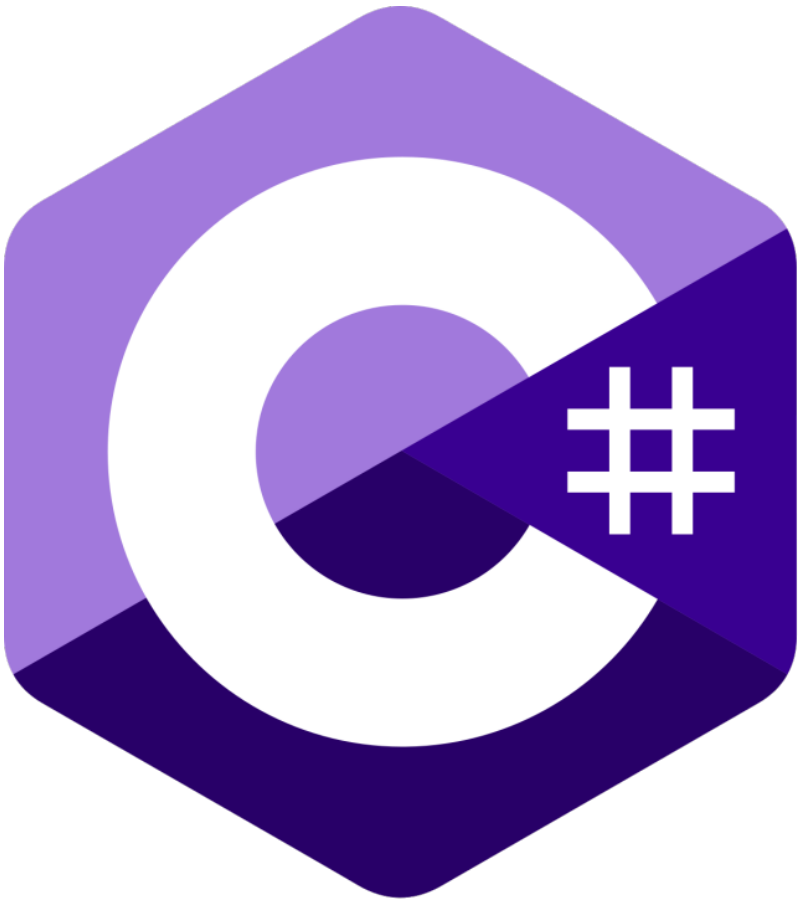
# Apostila de Apoio - Introdução à Lógica de Programação com C#



Autores:

- João Pedro Prado da Costa (28167333)

- Raphael Lins (27797660)

- João Gabriel Kreimer (28017188)

**Introdução**

Bem-vindo à apostila de apoio para estudantes de Ciência da computação. Aqui, você aprenderá os conceitos fundamentais de programação utilizando a linguagem C#, com exemplos práticos e explicações.

---

**Origem da Linguagem C#**

A linguagem C# foi desenvolvida pela Microsoft no final dos anos 90, sob a liderança de Anders Hejlsberg. Ela surgiu como parte da plataforma .NET, com o objetivo de ser uma linguagem moderna, orientada a objetos e voltada para o desenvolvimento de aplicações robustas e eficientes. Inspirada em linguagens como C++ e Java, o C# foi projetado para oferecer uma sintaxe clara, segurança na manipulação de memória e integração nativa com o ecossistema da Microsoft. Atualmente, o C# é amplamente utilizado no desenvolvimento de aplicativos desktop, web e jogos, especialmente com o framework .NET e o motor de jogos Unity.

---

**1. Fundamentos da Programação**

A programação é a arte de escrever instruções para um computador executar. Em C#, um programa básico segue a seguinte estrutura:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

**Estrutura Básica de um Programa C#**

A estrutura acima segue um padrão essencial para qualquer aplicação em C#:

- using System; → Importa a biblioteca System, que permite a utilização de funções básicas como entrada e saída de dados.
- class Program → Define uma classe chamada Program, que serve como ponto de entrada do programa.
- static void Main() → O método Main é o ponto de partida da execução. Todo programa C# precisa ter um método Main.
- Console.WriteLine("Hello, World!"); → Exibe uma mensagem no console.

**Fluxo de Controle**

O fluxo de controle define a ordem em que as instruções de um programa são executadas. Em C#, podemos utilizar:

- **Sequência**: As instruções são executadas em ordem.
- **Decisão**: Utiliza estruturas como if, else e switch.
- **Laços de repetição**: for, while, do-while.

Exemplo de estrutura condicional:

```
int idade = 18;
if (idade >= 18)
{
    Console.WriteLine("Você é maior de idade.");
}
else
{
    Console.WriteLine("Você é menor de idade.");
}
```

Exemplo de laço de repetição:

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Iteração: " + i);
}
```

Exemplo de laço while:

```
int contador = 0;
while (contador < 5)
{
    Console.WriteLine("Contador: " + contador);
    contador++;
}
```

---

**2. Tipos de Dados**

Os tipos de dados em C# definem o formato e a quantidade de memória alocada para armazenar informações. Alguns exemplos:

| Tipo | Descrição | Exemplo |
|---|---|---|
| int | Números inteiros | int idade = 25; |
| double | Números decimais | double altura = 1.75; |
| string | Textos | string nome = "Maria"; |
| bool | Valores lógicos (true/false) | bool ativo = true; |

Exemplo prático:

```
int quantidade = 10;
double preco = 5.99;
bool disponivel = true;
string produto = "Caderno";
Console.WriteLine($"Produto: {produto}, Preço: {preco}, Disponível: {disponivel}");
```

---

### 3. Estruturas de Dados

As estruturas de dados são componentes essenciais para armazenar, organizar e manipular coleções de valores de maneira eficiente, permitindo que os programas lidem com grandes quantidades de informações de forma estruturada e otimizada. Em C#, duas das principais estruturas de dados utilizadas são **arrays**, que fornecem um meio fixo e indexado de armazenamento, e **listas**, que oferecem flexibilidade ao permitir a adição e remoção dinâmica de elementos.

#### 3.1 Arrays

Um **array** é uma coleção de elementos do mesmo tipo, acessados por um índice.

**Exemplo de um array de inteiros:**

```
int[] numeros = { 10, 20, 30, 40, 50 };
Console.WriteLine(numeros[2]); // Saída: 30
```

**Criando um array com tamanho fixo:**

```
int[] notas = new int[5];
notas[0] = 7;
notas[1] = 8;
notas[2] = 9;
Console.WriteLine("Nota da primeira prova: " + notas[0]);
```

#### 3.2 Listas

As **listas** em C# são coleções dinâmicas. Elas permitem adicionar, remover e acessar elementos de forma eficiente, com o tamanho ajustando-se automaticamente conforme necessário. Diferente dos **arrays**, as **listas** oferecem maior flexibilidade para manipulação de dados em tempo de execução.

**Criando e manipulando uma lista de strings:**

```
using System;
using System.Collections.Generic;

List<string> nomes = new List<string>();
nomes.Add("Alice");
nomes.Add("Bob");
nomes.Add("Carlos");

Console.WriteLine("Primeiro nome: " + nomes[0]); // Saída: Alice
```

**Removendo um item da lista:**

```
nomes.Remove("Bob");
Console.WriteLine("Tamanho da lista: " + nomes.Count);
```

Listas oferecem diversos métodos úteis, como:

- Add(item): Adiciona um item à lista.

```
List<int> numeros = new List<int>();
numeros.Add(10);
```

- Remove(item): Remove um item específico.

```
List<int> numeros = new List<int> { 10, 20, 30 };
numeros.Remove(20);
```

- Count: Retorna o número de elementos na lista.

```
List<int> numeros = new List<int> { 10, 20, 30 };
Console.WriteLine(numeros.Count); // Saída: 3
```

- Contains(item): Verifica se um item está na lista.

```
List<int> numeros = new List<int> { 10, 20, 30 };
Console.WriteLine(numeros.Contains(20)); // Saída: True
```

---

**Exercícios propostos:**

1. Crie um programa que receba o nome e a idade do usuário e exiba uma mensagem personalizada.
2. Peça ao usuário dois números e exiba a soma, subtração, multiplicação e divisão deles.
3. Crie um programa que determine se um número é par ou ímpar.
4. Crie um array de cinco números inteiros e exiba a soma deles.
5. Crie uma lista de nomes e permita que o usuário adicione e remova nomes interativamente.

**Bons estudos!**

*"O conhecimento torna a alma jovem e diminui a amargura da velhice. Colhe, pois, a sabedoria. Armazena suavidade para o amanhã."*
*Leonardo DaVinci*