

Linguagem C - Estruturas de Seleção

PARTE 1

Lógica de Programação

Professor: Vinícius T. Guimarães

viniciusguimaraes@ifsul.edu.br



Tecnólogo em Sistema para Internet
Campus Charqueadas

Introdução

Nas últimas aulas trabalhamos com estruturas condicionais, onde aprendemos como tornar o nosso programa capaz de executar um bloco de código de acordo com testes que avaliam um conjunto de condições.



Introdução

Hoje aprenderemos como automatizar a execução de instruções recorrentes, ou repetitivas.



Aquecimento

Elabore um programa que verifica se o número digitado pelo usuário é par e divisível por 3.

```
1  #include <stdio.h>
2
3  int main(void) {
4
5      int n;
6
7      printf("Digite um número:");
8      scanf("%i", &n);
9
10     if( (n%2==0) && (n%3==0)){
11         printf("%i é par e divisível por 3\n", n);
12     }
13     else{
14         printf("%i é par e NÃO é divisível por 3\n", n);
15     }
16
17     return 0;
18 }
```

<https://replit.com/@vicoguim/exemplo-repeticao-1-1>

Lógica de Programação

Aquecimento

Agora, elabore um programa que verifica os números de 1 a 10 que são pares e divisíveis por 3.

<https://replit.com/@vicoguim/exemplo-repeticao-1-2>



Observe que o método de resolução apresentado nesse exemplo **não é adequado**, pois ainda não fizemos o uso de laços de repetição.

Dúvida

E se o exercício anterior solicitasse que fosse criado um algoritmo que verifica os número entre 1 e 1000 que são pares e divisíveis por 3?



Laços de repetição

- Em programação, precisamos fazer com que o algoritmo **repita um bloco de código inúmeras vezes** para chegar a solução desejada.
- Nesses casos, **não faz sentido ou se torna impossível, copiar e colar o código** para resolver todos os casos do problema solicitado pelo algoritmo.
- Para essas situações utilizamos **estruturas de repetição**.

O laço for

- O laço **for** é a estrutura mais básica de repetição, também conhecida como repetição contada.
- Ele estabelece que um determinado bloco de código deve ser repetido ***n*** vezes.
- Para controlar quantas vezes um bloco de código deverá ser repetido, utiliza-se uma **variável de iteração**, associada a uma **condição de parada**.



Estrutura do laço for

O comando *for* (repetição contada)

- *Forma geral:*

```
for(inicialização; condição; incremento/decremento)  
{  
    <bloco de comandos>  
}
```

→ Início do Bloco

→ Fim do Bloco

Estrutura do laço for

Normalmente, a inicialização se caracteriza por ser um **comando de atribuição** que é usado para atribuir um **valor inicial na variável** que irá **controlar o laço de repetição**.



```
for(inicialização; condição; incremento/decremento)  
{  
    <bloco de comandos>  
}
```

→ Início do Bloco

→ Fim do Bloco



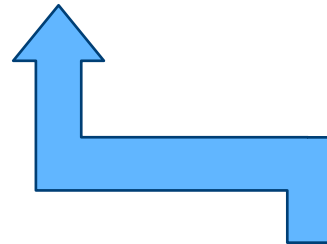
A inicialização é executada **uma única vez**, diferentemente da **condição** e do **incremento/decremento**.

Estrutura do laço for



Tipicamente, a **condição** se caracteriza por ser uma **expressão relacional** que **determina** quando o **laço deverá ser terminado**, ou seja, determina a **condição de parada** da estrutura de repetição.

A **condição** é executada logo após a inicialização. Se o resultado for **verdadeiro**, o bloco de comandos é executado e a **condição só será testada novamente**, depois de passar pelo **incremento/decremento**.



```
for(inicialização; condição; incremento/decremento)  
{  
    <bloco de comandos>  
}
```

→ Início do Bloco

→ Fim do Bloco

Estrutura do laço for



Na maior parte dos casos¹, a atualização da variável de controle (incremento/decremento) acontece **após a execução do bloco de comandos** e antecede a verificação da condição.

A seção de incremento/decremento **define como a variável que controla o laço de repetição deverá ser atualizada**, a cada repetição do laço.

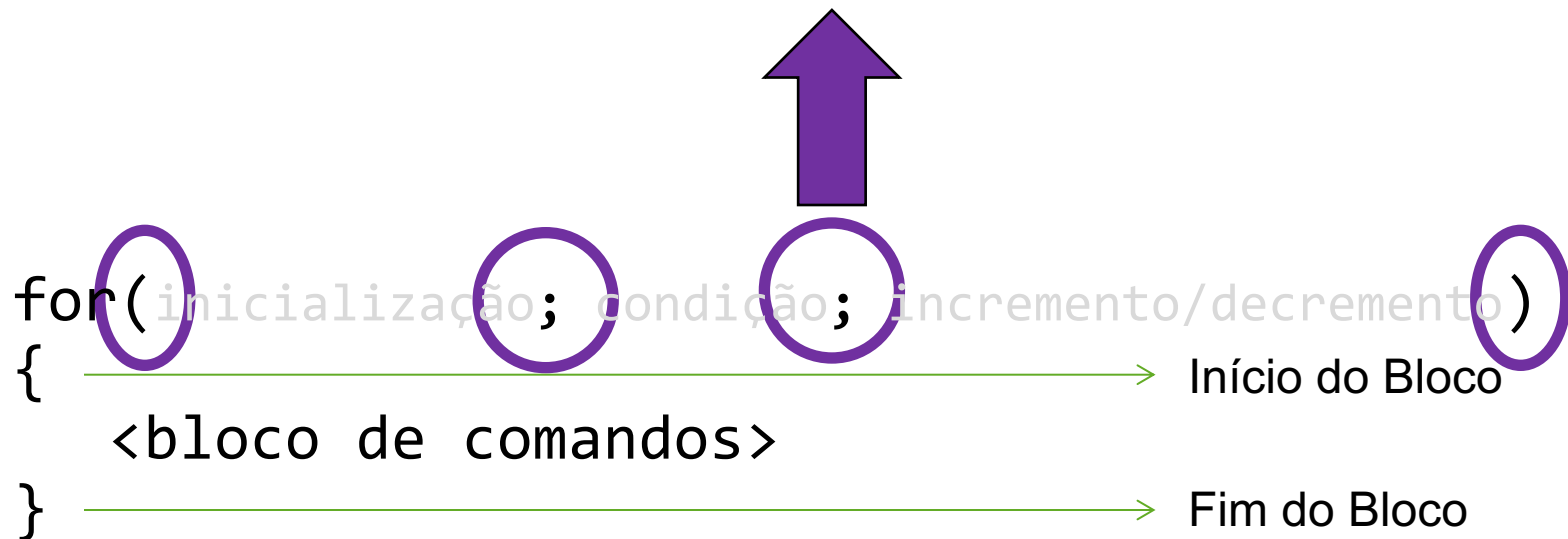


```
for(inicialização; condição; incremento/decremento)  
{  
    _____> Início do Bloco  
    <bloco de comandos>  
    _____> Fim do Bloco  
}
```

¹ Mais adiante veremos que é possível sair do laço sem fazer o incremento/decremento e testar a condição, por meio do comando break.

Estrutura do laço for

Não esqueçam que estas três seções principais do comando *for* deverão ser **separadas por pontos-e-vírgulas**. Além disso, estas seções deverão **estar entre parênteses**, assim como fizemos para o comando *if*.



The diagram illustrates the structure of a `for` loop. The code is shown as `for(inicialização; condição; incremento/decremento)` followed by a block of commands `{ <bloco de comandos> }`. Annotations include: three purple circles around the opening parenthesis, the semicolon after initialization, the semicolon after the condition, and the closing parenthesis; a large purple arrow pointing upwards from the semicolon after the condition to the explanatory text box above; a green arrow pointing from the opening parenthesis to the text 'Início do Bloco'; and another green arrow pointing from the closing brace to the text 'Fim do Bloco'.

```
for( inicialização; condição; incremento/decremento )  
{  
    <bloco de comandos>  
}
```

Início do Bloco

Fim do Bloco

Exemplo de código for

<https://replit.com/@vicoguim/exemplo-repeticao-1-3>

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for(i = 0; i <= 1000; i++){
6          printf("%i\n", i);
7      }
8      return 0;
9  }
```

Vai repetir por 1001 vezes,
imprimindo na tela os valores
de 0 (zero) até 1000 (mil) em
ordem crescente.



O que esse código faz?

Lógica de Programação

Exemplo de código for

<https://replit.com/@vicoguim/exemplo-repeticao-1-4>

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for(i = 1000; i >= 0; i--){
6          printf("%i\n",i);
7      }
8      return 0;
9  }
```

Vai repetir por 1001 vezes,
imprimindo na tela os valores
de 1000 (mil) até 0 (zero) em
ordem decrescente.



O que esse código faz?

Lógica de Programação

Incremento e decremento unário

`a++`; mesmo que `a = a + 1`;

`a--`; mesmo que `a = a - 1`;

`++a`; mesmo que `a = a + 1`;

`--a`; mesmo que `a = a - 1`;

A diferença é que, quando o operador **++** ou **--** precede a variável (ou seja, vem antes da variável), ele é imediatamente executado e o restante das operações utilizam o variável já atualizado. Quando operador aparece após a variável, ele só é executado após toda instrução ter sido executada, ou seja, utiliza o valor atual da variável para após fazer o incremento/decremento e, então atualizar o valor da variável.

<https://replit.com/@vicoguim/exemplo-repeticao-1-5>

Exemplo de código for

<https://replit.com/@vicoguim/exemplo-repeticao-1-6>

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5
6      // i+=2 mesmo que i = i + 2;
7      for(i = 2; i <= 1000; i+=2){
8          printf("%i \n", i);
9      }
10
11     return 0;
12 }
```

Vai repetir por 500 vezes, imprimindo na tela os valores de 2 (dois) até 1000 (mil) em ordem decrescente, de dois em dois.



O que esse código faz?

Exemplo de código for

<https://replit.com/@vicoguim/exemplo-repeticao-1-7>

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i;
5      for(i = 1; i <= 1000; i++){
6          if(i%2 == 0){
7              printf("%i\n", i);
8          }
9      }
10     return 0;
11 }
```

Vai repetir por 1000 vezes,
imprimindo na tela apenas
os valores pares.



O que esse código faz?

Dúvida (repetindo...)

E se o exercício solicitasse que fosse criado um algoritmo que verifica os número entre 1 e 1000 que são pares e divisíveis por 3?



Resolvendo

<https://replit.com/@vicoguim/exemplo-repeticao-1-8>

```
1  #include <stdio.h>
2
3  int main(void) {
4
5      int i;
6      for(i = 1; i <= 1000; i++){
7          if( (i%2==0) && (i%3==0)){
8              printf("%i é par e divisível por 3\n", i);
9          }
10         else{
11             printf("%i é par e NÃO é divisível por 3\n", i);
12         }
13     }
14     return 0;
15 }
```

Exemplos extras

<https://replit.com/@vicoguim/exemplo-repeticao-1-9>

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int x,y;
6      x = 10;
7      for(y = 10; y != x; y = y + 1)
8      {
9          printf("%i\n",y);
10     }
11     printf("%i\n",y);
12     return 0;
13 }
```



O que acontece nesse código?

O laço nunca será executado, pois x e y possuem o mesmo valor já no primeiro teste condicional. Desta forma, a expressão é avaliada como falsa e nem o corpo nem a porção de incremento do laço são executados. Assim, y continua com o valor 10 e a saída é o número 10 escrito apenas um vez na tela.

Exemplos extras

<https://replit.com/@vicoguim/exemplo-repeticao-1-10>

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int x;
6
7      for(x = 0; x != 123; )
8      {
9          scanf("%i", &x);
10         printf("%i\n", x);
11     }
12     return 0;
13 }
```



O que acontece nesse código? Está certo?

Este programa está correto, pois é facultativo o uso de expressões em qualquer uma das seções do comando *for*. Neste caso, a repetição só será terminada quando o usuário digitar o valor 123.