C#-3 - Variables, Data Types and Operators

1. Entrada e saída de dados
2. Variáveis no C#
3. Tipos Wrapper
4. Tipos Primitivos
5. Nomenclatura de variáveis
6. Converter tipos de variáveis

1. Entrada e saída de dados

Comparativos:

Portugol

Entrada de dados

```
leia(valor)
```

Saída de dados

```
escreva(valor)
```

C#

Entrada de dados

```
Console.ReadLine();
```

Saída de dados

```
Console.WriteLine("Hello, World");
```

2. Variáveis no C#

O C# é uma linguagem fortemente tipada. Todas as variáveis e constantes têm um tipo, assim como cada expressão que é avaliada como um valor. Cada declaração de método especifica um nome, o tipo e o tipo (valor, referência ou saída) para cada parâmetro de entrada e para o valor de retorno.

CLASSIFICAÇÃO

- **Tipos Internos:**

Elas representam inteiros, valores de ponto flutuante, expressões booleanas, caracteres de texto, valores decimais e outros tipos de dados. Também há tipos **string** e **object** internos. Esses tipos estão disponíveis para uso em qualquer programa em C#.

  - **Valores**:
    - bool
    - byte
    - sbyte
    - char
    - decimal
    - double
    - float
    - int
    - uint
    - nint
    - nuint
    - long
    - ulong
    - short
    - ushort
  - **Refêrencia**
    - object
    - string
    - dynamic
- **Tipos Personalizados:** Utilizado para criar seus proprios tipos personalizados
  - struct
  - class
  - interface
  - enum
  - record
- **Tipos de valores literais:** recebem um tipo do compilador. Você pode especificar como um literal numérico deve ser digitado anexando uma letra ao final do número. Por exemplo, para especificar que o valor `4.56` deve ser tratado como um `float` , acrescente "f" ou "f" após o número: `4.56f`.
- **Tipos genéricos:** Um tipo pode ser declarado com um ou mais *parâmetros de tipo* que servem como um espaço reservado para o tipo real (o *tipo concreto*). O código do cliente fornece o tipo concreto quando cria uma instância do tipo. Esses tipos são chamados de *tipos genéricos*

Exemplos de declaração de variáveis no C#

- Sem inicialização:

```
string nome;
char sexo;
int idade;
float preco;
double resultado;
bool condicao;
```

- Com inicialização:

```
string nome = "Gustavo Boaz";
char sexo = 'M'
int idade = 31;
float preco = 2.25;
double resultado = 4.568;
bool condicao = true;
```

- Atribuição de métodos:

```
string nome = Console.ReadLine();
char sexo = Convert.ToChar(Console.ReadLine());
int idade = Convert.ToInt32(Console.ReadLine());
float preco = (float) Convert.ToDouble(Console.ReadLine());
double resultado = Convert.ToDouble(Console.ReadLine());
bool condicao = Convert.ToBoolean(Console.ReadLine());
```

- Atribuição por operação:

```
string resultado1 = "Gustavo" + " " + "Boaz";
int resultado3 = 1 + 1
float resultado4 = 2.25 + 3.00
double resultado5 = 4.568 + 7.346
```

Operadores

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

- **Aritméticos** (+, -, *, /, %)
- **Atribuição** (=, +=, -=, *=, /=, %=)
- **Relacional** (==, !=, <, <=, >, >=)
- **Lógico** (&&, | |)

Operadores Aritméticos

```
int resultado1 = 1 + 1; // 2
int resultado2 = 3 * 2; // 6
int resultado3 = 4 / 2; // 2
int resultado4 = 6 % 5; // 1
int x = 7;

x = x + 1 * 2; // x = 9 7 x = x - 3; // x = 6 8
x = x / (6 - 2 + (3*5)/(16-1)); // x = 2
```

Operadores de Atribuição

```
int valor = 1; // valor = 1
valor += 2; // valor = 3
valor -= 1; // valor = 2
valor *= 6; // valor = 12
valor /= 3; // valor = 4
valor %= 3; // valor = 1
```

Operadores Relacionais

```
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
```

Operadores Lógicos

```
int valor = 30;
bool teste = false;

teste = valor < 40 && valor > 20; // teste = true
teste = valor < 40 && valor > 30; // teste = false
teste = valor > 30 || valor > 20; // teste = true
teste = valor > 30 || valor < 20; // teste = false
teste = valor < 50 && valor == 30; // teste = true
```

3. Tipos Wrapper

Wrapper é dado ao nome da classe ao que se esta referenciando o tipo primitivo. A vantagens de se manipular uma classe wrapper é que em algumas linguagens é possível acessar seus métodos.

4. Tipos Primitivos

Tipo primitivo representa uma instancia da classe, é o objeto propriamente dito. Na linguagem C# é mais utilizado para definir o tipo de variavel.

5. Nomenclatura de variáveis

As convenções de codificação atendem às seguintes finalidades:

- Criam uma aparência consistente para o código, para que os leitores possam se concentrar no conteúdo e não no layout.
- Permitem que os leitores entendam o código com mais rapidez, fazendo suposições com base na experiência anterior.
- Facilitam a cópia, a alteração e a manutenção do código.
- Demonstram as práticas recomendadas do C#.

Para compreender mais sobre a nomenclatura no C# acessar a documentação no Link.

6. Converter tipos de variáveis

Ao se manipular uma variável é muito comum ser necessário alterar seu tipo entre outras. Para isso é necessário considerar converter. No exemplo abaixo é feito uma leitura do teclado, no entanto o teclado produz uma string como resultado, caso seja necessário converter para outro tipo de dado é necessário efetuar um **Convert.[Tipo de conversão]**.

```
string nome = Console.ReadLine();
char sexo = Convert.ToChar(Console.ReadLine());
int idade = Convert.ToInt32(Console.ReadLine());
float preco = (float) Convert.ToDouble(Console.ReadLine());
double resultado = Convert.ToDouble(Console.ReadLine());
bool condicao = Convert.ToBoolean(Console.ReadLine());
```

Vale lembrar que existem valores que não possuem um **Convert** adequado, para esses casos é possivel utilizar um **cast**. Consiste em passar um tipo primitivo para outro tipo primitivo, porem de valores correlacionados. ex: float <- double (Numericos)