Fundamentos da Linguagem C#

Operadores

Para manipular os valores das variáveis de um programa, devemos utilizar os operadores oferecidos pela linguagem de programação adotada. A linguagem C# possui diversos operadores e os principais são categorizados da seguinte forma:

• Aritmético (+, -, *, /, %,(unário) ++, --)

• Atribuição (=, +=, -=, *=, /=, %=)

• Relacional (==, !=, <, <=, >, >=, (ternário) ?:)

• Lógico (&&, ||)

Aritmético

As operações aritméticas em C# obedecem as mesmas regras da matemática com relação à precedência dos operadores e parênteses. Portanto, as operações são resolvidas a partir dos parênteses mais internos até os mais externos, primeiro resolvemos as multiplicações, divisões e os módulos. Em seguida, resolvemos as adições e subtrações.

Sendo o módulo de um número o resto da divisão de um número por outro.

Relacional

Muitas vezes precisamos determinar a relação entre uma variável ou valor e outra variável ou valor. Nessas situações, utilizamos os operadores relacionais. As operações realizadas com os operadores relacionais devolvem valores do tipo primitivo bool.

```
int valor = 2;
bool t = false;
t = (valor == 2);    // t = true
t = (valor != 2);    // t = false
t = (valor < 2);     // t = false
t = (valor <= 2);    // t = true
t = (valor > 1);     // t = true
t = (valor >= 1);    // t = true
```

- Condicional <Condição>?valor1:valor2

```
int valor = 6;
string msg = "";
msg = valor > 7 ? "Aprovado" : "Reprovado";
```

Lógico

A linguagem C# permite verificar duas ou mais condições através de operadores lógicos. Os operadores lógicos devolvem valores do tipo primitivo bool. Os operadores lógicos são:

• “E” lógico &&

• “OU” lógico ||

Os operadores podem ser:

Unary operators: Trabalha com apenas um operando. Examplo: ++x, x++.

Binary operators: Trabalha com dois operandos. Examplo: x+y ou x>y.

Ternary operators: Trabalha com três operandos. Existe somente um operador ternário, ?: em C#

Arrays

Um array é uma coleção de itens em que cada item pode ser acessado por meio de um índice exclusivo.

```
int[] numbers = { 1, 2, 3, 4, 5 };
```

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

```
number[0] = 1;
```

int [,] matriz = new int[2,3];

int [,] matriz = {{1,2,3}, {2,3,4}};

Qualquer item da matriz pode ser acessado diretamente usando um índice. No .NET Framework, índices de matriz são baseados em zero. Isto significa que para acessar o primeiro elemento de uma matriz, use o índice 0; para acessar o segundo elemento, você usa o índice 1, e assim por diante.

Para acessar um elemento de matriz individual, você use o nome da matriz seguido pelo índice entre colchetes. Por exemplo, o número [0] irá retornar o valor 1 a partir da matriz declarada acima, e o número [4] irá retornar o valor 5. É ilegal acessar uma matriz fora de seus limites definidos. Por exemplo, você obterá um erro se você tentar acessar os números de elemento de array [5].

3 Formas de Inicializar um vetor:

1. int [ ] vetor = new int[3];

2. int [ ] vetor = {1,2,3};

3. int [ ] vetor;
   int T;
   Console.write(“Tamanho:”);
   T = int.parse(Console.readline());
   Vetor = new int[T];

Estruturas de Repetição – LEITURA DE ARRAYS

FOREACH

O loop foreach é útil para a iteração através dos elementos de uma coleção.

```
static void Main()
{
    int[] vetor = new int[] { 0, 1, 2, 3, 5, 8 };
    foreach (int elemento in vetor)
    {
        System.Console.WriteLine(elemento);
    }
    System.Console.WriteLine();


    // Compare com o loop.
    for (int i = 0; i < vetor.Length; i++)
    {
        System.Console.WriteLine(vetor[i]);
    }
    System.Console.WriteLine();


    // Para contar os elementos.
    int count = 0;
    foreach (int elemento in vetor)
    {
        count += 1;
        System.Console.WriteLine("Elemento {0}: {1}", count, elemento);
    }
    System.Console.WriteLine("Numero de elementos no vetor: {0}", count);
}
```

Métodos (Funções)

Métodos são uma forma de modularizar o código. Os métodos podem receber parâmetros por valor e por referência.

Tipos como int, float, double, char, string e struct são tipos de valor. Assim quando preciso passar sua referência (seu endereço de memória) como parâmetro para um método, devo utilizar *ref*

Tipos como como os **arrays** são tipos de referência. Assim , quando preciso passar sua referência(Seu endereço de memória) como parâmetro para um método, não tem necessidade de utilizar *ref*, pois ele **já é uma referência**.

Exemplo – Por valor

```
static void Metodo(int Param1)
{
    Param1 = 100; // Aqui a saída será 100
    Console.WriteLine("Valor no Metodo = " + Param1);
}

0 references
static void Main()
{
    int valor = 5;
    Metodo(valor);
    // Aqui a saída será 5, perceba que o valor não foi modificado
    Console.WriteLine("Valor na Main = " + valor);
    Console.ReadKey();
}
```

Exemplo – Por referência

```
static void Metodo(ref int Param1)
{
    Param1 = 100; // Aqui a saída será 100
    Console.WriteLine("Valor no Metodo = " + Param1);
}

0 references
static void Main()
{
    int valor = 5;
    Metodo(ref valor);
    // Aqui a saída será 100, perceba que o valor foi modificado
    Console.WriteLine("Valor na Main = " + valor);
    Console.ReadKey();
}
```

Estrutura (Struct)

Uma **struct** é uma variável especial que contém diversas outras variáveis normalmente de tipos diferentes. As variáveis internas contidas pela **struct** são denominadas membros da **struct**.

Uma struct é um tipo de valor, quando você passa uma struct por valor para um método, o método recebe e opera em uma cópia do argumento struct. O método não tem acesso ao original struct no método de chamada e, portanto, não é possível alterá-lo de forma alguma. O método pode alterar apenas a cópia.

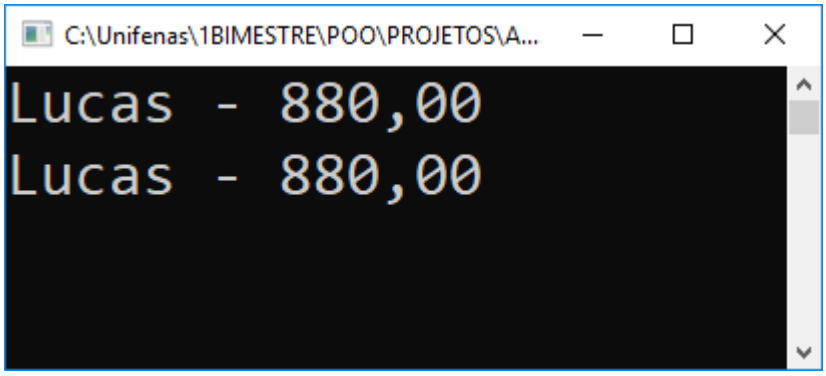
Assim quando for necessário alterar a estrutura original é preciso passar a referência da estrutura.

Veja os exemplos:

Exemplo 1

```
class Program
    {

        struct Funcionario
        {
            public string nome;
            public double salario;
        }
        static void Altera(Funcionario P) //ERRO
        {
            P.nome = "Lara";
            P.salario = 980;
        }
        static void Mostra(Funcionario P)
        {
            Console.WriteLine(P.nome + " - " + P.salario.ToString("0.00"));
        }
        static void Main()
        {
            Funcionario F;
            //Inicializar os campos da struct
            F.nome = "Lucas";
            F.salario = 880.0;
            Mostra(F);
            Altera(F);
            Mostra(F);
            Console.ReadKey();
        }
    }
```



Exemplo 2

```
class Program
    {

        struct Funcionario
        {
            public string nome;
            public double salario;
        }
        static void Altera(ref Funcionario P)
        {
            P.nome = "Lara";
            P.salario = 980;
        }
        static void Mostra(Funcionario P)
        {
            Console.WriteLine(P.nome + " - " + P.salario.ToString("0.00"));
        }
        static void Main()
        {
            Funcionario F;
            //Inicializar os campos da struct
            F.nome = "Lucas";
            F.salario = 880.0;
            Mostra(F);
            Altera(ref F);
            Mostra(F);
            Console.ReadKey();
        }
    }
```