

```
1
2
3  Algoritmo 'Lógica de' {
4
5      [ Programação 1 ]
6
7
8
9
10
11
12 }
13
14
```

Unidade 04

Prof. Daniel Caixeta



```
1  'Sumário' {
2
3      </01> Primeiros passos em C
4          < 1.1. Conceitos. 1.2. Tipos de dados e variáveis. 1.4. Formato de leitura de
5              variável. 1.4. Práticas. >
6
7      </02> Expressões comparativas & lógicas
8          < 2.1. Expressões de comparações. 2.2. Expressões lógicas. 2.3. Operadores
9              lógicos. >
10
11     </03> Estrutura condicional
12         < 3.1. Conceitos. 3.2. O encadeamento [...]. 3.3. Estrutura condicional
13             em C. >
14     Referência
15 }
```

1 01 {  
2  
3

4  
5 [ Primeiros passos em C ]  
6  
7

8 < Conceitos, sintaxe, primeiras linhas >  
9  
10

11  
12 }  
13  
14

## 1.1. Conceitos [...]

O que é uma linguagem de programação?

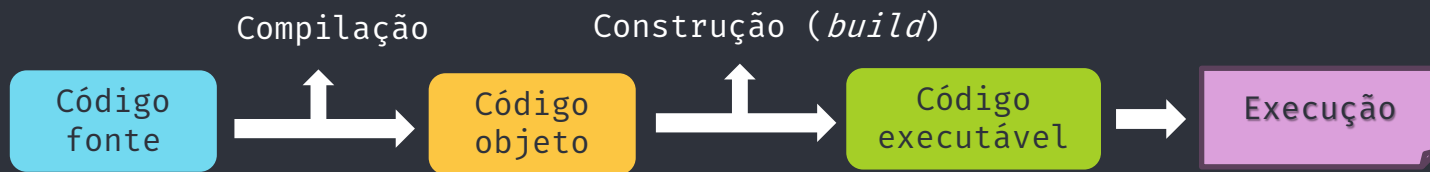
- É uma linguagem artificial e formal usada para comunicar instruções a um computador.
- Assim como uma linguagem natural, possui uma sintaxe, devendo ser rigorosamente seguida, sendo que em qualquer desvio, haverá impossibilidade do computador interpretar o código e executar o programa.
- A partir de exemplos práticos, vamos aprender a sintaxe desta linguagem.

## O que é um programa em linguagem C?

- É um texto cujo o conteúdo segue a sintaxe da linguagem C.

1º programa em C – “Hello World!”

```
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}
```



Compilação é o processo de tradução de um programa (código-fonte) para uma linguagem de máquina (código-objeto), para que suas instruções sejam executadas pelo processador (código executável).

- O programa `HelloWorld.c` ou `main.c` possui o sintaxe/formato básico de um programa em C.

```
1  #include <stdio.h>
2
3
4  int main()
5  {
6      /* Declaração de variáveis */
7      /* Sentenças ou lógica do programa */
8
9      return 0;
10 }
11
12
13
14
```

Diagram annotations:

- `#include <stdio.h>` → Declaração da biblioteca
- `int main()` → Inicia o programa e retorna um inteiro para o S.O.
- Corpo do código (bracketed lines 6-10)
- `return 0;` → **return:** Encerra a função main retornando para o S.O.
- Se retornar 0 (zero) programa bem sucedido, SENÃO, há uma condição de erro no código.

- O ponto e vírgula (;) não é exceção é REGRA!!!

## 1.2. Tipos de dados e variáveis

- Um programa minimamente útil precisa manipular dados na memória do computador.
- A memória armazena uma imensa sequência de bits, e sua representação é indiferente para o funcionamento da máquina.
- Linguagens como C, entretanto, facilitam a vida do programador disponibilizando tipos de dados em bibliotecas.
- Um tipo de dado é um conjunto de possíveis valores e operações que podem ser efetuadas.
- A linguagem C nos oferece alguns tipos de dados *primitivos*.

## Os principais tipos

Tipo	Nome	Número de bits	Intervalo	
			Início	Fim
char()	Caractere	8	-128	127
int()	Inteiro	16	-32.768	-32.767
float()	Flutuante	32	3,4E-38	3.4E+38
double()	Duplo	64	1,7E-308	1,7E+308
void()	Nulo	-	-	-
long int()	Inteiro	32	-2.147.483.648	2.147.483.647
long double()	Duplo	80	3,4E-4932	3,4E+4932



- Na linguagem C é permitido declarar variáveis, que possuem um tipo determinado.
- Declarar uma variável é como dar um nome a um pedaço da memória que guardará um valor de determinado tipo. Daí em diante podemos acessar o valor guardado e manipulá-lo através do nome da variável. Por exemplo:

Exemplo <1>

```
int x, y, soma;
```

```
x = 2;
```

```
y = 4;
```

```
soma = x + y;
```

Exemplo <2>

```
int x = 5
```

```
int y, z;
```

```
y = 1 + 2 * x * (3 - 2 * x);
```

```
z = x * y;
```

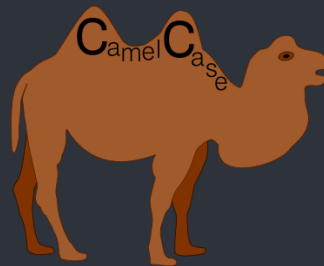
```
printf("y = ", y);
```

```
printf("x = ", z);
```

- O nome de uma variável pode ser qualquer sequência de letras e números e alguns símbolos como `_`.
- Um nome não pode começar com números e conter espaços. O uso de maiúsculas e minúsculas é permitido. Lembre-se que as linguagens de programação são *sensitive case*, ou seja, MAIÚSCULAS diferentes de minúsculas, de modo que soma e sOma são variáveis diferentes.

### Lembre-se !?!

- Não pode começar com dígito: use uma letra ou `_`.
- Não pode ter espaço em branco.
- Não usar acentos ou til.
- Sugestão: use o padrão “Camel Case”.



Exemplos: Soma, x, y, x12, x\_12, soma\_abc, nome\_bem\_grande, NomeBemGrande

## 1.3. Formatos de leitura de variável

- Os formatos de leitura das funções `printf()` e `scanf()` são bem semelhantes. Na tabela abaixo vejam alguns formatos possíveis de leitura e escrita.

Código	Função
%c	Lê um único caractere.
%s	Lê uma série de caracteres.
%d	Lê um número decimal.
%u	Lê um decimal sem sinal.
%ld	Lê um inteiro longo.
%f	Lê um número em ponto flutuante.
%lf	Lê um número double.

## 1.4. Práticas

### 1 Exercício <1>:

- Faça um programa que realiza operações aritméticas com variáveis inteiras (a, b e c) atribuindo valores a essas constantes:

```
6  /* Programa que calcula operações aritméticas  
7  básicas. Manipulação de dados na memória.
```

```
8  Autor: dfcaixeta - Data: 06.mai.2024 */
```

```
9  #include <stdio.h>
```

```
10 #include <locale.h>
```

```
11 int main(){
```

```
12     setlocale(LC_ALL, "");
```

```
13     int a, b = 5, c = 4;
```

```
14
```

```
a = b + c;
```

```
b = a + 6;
```

```
c = b - a;
```

```
printf("\nOs valores das operações são:\n");
```

```
printf("a = %d\n", a);
```

```
printf("b = %d\n", b);
```

```
printf("c = %d\n\n", c);
```

```
return 0;
```

```
}
```

## Exercício <2>:

- Qual o valor armazenado na variável `a`?

```
/* Programa que calcula o valor da variável a ao final da execução.  
Autor: dfcaixeta - Data: 06.mai.2024 */
```

```
#include <stdio.h>  
#include <locale.h>
```

```
int main(){
```

```
    setlocale(LC_ALL, "");  
    int a, b = 4, c = 2, d = 3;
```

Necessário?!?! 

```
    d = c + b;  
    a = d + 1;  
    a = a + 1;
```

```
    printf("\nO valor de a é: %d", a);  
    return 0;
```

```
}
```

## Exercício <3>:

- Escreva o código abaixo e indique qual o ERROR!?

```
/* Qual o erro nesse código? ... Descubra.  
Autor: dfcaixeta - Data: 06.mai.2024 */
```

```
#include <stdio.h>  
#include <locale.h>
```

```
int main(void) {  
    setlocale(LC_ALL, "");
```

```
    int a, b = 4;  
    double c = 2.4142, d = 3.1416;  
    int g;
```

```
    // Cálculos das operações ...  
    d = b + 5;  
    e = c * d;  
    a = b + 1;
```

```
printf("\n0 valor de b é: %d", b);  
printf("\n0 valor de a é: %d", a);  
printf("\n0 valor de e é: %.3f", e);  
  
return 0;
```

```
}
```

## Exercício <4>:

- Faça programa que lê um caractere, um número ponto flutuante e por fim um decimal e imprima na tela os dados lidos.

```
1  /* Programa que lê um char, float, int e imprime na tela os dados lidos.
2  Data: 14.mai.2024
3  Autor: Daniel Caixeta <@dfcaixeta> */
4
5  # include <stdio.h>
6  # include <stdlib.h>
7
8  int main(){
9      char c;
10     float b;
11     int a;
12
13     printf("\nDigite um caractere: ");
14     scanf("%c", &c);
15
16     printf("Digite um numero ponto flutuante: ");
17     scanf("%f", &b);
18
19     printf("Digite um numero inteiro: ");
20     scanf("%d", &a);
21
22     printf("\nOs dados lidos foram: %c, %f, %d\n\n", c, b, a);
23
24     system("pause");
25     return 0;
26 }
```

## Exercício <5>:

- Faça um programa que pede ao usuário dois números e depois imprime na tela a média aritmética entre eles:

```
/* Programa que calcula a media entre dois números.
```

```
Autor: @dfcaixeta - Data: 07.mai.2024. */
```

```
// Declaração das bibliotecas
```

```
# include <stdio.h>
```

```
# include <locale.h>
```

```
// Corpo do código ..
```

```
int main()
```

```
{
```

```
    setlocale(LC_ALL, "");
```

```
    double num1, num2, media;
```

```
    printf("\nEntrada dos valores: \n");
```

```
    printf("Digite o primeiro número: ");
```

```
    scanf("%lf", &num1);
```

```
    printf("Digite o segundo número: ");  
    scanf("%lf", &num2);
```

```
    // Lógica do cálculo ...
```

```
    media = (num1 + num2) / 2.0;
```

```
    // Imprime os resultados do cálculo ...
```

```
    printf("\nResultado: ");
```

```
    printf("\nMédia = %.1f\n\n", media);
```

```
    return 0;
```

```
}
```



## Exercício <6>:

- Faça um programa que realiza as operações aritméticas básicas (soma, subtração, multiplicação e divisão):

```
/* Programa que realiza operações aritméticas
entre dois números.
Autor: @dfcaixeta - Data: 07.mai.2024 */
```

```
# include <stdio.h>
# include <locale.h>
```

```
int main() {
```

```
    setlocale(LC_ALL, "");
    double num1, num2;
```

```
    printf("\n ** Entre com os valores: ** \n");
    printf("Digite o primeiro número: ");
    scanf("%lf", &num1);
```

```
    printf("Digite o segundo número: ");
    scanf("%lf", &num2);
```

```
--> printf("\n ** Os valores são: **");
      printf("\nSoma = %.2lf\n", num1 + num2);
      printf("Subtração = %.2lf\n", num1 - num2);
      printf("Multiplicação = %.2lf\n", num1 * num2);

      if (num2 != 0) {
          printf("Divisão = %.2lf\n\n", num1 / num2);
      } else {
          printf("\nNão é possível dividir por zero.\n\n");
      }
      return 0;
    }
```

02

{

[ Expressões comparativas &  
Lógicas ]

< Operadores de comparação e Lógicos >

}

## 2.1. Expressões de comparação

- Os operadores de comparação são símbolos usados em programação para comparar valores e expressões.
- Retornam um valor booleano (V ou F).
- Esses operadores são fundamentais para controle de fluxo e tomada de decisões em algoritmos/programas.

Operador	Significado	Operador	Significado
>	Maior	<=	Menor ou igual
<	Menor	=	Igual
>=	Maior ou igual	<>	Diferente

Operadores de comparação em VisualG.

- Por exemplo:

Expressão	V	F
$5 \leq 8$		
$10 > 12$		
$8 \wedge 2 = 64$		
$(2 * 4) <> (9 - 1)$		

- Os operadores de comparação são amplamente utilizados em estruturas de controle condicional, como instruções *if*, *else if* e *while*, para tomar decisões com base nas relações entre os valores das variáveis.

## 2.2. Expressões lógicas

- São combinações de símbolos e operadores lógicos que representam relações entre proposições ou valores lógicos.
- Essas expressões são usadas na lógica formal e na programação para representar condições, decisões e raciocínios.
- Os operadores lógicos comuns incluem **AND (E)**, **OR (OU)** e **NOT (NÃO)**, que são usados para combinar proposições ou inverter seu valor lógico.
- Por exemplo:  
    "Se estiver chovendo **OU** se estiver ensolarado, vou levar um guarda-chuva!"

# 2.3. Operadores lógicos

Operador	Significado
E (AND)	Verdadeiro se todas as condições forem verdadeiras.
OU (OR)	Verdadeiro se pelo menos uma condição for verdadeira.
NÃO (NOT)	Verdadeiro se a condição for falsa.

Operador NOT

A	S ou A'
0	1
1	0

Operador AND

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Operador OR

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

(A . B)

(A + B)

## A IDEIA POR TRÁS DO OPERADOR “E” (AND)

- **TODAS** as condições devem ser **VERDADEIRAS!**
- Por exemplo:

Você pode obter uma habilitação de motorista se:

1. For aprovado no exame psicotécnico, **E**
2. For aprovado no exame de legislação, **E**
3. For aprovado no exame de legislação.

A	B	A E B
0	0	0
0	1	0
1	0	0
1	1	1





## A IDEIA POR TRÁS DO OPERADOR “OU” (OR)


- Pelo menos **UMA** condições deve ser **VERDADEIRA!**
- Por exemplo:

Você pode estacionar na vaga especial se:

1. For idoso(a), **OU**
2. For uma pessoa com deficiência, **OU**
3. For uma gestante.

A	B	A ou B
0	0	0
0	1	1
1	0	1
1	1	1

## 1    EXEMPLOS DE EXPRESSÕES LÓGICAS

2    • Assumindo  $X = 3$ , então:3  
4    1.  $(X \leq 5)$  OU  $(X = 9)$                       Resultado:5  
6    2.  $(X > 1)$  OU  $(X < 6)$                       Resultado:7  
8    3.  $(X \leq 10)$  OU  $(X = 6)$  OU  $(X \neq 3)$       Resultado:9  
10    4.  $(X \leq 3)$  OU  $(X = 3)$  E  $(X \neq 3)$           Resultado:11  
12    5.  $(X > 2)$  E  $(X = 3)$  OU  $(X \leq 4)$           Resultado:13  
14  
 Expressões híbridas

## A IDEIA POR TRÁS DO OPERADOR “NÃO” (NOT)

- O operador “NÃO” inverte a condição.

- Por exemplo:

Você tem direito a receber uma bolsa de estudos se você:

- NÃO

Possuir renda maior que R\$ 2.500,00



Golias  
R\$: 2.850,00



V



Davi  
R\$: 2.300,00



N

A	S ou A'
0	1
1	0



03 {

[ Estrutura Condicional ]

< SE ... ENTÃO ... SENÃO ... FIMSE.  
E o pensamento computacional >

}

## 3.1. Conceito

1 </1> É uma estrutura de controle que permite definir que um  
2 certo bloco de comandos somente será executado depen-  
3 dendo de uma condição.

4 </2> Portanto, o comando só será executado se a condição  
5 for verdadeira. Uma condição é uma comparação que  
6 possui dois valores possíveis: verdadeiro ou falso.

### Estrutura condicional SIMPLES

10 SE <condição> ENTAO

11 .... <comando1>

12 .... <comando2>

13 FIMSE

### REGRA

V: Executa o bloco de comandos.

F: Pula o bloco de comandos.

14 Observação importante: Reparem na indentação.

## Estrutura condicional COMPOSTA

```
SE <condição> ENTAO
```

```
.... <comando1>
```

```
.... <comando2>
```

```
SENAO
```

```
.... <comando3>
```

```
.... <comando4>
```

```
FIMSE
```

## REGRA

V: Executa somente o bloco do SE.

F: Executa somente o bloco do SENÃO.

Observação importante: Reparem na indentação.

## Operador

## Significado

E (AND)

Verdadeiro se todas as condições forem verdadeiras.

OU (OR)

Verdadeiro se pelo menos uma condição for verdadeira.

NÃO (NOT)

Verdadeiro se a condição for falsa.

### 3.2. 0 encadeamento [...].

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14

Estrutura condicional encadeada

SE <condição1> ENTAO

.... <comando1>

.... <comando2>

SENAO

SE <condição2> ENTAO

..... <comando3>

..... <comando4>

SENAO

..... <comando5>

..... <comando6>

FIMSE

FIMSE

→ Estrutura 1

→ Estrutura 2



### 3.3. Estrutura condicional em C

- E finalmente, o C chegou ... Antes, vou apresentar apenas a sintaxe da estrutura condicional simples e composta nessa linguagem de programação.

#### Estrutura condicional SIMPLES

```
if (condição)
{
    .... comando1;
    .... comando2;
}
```

Em C, torna-se obrigatória a utilização de chaves {} quando existe mais de um comando a executar, sendo que só serão executados se a condição for verdadeira.

#### Estrutura condicional COMPOSTA

```
if (condição)
{
    .... comando1;
}
else
{
    .... comando2;
}
```

1

2

3 </1> ASCENCIO, Ana Fernanda G. ARAÚJO, Grazielle. S. Estruturas de Dados:  
4 algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo:  
Pearson Prentice Hall, 2010. (Biblioteca virtual).

5

6 </2> PUGA, Sandra. RISSETTE, Gerson. Lógica de Programação e Estrutura de Dados.  
2. ed. São Paulo? Pearson Prentice Hall, 2009.

7

8

9

10

11

12

13

14

# A 'Obrigado' Is {

## Algoritmo & Linguagem de Programação 1

|  
}

