C# - FUNDAMENTOS

Processo de compilação e execução usando CLR

C# : é uma linguagem tipada, compilada e gerenciada.

- Compilação: Código C# → Compilador C# → IL Code
- Execução: IL Code → JIT compiler → NativeCode
- IL (Intermediate Language) é o código de linguagem intermediária em bytecode.
- IL Code é gerado como Teste.exe e Teste.dll.

Revisão: O compilador C# gera o código IL, enquanto o JIT compiler converte o IL para código nativo, que é gerenciado pelo CLR e executado pelo processador.

SINTAXE C

// → Comentário em uma linha

//** → bloco de comentario

{} → Delimitador de bloco de comandos

; → todos os comandos “simples” terminam com ;

= → atribuição

== → Comparação de igualdade

Organização de Código

- **Projeto** → um projeto pode ter varias classes, uma classe com uma responsabilidade usamos namespaces para organizar as classes
- **Namespaces** → containers lógicos para as classes e outros namespaces

- **Namespaces →** usamos namespaces para agrupar as classes relacionadas

Exemplo de Namespaces com mesma classe:

```
Fornecedores.Cadastro

Produtos.Cadastro

Categorias.Cadastro
```

- **Utilizando a notação ponto** → '.' eu posso referenciar uma classe a um namespaces, posso ter varias classes Cadastro, mas cada cadastro vai pertencer a um namespaces especifico

sintaxe:

```
nome_do_namespace.nome_da_classe
```

outros maneiras:

```
nome_do_namespace.nome_do_namespace.nome_da_classe
```

- posso ter namespaces dentro de namespaces
- **using →** uso para simplificar os namespaces

Ex:

```
using nome_do_namespace

nome_da_classe
```

- **solution →** container para projetos criados no visual studio
- **solution →** pode conter um ou mais projetos

Estruturas de pastas

Dentro do arquivo de solução:

Arquivo de solução : mantém a informação sobre a organização dos projetos na solução

Dentro da pasta de projeto:

- **BIN** → Contém arquivos binários, que são o código executável real para seu aplicativo ou biblioteca
- **OBJ** → Contém arquivos objeto, ou intermediários, que são arquivos binários compilados que ainda não foram vinculados
- **Arquivo de projeto** → Contém informações sobre os arquivos incluídos no projetos, assemblies usados, GUID do projeto e versão do projeto

Principais Comandos

- **dotnet –info** → lista informações detalhadas do sistema
- **dotnet –version** → exibe a versão do .NET SDK atual
- **dotnet –list-sdks** → exibe a lista dos sdks instalados
- **dotnet –list-runtimes** → exibe a lista dos runtimes instalados
- **dotnet new list** → Lista todos os templates de projetos
- **dotnet new** → cria um projeto usando o template informado
- **dotnet run** → Executa projetos
- **dotnet restore** → Restaurar os pacotes do projeto
- **dotnet test** → Executa projetos de teste e testes de unidade
- **dotnet publish** → usado para publicar projetos
- **dotnet new sln** → cria uma nova solution
- **dotnet sln add/remove** → Adiciona/Remove projetos para solução

- **dotnet add/remove reference** → adiciona/remove referência de projetos para outros projetos
- **dotnet add/remove package** → adiciona referência de pacotes Nuget para um projeto

---

Instalando um projeto com uma versão especifica

- crie uma pasta
- dentro da pasta rode o comando:
- sintaxe:
  dotnet new globaljson –sdk-version numeroversaosdk –force
- exemplo:
  dotnet new globaljson –sdk-version 7.0.1 –force

  dentro da mesma pasta criar seu projeto

Instalando projeto com uma versão especifica 2

- **dotnet new console -o MeuApp2 -f net5.0** → criando o projeto e especificando a versão
- **dotnet new console -o MeuApp3 –use-program-main** → criar um projeto no dotnet atual, porem usando o método main, a estrutura de codificação abaixo do net6.0, sem top level stateman

---

Criar projeto e solução igual ao visual studio

Observação: crie uma pasta primeiro e faça tudo dentro dela

- criar solução
  dotnet new sln -o <nome solução>

- criar projeto dentro de solução
  dotnet new <nome_do_template> -o <nome_projeto>
- incluir o projeto criado na solução existente(a partir da pasta da solução)
  dotnet sln <nome_da_solucao>.sln add <pasta_projeto>/<arquivo>.csproj

start minhaSolucao.sln → abrir solução com projeto dentro

Atalhos C#

- **CTRL + F5** → Executar codigo sem depuração
- **F5** → Executar codigo com depuração]
- **CTRL + K + D** → Organizar código
- **CTRL + D** → Duplicar linha onde cursor esta
- **CTRL + F** → Localizar algo no documento atual

Tipos de dados de referência e de valor

Tipos de referência → Não armazenam os dados diretamente, cada variável contém uma referência ao local da memória onde os dados estão dados estão armazenados

Tipos referencia → class,object,string

Tipos de valor → Armazenam diretamente seus dados e cada variável tem sua própria cópia dos dados

Tipos valor → numericos,char,bool,struct

Os tipos de dados → são armazenados na memória Stack(LIFO - Last in First Out)

Sintaxe declaração:

```
tipo nomeVariavel = valor;
tipo nomeVariavel; → não inicializado
```

Tipos pré-definidos

- Tipos numéricos de ponto flutuante
- palavra-chave/tipo c# ,Tamanho, Tipo .NE
- float 4bytes System.Single
- double 8bytes System.Double
- decimal 16bytes System.Decimal

Representações:

double valor = 12.4; float valor = 12.4F; decimal valor = 12.4M; System.Double valor = 12.4; System.Single valor = 12.4f; System.Decimal valor = 12.4m;

Caracteristicas

- Armazenados na Stack
- Valor padrão 0
- Suporte a operadores aritméticos de comparação → (>,<,>=,<=,!=) e (==)
- double → para cálculos científicos
- decimal → para cálculos financeiros

Comandos

- Console.WriteLine(); → mostrar um valor na tela do terminal
- Console.ReadLine(); → Espera que o usuário digite para terminar o codigo no terminal

Tipos pré-definidos : Tipos não numéricos

- **valor padrão do bool →** false
- **valor padrão do tipo char é** → '\0'
- **(U + 0000) →** NULL
- **bool →** true or false **→** 8 bits **→** System.Boolean

- **char** → U+0000 A U+FFFF → 16 bits → System.Char
- **’’** → aspas simples char
- **“”** → aspas duplas string

Ex:

- bool ativo = true; char letra = ‘A’;
- System.Boolean ativo = true; System.Char letra = ‘A’;

Tipos de referência

- **string** → armazenamento de zero ou mais caracteres System.String
- **object** → é o tipo base para todos os outros tipos System.Object
- **dynamic** → resolvido em tempo de execução System.Object
- **dynamic** → util para reflection e dlr
- **string são imutáveis**

```
string valor = “Isto é uma string”;
valor = “Isto é uma string alterada”; → 3 espaços da memoria
valor = “teste”;
```

→ o valor não é reatribuído , ele cria espaço na memoria para cada atribuição ao valor que fiz

Ex:

```
string nome = "Curso Csharp Essencial";
System.String nome = "Curso Csharp Essencial";

object nota = 10;
object valor = 8.55m;
object nome = 'alife';
```

DATE TIME

Característica:

- tipo valor
- valor padrão : 01/01/0001 00:00:00
- **dd/mm/aaaa hh:mm:ss →** pega o formato de data do seu sistema operacional

Ex:

```
DateTime data = new DateTime(2022,09,04)

// new → para criar um data especifica

//sintaxe para definir data e hora – > (aaaa,mm,dd,hh,mm,ss)

Ex:
DateTime data = new DateTime(2022,09,04,11,10,20)

//Pegar a data e o horario atual :

DateTime data = DateTime.Now; → log : 24/10/2023 21:00:44

//Sem especificar horario:

DateTime dataHoje = new DateTime(2023, 10, 24);
Console.WriteLine(dataHoje); → log : 24/10/2023 00:00:00

//Especificando data e horario

DateTime dataHoraHoje = new DateTime(2023 , 10 , 24, 00, 23,01);
Console.WriteLine(dataHoraHoje);
```

Operações com data e hora

- **1 - Extrair informações como dia, mês, hora , ano, etc.**
- Year,Month, Day,Hour,Minute,Second,Millisecond
- **2 - Adicionar dias,horas,mês,anos,etc.**
- AddDays,AddHours,AddMonths,AddYears

- **3 - Obter dia da semana e do ano**
- DayOfWeek,DayOfYear
- **4 - Expressar data no formato longo e abreviado**
- ToLongDateString, ToShortDateString
- **5 - Expressar hora no formato logo e abreviado**
- ToLongHourString, ToShortHourString

Sintaxe :

```
DateTime hoje = DateTime.Now;
Console.WriteLine($“hoje : {hoje}”);
// extrair informações da data atual
Console.WriteLine(hoje.Year);
Console.WriteLine(hoje.Month);
Console.WriteLine(hoje.Day);
Console.WriteLine(hoje.Hour);
Console.WriteLine(hoje.Second);
Console.WriteLine(hoje.Millisecond);

//adicionando valores
Console.WriteLine(hoje.AddDays(20));
Console.WriteLine(hoje.AddMonths(3));
Console.WriteLine(hoje.AddHours(2));
Console.WriteLine(hoje.AddYears(5));

//obter o dia da semana e do ano
Console.WriteLine(hoje.DayOfWeek);
Console.WriteLine(hoje.DayOfYear);

// data no formato longo e curto
Console.WriteLine(hoje.ToLongDateString());
Console.WriteLine(hoje.ToShortDateString());
```

Nullable Types

Tipos de valor:

- **As variaveis do tipo : numéricos,char,bool,struct** → sempre possuem um valor e são armazenados na memória Stack
- **variável de um tipo:** por valor, não pode conter o valor null(nulo)
- **valor padrão:** de um tipo valor é (zero)

Ex:

int valor = 10;

```
      Stack          Heap
    valor = 10
```

```
int valor = null;  Error → Cannot convert null to 'type' because it is a non-nullal
```

Nullable Types ou Tipos Anuláveis

Importância :

- **Usado:** para representar um valor indefinido de um tipo, quando um tipo sem valor for valido para o meu contexto
- **Usado:** em banco de dados quando um valor de variável for indefinido ou ausente
- **Nullable Type →** é um tipo de valor que pode receber um valor null
- **Nullable Types →** ou Tipos Anuláveis permitem atribuir um valor null a um tipo de valor

Sintaxe:

```
Nullable<T><nome> = null; (T = int,double,float,bool,etc.)
```

Os Nullables Types suportam os valores do tipo mais o valor null

Exemplo:

```
Nullable<bool>b= null; //suporta true,false e null
```

Simplificando

```
int? i = null;
double? d = null;
bool? b = null;
```

- Nullable Types são diferentes do tipos por valor
- Nullable Type int !== int
- **int** é um tipo não anulável ou **Non-Nullable Type**
- **int?** é um NullableType

```
int? a = null;
int b = a;
Console.WriteLine(b);

Error: Cannot implicity convert type int? 'to 'int'. An explicit conversion exists(a
```

Solução:

- **??** → coalescência nula
- **??** → usar para atribuir um valor de tipo anulável a um não anulável

```
int ? a = null;

int b = a ?? 0; ⟶ se 'a' for nulo ele atribui 'a' se não ele atribui zero

Console.WriteLine(b);
```

Exemplos:

Problema

```
int? x = 4;
int y = 3;
int z = x * y;
```

```
//Error: Cannot implicity convert type int? ‘to ’int’. An explicit conversion exists

//Minha solução
int? h = 4;
int j = 3;
int z = (h * j) ?? 0;
Console.WriteLine(z);

//Solução Aula
int? h = 4;
int? j = 3;
int? z = h * j;
Console.WriteLine(z);
```

Propriedades somente leitura: HasValue e Value

- **São usadas** para examinar e obter um valor de uma variável de **Nullable Type**
- **HasValue:** true se tiver um valor, false se não tiver um valor(null)
- **Value:** Exibe o valor atribuído

```
int? b = 100;
if(b.HasValue)
{
Console.WriteLine($“b = {b.value}”)
}
else
{
Console.WriteLine(“b não possui valor”);
}
```

Convenções

CamelCase

- Ex: valorDoDesconto , nomeCompleto
- **Usado** para nome de **variáveis**,**parâmetros** e campos **internos** e **privados**.

Pascal Case

- Ex : CalculaImpostoDeRenda, ValorDoDesconto, NomeCompleto
- **Usado** para classes,métodos,interfaces,proprieadades

Constantes - Usar letras maiúsculas.

- **Ex:** PI,DESCONTO,VALOR,IMPOSTO
- **_ (Sublinhado) —**> usado para campos internos privados e somente leitura (camel case)
- Ex : _valorTotal, _calculoImposto, _precoComDesconto

Identificadores válidos:

```
string nome;
string nomeCompleto;
int idade;
int _valor;
int idade1;
```

identificadores inválidos:

```
int 5idade;
int $valor;
int valor#total;
string nome Completo;
```

Para nome de variáveis: CamelCase:

```
string descontoTotal;
string desconto_Total;
```

Constantes : maiúsculas:

```
double PI = 3.14;
string PREFIXO = “11”;
string PREFIXO_SP = “11”;

Console.ReadLine();

- para nomes de classe é métodos: pascal case
class ImprimirTexto
{
public void ImprimeNome()
{
Console.WriteLine(‘Macoratti’);
}
}
```

Saída de dados

```
Console.WriteLine(); → Exibe um conteudo com quebra de linha
Console.Write(); → Exibe conteudo sem quebra de linha

Console.WriteLine();
Console.WriteLine(“————-CONCATENAÇÃO————”);

//Escreve na mesma linha ‘Maria tem 25 anos’
// usar a concatenação : usando o operado +
Console.WriteLine(nome + ” tem ” + idade + ” anos “);
Console.Write(nome +” tem ” + idade + ” anos “);

Console.WriteLine();
Console.WriteLine(“————-INTERPOLAÇÃO————”);
//Escreve na mesma linha ‘Maria tem 25 anos’
// usar a interpolação de srtings : $ → interpolação {}
Console.WriteLine($"{nome} tem {idade} anos");
Console.Write($“{nome} tem {idade} anos”);

Console.WriteLine();
```

```
Console.WriteLine(“————-PLACEHOLDER————”);
// usar place holders : usa {} com numeração com inicio em zero
// numeração indica quem sera mostrado primeiro
Console.WriteLine(“{0} tem {1} anos”,nome,idade);
Console.Write(“{0} tem {1} anos”, nome, idade);
```

SEQUENCIA DE ESCAPES

Combinação de caracteres que ajudam a especificar uma string

```
string local = “c:\dados\poesias.txt”;
string frase = “Ele falou:"Não fui eu" ”;
string pizza = “”;
string bolo = “”;

Console.WriteLine(frase);
Console.WriteLine(pizza);
Console.WriteLine(bolo);
Console.ReadLine();

//Formatação usando sequencias Escapses
/*

 \a   → sinal sonoro (alerta)
 \b   → Backspace
 \f   → Alimentação de Formulário
 \n   → Nova Linha
 \r   → Carriage Return
 \t   → Tabulação horizontal
 \v   → Tabulação vertical
 \'  → Aspas Simples
 \"  → Aspas duplas
 \\  → Barra invertida
 \?  → Interrogação
 \u ooo → ASCII unicode
 \x hh → ASCII hexadecimal
```

```
*/
```

Conversões entre tipos

- **C# →** Estaticamente tipado em tempo de compilação
- Após uma variável ser **declarada** ela **não** pode ser **redeclarada**
- **Não** pode armazenar **valores** de outros **tipos de dados**
- A menos que o tipo da **variável possa ser convertido**

Conversão Implícita

```
// O C# faz a conversão automaticament caso os tipos seja compativeis

//Conversão Explícita
// A conversão tem que ser feita manualmente

// A conversão automatica ocorre do menor tipo tamanho para uma maior tipo
// O oposto não e verdadeiro

// int → double = ok
// double → int = error

//byte → 1 byte
//short → 2 bytes
//int → 4 bytes
//float → 4 bytes
//long → 8 bytes
// double → 8 bytes
// decimal – 16 bytes

//Conversões implicitas
int varInt = 100;
double varDouble = varInt;
Console.WriteLine(varDouble);
```

```
int numeroInt = 2145678;
long numeroLong = numeroInt;
float numeroFloat = numeroInt;
decimal numeroDecimal = numeroInt;
double numeroDouble = numeroInt;

Console.WriteLine(numeroInt);
Console.WriteLine(numeroLong);

Console.WriteLine(numeroFloat);
Console.WriteLine(numeroDouble);
Console.WriteLine(numeroDecimal);
```

Conversões Explicitas

```
Console.WriteLine("Conversões explicitas");
// Conversão de maneira manual
// Necessita de cast
// Cast definição do tipo a ser convertido usando (tipo) antes da variavel
double varDouble2 = 12.456; // 8 bytes
int varInt2 = (int)varDouble2; // 4 bytes (perda de precisão)
Console.WriteLine(varInt2);
//O resultado seria 2 pq foi convertido de flot para int
// usando o cast o resultado sera mostrando em float, 2,5
int num1 = 10;
int num2 = 4;
float resultado = (float)(num1 / num2);
Console.WriteLine(resultado);
Console.ReadLine();
```

Conversão de tipos 2

```
// conversão de tipos
// conversão para string
// ToString() → converte para string

int valorInt = 123;
double valorDouble = 12.45;
```

```
decimal valorDecimal = 12.5678m;

string s1 = valorInt.ToString();
string s2 = valorDouble.ToString();
string s3 = valorDecimal.ToString();

Console.WriteLine(s1);
Console.WriteLine(s2);
Console.WriteLine(s3);

Console.WriteLine("————————————————--");
```

Conversão de tipos usando a classe Convert()

```
ToBoolean() //→ converte um tipo para valor booleano
ToDouble() //→ converte um tipo para o tipo double
ToInt16() //→ converte um tipo para o tipo 16-bit
ToInt32()// → converte um tipo para o tipo 32-bit

int valorInt2 = 10;
double valorDouble2 = 5.35;
bool valorBoolean2 = true;

Console.WriteLine(Convert.ToString(valorInt2));
Console.WriteLine(Convert.ToDouble(valorInt2));
Console.WriteLine(Convert.ToString(valorBoolean2));
Console.WriteLine(Convert.ToInt32(valorDouble2));
Console.ReadLine();
```

Erros de conversão

```
int varInt = 10000;
Console.WriteLine(Convert.ToByte(varInt));
// Error → System.OverflowException
// 'Value was either too large or too small for an unsigned byte'
// Estou tentando converter um int com intervalo de -2.147.483.648 a 2.147.48
// Para byte que é de 0 a 255
```

Entrada de dados

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("## Entrada de dados");

//ReadLine(): le a linha digitada e retorna o que foi digitado
//Read(): le o que foi digitado(um unico caractere) e retorna o valor em ASCII
//ReadKey():le apenas um caractere, usando para segurar a tela até que o usu

Console.WriteLine("Informe o seu nome");

string nome = Console.ReadLine() ?? "";

Console.WriteLine($"seu nome é {nome}");

Console.WriteLine($"Informe sua idade {nome}");

int idade = Convert.ToInt32(Console.ReadLine());

Console.WriteLine($"A idade de {nome} é {idade}");

Console.ReadKey();
```

Operadores aritméticos

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Operadores Aritimeticos ");

Console.WriteLine("Informe o valor de x");
int x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Informe o valor de y");

int y = Convert.ToInt32(Console.ReadLine());
```

```
const double PI_NUMBER = Math.PI;
const double E_NUMBER = Math.E;

Console.WriteLine($"\n Raiz Quadrade de x = {Math.Sqrt(x)}");
Console.WriteLine($“Potencia de x = {Math.Pow(x,y)}”);
Console.WriteLine($"\n Valor minimo entre x e y = {Math.Min(x,y)}");
Console.WriteLine($“Valor Maximo entre x e y = {Math.Max(x,y)}”);
Console.WriteLine($"\n Conseno de x = {Math.Cos(x)}");
Console.WriteLine($“Seno de x = {Math.Sin(x)}”);
Console.WriteLine($“Exponencial de x = {Math.Exp(x)}”);

Console.ReadKey();

//Operações basicas

//Console.WriteLine($"Soma de x+y = {x+y}");
//Console.WriteLine($“Subtração de x-y = {x-y}”);
//Console.WriteLine($"Multiplicação de x*y = {x*y}");
//Console.WriteLine($“Divisão de x/y = {(double)x/y}”);
//Console.WriteLine($“Módulo de x%y = {x%y}”);
```

Inferência tipos

```
Console.WriteLine(“## Inferencia tipos ##”);

var idade = 25;
var nome = “Maria”;
var salario = 2500.00m;

os tipos serão definidos pela propria linguagem

Console.WriteLine($“Maria tem {idade} anos e ganha {salario.ToString(”c”)}”);

// var limitações
var sal = null; // → não é possivel atribuir um valor null
// →Cannot assign ‘expression’ to an implicitly typed local
```

```
var titulo; // → não é possivel só declarar a chave
// → Implicitly typed locals must be initialized

var salario, imposto, total; // → não é possivel encadear variaveis
// → Implicitly-typed variables cannot have multiple declarators.
// → Implicitly typed locals must be initialized

// não posso mudar o tipo apos inicializar
//Cannot implicitly convert type 'type' to 'type'
var num = 10;
num += 20;
num = "teste";

// posso utilizar o recurso na criação de classes
var test = new Test();
test.MeuMetodo();
class Test
{
public void MeuMetodo()
{
Console.WriteLine("Meu Método");
}
}

// # USOS DO VAR
// sugar syntax
// usado para declarar tipos anonimos
// usado em lações for e foreach
// usada em instruções using
```

Operadores de atribuição

```
var x = 10;
var y = "123";
```

```
y += “456”;
Console.WriteLine($"Valor inicial de x={x}");
Console.WriteLine($“x+=5 ⟹ {x+=5}”);
Console.WriteLine($"x-=5 ⟹ {x-=3}");
Console.WriteLine($“x=4 ⟹ {x=4}”);
Console.WriteLine($"x/=5 ⟹ {x/=5}");
Console.WriteLine($“x%=5 ⟹ {x%=5}”);
Console.WriteLine(y);

Console.WriteLine(“Operadores de atribuição”);

// usando operadores atribuição com tipos numéricos
```

Constantes

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine(“Constantes”);

// Constantes são valores imutaveis que são conhecidos em tempo de compp

// usam a palavra reservada const

// constantes são inicilizadas em caixa alta

// obrigatirio chave/valor

const int ANO = 12;

const int MES = 30, SEMANA = 7, QUINZENA = 15;

const int MESES_ANO = 12;
const int DIAS_ANO = 365;

const float DIAS_POR_MES = (float)DIAS_ANO / (float)MESES_ANO;

double raio, perimetro, area;
```

```
Console.WriteLine(“Informe o raio do circulo :”);
raio = Convert.ToDouble(Console.ReadLine());

perimetro = 2 * Math.PI * raio;
area = Math.PI * Math.Pow(raio,2);

Console.WriteLine($"Perímetro = {perimetro}");
Console.WriteLine($“Área = {area}”);
```

Operadores incrementais e decrementais

```
Console.WriteLine(“Operadores incrementais e decrementais”);

//Objetivo = Aumentar ou Diminuir exatamente em uma unidade o valor de um

//Operador de incremento : ++

//Operador de Decremento : –

// Operador Exemplo Significado
// ++ x++; ou ++x; x = x + 1;
// – x–; ou –x; x = x - 1

// Operador de Incremento : ++
// pré-incremento : ++x
// pós-incremnto : x++

// Operador de Decremento : –
// pré-decremento : –x
// pós-decremnto : x–

Console.WriteLine(“Pós-incremento”);
// primeiro resolve a expressão depois incrementa
int x = 0;
int resultado1 = x++ + 10;
// x = 1 depois da resolução da expressão
```

```
Console.WriteLine($“resultado ${resultado1}”);

Console.WriteLine(“Pré-incremento”);
//primeiro incrementa depois resolve a expressão
int x2 = 0;
// x = 1 antes da resolução da expressão
int resultado2 = ++x2 + 10;
Console.WriteLine($“resultado ${resultado2}”);

// mesma logica para o decremento –
```

Operadores relacionais

```
Console.WriteLine(“Operadores Relacionais ”);

/
*== → igualdade
> → maior que
< → menor que
>= → maior que
<= → menor que
!= → diferente*
/

int x = 10;
int y = 20;

Console.WriteLine($"Valor de x {x}");
Console.WriteLine($“Valor de y {y}”);
Console.WriteLine(x == y);
Console.WriteLine(x > y);
Console.WriteLine(x < y);
Console.WriteLine(x >= y);
Console.WriteLine(x <= y);
Console.WriteLine(x != y);
// vale tanto para numeros quanto para strings
// equals → verifica a igualdade entre duas strings e numeros
```

```
// equals é igual a ==
Console.WriteLine("————————————-");
string a = "curso";
string b = "Curso";
int n1 = 1;
int n2 = 2;
string t = "1";
int t2 = 1;
Console.WriteLine(t.Equals(t2));
Console.WriteLine(a.Equals(b)) ;
Console.WriteLine(n1.Equals(n2));
```

Operadores logicos

```
/*

&& → and
|| → or
! → not

- /

bool c1 = 5 >= 7;
bool c2 = 9 != 8;
bool resultado;
Console.WriteLine($"c1 = {c1}");
Console.WriteLine($"c2 = {c2}");

// operador AND → &&
resultado = c1 && c2;
Console.WriteLine("Operadodor AND(&&) :"+resultado);
resultado = c1 || c2;
Console.WriteLine("Operadodor OR(||) :"+resultado);
resultado = !(c1||c2);
Console.WriteLine("Operadodor NOT(!) :"+resultado);
```

```
Console.ReadKey();
```

Associatividade e precedência

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Precedencia e Associatividade");

/*

Operadores Ordem Decrescente de Precedência Associatividade
[] e (),++,– (sufixo) a++ ou a– Da esquerda para a direita
!,++,– (prefixo) ++a , –a, Da direita para a esquerda
Aritimeticos *,/,%,+,- Da esquerda para a direita
Relacionais <,>, <=,>=, == , != Da esquerda para a direita
Lógicos &&,|| Da esquerda para a direita
Atribuição =,* =, /=. %=, += , -= Da direita para a esquerda

*/
//exemplo 1
//int x = 10 - 2*  3;
//bool b = !(9 != 8) && 5 > 7 || 8 > 6;

//Console.WriteLine(b);
//Console.WriteLine(x);

//exemplo 2
int a = 5, b = 6, c = 4;
int r = –a * b - ++c; // 19
Console.WriteLine(r);
Console.ReadKey();
```

Nullable reference Types

- **Objetivo** → Minimizar a chance de seu aplicativo lançar um error/alerta → System.NullReferenceException quando executado

```
string nome = null; //– alerta —> Converting null literal or possible null value to
Console.WriteLine(nome);
Console.WriteLine(nome.ToUpper());—> System.NullRefenceException : 'Objec
```

- **salvação —>** Nullable Reference Types

recapitulação abaixo :

- Um tipo de referência pode não ter nenhuma referencia
- Isso é expresso com o valor null (nulo)
- Se uma variável de um tipo de referência for igual a null ela não tem nenhuma referência
  a um valor na memória heap
- isso também ocorre quando declaramos a variável de um tipo de referencia mas não atribuímos um valor a ela
- valor padrão para uma variável de referencia é null

Nullable Reference Types - Evitar o NullReferenceException

- **Usar ? →** ao atribuir o valor null
- **Usar ? →** ao acessar a referencia
- **? significa →** Nullable
- **?. significa →** Null Conditional Operator

exemplo :

```
Console.WriteLine("Nullable Refence Types");

string? nome = null; // ou poder ser isso → string? nome = "";

Console.WriteLine(nome?.ToUpper()); // se nome for null ele atribui o valor nu

Console.ReadKey();
```

Operador Unitário E Ternário

Unitário (+) e Unitário(-)

Ternário(?:)

unitário +(mais)

```
// operador positivo não tem feito na expressão
int positivo = 2;
int resultado;

resultado = +positivo;
Console.WriteLine(resultado); // apenas retorna o valor atribuido

//uniário -(menos)

Console.WriteLine("Informe o numero:");
var n = Convert.ToInt32(Console.ReadLine());

Console.WriteLine($"O negativo de {n} é igual a {-n}"); produz o negativo do s

Console.ReadKey();
```

Operador condicional ternário

Avalia uma expressão booleana e retorna o resultado de uma das duas expressões dependendo da expressão avaliada como true ou false

Operador ternário (?:) é usado para validar uma condição

sintaxe:

```
condição ? expressão_se_true : expressão2_se_false
```

C# - ESTRUTURAS DE CONTROLE

ARRAY,ARRAYLIST E LIST C#

CLASSES E MÉTODOS C#

GENERICS E COLEÇÕES GENÉRICAS

TRATAMENTO DE ERROS

DELEGATES,LAMBDA,EVENTOS E LINQ

CLASSE FILE

SERIALIZAÇÃO E DESSERIALIZAÇÃO

PROGRAMAÇÃO ASSÍNCRONA

LINQ

ATUALIZAÇÕES C#