C#-6 - Loops_v1

1. Array
2. Estruturas de repetição

1. Array

Fornece métodos para criar, manipular, pesquisar e classificar matrizes, servindo assim como a classe base para todas as matrizes no Common Language Runtime.

Matrizes unidimensionais

Você cria uma matriz unidimensional usando o novo operador que especifica o tipo de elemento de matriz e o número de elementos. O exemplo a seguir declara uma matriz de cinco inteiros:

```
int[] array = new int[5];
```

Essa matriz contém os elementos de array[0] a array[4]. Os elementos da matriz são inicializados para o valor padrão do tipo de elemento, para inteiros.

É possivel inicializar uma matriz unidirecional das seguintes formas:

```
int[] array2 = { 1, 3, 5, 7, 9 };
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

também é possivel declarar a variavel e instanciar seus valores depois:

```
int[] array3;
array3 = new int[] { 1, 3, 5, 7, 9 };
```

Retornando dados

Para retornar os dados de uma matriz é possivel acessar pelo seu indice começando em 0, veja a seguir:

```
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

Console.WriteLine(weekDays2[0]);
Console.WriteLine(weekDays2[1]);
Console.WriteLine(weekDays2[2]);
Console.WriteLine(weekDays2[3]);
Console.WriteLine(weekDays2[4]);
Console.WriteLine(weekDays2[5]);
Console.WriteLine(weekDays2[6]);

/*Output:
Sun
Mon
Tue
Wed
```

```
Thu
Fri
Sat
*/
```

Matrizes multdimensionais

Arrays podem ter várias dimensões. Uma array unidimensional é chamada de vetor. Já uma array bidimensional é chamada de matriz. É possivel inicializar uma matriz de diferentes maneiras, veja abaixo:

```
int[,] array2D = new int[,] {{ 1, 2 },
                             { 3, 4 },
                             { 5, 6 },
                             { 7, 8 }};

int[,] array2Da = new int[4, 2] {{ 1, 2 },
                                 { 3, 4 },
                                 { 5, 6 },
                                 { 7, 8 }};

string[,] array2Db = new string[3, 2] {{ "one", "two"},
                                       { "three", "four"},
                                       { "five", "six"}};

// Three-dimensional array.
int[,,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                { { 7, 8, 9 }, { 10, 11, 12 } } };
// The same array with dimensions specified.
int[,,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                      { { 7, 8, 9 }, { 10, 11, 12 } } };
```

É possivel também inicializar uma matriz sem especificar a classificação:

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

também é possivel declarar a variavel e instanciar seus valores depois:

```
int[,] array5;
array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };   // OK
```

Retornando dados

Para retornar os dados de uma matriz é possivel acessar pelo seu indice começando em 0, veja a seguir:

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" },{
"five", "six" } };

int[,,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },{ { 7, 8, 9 }, { 10,
11, 12 } } };
int[,,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } },{ { 7, 8, 9 },
{ 10, 11, 12 } } };
```

```
// Accessing array elements.
System.Console.WriteLine(array2D[0, 0]);
System.Console.WriteLine(array2D[0, 1]);
System.Console.WriteLine(array2D[1, 0]);
System.Console.WriteLine(array2D[1, 1]);
System.Console.WriteLine(array2D[3, 0]);
System.Console.WriteLine(array2Db[1, 0]);
System.Console.WriteLine(array3Da[1, 0, 1]);
System.Console.WriteLine(array3D[1, 1, 2]);
```

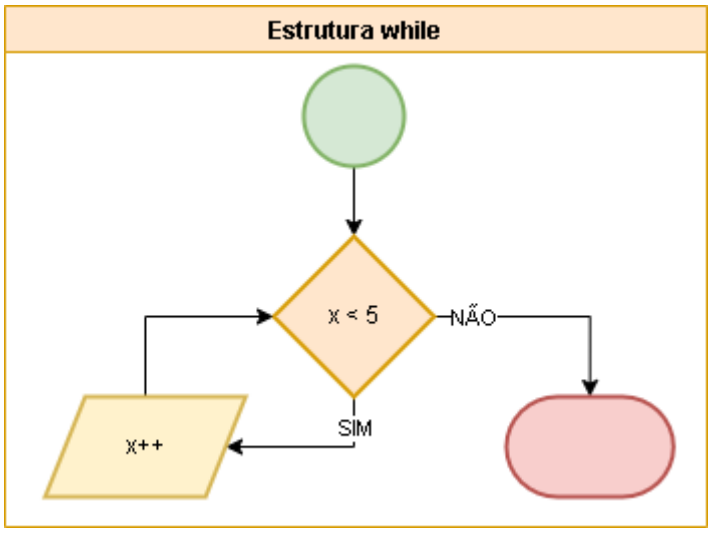
2. Estruturas de repetição

Essa estrutura repete ações enquanto uma condição permanecer verdadeira. Essa condição pode ser tanto uma condição simples, como também uma condição composta, que nada mais é do que um grupo de condições. Cada ciclo do loop é chamado de iteração.

Estrutura while

A instrução while executa uma instrução ou um bloco de instruções enquanto uma expressão booliana especificada é avaliada como true. Como essa expressão é avaliada antes de cada execução do loop, um loop while é executado zero ou mais vezes. Veja abaixo um algoritmo:

Fluxograma:



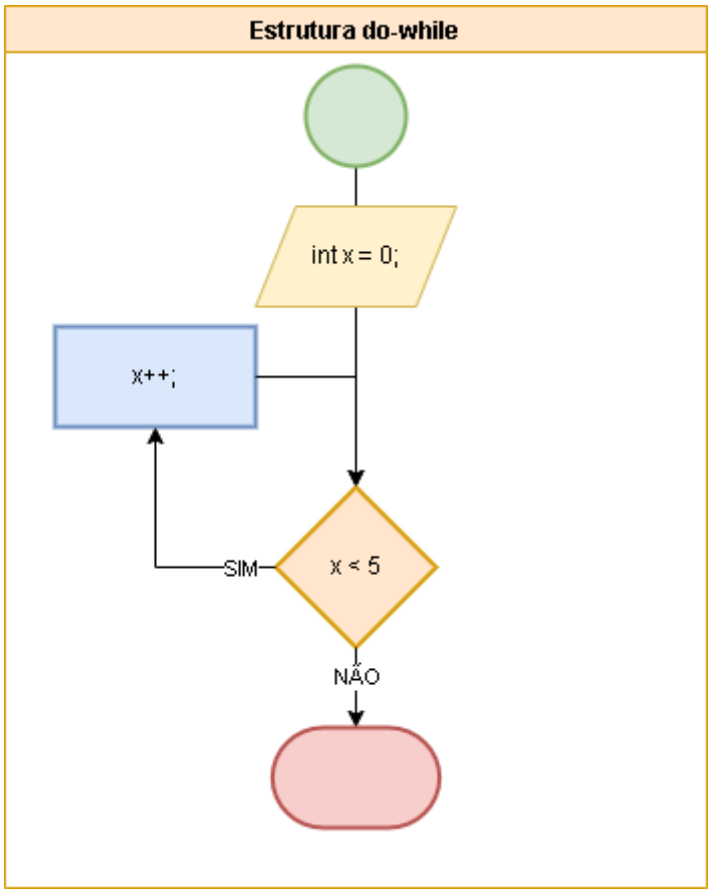

```
int x = 0;
while (x < 5)
{
    Console.Write(x);
    n++;
}
// Output:
// 01234
```

Antes da primeira iteração, as condições do loop são avaliadas. Por isso, esse tipo de loop é chamado de repetição pré-testada.

Estrutura do-while

Um outro tipo de loop é o do-while (faz-enquanto). Nesse tipo de loop, pelo menos uma iteração ocorre, porque a avaliação da condição se dá após a execução da iteração. Por isso, esse tipo de repetição é chamada de repetição pós-testada:

Fluxograma:




```
int x = 0;
do
{
    Console.Write(x);
    x++;
} while (x < 5);
// Output:
// 01234
```

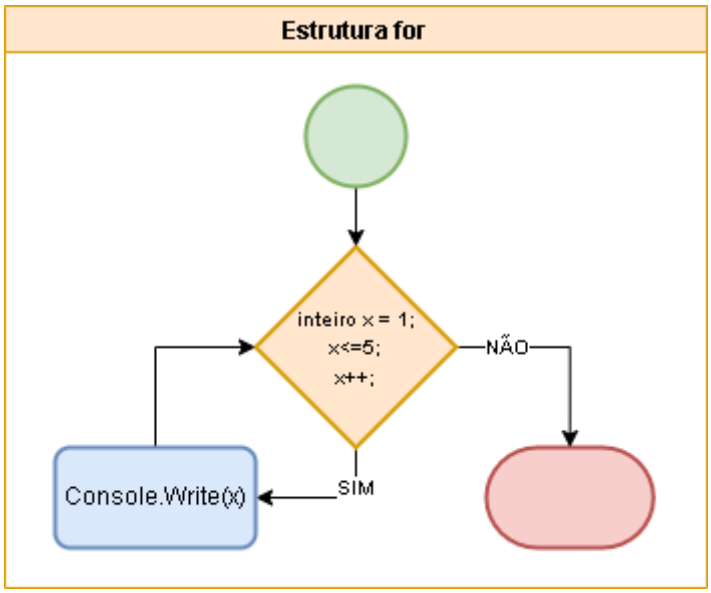
Note que aqui não é necessário atribuir um valor inicial para a variável x que seja menor que 5, porque a primeira iteração está garantida. Por isso, esse algoritmo ficou mais adequado a esse tipo de situação, até porque evitou a necessidade da atribuição de um valor inicial à variável usada na condição.

Estrutura for

A maioria das linguagens de programação suportam o for. Ele define três ações em uma sintaxe compacta. Essas ações são inicialização, condição e atualização. A inicialização é usada para inicializar variáveis e é a primeira coisa que é feita no for. Só é feita no começo. A condição é feita antes de cada iteração, inclusive a primeira. A atualização é feita após cada iteração.

O uso mais comum de um for é inicializar uma variável, que é chamada de variável de controle, testa-lá na condição do for e depois atualiza-lá. Por causa do uso da variável de controle, esse tipo de repetição é chamado também de repetição com variável de controle.

Fluxograma:




```
for (int x = 0; x < 3; x++)
{
    Console.Write(x);
}
// Output:
// 012
```

Instruções de salto

Uma instrução de salto altera o fluxo do loop e posiciona a execução em diferentes partes do código, segue abaixo algumas instruções de salto;

Break

A break instrução transfere o controle para a instrução que segue a instrução terminada, se houver.

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
foreach (int number in numbers)
{
    if (number == 3)
    {
        break;
    }

    Console.Write($"{number} ");
}
Console.WriteLine();
Console.WriteLine("End of the example.");
// Output:
// 0 1 2
// End of the example.
```

Continue

A continue instrução inicia uma nova iteração da continue de fechamento mais próxima.

```
for (int i = 0; i < 5; i++)
{
    Console.Write($"Iteration {i}: ");

    if (i < 3)
    {
        Console.WriteLine("skip");
        continue;
    }

    Console.WriteLine("done");
}
// Output:
// Iteration 0: skip
// Iteration 1: skip
// Iteration 2: skip
// Iteration 3: done
// Iteration 4: done
```

Não se esqueça, olhar a documentação fara que seu conhecimento sobre a linguagem seja refinado, então pesquise os termos aprendido e faça uma leitura mais rica no seu tempo. Link DOC.