BCC702 - Programação de Computadores II

Estruturas Condicionais e de Repetição

Prof José Romildo Malaquias
Profª Valéria de Carvalho Santos

UFOP

decom
departamento de computação

Relembrando...

```
#include <iostream>

using namespace std;

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

Tipos de Dados

- Cada variável tem um tipo associado

Tipo	Valores
número inteiro	short, int, long
número real	float, double
caractere	char
booleano	bool

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	
x != y	
x <= 10	
y > x+6	

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 != y	Verdadeiro
x <= 10	Verdadeiro
y > x+6	Falso

Operadores Relacionais

- Aplicados a variáveis que obedeçam a uma relação de ordem
- Retornam **true** (verdadeiro) ou **false** (falso)
- Formato: <expressão> **operador** <expressão>
- O valor inteiro **zero** é tratado como falso e os **diferentes de zero** como verdadeiro

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
b = x > y;	
b = x != y;	
b = y >= 100;	
b = x < y;	
b = x <= 10;	
b = y > x-6;	

Operadores Relacionais

- Considere as variáveis:
  - bool b;
  - int x=10, y=8;

Expressão	Valor de b
b = x > y;	true
b = x != y;	true
b = y >= 100;	false
b = x < y;	false
b = x <= 10;	true
b = y > x-6;	true

Operadores Lógicos

- Utilizados para conectar duas expressões relacionais, cujo resultado é verdadeiro ou falso;
- Formato: <expressão> **operador** <expressão>

Operador	Significado	Retorno
&&	Operador **E**	Verdadeiro, quando todas as expressões são verdadeiras.
||	Operador **OU**	Verdadeiro, quando pelo menos uma das expressões é verdadeira.
!	Operador **negação**	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
b = x > y && x > e;	
b = x > y || x < e;	
b = x > y && x < e;	
b = !(x < y);	
b = c == ‘W’ || c == ‘w’;	

Operadores Lógicos

- Considere as variáveis:
  - bool b;
  - int x=10, y=8, e=44;
  - char c = ‘w’;

Expressão	Valor de b
b = x > y && x > e;	false
b = x > y || x < e;	true
b = x > y && x < e;	true
b = !(x < y);	true
b = c == ‘W’ || c == ‘w’;	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

Condicional Simples

- Sintaxe

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

Condicional 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 declaração de um outro if mais externo
  - Deve-se ter atenção para saber exatamente a qual ***if*** um determinado else está ligado
    - A **indentação** é crucial neste caso

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;
}
```

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 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 gasolina consumidos por um carro em um percurso qualquer. Calcule o consumo em Km/l e escreva uma mensagem de acordo com a relação abaixo:

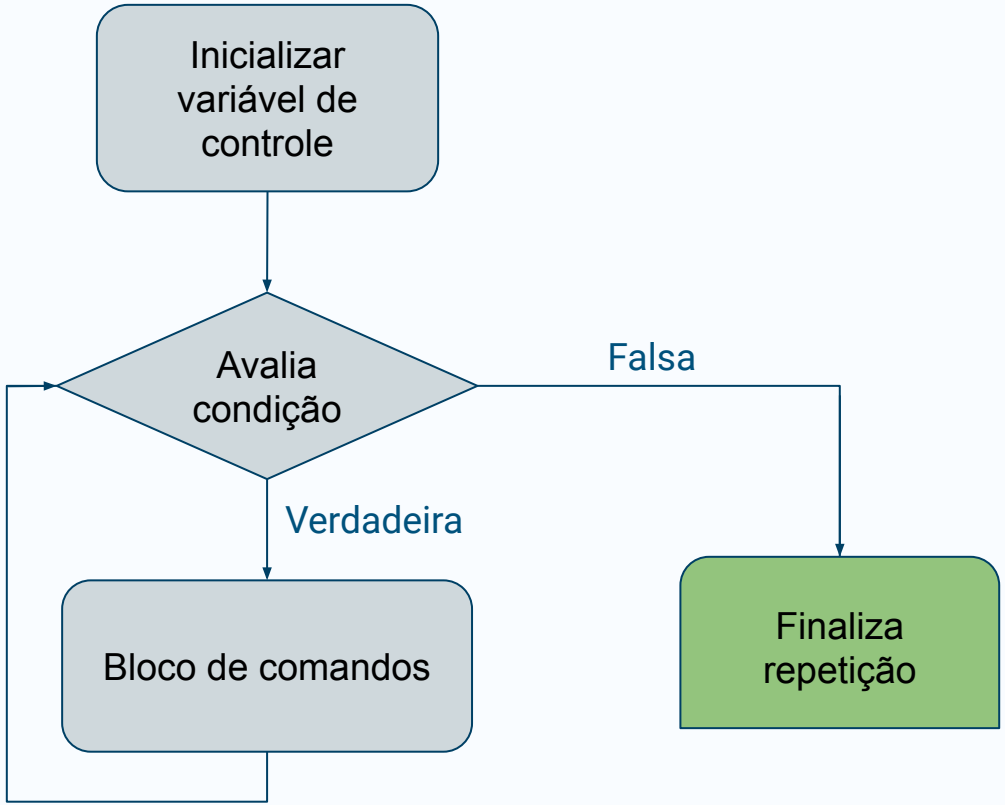
Consumo (Km/l)	Mensagem
Menor que 8	Venda o carro
Entre 8 e 14	Econômico
Maior que 14	Super econômico

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.

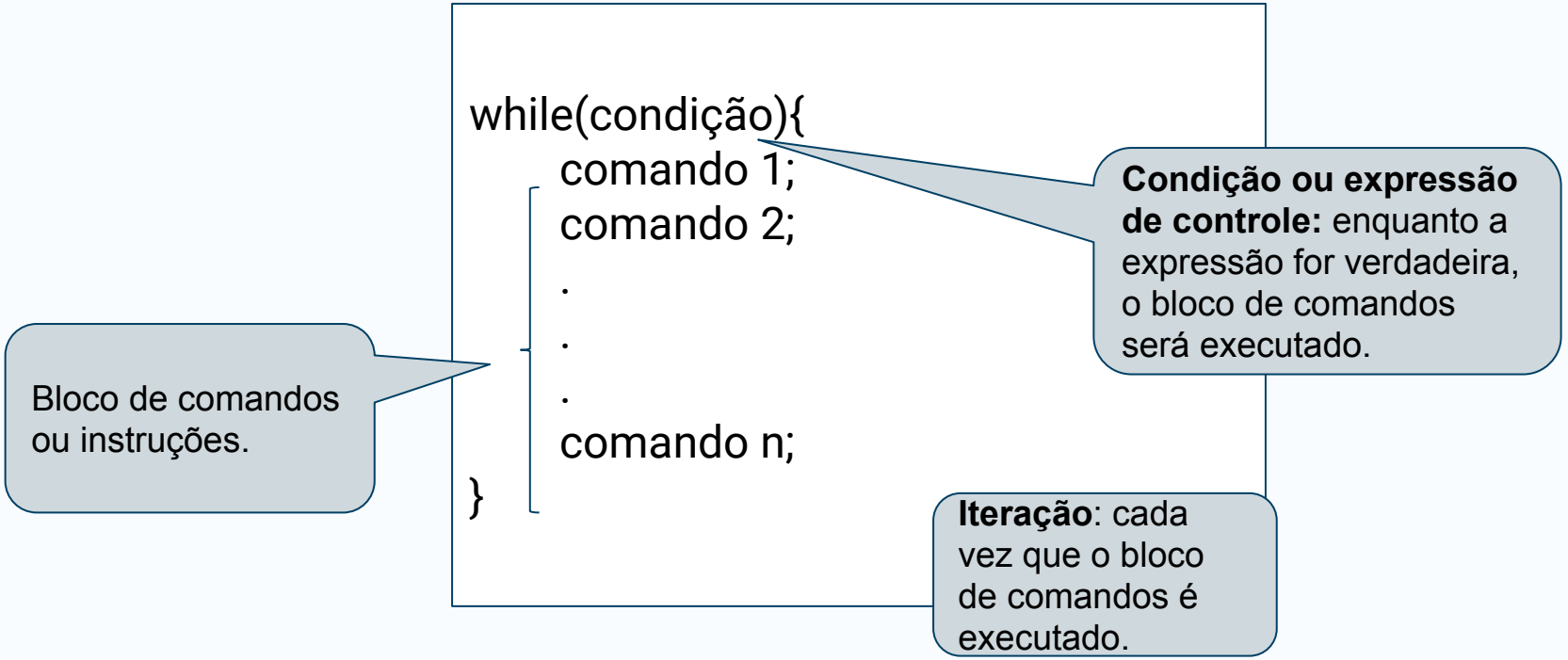
Estruturas de Repetição



Comando *while*

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

Comando *while*



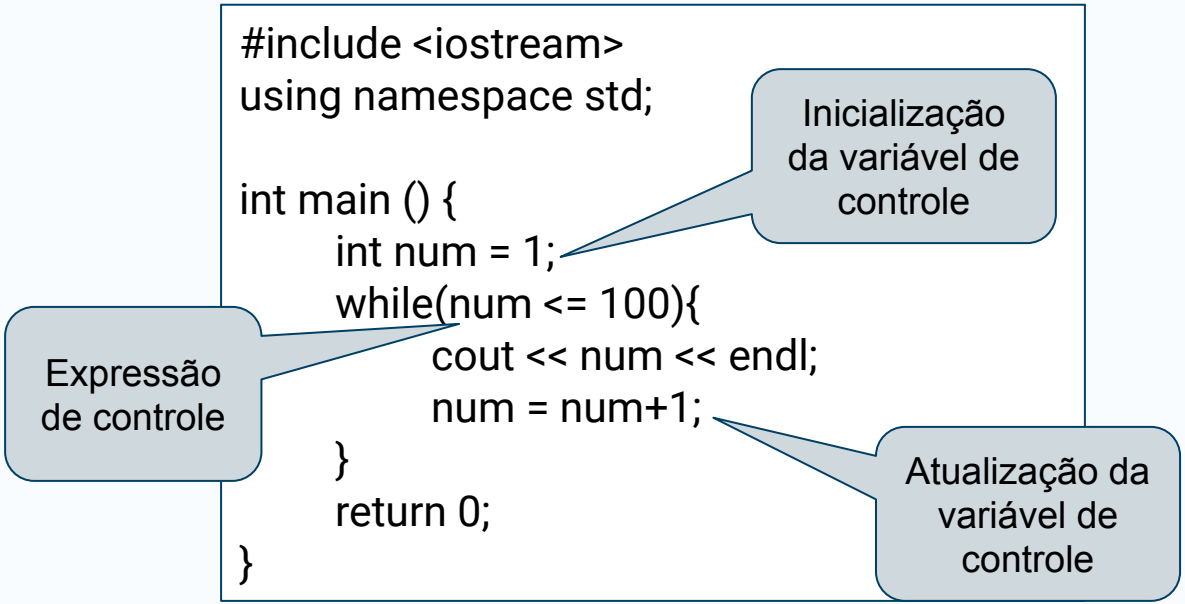
Comando *while*

- 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;
        cout << “Digite um número:”;
        cin >> n;
        int cont = 0;

        while (cont <= n){
                if (cont % 2 == 0)
                        cout << cont << “ ”;
                cont = cont+1;
        }

        return 0;
}
```

Solução 1

Solução 2

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

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

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...

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, cont=0, soma=0;
        double nota, media;
        cout << “Digite um número:”;
        cin >> n;
        while(cont < n){
                cout << “Digite uma nota:”;
                cin >> nota;
                soma = soma+nota;
                cont = cont+1;
        }
        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;
}
```

inicialização

condição

atualização

Operadores Compostos

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

#descubra

```
#include <iostream>

using namespace std;

int main(){
int x, y, c;
c = 0;
y = 0;
cout << "Digite um número: ";
cin >> x;
while(x != -1){
y += x;
c++;
cout << "Digite um número: ";
cin >> x;
}
cout << "Resposta: " << ((double)y)/c << endl;
return 0;
}
```

```
#include <iostream>

using namespace std;

int main(){
	int x, y, c;
	c = 0;
	y = 0;
	cout << "Digite um número: ";
	cin >> x;
	while(x != -1){
		y += x;
		c++;
		cout << "Digite um número: ";
		cin >> x;
	}
	cout << "Resposta: " << ((double)y)/c << endl;
	return 0;
}
```

```
#include <iostream>

using namespace std;

int main(){
	int numero, soma, cont;
	cont = 0;
	soma = 0;
	cout << "Digite um número: ";
	cin >> numero;
	while(numero != -1){
		soma += numero;
		cont++;
		cout << "Digite um número: ";
		cin >> numero;
	}
	cout << "Resposta: " << ((double)soma)/cont << endl;
	return 0;
}
```

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

Exercício

- Exemplo de execução:

```
Informe a idade da pessoa: 7
Informe a idade da pessoa: 15
Informe a idade da pessoa: 82
Informe a idade da pessoa: 64
Informe a idade da pessoa: 23
Informe a idade da pessoa: 11
Informe a idade da pessoa: -1

Quantidade de pessoas na faixa etária 1: 3
Quantidade de pessoas na faixa etária 2: 1
Quantidade de pessoas na faixa etária 3: 0
Quantidade de pessoas na faixa etária 4: 2
Maior idade: 82
```