



TÉCNICAS DE PROGRAMAÇÃO

Aula 02 – Estruturas de Decisão e Repetição em C++



Objetivos de Aprendizagem

1. Descrever e identificar a utilização das estruturas de decisão e de repetição.
2. Criar algoritmos com estruturas de decisão, de repetição.



Condições

- Os condicionais são usados para executar cálculos ou ações diferentes, dependendo de uma condição ser avaliada como verdadeira ou falsa.



A Instrução if

- A instrução **if** executa instruções quando uma expressão é verdadeira.
- Por esse motivo, o **if** também é conhecido como estrutura de decisão.
 - Assume a forma:

```
if (/* condição */)
{
    /* código */
}
```



A Instrução if

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

```
int main() {  
    int nota = 89;
```

```
    if (nota > 75)  
        printf("Você passou.\n");
```

```
    return 0;  
}
```



Operadores Relacionais

- Existem seis operadores relacionais que podem ser usados para formar uma expressão booleana, que retorna verdadeiro ou falso:
 - < menos que
 - <= menor ou igual a
 - > maior que
 - >= maior ou igual a
 - == igual a
 - != não é igual a



Operadores Relacionais

```
int num = 41;  
num += 1;  
if (num == 42) {  
    printf("Você ganhou!");  
}  
...  
int in_stock = 20;  
if (in_stock)  
    printf("Pedido recebido.\n");
```



A instrução if-else

- A instrução **if** pode incluir uma cláusula opcional **else** que executa instruções quando uma expressão é falsa.
- Por exemplo, o programa a seguir avalia a expressão **e**, em seguida, executa a instrução da cláusula **else**.



Exemplo

```
#include <stdio.h>

int main() {
    int nota = 89;

    if (nota >= 90)
        printf("Top 10%%.\n");
    else
        printf("Menos de 90.\n");

    return 0;
}
```



Exercício

- Altere a função que lê as notas do aluno e imprime a média para que a mensagem impressa se comporte da seguinte maneira:
 - Se a média foi maior ou igual a 7.0 a mensagem deve ser “Aprovado”. Caso contrário a mensagem deve ser “Estude mais para a final”.

Exercício

- O que o programa abaixo faz?

```
#include <stdio.h>
main() {
    int a,b;
    printf("Digite 2 números: ");
    scanf("%d %d",&a,&b);
    if (b){
        printf("%f",a/b);
    }else {
        printf("Não posso fazer a divisao");
    }
}
```

Operador Ternário Condicional

- Outra maneira de formar uma instrução **if-else** é usando o operador **?** : Em uma expressão condicional.

`<condição> ? <valor_verdade> : <valor_falso>`

- O operador **?** : Pode ter apenas uma instrução associada ao **if** e ao **else**.

Exemplo

```
#include <stdio.h>
int main() {
    int y;
    int x = 3;
    y = (x >= 5) ? 5 : x;
    /* Isto é equivalente a:
       if (x >= 5)
           y = 5;
       else
           y = x;
    */
    return 0;
}
```



Instruções if aninhadas

- Uma instrução **if** pode incluir outra instrução **if** para formar uma instrução aninhada.
- Aninhar um **if** permite que uma decisão seja baseada em requisitos adicionais.

Instruções if aninhadas

- Considere a seguinte declaração:

```
if (lucro > 1000)
    if (clientes > 15)
        bonus = 100;
    else
        bonus = 25;
```



Instruções if aninhadas

- Recuar adequadamente as instruções aninhadas ajudará a esclarecer o significado para o leitor:

```
if (lucro > 1000) {  
    if (clientes > 15)  
        bonus = 100;  
}  
else  
    bonus = 25;
```


Importância da indentação

- Cada código que estiver dentro de um bloco deve estar um nível a mais de indentação à direita do bloco mais externo.
- **Indentar** o código é **FUNDAMENTAL** para compreensão do escopo de código (identificar onde começa e onde termina).

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



Operadores Lógicos

- Operadores lógicos `&&` e `||` são usados para formar uma expressão booleana composta que testa várias condições.
- Um terceiro operador lógico é `!` usado para reverter o estado de uma expressão booleana.



O operador && (AND “E”)

- O operador AND lógico && retorna um resultado verdadeiro somente quando ambas as expressões são verdadeiras.
- Por exemplo:

```
if (n > 0 && n <= 100)
    printf("Range (1 - 100).\n");
```



O operador && (AND “E”)

- Uma expressão booleana composta é avaliada da esquerda para a direita.
- A avaliação é interrompida quando nenhum teste adicional é necessário para determinar o resultado:
 - portanto, certifique-se de considerar a disposição dos operandos quando um resultado afeta o resultado de um resultado posterior.

O operador `||` (OR “OU”)

- O operador OR lógico `||` retorna um resultado verdadeiro quando qualquer expressão ou ambas as expressões são verdadeiras.
- Por exemplo:

```
if (n == 'x' || n == 'X')  
    printf("Roman numeral value 10.\n");
```



O operador `||` (OR “OU”)

- Qualquer número de expressões pode ser unido por `&&` e `||`.
- Por exemplo:

```
if (n == 999 || (n > 0 && n <= 100))  
    printf("Input valid.\n");
```

O operador ! (NOT “NÃO”)

- O operador NOT lógico ! retorna o inverso de seu valor.
 - NOT true retorna false e NOT false retorna true.
- Por exemplo:

```
if (!(n == 'x' || n == 'X'))  
    printf("Roman numeral is not 10.\n");
```



A instrução if-else if

- Quando uma decisão entre três ou mais ações é necessária, a instrução **if-else if** pode ser usada.
- Pode haver várias outras cláusulas **if** e a última cláusula **else** é opcional.

A instrução if-else if

```
int nota = 89;

if (nota >= 90)
    printf("%s", "Top 10%\n");
else if (nota >= 80)
    printf("%s", "Top 20%\n");
else if (nota > 75)
    printf("%s", "Você passou.\n");
else
    printf("%s", "Você está reprovado.\n");
```



A instrução if-else if

- Considere cuidadosamente a lógica envolvida ao desenvolver uma instrução **if-else if**.
 - O fluxo do programa ramifica para as instruções associadas à primeira expressão verdadeira e nenhuma das demais expressões será testada.
- Embora os recuos não afetem o código compilado, a lógica do **if-else** se será mais fácil de entender por um leitor quando as cláusulas **else** estiverem alinhadas.

Simplificando seleções

- Como simplificar a seguinte selecao aninhada?

```
if (condicao1) {  
    if (condicao2) {  
        if (condicao3) {  
            if (condicao4) {  
                W;  
            }  
        }  
    }  
}
```



```
if (condicao1  
    && condicao2  
    && condicao3  
    && condicao4) {  
    W;  
}
```



Exercício

- Escreva um programa que lê a idade de um usuario e em seguida diz se o usuário é ou não maior de idade.
- Escreva um programa que lê um número inteiro e diz se o número é par ou ímpar.
- Escreva um programa que lê tres numeros e em seguida imprime quantos deles são iguais.
- Escreva um programa que lê três número s inteiros e em seguida imprime os números em ordem crescente.



A instrução switch

- A instrução switch ramifica o controle do programa combinando o resultado de uma expressão com um valor de maiúsculas e minúsculas constante.
- A instrução switch geralmente fornece uma solução mais elegante para as instruções **if-else if** e aninhadas **if**.

A instrução switch

- A opção assume a forma:

```
switch (expressão) {  
    case val1:  
        instruções  
    break;  
    case val2:  
        instruções  
    break;  
    default:  
        instruções  
}
```

A instrução switch

```
int num = 2;
switch (num) {
case 1:
    printf("Um\n");
    break;
case 2:
    printf("Dois\n");
    break;
default:
    printf("Não é 1, nem 2.\n");
}
```



Quiz

- Qual opção mostra a sintaxe correta para a instrução switch?
 - a. teste de troca;
 - b. switch (teste) {}
 - c. Teste SWITCH {}



A instrução switch

- Pode haver vários casos com rótulos exclusivos.
- O caso padrão opcional é executado quando nenhuma outra correspondência é feita.



A instrução switch

- Em cada caso, é necessária uma instrução de interrupção para ramificar até o final da instrução de chave.
- Sem a instrução break, a execução do programa passa para a próxima instrução case. Isso pode ser útil quando a mesma declaração é necessária para vários casos.



A instrução switch

```
switch (num) {  
    case 1:  
    case 2:  
    case 3:  
        printf("Um, Dois, ou Três.\n");  
        break;  
    case 4:
```



A instrução switch

```
case 5:  
case 6:  
    printf("Quatro, Cinco, ou Seis.\n");  
    break;  
default:  
    printf("Maior que Seis.\n");  
}
```



Exercicio

- Escreva uma função que solicita o usuário digitar um numero de 1 à 7. Em seguida a função imprime uma mensagem de acordo com o numero digitado:
 - 1 – “Você pertence ao curso de Psicologia”
 - 2 – “Você pertence ao curso de Gastronomia”
 - 3 – “Você pertence ao curso de Farmácia”
 - 4 – “Você pertence ao curso de Enfermagem”
 - 5 – “Você pertence ao curso de Gestão da Tecnologia da Informação”
 - 6 – “Você pertence ao curso de Engenharia Elétrica”
 - 7 – “Você pertence ao curso de Ciência da Computação”
 - Qualquer outro numero - “Você não pertence a curso algum da UNIFG”

O loop while

- A instrução **while** é chamada de estrutura de loop porque executa instruções repetidamente enquanto uma expressão é verdadeira, repetindo repetidamente.
- Assume a forma:

```
while (expressão) {  
    instruções  
}
```



O loop while

- A expressão é avaliada como verdadeira ou falsa e as instruções podem ser uma única instrução ou, mais comumente, um bloco de código entre chaves `{ }`.



Exemplo

```
#include <stdio.h>

int main() {
    int contador = 1;

    while (count < 8) {
        printf("Contador = %d\n", contador);
        contador++;
    }

    return 0;
}
```





O loop while

- Um loop infinito é aquele que continua indefinidamente porque a condição do loop nunca é avaliada como falsa.
 - Isso pode causar um erro em tempo de execução.



Exercício

- Como seria um programa para calcular a média de 50 alunos da uma turma?
- Escreva um programa que calcula o produto de dois números lidos sem usar o operador de multiplicação ('*').
- Construa um algoritmo que fica lendo indefinidamente números positivos. Caso o numero lido seja igual a 0 o algoritmo pára de ler números e imprime a média dos números pares lidos anteriormente.



Exercício Desafio

- Escreva um programa que lê um número e em seguida calcula e imprime seu fatorial.



O loop do-while

- O loop **do-while** executa as instruções do loop antes de avaliar a expressão para determinar se o loop deve ser repetido.
- Assume a forma:

```
do {  
    instruções  
} while (expressão);
```



Exemplo

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

```
int main() {
```

```
    int contador = 1;
```

```
    do {
```

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

```
        contador++;
```

```
    } while (contador < 8);
```

```
    return 0;
```

```
}
```





Comandos break e continue

- A instrução **break** foi introduzida para uso na instrução **switch**.
 - Também é útil para sair imediatamente de um loop.
- Por exemplo, o programa a seguir usa uma pausa para sair de um loop **while**.



Comandos break e continue

```
int num = 5;

while (num > 0) {
    if (num == 3)
        break;
    printf("%d\n", num);
    num--;
}
```



Comandos break e continue

- Quando você deseja permanecer no loop, mas pule para a próxima iteração, use a instrução continue.

Exemplo:

```
int num = 5;
while (num > 0) {
    num--;
    if (num == 3)
        continue;

    printf("%d\n", num);
}
```




Comandos break e continue

- Embora as instruções break e continue possam ser convenientes, elas não devem substituir um algoritmo melhor.



Exercício

- Como ficaria o algoritmo para calcular a média dos 50 alunos com teste no final usando o comando do-while?



O loop for

- A instrução for é uma estrutura de loop que executa instruções um número fixo de vezes.
- Assume a forma:

```
for (valorInicial; condição; incremento) {  
    instruções;  
}
```



O loop for

- O **valorInicial** é um contador definido para um valor inicial. Esta parte do loop for é realizada apenas uma vez.
- A **condição** é uma expressão booleana que compara o contador com um valor após cada iteração do loop, interrompendo o loop quando o retorno de false.
- O **incremento** aumenta (ou diminui) o contador por um valor definido.



O loop for

```
int i;
```

```
int max = 10;
```

```
for (i = 0; i < max; i++) {  
    printf("%d\n", i);  
}
```



O loop for

- O loop for pode conter várias expressões separadas por vírgulas em cada parte.
- Por exemplo:

```
for (x = 0, y = num; x < y; i++, y--) {  
    instruções;  
}
```

O loop for

- Além disso, você pode pular o valor de inicialização, condição e / ou incremento.
- Por exemplo:

```
int i=0;  
int max = 10;  
for (; i < max; i++) {  
    printf("%d\n", i);  
}
```



O loop for

- Os loops também podem ser aninhados.
- Ao escrever um programa dessa maneira, há um loop externo e um loop interno. Para cada iteração do loop externo, o loop interno repete todo o ciclo.
- No exemplo a seguir, aninhados para loops são usados para gerar uma tabela de multiplicação.



Exemplo

```
int i, j;
int tabela = 10;
int max = 12;

for (i = 1; i <= tabela; i++) {
    for (j = 0; j <= max; j++) {
        printf("%d x %d = %d\n", i, j, i*j);
    }
    printf("\n"); /* linha em branco entre as
tabelas */
}
```



Exercício

- Como ficaria o algoritmo para calcular a media dos 5 alunos usando repeticao fixa?
- Escreva um algoritmo que lê 5 números inteiros e em seguida mostra a soma de todos os ímpares lidos.
- Altere o algoritmo anterior para que ele considere apenas a soma dos ímpares que estejam entre 100 e 200.
- Construa um algoritmo que leia um conjunto de 20 numeros inteiros e mostre qual foi o maior e o menor valor fornecido.

Comparação entre repetições

- Lacos são equivalentes

```
se (numero de vezes é conhecido) {  
    usa-se o for  
} senao {  
    se (teste precisa ser feito no inicio) {  
        usa-se while  
    } senao {  
        usa-se do-while  
    }  
}
```