

Linguagem de Programação 3

Aula 2

Introdução à Linguagem de Programação C#

`l.bertholdo@ifsp.edu.br`

Conteúdo

- Introdução à Linguagem C#
- Tipos de Dados
- Variáveis
- Vetores
- Matrizes
- Operadores
- Estruturas Condicionais
- Estruturas de Repetição

Introdução à Linguagem C#

- A linguagem C# (lê-se “C Sharp”) foi criada pela Microsoft visando o desenvolvimento de aplicações na plataforma .NET.
- É uma linguagem de programação orientada a objetos e fortemente tipada, assim como Java, C e C++.
- Embora tenha sido criada para facilitar o desenvolvimento de aplicações Web, ela também pode ser usada na criação de aplicações *desktop* e *mobile*.

Introdução à Linguagem C#

- Histórico
 - **2000** – A Microsoft apresenta a linguagem C# na *Professional Developers Conference (PDC)*.
 - **2002** – Devido ao crescimento de sua popularidade, a linguagem é incluída na ferramenta de desenvolvimento Visual Studio .NET.
 - **2005** – O sucesso crescente do Visual Studio torna a linguagem C# uma das mais utilizadas no mundo. A versão 2.0 da linguagem é lançada junto do Visual Studio 2005.
 - **2008** – A versão 3.0 é lançada junto do Visual Studio 2008.

Introdução à Linguagem C#

- Histórico
 - **2010** – A versão 4.0 da linguagem C# é lançada junto do Visual Studio 2010.
 - **2012** – A versão 5.0 passa a integrar o Visual Studio 2012.
 - **2015** – A versão 6.0 é lançada com o Visual Studio 2015.
 - A última versão estável da linguagem C# é a 7.3, que está integrada ao Visual Studio 2017.

Tipos de Dados

- Representam o tipo de informação que as variáveis, constantes ou atributos podem armazenar, ou tipo de informação que as funções ou métodos podem receber ou retornar.
- Números inteiros positivos
 - **byte** – Armazena números inteiros entre 0 e 255.
 - **ushort** – Armazena números inteiros entre 0 e 65.535.
 - **uint** – Armazena números inteiros entre 0 e ~4,3 bilhões.
 - **ulong** – Armazena números inteiros entre 0 e ~18,4 quintilhões.

Tipos de Dados

- Números inteiros positivos e negativos
 - **sbyte** – Armazena números inteiros entre -128 e 127.
 - **short** – Armazena números inteiros entre -32.768 e 32.767.
 - **int** – Armazena números inteiros entre $\sim -2,1$ bilhões e $\sim 2,1$ bilhões.
 - **long** – Armazena números inteiros entre $\sim -9,2$ quintilhões e $\sim 9,2$ quintilhões.

Tipos de Dados

- Números inteiros e reais (positivos e negativos)
 - **float** – Armazena números reais entre $\sim 1,5 \cdot 10^{-45}$ e $\sim 3,4 \cdot 10^{38}$, com precisão de até 7 casas decimais.
 - **double** – Armazena números reais entre $\sim 5,0 \cdot 10^{-324}$ e $\sim 1,7 \cdot 10^{308}$, com precisão de até 16 casas decimais.
 - **decimal** – Armazena números reais entre $\sim 1,0 \cdot 10^{-28}$ e $\sim 7,9 \cdot 10^{28}$, com precisão de até 29 casas decimais. É o tipo ideal para representar valores monetários, pois evita erros críticos de arredondamento.

Tipos de Dados

- Caracteres e Cadeias de Caracteres
 - **char** – Armazena um único caractere que pode ser número, letra, espaço, símbolo ou sequência de escape (conjunto de caracteres iniciados por “\”, exemplo: \n). A atribuição deve ser feita sempre com aspas simples. **Exemplo:** `char opcao = 's';`
 - **string** – Armazena cadeias de caracteres que podem conter números, letras, espaços, símbolos ou sequências de escape. A atribuição deve ser feita sempre com aspas duplas. **Exemplo:** `string mensagem = "Opção inválida!";`
- Valores lógicos
 - **bool** – Armazena valores booleanos, sendo utilizado em expressões lógicas. Os valores possíveis são **true** e **false**.

Variáveis

- Uma variável pode ser declarada de três formas distintas:
 - Sem atribuição de valor:
`int idade;`
 - Com atribuição de valor:
`double altura = 1.78;`
 - Em conjunto com outras variáveis (com ou sem atribuição de valores):
`string nome = "João", email, profissao = "Professor";`

Variáveis

- Uma variável pode ser **local** ou **global**.
 - **Local:** Quando é declarada dentro de um método. Nesse caso, pode ser acessada somente neste método.
 - **Global:** Quando é declarada no início da classe e fora de qualquer método. Nesse caso, pode ser acessada a partir de qualquer local da classe.

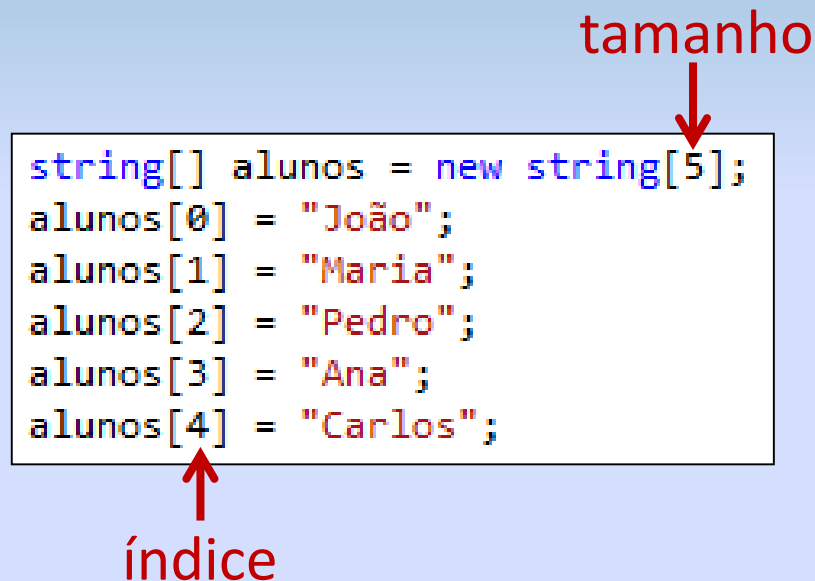
```
class Program
{
    int x=0, y=0; // Variáveis globais

    void Main(string[] args)
    {
        // Variáveis locais
        int cont = x;
        double num = y;
    }
}
```

Vetores

- Vetores ou *arrays* unidimensionais são variáveis que permitem armazenar mais de um valor ao mesmo tempo.
- Cada valor é armazenado em uma posição do vetor, a qual é referenciada por um **índice**.

tamanho



```
string[] alunos = new string[5];  
alunos[0] = "João";  
alunos[1] = "Maria";  
alunos[2] = "Pedro";  
alunos[3] = "Ana";  
alunos[4] = "Carlos";
```

índice

Outra forma de inicialização do vetor:

```
string[] alunos = new string[5] { "João", "Maria", "Pedro", "Ana", "Carlos" };
```

Matrizes

- Matrizes ou *arrays* bidimensionais permitem armazenar de forma tabular mais de um valor ao mesmo tempo.
- Porém, cada posição do vetor é referenciada por dois **índices**, um representando as **linhas** e outro as **colunas** da matriz.

```
string[,] dados_alunos = new string[3,2];
dados_alunos[0,0] = "João";
dados_alunos[0,1] = "joao@gmail.com";
dados_alunos[1,0] = "Maria";
dados_alunos[1,1] = "maria@gmail.com";
dados_alunos[2,0] = "Carlos";
dados_alunos[2,1] = "carlos@gmail.com";
```

linhas colunas

índice da linha índice da coluna

Outra forma de inicialização da matriz:

```
string[,] dados_alunos = new string[3,2] { {"João", "joao@gmail.com"},
                                             {"Maria", "maria@gmail.com"},
                                             {"Carlos", "carlos@gmail.com"} };
```

Operadores Matemáticos

- Operadores para realização de cálculos matemáticos com dados numéricos.

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

```
double a = 5, b = 2;  
double resultado;  
  
resultado = a + b;  
Console.WriteLine(resultado);  
resultado = a - b;  
Console.WriteLine(resultado);  
resultado = a * b;  
Console.WriteLine(resultado);  
resultado = a / b;  
Console.WriteLine(resultado);  
resultado = a % b;  
Console.WriteLine(resultado);
```

Operadores de Comparação

- Operadores para realização de comparações entre os valores de duas variáveis ou de uma variável e um número.

Operador	Descrição
var1 == var2	Verdadeiro se var1 for igual a var2.
var1 != var2	Verdadeiro se var1 for diferente de var2.
var1 > var2	Verdadeiro se var1 for maior que var2.
var1 < var2	Verdadeiro se var1 for menor que a var2.
var1 >= var2	Verdadeiro se var1 for maior ou igual a var2.
var1 <= var2	Verdadeiro se var1 for menor ou igual a var2.

Operadores de Comparação

```
int x = 5, y = 2;

if (x == y)
    Console.WriteLine("Os operandos são iguais.");
else
    Console.WriteLine("Os operandos são diferentes.");

if (x != y)
    Console.WriteLine("Os operandos são diferentes.");
else
    Console.WriteLine("Os operandos são iguais.");

if (x > y)
    Console.WriteLine("O 1º operando é maior que o 2º operando.");
else
    Console.WriteLine("O 1º operando é menor ou igual que o 2º operando.");
```


Operadores Lógicos

- Operadores usados na construção de expressões condicionais.

Operador	Descrição
! exp1	Verdadeiro se exp1 for falsa.
exp1 && exp2	Verdadeiro se exp1 e exp2 forem verdadeiras.
exp1 exp2	Verdadeiro se exp1 ou exp2 forem verdadeiras.

Operadores Lógicos

```
int media = 7, presenca = 80;
if ((media >= 6) && (presenca >= 75))
    Console.WriteLine("O aluno foi aprovado.");
else
    Console.WriteLine("O aluno foi reprovado.");

string nota = "ótimo";
if ((nota == "ótimo") || (nota == "bom"))
    Console.WriteLine("O aluno foi aprovado.");
else
    Console.WriteLine("O aluno foi reprovado.");

string situacao = "reprovado";
if (!(situacao == "aprovado"))
    Console.WriteLine("O aluno foi reprovado.");
else
    Console.WriteLine("O aluno foi aprovado.");
```

Comandos Condicionais

- Permitem avaliar uma expressão e, conforme o resultado obtido, executar um determinado trecho de código.
- São comandos usados em tomadas de decisão dentro de um programa.

Comandos Condicionais – IF

- Um bloco de código **if** pode possuir como complemento blocos **else if** (senão se) e/ou um bloco **else** (senão). Se todos os blocos anteriores forem falsos, o bloco **else** é executado.
- Em um bloco **if**, podem existir somente um **if** e um **else**, mas **else if** podem existir vários.
- Somente um dos blocos (if, else if ou else) pode ser processado em cada execução. Ao processar um bloco, todos os blocos seguintes são ignorados.
- Se houver mais de uma linha de código dentro de um bloco (if, else if ou else), é preciso usar chaves (“{ }”) no início e no fim do bloco.

Comandos Condicionais – IF

```
double prova1 = 7;
double prova2 = 5;
double media = 0;
string desempenho;

media = (prova1 + prova2) / 2;

if (media <= 5)
    desempenho = "INSATISFATÓRIO";
else if (media <= 7)
    desempenho = "REGULAR";
else if (media <= 8.5)
    desempenho = "BOM";
else
    desempenho = "ÓTIMO";

Console.WriteLine("O desempenho do aluno foi " + desempenho);
```

Comandos Condicionais – SWITCH

- Usado para verificar se o conteúdo de uma variável é **igual** a um valor dentre vários possíveis.
- O comando **break** encerra o bloco **switch**, de modo que a execução do programa continue após o bloco.
- A opção **default** tem a mesma função da opção **else** do comando **if**. Se todos os blocos anteriores forem falsos, o bloco **default** é executado.
- Para testar mais de um valor e executar o mesmo código, basta declarar os testes desejados (cláusulas **case**) em sequência e incluir o código a ser executado logo após o último **case**.

Comandos Condicionais – SWITCH

```
string opcao = "sim";

switch (opcao)
{
    case "sim":
        Console.WriteLine("Você escolheu a opção SIM");
        break;
    case "não":
    case "talvez":
        Console.WriteLine("Você não escolheu a opção SIM");
        break;
    default:
        Console.WriteLine("A opção digitada é inválida");
        break;
}
```

Comandos de Repetição

- Permitem executar um trecho de código repetidamente por um determinado número de vezes, ou até que uma condição seja satisfeita.

Comandos de Repetição – WHILE

- Avalia uma expressão e, enquanto a expressão for verdadeira, a execução do bloco de comandos é repetida. Quando a expressão for falsa, o laço de repetição é encerrado e a execução do programa continua após o final do bloco **while**.
- Se houver mais de uma linha de código dentro de um bloco **while**, é preciso usar chaves (“{ }”) no início e no fim do bloco.

```
int cont = 1;

while (cont < 5)
{
    Console.WriteLine("O valor do contador é " + cont);
    cont++;
}
```

Comandos de Repetição – DO/WHILE

- Semelhante ao comando **while**. Porém, como a expressão **while** é avaliada somente no final do laço, o bloco de comandos é executado pelo menos uma vez.

```
int cont = 1;

do
{
    Console.WriteLine("O valor do contador é " + cont);
    cont++;
} while (cont < 5);
```

Comandos de Repetição – FOR

- Usado quando se deseja executar um trecho de código um determinado número de vezes. Exemplo: Imprimir todos ou parte dos elementos de um vetor.
- Para controlar os laços de repetição, o comando **for** utiliza um **índice**.
- O **índice** tem um **valor inicial**, um **valor final** (condição de parada) e um **valor de incremento** ou **decremento**, que aumenta (+1) ou diminui (-1) o **índice** a cada laço de repetição executado.
- Se houver mais de uma linha de código dentro de um bloco **for**, é preciso usar chaves (“{”}”) no início e no fim do bloco.

Comandos de Repetição – FOR

```
for (int cont = 0; cont < 10; cont++)  
    Console.WriteLine("O contador vale " + cont);
```

```
string[,] vetor = new string[2, 2];  
vetor[0, 0] = "A";  
vetor[0, 1] = "B";  
vetor[1, 0] = "C";  
vetor[1, 1] = "D";
```

```
for (int i = 0; i < 2; i++)  
    for (int j = 0; j < 2; j++)  
        Console.WriteLine("O elemento da posição (" + i + "," + j + ") é " + vetor[i, j]);
```

Para cada valor do índice *i* do 1º FOR, serão percorridos todos os valores do índice *j* do 2º FOR.

1º laço: *i*==0 / *j*==0

2º laço: *i*==0 / *j*==1

3º laço: *i*==1 / *j*==0

4º laço: *i*==1 / *j*==1

Comandos de Repetição – FOREACH

- Usado quando se deseja acessar todos os elementos de um vetor ou matriz, não sendo necessária uma condição de parada.
- Os laços de repetição não são controlados por um índice, pois a execução sempre será realizada para todos os elementos.

```
int[] elementos = new int[5]{1, 2, 3, 4, 5};  
  
foreach (int x in elementos)  
    Console.WriteLine("O valor do elemento é " + x);
```

Comandos de Repetição – BREAK

- Usado quando se deseja interromper a execução de um bloco de repetição.
- Quando o programa lê o comando **break**, a execução do bloco é interrompida independentemente da condição de parada do comando de repetição.

```
int i = 1;

while (i < 10)
{
    if (i == 5)
        break;

    Console.Write(i + ", ");
    i++;
}
```



1, 2, 3, 4,

Comandos de Repetição – CONTINUE

- Usado quando se deseja ir para o próximo laço de repetição de um bloco, sem que o laço atual tenha chegado ao final.
- Quando o programa lê o comando **continue**, a execução do laço atual é interrompida e um novo laço de repetição é iniciado.

```
int i = 1;

while (i < 10) {
    if (i == 5) {
        i++;
        continue;
    }
    Console.Write(i + ", ");
    i++;
}
```



1, 2, 3, 4, 6, 7, 8, 9,

Referências

- Henrique Loureiro; C# 6.0 com Visual Studio – Curso Completo. FCA, 2015.
- John Sharp; Microsoft Visual C# 2013: Passo a Passo. Bookman, 2014.