



UFOP

BCC702 - Programação de Computadores II

# Estruturas Condicionais e de Repetição

Prof. José Romildo Malaquias  
Prof<sup>a</sup> Valéria de Carvalho Santos



# Relembrando...

```
#include <iostream>
using namespace std;

//programa que calcula a media de tres numeros
//inteiros
int main () {
    double num1, num2, num3, soma;
    double media;
    cin >> num1 >> num2 >> num3;
    soma = num1+num2+num3;
    media = soma/3;
    cout << media;
    return 0;
}
```

# Tipos de Dados

---

- ❑ Cada variável tem um tipo associado
  - ❑ char, int, float, double, **bool**
  - ❑ **Bool**: valores lógicos
    - verdadeiro (**true**)
    - falso (**false**)
- ❑ Em alguns contextos que esperam um valor lógico pode-se usar um valor numérico:
  - ❑ zero corresponde a falso
  - ❑ qualquer outro valor corresponde a verdadeiro

# Tipos de Dados Lógico

---

- ❑ Assume os valores verdadeiro ou falso
- ❑ Pode ser o resultado de uma expressão lógica
- ❑ Exemplo: considere **x=10** e **y=8**

| Expressão   | Resultado  |
|-------------|------------|
| $x < y$     | Falso      |
| $x \neq y$  | Verdadeiro |
| $x \leq 10$ | Verdadeiro |
| $y > (x+6)$ | Falso      |

# Operadores Relacionais

- ❑ Aplicados a valores que obedecem a uma relação de ordem
- ❑ Resultam em **verdadeiro** **falso**
- ❑ Formato: <expressão> **operador** <expressão>

| Operador | Significado    |
|----------|----------------|
| ==       | Igual a        |
| !=       | Diferente      |
| >=       | Maior ou igual |
| >        | Maior          |
| <=       | Menor ou igual |
| <        | Menor          |

# Operadores Relacionais

❏ Considere as variáveis:

- ❏ `bool b;`
- ❏ `int x=10, y=8;`

| Expressão                      | Valor de b |
|--------------------------------|------------|
| <code>b = x &gt; y;</code>     |            |
| <code>b = x != y;</code>       |            |
| <code>b = y &gt;= 100;</code>  |            |
| <code>b = x &lt; y;</code>     |            |
| <code>b = x &lt;= 10;</code>   |            |
| <code>b = y &gt; (x-6);</code> |            |

# Operadores Relacionais

❏ Considere as variáveis:

- ❏ `bool b;`
- ❏ `int x=10, y=8;`

| Expressão                      | Valor de b |
|--------------------------------|------------|
| <code>b = x &gt; y;</code>     | true       |
| <code>b = x != y;</code>       | true       |
| <code>b = y &gt;= 100;</code>  | false      |
| <code>b = x &lt; y;</code>     | false      |
| <code>b = x &lt;= 10;</code>   | true       |
| <code>b = y &gt; (x-6);</code> | true       |

# Operadores Lógicos

- ❑ Utilizados para conectar duas expressões lógicas (cujo valor) é verdadeiro ou falso;
- ❑ Formato: <expressão> **operador** <expressão>

| Operador | Significado             | Retorno                                                        |
|----------|-------------------------|----------------------------------------------------------------|
| &&       | Operador <b>E</b>       | Verdadeiro, quando todas as expressões são verdadeiras.        |
|          | Operador <b>OU</b>      | Verdadeiro, quando pelo menos uma das expressões é verdadeira. |
| !        | Operador <b>negação</b> | Verdadeiro, quando a expressão é falsa e vice-versa.           |



# Operadores Lógicos

❏ Considere as variáveis:

- ❏ `bool b;`
- ❏ `int x=10, y=8, e=44;`
- ❏ `char c = 'w';`

| Expressão                                          | Valor de b |
|----------------------------------------------------|------------|
| <code>b = (x &gt; y) &amp;&amp; (x &gt; e);</code> |            |
| <code>b = (x &gt; y)    (x &lt; e);</code>         |            |
| <code>b = (x &gt; y) &amp;&amp; (x &lt; e);</code> |            |
| <code>b = !(x &lt; y);</code>                      |            |
| <code>b = (c == 'W')    (c == 'w');</code>         |            |

# Operadores Lógicos

❏ Considere as variáveis:

- ❏ `bool b;`
- ❏ `int x=10, y=8, e=44;`
- ❏ `char c = 'w';`

| Expressão                                          | Valor de b |
|----------------------------------------------------|------------|
| <code>b = (x &gt; y) &amp;&amp; (x &gt; e);</code> | false      |
| <code>b = (x &gt; y)    (x &lt; e);</code>         | true       |
| <code>b = (x &gt; y) &amp;&amp; (x &lt; e);</code> | true       |
| <code>b = !(x &lt; y);</code>                      | true       |
| <code>b = (c == 'W')    (c == 'w');</code>         | true       |

# Tabelas-Verdade dos Operadores Lógicos

---

| A | B | A && B |
|---|---|--------|
| V | V | V      |
| V | F | F      |
| F | V | F      |
| F | F | F      |

| A | B | A    B |
|---|---|--------|
| V | V | V      |
| V | F | V      |
| F | V | V      |
| F | F | F      |

| A | !A |
|---|----|
| V | F  |
| F | V  |

# Estrutura condicional

- ❑ Classificada em três tipos:
  - ❑ Condicional simples;
  - ❑ Condicional composto;
  - ❑ Seleção entre duas ou mais sequências de comandos.

# Condicional Simples

---

- ❑ Permite a escolha do grupo de ações a ser executado quando determinada condição é satisfeita.
- ❑ Comando ***if***
  - ❑ Todo comando ***if*** requer uma **condição**
    - ❑ Verdadeira ou falsa
    - ❑ Caso seja verdadeira, executa um conjunto de instruções

# Condicionais Simples

## Sintaxe

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

# Condicionais Simples

## Exemplo

```
#include <iostream>
using namespace std;

int main () {
    int num;
    cout << "Digite um número:";
    cin >> num;
    if (num > 0)
        cout << "positivo" << endl;
    return 0;
}
```

# Condicional Composto

---

- ❑ Permite a escolha entre dois grupos de ações a serem executados, dependendo se uma determinada **condição** é satisfeita ou não.
- ❑ Comando ***if-else***
  - ❑ Utilizado quando existe um outro conjunto de instruções a ser executado quando o valor da condição é falso.



# Condicional Composto

## Exemplo

```
#include <iostream>
using namespace std;

int main () {
    int num;
    cout << "Digite um número:";
    cin >> num;
    if (num > 0)
        cout << "positivo" << endl;
    else
        cout << "não é positivo" << endl;
    return 0;
}
```

# Aninhamento de if

---

- ❑ É possível aninhar construções do tipo **if-else** em diversos níveis:
  - ❑ O **if** aninhado é simplesmente um if dentro da estrutura de um outro if mais externo
  - ❑ Deve-se ter atenção para saber exatamente a qual **if** um determinado **else** está ligado
    - ❑ A **indentação** é importante neste caso, pois ajuda a visualizar esta relação
  - ❑ Um else está ligado ao if mais próximo que o contém



```
#include <iostream>
using namespace std;
```

```
int main () {
    int num;
    cout << "Digite um número:";
    cin >> num;
    if (num > 0)
        cout << "positivo" << endl;
    else
        if (num < 0)
            cout << "negativo" << endl;
        else
            cout << "não é positivo" << endl;
    return 0;
}
```

Atenção à  
indentação

**Opção 1**

Seleção de  
duas ou mais  
opções

```
#include <iostream>
using namespace std;
```

```
int main () {
    int num;
    cout << "Digite um número:";
    cin >> num;
    if (num > 0)
        cout << "positivo" << endl;
    else if (num < 0)
        cout << "negativo" << endl;
    else
        cout << "não é positivo" << endl;
    return 0;
}
```

Atenção à  
indentação

**Opção 2**

# Exercício

---

- ❑ Escreva um programa que leia a distância em Km e a quantidade de litros de combustível consumidos por um carro em um percurso qualquer. Calcule o consumo em Km/l e escreva uma mensagem de acordo com a relação abaixo:

| Consumo (Km/l) | Mensagem        |
|----------------|-----------------|
| Menor que 8    | Venda o carro   |
| Entre 8 e 14   | Econômico       |
| Maior que 14   | Super econômico |

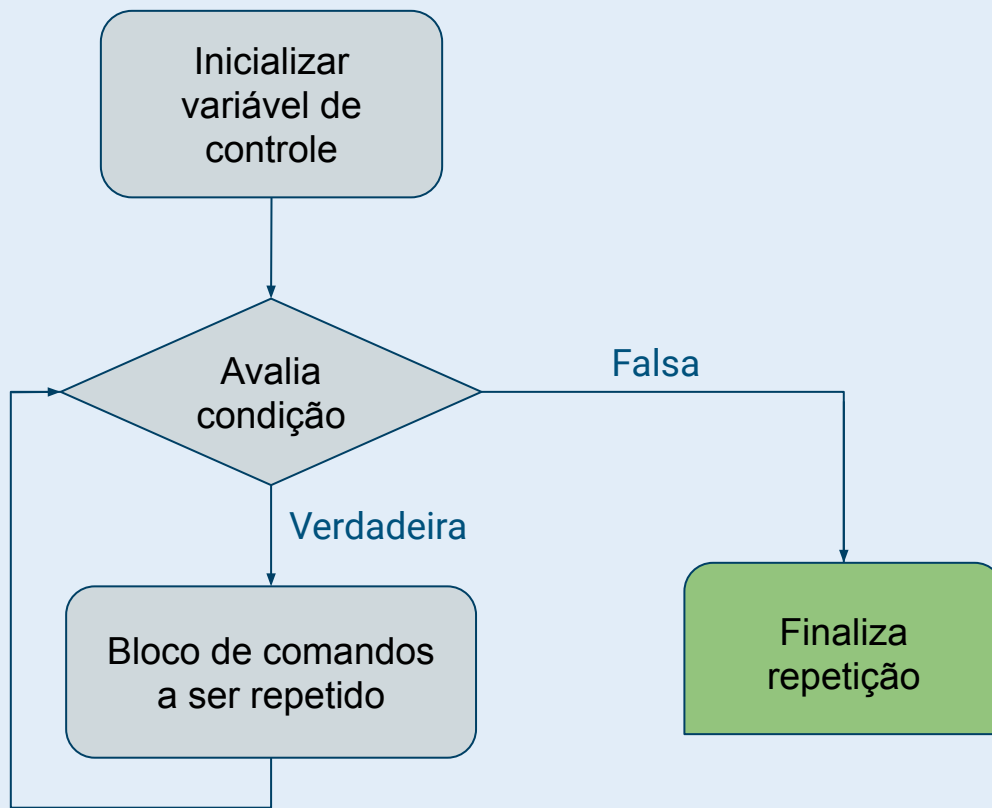
# Estruturas de Repetição

---

- ❑ Permite que um bloco de comandos seja executado mais de uma vez.
- ❑ Expressão de controle: controla quantas vezes o bloco de instruções será executado.

# Estruturas de Repetição

---



# Comando *while*

---

```
while (condição) {  
    comando 1;  
    comando 2;  
    .  
    .  
    .  
    comando n;  
}
```



# Comando *while*

---

```
while (condição) {  
    comando 1;  
    comando 2;  
    .  
    .  
    .  
    comando n;  
}
```

Bloco de comandos  
ou instruções.

**Condição ou expressão  
de controle:** enquanto a  
expressão for verdadeira,  
o bloco de comandos  
será executado.

**Iteração:** cada  
vez que o bloco  
de comandos é  
executado.

# Comando *while*

---

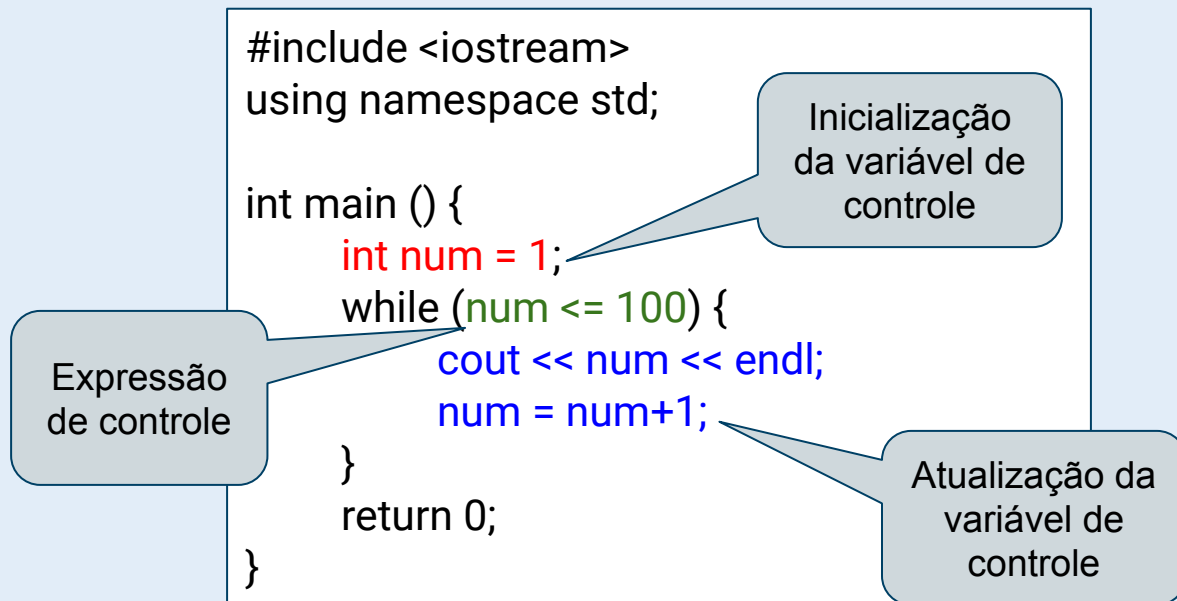
- ❏ Exemplo: Faça um programa que escreve os números inteiros de 1 a 100.

```
#include <iostream>
using namespace std;

int main () {
    int num = 1;
    while (num <= 100) {
        cout << num << endl;
        num = num+1;
    }
    return 0;
}
```

# Comando *while*

- Exemplo: Faça um programa que escreve os números inteiros de 1 a 100.



# Comando *while*

- ❏ Exemplo: Faça um programa que escreve os números inteiros de 1 a 100.

```
#include <iostream>
using namespace std;

int main () {
    int num = 1;
    while (num <= 100) {
        cout << num << endl;
        num = num+1;
    }
    return 0;
}
```

Contador

# Contador

---

- ❑ Variável utilizada para contar o número de ocorrências de um determinado evento
- ❑ Deve possuir um valor inicial
- ❑ A cada iteração, seu valor é atualizado por um valor constante

# Contador

---

- ❏ Exemplo:
  - ❏ Faça um programa que gera os números pares de 0 até um número  $n$  recebido como entrada.



```
#include <iostream>
using namespace std;
```

```
int main () {
    int n, cont;
    cout << "Digite um número:";
    cin >> n;
    cont = 0;
    while (cont <= n) {
        if (cont%2 == 0)
            cout << cont << " ";
        cont = cont+1;
    }
    return 0;
}
```

**Solução 1**

```
#include <iostream>
using namespace std;

int main () {
    int n, cont;
    cout << "Digite um número:";
    cin >> n;
    cont = 0;
    while (cont <= n) {
        cout << cont << " ";
        cont = cont+2;
    }
    return 0;
}
```

## Solução 2



# Incremento e decremento

---

- ❏ Incremento de 1 em 1

- ❏ `i++`

- *equivale a* `i = i + 1`

- ❏ Decremento de 1 em 1

- ❏ `i--`

- *equivale a* `i = i - 1`

# Acumulador

---

- ❑ Segue o mesmo princípio do contador
- ❑ Deve possuir um valor inicial
- ❑ O acumulador é atualizado por um valor de variável
- ❑ Utilizado para problemas que envolvem somatórios, produtórios, etc...

# Acumulador

---

- ❏ Exemplo:
  - ❏ Faça um programa que recebe as notas de N alunos e calcula a média das notas.



```
#include <iostream>
using namespace std;

int main () {
    int n;
    cout << "Digite a quantidade de notas: ";
    cin >> n;
    int cont = 0;
    double soma = 0;
    while (cont < n) {
        double nota;
        cout << "Digite uma nota:";
        cin >> nota;
        soma = soma + nota;
        cont = cont + 1;
    }
    double media = soma/cont;
    cout << "Media=" << media;
    return 0;
}
```

# Comando *do...while*

---

- ❏ O bloco de comandos é executado **antes** que a condição seja avaliada

```
do {  
    comando 1;  
    comando 2;  
    .  
    .  
    .  
    comando n;  
} while (condição);
```

# Comando *do...while*

❏ Exemplo:

```
#include <iostream>
using namespace std;

int main () {
    int num = 1;
    do {
        cout << num << " ";
        num = num + 1;
    } while (num <= 10);
    return 0;
}
```

# Comando *for*

---

- ❑ Adequado em situações em que o número de repetições é conhecido
- ❑ Estrutura mais compacta

# Comando *for*

☐ Sintaxe:

```
for (valor_inicial; condição; atualiza_contador) {  
    comando 1;  
    comando 2;  
    .  
    .  
    .  
    comando n;  
}
```



# Comando *for*

---

- ❏ Exemplo: Faça um programa que escreve os números inteiros de 1 a 100.

```
#include <iostream>
using namespace std;

int main () {

    for (int num = 1; num <= 100; num++)
        cout << num << endl;

    return 0;
}
```

# Estruturas de Repetição

- ❑ Exemplo: Faça um programa que escreve os números inteiros de 1 a 100.

```
#include <iostream>
using namespace std;

int main () {
    for (int num=1; num <= 100; num++)
        cout << num << endl;

    return 0;
}
```

The diagram illustrates the three components of a for loop in the provided C++ code:

- inicialização** (initialization): `int num=1`
- condição** (condition): `num <= 100`
- atualização** (update): `num++`

# Operadores Combinados com Atribuição

---

| Operador | Equivale a  |
|----------|-------------|
| $a += b$ | $a = a + b$ |
| $a -= b$ | $a = a - b$ |
| $a *= b$ | $a = a * b$ |
| $a /= b$ | $a = a / b$ |

# Exercício

---

- ❏ Faça um programa que receba a idade de pessoas, até que o valor -1 seja informado. O programa deve calcular e mostrar a quantidade de pessoas em cada faixa etária e a maior idade.

| Faixa etária | Idade            |
|--------------|------------------|
| 1            | Até 15 anos      |
| 2            | De 16 a 40 anos  |
| 3            | De 41 a 60 anos  |
| 4            | Acima de 60 anos |