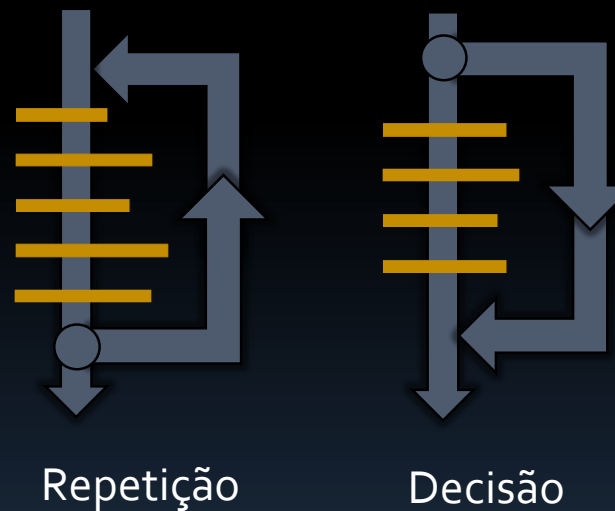


Programação de Computadores

LAÇOS WHILE E DO-WHILE

Introdução

- Computadores são máquinas capazes de realizar cálculos e **comparar valores** de forma eficiente
 - Esta é a ferramenta base para:
 - Tratar ações repetitivas
 - Tomar decisões



Expressões Relacionais

- As comparações são feitas através de **expressões relacionais**
 - Expressões que mostram a relação entre dois valores

Expressão Relacional

`for (i = 0; i < 5; i++)`

`cout << "C++ sabe repetir.\n";`

- As expressões relacionais são criadas com **operadores relacionais**

Expressões Relacionais

- Operadores Relacionais:

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

Uma expressão relacional
tem um **resultado**
booleano

Expressões Relacionais

- Alguns **exemplos**:

```
for (x=20; x > 5; x--)           // verdadeiro se x é maior que 5
for (x=1; y != x; ++x)           // verdadeiro se y não é igual a x
for (cin >> x; x == 0; cin >> x) // verdadeiro se x é 0
```

- Os operadores aritméticos têm **precedência** maior que os operadores relacionais

```
x + 3 > y - 2           // primeira expressão
(x + 3) > (y - 2)       // expressão equivalente a primeira
x + (3 > y) - 2         // expressão diferente da primeira
```

Expressões Relacionais

- Não confundir **atribuição** com **comparação**

```
material == 4    // comparação (expressão é true ou false)
material = 4     // atribuição (expressão é igual a 4)
```

- Normalmente atribuições não são interpretadas como erro:

```
// repita enquanto x for igual a 1
```

```
for (cin >> x; x == 1; cin >> x)
```

```
// teste é sempre verdadeiro
```

```
for (cin >> x; x = 1; cin >> x)
```

Expressões Relacionais

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

int main()
{
    int nota[5]= {10, 10, 10, 9, 7};

    cout << "Fazendo da maneira certa:\n";
    int i;
    for (i = 0; nota[i] == 10; i++)
        cout << "Nota " << i << " é um 10\n";

    cout << "Fazendo da forma errada:\n";
    for (i = 0; nota[i] = 10; i++)
        cout << "Nota " << i << " é um 10\n";
}
```

Expressões Relacionais

- A saída do programa:

Fazendo da maneira certa:

Nota 0 é um 10

Nota 1 é um 10

Nota 2 é um 10

Fazendo da forma errada:

Nota 0 é um 10

Nota 1 é um 10

Nota 2 é um 10

Nota 3 é um 10

Nota 4 é um 10

Nota 5 é um 10

Nota 6 é um 10

Nota 7 é um 10

Nota 8 é um 10

...

Comparando Strings

- Operadores relacionais **não devem ser usados** com strings

- Uma string é o endereço do primeiro caractere na memória:

```
char palavra[10] = "colega";
```

```
palavra == "colega" // falso  
palavra > "colega"  // verdadeiro  
palavra < "colega"  // falso
```

| | |
|----|-------------------|
| c | 0xCB19 = "colega" |
| o | 0xCB20 |
| l | 0xCB21 |
| e | 0xCB22 |
| g | 0xCB23 |
| a | 0xCB24 |
| \0 | 0xCB25 |

| | |
|----|------------------|
| c | 0xCB28 = palavra |
| o | 0xCB29 |
| l | 0xCB30 |
| e | 0xCB31 |
| g | 0xCB32 |
| a | 0xCB34 |
| \0 | 0xCB35 |

Comparando Strings

- Operadores relacionais **não devem ser usados** com strings

- A **função strcmp()** compara strings

- Recebe duas strings e retorna:

- O valor 0 se as duas strings são iguais
- <0 se a primeira é alfabeticamente[†] menor
- >0 se a primeira é alfabeticamente[†] maior

```
// resultado verdadeiro  
strcmp(palavra, "colega");
```

| | |
|----|-------------------|
| c | 0xCB19 = "colega" |
| o | 0xCB20 |
| l | 0xCB21 |
| e | 0xCB22 |
| g | 0xCB23 |
| a | 0xCB24 |
| \0 | 0xCB25 |

| | |
|----|------------------|
| c | 0xCB28 = palavra |
| o | 0xCB29 |
| l | 0xCB30 |
| e | 0xCB31 |
| g | 0xCB32 |
| a | 0xCB34 |
| \0 | 0xCB35 |

Comparando Strings

```
#include <iostream>
#include <cstring> // protótipo para strcmp()
using namespace std;

int main()
{
    char palavra[8]= "?oleque";

    for (char ch = 'a'; strcmp(palavra, "moleque"); ch++)
    {
        cout << palavra << endl;
        palavra[0] = ch;
    }

    cout << "A palavra é " << palavra << endl;
}
```

Comparando Strings

- A saída do programa:

```
?oleque  
aoleque  
boleque  
coleque  
doleque  
eoleque  
foleque  
goleque  
holeque  
ioleque  
joleque  
koleque  
loleque  
A palavra é moleque
```

Comparando Ponto Flutuantes

- Valores **ponto flutuantes** não devem ser comparados usando o **operador relacional de igualdade**

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

```
int main()
{
```

```
    float a = 6.9f;
    float b = 0.9f;
```

```
    cout << boolalpha << ( a - int(a) == b ) << endl;
}
```

6.9000001

0.89999998

A saída do
programa:

false

0.9000001

0.89999998

Comparando Ponto Flutuantes

- Soluções para **comparar pontos flutuantes**:
 - Ler os números como **strings**
 - Comparar as strings usando **strcmp()**
 - Converter as strings para float usando **atof()**
 - Calcular a **diferença** entre os números
 - Eles são iguais se a diferença estiver dentro de um fator de erro

```
bool fltcmp(float a, float b)
{
    const float erro = 0.001f;
    if (abs(a - b) < erro) return true;
    else return false;
}
```

Laço while

- O **laço while** é um laço de repetição sem as partes de inicialização e atualização

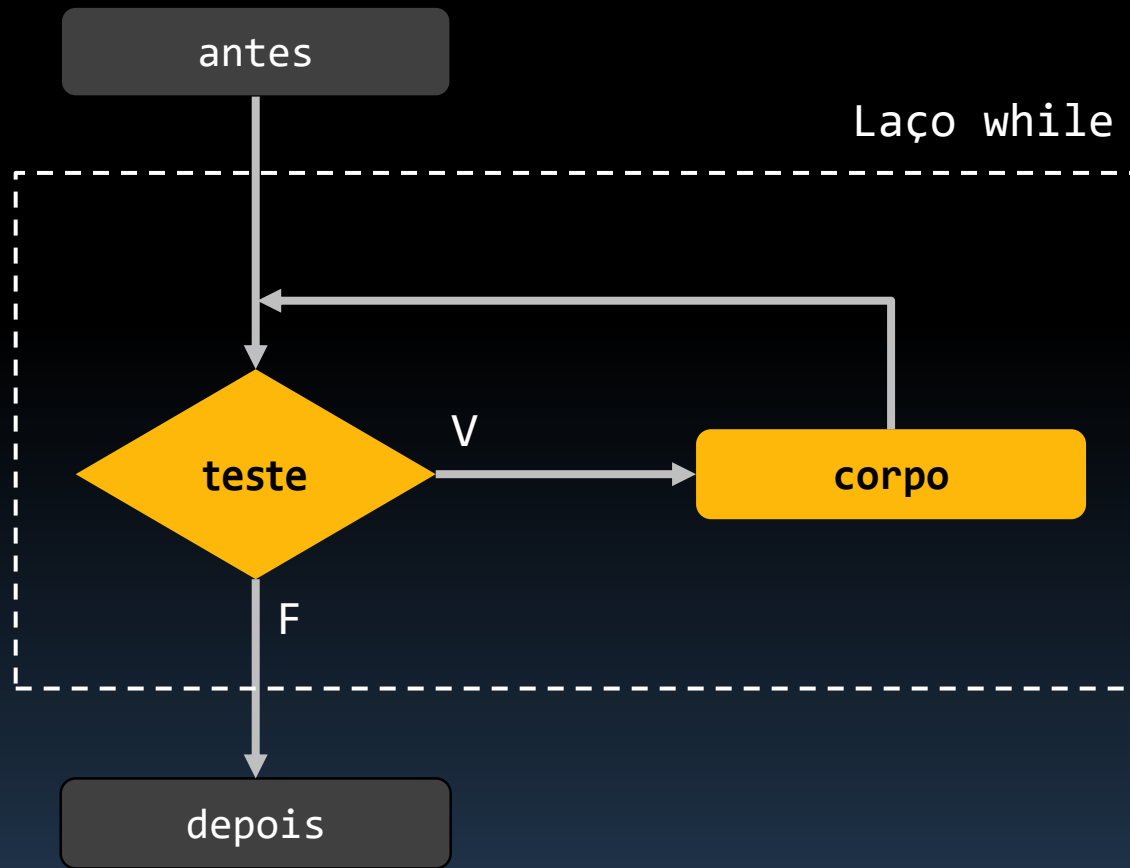
Repete enquanto o teste é verdadeiro

while (teste)

corpo

Instrução (ou bloco de instruções) a ser repetida

Laço while



```
antes;  
while (teste)  
    corpo;  
depois;
```

Laço while

- A **repetição** continua até que o teste seja falso
 - Para **encerrar o laço** é preciso fazer algo no corpo que torne o teste falso

```
while (teste)  
    corpo
```

- O while **avalia a expressão de teste na entrada** do laço
 - Se o teste é falso na primeira passagem, o corpo do laço nunca é executado

Laço while

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

int main()
{
    cout << "Entre com números para somar (0 para sair):\n";
    int num;
    int soma = 0;

    cin >> num;
    while (num != 0)
    {
        soma += num;    // soma = soma + num;
        cin >> num;    // lê o próximo número
    }
    cout << "A soma é " << soma << "\n";
}
```

Laço while

- A saída do programa:

Entre com números para somar (0 para sair):

34

20

15

41

0

A soma é 110

- Se o primeiro valor for zero o laço não repete nenhuma vez

```
cin >> num;
```

```
while (num != 0)
```

```
...
```

Laço while

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

int main()
{
    cout << "Digite seu primeiro nome: ";
    char nome[20];
    cin >> nome;

    cout << "Seu nome na vertical e em códigos ASCII:\n";
    int i = 0;                // começa no início da string
    while (nome[i] != '\0')    // processa até o fim da string
    {
        cout << nome[i] << ": " << int(nome[i]) << endl;
        i++;
    }
}
```

Laço while

- A saída do programa:

```
Digite seu primeiro nome: Joao  
Seu nome na vertical e em códigos ASCII:  
J: 74  
o: 111  
a: 97  
o: 111
```

- As duas formas abaixo são equivalentes:

- O caractere nulo tem código zero (falso)

```
while (nome[i] != '\0')  
while (nome[i])
```

for *versus* while

- Os laços **for** e **while** são **equivalentes**

- O laço for

```
for (inicialização; teste; atualização)
{
    corpo;
}
```

- Pode ser reescrito como um while:

```
inicialização;
while (teste)
{
    corpo;
    atualização;
}
```

for *versus* while

- Os laços **for** e **while** são **equivalentes**

- O laço while

```
while (teste)
{
    corpo;
}
```

- Pode ser reescrito como um for:

```
for ( ; teste; )
{
    corpo;
}
```

for *versus* while

- A **escolha** é uma questão de preferência
 - Normalmente utiliza-se o **laço for** com **contadores** porque a inicialização, teste e atualização ficam concentrados

```
for (inicialização; teste; atualização)
    corpo
```

- Normalmente utiliza-se o **laço while** quando **não se sabe** antecipadamente **quantas vezes** o laço será executado

```
while (teste)
    corpo
```

Cuidados com o Laço while

- **Indentação** não define um bloco

```
i = 0;
while (nome[i] != '\0')
    cout << nome[i] << endl;
    i++;    // instrução fora do laço, nunca é executada
cout << "Pronto\n";
```

- O laço while não contém **ponto e vírgula**

```
i = 0;
while (nome[i] != '\0'); // problema com ponto e vírgula
{
    cout << nome[i] << endl;
    i++;
}
cout << "Pronto\n";
```

Laço do-while

- O laço **do-while** é diferente dos demais
 - O teste é feito apenas na saída do laço

Instrução (ou bloco de instruções) a ser repetida

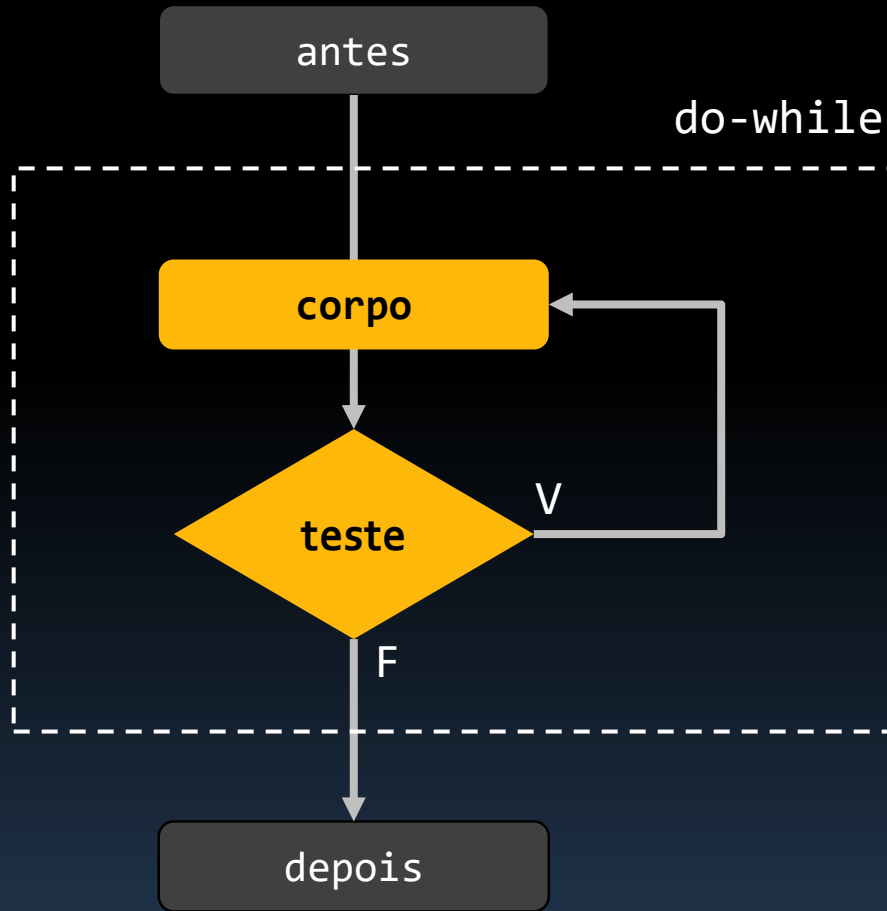


```
do
    corpo
while (teste);
```

The diagram shows a code block with the text 'do', 'corpo', and 'while (teste);'. A vertical line points from the text 'Instrução (ou bloco de instruções) a ser repetida' to the word 'corpo'. Another vertical line points from the text 'Repete enquanto o teste é verdadeiro' to the expression '(teste)'.

Repete enquanto o teste é verdadeiro

Laço do-while



```
antes;  
do  
    corpo;  
while (teste);  
depois;
```

Laço do-while

- O corpo do laço é **executado pelo menos uma vez**
 - Ele pode ser um bloco de instruções

```
do
{
    instrução 1;
    instrução 2;
    ...
    instrução n;
}
while (teste);
```

} O corpo é um bloco de instruções

Laço do-while

- O laço do-while **termina com ponto e vírgula**
 - O ponto e vírgula é necessário porque um while **pode ser aninhado** dentro de um do-while:

```
int x = 10;
int y = 10;

do
    while(x > 0)
        x--;
while(x = --y);
```

Laço do-while

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

int main()
{
    int n;
    cout << "Digite um número entre 0-10\n";

    do
    {
        cin >> n;
    }
    while (n != 7);

    cout << "Sim, 7 é meu número favorito.\n";
}
```

Laço do-while

- A saída do programa:

```
Digite um número entre 0-10:
```

```
4
```

```
5
```

```
9
```

```
3
```

```
1
```

```
7
```

```
Sim, 7 é meu número favorito.
```

Resumo

- C++ oferece **três variedades de laços**:
 - for** (inicialização; teste; atualização) corpo;
 - while** (teste) corpo;
 - do** corpo; **while** (teste);
- Um laço repete um conjunto de instruções
 - A **repetição continua** enquanto o teste é **verdadeiro** (ou não nulo)
 - A **repetição encerra** quando o teste é **falso** (ou equivalente ao valor zero)

Resumo

- **Expressões relacionais** são comumente usadas para compor os testes de parada dos laços
 - Expressões relacionais resultam em um valor booleano **true** ou **false**
 - Os **operadores relacionais** são **<, <=, ==, >=, >, !=**
 - Não se deve usar o operador **==** com **strings** ou **ponto-flutuantes**