

# **Disciplina: Introdução à Programação**

## **Aula 7: Estrutura de repetição com pré-teste e com pós-teste**

# Apresentação

Imagine que você precisa escrever um algoritmo ou um programa que exiba todos os números pares entre 10 e 100.

Com os comandos que você já aprendeu até o momento, seriam necessárias muitas linhas para imprimir os números 10, 12, 14, 16, 18, e assim sucessivamente até 100.

Para situações nas quais existe um padrão de repetição, como no exemplo hipotético acima, podemos recorrer às estruturas repetitivas, ou estruturas de repetição. Elas nos ajudam a escrever menos código quando precisamos repetir determinados trechos várias vezes e facilitam demais o nosso trabalho! Nesta aula, veremos como funcionam.

---

## Objetivos

- Identificar as estruturas de repetição;
- Diferenciar as estruturas de repetição com pré-teste das com pós-teste;
- Registrar algoritmos e programas, utilizando a estrutura de repetição adequada.

# Estruturas repetitivas

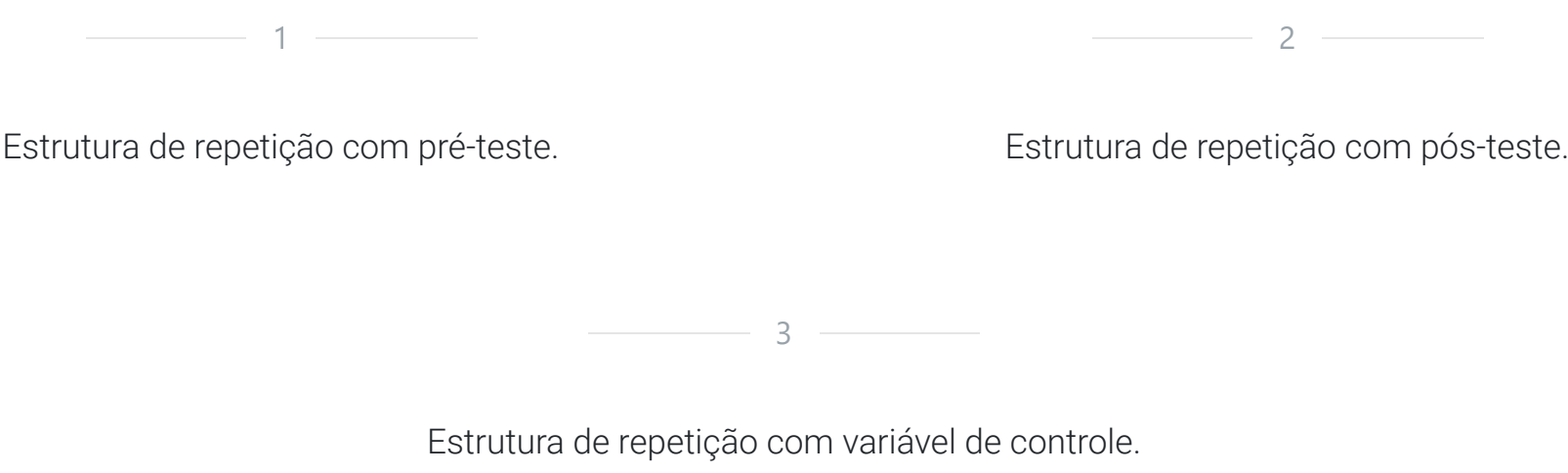
---

Você se lembra da situação hipotética criada no início desta aula, na qual você precisa de um código capaz de exibir os números pares entre 10 e 100?

Em um primeiro momento, recorrendo ao que já aprendeu sobre algoritmos e programação, você pode ter ficado assustado pensando na quantidade de linhas de código que precisaria escrever para resolver tal situação. Mas não há razão para isso.

Temos estruturas específicas às quais podemos recorrer em situações nas quais é necessário repetir determinado trecho do código: as estruturas repetitivas ou de repetição.

As estruturas repetitivas estão disponíveis em três formatos distintos:



Veremos como funcionam as duas primeiras formas: com pré-teste e com pós-teste. Para começar, vamos ver a sintaxe da estrutura com pré-teste?

É comum chamarmos as estruturas repetitivas de **laço** ou **loop**.

# Estrutura de repetição com pré-teste

As estruturas repetitivas com pré-teste, ou teste inicial, farão a repetição de determinada parte do código **enquanto uma condição for verdadeira**. Isso significa que, no momento em que a condição for falsa, o trecho descrito dentro da estrutura de repetição deixará de ser executado. Ela é definida como sendo uma estrutura com pré-teste porque o teste da condição é feito **antes** da repetição do trecho.

Observe sua sintaxe no Portugol Studio e no C++:

```
* Portugol Studio
enquanto (condição)
{
    //Comandos a serem executados enquanto a condição for verdadeira.
}

* C++
while (condição)
{
    //Comandos a serem executados enquanto a condição for verdadeira.
}
```

A condição a ser testada é semelhante àquela que utilizamos nas estruturas de seleção, ou seja, uma expressão que, após avaliada, retornará o valor verdadeiro ou falso. Para entender melhor como funciona a estrutura repetitiva com pré-teste, vamos ver como ficaria o algoritmo para exibição dos números pares entre 10 e 100?

Observe:

## \* Portugol Studio

```
1  programa
2  {
3      funcao inicio()
4      {
5          inteiro num
6          num=10
7          enquanto (num<=100)
8          {
9              escreva(num, "\n")
10             num+=2
11         }
12     }
13 }
```

A estrutura repetitiva começa na linha 7 e termina na linha 11.

As chaves nas linhas 8 e 11 delimitam, respectivamente, o início e o fim do trecho a ser repetido, que está descrito nas linhas 9 e 10. Ele será repetido enquanto a variável **num** estiver armazenando um valor menor ou igual a 100.

Teste o exemplo no Portugol Studio e você verá que o resultado final será a exibição dos números pares entre 10 e 100, um em cada linha; já que usamos “\n”.

Vamos ver como ficaria esse algoritmo escrito em C++?

\* Programa em C++

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int num=10;
6      while (num<=100)
7      {
8          cout <<num<<"\n";
9          num+=2;
10     }
11 }
```

Agora, imagine que você precisa criar um algoritmo que exiba a tabuada de um número qualquer informado pelo usuário. Utilizado a estrutura **enquanto**, teríamos o seguinte código:


\* Portugol Studio

```
1  programa
2  {
3      funcao inicio()
4      {
5          inteiro num, contador, res
6          contador=0
7          escreva("Você deseja ver a tabuada de que número? ")
8          leia (num)
9          enquanto (contador<=10)
10         {
11             res = num * contador
12             escreva(num, " x ", contador, " = ", res, "\n")
13             contador++
14         }
15     }
16 }
```

No exemplo anterior, a repetição é controlada pela variável **contador**. Veja que ela é incrementada a cada iteração (repetição) e testada no início da estrutura, antes que o trecho a ser repetido (linhas 11, 12 e 13) seja efetivamente executado.

Toda repetição precisa de uma variável de controle que seja alterada dentro do laço; o que é feito na linha 13 com a mudança do valor da variável **contador**. Se você se esquecer disso, a repetição não terá fim e o programa ficará executando eternamente, preso no que chamamos de *loop* infinito.

Para entender melhor o que isso significa, teste o algoritmo anterior retirando do código a linha 13 e observe como ele irá se comportar. Veja que ele vai sempre apresentar a tabuada do número informado pelo usuário multiplicado por 0, que é o valor inicial da variável de controle.

No Portugol Studio, para interromper a execução do código em loop infinito, clique no botão 

Para que fique mais claro o papel de cada variável no algoritmo anterior, imagine que o usuário solicitou ver a tabuada de 5 e observe, a seguir, o que seria exibido:

Você deseja ver a tabuada de que número? 5

5 x 0 = 0

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

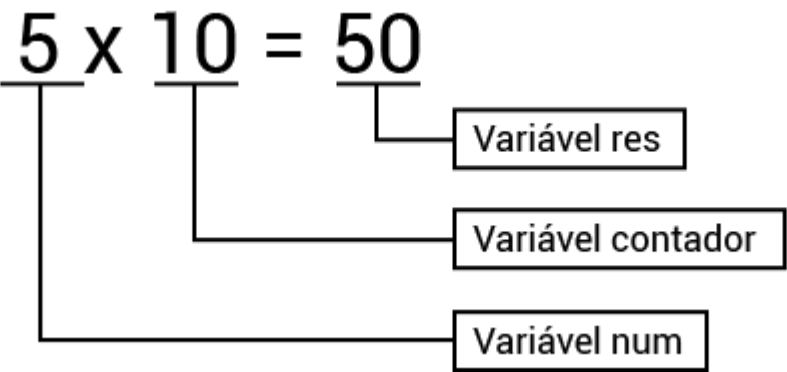
5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50



Agora, imagine que você precisa receber uma sequência de números informada pelo usuário e, para cada número, dizer se ele é par ou ímpar. Quando o usuário quiser interromper a sequência, ele informará o número -1.

Vamos ver como ficaria o algoritmo para solução desse problema?

## \* Portugol Studio

```
1  programa
2  {
3  funcao inicio()
4  {
5  inteiro num
6  escreva("Informe um número qualquer: ")
7  leia(num)
8  enquanto (num!=-1)
9  {
10     se (num%2==0)
11     {
12     escreva("Este número é par!\n")
13     }
14     senao
15     {
16     escreva("Este número é ímpar!\n")
17     }
18     escreva("Informe um número qualquer: ")
19     leia(num)
20 }
21 }
22 }
```

Observe que este algoritmo utiliza tanto a estrutura seletiva **se** quanto a estrutura repetitiva **enquanto**. Isso mostra que as estruturas não são excludentes, ou seja, o fato de você utilizar uma delas não significa que não poderá usar nenhuma outra. É bastante comum que os algoritmos e programas combinem as estruturas nas soluções apresentadas.

Vamos fazer alguns testes para entender melhor o programa? Na linha 5, inicialize a variável **num** com zero e coloque as linhas 6 e 7 como comentários. Após essas alterações, essas linhas devem estar da seguinte maneira:

```
inteiro num=0
//escreva("Informe um número qualquer: ")
//leia(num)
```

Execute o algoritmo e observe o que é exibido.

Veja que, mesmo sem o usuário informar qualquer número, a primeira mensagem mostrada em tela é:

**Este número é par.**

De que número o algoritmo está falando?

Observe que a estrutura seletiva avalia a variável **num** em sua condição. Essa variável não está vazia, lembra?

Ela foi inicializada com zero na linha de sua declaração.

Agora está claro por que precisamos manter as linhas 6 e 7 no código?

Para evitarmos que um valor que não foi informado pelo usuário seja avaliado na primeira execução da repetição.

Vamos fazer um novo teste. Desfaça as alterações nas linhas 6 e 7 e coloque como comentários as linhas 18 e 19. Feito isso, execute o algoritmo.

Observe que, após informar um número qualquer, o algoritmo exibiu a mensagem dizendo se este número é par ou ímpar e entrou em *loop* infinito. Você sabe por que isso aconteceu? Com certeza você se lembra do que foi dito há pouco, sobre a necessidade de a variável de controle ser alterada dentro da estrutura repetitiva, certo?

Do jeito como ficou o algoritmo, sem a execução das linhas 18 e 19, isso não está acontecendo. A variável, nesta situação, recebeu um valor fora da estrutura repetitiva (linha 7) e permaneceu com ele infinitamente. Precisamos pedir um novo número ao usuário dentro do laço para evitar que a execução do algoritmo fique presa dentro da estrutura de repetição.

# Atividade

1. Leia os enunciados a seguir e crie os algoritmos e os programas em C++ que solucionem os problemas propostos.

a) Um instituto está realizando uma pesquisa para descobrir a idade e o peso médios da população de um bairro. Cada entrevistado informa seu peso e sua idade e, quando for informado 0 para o peso, o recebimento de dados deve ser interrompido e devem ser exibidas a quantidade de entrevistados, a média de peso e a média de idade dos mesmos.

b) O Jockey Clube está organizando uma competição de hipismo e criou as seguintes categorias:

INFANTIL    competidores entre 7 e 12 anos

JUVENIL    competidores entre 13 e 17 anos

ADULTO    competidores entre 18 e 49 anos

SENIOR    competidores com 50 anos ou mais

Para ajudar na classificação correta, crie um código capaz de informar a categoria do competidor a partir de sua idade. O programa deve receber a idade dos competidores enquanto a mesma for diferente de -1.

Caso seja informada uma idade abaixo de sete anos, informe ao usuário que o competidor está fora da faixa etária permitida na competição.



# Estrutura de repetição com pós-teste

---

As estruturas repetitivas com pós-teste, ou teste final, farão a repetição de determinada parte do código **enquanto uma condição for verdadeira**.

Isso significa que, no momento em que a condição for falsa, o trecho descrito dentro da estrutura de repetição deixará de ser executado. Ela é definida como sendo uma estrutura com pós-teste porque o teste da condição é feito **depois** da repetição do trecho. O que difere esta estrutura da estrutura com pré-teste é o fato de que **o bloco de comandos a ser repetido será executado ao menos uma vez**.

Observe sua sintaxe no Portugol Studio e no C++:

```

* Portugol Studio
faca
{
    //Comandos a serem executados enquanto a condição for verdadeira.
} enquanto (condição)

* C++
do
{
    //Comandos a serem executados enquanto a condição for verdadeira.
} while (condição)
```

Você se lembra do último exemplo, no qual recebíamos uma sequência de números do usuário e, para cada um deles, informávamos se o mesmo era par ou ímpar?

Vamos ver como ele seria escrito com a estrutura de repetição com pós-teste?

Observe:

## \* Portugol Studio

```
1  programa
2  {
3  funcao inicio()
4  {
5  inteiro num
6  escreva("Informe um número qualquer: ")
7  leia(num)
8  faca
9  {
10     se (num%2==0)
11     {
12         escreva("Este número é par!\n")
13     }
14     senao
15     {
16         escreva("Este número é ímpar!\n")
17     }
18     escreva("Informe um número qualquer: ")
19     leia(num)
20     } enquanto (num!=-1)
21 }
22 }
```

Execute este algoritmo e veja como ele se comporta.

Você pode estar pensando que não há nenhuma diferença em relação ao algoritmo que utiliza a estrutura com teste inicial, mas não é bem assim.

A execução do programa será finalizada quando solicitarmos um número e o usuário digitar -1.

Essa condição que finaliza a repetição é chamado de flag (palavra em inglês para “bandeira”) e tem a função de sinalizar quando a iteração deve ser interrompida. O número -1 – que é *flag* deste algoritmo – não deve, portanto, ser avaliado pela estrutura seletiva e não deve ser classificado como par ou ímpar.

Se, ao executar o algoritmo, o primeiro número informado por você for -1, será exibida a mensagem **Este número é ímpar** e o algoritmo solicitará a digitação de um novo número.

**“Mas não era para ele finalizar a execução quando eu digitasse -1?” (você pode estar se perguntando)**

**Sim, era!**

Mas, a estrutura de repetição com teste final faz com que todo o trecho dentro dela seja executado ao menos uma vez, o que ocasiona esta situação.

Vamos ver como ficaria o algoritmo para exibição da tabuada de um número fornecido pelo usuário? Veja a seguir:

\* **Portugol Studio**

```
1  programa
2  {
3      funcao inicio()
4      {
5          inteiro num, contador, res
6          contador=0
7          escreva("Você deseja ver a tabuada de que número? ")
8          leia (num)
9          faca
10         {
11             res = num * contador
12             escreva(num, " x ", contador, " = ", res, "\n")
13             contador++
14         } enquanto (contador<=10)
15     }
16 }
```

Neste exemplo, não teremos o problema descrito há pouco, no qual a *flag* é ignorada quando o primeiro número informado for -1. Isso significa que a estrutura repetitiva com pós-teste não será sempre um dificultador. É preciso avaliar o problema e escolher a estrutura mais adequada para a solução.

# Atividade

2. Refaça os algoritmos e programas propostos na atividade 1 utilizando a estrutura repetitiva faça..enquanto. Realize as adequações necessária para que os códigos funcionem adequadamente, sem problemas com as respectivas flags.

Vamos ver como fica o programa para exibição da tabuada no C++? Observe:

# C++

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int num, contador, res;
6      contador=0;
7      cout << "Você deseja ver a tabuada de que número? ";
8      cin >> num;
9      do
10     {
11         res = num * contador;
12         cout << num << " x " << contador << " = " << res << "\n";
13         contador++;
14     } while (contador<=10);
15 }
```

É muito importante que você perceba as diferenças nas estruturas repetitivas com teste inicial e com teste final. Não deixe de fazer as atividades propostas, pois são elas que te ajudarão a perceber suas dificuldades e identificar os pontos da aula que precisam ser revistos. Bom estudo!

# Atividade

3. Você se lembra do exercício da lanchonete, no qual recebíamos um pedido de um cliente e exibíamos o valor a pagar pelo mesmo? Reveja o enunciado:

Uma lanchonete está disponibilizando para seus clientes um terminal por meio do qual poderão fazer seu pedido. Para que isso aconteça, o terminal deve exibir o menu de itens disponíveis conforme se vê a seguir:

- \*\*\* Cardápio \*\*\*
- 100 – Hambúrguer – R\$5,50
  - 101 – Cachorro-quente – R\$4,50
  - 102 – Milk-shake – R\$7,00
  - 103 – Pizza brotinho – R\$8,00
  - 104 – Cheeseburger – R\$8,50

Informe o código do seu pedido:

Uma vez que o cliente informe o código do item desejado, o terminal deverá perguntar a quantidade de itens que ele deseja pedir. Ao final, o usuário deverá informar o pedido do cliente e o valor a pagar.

Escreva o algoritmo e seu equivalente em C++ que possam resolver o problema da lanchonete. Utilize as estruturas CASO e SWITCH.

Da forma como está descrito o problema, o cliente da lanchonete pode comprar vários itens de um mesmo tipo, por exemplo, 2 hambúrgueres. Não é possível, entretanto, pedir 2 hambúrgueres e 2 milk-shakes ao mesmo tempo. Que tal modificar o código para resolver esta questão? Modifique a programação para que, após informar o código de um item, a solução pergunte ao usuário se ele deseja incluir mais algum item no pedido. Se sim (“S”), o algoritmo/programa deve pedir o código do novo item. Se não (“N”), o algoritmo/programa deve exibir o valor a pagar. Não é necessário exibir o pedido completo do cliente.

## Notas Referências

MANZANO, J. A. N. G., OLIVEIRA, J. F. **Algoritmos:** lógica para desenvolvimento de programação de computadores. 28.ed. São Paulo: Érica, 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados com aplicações em Java.** 2.ed. São Paulo: Prentice Hall, 2005.

## Próxima aula

- Estruturas de repetição com variável de controle;
- Diferenças das estruturas de repetição com variável de controle das estruturas de repetição com pré-teste ou pós-teste;

- Algoritmos e programas com a estrutura de repetição.

## Explore mais

---

Você conhece o [Flappy Bird <https://studio.code.org/flappy/1>](https://studio.code.org/flappy/1) , aquele famoso jogo para dispositivos móveis? Que tal construir o seu próprio Flappy Bird?