O homem é o único ser que tem o poder de modificar as coisas.

Raul Santos Seixas

1 O uso do *for* - outra estrutura de repetição

A estrutura de repetição *for*, como *while* e *do..while*, permite a repetição de um bloco de instruções (comandos). Esta estrutura de repetição idêntica às estruturas de repetição vistas anteriormente e recebe o nome de repetição com número pré-definido de ciclos.

A diferença é que é logo no início, é especificado o número de ciclos (ou iterações) que serão efetuados, isto é, o número de vezes que a ação será processada (figura 1)

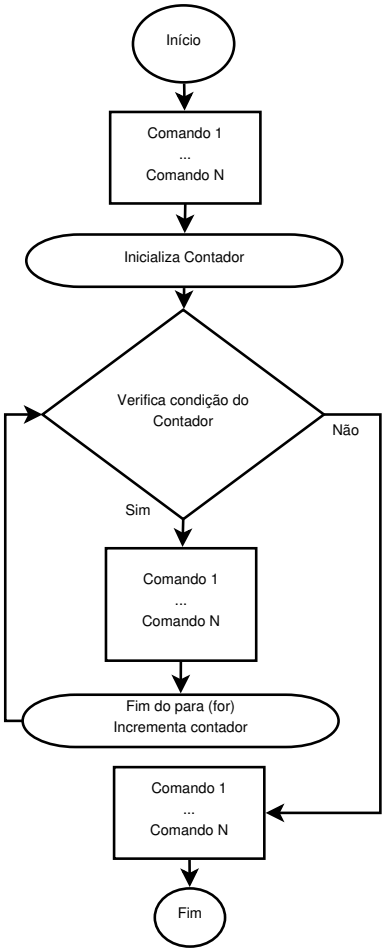

Figura 1: Fluxograma de Repetição com número pré-definidos de ciclos.

A estrutura de repetição *for* é garantida, pois a quantidade de repetições é pré-definidas. Um exemplo de uso do *for* em programação de jogos, seria para percorrer todo o inventário de um personagem (típicos em jogos de RPG).

A sintaxe de uso de *for* é simples, na sua declaração, o comando for determina três áreas distintas:

```
for( inicialização;
     condição de parada;
     incremento)
{
   bloco de comandos;
}
```

- A primeira área de comando são os comandos que serão executados inicialmente. Deve-se colocar nesta área comandos de inicialização de variáveis.
- A segunda área é a de teste. A cada interação, as condições colocadas são testadas e caso sejam verdadeiras segue-se com a execução do bloco de comandos.
- A última área possui comandos que serão executados ao final da interação. Geralmente são colocados nesta área os comandos de incrementos de variáveis.

Pode-se omitir os comandos da área de inicialização e de incremento, bastando-se colocar o ponto e virgula.

O termo iteração é utilizado quando nos referimos à repetição de uma ou mais ações. Portanto, quando dizemos que "algo deve ser executado a cada iteração de um *loop*"estamos querendo dizer que "a cada rodada de um *loop* algo deve ser executado".

1.1 Uso de contadores - iniciando os diversos tipos de contagem

O programa 1 demonstra vários tipos de contagem: para frente (crescente), para trás (decrescente), alternado (2 em 2) e linhas e colunas e a figura 2 apresenta graficamente:

Programa 1: Várias contagens diferentes.

```
// Introdução ao contador
// programa_001.cpp
#include "biblaureano.h"

int main()
{
   int contador;

   cout << "Contando para frente:";
   for( contador = 1; contador <= 9; ++contador)
   {
      cout << contador << " ";
   }
   cout << endl; //quebra de linha

   cout << "Contando para trás:";
   for( contador = 9; contador >= 1; --contador)
   {
      cout << contador << " ";
   }
   cout << endl; //quebra de linha

   cout << "Contando de 2 em 2:";
   for( contador = 2; contador <= 20; contador+=2)
   {
```

```
26        cout << contador << " ";
27     }
28     cout << endl; //quebra de linha
29
30     cout << "Contando em linhas e colunas:" << endl;
31
32     const int LINHAS = 5,
33               COLUNAS = 3;
34     for( int linhas = 1; linhas <= LINHAS; ++linhas)
35     {
36        for( int colunas = 1; colunas <= COLUNAS; ++colunas)
37        {
38           cout << linhas << "," << colunas << " ";
39        }
40        cout << endl;//quebra de linha
41     }
42
43     cout << "Contando, mas omitindo alguns parâmetros:";
44     contador = 1;
45     for( ; contador <= 5; )
46     {
47        cout << contador << " ";
48        ++contador;
49     }
50     cout << endl; //quebra de linha
51
52     cout << "Game over!!" << endl;
53     return 0;
54  }
```

1.2 Entendo o programa

Na primeira parte do programa, ocorre a contagem crescente:

```
1     cout << "Contado para frente:";
2     for( contador = 1; contador <= 9; ++contador)
3     {
4        cout << contador << " ";
5     }
6     cout << endl; //quebra de linha
```

A contagem começa em 1 (*contador = 1*), ocorre até o valor 9 (*contador <= 9*) e incrementa de 1 em 1 (*++contador*).

Repare que existe um ponto e vírgula (;) separando cada parte das instruções.

Na segunda parte do programa, ocorre a contagem decrescente:

```
1     cout << "Contado para trás:";
2     for( contador = 9; contador >= 1; --contador)
3     {
4        cout << contador << " ";
5     }
6     cout << endl; //quebra de linha
```

A contagem começa em 9 (*contador = 9*), ocorre até o valor 1 (*contador >= 1*) e o decremento ocorre de 1 em 1 (*–contador*).

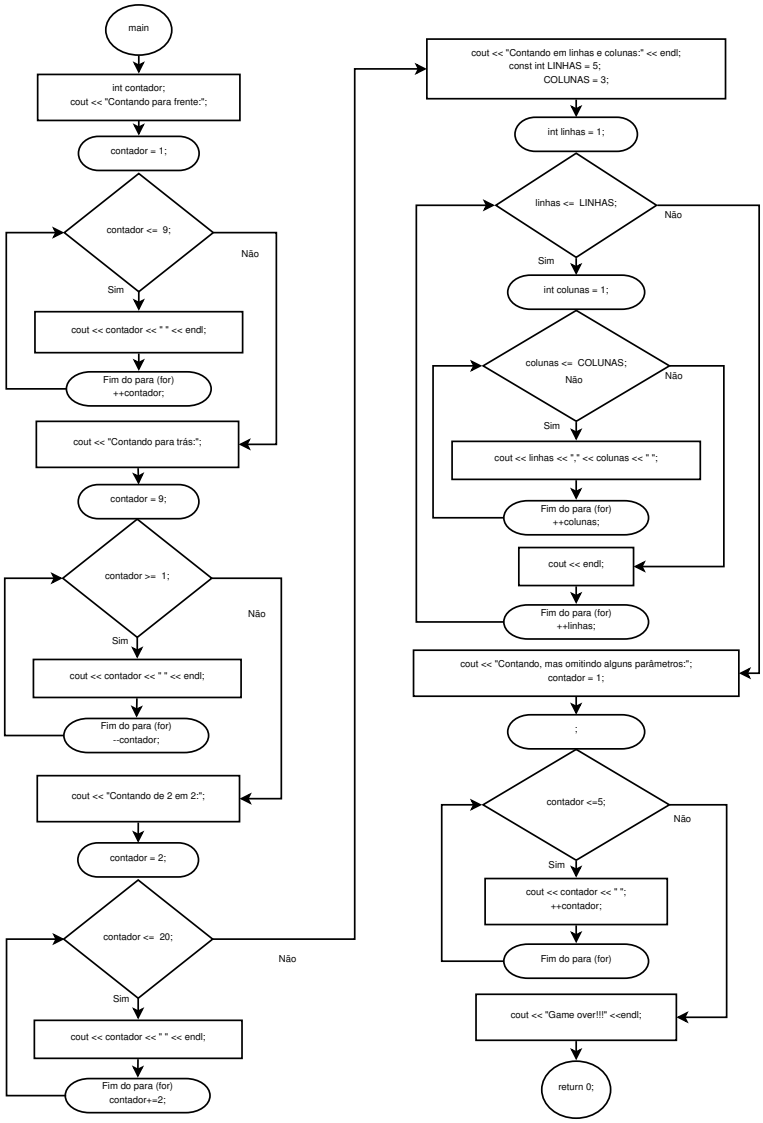

Figura 2: Fluxograma do programa 1.

Ao utilizar o comando *for* tenha cuidado para iniciar o contador com o valor inicial correto e o incremento/decremento adequado.
Evite situações como esta:

```
1  for( contador = 1; contador <= 5; --contador)
2  {
3      cout << contador << " ";
4  }
```

Este é o típico caso de ***loop* infinito**, pois o valor inicial (1) é menor que o valor estipulado na condição ($<= 5$) e ocorre o decremento da variável, ou seja, a variável *sempre* será menor ou igual a 5.

Além de fazer o incremento de 1 em 1, é possível realizar o incremento de 2 em 2:

```
1   cout << "Contando de 2 em 2:";
2   for( contador = 2; contador <= 20; contador+=2)
3   {
4      cout << contador << " ";
5   }
6   cout << endl; //quebra de linha
```

Com a utilização do operador += é possível incrementar com qualquer valor desejado. Claro que também é possível utilizar o operador -=.

Como todos os outros comandos vistos até o momento (*if*, *while*, etc), é possível combinar a utilização do comando *for*. Observe o trecho do programa:

```
1   cout << "Contando em linhas e colunas:" << endl;
2
3   const int LINHAS = 5,
4             COLUNAS = 3;
5   for( int linhas = 1; linhas <= LINHAS; ++linhas)
6   {
7      for( int colunas = 1; colunas <= COLUNAS; ++colunas)
8      {
9         cout << linhas << "," << colunas << " ";
10     }
11     cout << endl;//quebra de linha
12  }
```

Além de utilizar constantes para controlar o inicio e o fim do laço de repetição, foi combinado a utilização de 2 laços *for*. Fazendo a analogia com um tabuleiro de xadrez, é possível percorrer todas as linhas e colunas (coordenadas x,y).

Funcionamento de *for* dentro de outro *for*: a execução inicia pelo *for* de fora (mais externo), depois desvia para o *for* de dentro e só volta para o *for* de fora quando terminar toda execução do *for* de dentro (quando a variável de controle chegar no valor final).
Um *for* fica *parado* enquanto o outro *for* é executado, ou seja, enquanto sua variável de controle varia até chegar no valor final determinado para ela.

Finalmente, o último trecho do programa demonstra que não é necessário utilizar todos os parâmetros do comando *for*:

```
1   cout << "Contando, mas omitindo alguns parâmetros:";
2   contador = 1;
3   for( ; contador <= 5; )
4   {
5      cout << contador << " ";
6      ++contador;
7   }
8   cout << endl; //quebra de linha
```

Neste caso, não foi utilizado o parâmetro de inicialização e o de incremento. Na prática, o trecho de programa ficou similar a utilização da estrutura de repetição *while*.

É possível utilizar o comando *for* sem parâmetro algum:

```
for(;;)
```

Só que neste caso, deve existir dentro da estrutura de repetição alguma condição que levasse a execução de um comando *break*:

```
comando 1
...
comando n
for(;;)
{
    comando 1
    ...
    comando n
    if( condição )
    {
        break; //sai  do for
    }
}
comando 1
...
comando n
```

Na inicialização e no incremento, é possível trabalhar com várias variáveis. Para tal, basta separar por vírgulas (,).

```
for( linha = 1, coluna = 1;
        linha <= 10 && coluna <= 10;
        ++linha, ++coluna)
{
   cout << linha << "," << coluna << endl;
}
```

1.3 Convertendo um *for* em um *while*

O comando *while* pode ser escrito para comporta-se exatamente como o comando *for* (programa 2):

Programa 2: Convertendo *for* para *while*.

```
// convertendo for para while
// programa_002.cpp
```

```
#include "biblaureano.h"

int main()
{
   int valor, contador;
   valor = readInt("Contar até : ");

   for (contador=1;            // Atribuição inicial. Executado somente 1 vez,
                               // sempre no início
        contador <= valor; // A condição sempre será avaliada antes da execução
                               // das instruções agrupadas embaixo do comando for
        ++contador)            // O incremento (ou decremento), sempre ocorrerá após
                               // a execução das instruções agrupadas embaixo do comando for
   {
      cout << contador << endl;
   }

   cout << endl << "Mesmo programa convertido para while." << endl;
   contador = 1;                  // Atribuição inicial. Executado somente 1 vez,
                                  // sempre no início
   while( contador <= valor ) // A condição sempre será avaliada antes da
                                  // execução das instruções agrupadas embaixo
                                  // do comando for
   {
      cout << contador << endl;
      ++contador;  // O incremento (ou decremento), sempre ocorrerá após a
                   // execução das instruções agrupadas embaixo do comando while
   }

   cout << "Game over!!!" << endl;
   return 0;
}
```

1.4 Existência de uma variável

Quando declaramos uma variável de laço da forma como fizemos para *linhas* e *colunas*, seu *escopo*[1] é limitador ao laço *for*. O observe o trecho de código a seguir (retirado do programa 1):

```
   cout << "Contando em linhas e colunas:" << endl;

   const int LINHAS = 5,
             COLUNAS = 3;
   for( int linhas = 1; linhas <= LINHAS; ++linhas)
   {
      for( int colunas = 1; colunas <= COLUNAS; ++colunas)
      {
         cout << linhas << "," << colunas << " ";
      }
      cout << endl; //quebra de linha
   }
```

Como a declaração da variável ocorre dentro do comando *for*, a variável existirá na memória somente durante a execução do laço.

[1] O escopo de uma variável é a parte do programa em que ela pode ser acessada

Atenção: lembre-se que declarar uma variável na prática significa que estamos reservando uma porção da memória para guardar um valor. A memória é liberada ao término do programa. A declaração de uma variável dentro de um laço *for* significa que esta variável será eliminada da memória assim que a estrutura de repetição terminar.

2 Exercícios

1. Escreva um programa para que calculem o resultado de cada uma das seguintes séries com 10 termos (utilizando o comando *for*):
   - 1 + 3 + 9 + 27 + 81 + ...
   - 1/1 + 2/4 + 3/9 + 4/16 + 5/25 + ...
   - 1 - 2 + 3 - 4 + 5 - 6 +.....
2. Faça um programa que leia 2 números inteiros e positivos. O programa deverá mostrar a multiplicação de todos os números entre o intervalo (inclusive). Exemplo: se for digitado 15 e 19 deverá ser apresentado o resultado da multiplicação de 15 * 16 * 17 * 18 * 19. Utilizar o comando *for*.
3. Escrever um programa que leia 50 valores, encontre o maior e o menor deles e mostre o resultado. Utilizar o comando *for*.
4. Escreva um programa que declare e inicialize uma variável que receberá o maior número possível do tipo inteiro. Divida o valor dessa variável por 2 até que o resultado obtido seja inferior a 100 (não inclusivo). A cada iteração imprima o resultado.
5. A conversão de graus Fahrenheit para Centígrados é obtida por $C = \frac{5}{9} * (F - 32)$, onde C=centígrados, F=Fahrenheit. Faça um programa que calcule e escreva uma tabela de graus Centígrados e graus Fahrenheit, que variam de 50 a 65 de 1 em 1. Exemplo:

```
Fahrenheit  50  Centígrados 10
Fahrenheit  51  Centígrados 10,55
Fahrenheit  52  Centígrados 11,11...
```