5950257 - Programação de Computadores

Aula 07 - Estruturas de Controle

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

- Compreender que o fluxo de controle define a ordem e o caminho que as instruções do programa seguirão.
- Entender o conceito de escopo em agrupamento de instruções e variáveis.
- Implementar lógica condicional usando **if**, **else if** e **switch**.
- Implementar os três paradigmas de laço de repetição: **for**, **while** e **do-while**.
- Explorar expressões booleanas e operadores lógicos (**and**, **or**, **not**).
- Aplicar esses conceitos na resolução de problemas práticos, como validação de entrada e contagem de valores.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Observação

Para os exercícios desta aula, você pode utilizar IDEs online como:

- https://www.onlinegdb.com/online_c_compiler
- https://www.programiz.com/c-programming/online-compiler/

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Fluxo de Execução

- O computador não "lê" o código como humanos leem um livro; ele executa instruções sequencialmente, a menos que uma estrutura de controle altere esse fluxo.
- **Fluxo Sequencial (Padrão):** A CPU executa a linha 1, depois a linha 2, e assim por diante.
- **Estruturas de Controle:** São os mecanismos que desviam esse fluxo linear (decisão) ou repetem trechos (repetição).
- **Importância:** Sem estruturas de controle, um programa em C seria apenas uma lista estática de instruções sem inteligência.

```c
int x = 10; // Linha 1
printf("%d", x); // Linha 2 (Executa)
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Blocos de Comandos

- **Conceito:** Um bloco de comandos é um grupo de instruções que devem ser executadas como uma única unidade lógica.
- **Sintaxe:** As chaves `{}` definem um bloco. Tudo dentro delas pertence ao mesmo escopo lógico.

```
{
    instrução 1;
    instrução 2;
    // ...
    instrução n;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Escopo

- **Escopo de Variáveis:** Uma variável declarada dentro de um bloco só existe dentro desse bloco.
- Blocos servem para agrupar código para delimitar o escopo das variáveis locais.

```c
void calcular_area(int largura, int altura) {
    int area = largura * altura; // Variável 'area' só existe neste escopo!
    printf("A área é: %d\n", area);
} // Aqui, 'area' sai do escopo e não pode ser usada fora.

int main() { // Escopo: main
    int largua = 10;
    int altura = 5;
    calcular_area(largura, altura);
    printf("A área é: %d\n", area); // ERRO! area não é visível aqui
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão

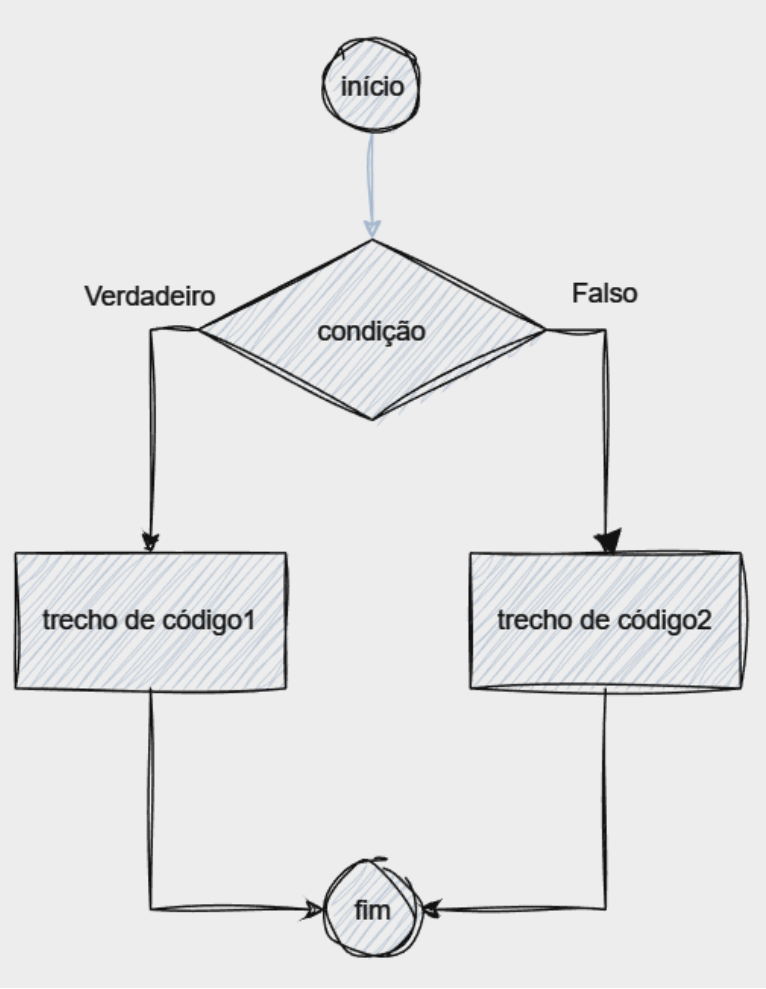
Mecanismos que permitem decidir o fluxo de execução de um programa.

Estruturas de Decisão: **if** E **else**

O comando **if** permite que o programa escolha qual caminho seguir com base na avaliação de uma condição **booleana**.

```c
/* Verificação de idade para acesso: */
int idade = 18;
if (idade >= 18) {
    printf("Acesso concedido.\n");
} else {
    printf("Acesso negado. Menor de idade.\n");
}
```

O bloco **if** é executado se a condição for verdadeira (**true**). Caso contrário, o bloco **else** é executado.



> Na imagem, fluxograma **if-else**. Fonte: TutorialDev.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: `if` E `else` (cont.)

Se você esquecer as chaves em um **`if`** ou **`for`**, o compilador só considerará o *próximo comando* como pertencente ao bloco condicional, ignorando qualquer coisa que venha depois!

```
int x = 10;
if (x > 5)
    printf("Maior que 5\n"); // Apenas este comando está no bloco IF
else
    printf("Menor ou igual a 5\n"); // Este é o bloco ELSE
```

> **Observação**: Em C, qualquer valor diferente de **`0`** é considerado "Verdadeiro" (**`true`**). O valor **`0`** é "Falso" (**`false`**).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: `if` E `else` (cont.)

```
ehDomingo = false

if (ehDomingo == true) {
    printf("Hoje é domingo");
}
else {
    printf("Hoje é não domingo");
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: `if` E `else` (cont.)

- Quando há mais de duas possibilidades mutuamente exclusivas, usamos **`else if`**.
- Assim que uma condição é verdadeira, o bloco correspondente é executado e **todo o restante** da estrutura é ignorado.

Exemplo Prático (Sistema de Notas):

```c
char nota = 'B';
if (nota == 'A') {
    printf("Excelente! Parabéns.\n");
} else if (nota == 'B') {
    printf("Bom desempenho. Continue assim.\n"); // Este bloco é executado
} else if (nota == 'C') {
    printf("Precisa melhorar o foco.\n");
} else {
    printf("Nota inválida fornecida.\n");
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: `if` E `else` (cont.)

- As estruturas **`if/else`** frequentemente precisam avaliar *múltiplos critérios simultaneamente*.
- Os Operadores Booleanos **`&&`** (E), **`||`** (OU) e **`!`** (NÃO) permitem que você combine testes lógicos para criar condições complexas.

```c
int idade = 18;

if (idade >= 18 && idade < 65) {
    printf("Você é um adulto ativo.\n");
} else if (idade < 18) {
    printf("Você é menor de idade.\n");
} else {
    printf("Você é aposentado.\n");
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Elegibilidade de Desconto

```c
#include <stdio.h>

int main() {
    int idade = 18;
    int is_premium = 1;

    // A condição só é verdadeira se AMBOS forem verdadeiros:
    if (idade >= 18 && is_premium == 1) {
        printf("Desconto aplicado! Cliente elegível.\n");
    } else if (idade < 18 && is_premium == 1) {
        // Exemplo de outra condição: Premium, mas menor de idade.
        printf("Atenção: Cadastro premium, mas abaixo da idade mínima.\n");
    } else {
        printf("Cliente não elegível para o desconto especial.\n");
    }
}
```

> Substitua os valores das variáveis para alterar o comportamento do programa.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo: Verificação de Status

```c
#include <stdio.h>

int main() {
    int esta_bloqueado = false;
    int senha_correta = true;

    // Usamos NOT: !esta_bloqueado significa que 'esta_bloqueado' deve ser 0 (Falso).
    if (!esta_bloqueado && senha_correta == 1) {
        printf("Login bem-sucedido. Status OK.\n");
    } else if (esta_bloqueado == 1) {
        // Se o primeiro critério falhou, verificamos a causa: bloqueio.
        printf("Falha de login. Conta temporariamente suspensa.\n");
    } else {
        printf("Login falho. Verifique sua senha.\n");
    }
}
```

> Substitua os valores das variáveis para alterar o comportamento do programa.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: switch

O comando **switch** é otimizado para comparação de valores exatos (equivalência), ideal para menus ou estados discretos.

- **Sintaxe:** Usa **case** para definir valores e **break** para sair do bloco.
- **Quando usar?** Ideal quando a variável é **discreta**. É mais limpo que múltiplos **else if**.

```
switch (expressao) {
    case contante1:
        // escopo do caso 1
        break; // PARE aqui!
    case contante2:
        // escopo do caso 2
        break; // PARE aqui!
    default: // Executado se nenhum case for verdadeiro
        // escopo do caso default
        break; // PARE aqui!
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```c
int dia = 3;

switch (dia) {
    case 1: printf("Domingo"); break;
    case 2: printf("Segunda"); break;
    case 3: printf("Terça"); break;
    case 4: printf("Quarta"); break;
    case 5: printf("Quinta"); break;
    case 6: printf("Sexta"); break;
    case 7: printf("Sábado"); break;
    default: printf("Dia inválido."); break;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Decisão: `switch`

- **Fall-through:** Se esquecer o **`break`**, o código executa todos os casos subsequentes até encontrar um **`break`** ou fim do switch. Isso é intencional, mas muitas vezes causa bugs.
- **Padrão de Fallback:** Use **`default`** para lidar com valores inesperados.

```c
int dia = 3;

switch (dia) {
    case 1: printf("Domingo"); break;
    case 2: printf("Segunda"); break;
    case 3: printf("Terça");
            // FALTA BREAK! Vai imprimir "Quarta" também.
    case 4: printf("Quarta"); break;
    default: printf("Dia inválido."); break;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício

Implemente um algoritmo que dada a idade de um nadador (lida do teclado) imprime a categoria na qual ele está:

- infantil A = 5 - 7 anos
- infantil B = 8-10 anos
- juvenil A = 11-13 anos
- juvenil B = 14-17 anos
- adulto = maiores de 18 anos

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição

Mecanismos que permitem a execução repetida de um bloco de código.

Estruturas de Repetição: `for`

O loop **`for`** é ideal quando sabemos quantas vezes queremos repetir ou quando o contador é explícito.

Fluxo:

1. Inicializa a variável de controle.
2. Verifica a condição antes de cada iteração.
3. Executa o bloco.
4. Atualiza o contador (incremento).

```c
for (int i = 0; i < 5; i++) {
    printf("Iteração %d\n", i);
} // 'i' não existe aqui fora do loop
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição: Loops aninhados

Um loop aninhado ocorre quando um laço de repetição (o **loop interno**) é colocado dentro do corpo de outro laço de repetição (o **loop externo**).

```
// Estrutura geral:
for (int i = 0; i < N_LINHAS; i++) { // Loop Externo (Linhas)
    // O corpo do loop externo executa para cada linha 'i'
    for (int j = 0; j < N_COLUNAS; j++) { // Loop Interno (Colunas)
        // Esta instrução será executada N_LINHAS * N_COLUNAS vezes.
        printf("(%d, %d)\n", i, j);
    }
}
```

> Pense em uma planilha eletrônica (como o Excel). O loop externo percorre as linhas (índice `i`), e o loop interno percorre as colunas (índice `j`). Para cada linha, você precisa repetir a ação de percorrer todas as colunas.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição: Loops aninhados (cont.)

Entendendo a Ordem de Execução (Controle de Fluxo):
O loop interno não é executado apenas uma vez. Ele deve completar *todas* as suas iterações para cada *única* iteração do loop externo.

Passos Lógicos:

1. O contador `i` (externo) incrementa de 0 até N-1.
2. Para o valor atual de `i`, o contador `j` (interno) é resetado para 0.
3. O loop interno executa todas as suas iterações, completando seu ciclo completo.
4. Somente após o loop interno terminar, o controle volta ao loop externo e incrementa `i`.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```c
#include <stdio.h>

int main()
{
    // Exemplo: Quadrado 5x5
    int tamanho = 5;
    for (int i = 0; i < tamanho; i++) { // Linhas
        for (int j = 0; j < tamanho; j++) { // Colunas
            printf("* ");
        }
        printf("\n"); // Quebra de linha após completar a linha 'i'
    }

    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício

Utilize dois loops aninhados (`for`) para imprimir um triângulo retângulo onde a linha `i` contém `i` asteriscos.

```
*
**
***
****
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição: `while` E `do-while`

`while`: Verifica a condição *antes* da execução. Se for falso logo no início, o bloco nunca roda (zero iterações).

```
// Sintaxe básica
while (condicao_booleana) {
    // Bloco de código que será repetido
}
// O programa continua aqui após a saída do loop
```

Observações Importantes:

- A condição deve ser capaz de se tornar falsa em algum momento para evitar um **Loop Infinito**.
- O fluxo é sempre: **Verifica Condição** $\rightarrow$ **Executa Corpo** $\rightarrow$ **Atualiza Variável (Iteração)**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição: `while` E `do-while` (cont.)

- O loop **`while`** é um *pre-test loop*. Isso significa que a condição é avaliada (testada) **antes** da primeira execução do bloco de código.
- Se a condição for falsa na primeira checagem, o corpo do loop NUNCA será executado.

```c
#include <stdio.h>

int main() {
    int contador = 0;
    int limite = 5;
    while (contador < limite) {
        printf("Execução %d\n", contador);
        contador++;  // Incrementa o contador
    }
    printf("Loop finalizado! Contador atual: %d\n", contador);
    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício

Escreva um programa que utilize o loop **while** para imprimir todos os números pares entre 2 e 20 (inclusive). O contador deve ser incrementado de forma eficiente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Estruturas de Repetição: `while` E `do-while` (cont.)

O loop **do-while** é uma estrutura de repetição que garante, por definição, a execução do seu bloco de código pelo menos uma vez. Ele só verifica a condição após o término da primeira iteração.

- **while**: Teste $\rightarrow$ Execução $\rightarrow$ Teste... (Se falso no início, nunca roda).
- **do-while**: Execução $\rightarrow$ Teste $\rightarrow$ Execução... (Roda pelo menos uma vez).

```c
int x = 0;

do {
    printf("Executa pelo menos uma vez\n");
    x++;
} while (x < 5); // Verifica depois da execução.
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```c
#include <stdio.h>

int main() {
    int contador = 10;
    do {
        printf("Executando bloco. Contador: %d\n", contador);
        contador++; // Incrementa o contador
    } while (contador < 5); // Condição falsa desde o início
    printf("\nLoop finalizado. O código rodou UMA VEZ, mesmo que a condição fosse FALSA.\n");
    return 0;
}
```

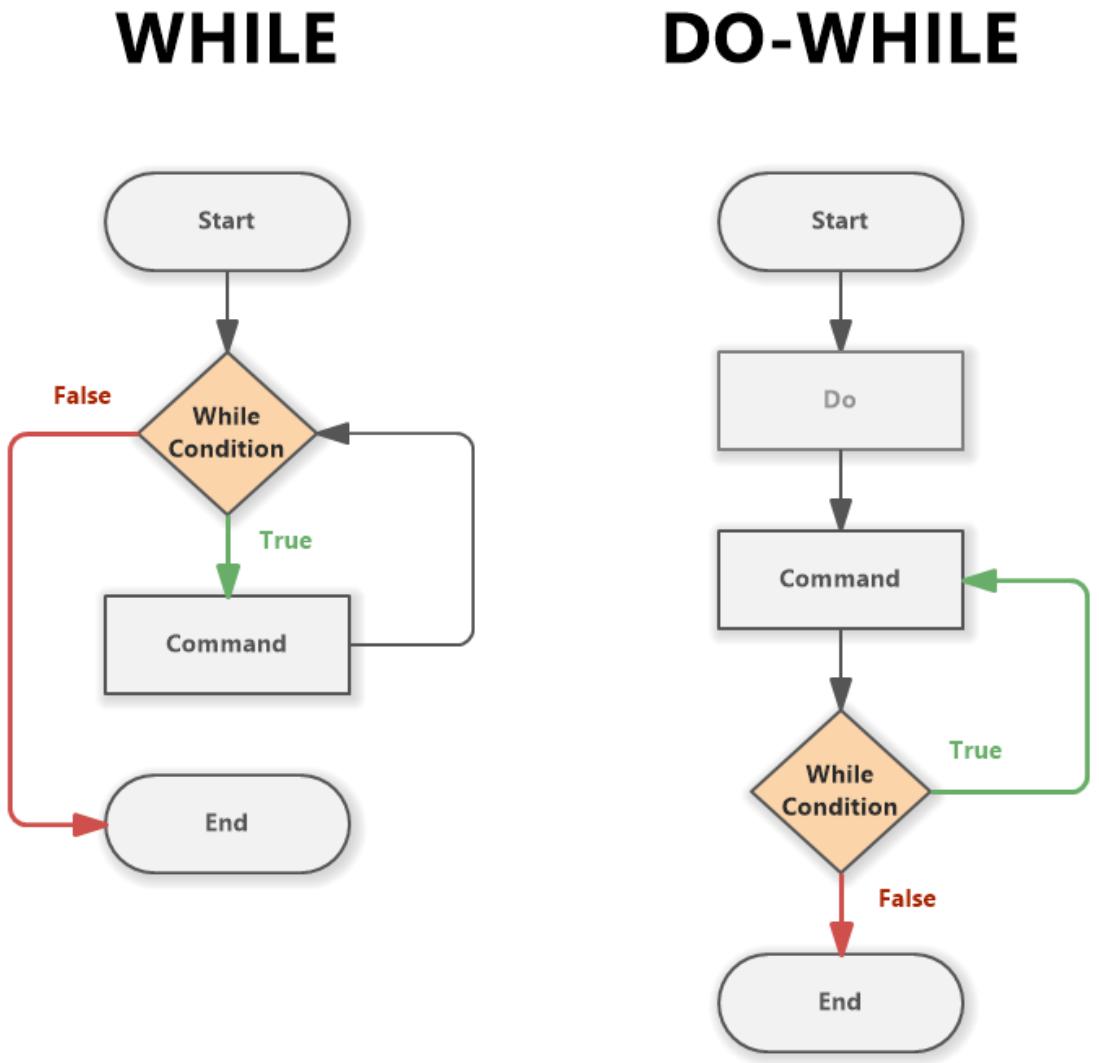
> Note que **contador** começa em 10 e a condição é **< 5**. Um loop **while** teria pulado o bloco. O **do-while**, no entanto, executou o código uma vez antes de verificar a condição falsa.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Comparação `while` e `do-while`

Característica	while	do-while
Ponto de Teste	Início do bloco (Pré-Teste)	Fim do bloco (Pós-Teste)
Execução Mínima	Zero vezes	Uma vez
Sintaxe Chave	`while (condicao)`	`do { ... } while (condicao);`
Melhor Uso	Processamento condicional em *streams* ou buffers.	Interação com usuário, menus de validação.

> Na imagem: Fluxogramas de loops `while` e `do-while`. Fonte: Egoshkin Danila Igorevich



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício

Implemente um sistema de menu básico utilizando **do-while**. O menu deve exibir as opções A, B, C ou S. Dentro do loop, o programa deve solicitar ao usuário que escolha uma opção. O loop só deve parar quando a opção escolhida for **S** (Sair). Se a entrada não for válida, exiba um erro e continue pedindo até receber 'S'.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Loop infinito

Ao utilizar o comando **while**, é fundamental adotar boas práticas para evitar a ocorrência de loops infinitos, que podem comprometer a execução do programa, causando a sua paralisação.

- **Definição**: Um loop infinito ocorre quando a condição de parada nunca se torna **false**, resultando em uma execução contínua sem término.
- **Como evitar**: é essencial garantir que a variável de controle seja devidamente atualizada dentro do bloco de repetição.

```
while (true):
    printf("Loop");
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Controle de fluxo interno: `break` e `continue`

- `break`: Sai imediatamente do bloco mais próximo (loop ou switch). Equivalente a "quebrar" a corrente da execução.
- `continue`: Pula o restante do corpo atual do loop e vai para a próxima iteração (incremento/verificação).

```c
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // Para no 5. Loop encerra.
    }
    printf("%d ", i);
}
for (int j = 0; j < 5; j++) {
    if (j % 2 == 0) {
        continue; // Pula números pares
    }
    printf("%d\n", j);
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Mais Exemplos

```
int contador = 10;
while (count > 0) {
    printf("Inicio da iteracao");
    if (count == 5):
        printf("Fim da iteracao por break");
        break;
        printf("Fim da iteracao");
}

printf("Saiu do Loop");
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Boas Práticas

Código limpo é código legível e manutenível.

- **Indentação:** Use sempre espaços ou tabs para indicar blocos.
- **Nomes de Variáveis:** Em loops, evite nomes genéricos. Use **index** ou **contador** se o contexto permitir.
- **Manutenção:** Comentários explicam *por que*, não apenas *o que*.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Resumo e Próximos Passos

O domínio das estruturas de controle é a base da lógica algorítmica.

Resumo:

- **Blocos:** `{}` definem escopo e agrupamento.
- **Decisão:** `if/else` para condições, `switch` para estados discretos.
- **Repetição:** `for` (contagem), `while` (condição prévia), `do-while` (condição posterior).
- **Controle:** `break` (sair), `continue` (pular iteração).

Próxima Aula:

- Variáveis Compostas Homogêneas: vetores, matrizes e strings.

Material Adicional

- C para Seres Humanos
- Vídeo do YouTube: Lógica de programação 10 - Estruturas de repetição
- Loops in C por GeeksForGeeks
- Exercícios: https://www.w3resource.com/c-programming-exercises/for-loop/index.php

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercícios Práticos

Problema 1: O Menu de Simulação. Crie um programa que exiba um menu simples com as opções 1 a 3. O programa deve ler o número digitado pelo usuário e utilizar **switch** para decidir qual mensagem correspondente será exibida. Se o usuário digitar algo fora do padrão, use **default**.

Problema 2: Tabuada Dinâmica. Escreva um programa que peça ao usuário um número inteiro **n** e imprima a tabuada de 1 a 10 para esse número utilizando um loop **for**.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 1: Escopo e Visibilidade

Qual é o resultado da execução deste código? Explique o motivo.

```c
int x = 5;
{
    int x = 10;
    printf("%d", x); // O que sai aqui?
}
printf("%d", x); // O que sai aqui?
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 2: Lógica de Switch

Por que o **break** é obrigatório no final de cada **case** em um bloco **switch**? O que acontece se omitirmos?

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questão 3: Loop Infinito

Identifique o erro lógico neste loop e explique como corrigi-lo para imprimir números de 1 a 5.

```c
int i = 1;
while (i <= 5) {
    printf("%d\n", i);
} // Falta atualização de i
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br