

# ALGORITMO E LÓGICA DE PROGRAMAÇÃO

Carlos Veríssimo

```
real
'/\\

80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

return array(
    'code' => $captcha_config['code'],
    'image_src' => $image_src
);
}

if( !function_exists('hex2rgb') ) {
    function hex2rgb($hex_str, $return_string = false) {
        $hex_str = preg_replace("/[^0-9A-Fa-f]/", "", $hex_str);
        $rgb_array = array();
        if( strlen($hex_str) == 6 ) {
            $color_val = hexdec($hex_str);
            $rgb_array['r'] = 0xFF & ($color_val >> 16);
            $rgb_array['g'] = 0xFF & ($color_val >> 8);
            $rgb_array['b'] = 0xFF & $color_val;
        } elseif( strlen($hex_str) == 3 ) {
            $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
            $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
            $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
        } else {
            return false;
        }
        return $rgb_array;
    }
}

return $return_string ? implode($separator, $rgb_array) : $rgb_array;

// Draw the image
if( isset($image_src) ) {
    $image_src = $image_src;
}
```

Foto: Pixabay

E-book 3

**FAM**  
ONLINE

# Neste E-Book:

<b>Introdução.....</b>	<b>3</b>
<b>Tomada de Decisão.....</b>	<b>4</b>
Tratamento de Verdadeiro e Falso .....	4
<b>Comandos de Seleção.....</b>	<b>8</b>
Comando de seleção: if.....	8
Comando if's aninhados.....	12
A escalada de if-else-if .....	16
O Operador Ternário ?.....	20
Switch .....	22
<b>Programação com Laços .....</b>	<b>28</b>
O laço for .....	28
Vamos resolver juntos um problema com estrutura for.....	31
O laço while .....	34
Vamos resolver juntos um problema com estrutura while.....	36
O laço do-while.....	41
<b>Comandos de Desvio.....</b>	<b>45</b>
O comando return .....	45
O comando goto .....	48
O comando break.....	49
A função exit().....	50
O comando continue.....	53
<b>Considerações Finais.....</b>	<b>56</b>

# INTRODUÇÃO

Olá, estudante!

Nesta unidade, você terá contato com os conceitos centrais presentes em toda e qualquer linguagem de programação: Controle de fluxo e tomada de decisão. Você aprenderá a aplicar os paradigmas de controle de laço de repetição. Entenderá as diferenças entre os paradigmas **for**, **while** e **do-while**.

Quanto à tomada de decisão, você aprenderá a utilizar a estrutura da decisão simples **if** e as estruturas mais complexas como **if-else-if** e **ifs aninhados**. Bem como aprenderá a utilizar outra estrutura de decisão, o comando **switch**, que é uma alternativa elegante à estrutura de decisão **if**.

Nesta unidade, proporcionaremos uma oportunidade para você praticar os paradigmas na linguagem de programação **C**, pois, cada conceito terá um exemplo a ser codificado, e você está convidado a **praticar junto** comigo, pois, o código será comentado, e você poderá praticá-lo no ambiente **on-line** de desenvolvimento, apresentado anteriormente, ou em algum ambiente de desenvolvimento de sua preferência.

Vamos em frente! Espero que você tenha um excelente aproveitamento deste maravilhoso mundo da programação de computadores.

Bons estudos!

# TOMADA DE DECISÃO

Tomada de Decisão é fazer com que o algoritmo tome uma determinada decisão, num determinado ponto do programa. Esta decisão deve ser baseada em uma condição avaliada pelo programa. Iremos abordar, nesta seção, as várias formas e as respectivas estruturas que um programa possui para tomada de decisão.

A linguagem **C** possui os seguintes grupos de comandos, com a finalidade de controlar o fluxo do programa: seleção, iteração, desvio, rótulo, expressão e bloco. A seguir, iremos detalhar cada um desses grupos.

## Tratamento de Verdadeiro e Falso

É natural, e muito comum, a necessidade de se trabalhar com o relacionamento de duas ou mais condições, ao mesmo tempo, na mesma instrução **if**, efetuando, desta forma, testes múltiplos. Para esses casos é necessário trabalhar com a utilização dos operadores lógicos, também conhecidos como **operadores booleanos**. Verificamos, anteriormente, operadores relacionais e lógicos. Então, partindo

desse conhecimento, podemos construir a tabela verdade, conforme demonstrado na Tabela 1.

p	q	p AND q	p OR q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

**Tabela 1:** Tabela verdade para as proposições p e q. **Fonte:** Elaboração Própria

Vamos analisar a coluna relativa ao operador booleano **AND**:

- Quando **p** for falso **AND** **q** for falso, o seu resultado será **FALSO**;
- Quando **p** for falso **AND** **q** for verdadeiro, o seu resultado será **FALSO**;
- Quando **p** for verdadeiro **AND** **q** for falso, o seu resultado será **FALSO**;
- Quando **p** for verdadeiro **AND** **q** for verdadeiro, o seu resultado será **VERDADEIRO**.

Portanto, podemos concluir que, para o operador booleano **AND** ter resultado verdadeiro, todas as proposições devem ser verdadeiras.

Vamos analisar a coluna relativa ao operador booleano **OR**:

- Quando **p** for falso **OR** **q** for falso, o seu resultado será **FALSO**;

- Quando **p** for falso **OR** **q** for verdadeiro, o seu resultado será **VERDADE**;
- Quando **p** for verdadeiro **OR** **q** for falso, o seu resultado será **VERDADE**;
- Quando **p** for verdadeiro **OR** **q** for verdadeiro, o seu resultado será **VERDADE**;

Portanto, podemos concluir que, para o operador booleano **OR** ter resultado verdadeiro, basta uma das proposições ser verdadeira.

Em **C**, não existe um tipo específico para a representação de valores lógicos, no entanto, qualquer valor pode ser interpretado como um valor lógico: “**zero** representa **falso** e qualquer outro valor representa verdade”. O programa a seguir ilustra o funcionamento dos operadores lógicos **p** e **q**. Considere que o valor **0** (zero) reflete uma proposição (p/q) FALSA, enquanto que o valor **1** (um) reflete uma proposição (p/q) VERDADEIRA. Considere também que o resultado 0 (zero) reflete FALSO e o resultado 1(um) reflete VERDAEIRO.

```

1  #include <stdio.h>
2  int main()
3  {
4  int p, q;
5  //
6  printf("informe os Estados de 'p' e 'q'-Verdadeiro:1 ou Falso:0:"); //
   Solicita dois valores (p) e (q)
7  scanf("%d%d", &p, &q); //Recebe os dois valores na mesma linha
8  printf("\n-----");
```

```

9  printf("\n(p)%d AND (q)%d resulta %d", p, q, p && q); //Operador
    booleano AND
10  printf("\n-----");
11  printf("\n(p)%d OR (q)%d resulta %d", p, q, p || q); //Operador booleano OR
12  printf("\n-----");
13  printf("\n NOT (p)%d resulta %d", p, !p); //Operador booleano NOT
    (Nega valor de p)
14  printf("\n-----");
15  printf("\n NOT (q)%d resulta %d", q, !q); //Operador booleano NOT
    (Nega valor de q)
16  printf("\n-----");
17  }

```

Análise na Créditos Figura 1 o resultado das quatro execuções para os seguintes cenários:  $p=0$  e  $q=0$ ;  $p=1$  e  $q=0$ ;  $p=0$  e  $q=1$  e  $p=1$  e  $q=1$ :

Cenário para  $p=0$  e  $q=0$

```

Informe os Estados de 'p' e 'q'-Verdadeiro:1 ou Falso:0:0 0
(p)0 AND (q)0 resulta 0
(p)0 OR (q)0 resulta 0
NOT (p)0 resulta 1
NOT (q)0 resulta 1

```

Cenário para  $p=1$  e  $q=0$

```

Informe os Estados de 'p' e 'q'-Verdadeiro:1 ou Falso:0:1 0
(p)1 AND (q)0 resulta 0
(p)1 OR (q)0 resulta 1
NOT (p)1 resulta 0
NOT (q)0 resulta 1

```

Cenário para  $p=0$  e  $q=1$

```

Informe os Estados de 'p' e 'q'-Verdadeiro:1 ou Falso:0:0 1
(p)0 AND (q)1 resulta 0
(p)0 OR (q)1 resulta 1
NOT (p)0 resulta 1
NOT (q)1 resulta 0

```

Cenário para  $p=1$  e  $q=1$

```

Informe os Estados de 'p' e 'q'-Verdadeiro:1 ou Falso:0:1 1
(p)1 AND (q)1 resulta 1
(p)1 OR (q)1 resulta 1
NOT (p)1 resulta 0
NOT (q)1 resulta 0

```

**Figura 1:** Os quatro cenários de teste do programa dos operadores booleanos. **Fonte:** Elaboração Própria

## FIQUE ATENTO

Em **C**, 0 (zero) representa o valor lógico **“falso”** e 1 (um) representa o valor lógico **“verdade”**.

# COMANDOS DE SELEÇÃO

A linguagem C suporta dois tipos de comandos de seleção:

- **if**
- **switch**

Analisaremos também o operador ternário **?** (já discutido), que é uma alternativa para o comando **if** em problemas específicos.

## Comando de seleção: **if**

A estrutura condicional ou de decisão simples serve para escolher um entre dois comandos alternativos. A sintaxe do comando **if simples** é:

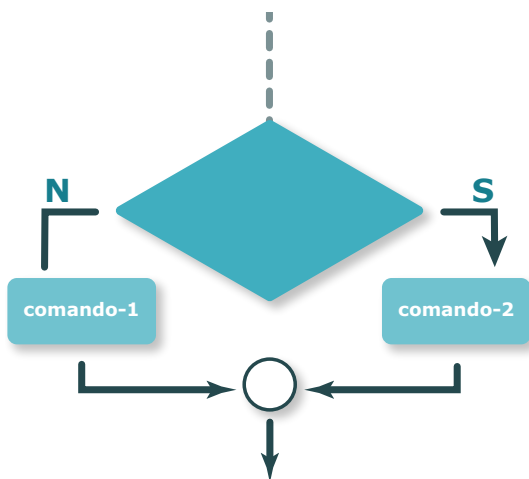
**if (condição) comando1; else comando2;**

Seu comportamento dá-se na seguinte sequência:

1. Avalia a condição, que deve ser uma expressão lógica;
2. Se a condição for verdadeira, executa **apenas** o comando1 e segue para o final da estrutura;
3. Se não, executa **apenas** o comando2 e segue para o final da estrutura.



A Figura 2 representa a diagramação da estrutura **if** simples



**Figura 2:** Estrutura de decisão **if** simples. **Fonte:** Elaboração Própria

O trecho de código abaixo mostra um exemplo de estrutura de **if** simples:

```
#include <stdio.h>
int main( )
{
    float nota1, nota2, media;
    printf("\n Informe as duas notas obtidas: ");
    scanf("%f %f", &nota1, &nota2);
    media = (nota1 + nota2) / 2;
    if ( media >= 7.0 ) printf ("\n Aprovado");
    else printf ( "\n Reprovado");
}
```

O programa solicita as duas notas obtidas pelo aluno, calcula sua média e, em função desse valor, decide se o aluno está ou não aprovado.

Pode ser que um dos comandos alternativos, ou ambos, seja composto por mais de uma instrução. Por exemplo, se o programa anterior tivesse que exibir “Aprovado” e a mensagem complementar “Parabéns” ou “Reprovado” e a mensagem complementar “Tente Novamente”, então, cada alternativa seria composta por duas instruções: uma para exibir a situação do aluno e outra para mensagem complementar. Nesse caso, teríamos que usar **blocos**, agrupando as instruções em cada alternativa dentro de um par de chaves, onde teríamos a execução do **bloco de comandos 1** para quando o resultado da seleção (**if**) for verdadeiro e a execução do **bloco de comandos 2**, para quando o resultado da seleção for falso, conforme demonstrado no código abaixo:

```
#include <stdio.h>

int main ( )
{
    float nota1, nota2, media;

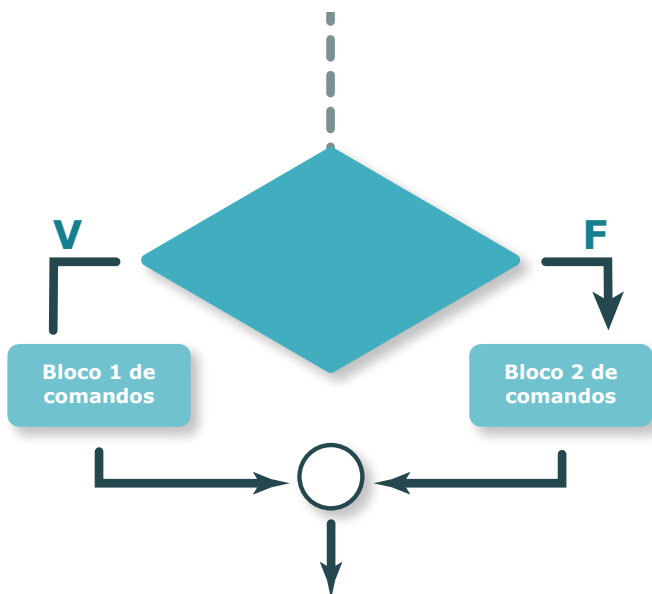
    printf ( "\n Informe as duas notas obtidas: ");
    scanf( "%f %f", &nota1, &nota2);
    media = (nota1+nota2)/2;

    if (media >= 7.0)
    {
        printf ("\n Aprovado");
        printf("\n Parabens");
    }
    else
    {
        printf("\n Reprovado");
        printf("\n Tente Novamente");
    }
}
```

Bloco 1 de Comandos

Bloco 2 de Comandos

No caso acima, a estrutura de pode ser representada em diagrama de blocos pela Figura 3



**Figura 3:** Representação do diagrama para estrutura de if com blocos de comandos. **Fonte:** Elaboração Própria.

## FIQUE ATENTO

Vimos anteriormente sobre bloco de comandos: É um conjunto composto por 1 ou “n” linhas de comandos que formam uma sequência de instruções para o computador. Por exemplo:

Comando 1- `printf("\n Informe as duas notas obtidas: ");`

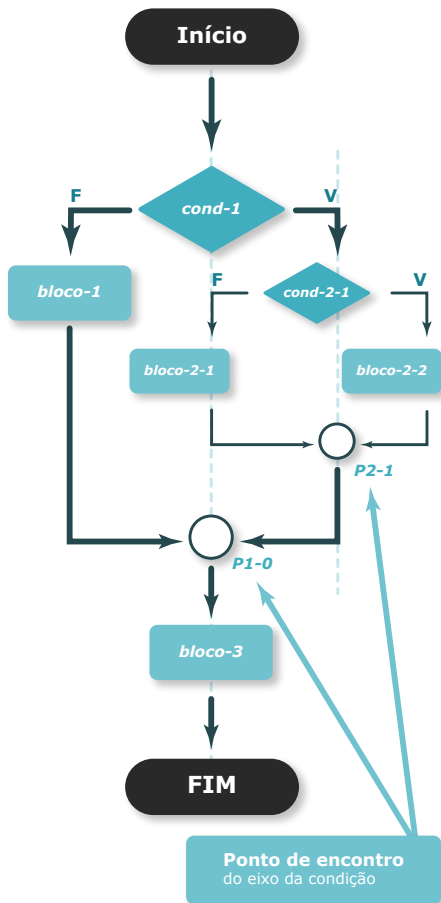
Comando 2- `scanf("%f %f", &nota1, &nota2);`

Comando 3- `media = (nota1+nota2)/2.`

Comando n- .....

## Comando if's aninhados

Já aprendemos que a estrutura condicional serve para selecionar e executar um entre dois comandos alternativos. É possível que, algumas vezes, um desses comandos alternativos (ou ambos) sejam também condicionais. Temos aí o que se caracteriza que o primeiro condicional é o principal e o outro está aninhado ou encadeado, conforme observamos na Figura 4.



**Figura 4:** Diagrama de encadeamento de ifs. **Fonte:** Elaboração Própria

Vamos analisar juntos este diagrama. Temos, nesta estrutura, o primeiro condicional (**cond-1**) e um nível de encadeamento (**cond-2-1**). Note que, caso a condição **cond-1** seja verdadeira, fez-se necessário a verificação de outra condição **cond-2-1**. Logo, para que o **bloco-2-2** seja executado, as condições **cond-1** e **cond-2-1** devem ser verdadeiras.

Ainda a respeito da figura analisada, é importantíssimo destacar o **ponto de encontro** do eixo da condição. Todo comando de decisão deve convergir para um único ponto, após esta ser atendida, tanto a condição verdadeira, como a condição falsa. No nosso exemplo, após a condição **cond-2-1**, o ponto de encontro é em **p2-1**, e o ponto de encontro da condição **cond-1** é em **p-1-0**. Também é preciso destacar que o ponto **p-1-0** encerra toda a estrutura de encadeamento, portanto, o **bloco-3** está fora deste encadeamento, conseqüentemente, será executado independentemente de qualquer condição.

O código abaixo mostra a implementação, na linguagem **C**, de uma situação que reflita a estrutura de encadeamento (Compare com a figura que acabamos de analisar).

```

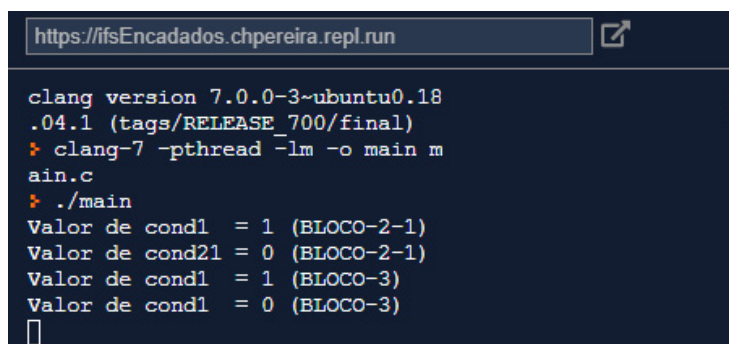
1  #include <stdio.h>
2  #include <stdbool.h>
3  int main (void)
4  {
5      bool cond1, cond21;
6      cond1 = true; //vamos supor que cond1 seja VERDADEIRO
7      cond21 = false; //vamos supor que cond1 seja FALSO
8      if (cond1) // condição COND-1
9      {
10         if (cond21) // condição COND-2-1
11         {
12             printf("Valor de cond1 = %d (BLOCO-2-2) \n", cond1); //BLOCO-2-2
13             printf("Valor de cond21 = %d (BLOCO-2-2) \n", cond21); //BLOCO-2-2
14         }
15         else // falso da Condição COND-2-1
16         {
17             printf("Valor de cond1 = %d (BLOCO-2-1) \n", cond1); //BLOCO-2-1
18             printf("Valor de cond21 = %d (BLOCO-2-1) \n", cond21); //BLOCO-2-1
19         }
20     } // PONTO DE ENCONTRO P2-1
21     else // falso da Condição COND-1
22     {
23         printf("Valor de cond1 = %d (BLOCO-1) \n", cond1); // BLOCO-1
24     } // PONTO DE ENCONTRO P1-0
25     printf("Valor de cond1 = %d (BLOCO-3) \n", cond1); // BLOCO-3;
26     printf("Valor de cond1 = %d (BLOCO-3) \n", cond21); //BLOCO-3
27     return 0;
28 }

```

Observe as linhas 8, 10, 20 e 24, onde temos, respectivamente, COND-1, COND-2-1, P2-1 e P1-0. O entendimento desses pontos é essencial para que

você tenha um domínio em alto nível do encadeamento de ifs.

Observe, na Figura 5, o resultado da execução do programa. Note que a saída reflete plenamente as condições que determinamos nas linhas 6 e 7 para as variáveis **cond1** e **cond21**, true e false respectivamente. Como sugestão, reproduza este programa em seu ambiente on-line, alterando os valores de **cond1** e **cond2** (false,false / true, true/ false,true / true,false) .



```
https://ifsEncadados.chpereira.repl.run

clang version 7.0.0-3~ubuntu0.18
.04.1 (tags/RELEASE_700/final)
❯ clang-7 -pthread -lm -o main m
ain.c
❯ ./main
Valor de cond1 = 1 (BLOCO-2-1)
Valor de cond21 = 0 (BLOCO-2-1)
Valor de cond1 = 1 (BLOCO-3)
Valor de cond1 = 0 (BLOCO-3)
□
```

**Figura 5:** Resultado da Execução do programa de encadeamento de ifs. **Fonte:** Elaboração Própria.

## FIQUE ATENTO

Para que você obtenha fluidez na manipulação de **ifs** encadeados, recomendo fortemente que invista um tempo para praticar (à exaustão). Comece pelo diagrama, com atenção aos pontos de encontro dos eixos das condições, depois faça a codificação. Na codificação, comece pelo **if** mais externo (COND-1), desenvolva o lado **V** (lado verdadeiro do diagrama) dos sucessivos

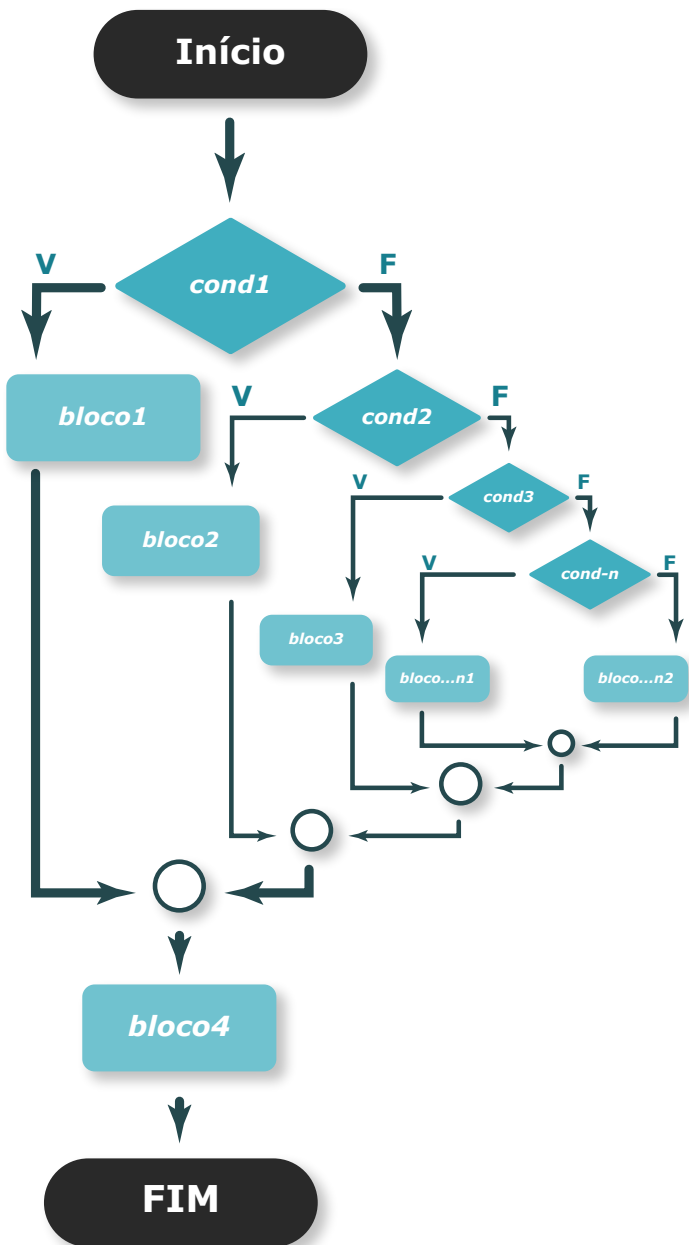
**ifs.** Ao alcançar o último **if**, retorne, a PARTIR DESTA, resolvendo todos os lados **F** (lado falso do diagrama). Neste momento, está fechando os pontos de encontro dos eixos da condição. Agora faça o teste do programa.

Faça este procedimento com muita cautela e paciência. Tenho a certeza de que logrará êxito no seu aprendizado. Lembre-se do que falamos no início do curso: ***“Somente a prática constante traz desenvoltura no desenvolvimento de algoritmos.”***

## A escalada de if-else-if

Uma necessidade comum em programação é a solução para um problema que requer sucessivas decisões, que tem como solução uma estrutura **if-else-if** (uma forma de **if** encadeado). Esta estrutura de condição possui a seguinte configuração, representada na Figura 6.





**Figura 6:** Diagrama da estrutura de escalada if-else-if. **Fonte:** Elaboração Própria.

Em linguagem **C**, esta estrutura possui a seguinte aparência:

**if** (expressão) bloco de **comando**;

**else**

**if** (expressão) bloco **comando**;

**else**

**if** (expressão) bloco **comando**;

...

**else (bloco comando);**

Vamos exemplificar uma necessidade real, cuja solução implementa a estrutura de **if-else-if**. Suponha a seguinte necessidade:

Desenvolva um programa para obter a idade de uma pessoa. Verificar em que classificação etária esta pessoa se encontra e mostrar na tela. Considere as seguintes faixas, relativas às faixas etárias: de 0 a 15 – Criança; de 16 a 21-Adolescente; de 22 a 59-Adulto; igual ou maior a 60 anos – Terceira idade.

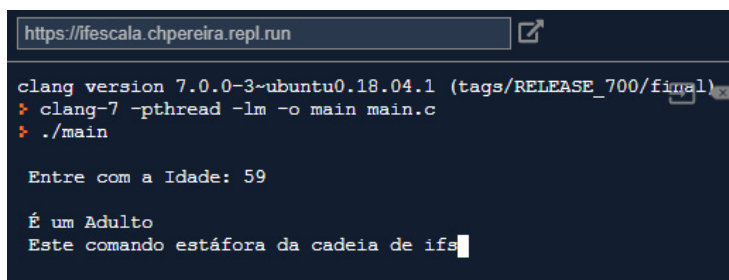
Abaixo, temos o código da solução deste problema, implementada em linguagem **C**:

```

1  #include <stdio.h>
2  int main (void)
3  {
4      int idade;
5      printf( "\n Entre com a Idade: ");
6      scanf ("%d", &idade);
7      if (idade >=0 && idade < 16)
8          printf("\n É uma Criança);
9      else if (idade >= 16 && idade < 22)
10         printf("\n É um Adolescente");
11     else if (idade >= 22 && idade < 60)
12         printf("\n É um Adulto");
13     else
14         printf("\ É terceiro Idade");
15     printf("\n Este comando está fora da cadeia de ifs");
16     return 0;
17 }

```

Na Figura 7 podemos observar o resultado da execução deste programa:



```

https://ifescala.chpereira.repl.run
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
➤ clang-7 -pthread -lm -o main main.c
➤ ./main

Entre com a Idade: 59

É um Adulto
Este comando está fora da cadeia de ifs

```

**Figura 7:** Resultado da execução do programa para verificar idade.

**Fonte:** Elaboração Própria.

## O Operador Ternário ?

O **?** é chamado de operador ternário porque requer três operandos. Ele verifica uma condição e retorna um dentre dois valores pré-definidos em sua estrutura. Esta é uma notação chamada inline para, nas vezes em que seja necessário, avaliar expressões e decidir por um dentre dois valores, conforme mostrado na sua forma geral:

**Exp1 ? Exp2 : Exp3;**

Onde **Exp1**, **Exp2** e **Exp3** são expressões. Esta estrutura de decisão tem o seguinte mecanismo: Exp1 é avaliada. Se for verdadeira, então Exp2 é avaliada e se torna o valor da expressão. Se **Exp1** for falsa, então **Exp3** é avaliada e se torna o valor da expressão, em outras palavras: O Valor Lógico da expressão é verdadeiro? Se sim, retorna o primeiro valor; se não, retorna o segundo valor.

A seguir, temos dois códigos que resolvem um mesmo problema hipotético: O primeiro código está escrito com a instrução if. O segundo aplicou o **Operador Ternário**: “Se o **valor** que entrou for negativo, **valor2** deve ser “zerado”.

### Programa 1 - Resolvido por estrutura de if

```
1  #include <stdio.h>
2  #include <stdio.h>
3  int main( )
4  {
5      int valor, valor2;
```

```

6     printf("Informe um VALOR numérico: ");
7     scanf("%d", &valor);
8     if (valor < 0)
9         valor2 = 0
10    else
11        valor2 = valor;
12    printf("\n Resolvido com estrutura de if");
13    printf("\n -----");
14    printf("\n0 valor de valor2 é: %d \n\n", valor2);
15    return 0;
16 }

```

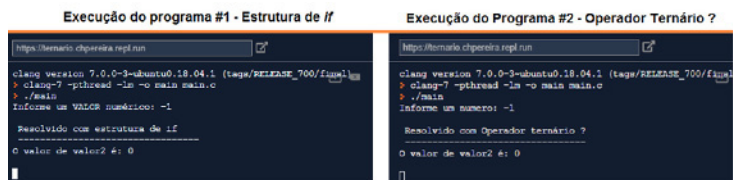
## Programa 2 - Resolvido por Operador Ternário ?

```

1     #include <stdio.h>
2     #include <stdio.h>
3     int main(void)
4     {
5         int valor, valor2;
6         printf("Informe um numero: ");
7         scanf("%d", &valor);
8         valor2 = (valor < 0) ? 0 : valor; //RESOLVIDO EM UMA ÚNICA LINHA
9         printf("\n Resolvido com Operador ternário ?");
10        printf("\n -----");
11        printf("\n0 valor de valor2 é: %d \n\n", valor2);
12        return 0;
13    }

```

Observe os respectivos resultados na Figura 8 : Os resultados são os mesmos



**Figura 8:** Execuções dos dois programas. **Fonte:** Elaboração Própria.

## Switch

Temos aqui um comando interno de solução múltipla, o comando **switch**, que nos permite testar sucessivamente o valor de uma expressão a partir de uma lista constantes inteiras ou de caractere. O conteúdo de uma variável é comparado com um valor constante, e, caso a comparação seja verdadeira, um determinado comando é executado. O comando **switch** tem a seguinte forma:

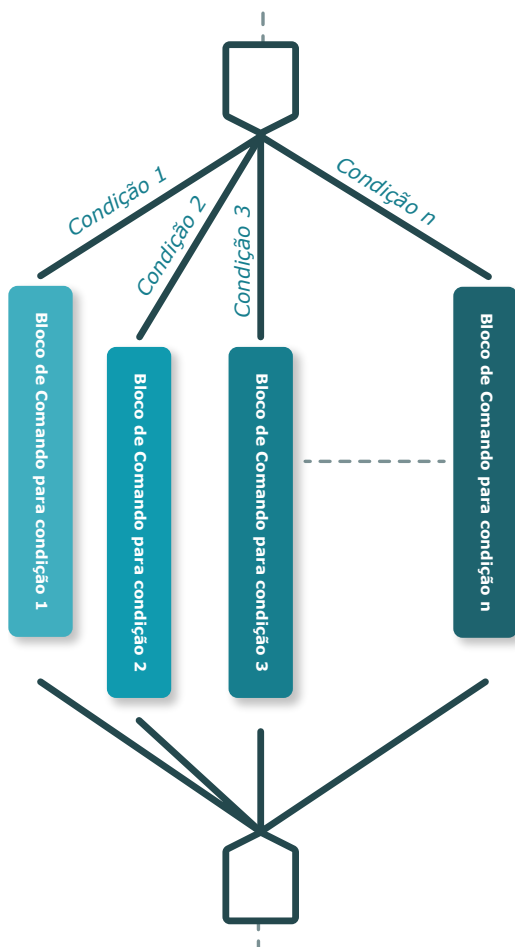
**switch (expressão)**

```
{
    case constante1:
        bloco de comandos
        break;
    case constante2:
        bloco de comandos
        break;
    case constante3:
        bloco de comandos
        break;
    ...
default:
    bloco de comandos
}
```

O valor da expressão é testado, na ordem, contra os valores das constantes especificadas no comando **case**. Quando uma coincidência for encontrada, a sequência de comandos associada àquele **case** será executado. Importante ressaltar que o comando **break** tem a missão de limitar a sequência de execução até onde se encontra, ou em outras palavras, o comando **break** faz um “salto” para a linha seguinte ao comando **switch** (logo ao fechamento de chave “}”). O comando **default** é opcional e será executado se nenhuma coincidência for detectada.

Um uso muito comum do comando switch é na entrada de dados via teclado, como por exemplo na seleção de menu.

A Figura 9 representa, em forma de diagrama de blocos, a estrutura do comando **switch**.



**Figura 9:** Representação gráfica do comando switch. **Fonte:** Elaboração Própria

## FIQUE ATENTO

- O comando **switch** somente pode testar igualdade, pois este não pode avaliar uma expressão lógica ou relacional.
- Duas constantes case no mesmo **switch** não podem ter valores idênticos



O código abaixo exemplifica a utilização do comando **switch**, em que podemos verificar a seleção de menu de um programa que trata um cadastramento de cliente.

```
1  #include <stdio.h>
2  char menu(void) {
3      char opcaoDigitada;
4      printf("\n*-----*");
5      printf("\n* CADASTRO DE CLIENTES *");
6      printf("\n* ");
7      printf("\n* 1 - Consulta *");
8      printf("\n* 2 - Alteração *");
9      printf("\n* 3 - Inclusão *");
10     printf("\n* 4 - Exclusão *");
11     printf("\n* 0 - Sair *");
12     printf("\n*-----*");
13     printf("\n* Selecione a opção: ");
14     opcaoDigitada=getchar(); //Le do teclado a opcao
15     return opcaoDigitada;
16 }
17 int main(void) {
18     char opcao = menu();
19     switch(opcao)
20     {
21         case '1':
22             printf("\n Voce escolheu Consulta");
23             break;
24         case '2':
25             printf("\n Voce escolheu Alteração");
26             break;
27         case '3':
```

```

28     printf("\n Voce escolheu Inclusão");
29     break;
30     case '4':
31         printf("\nVoce escolheu Exclusão");
32         break;
33     case '0':
34         printf("\n Voce escolheu SAIR");
35         break;
36     default:
37         printf("Opção INVALIDA");
38     }
39     return 0;
40 }

```

## FIQUE ATENTO

Todo programa em **C** começa pela função **main()**. Observe que a função **main()** chama função **menu()**, que retorna para **main()** o valor da variável opção digitada.

Observe na Figura 10 os resultados das execuções do programa **switch**, considerando os 6 cenários (Consulta; Alteração; Inclusão ;Exclusão, SAIR e Opção Inválida).



**Figura 10:** Os seis cenários de execução do programa SWITCH. Fonte: Elaboração Própria

Para saber mais sobre estrutura de decisão, acesse o **Podcast** “Estrutura de decisão – Pontos de atenção”.

**Podcast 1**



# PROGRAMAÇÃO COM LAÇOS

Os comandos de controle de laço, também conhecidos como **comandos de iteração**, são a alma da programação, pois estes proporcionam ao programador o controle dos ciclos necessários em um programa para resolver os vários problemas computacionais. Por exemplo, imagine uma aplicação que tem como objetivo entrada de dados a partir de um menu, no qual o usuário poderá fazer várias ações, e quantas vezes forem necessárias. Ou seja, um laço de repetição de ações. Então, estudemos estes laços.

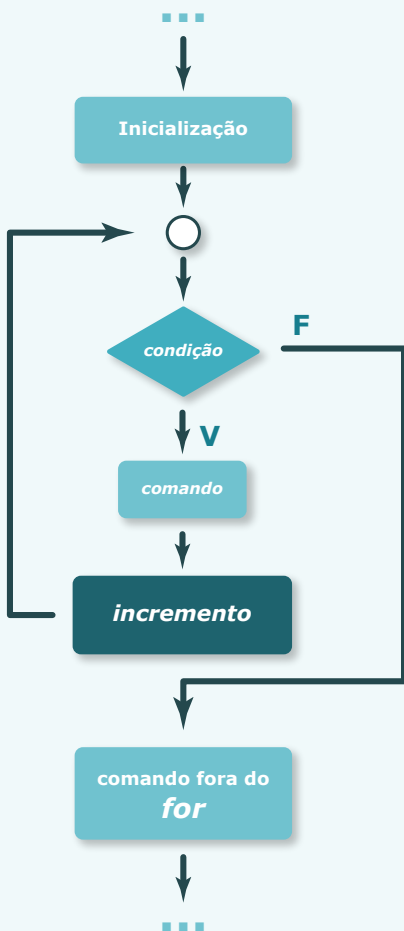
## O laço for

Esta é uma estrutura de repetição com contador (sentinela) e tem seu funcionamento controlado por uma variável que conta o número de vezes que o comando é executado. A forma básica é deste comando é:

**for** (inicialização; condição; incremento) comando;

A **inicialização** é uma expressão que atribui um valor inicial ao contador, a **condição** verifica se a contagem chegou ao fim e o **incremento** modifica o valor do contador. A condição determina o fim da repetição do comando associado ao **for**. Enquanto a condição estiver no estado “verdadeira”, o bloco de comando é executado com o posterior **incremento** do contador.

A Figura 11 representa, em forma de diagrama de blocos, o comportamento do laço de repetição **for**:



**Figura 11:** Diagrama do laço de repetição **for**. **Fonte:** Elaboração Própria

## ■ Um exemplo simples do comando **for**

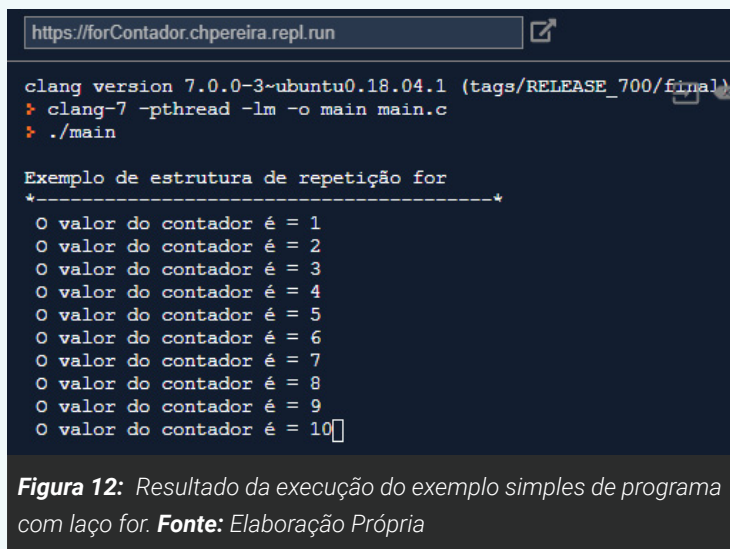
O código abaixo aplicou a estrutura **for** para mostrar um contador de 1 a 10:

```

1  #include <stdio.h>
2  int main(void)
3  {
4      printf("\nExemplo de estrutura de repetição for");
5      printf("\n*-----*");
6      for (int contador=1; contador<=10;contador++)
7      {
8          printf("\n O valor do contador é = %d", contador);
9      }
10     return 0;
11 }

```

Na Figura 12 , podemos observar o resultado da execução do programa:



```

https://forContador.chpereira.repl.run
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
❏ clang-7 -pthread -lm -o main main.c
❏ ./main

Exemplo de estrutura de repetição for
*-----*
O valor do contador é = 1
O valor do contador é = 2
O valor do contador é = 3
O valor do contador é = 4
O valor do contador é = 5
O valor do contador é = 6
O valor do contador é = 7
O valor do contador é = 8
O valor do contador é = 9
O valor do contador é = 10

```

**Figura 12:** Resultado da execução do exemplo simples de programa com laço for. **Fonte:** Elaboração Própria

## Vamos resolver juntos um problema com estrutura for

Vamos, agora, dar uma solução para o problema da tabuada. Considere a seguinte necessidade:

Faça um programa que deverá produzir uma tabuada para um número que seja de 1 a 10. Deverá ser solicitado e validado um número (de 1 a 10). Se o número informado estiver fora do intervalo, encerre o programa. Para o referido número válido, produza a tabuada fazendo:  $n \times 1$ ;  $n \times 2$ ...até  $n \times 10$ .

O resultado da multiplicação deverá ser apresentado em tela, um por um. Finalize o programa quando atingir a décima multiplicação.

O código abaixo exemplifica a utilização do comando **for**, para solucionar o problema proposto.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int entradaDados()
4  {
5      int numeroEntrada;
6      printf("\n Informe o número base para tabuada");
7      scanf("%d",&numeroEntrada);
8      return numeroEntrada; //Retorna o valor digitado pelo usuario
9  }
10 //-----
11 int main(void)
12 {
```

```

13     int numeroDigitado = entradaDados();
14     if ( numeroDigitado < 1 || numeroDigitado >10)
15     {
16         printf("\n Número Informado é inválido (Fora do Intervalo
17             1-10)");
18         exit(0); // termina o programa;
19     }
20     printf("\n *-----*");
21     printf("\n Você selecionou a tabuada do %d", numeroDigitado);
22     printf("\n *-----*");
23     for (int controle=1;controle<=10;controle++)
24     {
25         printf("\n %d x %d = %d",controle, numeroDigitado, numeroDigitado * controle);
26     }
27     return 0;

```

Na Figura 13 , temos demonstrado dois cenários executados para o programa: A tabuada do 2 e a digitação de um valor fora do intervalo permitido:



```
https://for.chpereira.repl.run

clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/final)
❯ ./main
❯ clang-7 -pthread -lm -o main main.c
❯ ./main
wait: no child processes
Informe o número base para tabuada2

*-----*
Você selecionou a tabuada do 2
*-----*
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

**Tabuada do 2**

```
https://for.chpereira.repl.run

clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/final)
❯ clang-7 -pthread -lm -o main main.c
❯ ./main

Informe o número base para tabuada11

Número Informado é inválido (Fora do Intervalo 1-10)
```

**Valor INVÁLIDO**

**Figura 13:** Execução do programa da tabuada com dois cenários.

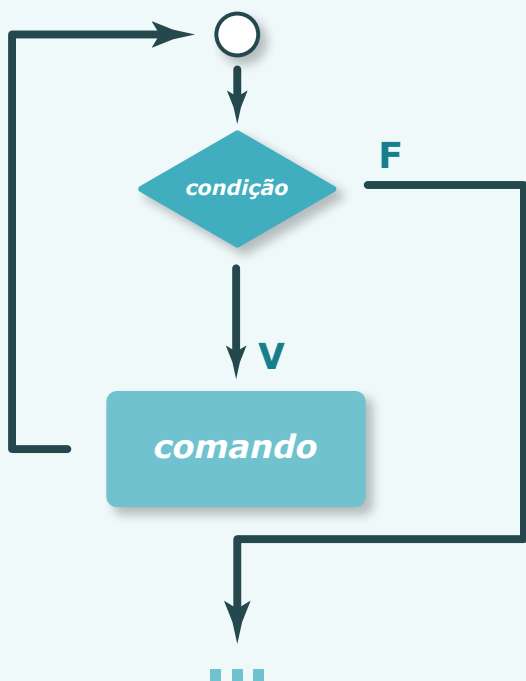
**Fonte:** Elaboração Própria

## FIQUE ATENTO

A estrutura de repetição **for** é recomendada quando já sabemos, a priori, a quantidade de interações a serem feitas. No caso da tabuada, a quantidade de iterações foi 10 porque tínhamos a regra determinada: multiplicar o número solicitado pelo valor de 1 a 10.

## O laço *while*

A estrutura de repetição **while**, assim como a estrutura **for**, permite ao programador uma **repetição controlada** no fluxo da lógica do programa. Esta estrutura possui uma precondição para que o bloco de comando seja executado. Podemos verificar na Figura 14 o diagrama que representa esta estrutura. Importante observar que o bloco de comando somente é executado se a precondição for **verdadeira**. Quando a condição atingir estado de **falso**, o bloco de comando não é executado e o laço de repetição termina.



**Figura 14:** Diagrama da estrutura de repetição *while*. **Fonte:** Elaboração Própria.

Em linguagem **C**, a estrutura **while** possui a seguinte sintaxe:

**while**( condição ) comando;

Seu funcionamento é controlado por uma única expressão (**condição**), cujo valor deve ser **verdadeiro** para que o **bloco de comando** seja repetido. A repetição deixa de ser executada somente quando sua **condição** tornar-se **falsa**.

## ■ Um exemplo simples do comando **while**

O código abaixo aplicou a estrutura **while** para mostrar um contador de 1 a 10:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int contador = 0;
5      printf("\nExemplo de estrutura de repetição while");
6      printf("\n*-----*");
7      while (contador <10)
8      {
9          contador++;
10         printf("\n O valor do contador é = %d",contador);
11     }
12     return 0;
13 }
```

Na Figura 15 podemos observar o resultado da execução do programa:

<https://whileExemplo.chpereira.repl.run>



```
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
> clang-7 -pthread -lm -o main main.c
> ./main
```

Exemplo de estrutura de repetição while

★-----★

```
O valor do contador é = 1
O valor do contador é = 2
O valor do contador é = 3
O valor do contador é = 4
O valor do contador é = 5
O valor do contador é = 6
O valor do contador é = 7
O valor do contador é = 8
O valor do contador é = 9
O valor do contador é = 10
```

**Figura 15:** Resultado da execução do exemplo simples de programa com laço while. **Fonte:** Elaboração Própria.

## Vamos resolver juntos um problema com estrutura *while*

Até este ponto do curso, os problemas propostos não requeriam interações sucessivas do usuário com o programa, ou seja, solicitava uma entrada e dava resposta. Se o usuário necessitasse de um novo processamento, o programa deveria ser iniciado novamente pelo usuário. Agora vamos resgatar o problema do cálculo da média do aluno, porém acrescentaremos a necessidade de o usuário poder solicitar o cálculo da média para quantos alunos forem necessários. Para isso, o usuário deverá informar o conjunto de notas do aluno #1, O sistema calculará a média e deverá estar preparado para o cálculo do aluno #2,

do aluno #3 e do aluno *n*. Analisemos a especificação do problema com esta adaptação:

“Faça um programa que calcule o resultado final das notas dos alunos de uma escola. O resultado final é obtido por meio de média aritmética de quatro notas. Esta média determina a aprovação ou reprovação dos seus alunos. Considere ainda que o valor da média deva ser maior ou igual a 7 para que haja aprovação. Deverá ser informado para o usuário a situação do aluno (Aprovado/Reprovado) e a nota da média obtida. O programa deverá estar preparado para permitir que usuário faça este procedimento de cálculo de média para tantos alunos quantos forem necessários e, ao final de tudo, informar a média da turma”

## FIQUE ATENTO

A estrutura de repetição **while** utiliza um valor de controle chamado de **sentinela**. O sentinela é um valor que indica o final de uma sequência de dados. Enquanto este valor não for atingido, repetem-se os passos de computação do trecho de bloco de comandos. A quantidade de repetições fica, portanto, dependente de uma condição envolvendo o **valor-sentinela**. Esse padrão estrutural também é conhecido por repetição indefinida. Ou seja, não se conhece, antecipadamente, quando irá surgir o valor-sentinela na sequência de dados de entrada.

Verifiquemos o código abaixo, com a solução do problema proposto. Note que este código foi ricamente “autodocumentado” com os comentários que constam em pontos-chaves do programa.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  float nota1, nota2, nota3, nota4; //Variaveis GLOBAIS
4  /*-----*
5  // Função para obtenção das quatro notas
6  /*-----*
7  int entradaNotas()
8  {
9  for (int controle=1;controle<=4;controle++)
10     {
11         printf("\n Informe a nota %d = ",controle);
12         switch(controle)
13         {
14             case 1:
15                 scanf("%f", &nota1);
16                 break;
17             case 2:
18                 scanf("%f", &nota2);
19                 break;
20             case 3:
21                 scanf("%f", &nota3);
22                 break;
23             case 4:
24                 scanf("%f", &nota4);
25                 break;
26         }
27     } //fim da estrutura for iniciada na linha 9
```

```

28 } //Aqui termina a funcao entradaOpcao()
29 /*-----*
30 // Função para obter do usuário
31 /*-----*
32 bool entradaOpcao()
33 {
34     int opcaoDigitada;
35     int digitacaoOK=1;
36     printf("\n Você deseja Calcular outra média? (1=sim/0=Não)");
37     while (digitacaoOK == 1)
38     {
39         scanf("%d",&opcaoDigitada); //Le do teclado a opcao
40         if(opcaoDigitada == 1 || opcaoDigitada == 0 )
41             { digitacaoOK = 0;}
42         else {
43             digitacaoOK = 1;
44             printf("\n Opção invalida (Deve ser 1-Sim 0=Não):%d
45                 ", opcaoDigitada);
46         }
47     } //Fim da estrutura de repetição while iniciada na linha 37
48     if (opcaoDigitada == 1 )
49     {
50         return true; }
51     else{
52         return false;}
53 } //Aqui termina a função entradaOpcao()
54 /*-----*
55 //Função principal do programa
56 /*-----*
57 int main(void)
58 {
59     float media, mediaTurma;

```

```

59     float acumulaMedia=0;
60     int qtdAlunos=0;
61     bool continuaProcessamento = true;
62     while (continuaProcessamento)
63     {
64         entradaNotas(); //Chama a função para obter as 4 notas
65         media = (nota1+nota2+nota3+nota4)/4;
66         printf("\n *-----*");
67         acumulaMedia = acumulaMedia + media;
68         qtdAlunos = qtdAlunos + 1;
69         if (media >= 7.0)
70         {
71             printf("\n Este Aluno foi APROVADO com média =
%.2f",media);}
72         else {
73             printf("\n Este Aluno foi REPROVADO com média =
%.2f",media);
74         }
75         continuaProcessamento = entradaOpcao(); /*Chama função
para verificar se usuario quer continuar*/
76     } //Fim da estrutura de repetição while iniciada na linha 62
77     mediaTurma = acumulaMedia / qtdAlunos;
78     printf("\n *=====*");
79     printf("\n A média da Turma foi = %.2f",mediaTurma);
80     printf("\n Obrigado por usar o sistema");
81     return 0;
82 }

```

Algumas considerações sobre este código:

- Como sempre, o programa é iniciado pela função **main()**;

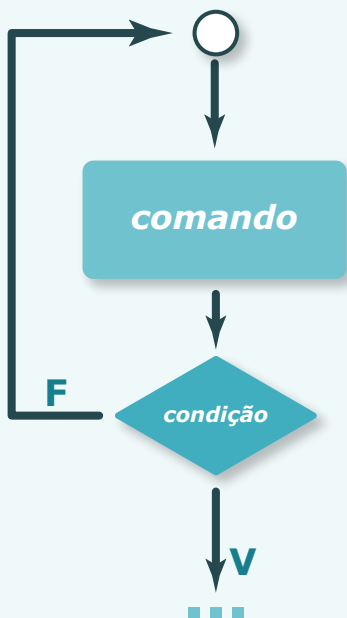


- A função **main()** possui a estrutura de controle de repetição **while** (**Linha 62**), na qual é controlada se o usuário quer calcular a média de um próximo aluno. Importante destacar que o processamento central se dá nesta estrutura, ou seja, cálculo da média do aluno;
- A entrada de dados é feita à parte, pois a função para este fim é chamada na linha 64, função **entradaNotas()** (observe o intervalo de linhas 2 a 28). O objetivo desta função é somente obter as 4 notas e colocá-las nas respectivas variáveis globais **nota1**, **nota2**, **nota3** e **nota4**;
- Na **linha 75** temos um ponto chave, pois é chamada a função **entradaOpcao()** (observe o intervalo de linhas 32 a 52). Esta função tem o objetivo de solicitar ao usuário informar se ele quer calcular média para um novo aluno ou se quer terminar. Esta função retorna um valor **booolano** (V/F) para a variável chave que controla a estrutura de repetição (**continuaProcessamento**);
- Ao final do programa, **linha 77**, faz-se o calculo da média da turma para ser apresentada em tela.

## O laço do-while

A estrutura de repetição **do-while** é semelhante à estrutura de repetição **while**. O que as diferencia é o ponto onde ambas testam a condição de repetição. Enquanto a condição **while** efetua o teste de repeti-

ção antes que o bloco de comandos seja executado, a estrutura **do-while** executa o teste de repetição após o bloco de comandos ser executado. Ou seja, na estrutura **do-while**, o bloco de comando é executado, necessariamente, ao menos uma vez. Podemos observar na Figura 16 o diagrama que representa esta estrutura. Importante observar que o bloco de comando somente é executado novamente se a pre-condição for **falsa**. Quando a condição atingir estado de **verdadeiro**, o bloco de comando não é executado e o laço de repetição termina.



**Figura 16:** Diagrama da estrutura de repetição **do-while**. **Fonte:** Elaboração Própria.

Em linguagem **C**, a estrutura **do-while** possui a seguinte sintaxe:

**do** (comando) **while**( condição ) ;


Seu funcionamento é controlado por uma única expressão (**condição**), cujo valor deve ser **falso** para que o **bloco de comando** seja repetido. A repetição deixa de ser executada somente quando sua **condição** tornar-se **verdadeira**

## Um exemplo simples do comando do-while

O código abaixo aplicou a estrutura do-**while** para mostrar um contador de 1 a 10:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      printf("\n Exemplo de estrutura de repetição do-while");
5      printf("\n *-----*");
6      int contador=0;
7      do
8      {
9          contador++;
10         printf("\n O valor do contador é = %d", contador);
11     }
12     while (contador <10);
13     printf("\n *-----*");
14     printf("\n Valor do contador na saída do laço = %d", contador);
15     return 0;
16 }
```

Na Figura 17 podemos observar o resultado da execução do programa:

```
https://dowhileexemplo.chpereira.repl.run 
clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
❖ clang-7 -pthread -lm -o main main.c
❖ ./main

Exemplo de estrutura de repetição do-while
*-----*
O valor do contador é = 1
O valor do contador é = 2
O valor do contador é = 3
O valor do contador é = 4
O valor do contador é = 5
O valor do contador é = 6
O valor do contador é = 7
O valor do contador é = 8
O valor do contador é = 9
O valor do contador é = 10
*-----*
Valor do contador na saída do laço=10
```

**Figura 17:** Resultado da execução do exemplo simples de programa com laço do-while. **Fonte:** Elaboração Própria.

Para saber mais sobre laços de repetição, acesse o **Podcast** “Decidindo a melhor estrutura de laço para meu programa”.

Podcast 2



# COMANDOS DE DESVIO

Apresentaremos, a seguir, um conjunto de comandos que tem o objetivo de “interferir” no fluxo natural do programa: **return**, **goto**, **break**, **continue** e a **função exit()**.

## FIQUE ATENTO

Sobre fluxo do programa, abordamos anteriormente o seguinte conceito:

“...uma estrutura de sequência, na qual podemos observar que, ao término de cada instrução, dá-se início a uma nova instrução. As instruções são executadas em uma sequência, parte-se da primeira instrução do algoritmo e, pelo fato de não haver desvios (Condições e laços de repetição), segue “gravitacionalmente” até alcançar a última instrução da sequência.”

## O comando return

Utiliza-se o comando **return** para retornar da função chamada para a função que a chamou. Considera-se **return** como comando de desvio, porque este faz com que a execução retorne (salte de volta) ao ponto onde a chamada da função foi feita.

Importante: Se a função for declarada com devolução de valores (mais detalhes abaixo, em **Saiba Mais**), este valor deve ser retornado pelo comandando **return**. A sua sintaxe é:

**return** expressão;

Importante: a expressão é opcional. Isso acontece se a função retornar e não tiver um valor associado a ela (**void**).

No código abaixo, temos três comandos “**return**”, e os que estão codificados nas linhas 7 e 9 referem-se ao retorno da função `verificaPar()`, que deve retornar um valor booleano para indicar se o numero é par ou ímpar. Note que a função **main()** sempre emite um **return 0** (linha 19), pois ela foi declarada como **int main()**.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  //função com retorno de valor tipo booleano
4  bool verificaPar(int numero)
5  {
6      if (numero % 2 == 0){
7          return true; //retorna true, indica que o valor é par
8      }else {
9          return false; //retorna false, indica que o valor é
10     }
11 }
12 int main(void)
13 {
14     int numeroEntrada = 4;
```

```
15     if (verificaPar(numeroEntrada))
16         printf("\n Sim. O numero %d é PAR", numeroEntrada);
17     else
18         printf("\n Não. O numero %d é IMPAR", numeroEntrada);
19     return 0;
20 }
```

## SAIBA MAIS

Estudamos anteriormente que um código em **C** pode ser estruturado com várias funções, e que a função que obrigatoriamente deve aparecer é a **main()**.

A sintaxe básica da declaração da função é:

**tipo\_de\_retorno** nome\_da\_função (lista\_de\_parâmetros)

Onde:

- **tipo\_de\_retorno**: é o tipo de dado que a função retorna ao chamador (void significa que não há retorno)
- **nome\_da\_função**; é o nome pelo qual a função será chamada
- **lista\_de\_parâmetros**: São os parâmetros de entrada de dados que a função poderá tratar.

Uma função pode, ou não, ao seu término **devolver um valor**. Este valor é devolvido por meio do comando **return**. Importante ressaltar que o valor retornado no comando **return** deve ser do mesmo tipo declarado em “**tipo\_de\_retorno**”, que consta

na declaração da função. Exemplo: para uma função que foi declarada sem retorno [exemplo: **void** nome()], deverá ser emitido somente o comando **"return;"**. Já para o exemplo de uma função declarada com retorno tipo **int** [exemplo: **int** func1()], deverá ser emitido o comando **return** valor, onde "valor" deverá uma variável do tipo **int**.

## O comando goto

O comando **goto** faz o desvio para o ponto do programa que o programador especificar (label). Este comando deve ser utilizado com prudência, pois o desvio **goto** foge do fluxo normal do programa. E isso proporciona uma grande possibilidade de se perder o controle do programa.

Sua sintaxe é:

```
goto rótulo;
```

```
...
```

```
...
```

```
rótulo:
```

Onde **rótulo** é qualquer rótulo válido, antes ou depois, do comando **goto**.

Exemplo de código com comando **goto**, onde o laço de repetição é controlado pelo programador, com o comando goto (Volta para o label **loop1**), e somente sai quando o contador chegar a 10.



```

1  #include <stdio.h>
2  int main(void) {
3      int contador = 0;
4      loop1:
5      if (contador++ < 10)
6      {
7          printf("\n Contador = %d",contador);
8          goto loop1;
9      }
10     return 0;
11 }

```

## FIQUE ATENTO

Não utilize o comando **goto** sem alguma necessidade que realmente justifique, pois o **C** oferece uma gama abrangente para controle de fluxo e o **goto** traz o risco de se perder o controle do fluxo. Esse é um erro de lógica em que os programadores não ficam muito felizes quando acontece!

## O comando break

O comando **break** pode ser utilizado em duas situações: 1-) Terminar um **case** (comando **switch**); 2-) Forçar a terminação de um laço de repetição (**for**; **while**; **do-while**).

No código abaixo, temos um exemplo do comando **break** (linha 7) dentro de uma estrutura de repetição **for** (intervalo de linhas 3:8). Nesse caso, o comando **for** está codificado para executar o comando **printf** por 100 vezes. Porém, note que, na décima iteração, será executado o comando **break**, pois, quando contador chegar a 10, ele será executado. Logo, o laço de repetição será encerrado neste ponto e será impresso (**printf**) somente até a décima ocorrência.

```
1  #include <stdio.h>
2  int main(void)
3  {
4      for (int contador=1;contador <=100;contador++)
5      {
6          printf("\n Valor de contador = %d",contador);
7          if (contador ==10)
8              break;
9      }
10     return 0;
11 }
```

## A função `exit()`

Esta função, ao ser executada, encerra a execução do programa imediatamente. A sua sintaxe é:

```
void exit(int código_de_retorno)
```

Onde: o `código_de_retorno` é retornado ao processo chamado (geralmente é o sistema operacional). O zero indica término normal. Diferentemente disso,

o sistema operacional entenderá que o programa terminou com situação anormal.

## SAIBA MAIS

Todo programador deve estar atento às **condições de exceções**, que são elementos presentes de forma oculta nos sistemas computacionais. Por exemplo: supondo que o seu programa chama uma função externa a ele. Supondo também que esta função tenha como objetivo devolver um valor válido (por exemplo: dígito verificador de CPF) para seu programa. O programador deverá codificar a chamada da função, bem como codificar a verificação do valor devolvido. É aí que entra em cena a utilização do código de retorno, pois, neste caso, se o valor retornado for inválido, o programador deverá encerrar o programa de forma anormal, com o comando **return** e um código específico de erro, que caracterizará esse erro.

Abaixo temos um exemplo de utilização de **exit()** (linha21) para indicar um erro lógico programa.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  //função com retorno de valor tipo booleano
5  bool verificaPar(int numero)
6  {
7      if (numero % 2 == 0){
8          return true; //retorna true, indica que o valor é par
9      }else {
10         return false; //retorna true, indica que o valor é
11     }
12 }
13 int main(void)
14 {
15     int numeroEntrada = 3;
16     if (verificaPar(numeroEntrada))
17     {
18         printf("\n OK Era esperado PAR");
19     } else {
20         printf("\n Não retornou par. Vou cancelar o
21             programa!!!\n");
22         exit(1); //termina o programa neste ponto
23     }
24     return 0;
25 }

```

Observe o resultado do comando **exit()**, na Figura 18 . Note que aparece a mensagem “**exit status 1**”. Esta mensagem não foi emitida por este programa, e sim pela função **exit()**.

```
https://exemploExit.chpereira.repl.run

clang version 7.0.0-3~ubuntu0.18.04.1 (tags/RELEASE_700/final)
❯ clang-7 -pthread -lm -o main main.c
❯ ./main

Não retornou par. Vou cancelar o programa!!!
exit status 1
```

**Figura 18:** Resultado do comando `exit()`. **Fonte:** Elaboração Própria.

## O comando **continue**

O comando **continue** possui comportamento semelhante ao comando **break**. Este último força término do laço de repetição; já o comando **continue** vai para a próxima iteração do laço de repetição (volta para o início deste). Para o laço **for**, o comando **continue** faz com o teste de controle e o incremento do laço **for** seja executado. Para os comandos **while** e **do-while**, o controle do programa passa para o teste condicional.

No código abaixo, temos um exemplo da utilização do comando **continue** (linha 23), sendo utilizado dentro da estrutura de repetição **for** (intervalo de linhas 15:26). Observe que a presença deste comando faz com que o controle do programa volte para a linha 15 imediatamente, portanto, o comando da linha 25 não é executado.

```

1  #include <stdio.h>
2  #include<stdlib.h>
3  #include <stdbool.h>
4  //função com retorno de valor tipo booleano
5  bool verificaPar(int numero)
6  {
7      if (numero % 2 == 0){
8          return true; //retorna true, indica que o valor é par
9      }else {
10         return false; //retorna true, indica que o valor é
11     }
12 }
13 int main(void)
14 {
15     for (int numero=1;numero<=10;numero++)
16     {
17         printf("\n *-----*");
18         if (verificaPar(numero))
19         {
20             printf("\n OK Era esperado PAR");
21         } else {
22             printf("\n Não retornou par");
23             continue; //Para números ímpares, o programa volata
                para a linha 15 e não executa a linha 25
24         }
25         printf("\n produto numero par %d*2 é:%d",numero, numero *
26             2);
27     } //fim do for
28     return 0;
29 }

```

Na Figura 19 , podemos observar o resultado do programa com exemplo do comando **continue**:

```
https://continue.chpereira.repl.run
clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/final)
> clang-7 -pthread -lm -o main main.c
> ./main

*-----*
Não retornou par
*-----*
OK Era esperado PAR
produto de numero par 2 * 2 é: 4
*-----*
Não retornou par
*-----*
OK Era esperado PAR
produto de numero par 4 * 2 é: 8
*-----*
Não retornou par
*-----*
OK Era esperado PAR
produto de numero par 6 * 2 é: 12
*-----*
Não retornou par
*-----*
OK Era esperado PAR
produto de numero par 8 * 2 é: 16
*-----*
Não retornou par
*-----*
OK Era esperado PAR
produto de numero par 10 * 2 é: 20
```

**Figura 19:** Resultado do exemplo da utilização do comando *continue*.  
**Fonte:** Elaboração Própria.

# CONSIDERAÇÕES FINAIS

Caro estudante, aprendemos e praticamos juntos conceitos centrais que fazem parte de toda e qualquer linguagem de programação: Controle de fluxo do programa e estruturas de decisão. Detalhamos cada estrutura com suas respectivas representações gráficas, seguidas de codificação na linguagem C. Observamos as sintaxes e exemplos de estruturas simples de if, bem como estruturas mais complexas, como as estruturas if-else-if e ninho de if. Completando este aprendizado de tomada de decisão, verificamos uma alternativa interessante ao uso do if, o comando switch, que traz ao programador uma forma elegante na codificação de tomada de decisão por chaveamento.

Quanto ao controle de fluxo, aprendemos os variados paradigmas na condução do fluxo de um programa. Abordamos os laços de repetição, que são elementos essenciais para um algoritmo atender às várias soluções de problemas computacionais. Aprendemos como manipular os laços de repetição, como for, while e do-while, que estão presentes em nas linguagens de programação. Analisamos suas estruturas representadas graficamente, suas sintaxes e seus respectivas exemplos implementados na linguagem C.



Tivemos a preocupação de demonstrar graficamente as estruturas de decisão e estruturas de controle de fluxo. Mas, para que este aprendizado seja proveitoso e consistente, foi muito importante a sua participação, ao reproduzirmos juntos, cada exemplo na linguagem C. Conforme ressaltamos desde o início do curso, a prática é fator muito importante no aprendizado da lógica de programação; aconselhamos fortemente que você revise cada exemplo feito e comentado neste módulo, e tente refazê-los de forma independente, ou seja, antes de consultar o resultado resolvido.

Parabéns por ter chegado até este ponto, por se tratar de um denso conteúdo da lógica de programação, em que os conceitos são fundamentais para a continuidade da jornada. Com este aprendizado, sinta-se fortalecido para lograr êxito no campo da programação de computadores.

# Síntese



## CONTROLE DE FLUXO DE PROGRAMA

Nesta unidade tivemos contato com os conceitos da linguagem de programação, que são o controle de fluxo e tomada de decisão. Aprendemos a aplicar os paradigmas de controle de laço de repetição, quais sejam, os paradigmas `for`, `while` e `do-while`.

Abordamos a tomada de decisão, onde aprendemos a estrutura da decisão simples `if` e as estruturas mais complexas como `if-else-if` e ifs aninhados. Aprendemos a outra estrutura de decisão do comando `switch`. Para finalizar aprendemos a manipular os comandos de desvios da linguagem C.

Para este fim, estruturamos este material da seguinte forma:

### Tomada de Decisão

Tratamento de Verdadeiro e Falso

### Comandos de Seleção

Comando de seleção: ***if***

Comando `if`'s Aninhados

A escalada de **`if-else-if`**

O Operador Ternário ?

O comando **`switch`**

### Programação com laço de repetição

O laço **`for`**

O laço **`while`**

O laço **`do-while`**

### Comandos de Desvio

O comando **`return`**

O comando **`goto`**

O comando **`break`**

A função **`exit()`**

O comando **`continue`**

# Referências

---

ALVES, W. P. **Linguagem e Lógica de Programação**. São Paulo: Érica, 2015.

EDELWEIS, N.; LIVI, M. A. C.; LOURENÇO, A. E. **Algoritmos e programação com exemplos em Pascal e C**. São Paulo: Bookman, 2014.

FORBELLONE, A. L. V; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. São Paulo: Prentice Hall, 2015.

MANZANO, J. A. N. G; MATOS, E; LOURENÇO, A. E. **Algoritmos**: Técnicas de Programação. São Paulo: Érica, 2015.

MANZANO, J. A. N. G; OLIVEIRA, J. F. **Algoritmos**: Lógica para desenvolvimento de programação de computadores. São Paulo: Saraiva, 2016

PERKOVIC, L. **Introdução à computação usando Python**. Rio de Janeiro: LTC-Livros Técnicos e Científicos, 2016.

ZIVIANI, N. **Projeto de Algoritmos**: Com implementações em Pascal e C. São Paulo: Cengage Learning, 2011.

**FaM**  
**ONLINE**