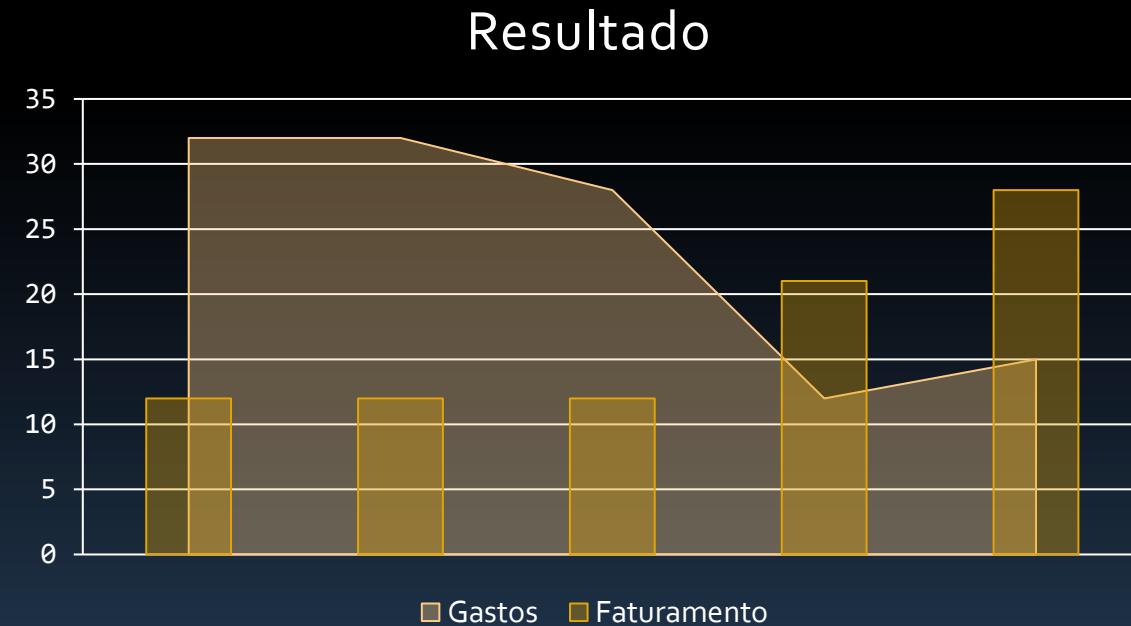


Programação de Computadores

LAÇO DE REPETIÇÃO FOR

Introdução

- Computadores fazem mais que apenas **armazenar dados**, eles também:
 - Calculam
 - Analisam
 - Rearranjam
 - Modificam
 - Sintetizam
 - ... **Manipulam dados**



Introdução

- Para fazer tais manipulações as linguagens de programação precisam de **ferramentas** para:

Tratar
Ações Repetitivas

Tomar
Decisões

for

if

while

if else

do while

switch

Introdução

- Estas estruturas são chamadas de **estruturas de controle**
 - Controlam o **fluxo de execução** do programa



Sequencial



Repetição



Decisão

Introdução

- Estas estruturas de controle fazem uso frequente de:
 - Expressões Relacionais
 - > (maior), < (menor)
 - >= (maior ou igual), <= (menor ou igual)
 - == (igual), != (diferente)
 - Expressões Lógicas
 - && (and), || (or)
 - ! (not)

Laço for

- Em muitas circunstâncias é preciso executar uma mesma tarefa **repetidas vezes**:
 - Ler números para um vetor
 - Exibir uma linha de caracteres
- O **laço for** permite executar um conjunto de instruções um número fixo de vezes:
 - Ler **50 elementos** para um vetor
 - Exibir **20 caracteres iguais**

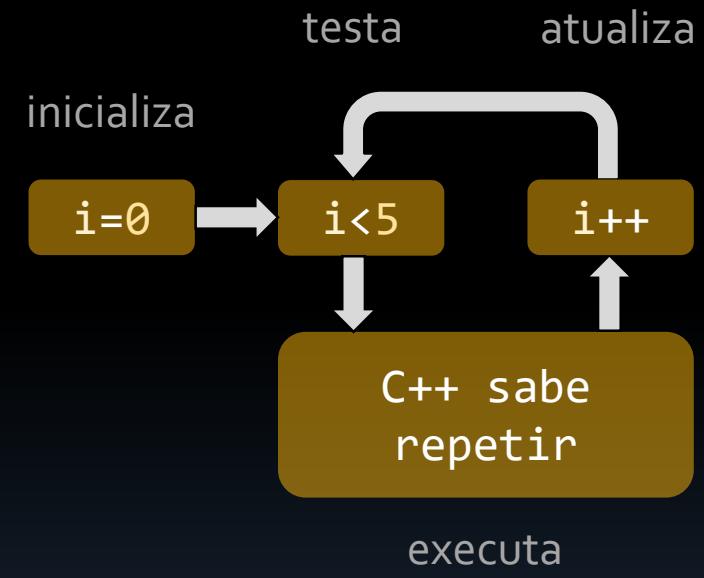
Laço for

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

int main()
{
    // cria um contador
    int i;

    // inicializa; testa; atualiza
    for (i = 0; i < 5; i++)
        cout << "C++ sabe repetir." << endl;
    cout << "C++ sabe quando parar." << endl;

    return 0;
}
```



Laço for

Laço for

- Saída do Programa:

```
C++ sabe repetir.  
C++ sabe quando parar.
```

- O operador **++** incrementa seu operando em uma unidade:

```
i++; // i = i + 1;
```

Laço for

- As partes de um laço for

Testa se o contador chegou a um limite

Inicializa uma variável contador

Atualiza o contador

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

corpo

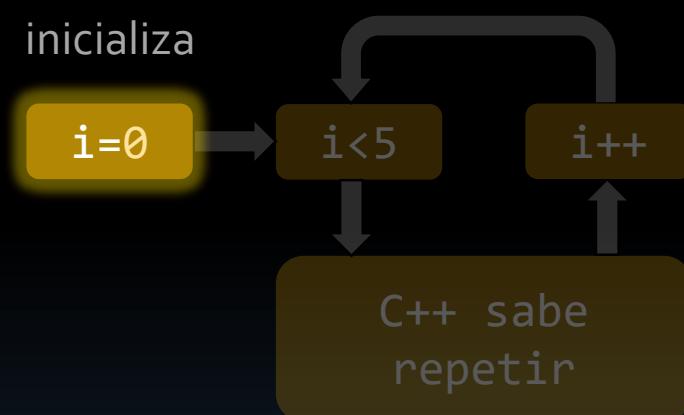
```
for (i = 0; i < 5; i++)  
    cout << "C++ ";
```

Instrução a ser repetida

Laço for

- Inicialização:
 - É realizada apenas uma vez
 - Geralmente usada para definir o valor inicial de uma variável (contador)

```
int i;    // utilizado como contador  
for ( i = 0 ; i < 5; i++)  
    cout << "C++ sabe repetir.\n";
```

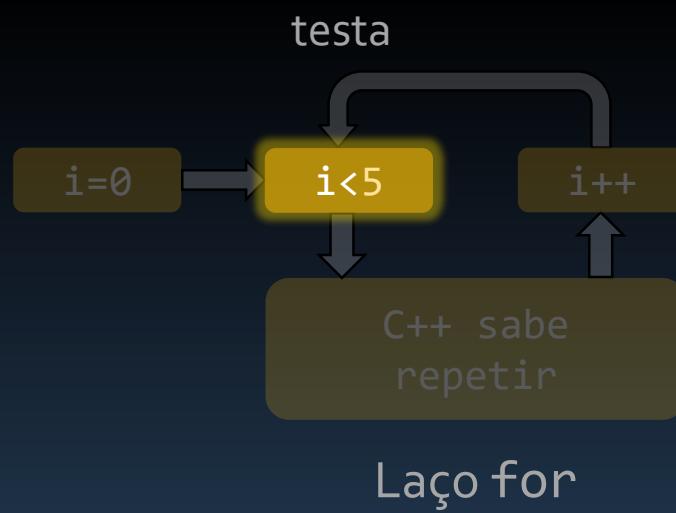


Laço for

Laço for

- Teste:
 - Determina se o corpo do laço é executado
 - Tipicamente compara dois valores e o resultado é booleano
 - C++ converte automaticamente qualquer outra expressão para um valor booleano

```
int i;  
for (i = 0; i < 5 ; i++)  
    cout << "C++ sabe repetir.\n";
```



Laço for

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

int main()
{
    cout << "Digite o valor do contador: ";
    int limite;
    cin >> limite;

    int i;
    for (i = limite; i ; i--) // encerra quando i é 0
        cout << "i = " << i << "\n";

    cout << "Finalizando agora que i = " << i << endl;

    return 0;
}
```

Laço for

- Saída do Programa:

```
Digite o valor do contador: 4
```

```
i = 4  
i = 3  
i = 2  
i = 1
```

```
Finalizando agora que i = 0
```

- O for é um laço que testa a condição na entrada, antes de executar o corpo do laço

```
for (i = 0; i < 0; i++)  
    cout << "C++ sabe repetir?"; // não será executado
```

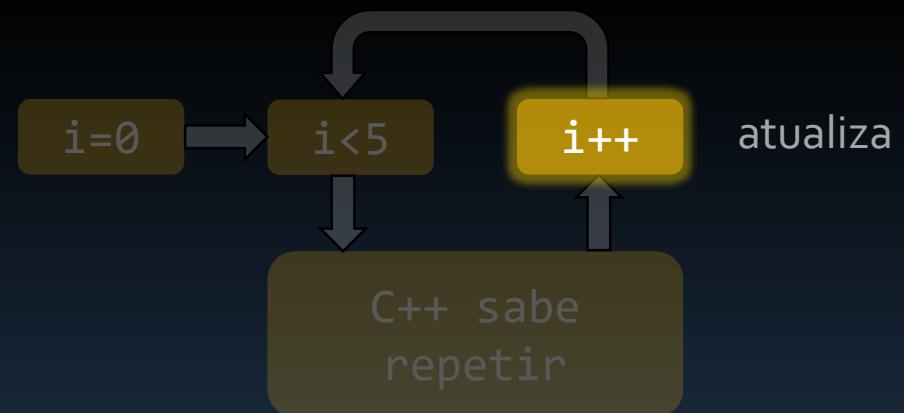
Laço for

- Atualização:

- É realizada após a execução do corpo do laço
- Normalmente é usada para incrementar ou decrementar uma variável (contador)
- Pode ser qualquer expressão

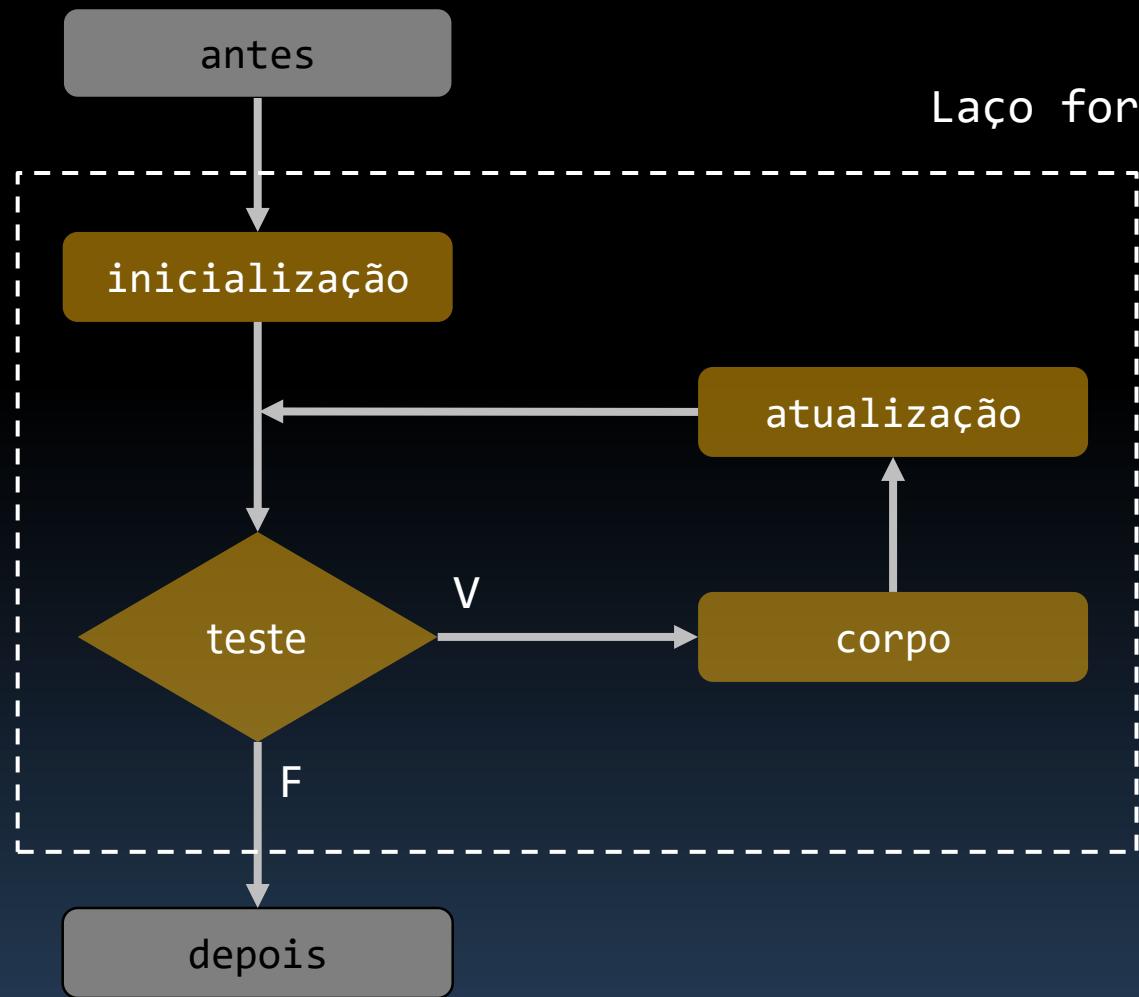
```
for (i = 0; i < 5; i++)  
    cout << "C++ sabe repetir.\n";
```

```
for (i = 5; i > 0; i--)  
    cout << "C++ sabe repetir.\n";
```



Laço for

Laço for



```
antes;  
for (inicializa; testa; atualiza)  
    corpo;  
depois;
```

Expressões e Instruções

- Um laço for usa três expressões:
 - Inicialização
 - Teste
 - Atualização
- Uma expressão é qualquer valor ou qualquer combinação válida de valores (constantes e variáveis) e operadores
 - 10 é uma expressão com o valor 10
 - $28+20$ é uma expressão com o valor 48

Expressões e Instruções

- Em C++, toda expressão tem um valor
 - Muitas vezes o valor é óbvio: $(2 * 25)$ tem valor 50
 - Outras vezes nem tanto: $(x = 20)$ tem valor 20
- Uma atribuição tem o valor do seu operando esquerdo

```
empregados = (copeiros = 4) + 3;      x = y = z = 0;  
                                         ^  
                                         4  
                                         ^  
                                         4 + 3  
                                         ^  
                                         7  
  
                                         ^  
                                         0  
                                         ^  
                                         0  
                                         ^  
                                         0
```

Expressões e Instruções

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

int main()
{
    int x;

    cout << "A expressão x = 100 tem o valor ";
    cout << (x = 100) << endl;
    cout << "Agora x = " << x << endl;
    cout << "A expressão x < 3 tem o valor ";
    cout << (x < 3) << endl;
    cout << "A expressão x > 3 tem o valor ";
    cout << (x > 3) << endl;

    // cout.setf(ios_base::boolalpha);
    cout << boolalpha;
    cout << "A expressão x < 3 tem o valor ";
    cout << (x < 3) << endl;
    cout << "A expressão x > 3 tem o valor ";
    cout << (x > 3) << endl;
}
```

Expressões e Instruções

- Saída do Programa:

A expressão `x = 100` tem o valor `100`

Agora `x = 100`

A expressão `x < 3` tem o valor `0`

A expressão `x > 3` tem o valor `1`

A expressão `x < 3` tem o valor `false`

A expressão `x > 3` tem o valor `true`

- O `cout` converte valores booleanos para inteiros
- A instrução `cout.setf(ios_base::boolalpha)` configura `cout` para exibir as palavras `true` e `false`

Expressões e Instruções

- A avaliação de algumas expressões possuem efeitos colaterais (modificam variáveis)

```
// C++ é obrigado a atribuir o valor 100 para x  
cout << x = 100;
```

- Nem todas as expressões tem efeitos colaterais

```
// calcula um novo valor  
// mas não altera o valor  
// da variável x  
cout << x + 15;
```

Expressões e Instruções

- Para passar de uma expressão para uma instrução basta acrescentar ponto e vírgula

```
idade = 100 // uma expressão
```

```
idade = 100; // uma instrução de atribuição
```

- É possível transformar qualquer expressão em uma instrução acrescentando ponto e vírgula

```
// válido mas sem sentido  
solteiros + 6;
```

Expressões e Instruções

- Para passar de uma **expressão** para uma **instrução** basta acrescentar um ponto e vírgula,
mas e o contrário é **verdade**?

```
int total = 1; // uma instrução  
int total = 1 // não é uma expressão
```

```
// por isso a instrução abaixo não é válida  
resultado = int total = 1 * 1000;
```

Expressão?

Não

Expressões e Instruções

- C++ possui um recurso que não está presente em C

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

não é expressão

- A declaração de uma variável não é uma expressão, mas esta regra foi flexibilizada dentro do for pela praticidade

```
for (int i=0; i<5; i++)  
    cout << "C++ sabe repetir.\n";  
    cout << i << endl; // cuidado, i não está mais definido
```

Laços com Vetores

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

const int TamVet = 16;

int main()
{
    long long fatorial[TamVet];

    fatorial[1] = fatorial[0] = 1LL;

    for (int i = 2; i < TamVet; i++)
        fatorial[i] = i * fatorial[i-1];

    for (int i = 0; i < TamVet; i++)
        cout << i << "!" = " " << fatorial[i] << endl;

    return 0;
}
```

0	1	0xCB20 = fatorial
1	1	0xCB28
2	2	0xCB30
3	6	0xCB38
4	24	0xCB40
...		
15	1307674368000	0xCB50

Laços com Vetores

- A saída do programa:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
```

Laços em Funções

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

void Crescente(int ini, int fim); // protótipo da função

int main()
{
    Crescente(3, 9);           // chamada da função
    return 0;
}

void Crescente(int ini, int fim) // definição da função
{
    for (int i = ini; i <= fim; i++)
        cout << i << " ";
    cout << endl;
}
```

Laços em Funções

- A saída do programa:

3 4 5 6 7 8 9

- Os limites do laço são definidos pelos argumentos da função

```
void Crescente(int ini, int fim)
{
    for (int i = ini; i <= fim; i++)
        cout << i << " ";
    cout << endl;
}
```

Laços em Funções

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

void Inverte(int[], int); // protótipo da função

int main()
{
    int nums[5] = { 40, 50, 60, 70, 80 };
    Inverte(nums, 5); // chamada da função
    return 0;
}

void Inverte(int vet[], int tam) // definição da função
{
    for (int i = tam-1; i >= 0; i--)
        cout << vet[i] << " ";
    cout << endl;
}
```

0	40	0xCB20	nums	←
1	50	0xCB24		
2	60	0xCB28		
3	70	0xCB2C		
4	80	0xCB30		
		0xCB34		
	0xCB20	0xCB38	vet	—
	5	0xCB3C	tam	
	-1	0xCB40	i	

Laços em Funções

- A saída do programa:

80 70 60 50 40

- O **tamanho do vetor** deve ser passado para a função
 - O parâmetro pode usar a notação de vetor ou de ponteiro

```
void Inverte(int * vet, int tam)
{
    for (int i = tam-1; i >= 0; i--)
        cout << vet[i] << " ";
    cout << endl;
}
```

Resumo

- O laço for é utilizado para realizar tarefas repetitivas
 - Ideal para quando se conhece o número de repetições

```
for (i = 0; i < 5; i++)
    cout << "C++ sabe repetir?";
```

- Uma de suas principais aplicações é o processamento dos elementos de um vetor

```
for (int i = 0; i < TamVet; i++)
    cout << vet[i] << endl;
```