Universidade de Brasília

# Comandos de Repetição

Computação para Engenharia — Tópico 5

Daniel Guerreiro e Silva

Departamento de Engenharia Elétrica (ENE), Faculdade de Tecnologia (FT)

# Roteiro

Comandos while e do-while

- while (condicao) { comandos }
- do { comandos } while (condicao);
- Exemplos

Comando for

- for (inicio ; condicao ; passo) { comandos ;}
- Exemplos

# Comandos while e do-while

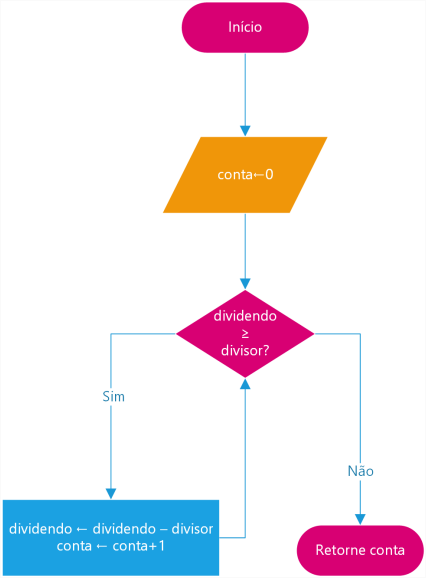
# Introdução

Até agora, vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.

- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para obter o resultado.

**Exemplo**
Calcule a divisão inteira de dois números usando apenas soma e subtração

# Algoritmo




Repare que conta resulta na divisão inteira de dividendo por divisor

# Algoritmo

Três variáveis: dividendo, conta, divisor

1. conta $\leftarrow$ 0;
2. Repita enquanto *dividendo* $\geq$ *divisor*
   2.1 dividendo $\leftarrow$ dividendo - divisor
   2.2 conta $\leftarrow$ conta + 1
3. Exiba conta

# Introdução

- Para cada comparação, fazemos:
  1. subtraimos divisor de dividendo.
  2. incrementamos conta
- Quando divisor supera dividendo, as demais comparações retornariam falso e não se executaria o bloco de comandos.

# Introdução

Seria, portanto, interessante existir um comando da linguagem que repetisse a comparação e executasse os comandos dentro enquanto a condição dada fosse verdadeira

```
/* Enquanto é verdade que dividendo>=divisor, execute */
{
  dividendo = dividendo - divisor;
  conta++;
}
```

# while (condicao) { comandos }

**Estrutura:**

```
while ( condicao )
  comando;
```

ou

```
while ( condicao ) {
  comando1;
  comando2;
  ...
  comandoN;
}
```

Enquanto a condição for verdadeira, ele executa o(s) comando(s);

# Divisão inteira

```cpp
#include <iostream>
using namespace std;
//programa que calcula a divisao inteira
int main() {
  int dividendo, divisor;
  int conta;

  cout << "Dividendo: ";
  cin >> dividendo; //entrada
  cout << "Divisor: ";
  cin >> divisor; //entrada

  conta = 0;
  while (dividendo >= divisor) { //repeticao
    conta++;
    dividendo = dividendo - divisor;
  }

  cout  << "Resultado: " << conta
        << "\nResto: " << dividendo << endl;
  return 0;
}
```

# Imprimindo os 100 primeiros números inteiros, separados por espaço

```
int i=1;
while (i<=100)
{
   cout << i << ' ';
   i++;
}
```

# Imprimindo os *n* primeiros números inteiros

```
int i=1,n;
cin >> n;
while (i<=n)
{
   cout << i << ' ';
   i++;
}
```

# Imprimindo as $n$ primeiras potências de 2

```
int i=1, n, pot=2;
cin >> n;
while (i<=n)
{
  cout << "2^" << i << " = " << pot << endl;
  i++;
  pot *= 2;
}
```

# `while (condicao) { comandos }`

1. O que acontece se a condição for falsa na primeira vez?

```
a=-1;
while (a>0) a=a-1;
```

2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a=a+1;
```

1. O que acontece se a condição for falsa na primeira vez?

```
a=-1;
while (a>0) a=a-1;
```

   R: Não executa `a=a-1` i.e. programa nunca entra na repetição (*loop*).

2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a=a+1;
```

   R: Executa `a=a+1` indefinidamente i.e. programa entra na repetição e nunca sai (*loop* infinito).

Estudando a estrutura “normal” do `while` mais a fundo:

```
while (i<=n) ← condição de repetição
{
   cout << i;
   i++; ← Comando de passo
}
```

O oposto (negação) da condição de repetição é conhecida como condição de parada:

`!(i<=n)` $\Rightarrow$ `i>n` é a condição de parada, neste exemplo.

# `while (condicao) { comandos }`

## *Loop* de fim determinado

```
cin >> preco;
while (i<=n) {
   total = total + preco;
   i++;
   cin >> preco;
}
```

## *Loop* de fim indeterminado

```
cin >> preco;
while (preco>0) {
   total = total + preco;
   cin >> preco;
}
```

**Estrutura:**

```
do
  comando;
while ( condicao );
```

ou

```
do {
  comando1;
  comando2;
  ...
  comandoN;
} while ( condicao );
```

Diferença para o comando `while`: sempre entra na primeira vez.

# Máximo Divisor Comum - MDC(x,y)

Supondo, sem perda de generalidade, que $x \geq y$, o $\text{MDC}(x, y)$ é definido da seguinte forma:

$$\text{MDC}(x, y) = \begin{cases} y & \text{caso } x \bmod y = 0 \\ \text{MDC}(y, x \bmod y) & \text{caso contrário} \end{cases}$$

# MDC(x,y)

```
/* assume-se aqui que x > y */
do
{
   r = x % y;
   x = y;
   y = r;
} while (r!=0);
```

- Repare que `r` só é calculado dentro do *loop*
- Veja exemplo em `mdc-completo.cpp`

# Soma de n valores inteiros

```
soma = 0;
while (n > 0) {
  cout << "numero a ser somado: ";
  cin >> parcela;
  soma += parcela;
  n--;
}
cout << "Soma: " << soma << endl;
```

- Veja exemplo em `soma-n.cpp`

# Soma até o

```
soma = 0;
cout << "numero a ser somado (0 para sair): ";
cin >> parcela;

while (parcela != 0) {
  soma += parcela;
  cout << "numero a ser somado (0 para sair): ";
  cin >> parcela;
}

cout << "Soma: " << soma << endl;
```

- Veja exemplo em `soma-ate-0.cpp`

# Soma até o

```
soma = 0;

do {
  cout << "numero a ser somado (0 para sair): ";
  cin >> parcela;
  soma += parcela;
} while (parcela != 0);

cout << "Soma: " << soma << endl;
```

- Veja exemplo em `soma-ate-0-do-while.cpp`

# Comando for

# Uso comum de comandos de repetição

```
i = 0;
while (i < n) {
   /* Vários comandos */
   i++;
}
```

# Problemas do `while` e do `do ... while`

1. Onde são inicializadas as variáveis usadas na condição do *loop*?
2. O passo pode estar em qualquer ponto do loop.

Apenas a condição está destacada no contexto.

**Exemplo**

```
i = 0;                          ← Inicialização de i
/* várias linhas de código */
while (i < 10) {                ← Condição de loop
   j = j * 2;
   l = j - i;
   i++;                         ← Passo
   k = i + j;
}
```

# for (inicio ; condicao ; passo) { comandos }

**Estrutura:**

```
for (inicio ; condicao ; passo)
  comando;
```

ou

```
for (inicio ; condicao ; passo) {
  comando1;
  comando2;
  ...
  comandoN;
}
```

- Início: Uma ou mais atribuições, separadas por “,”
- Condição: Idêntico ao while
- Passo: Um ou mais comandos, separados por “,”

# Exemplo

Programa com `while`

```
i = 0;
while(i < n){
    cout << i << endl;
    i++;
}
```

# Exemplo

Programa com `for`

```
for (i = 0 ; i < n ; i++){
   cout << i << endl;
}
```

```
for (inicio ; condicao ; passo) { comandos ;}
```

Quando usar `for`? Quando usar `while`?

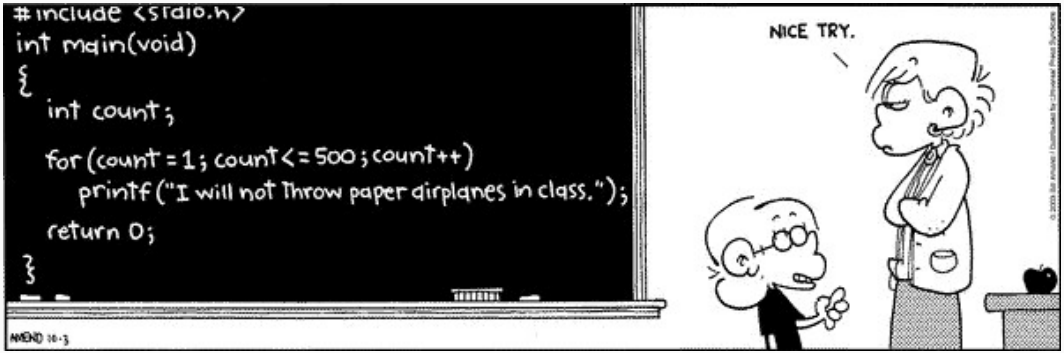
- Em termos de implementação, ambos são intercambiáveis, porém...
- Em termos de escrita de código claro, há uma diferença:
  - *Loop* de fim determinado
    "Para c=1 até c=100, faça"
  - *Loop* de fim indeterminado
    "Enquanto não digitar enter, continue lendo"

```
for (inicio ; condicao ; passo) { comandos ;}
```

Quando usar `for`? Quando usar `while`?

- Em termos de implementação, ambos são intercambiáveis, porém...
- Em termos de escrita de código claro, há uma diferença:
  - *Loop* de fim determinado
    "Para c=1 até c=100, faça" ← recomenda-se usar `for`
  - *Loop* de fim indeterminado
    "Enquanto não digitar enter, continue lendo" ← recomenda-se usar `while` ou `do-while`

# I'll not throw paper airplanes in class



# Como imprimir os $n$ primeiros números ímpares?

```
impar = 1;
for(i = 0; i < n; i++) {
  cout << impar << ' ';
  impar += 2;
}
```

- Veja exemplo em `n-impares-for.cpp`

# Como imprimir os *n* primeiros números ímpares?

```
for (i = 0, impar = 1; i < n; i++, impar += 2)
  cout << impar << ' ';
```

Inicialização e/ou atualização podem ter vários comandos separados por vírgulas!

- Veja exemplo em `n-impares2-for.cpp`

# Fatorial

```
cout << "Entre com um numero inteiro positivo: ";
cin >> n;

fat = 1;
for (i = 2; i <= n; i++)
  fat *= i;

cout  << "O fatorial de " << n
      << " e igual a " << fat
      << endl;
```

O que acontece com números muito grandes?

- Veja exemplo em `fatorial.cpp`

# Como imprimir uma linha de ’*’s usando o comando for

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

for (i = 0; i < n; i++)
    cout << "*";
cout << "\n";
```

- Veja exemplo em `linha-for.cpp`

# Como calcular o maior número entre n lidos?

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

cin >> n;
maior = INT_MIN;
for(i=1; i <= n; i++) {
   cout << "Entre com um inteiro: ";
   cin >> num;
   if (num > maior)
      maior = num;
}
cout << "O maior inteiro lido foi " << maior << endl;
```

- Veja exemplo em `maiorn.cpp`

# Até o próximo tópico...