-gap]){
14                 lista[j] = lista[j - gap];
15                 j = j - gap;
16             }
17             lista[j]=aux;
18         }
19         gap = gap/3;
20
21     }
22 }
23
24 function int main(){
25     int n;
26     read(n);
27     array int lista[n];
28     for(int i, 0, n, 1){
29         read(lista[i]);
30     }
31     for(int i, 0, n, 1){
32         write("#i", lista[i]);
33     }
34     shellSort(n, lista);
35     for(int i, 0, n, 1){
36         write("#i", lista[i]);
37     }
38     write("\n");
39     return 0;
40 }
```

### 5.0.3 Série de Fibonacci

```
1  function void fibonacci(int n) {
2      int a, b, c;
3      string space;
4      a = 0;
5      b = 1;
6
7      if(n == 0) {
8          write("#i\n", n);
9      }
10     elseif(n == 1) {
11         write("0, #i\n", n);
12     }
13     else {
14         write("0, 1");
15         c = a + b;
16         while(c < n) {
17             if(c < n) {
18                 write(", #d", c);
19             }
20             else {
21                 write("\n");
22                 break;
23             }
24             a = b;
25             b = c;
26         }
27     }
28     return;
29 }
30
31 function int main () {
32     int n;
33     read(n);
34     fibonacci(n);
35     return 1;
36 }
```