**Linguagem de programação C#**

**Lucas Matheus Casarotti Rodrigues**

**Sumário**

[1. Introdução 4](#_Toc145413128)

[1.1. O que é C#? 4](#_Toc145413129)

[1.2. O que é .NET? 4](#_Toc145413130)

[1.3. O que é uma linguagem de programação de alto nível? 4](#_Toc145413131)

[1.4. O que é tipagem de dados? 4](#_Toc145413132)

[1.5. O que é compilação e interpretação? 4](#_Toc145413133)

[1.7. Porque utilizar a linguagem de programação C#? 6](#_Toc145413134)

[2. Escopo de um programa em C# 7](#_Toc145413135)

[2.1. Definições 7](#_Toc145413136)

[2.2. O que é um namespace? 7](#_Toc145413137)

[2.3. O que é Using? 7](#_Toc145413138)

[2.4. Resumindo a execução do escopo 7](#_Toc145413139)

[3. Criando o nosso primeiro programa em C# 8](#_Toc145413140)

[3.1. Primeiro programa em c# 8](#_Toc145413141)

[4. C# Fundamental 9](#_Toc145413142)

[4.1. Variáveis e tipos de dados 9](#_Toc145413143)

[4.1.1. Constantes 9](#_Toc145413144)

[4.1.2. Var 10](#_Toc145413145)

[4.1.3. Object 10](#_Toc145413146)

[4.2. Nomes reservados 10](#_Toc145413147)

[4.3. Nullable Types 11](#_Toc145413148)

[4.4. Entrada e saída de dados em C# 11](#_Toc145413149)

[4.5. Conversão de dados 12](#_Toc145413150)

[4.5.1. Implícita 12](#_Toc145413151)

[4.5.2. Explícita 12](#_Toc145413152)

[4.5.3. Parse 12](#_Toc145413153)

[4.5.4. Convert 13](#_Toc145413154)

[4.6. Operadores 14](#_Toc145413155)

[4.6.1. Operadores aritméticos 14](#_Toc145413156)

[4.6.2. Operadores de atribuição 14](#_Toc145413157)

[4.6.3. Operadores de comparação 14](#_Toc145413158)

[4.6.4. Operadores lógicos 15](#_Toc145413159)

[4.7. Estruturas de condição 16](#_Toc145413160)

[4.7.1. IF e Else 16](#_Toc145413161)

[4.7.2. Switch Case 16](#_Toc145413162)

[4.8. Estruturas de repetição 17](#_Toc145413163)

[4.8.1. For 17](#_Toc145413164)

[4.8.2. While 18](#_Toc145413165)

[4.8.3. Do-while 18](#_Toc145413166)

[4.9. Métodos e funções 19](#_Toc145413167)

[4.10. Array 22](#_Toc145413168)

[4.10.1. Declaração de um array 22](#_Toc145413169)

[4.10.2. Inicialização de um array: 22](#_Toc145413170)

[4.10.3. Acesso aos elementos do array: 22](#_Toc145413171)

[4.11. Value Types e Reference Types 23](#_Toc145413172)

# Introdução

## O que é C#?

É uma linguagem de programação de alto nível, tipada e compilada, desenvolvida pela Microsoft como parte da plataforma .NET.

## O que é .NET?

É uma plataforma de desenvolvimento criada pela Microsoft para criar diversos tipos de aplicações.

* + 1. **O que o .NET possui?**
* BLC – Base Class Library (Biblioteca de classes base)
* CLR – Common languagem Runtime (Máquina virtual)
* Gargage Collection (Objetos não utilizados são automaticamente desalocados da memória)

## O que é uma linguagem de programação de alto nível?

A linguagem de programação de alto nível é mais próxima do programador do que do dispositivo, assemelhando-se mais à linguagem humana. Seu objetivo é facilitar a programação, sendo mais intuitiva e fácil de aprender. Diferente da linguagem de programação de baixo nível que é voltada para dispositivos, e tem como principal objetivo se comunicar com o computador.

## O que é tipagem de dados?

A linguagem de programação C# é uma linguagem de tipagem estática, o que significa que é obrigatório especificar o tipo de dado durante a declaração de variáveis, parâmetros de função e retornos de função. Isso permite um alto grau de verificação de tipo em tempo de compilação, o que ajuda a evitar erros de tipo e torna o código mais robusto e seguro.

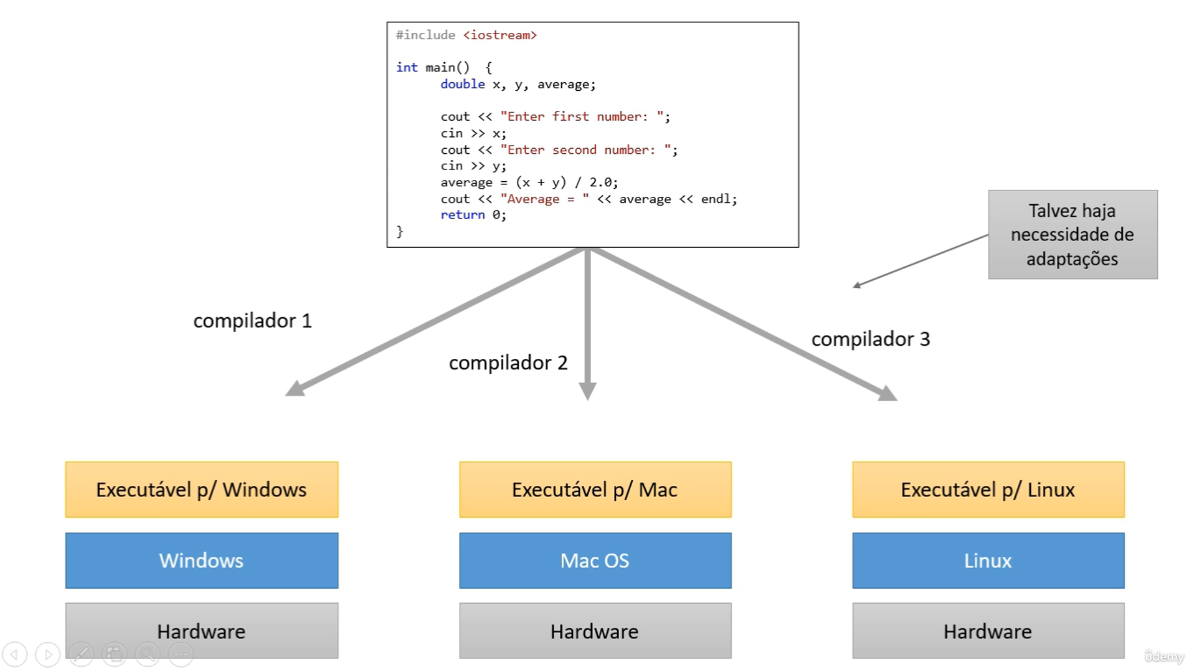
Exemplo:

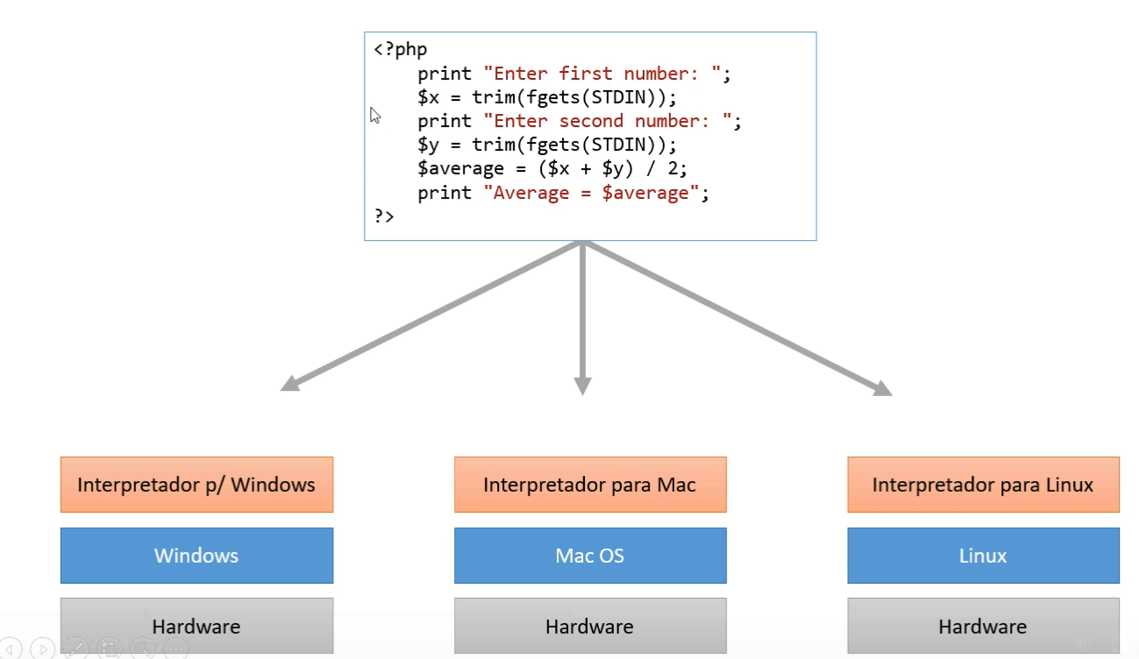
INT idade = declarando uma variável chamada idade do tipo numérico que aceita apenas números inteiros.

* 1. **Qual a diferença entre linguagem de programação compilada e interpretada?**

Na linguagem de programação compilada, o compilador irá ler o código, fazer todos os processos e gerar um arquivo executável. Já na interpretação, nenhum arquivou é gerado e sim uma tradução em tempo real.

Exemplo:

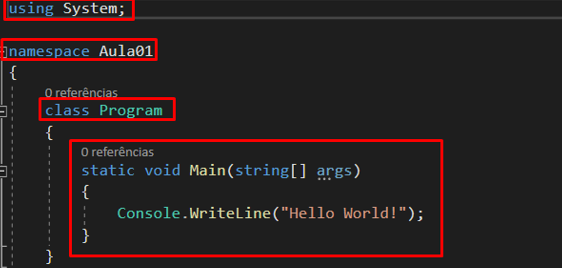
* Linguagens compiladas: C, C++, etc...
* Linguagens de programação interpretadas: PHP, Javascript, etc...



## Porque utilizar a linguagem de programação C#?

Porque é uma linguagem de programação completa, além de ser utilizada para o desenvolvimento web e mobile, pode ser utilizada também para o desenvolvimento de jogos, entre outros. Além disso, é uma tecnologia que está no mercado há um bom tempo, o que pode ser uma grande vantagem, pois consequentemente possui um amplo mercado de trabalho e muitos materiais de estudo disponíveis na internet.

# Escopo de um programa em C#



## Definições

* Importações
* Namespace
* Classe
* Método principal

## O que é um namespace?

Namespaces são divisões lógicas em uma linguagem de programação, onde não é permitido ter duas classes com o mesmo nome em um único namespace. Além disso, um namespace pode ser utilizado em diversos arquivos.

## O que é Using?

O 'using' é uma diretiva em algumas linguagens de programação que permite importar bibliotecas de código externas para serem utilizadas em um arquivo. Outros namespaces, sejam eles internos ou externos, podem ser importados usando o 'using'. Por padrão, apenas as bibliotecas básicas são incluídas, e outras bibliotecas precisam ser importadas explicitamente.

## Resumindo a execução do escopo

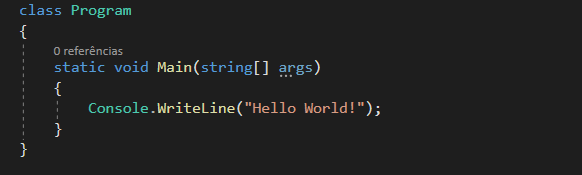
O arquivo 'Program.cs' contém a classe 'Program' e o método 'Main' (principal) em muitos projetos em C# e em outras linguagens de programação. O método 'Main' é o ponto de entrada do programa, onde a execução começa, e é geralmente o primeiro método a ser chamado quando um programa é executado. É responsável por iniciar a lógica do programa e chamar outros métodos e classes necessários para a execução correta do programa.

# Criando o nosso primeiro programa em C#

## Primeiro programa em c#

Neste primeiro programa em C#, utilizaremos o comando Console.WriteLine("Hello World"); para exibir a mensagem 'Hello World'. O método Console.WriteLine é usado para escrever uma linha de texto na saída padrão do console, que pode ser exibida no terminal ou na janela de saída de um ambiente de desenvolvimento integrado (IDE). Neste caso, ele imprimirá a mensagem 'Hello World' na saída padrão do console quando o programa for executado.

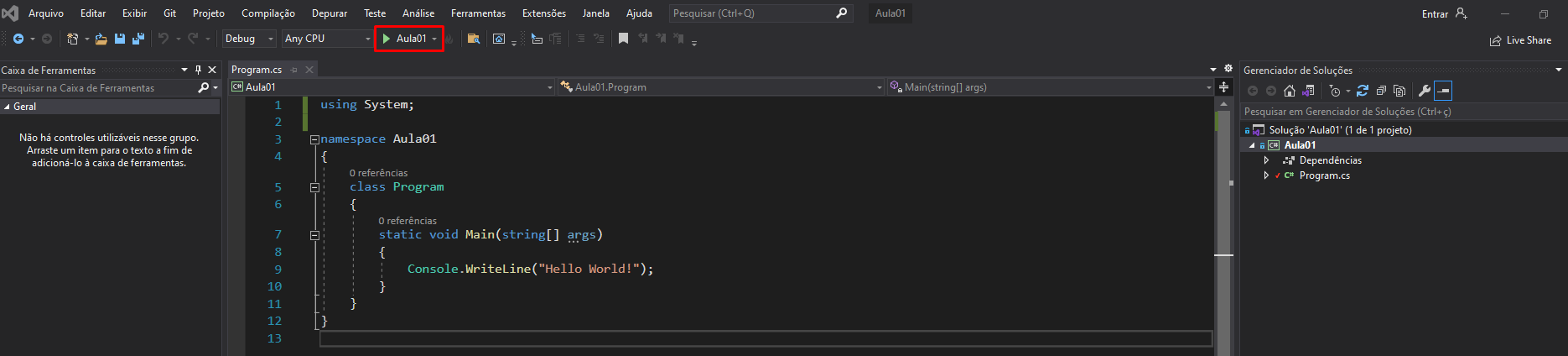
Exemplo:



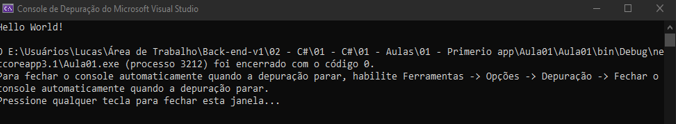
* 1. **Como executar o programa?**

Para executar o programa, é necessário clicar no botão verde que fica localizado na parte superior da IDE.

Exemplo:



Resultado:

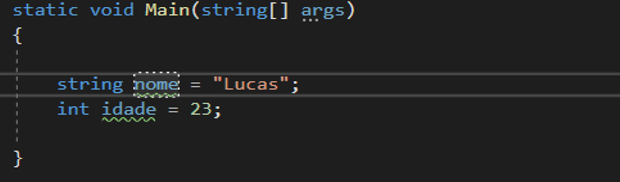


# C# Fundamental

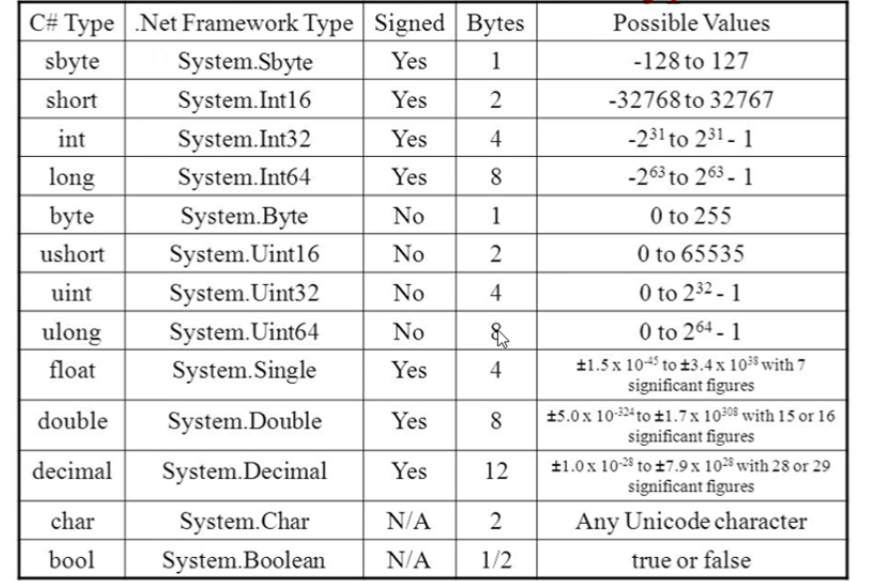
## Variáveis e tipos de dados

Variáveis são espaços na memória utilizados para armazenar valores. Como o próprio nome sugere, elas são variáveis, ou seja, seu valor pode ser alterado a qualquer momento. Sempre que criarmos uma variável, é necessário informar o seu tipo.

Exemplo:



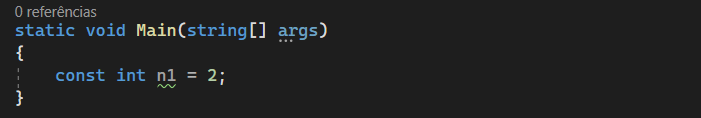
Outros tipos de dados:



### Constantes

Também utilizamos constantes para armazenar valores. No entanto, ao contrário das variáveis, o valor atribuído a uma constante não pode ser modificado após a sua atribuição.

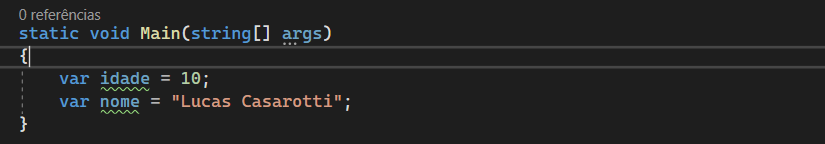
Exemplo:



### Var

‘Var’ é utilizado para permitir que o compilador do C# insira automaticamente o tipo de dado de uma variável com base no valor atribuído a ela. Ou seja, o tipo da variável é determinado automaticamente pelo compilador com base no valor inicial atribuído, tornando desnecessário especificar explicitamente o tipo.

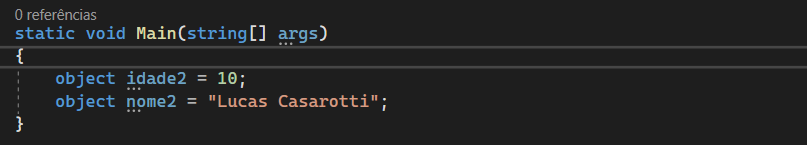
Exemplo:



### Object

O tipo de dado ‘object’ em C# é um tipo de dado que representa um objeto genérico, ou seja, pode armazenar qualquer tipo de valor. É utilizado quando não se sabe previamente o tipo de dado que será armazenado em uma variável e pode ser usado como um tipo de dado genérico.

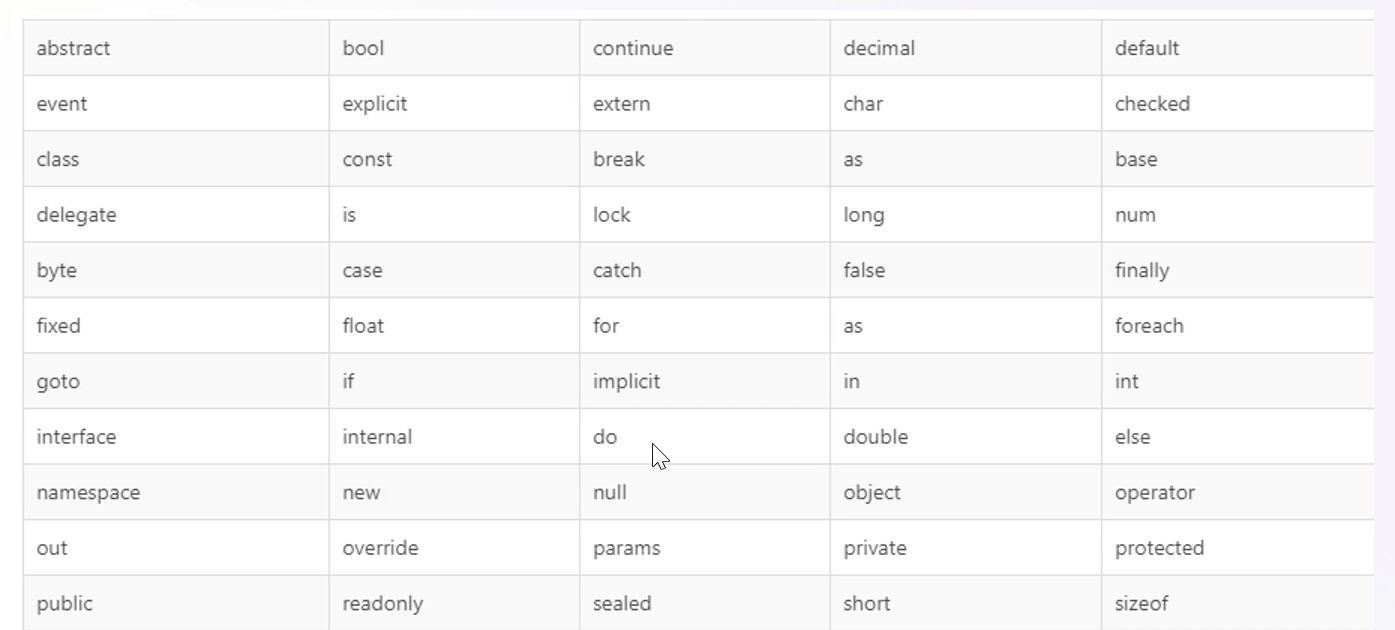
Exemplo:



## Nomes reservados

Também conhecidas como palavras-chave, são palavras exclusivas do C# que não podem ser utilizadas como nomes de variáveis.

Exemplo:



## Nullable Types

Significa vazio, nulo, é diferente de zero ou uma string vazia, todo tipo de dado primitivo ou complexo pode receber o valor null.

Exemplo:

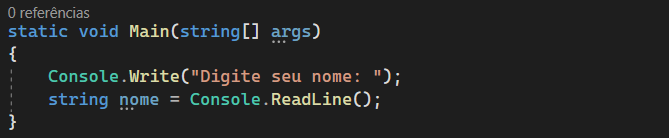


## Entrada e saída de dados em C#

Em C#, a entrada e saída são realizadas por meio da classe 'Console' para interagir com o console padrão (terminal). A partir desse console, os usuários têm a possibilidade de interagir, fornecendo entrada e recebendo saída.

Exemplo:

**Entrada de dados**



Nesse exemplo, o programa solicita ao usuário que insira seu nome e idade. A entrada é lida usando ‘Console.ReadLine()’.

**Saída de dados**

**Placeholder**



**Interpolação**



**Concatenação**



Neste exemplo, o programa simplesmente exibe uma mensagem na tela do console.

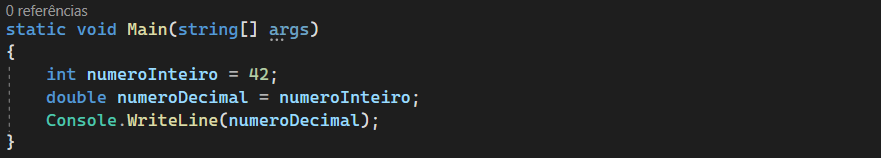
## Conversão de dados

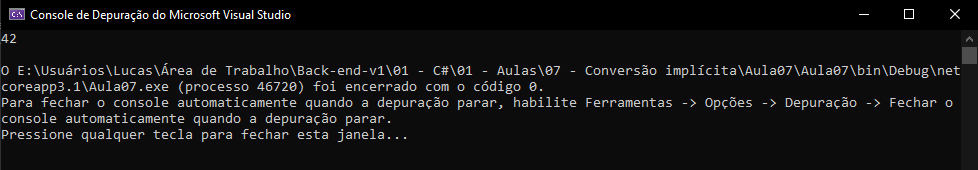
Em C#, a conversão de dados é o processo de converter um valor de um tipo de dado em outro tipo de dado. Isso é necessário quando você deseja realizar operações com valores de tipos diferentes ou quando precisa armazenar um valor em uma variável de outro tipo.

### Implícita

A conversão implícita ocorre quando o C# realiza a conversão automaticamente de um tipo de dado para outro sem perda de dados e sem a necessidade de intervenção do programador.

Exemplo:

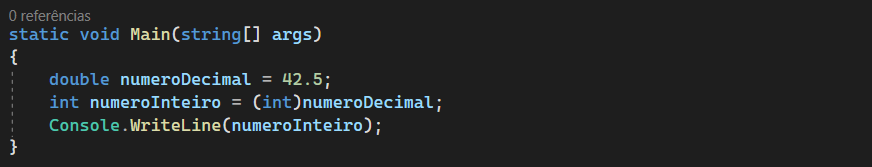


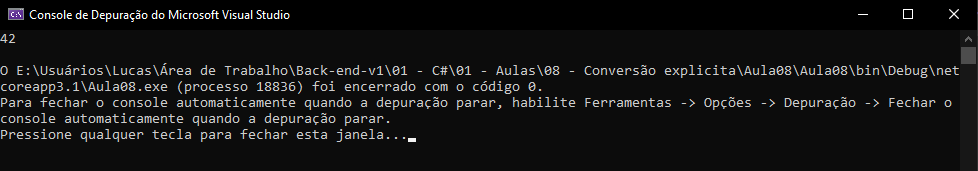


### Explícita

A conversão explícita ocorre quando você realiza uma conversão de um tipo de dado para outro tipo de forma explícita, o que pode resultar na perda de dados ou em um comportamento inesperado.

Exemplo:



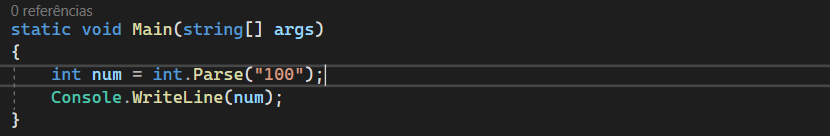


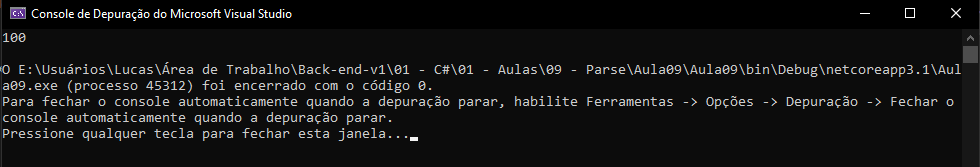
### Parse

Essa extensão é usada para converter um caractere ou uma string para qualquer tipo específico.

Exemplo:

Convertendo uma string “100” para inteiro.



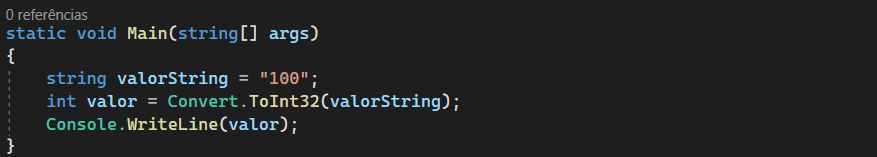


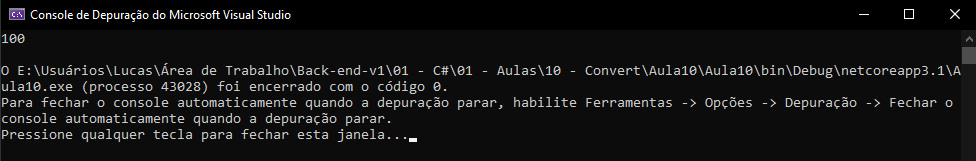
### Convert

Similar ao parse visto anteriormente, porém permite converter vários tipos de valores. Devemos informar o tipo na chamada da conversão.

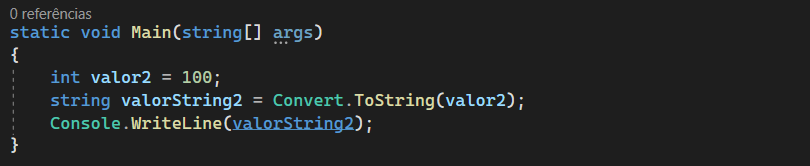
Exemplo:

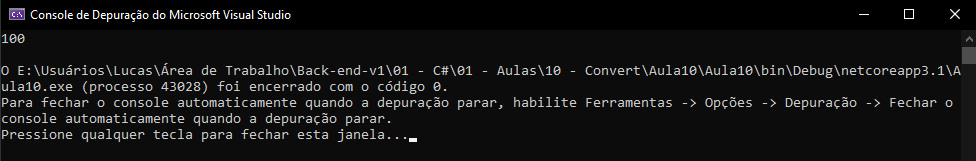
Convertendo inteiro para string





Convertendo string para inteiro





## Operadores

Os operadores em C# são símbolos especiais que realizam operações em operandos para produzir um resultado. Os operadores são fundamentais para a construção de expressões e a realização de cálculos em programas C#

### Operadores aritméticos

São operadores matemáticos dentro do c#, como soma, subtração, divisão e multiplicação.

Exemplo:



### Operadores de atribuição

É utilizado quando queremos atribuir um valor a uma variável.

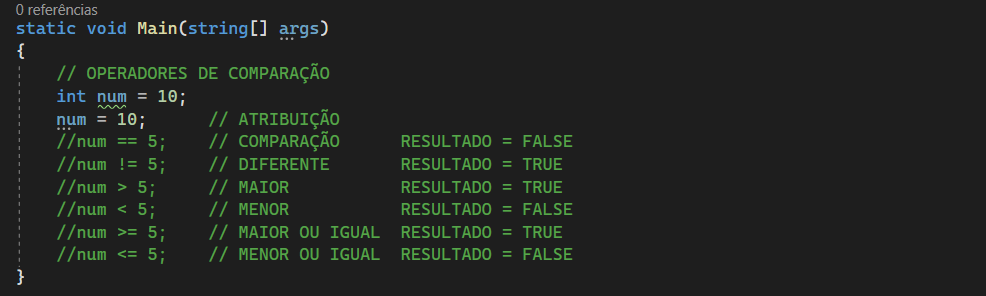
Exemplo:



### Operadores de comparação

É utilizado para comparar dados entre si, como números, strings, bytes, etc. A comparação sempre retorna um valor booleano, verdadeiro (True) ou falso (False).

Exemplo:



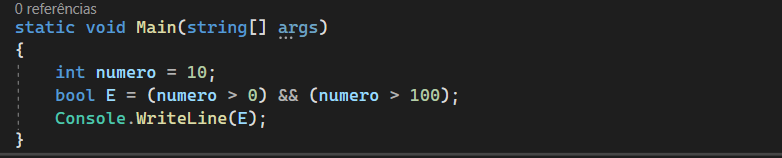
### Operadores lógicos

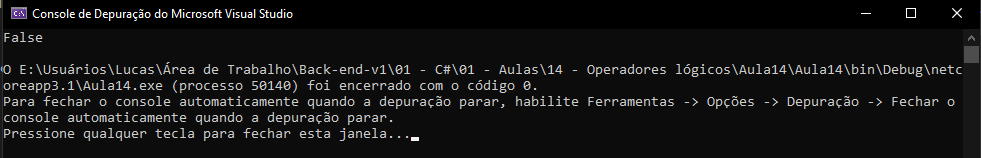
Os operadores condicionais são utilizados para executar instruções com base em condições, retornando um valor booleano, verdadeiro (True) ou falso (False).

#### Operador lógico ‘E’

O operador && realiza uma operação lógica "E" entre duas expressões. Ele retorna true apenas se ambas as expressões forem verdadeiras.

Exemplo:

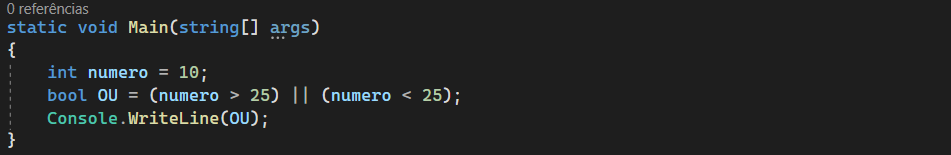


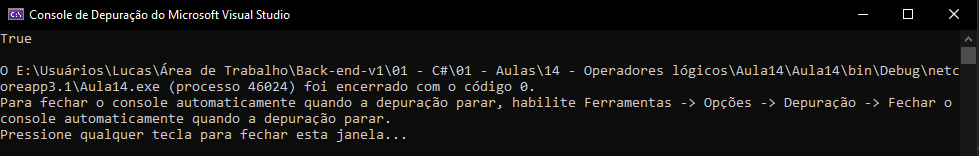


#### Operador lógico ‘OU’

O operador ‘||' realiza uma operação lógica "OU" entre duas expressões. Ele retorna true se pelo menos uma das expressões for verdadeira.

Exemplo:

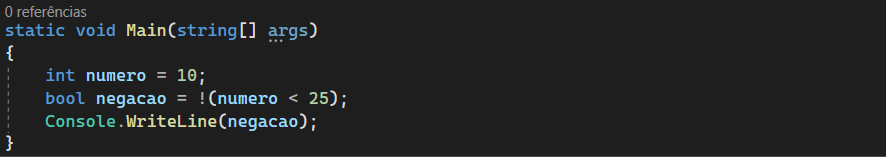


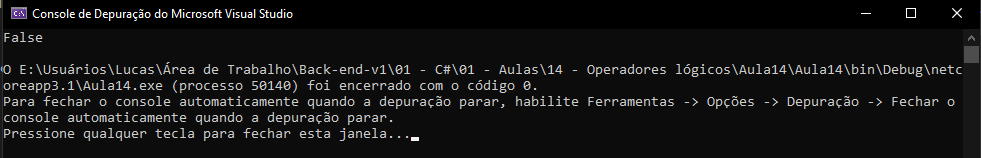


#### Operador lógico ‘OU’

O operador ‘!’ nega uma expressão booleana. Se a expressão for verdadeira, ! a torna falsa e vice-versa.

Exemplo:





## Estruturas de condição

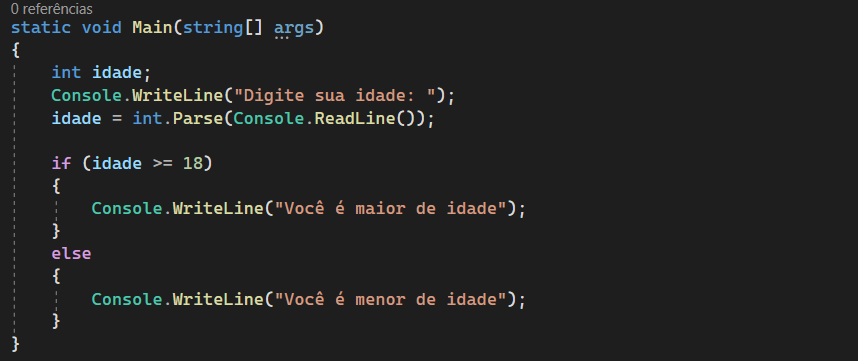
A estrutura de condição é utilizada para verificar uma ou mais condições e executar um bloco de código se a condição for verdadeira (True) e outro bloco de código se a condição for falsa (False).

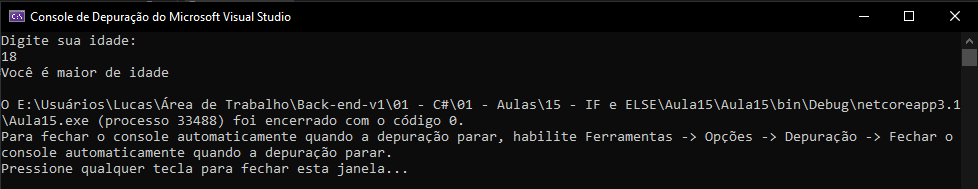
### IF e Else

Em C#, a estrutura de condição ‘if’ e ‘else’ permite que você controle o fluxo de execução do programa com base em uma condição

Exemplo:

Se a condição da variável idade for maior ou igual a 18 retornar true e executa o código dentro do ‘if’, caso contrário, retorna false e o bloco ‘else’ será executado.

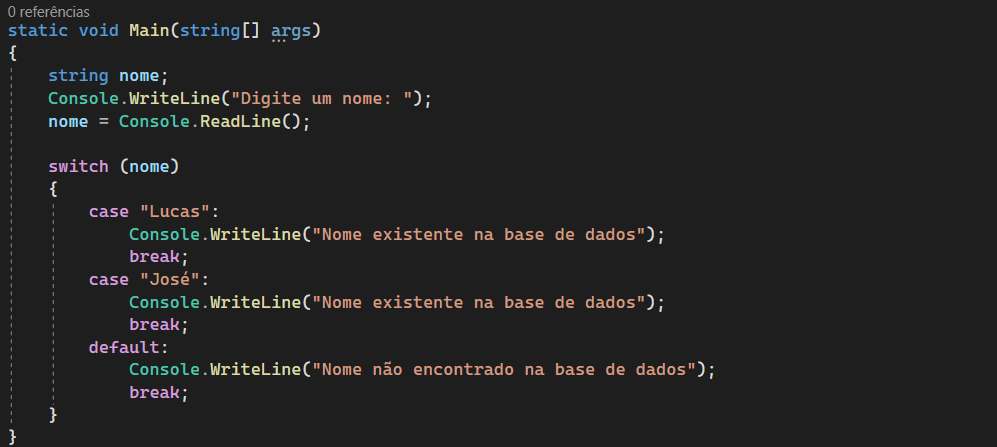


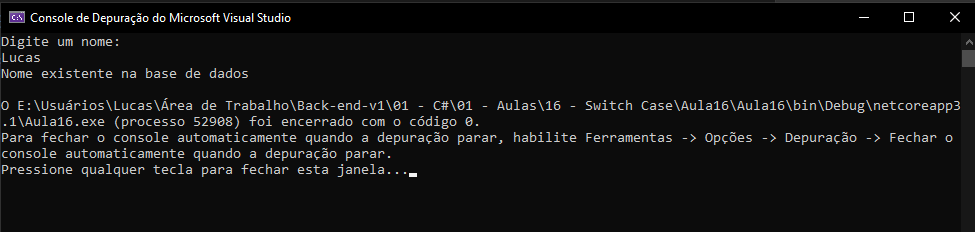


### Switch Case

É uma forma de reduzir a complexidade de vários ‘if’ e ‘else’ encadeados. É muito utilizado, principalmente em estruturas de menu. O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.

Exemplo:





## Estruturas de repetição

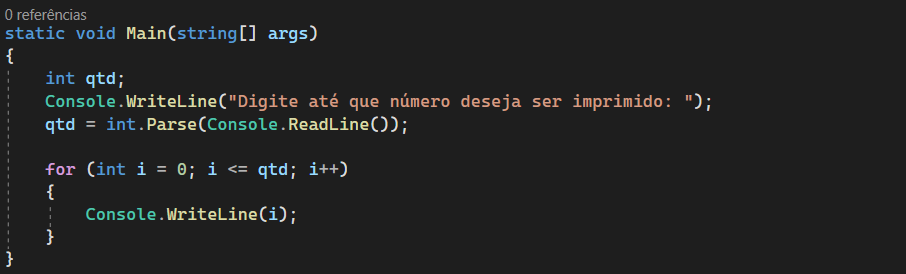
Em C#, existem várias estruturas de repetição que permitem que você execute um bloco de código várias vezes com base em uma condição. As estruturas de repetição mais comuns são for, while e do-while.

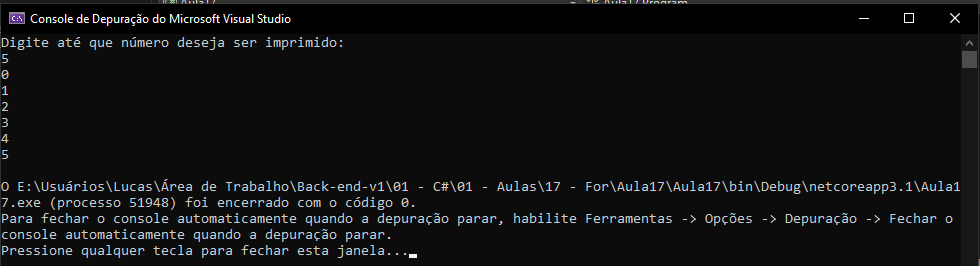
### For

Na tradução “para”, o laço de repetição ‘for’ é muito utilizado em programação para executar uma determinada ação várias vezes. Ele é definido por três parâmetros: o contador, a condição de repetição e o incremento.

Exemplo:

A variável ‘i‘ é igual a 0, enquanto ela for menor ou igual a um determinado valor o programa será executado. Porém, a cada execução a variável ‘i’ recebe o valor dela mesmo mais um.

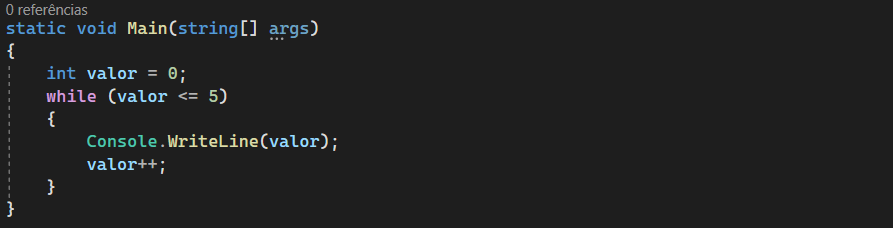


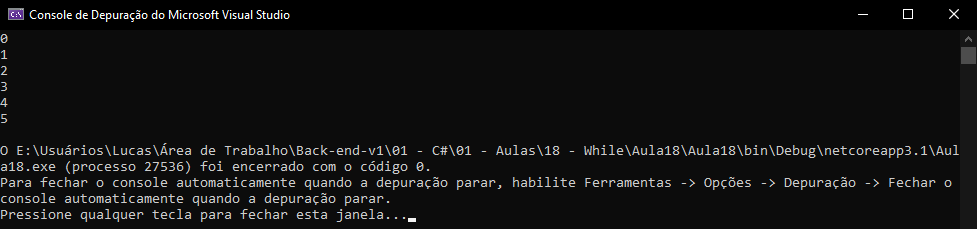


### While

Na tradução "enquanto", o laço de repetição ‘while’ é utilizado para executar um bloco de código enquanto uma determinada condição for verdadeira. O código será executado repetidamente até que a condição se torne falsa.

Exemplo:





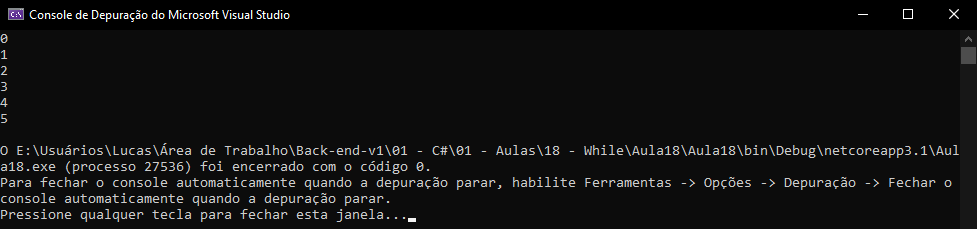
Resumidamente, no começo do loop ‘while’, a condição é definida como "enquanto a variável 'valor' for menor ou igual a 5", ou seja, a condição é verdadeira (true). A cada interação do loop, o valor da variável 'valor' é incrementado em 1. Após executar o loop 6 vezes consecutivas, o valor da variável 'valor' será maior do que 5, tornando a condição falsa (false) e assim interrompendo a execução do loop "while".

### Do-while

Na tradução "Faça", o loop ‘do-while’ é utilizado para executar uma ação enquanto uma condição seja verdadeira, porém a condição é verificada após a execução do código, ao contrário do ‘while’ que verifica a condição no início. Isso significa que, mesmo que a condição seja falsa, o código dentro do ‘do-while’ será executado pelo menos uma vez antes de verificar a condição.

Exemplo:





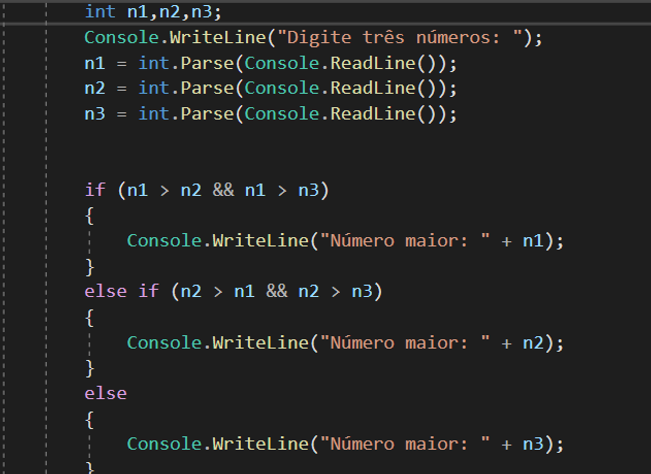
## Métodos e funções

No contexto da programação, uma "função" é um conjunto de comandos que realiza uma tarefa específica e pode ser reutilizado em diferentes partes de um programa. Funções são como pequenos programas independentes que podem ser chamados por outros programas para realizar uma determinada ação. Elas ajudam a modularizar e organizar o código, tornando-o mais legível, manutenível e reutilizável. As funções podem receber argumentos, processar dados e retornar um resultado, permitindo uma abordagem modular e estruturada na construção de software.

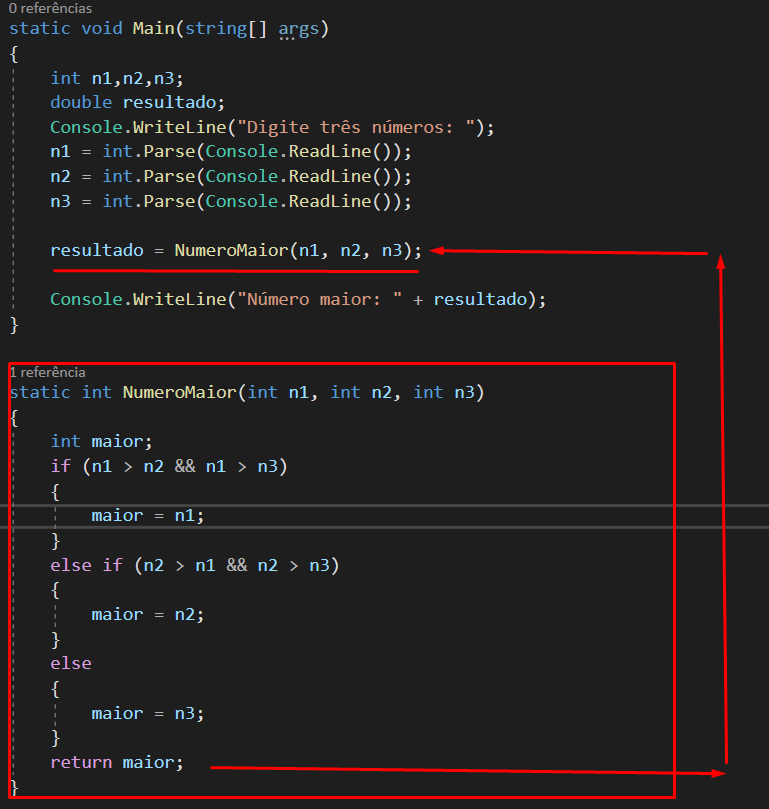
Exemplo:

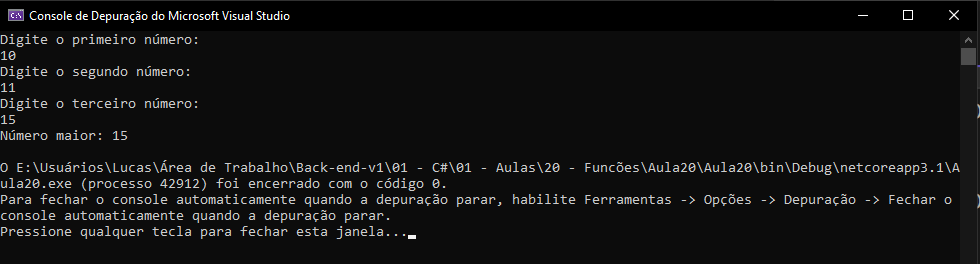
1 – Crie um programa que leia três números e mostre o maior número digitado

Na imagem abaixo, o exercício foi realizado sem a utilização de uma função. A desvantagem é que toda vez que for necessário ler três números e mostrar o maior número digitado, será preciso repetir a verificação utilizando if-else. Isso pode resultar em duplicação de código, tornando o programa mais extenso e difícil de manter.



Na imagem abaixo, foi utilizada a função criada "NumeroMaior" para fazer a verificação do maior número entre três valores. A vantagem é que, em outra parte do código, se for necessário fazer a mesma verificação, basta chamar a função e passar os parâmetros de entrada. Dessa forma, evita-se a duplicação de código e torna-se mais modular e reutilizável.





A variável "resultado" está recebendo o valor retornado pela função "NumeroMaior", que é chamada com os parâmetros de entrada "n1", "n2" e "n3". Dentro da função, esses parâmetros são verificados para determinar qual deles é o maior número, e em seguida é utilizado o comando "return maior" para retornar o valor do maior número encontrado.

## Array

Em C#, um array é uma estrutura de dados que permite armazenar uma coleção de elementos do mesmo tipo, acessíveis por meio de um índice numérico. Os arrays são usados ​​para armazenar dados em uma estrutura unidimensional, bidimensional ou multidimensional, dependendo da necessidade do programador.

### Declaração de um array

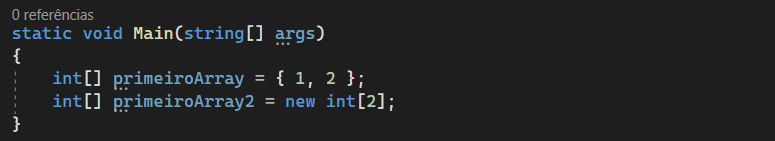
Para declarar um array em C#, você precisa especificar o tipo dos elementos que ele conterá, seguido pelo nome do array e, opcionalmente, seu tamanho.

Exemplo:



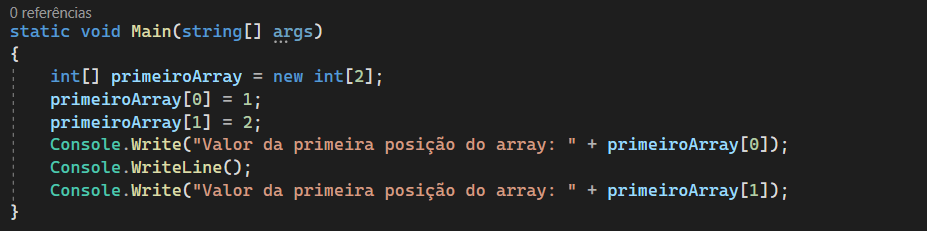
### Inicialização de um array:

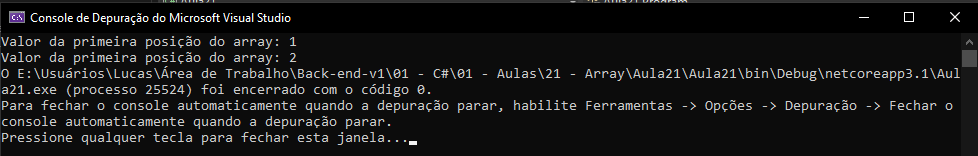
Após declarar um array, você pode inicializá-lo de várias maneiras:



### Acesso aos elementos do array:

Os elementos em um array são acessados pelo índice. Os índices em C# começam em 0. Por exemplo, para acessar o primeiro elemento do array números, você faria o seguinte:





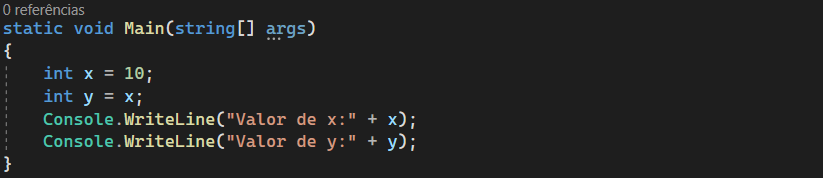
## Value Types e Reference Types

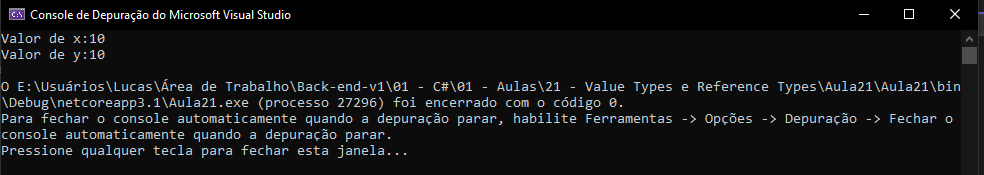
Value Types e Reference Types são dois conceitos importantes na programação, principalmente em linguagens que suportam tipagem forte. Eles se referem à forma como os dados são armazenados e manipulados na memória.

Value Types (tipos de valor) são tipos de dados que são armazenados diretamente no local em que são declarados. Isso significa que, ao atribuir um valor a uma variável de um tipo de valor, o valor em si é armazenado na variável. Reference Types (tipos de referência), por outro lado, são tipos de dados que são armazenados em uma área de memória separada chamada heap. Em vez de armazenar o valor diretamente na variável, uma referência ao valor é armazenada na variável. Isso significa que a variável não contém o valor real, mas sim um endereço de memória que aponta para o local onde o valor está armazenado.

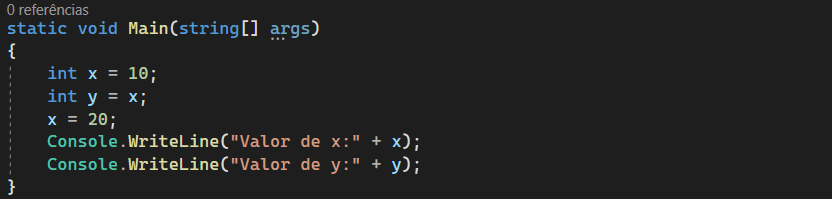
Exemplo de Value Types:

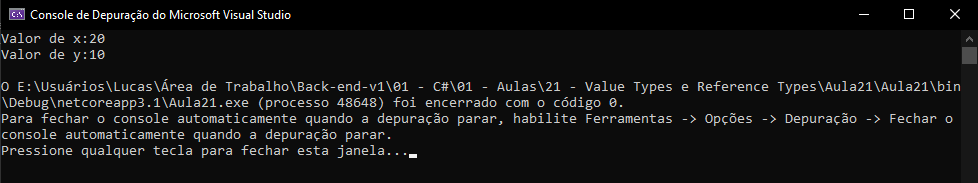
Foi criado uma variável ‘x’ e uma ‘y’, a variável ‘x’ é igual a dez e a variável ‘y’ é uma cópia de x, ou seja, também é dez.





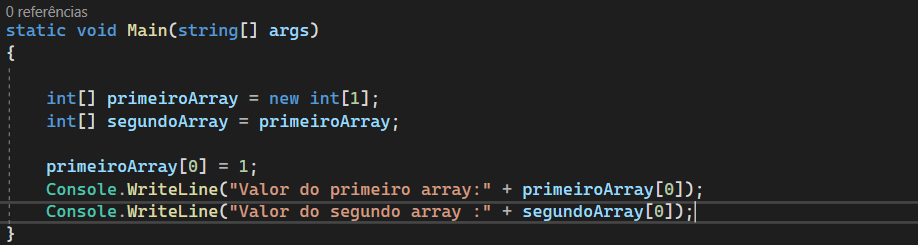
Porém, quando alteramos o valor de ‘x’ a variável ‘y’ continua com o valor inicial de ‘x’, pois é uma cópia armazenada no stack da memória, onde não temos referência. Diferente do Reference Types que armazena o endereço do objeto que contém os dados.

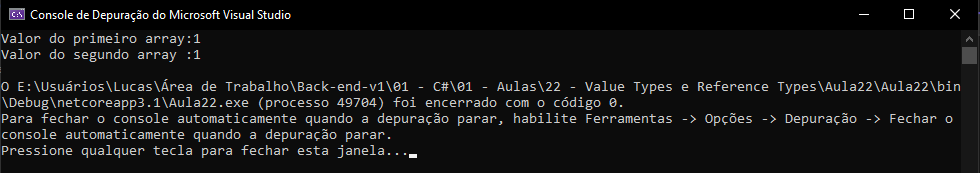




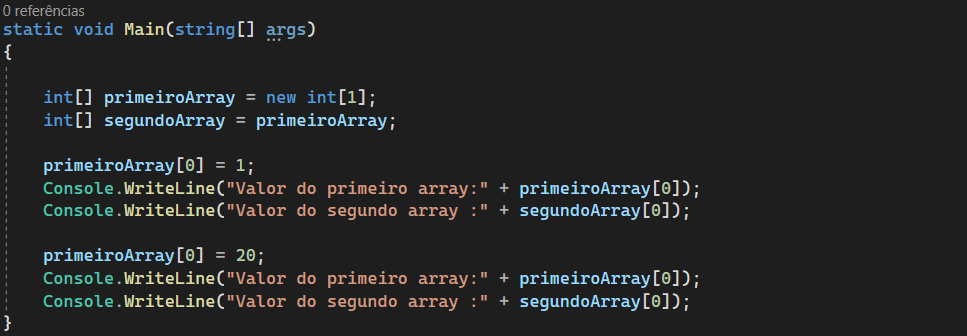
Exemplo de Reference Types:

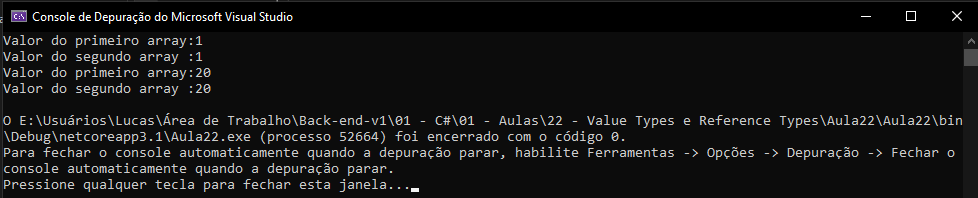
Foi criado dois arrays chamados ‘primeiroArray’ e ‘segundoArray’, porém o segundo array está recebendo o primeiro array. Logo em seguida foi definido um valor para o primeiro elemento do primeiro array.





Desta forma, mesmo se alterarmos o valor do elemento do primeiro array o valor do segundo array também será alterado pois arrays são do tipo Reference Types, ou seja, eles não salvam o valor diretamente e sim a referência da variável.



Classes, objetos e arrays são do tipo Reference Types.