



Programação e Sistemas de Informação

CURSO PROFISSIONAL TÉCNICO DE
GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

Mecanismos de Controlo de Execução

MÓDULO 2

Professor: João Martiniano

Índice

- [Introdução](#)
- [Sintaxe](#)
- [Estrutura de um programa](#)
- [A consola](#)
- [Variáveis](#)
- [Tipos de dados](#)
- [Operadores](#)
- [Instruções condicionais](#)
- [Instruções repetitivas](#)

Introdução

- Neste módulo iremos utilizar a linguagem C# para aprender a programar
- Importa saber que a linguagem C# é um componente da .NET Framework

A .NET Framework

- A .NET Framework é uma tecnologia da Microsoft, utilizada principalmente no sistema operativo Windows
- A primeira versão foi lançada em 2002
- É uma plataforma de software, de grande dimensão e complexidade, que disponibiliza um conjunto de funcionalidades e recursos aos programadores
- Suporta várias linguagens de programação:
 - C#, C++/CLI, F#, L#, IronPython, Visual Basic .NET, entre várias outras
- Todas as linguagens de programação suportadas têm acesso às funcionalidades e recursos da *framework*

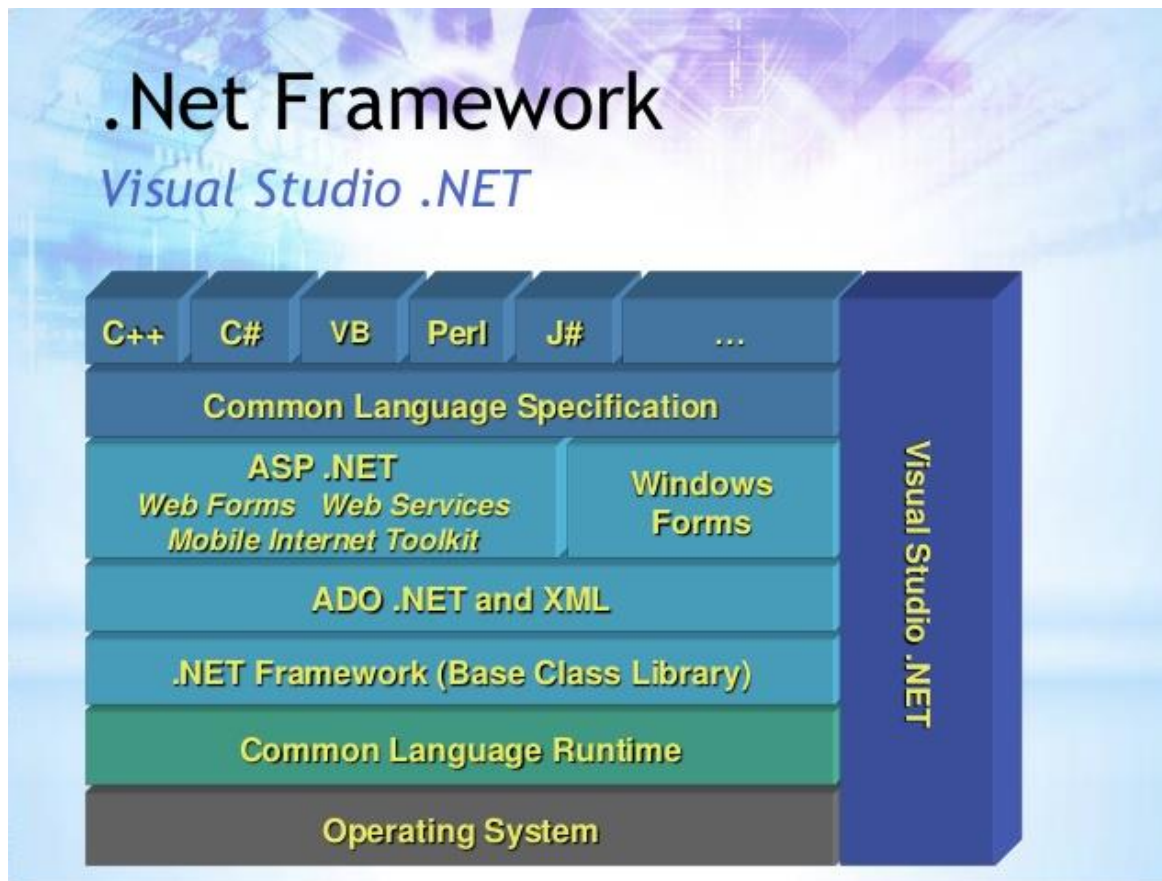
Introdução

A .NET Framework (continuação)

- Para além disso, existem várias tecnologias incorporadas na *framework* que permitem o desenvolvimento de vários tipos de aplicações, tais como:
 - desktop, web, mobile, embedded, etc.
- Algumas das tecnologias presentes na *framework*:
 - WinForms, WPF, ASP.NET, LINQ , ADO.NET, WCF, etc.
- Geralmente os programas escritos numa linguagem .NET são menos rápidos a executar, do que os programas que não correm na .NET Framework
- No entanto:
 - são mais seguros
 - são mais estáveis
 - o seu desenvolvimento é mais fácil e rápido

Introdução

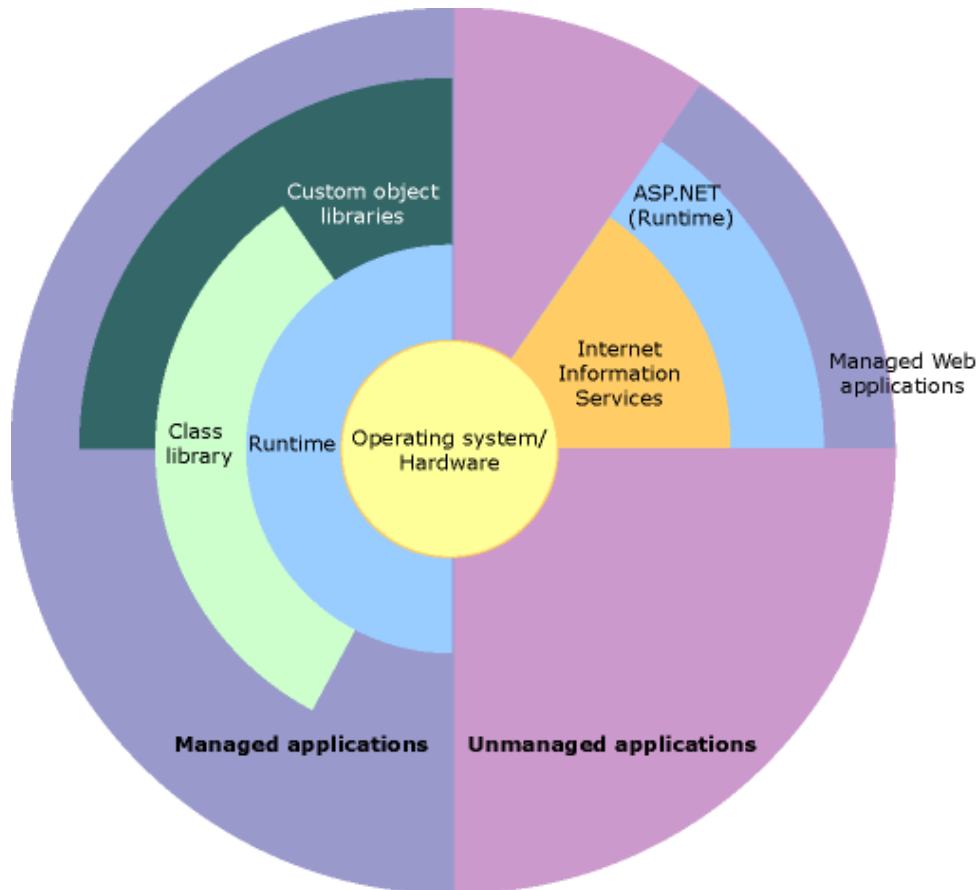
- A .NET Framework é composta por tecnologias que se organizam em camadas:



<https://www.slideshare.net/avermamsd/microsoft-dot-net-framework-15549278>

Introdução

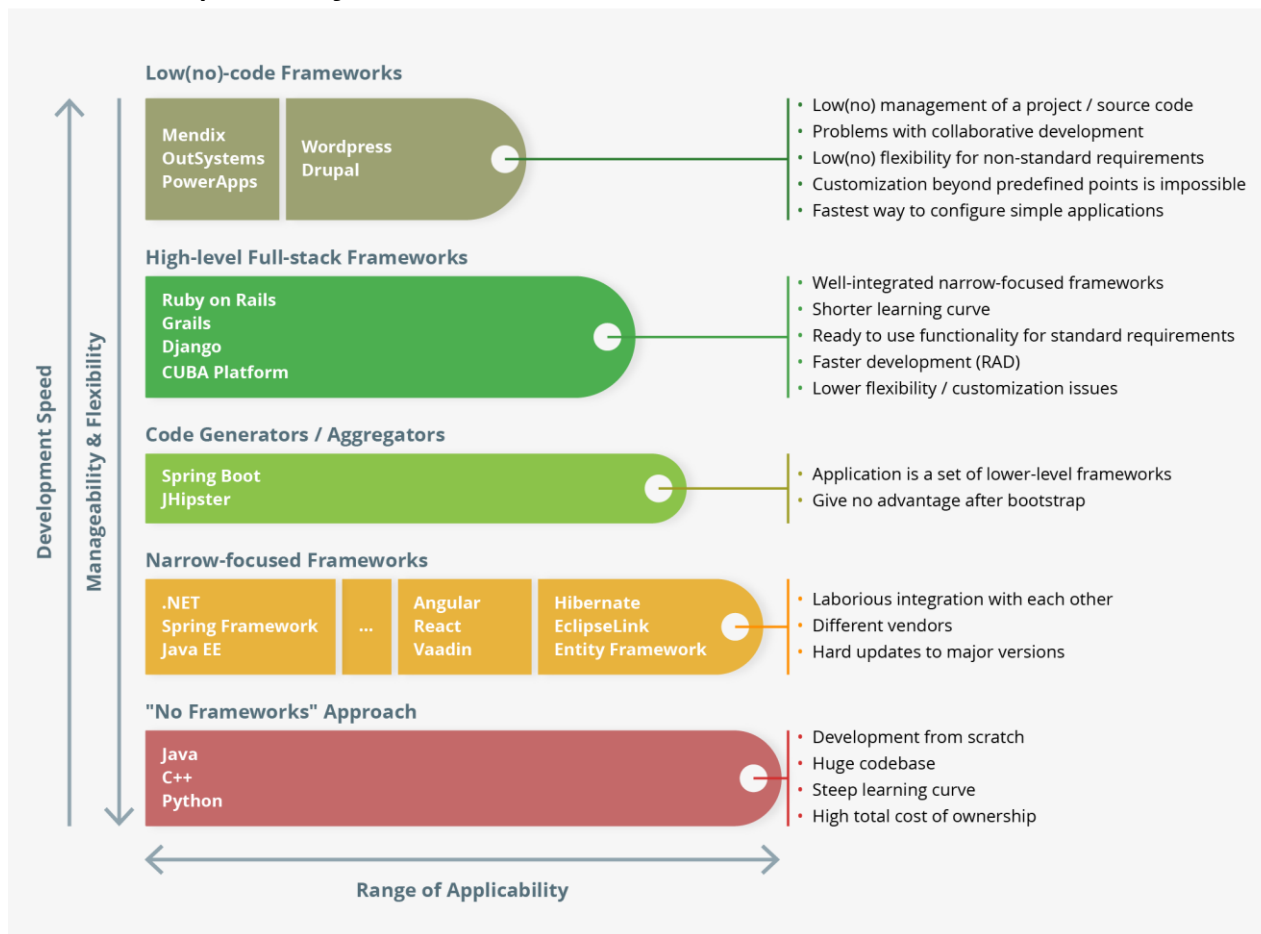
- Visão esquemática de alguns componentes da .NET Framework:



<https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview>

Introdução

- Existem outros tipos de *frameworks*:



<https://www.cuba-platform.com/blog/classification-of-development-frameworks-for-enterprise-applications>

A linguagem C#

- A linguagem de programação C# foi desenvolvida pela Microsoft em 2000
- Características:
 - é uma linguagem moderna
 - orientada a objetos
 - *type-safe* (operações de escrita/leitura de memória seguras)
 - *strongly typed* (cada variável tem um tipo de dados)
 - sensível a maiúsculas/minúsculas
 - possui mecanismo de *garbage collecting* (processo automático e contínuo de libertação de memória)
 - possui mecanismo de *exception handling* (mecanismo de recuperação de erros)

A linguagem C#

- Características (continuação):
 - *general purpose*: permite o desenvolvimento de diferentes tipos de programas em diferentes tipos de plataformas de hardware
 - periodicamente são acrescentadas novas funcionalidades
- A linguagem C# foi sujeita a um processo de standardização internacional, nomeadamente através das normas **ECMA-334** e **ISO/IEC 23270**

Terminação de instruções

- As instruções devem **sempre** terminar com o caracter ponto-e-vírgula (;)

```
// Correto
int x = 1;

x = 5 + 5;

Console.WriteLine(x);
Console.ReadKey();

// Incorreto: instruções não terminam com ;
++x
string nome = ""
```

Sensibilidade maiúsculas/minúsculas

- A linguagem C#, à semelhança de outras, é *case sensitive*
- Ou seja, diferencia as letras maiúsculas das minúsculas
- Quando se escrevem as instruções deve-se respeitar sempre a forma como são apresentadas na sintaxe
- Caso contrário ocorre um erro de compilação
- Por exemplo, o método `Console.WriteLine()`:

<code>Console.WriteLine()</code>	✓ Correto
<code>Console.<u>w</u>rite<u>l</u>ine()</code>	✗ Incorreto
<code><u>c</u>onsole.WriteLine()</code>	✗ Incorreto
<code>Console.<u>w</u>riteLine()</code>	✗ Incorreto
<code><u>CONSOLE</u>.<u>WRITELINE</u>()</code>	✗ Incorreto

Blocos de instruções

- Os caracteres `{` e `}` marcam respetivamente, o início e o fim de um bloco de instruções
- Um bloco de instruções é um conjunto de uma ou mais instruções
- Essencialmente, os blocos de instruções são utilizados em duas ocasiões:
 - para marcar o início e o fim de conjuntos de instruções pertencentes a outras instruções (`for`, `while`, `do while`, `if`, `switch`, etc.)
 - para marcar o início e o fim de conjuntos de instruções pertencentes a funções/métodos
- Exemplo:

```
while (i <= 3)
{
    Console.WriteLine(i);
    ++i;
}
```

→ Início do bloco de instruções

→ Fim do bloco de instruções

Blocos de instruções

- Os caracteres `{` e `}` marcam respetivamente, o início e o fim de um bloco de instruções
- Um bloco de instruções é um conjunto de uma ou mais instruções
- Essencialmente, os blocos de instruções são utilizados em duas ocasiões:
 - para marcar o início e o fim de conjuntos de instruções pertencentes a outras instruções (`for`, `while`, `do while`, `if`, `switch`, etc.)
 - para marcar o início e o fim de conjuntos de instruções pertencentes a funções/métodos
- Exemplo:

```
while (i <= 3)
{
    Console.WriteLine(i);
    ++i;
}
```

Estas instruções
pertencem ao bloco de
instruções

Comentários

- Na linguagem C# existem dois tipos de comentários:

- comentários que ocupam apenas uma linha

```
// Comentário
```

- comentários que podem ocupar várias linhas, delimitados pelos caracteres `/*` e `*/`

```
/* Comentário */
```

```
/* Comentário  
   que ocupa  
   várias  
   linhas */
```

Comentários

Para que servem os comentários?

- Servem duas funções principais:
 1. Para comentar ou documentar o código
 - Os comentários são um instrumento precioso para ajudar a manter e compreender qualquer programa
 2. Para impedir que determinado código seja executado
 - Por vezes é necessário manter código mas não queremos que este seja executado

Comentários

- Exemplo 1: Comentários que ocupam uma linha

```
// Declarar e inicializar variáveis  
int a = 2;  
int b = 2;  
int c = 0;  
  
// Somar a e b  
c = a + b;  
  
// Mostrar o resultado na consola  
Console.WriteLine(c);
```


Comentários

- Exemplo 2: Comentários que ocupam várias linhas

```
/* O seguinte código mostra, na consola,  
 * os números entre 0 e 3. */  
int i = 0;  
  
while (i <= 3)  
{  
    /* Mostrar na consola o conteúdo  
     * da variável i.  
     * De seguida incrementar  
     * o valor de i, em uma unidade.  
     */  
    Console.WriteLine(i);  
    ++i;  
}
```

Comentários

Exercício

- Quais dos seguintes comentários estão corretos?

- a) / Comentário
- b) /* Comentário */
- c) // Comentário //
- d) \$ Comentário
- e) // Comentário
- f) /* Comentário

*/

- g) # # Comentário
- h) */ Comentário */
- i) <!-- Comentário -->
- j) // Comentário /* que continua
por mais uma linha */

Comentários

Exercício

- Quais dos seguintes comentários estão corretos?

- | | |
|---------------------|-----------------------------|
| a) / Comentário | ✗ Incorreto (começa com /) |
| b) /* Comentário */ | ✓ Correto |
| c) // Comentário // | ✓ Correto |
| d) \$ Comentário | ✗ Incorreto (começa com \$) |
| e) // Comentário | ✓ Correto |
| f) /* Comentário | ✓ Correto |

*/

- | | |
|---|--|
| g) # # Comentário | ✗ Incorreto (começa com #) |
| h) */ Comentário */ | ✗ Incorreto (deveria iniciar com /* e acabar com */) |
| i) <!-- Comentário --> | ✗ Incorreto (comentário HTML) |
| j) // Comentário /* que continua
por mais uma linha */ | ✗ Incorreto |

Palavras reservadas

- Existe um conjunto de palavras que não podem ser utilizadas como identificadores
- São palavras reservadas porque correspondem a instruções ou porque têm um significado especial na linguagem C#

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

| SINTAXE

Sintaxe

- Sintaxe são as regras que regem a escrita de uma linguagem
- Neste documento é apresentada a sintaxe de vários elementos da linguagem de programação C#
- Ao apresentar a sintaxe são utilizados certos símbolos que têm um determinado significado

[]

- Significado:

Indica um componente opcional

- Exemplo:

```
for ([inicialização]; [condição]; [incremento])  
    instruções
```

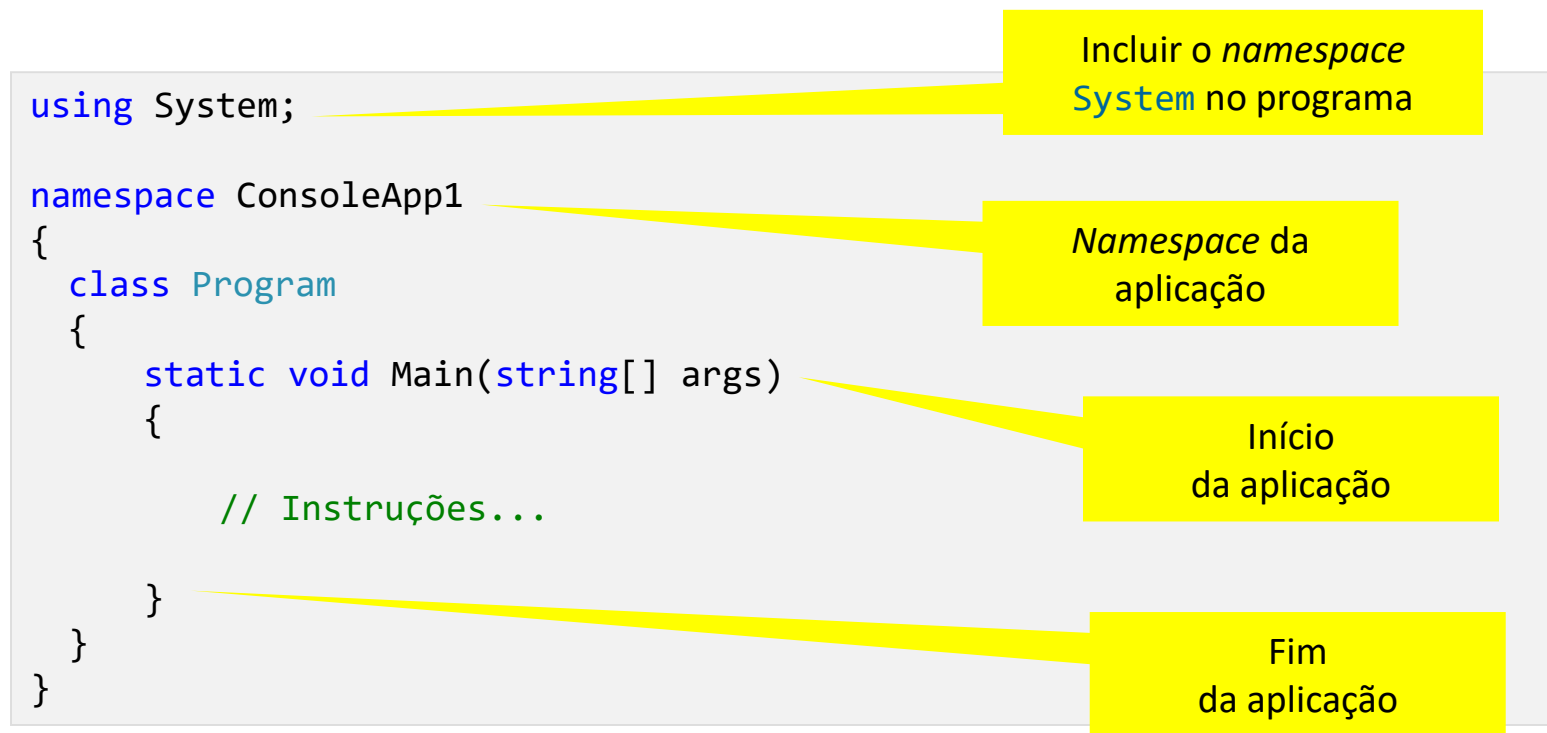
- Explicação:

Os componentes `inicialização`, `condição` e `incremento` são opcionais, podendo ser incluídos ou não na instrução `for`

ESTRUTURA DE UM PROGRAMA

Estrutura de um programa

- Exemplo da estrutura básica de um programa C#:



Estrutura de um programa

- Exemplo da estrutura básica de um programa C#:

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Instruções...

        }
    }
}
```

As instruções do programa
são colocadas aqui

O método `main()`

- O *entry point* (ou ponto de entrada) de uma aplicação ocorre sempre no método `main()`
- O *entry point* é o local onde tem início a execução de uma aplicação
- O método `main()` pode assumir diferentes formas:

	Recebe dados do exterior	Devolve dados para o exterior
<code>static void Main()</code>		
<code>static void Main(string[] args)</code>	✓	
<code>static int Main()</code>		✓
<code>static int Main(string[] args)</code>	✓	✓

- Por defeito, é utilizado o segundo formato, mas nada impede que seja utilizado qualquer um dos outros formatos

O método `main()`

- Por exemplo, o seguinte programa:
 - recebe dois números inteiros através da linha de comando
 - efetua a soma dos números
 - mostra na consola o resultado da soma

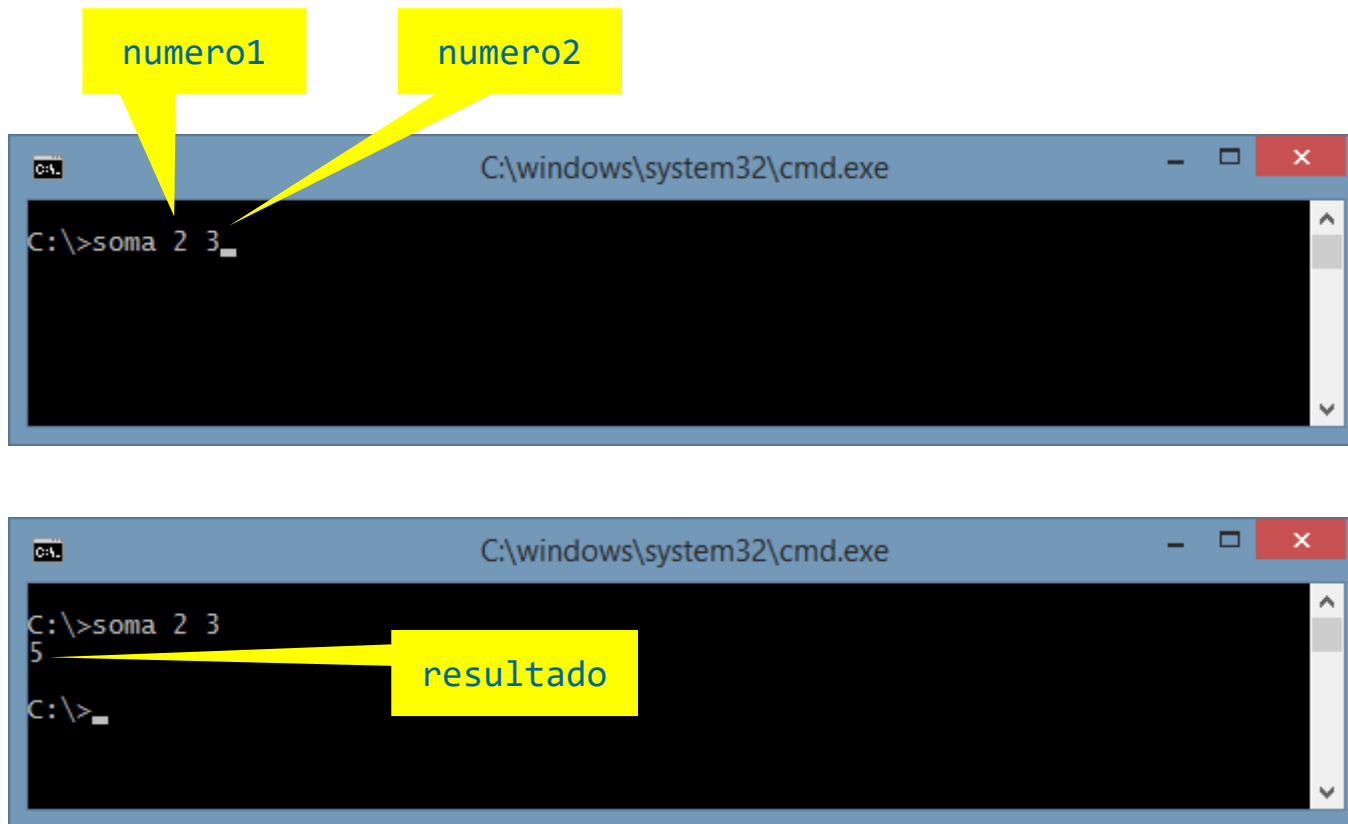
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Soma
{
    class Program
    {
        static void Main(string[] args)
        {
            int numero1 = Convert.ToInt32(args[0]);
            int numero2 = Convert.ToInt32(args[1]);
            int resultado = 0;

            resultado = numero1 + numero2;

            Console.WriteLine(resultado);
        }
    }
}
```

O método `main()`



```
C:\windows\system32\cmd.exe

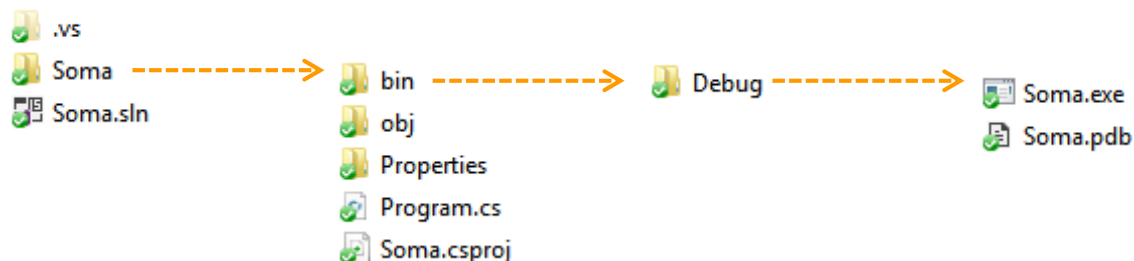
C:\>soma 2 3_

C:\>soma 2 3
5
C:\>
```

- Importa notar que o programa anterior não valida os dados que o utilizador insere
- Nomeadamente: se insere dados e se esses dados são 2 números

Estrutura de pastas

- Os projetos em Visual Studio têm uma estrutura de pastas e ficheiros
- Tomando como exemplo o projeto **Soma** (analisado anteriormente):

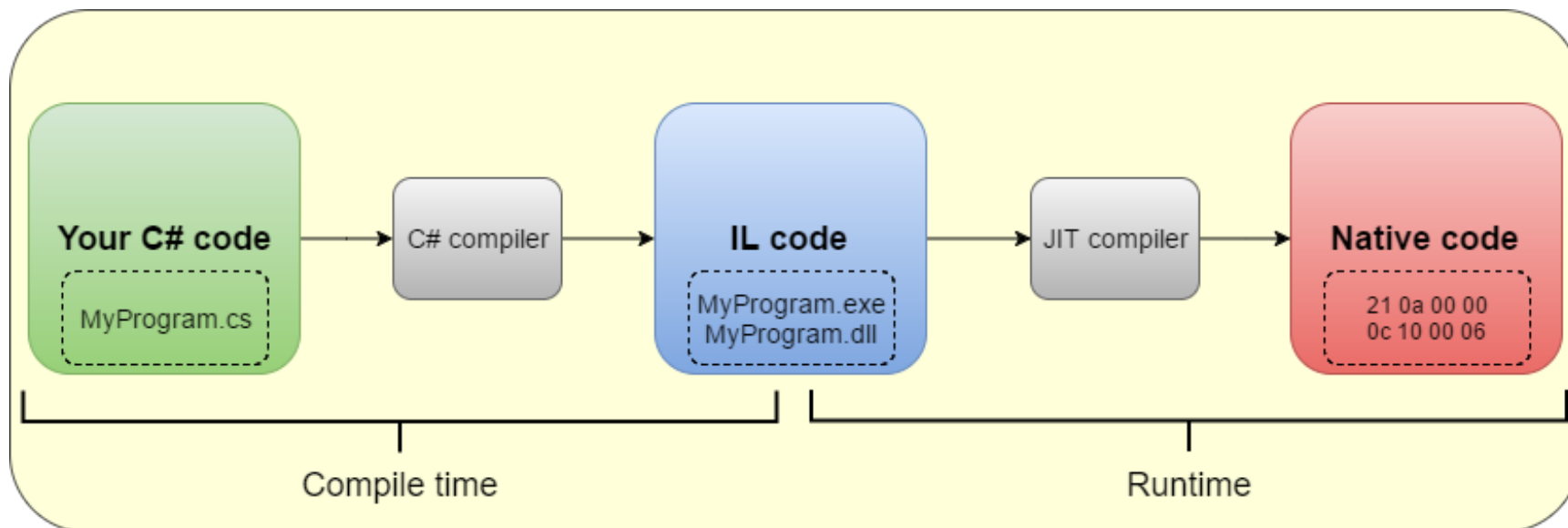


- Soma.sln**: o *solution file* (contém informações para manter e organizar o(s) projeto(s) de uma solução)
- Soma.csproj**: o *project file* ou ficheiro de projeto (contém as informações necessárias para compilar o projecto)
- Program.cs**: o ficheiro com código do programa
- Pasta **bin** → **Debug**: contém o ficheiro executável **Soma.exe**, atualizado sempre que ocorre uma compilação bem sucedida do programa
- Pasta **obj**: contém ficheiros intermédios e temporários, gerados durante o processo de compilação

O processo de compilação e execução

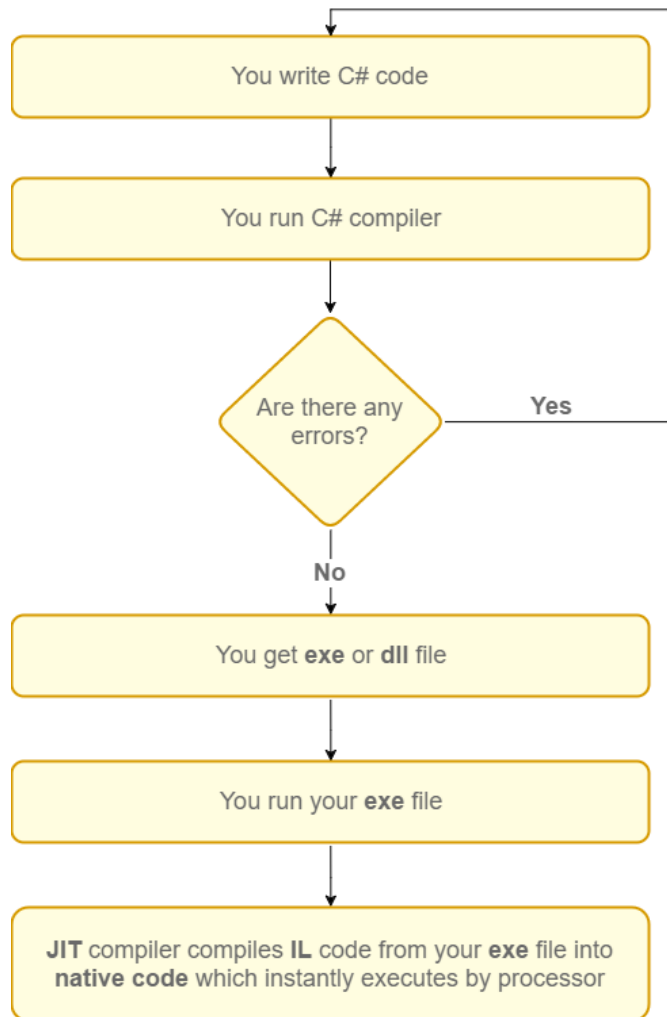
- Antes de um programa poder ser executado, necessita de ser **compilado**
- O processo de compilação consiste em:
 - analisar o código C# para verificar que está correto
 - se forem detetados erros, o programador é informado e deverá corrigi-los
 - processar e transformar o código C# em código no formato *Microsoft Intermediate Language* (MSIL)
 - com o código MSIL é gerado um ficheiro EXE ou DLL (por exemplo [Soma.exe](#))
- Para executar o programa:
 - o *Common Language Runtime* (CLR) encarrega-se de processar o ficheiro EXE
 - transforma o código MSIL em código máquina (sequências de 0 e 1) que é a única linguagem que o processador (CPU) percebe
 - o CLR utiliza um compilador *Just In Time* (JIT) que compila o código MSIL para código máquina

O processo de compilação e execução



https://codeeasy.net/lesson/c_sharp_compilation_process

O processo de compilação e execução



https://codeeasy.net/lesson/c_sharp_compilation_process

| A CONSOLA

A consola

- A consola é uma janela de texto na qual:
 - é mostrada informação, do sistema operativo ou de uma aplicação
 - o utilizador pode emitir comandos
- Tipos de consola:
 - do sistema operativo
 - das aplicações
- No sistema operativo, a consola é conhecida como linha de comandos
- O sistema operativo Linux funciona muito à base de linha de comandos
- Neste módulo e seguintes, iremos programar utilizando a consola para:
 - mostrar dados (*output*) ao utilizador
 - receber dados (*input*) a partir do utilizador

A consola

- Neste módulo iremos abordar a classe `Console`
- Esta classe disponibiliza funcionalidades para trabalhar com a consola
- Nesta secção iremos:
 - escrever dados na consola (*output*)
 - receber dados da consola (*input*)
 - customizar a consola

A consola: *Output* de dados

- Para efetuar *output* de dados são utilizados os seguintes métodos:
 - `Write()`
 - `WriteLine()`
- Estes métodos permitem escrever informação na consola
- O método `WriteLine()` acrescenta o caracter *newline* ao final da informação, forçando qualquer comando `Write()` ou `WriteLine()` subsequente, a começar na linha seguinte
- Estes métodos podem ser utilizados de diversas formas
- Começamos por analisar a mais simples

A consola: *Output* de dados

- Sintaxe:

```
Console.Write(informação)
```

```
Console.WriteLine([informação])
```

- Em que:

informação

A informação a mostrar na consola. Pode ser de diferentes tipos:

- números (inteiros, reais, etc.)
- texto
- valores lógicos
- valor de variáveis
- resultado de expressões e métodos
- etc.

- Atenção:** o parâmetro *informação* é opcional no método `WriteLine()`

A consola: *Output* de dados

- Os seguintes exemplos demonstram a diferença entre os métodos `Write()` e `WriteLine()`:

```
Console.Write("Linguagem C#");  
Console.Write(1234);  
Console.Write("***");
```

```
Console.WriteLine("Linguagem C#");  
Console.WriteLine(1234);  
Console.WriteLine("***");
```

- Resultado:

```
Linguagem C#1234***
```

```
Linguagem C#  
1234  
***
```

- Como se pode observar, após uma instrução `WriteLine()`, o *output* passa obrigatoriamente para a linha seguinte

A consola: *Output* de dados

Exercício

- Qual o *output* das seguintes alíneas?

a)

```
Console.WriteLine("****");  
Console.Write("1");  
Console.Write("2");  
Console.WriteLine("34");  
Console.Write("****");
```

b)

```
Console.Write("A");  
Console.WriteLine(" B");  
Console.Write(" C");  
Console.Write(" D");
```

c)

```
Console.Write("A");  
Console.Write("");  
Console.Write("");  
Console.Write("");  
Console.Write("B");
```

A consola: *Output* de dados

Exercício: Solução

- Qual o *output* das seguintes alíneas?

a)

```
****  
1234  
****
```

b)

```
A B  
C D
```

c)

```
AB
```


A consola: *Output* de dados

- Como foi referido anteriormente, os métodos `Write()` e `WriteLine()` podem mostrar diferentes tipos de informação
- Exemplos:

Números

```
Console.WriteLine(10);
```

Texto

```
Console.WriteLine("Hello world");
```

Datas

```
Console.WriteLine(DateTime.Now.ToString());
```

Valores lógicos

```
Console.WriteLine(true);
```

Valor de variáveis

```
int x = 1; Console.WriteLine(x);
```

Resultado de expressões
e métodos

```
Console.WriteLine(2 + 2);  
Console.WriteLine(Math.Abs(-14.7));
```

A consola: *Output* de dados

- Uma forma de efetuar *output* de diferentes tipos de dados consiste em utilizar o operador `+` para juntar as várias expressões numa única string (operação de concatenação)
- Exemplo:

```
string prod = "Maçã"  
decimal preco = 0.20M;  
  
Console.WriteLine("Produto: " + prod + " " + preco + " €");
```

- Esta forma de efetuar *output*, no entanto, é pouco sofisticada e pouco flexível
- É aconselhável utilizar o mecanismo de *composite formatting* analisado seguidamente

A consola: *Output* de dados

- Podemos formatar o *output* utilizando o mecanismo de *composite formatting*
- Este mecanismo é utilizado com os métodos `Write()` e `WriteLine()` e também com outros métodos
- Este mecanismo consiste:
 - numa string com *placeholders* indexados
 - uma lista de objetos que fornecem valores para os *placeholders*
- Exemplo:

```
Console.WriteLine("Produto: {0} {1} €", prod, preco)
```

A consola: *Output* de dados

- Podemos formatar o *output* utilizando o mecanismo de *composite formatting*
- Este mecanismo é utilizado com os métodos `Write()` e `WriteLine()` e também com outros métodos
- Este mecanismo consiste:
 - numa string com *placeholders* indexados
 - uma lista de objetos que fornecem valores para os *placeholders*
- Exemplo:

```
Console.WriteLine("Produto: {0} {1} €", prod, preco)
```



String

A consola: *Output* de dados

- Podemos formatar o *output* utilizando o mecanismo de *composite formatting*
- Este mecanismo é utilizado com os métodos `Write()` e `WriteLine()` e também com outros métodos
- Este mecanismo consiste:
 - numa string com *placeholders* indexados
 - uma lista de objetos que fornecem valores para os *placeholders*
- Exemplo:

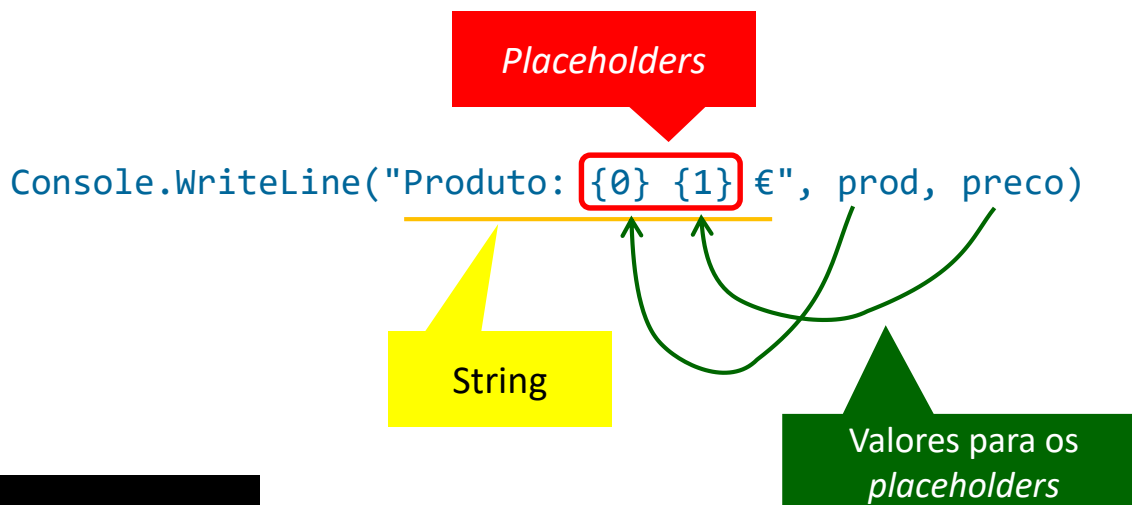
`Console.WriteLine("Produto: {0} {1} €", prod, preco)`

Placeholders

String

A consola: *Output* de dados

- Podemos formatar o *output* utilizando o mecanismo de *composite formatting*
- Este mecanismo é utilizado com os métodos `Write()` e `WriteLine()` e também com outros métodos
- Este mecanismo consiste:
 - numa string com *placeholders* indexados
 - uma lista de objetos que fornecem valores para os *placeholders*
- Exemplo:



- Resultado:

Produto: Maçã 10 €

A consola: *Output* de dados

- É possível atribuir aos *placeholders*, variáveis, resultados de métodos, propriedades e campos de classes, e expressões
- Exemplo:

```
string nome = "Yoda";  
float temp = 30.2F;  
  
Console.WriteLine("O seu nome é {0} \nA temperatura atual é {1}ºC  
                \nO valor de PI é {2} \n2 ao quadrado é {3}  
                \n5 + 314 = {4}", nome, temp, Math.PI, Math.Pow(2,2),  
                5 + 314);
```

- Resultado:

```
O seu nome é Yoda  
A temperatura atual é 30,2ºC  
O valor de PI é 3,14159265358979  
2 ao quadrado é 4  
5 + 314 = 319
```

A consola: *Output* de dados

- É possível atribuir aos *placeholders*, variáveis, resultados de métodos, propriedades e campos de classes, e expressões
- Exemplo:

```
string nome = "Yoda";
float temp = 30.2F;

Console.WriteLine("O seu nome é {0} \nA temperatura atual é {1}°C
                  \nO valor de PI é {2} \n2 ao quadrado é {3}
                  \n5 + 314 = {4}", nome, temp, Math.PI, Math.Pow(2,2),
                  5 + 314);
```

- Resultado:

Expressão

Variável

Variável

Campo

Método

```
O seu nome é Yoda
A temperatura atual é 30,2°C
O valor de PI é 3,14159265358979
2 ao quadrado é 4
5 + 314 = 319
```


A consola: *Output* de dados

- A melhor forma para efetuar *output* de variáveis e expressões é o mecanismo de *string interpolation*
- Neste mecanismo uma string é precedida do carácter **\$**
- Dentro da string são utilizados os caracteres **{** e **}** para inserir uma *interpolation expression*
- Exemplo:

```
Console.WriteLine($"Data e hora atuais: {DateTime.Now}");
```



Caracter **\$**

A consola: *Output* de dados

- A melhor forma para efetuar *output* de variáveis e expressões é o mecanismo de *string interpolation*
- Neste mecanismo uma string é precedida do caracter \$
- Dentro da string são utilizados os caracteres { e } para inserir uma *interpolation expression*
- Exemplo:

```
Console.WriteLine($"Data e hora atuais: {DateTime.Now}");
```

Caracter \$

String

A consola: *Output* de dados

- A melhor forma para efetuar *output* de variáveis e expressões é o mecanismo de *string interpolation*
- Neste mecanismo uma string é precedida do caracter \$
- Dentro da string são utilizados os caracteres { e } para inserir uma *interpolation expression*
- Exemplo:

```
Console.WriteLine($"Data e hora atuais: {DateTime.Now}");
```

Caracter \$

String

Interpolation expression

A consola: *Output* de dados

- Exemplo: mostrar a designação e preço de um produto, utilizando diferentes métodos



```
string prod = "Maçã"  
decimal preco = 0.20M;
```

```
Console.WriteLine("Produto: " + prod + " " + preco + " €");
```

Concatenação
Evitar este estilo de programação

```
Console.WriteLine("Produto: {0} {1} €", prod, preco);
```

Composite formatting
Melhor que a concatenação



```
Console.WriteLine($"Produto: {prod} {preco} €");
```

String interpolation
Utilização aconselhada

A consola: *Output* de dados

- Podemos incluir expressões de diferentes tipos diretamente no interior de uma *interpolated string*
- Exemplos:

```
Console.WriteLine($"2 + 2 = {2 + 2}");
```

```
decimal precoSemIva = 0.20M;  
Console.WriteLine($"Preço com IVA: {precoSemIva + (precoSemIva * 0.23M)} €");
```

A consola: *Input* de dados

- Para efetuar *input* de dados são utilizados os seguintes métodos:
 - `ReadKey()`
 - `ReadLine()`
- Estes métodos permitem receber informação (teclas pressionadas) a partir da consola

A consola: O método `ReadKey()`

- O método `ReadKey()` retorna uma tecla pressionada pelo utilizador

- Sintaxe:

`Console.ReadKey([intercept])`

- Em que:

`intercept` (opcional) Valor booleano que determina se o caracter é mostrado na consola (se `true`, o caracter é intercetado, não sendo mostrado)

- Este método retorna um objeto `ConsoleKeyInfo` que permite determinar o tipo de tecla pressionada pelo utilizador

A consola: O método `ReadKey()`

- Exemplo 1: pedir que o utilizador pressione uma tecla e terminar a aplicação

```
Console.WriteLine("Pressione uma tecla para terminar o programa");  
Console.ReadKey();
```

- Exemplo 2: terminar o programa apenas se o utilizador pressionar a tecla `Enter`

```
Console.WriteLine("Pressione a tecla Enter para terminar o programa");  
while (Console.ReadKey().Key != ConsoleKey.Enter) { }
```

- Exemplo 3: não mostrar os caracteres que o utilizador pressionou

```
Console.WriteLine("Pressione a tecla Enter para terminar o programa");  
while (Console.ReadKey(true).Key != ConsoleKey.Enter) { }
```

Intercetar o caracter

A consola: O método `ReadKey()`

- Exemplo 4: testar se o utilizador pressiona teclas específicas (nomeadamente a tecla `1`, a tecla `A` e a tecla `ESC`)

```
ConsoleKeyInfo k;  
Console.WriteLine("Pressione uma das seguintes teclas:");  
Console.WriteLine("  1: Calcular 2+2");  
Console.WriteLine("  A: Mostrar mensagem");  
Console.WriteLine("  ESC: Terminar o programa");
```

```
do  
{
```

```
    Console.Write("> ");  
    k = Console.ReadKey();  
    Console.WriteLine();
```

```
    if (k.Key == ConsoleKey.D1)  
    {  
        Console.WriteLine("2 + 2 = {0}", 2 + 2);  
    }
```

```
    else if (k.Key == ConsoleKey.A)  
    {  
        Console.WriteLine("Bom dia");  
    }
```

```
} while (k.Key != ConsoleKey.Escape);
```

Verificar se foi
pressionada a tecla `1`

Verificar se foi
pressionada a tecla `A`

Verificar se não foi
pressionada a tecla `ESC`

A consola: O método `ReadLine()`

- O método `ReadLine()` lê um conjunto de caracteres, apenas terminando quando o utilizador pressiona a tecla `Enter`
- Este método retorna uma string com os caracteres inseridos pelo utilizador
- Sintaxe:

`Console.ReadLine()`

- Exemplo: pedir o nome ao utilizador e de seguida mostrar uma saudação

```
string nome = "";

Console.Write("Insira o seu nome: ");
nome = Console.ReadLine();
Console.WriteLine("Bom dia {0}", nome);
```

- Resultado:

```
Insira o seu nome: Darth Vader
Bom dia Darth Vader
```

A consola: O encoding

- A propriedade `OutputEncoding` permite obter ou modificar o *encoding* do *output* na consola
- De modo a poder mostrar o maior número de caracteres, utilize o *encoding* `Unicode`
- Exemplo:

```
Console.WriteLine("10 €");
```



10 ?

```
Console.OutputEncoding = Encoding.Unicode;  
Console.WriteLine("10 €");
```



10 €

Título da consola: A propriedade `Title`

- A propriedade `Title` permite especificar um título para a janela da consola

- Sintaxe:

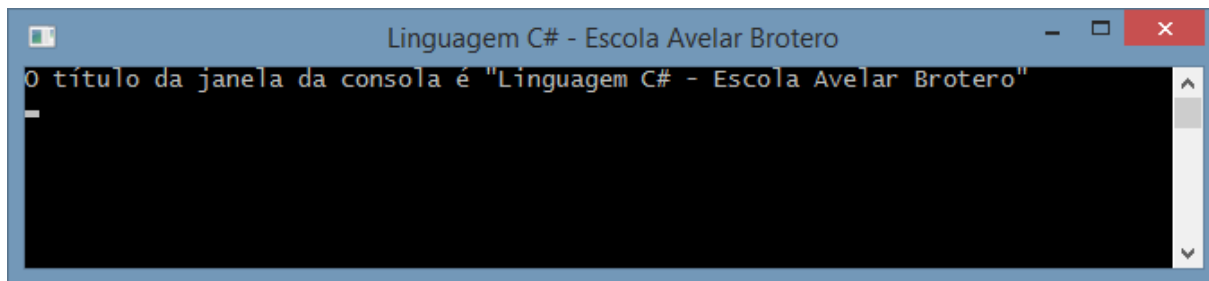
```
Console.Title = "título"
```

- Em que:

`título` (string) O título a atribuir à janela da consola

- Exemplo:

```
Console.Title = "Linguagem C# - Escola Avelar Brotero";  
Console.WriteLine("O título da janela da consola é {0}", Console.Title);  
Console.ReadKey();
```



Cor de caracter: A propriedade `ForegroundColor`

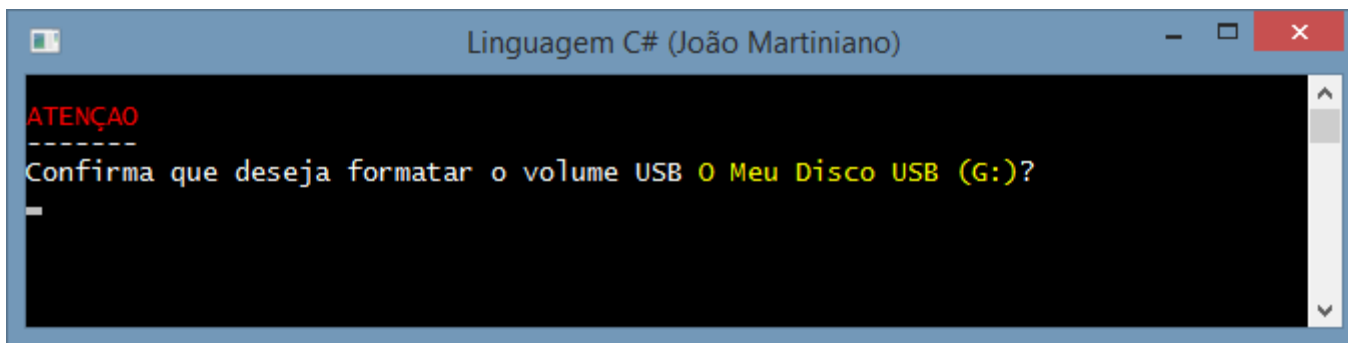
- A propriedade `ForegroundColor` permite especificar uma cor para os caracteres da consola
- Apenas podem ser especificadas cores da enumeração `ConsoleColor`
- Sintaxe:
`Console.ForegroundColor = cor`
- Em que:
`cor` Uma cor da enumeração `ConsoleColor`



Cor de caracter: A propriedade `ForegroundColor`

- Exemplo:

```
Console.WriteLine();  
Console.ForegroundColor = ConsoleColor.Red;  
Console.WriteLine("ATENÇÃO");  
Console.ForegroundColor = ConsoleColor.White;  
Console.WriteLine("-----");  
Console.Write("Confirma que deseja formatar o volume USB ");  
Console.ForegroundColor = ConsoleColor.Yellow;  
Console.Write("O Meu Disco USB (G:)");  
Console.ForegroundColor = ConsoleColor.White;  
Console.WriteLine("?");
```



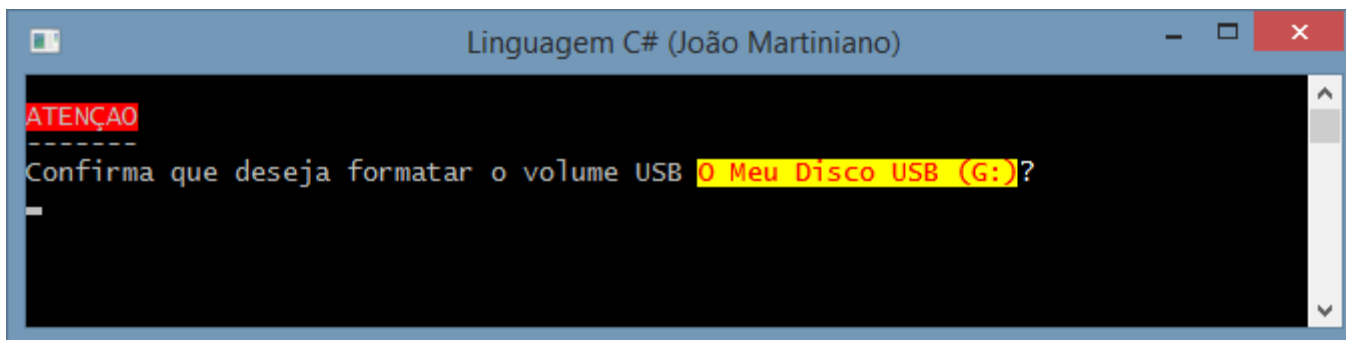
Cor de fundo: A propriedade `BackgroundColor`

- A propriedade `BackgroundColor` permite especificar uma cor de fundo para os caracteres da consola
- Apenas podem ser especificadas cores da enumeração `ConsoleColor`
- Sintaxe:
`Console.BackgroundColor = cor`
- Em que:
`cor` Uma cor da enumeração `ConsoleColor`

Cor de fundo: A propriedade `BackgroundColor`

- Exemplo:

```
Console.WriteLine();  
Console.BackgroundColor = ConsoleColor.Red;  
Console.WriteLine("ATENÇÃO");  
Console.BackgroundColor = ConsoleColor.Black;  
Console.WriteLine("-----");  
Console.Write("Confirma que deseja formatar o volume USB ");  
Console.BackgroundColor = ConsoleColor.Yellow;  
Console.ForegroundColor = ConsoleColor.Red;  
Console.Write("O Meu Disco USB (G:)");  
Console.BackgroundColor = ConsoleColor.Black;  
Console.ForegroundColor = ConsoleColor.White;  
Console.WriteLine("?");
```



Limpar a consola: O método `Clear()`

- O método `Clear()` limpa o conteúdo da janela da consola

- Sintaxe:

`Console.Clear()`

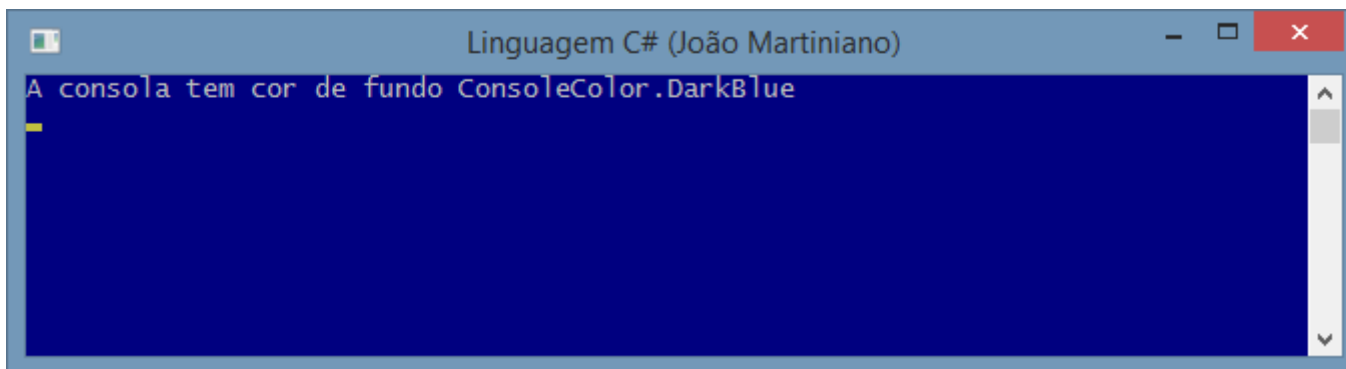
- Exemplo:

```
Console.WriteLine("abc");  
Console.WriteLine("123");  
Console.WriteLine("456");  
Console.WriteLine();  
Console.WriteLine("Pressione qualquer tecla para limpar o conteúdo da  
consola...");  
Console.ReadKey();  
Console.Clear();  
Console.WriteLine("O conteúdo da consola foi limpo");
```

Limpar a consola: O método `Clear()`

- Utilizando este método após a propriedade `BackgroundColor` é possível atribuir uma cor de fundo a toda a janela da consola

```
Console.BackgroundColor = ConsoleColor.DarkBlue;  
Console.Clear();  
Console.WriteLine("A consola tem cor de fundo ConsoleColor.DarkBlue");
```



Posicionar o cursor

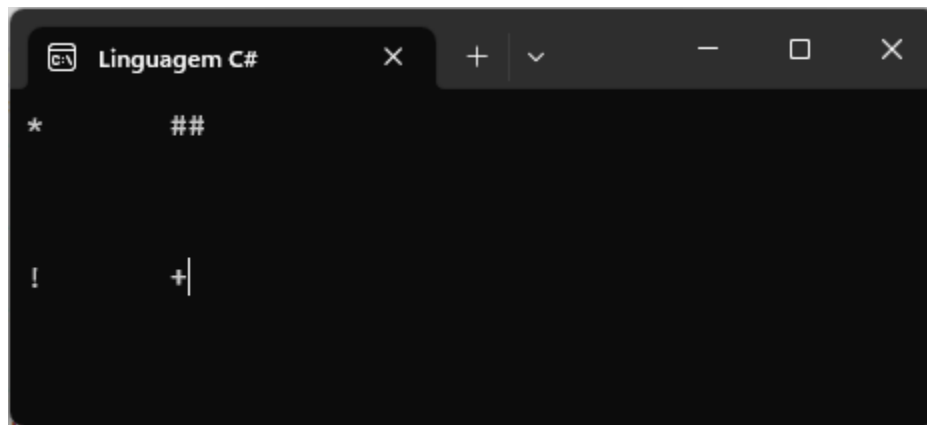
- O método `SetCursorPosition()` permite posicionar o cursor numa determinada coordenada
- Qualquer instrução de *output* é efetuada com base nessa coordenada
- Sintaxe:

`Console.SetCursorPosition(x, y)`

Posicionar o cursor

- Exemplo:

```
Console.SetCursorPosition(0, 0);  
Console.Write("*");  
Console.SetCursorPosition(8, 0);  
Console.Write("##");  
Console.SetCursorPosition(0, 4);  
Console.Write("!");  
Console.SetCursorPosition(8, 4);  
Console.Write("+");
```

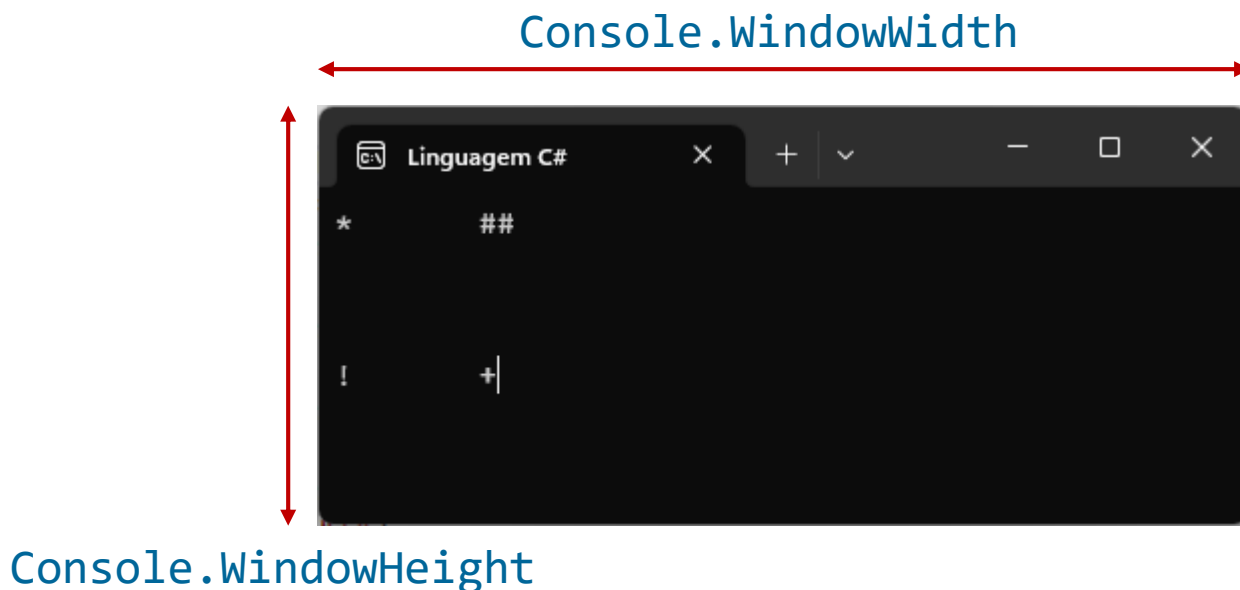


Obter as dimensões da consola

- Para podermos posicionar o texto na consola necessitamos de saber quais as dimensões da janela num dado momento
- Para tal recorremos às seguintes propriedades:

`WindowWidth` = a *width*, ou largura, da consola em colunas

`WindowHeight` = a *height*, ou altura, da consola em linhas



Obter as dimensões da consola

- Exemplo:

```
int width = Console.WindowWidth;  
int height = Console.WindowHeight;
```

| VARIÁVEIS

Variáveis

- As variáveis armazenam informação
- Essa informação é designada por **valor**
- Além disso, uma variável é de um determinado **tipo**
- Ou seja, o tipo de uma variável determina o tipo de informação que esta armazena
- As variáveis podem ser declaradas de duas formas:
 1. Especificando o tipo da variável: declaração **explícita**
 2. Utilizando a instrução **var**: declaração **implícita**
- Se uma variável é declarada utilizando **var**, o sistema infere o tipo da variável de acordo com o valor que lhe é atribuído

Variáveis: Declaração

- Sintaxe para declaração **explícita** de uma variável:

```
tipo nome [= valor];
```

- Em que:

tipo	O tipo da variável
nome	O nome da variável
valor	(Opcional) O valor de inicialização da variável

- Exemplo: declaração simples de uma variável (do tipo **int**)

```
int x;
```

- Exemplo: declaração de uma variável com inicialização de valor

```
int numero_aluno = 14;
```

- Exemplo: declaração de uma variável do tipo **string**

```
string cidade = "Coimbra";
```

Variáveis: Declaração

- As variáveis também podem ser inicializadas com o resultado de uma expressão
- Exemplo: a variável `x` é inicializada com o valor `4`

```
int x = 2 + 2;
```

- É possível declarar e inicializar várias variáveis em simultâneo
- Exemplo: declarar simultaneamente 4 variáveis

```
int a = 1, b = 2, c, d;
```

Nota: este estilo de declaração não é aconselhado por tornar o código mais difícil de ler

Variáveis: Nomes das variáveis

- Um identificador é o nome atribuído a uma variável, a uma classe, a uma propriedade, a um *namespace*, etc.
- Um identificador deve:
 - começar por uma letra ou pelo caracter *underscore* (`_`)
 - ser seguido por uma qualquer combinação de letras, números ou *underscores*
 - pode também conter caracteres Unicode/letras com acentuação

Variáveis: Nomes das variáveis

- Regras importantes:
 - o nome deve ser curto
 - o nome deve indicar o conteúdo da variável
 - os nomes de variáveis são *case-sensitive*

```
string nome = ""; // Nome de variável correto

string Habilitações = "Licenciatura"; // Nome de variável correto

int _log_event_type = 1; // Nome de variável correto

float Q = 1.0F; // Nome de variável correto (caracter Unicode)

string 1ClienteNome = "Ana"; // Nome de variável incorreto: começa com um número
```

Variáveis: Nomes das variáveis

- Adicionalmente, um identificador pode começar com o caracter @ de modo a poder utilizar palavras reservadas

```
string if = "Olá"; // Nome incorreto: a palavra if é reservada  
string @if = "Hello World"; // Nome de variável correto
```

- Este tipo de identificadores são conhecidos como *verbatim identifiers*
- Regra geral deve-se evitar utilizar palavras reservadas em identificadores, mesmo utilizando o prefixo @

Variáveis: Nomes das variáveis

Exercício

- Indique, para cada variável, se o nome está correto ou incorreto:

- | | |
|---------------------------|-----------------------------|
| a) \$NomeCliente | m) _total |
| b) NomeCliente | n) email@variavel.pt |
| c) Nome-Cliente | o) string |
| d) 1b | p) NUMERO_MAXIMO_CARACTERES |
| e) b1 | q) Linha2Coluna3 |
| f) taxa_IVA | r) NumeroAvisos! |
| g) Numero_Fatura_Cliente8 | s) Nome Cliente |
| h) ClassificaçãoAluno | t) @string |
| i) Δ | |
| j) tama\$nh | |
| k) 2_lugar | |
| l) calculoTaxa | |

Variáveis: Nomes das variáveis

Exercício: Resolução

- Indique, para cada variável, se o nome está correto ou incorreto:

a) \$NomeCliente	✗	m) _total	✓
b) NomeCliente	✓	n) email@variavel.pt	✗
c) Nome-Cliente	✗	o) string	✗
d) 1b	✗	p) NUMERO_MAXIMO_CARACTERES	✓
e) b1	✓	q) Linha2Coluna3	✓
f) taxa_IVA	✓	r) NumeroAvisos!	✗
g) Numero_Fatura_Cliente8	✓	s) Nome Cliente	✗
h) ClassificaçãoAluno	✓	t) @string	✓
i) Δ	✓		
j) tama\$nh	✗		
k) 2_lugar	✗		
l) calculoTaxa	✓		

Variáveis: Mostrar o valor

- Para mostrar o valor de uma variável:

```
string cidade = "Coimbra";  
  
Console.WriteLine(cidade);
```


| TIPOS DE DADOS

Tipos de dados

- A linguagem C# possui vários tipos de dados
- Ou seja, o tipo de uma variável determina o tipo de informação que esta armazena
- Por um lado, existem os ***built-in types***, ou seja, **tipos pré-definidos da linguagem**
- Dentro destes, os mais comuns são:

`bool` Valores lógicos (`true` e `false`)

`int` Números inteiros

`float` Números reais

`decimal` Números reais com maior precisão (adequado para efetuar cálculos financeiros e/ou com valores monetários)

`char` Um (e apenas um) caracter

`string` Conjunto de caracteres

- Existem também os ***custom types***, definidos utilizando `struct`, `class`, `interface` e `enum`, e que serão abordados noutros módulos desta disciplina

Tipos de dados

- Exemplo:

```
// Declarar variáveis booleanas
bool a = true;
bool b = false;

// Declarar uma variável inteira
int x = 80;

// Declarar variáveis para armazenar números reais
float taxaIva = 0.23F;
decimal precoProduto = 19.99M;

// Declarar uma variável que armazena texto
string disciplina = "Programação e Sistemas de Informação";
```

Valores **float**:
terminar com **F**

Valores **decimal**:
terminar com **M**

Tipos de dados

- Como C# é uma linguagem *strongly typed*, não é possível atribuir a uma variável de um tipo, valores de outros tipos
- O seguinte exemplo origina um erro:
 - a variável `i` é declarada como sendo do tipo `int`
 - mas é-lhe atribuído um valor booleano (`bool`)

```
// A seguinte instrução irá originar um erro  
int i = true;
```

Tipos de dados

Exercício

- Declare e inicialize variáveis com os seguintes dados:
 - o seu nome
 - a sua idade
 - a sua altura
 - a sua turma
 - o seu número de contribuinte
 - o seu número de cartão de cidadão (completo)
 - é estudante do 10º ano?

Atenção ao nome e ao tipo de cada variável

Variáveis: Tipos de dados

Exercício

- Mostre o conteúdo das variáveis definidas anteriormente, de acordo com o seguinte *output* (os dados mostrados são um exemplo):

```
Nome: Joao Martiniano  
Idade: 30 anos  
Altura: 2,05 m  
Turma: 10 PSI  
NIF: 123456789  
Cartao de cidadao: 12345678 0 ZZ4  
Estudante do 10º ano: False
```

Tipos de dados: `bool`

- O tipo de dados `bool` é o mais simples
- Apenas pode guardar um de dois valores:
 - `true`: significa verdadeiro
 - `false`: significa falso
- Exemplo:

```
bool a = true;  
bool b = false;
```

Tipos de dados

- Tipos de dados **numéricos inteiros**:

Tipo	Gama de valores	Tamanho
sbyte	-128 a 127	8 bits
byte	0 a 255	8 bits
short	-32.768 a 32.767	16 bits
ushort	0 a 65.535	16 bits
int	-2.147.483.648 a 2.147.483.647	32 bits
uint	0 a 4.294.967.295	32 bits
long	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64 bits
ulong	0 a 18.446.744.073.709.551.615	64 bits

Tipos de dados

- Tipos de dados **numéricos de vírgula flutuante**:

Tipo	Gama de valores	Precisão
float	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	~6 a 9 dígitos
double	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	~15 a 17 dígitos
decimal	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9228 \times 10^{28}$	28 a 29 dígitos

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{-4}}_{\text{base}}^{\text{exponent}}.$$

https://en.wikipedia.org/wiki/Floating-point_arithmetic

- Exemplo:

```
float taxaIva = 0.23F;
double altura = 1.85D;
decimal precoProduto = 19.99M;
```

Tipos de dados

- Tipos de dados de **texto**:

Tipo	Descrição	Tamanho
char	Um caracter Unicode Gama de valores: U+0000 to U+FFFF	16 bits
string	Conjunto de caracteres Unicode	

- Exemplo 1: tipo **char**

```
char letra1 = 'A';  
// O mesmo caracter pode ser especificado com o código Unicode  
char letra2 = '\u0041';
```

- Exemplo 2: tipo **string**

```
string texto1 = "Olá";  
// A mesma string pode ser especificada através de caracteres Unicode  
string texto2 = "\u004F\u006C\u00E1";
```

Tipos de dados

- Numa string alguns caracteres e combinações de caracteres têm um significado especial
- Para além disso, certos caracteres para poderem ser utilizados necessitam de ser precedidos de um *escape character*
- Algumas *character escape sequences* que podem ser utilizadas em strings:

<code>\'</code>	<i>Single quote</i> (plica): utilizado em valores do tipo <code>char</code>
<code>\"</code>	<i>Double quote</i> (aspas)
<code>\\</code>	<i>Backslash</i>
<code>\n</code>	<i>New line</i> : força o texto seguinte a começar numa nova linha
<code>\t</code>	<i>Horizontal tab</i>
<code>\uxxxx</code>	Character Unicode (os caracteres <code>xxxx</code> representam o código numérico Unicode)

Tipos de dados

- Exemplo 1:

```
string texto1 = "Eça de Queiroz escreveu \"Os Maias\"";
```

- Resultado:

```
Eça de Queiroz escreveu "Os Maias"
```

- Exemplo 2:

```
string texto2 = "O caracter backslash \\ é diferente\ndo caracter forward  
slash /";
```

- Resultado:

```
O caracter backslash \ é diferente  
do caracter forward slash /
```

Tipos de dados

Exercício

- Declare e inicialize três variáveis do tipo `string` para obter o seguinte resultado:
 - a) Arnold said: "I'll be back"
 - b) Eliminar ficheiros: `del c:\Documentos*.*`
 - c) Greek Small Letter Psi: ψ (código Unicode 03C8)
Ohm Sign: Ω (código Unicode 2126)

Tipos de dados

Exercício: Resolução

- Declare e inicialize três variáveis do tipo `string` para obter o seguinte resultado:
 - a) Arnold said: "I'll be back"
 - b) Eliminar ficheiros: del c:\Documentos*.*
 - c) Greek Small Letter Psi: ψ (código Unicode 03C8)
Ohm Sign: Ω (código Unicode 2126)

```
Console.OutputEncoding = System.Text.Encoding.UTF8;  
  
string a = "Arnold said: \"I'll be back\"";  
string b = "Eliminar ficheiros: del c:\\Documentos\\*.*";  
string c = "Greek Small Letter Psi: \u03C8 (código Unicode 03C8)\nOhm  
Sign: \u2126 (código Unicode 2126)";
```

Tipos de dados

Exercício

- Identifique e corrija os erros no seguinte código:

```
*/ Efetuar um cálculo */  
  
int x = 4, y = 0;  
bool while = 3 + 3  
string morada = 'Avenida "Sá da Bandeira", nº 14';  
decimal preço = 16.40;  
  
CONSOLE.WriteLine(Y);
```

Tipos de dados

Exercício: Resolução

- Erros:

```
*/ Efetuar um cálculo /*  
  
int x = 4, y = 0;  
bool while = 3 + 3  
string morada = 'Avenida "Sá da Bandeira", nº 14';  
decimal preço = 16.40_;  
  
console.WriteLine(Y);
```

- Correção:

```
/* Efetuar um cálculo */  
  
int x = 4, y = 0;  
bool @while;  
string morada = "Avenida 'Sá da Bandeira', nº 14";  
decimal preço = 16.40M;  
  
Console.WriteLine(y);
```


Tipos de dados: Conversões

- A conversão do tipo `string` para tipos numéricos é uma operação bastante habitual
- Pode ser efetuada utilizando a classe `Convert`, a qual contém vários métodos, tais como:

`Convert.ToByte()`: converter para número inteiro

`Convert.ToInt32()`: converter para número inteiro

`Convert.ToDouble()`: converter para double

`Convert.ToDecimal()`: converter para decimal

etc.

- Mas a conversão também pode ser efetuada utilizando os métodos `Parse()` e `TryParse()`

Tipos de dados: Conversões

- O método `TryParse()` está disponível nos tipos numéricos
- Este método tenta efetuar a conversão de um valor do tipo `string`, para um tipo numérico (`byte`, `int`, `float`, `double`, etc.)
- Se a conversão for possível, efetua a mesma e retorna o valor `true`
- Caso contrário retorna o valor `false`
- Sintaxe:
`TryParse(variável_string, out variável)`
- Exemplo 1: Converter a string `s`, para `int` e colocar o resultado na variável `numero`

```
string s = "123";  
int numero;  
  
Int32.TryParse(s, out numero);
```

Tipos de dados: Conversões

- Exemplo 2: Converter a string `s` e verificar se a conversão foi bem sucedida

```
string s = "Abc";  
int numero;  
  
if (Int32.TryParse(s, out numero) == true)  
{  
    Console.WriteLine("A conversão foi bem sucedida");  
}  
else  
{  
    Console.WriteLine("A conversão não foi bem sucedida");  
}
```

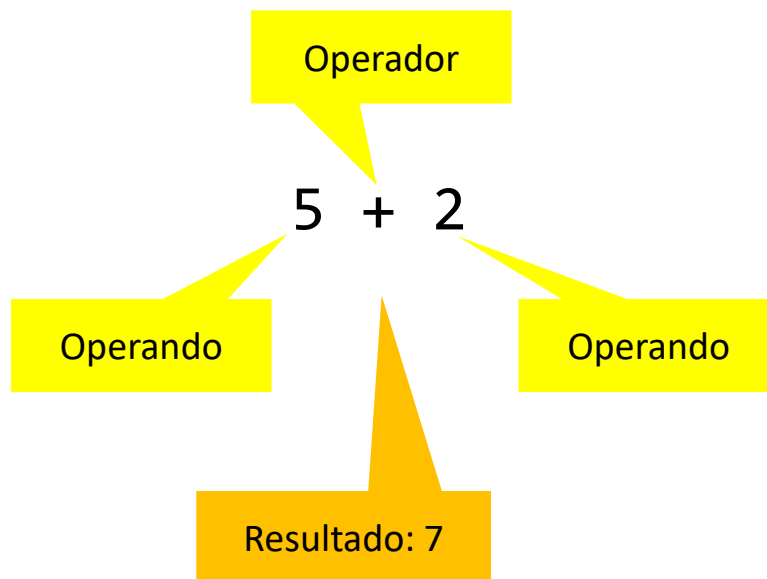
- Exemplo 3: Converter para o tipo `float`

```
if (float.TryParse(s, out numero) == true)  
{  
    Console.WriteLine("A conversão foi bem sucedida");  
}
```

| OPERADORES

Operadores

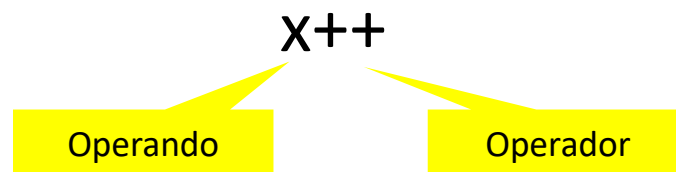
- Os operadores são elementos da linguagem C#, que efetuam uma determinada **operação** sobre os **operandos** e produzem um **resultado**
- Exemplo:



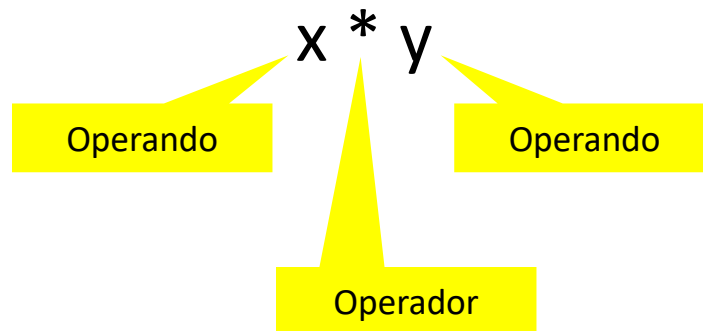
Operadores

- Os operadores podem ser classificados quanto ao número de operandos que aceitam:

- Unários: aceitam apenas um operando



- Binários: aceitam dois operandos



- Ternários: aceitam três operandos

Operadores

- A linguagem C# possui vários tipos de operadores
- Iremos analisar os seguintes tipos de operadores:
 - operadores aritméticos
 - operadores relacionais e de igualdade
 - operadores de atribuição (*assignment*)
 - operadores lógicos

Operadores aritméticos

- Os operadores aritméticos efetuam operações matemáticas utilizando operandos numéricos

Operador	Formato	Designação	Exemplo	Resultado
+	$a + b$	Soma	$5 + 2$	7
-	$a - b$	Subtração	$5 - 2$	3
*	$a * b$	Multiplicação	$5 * 2$	10
/	a / b	Divisão	$5 / 2$	2.5
%	$a \% b$	Resto da divisão	$5 \% 2$	1
-	-a	Negação unária	$-(5 + 2)$	-7

Operadores aritméticos: incremento/decremento

- Existem também operadores aritméticos de incremento e decremento
- Os operadores de incremento e decremento permitem:
 - incrementar uma variável em uma unidade
 - decrementar uma variável em uma unidade
- Exemplo:

```
int x = 5;  
  
// A variável x passa a ter o valor 6  
x++;  
  
// A variável x volta a ter o valor 5  
x--;
```

- Permitem escrever código mais compacto:

$x++$ é equivalente a $x = x + 1$

$x--$ é equivalente a $x = x - 1$

Operadores aritméticos: incremento/decremento

- Operadores de incremento/decremento:

Operador	Designação	Descrição	Exemplo	Resultado
<code>++a</code>	Pré-incremento	Incrementa <code>a</code> e depois retorna <code>a</code>	<code>a = 1;</code> <code>++a + 1;</code>	3
<code>a++</code>	Pós-incremento	Retorna <code>a</code> e só depois incrementa <code>a</code>	<code>a = 1;</code> <code>a++ + 1;</code>	2
<code>--a</code>	Pré-decremento	Decrementa <code>a</code> e depois retorna <code>a</code>	<code>a = 1;</code> <code>--a + 1;</code>	1
<code>a--</code>	Pós-decremento	Retorna <code>a</code> e só depois decrementa <code>a</code>	<code>a = 1;</code> <code>a-- + 1;</code>	2

Operadores aritméticos

Exercício

- Reescreva as seguintes expressões matemáticas, utilizando os operadores aritméticos $+$, $-$, $*$, $/$:

a) $x \times y$

b) $A + B + 10C + 11D$

c) r^2

d) $a^3 + b^4$

e) $\frac{A+B}{3}$

f) $c = \frac{5}{9}(f - 32)$

g) $P = \frac{PR}{1-(1+R)}$

Operadores aritméticos

Exercício: Resolução

- Reescreva as seguintes expressões matemáticas, utilizando os operadores aritméticos $+$, $-$, $*$, $/$:

$a) x \times y$	\rightarrow	$x * y$
$b) A + B + 10C + 11D$	\rightarrow	$A + B + 10 * C + 11 * D$
$c) r^2$	\rightarrow	$r * r$
$d) a^3 + b^4$	\rightarrow	$(a * a * a) + (b * b * b * b)$
$e) \frac{A+B}{3}$	\rightarrow	$(A + B) / 3$
$f) c = \frac{5}{9}(f - 32)$	\rightarrow	$c = (5 / 9) * (f - 32)$
$g) P = \frac{PR}{1-(1+R)}$	\rightarrow	$P = PR / (1 - (1 + R))$

Concatenação de strings

- A **concatenação** (ou junção) de strings é efetuada utilizando o operador **+**

Operador	Formato	Descrição	Exemplo
+	a + b	Concatena a e b	string a = "Hello"; string b = a + " world";

- Exemplo:

```
string a = "Hello";  
string b = a + " world";  
  
Console.WriteLine(b);
```

- Resultado:

```
Hello world
```

Concatenação de strings

- Se um dos operandos não é string, então é automaticamente convertido para string
- Exemplo:

```
int a = 1;  
string b = "1";  
  
Console.WriteLine(a + b);
```

- Resultado:

11

Operadores

Exercício

- Qual o resultado das seguintes expressões?
 - a) `int e = 10;`
`Console.WriteLine(e + 1);`
 - b) `Console.WriteLine(3 + 7);`
 - c) `Console.WriteLine("abc" + "def");`
 - d) `Console.WriteLine((3 * 2) + 10);`
 - e) `Console.WriteLine("123" + " " + "456");`
 - f) `int x = 1;`
`Console.WriteLine(++x - 1);`
 - g) `int y = 1;`
`Console.WriteLine(y-- - 1);`

Operadores

Exercício: Resolução

- Qual o resultado das seguintes expressões?

a) <code>int e = 10;</code> <code>Console.WriteLine(e + 1);</code>	→ 11
b) <code>Console.WriteLine(3 + 7);</code>	→ 10
c) <code>Console.WriteLine("abc" + "def");</code>	→ abcdef
d) <code>Console.WriteLine((3 * 2) + 10);</code>	→ 16
e) <code>Console.WriteLine("123" + " " + "456");</code>	→ 123 456
f) <code>int x = 1;</code> <code>Console.WriteLine(++x - 1);</code>	→ 1
g) <code>int y = 1;</code> <code>Console.WriteLine(y-- - 1);</code>	→ 0

Operadores relacionais e de igualdade

- Estes operadores comparam dois valores
- Retornam um valor booleano: `true` ou `false`

Operador	Designação	Exemplo	Resultado
<code>==</code>	Igualdade: devolve <code>true</code> se os operandos forem iguais	<code>5 == 5</code> <code>1 == 5</code>	<code>true</code> <code>False</code>
<code>!=</code>	Diferente: devolve <code>true</code> se os operandos forem diferentes	<code>1 != 5</code> <code>5 != 5</code>	<code>true</code> <code>false</code>
<code><</code>	Menor que: retorna <code>true</code> se o 1º operando for menor que o 2º	<code>1 < 5</code> <code>5 < 1</code>	<code>true</code> <code>false</code>
<code>></code>	Maior que: retorna <code>true</code> se o 1º operando for maior que o 2º	<code>5 > 1</code> <code>1 > 5</code>	<code>true</code> <code>false</code>
<code><=</code>	Menor ou a igual a: retorna <code>true</code> se o 1º operando for menor ou igual que o 2º	<code>1 <= 5</code> <code>5 <= 5</code> <code>5 <= 1</code>	<code>true</code> <code>true</code> <code>false</code>
<code>>=</code>	Maior ou igual a: retorna <code>true</code> se o 1º operando for maior ou igual que o 2º	<code>5 >= 1</code> <code>5 >= 5</code> <code>1 >= 5</code>	<code>true</code> <code>true</code> <code>false</code>

Operadores de atribuição

- Os operadores de atribuição (*assignment operators*) permitem atribuir valores a variáveis:

Operador	Formato	Equivale a	Exemplo	Resultado
=	a = b		a = 1	1
+=	a += b	a = a + b	a = 1; a += 2;	3
-=	a -= b	a = a - b	a = 1; a -= 2;	-1
*=	a *= b	a = a * b	a = 1; a *= 2;	2
/=	a /= b	a = a / b	a = 1; a /= 2;	0.5
%=	a %= b	a = a % b	a = 2; a %= 2;	0

Operadores lógicos

- Os operadores lógicos (ou booleanos) avaliam os operandos e retornam um valor booleano: `true` ou `false`
- A seguinte tabela mostra os operadores lógicos mais importantes em C#

Operador	Formato	Designação / Descrição	Resultado
!	!a	Negação lógica (NOT) (inverte o valor lógico do operando)	<code>true</code> se <code>a</code> não é <code>true</code>
&&	a && b	Conjunção lógica (AND)	<code>true</code> se <code>a</code> e <code>b</code> forem ambos <code>true</code>
	a b	Disjunção lógica (OR)	<code>true</code> se <code>a</code> ou <code>b</code> for <code>true</code>

Operadores lógicos

Exercício

- Sabendo que $x = 10$ e $y = 5$ determine o valor lógico (True ou False) de cada uma das seguintes expressões:
 - a) $5 \neq 1$
 - b) $5 == 200$
 - c) $5 \leq 200$
 - d) $x > y$
 - e) $(x > 5) \ \&\& \ (x \geq 10)$
 - f) $(x > 0) \ || \ (x > 20)$
 - g) $(x == 10) \ \&\& \ (y \neq 5)$
 - h) $!(y == 5) \ || \ (x \neq 10)$
 - i) $(x \geq 10) \ \&\& \ (y < 5)$
 - j) $!((x > 10) \ \&\& \ (y < 5))$
 - k) $!((x \neq 10) \ || \ (y \leq 5))$
 - l) $(x \geq 5) \ \&\& \ (x < 10) \ || \ (y \geq 0) \ \&\& \ (y \leq 5)$
 - m) $((10 + 5) > 10) \ \&\& \ (10 < (10 - 5))$

Operadores lógicos

Exercício

- Sabendo que $x = 10$ e $y = 5$ determine o valor lógico (True ou False) de cada uma das seguintes expressões:
 - a) $5 \neq 1$ → True
 - b) $5 == 200$ → False
 - c) $5 \leq 200$ → True
 - d) $x > y$ → True
 - e) $(x > 5) \ \&\& \ (x \geq 10)$ → True
 - f) $(x > 0) \ || \ (x > 20)$ → True
 - g) $(x == 10) \ \&\& \ (y \neq 5)$ → False
 - h) $!(y == 5) \ || \ (x \neq 10)$ → False
 - i) $(x \geq 10) \ \&\& \ (y < 5)$ → False
 - j) $!((x > 10) \ \&\& \ (y < 5))$ → True
 - k) $!((x \neq 10) \ || \ (y \leq 5))$ → False
 - l) $(x \geq 5) \ \&\& \ (x < 10) \ || \ (y \geq 0) \ \&\& \ (y \leq 5)$ → True
 - m) $((10 + 5) > 10) \ \&\& \ (10 < (10 - 5))$ → False

Operadores lógicos: Prioridade dos operadores

- Cada operador (não apenas os operadores lógicos) tem uma prioridade, ou ordem pela qual efetuam as operações
- Prioridade dos operadores lógicos:
 - !
 - &&
 - ||
- Por exemplo, no caso da seguinte expressão (em que $x = 10$ e $y = 5$):

1º → $(x \geq 5) \ \&\& \ (x < 10) \ || \ (y \geq 0) \ \&\& \ (y \leq 5)$

$\text{true} \ \&\& \ \text{false} \ || \ \text{true} \ \&\& \ \text{true}$

Operadores lógicos: Prioridade dos operadores

- Cada operador (não apenas os operadores lógicos) tem uma prioridade, ou ordem pela qual efetuam as operações
- Prioridade dos operadores lógicos:
 - !
 - &&
 - ||
- Por exemplo, no caso da seguinte expressão (em que $x = 10$ e $y = 5$):

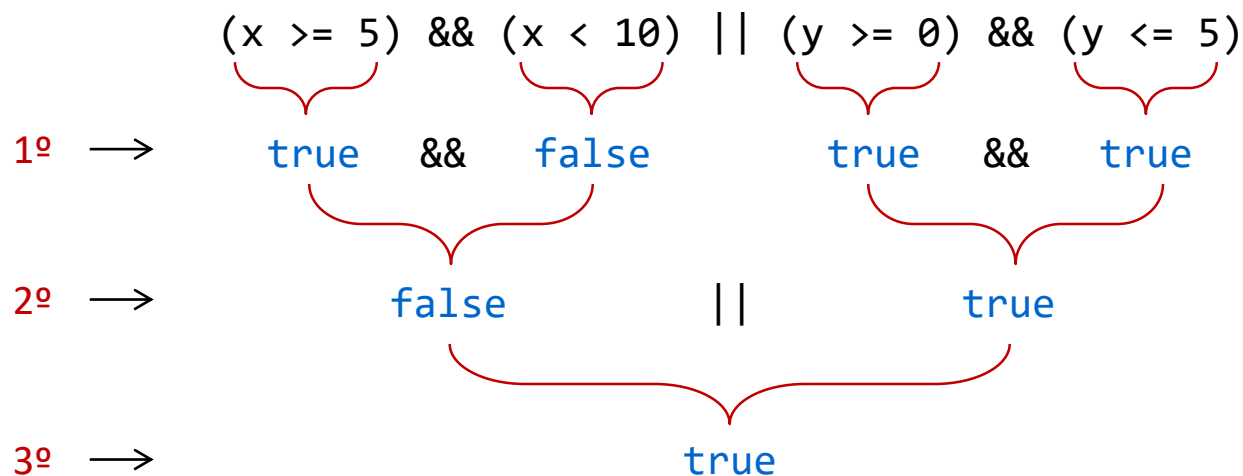
$$(x \geq 5) \ \&\& \ (x < 10) \ || \ (y \geq 0) \ \&\& \ (y \leq 5)$$

$$1^{\circ} \rightarrow \quad \underbrace{\text{true} \ \&\& \ \text{false}}_{\text{false}} \quad || \quad \underbrace{\text{true} \ \&\& \ \text{true}}_{\text{true}}$$

$$2^{\circ} \rightarrow \quad \text{false} \quad || \quad \text{true}$$

Operadores lógicos: Prioridade dos operadores

- Cada operador (não apenas os operadores lógicos) tem uma prioridade, ou ordem pela qual efetuam as operações
- Prioridade dos operadores lógicos:
 - !
 - &&
 - ||
- Por exemplo, no caso da seguinte expressão (em que $x = 10$ e $y = 5$):



| INSTRUÇÕES CONDICIONAIS

Instruções condicionais

- As instruções condicionais permitem condicionar (ou controlar) a execução de uma determinada parte de um programa
- Ou seja, determinados blocos de código só serão executados se determinadas condições forem verdadeiras
- Existem duas instruções condicionais em C#:
 - a instrução `if`
 - a instrução `switch`
- Para além disso, o operador `?:` funciona também como uma instrução condicional

A instrução `if`

- A instrução `if` permite executar uma ou mais instruções, se uma determinada condição for verdadeira

- Sintaxe completa:

```
if (condição)
    instruções
```

```
[else if (condição)
    instruções]
```

```
[else if (condição)
    instruções]
```

```
...
```

```
[else
    instruções]
```

Podem existir várias instruções `else if`

Apenas pode existir uma instrução
`else`

A instrução `if`

- Exemplo 1:

```
int a = 1;  
string numero = "";  
  
if (a == 1)  
    numero = "Um";
```

- Caso fosse executada mais do que uma instrução, seria necessário criar um bloco de código `{ }`
- Exemplo 2:

```
a = 1;  
string numero = "";  
  
if (a == 1)  
{  
    numero = "Um";  
    Console.WriteLine("Obrigado");  
}
```

A instrução `if`

- Utilizando a instrução `else` são executadas instruções caso a condição não seja verdadeira
- Exemplo 3:

```
a = 1;  
string numero = "";  
  
if (a == 1)  
{  
    numero = "Um";  
}  
else  
{  
    numero = "Outro número";  
}
```

A instrução `if`

- Exemplo 4:

```
int a = 1;
string numero = "";

if (a == 1)
{
    numero = "Um";
}
else if (a == 2)
{
    numero = "Dois";
}
else if (a == 3)
{
    numero = "Três";
}
else if (a == 4)
{
    numero = "Quatro";
}
else
{
    numero = "Outro número";
}
```

A instrução `if`

Exercício

- Crie um novo projeto C# no editor Visual Studio, com o nome `ExercicioNumeroParImpar`
- O objetivo deste exercício é escrever um programa que:
 - recebe um número do utilizador (através da consola)
 - verificar se o número inserido é par ou ímpar
 - mostrar ao utilizador (através da consola) o tipo de número que este inseriu
- Para tal copie o seguinte código e complete-o no Visual Studio:

```
  ? numero = 0;  
  
Console.Write("Insira um número: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
/* Insira o resto do código a partir deste ponto */
```

A instrução `if`

Exercício (continuação)

- Para testar se o número é par ou ímpar:
 - se o resto da divisão do número por 2 for igual a 0 (zero), o número é par
 - caso contrário o número é ímpar
- Para calcular o resto da divisão utilize o operador `%`
- Exemplo: `5 % 2`

Insira um número: 5



Insira um número: 5
O número é ímpar

Insira um número: 8



Insira um número: 8
O número é par

A instrução `if`

- Utilizando os operadores lógicos (ou booleanos) é possível conjugar várias condições numa mesma instrução `if`
- Operadores booleanos:

`&&` (AND / E)

- retorna `true` se ambos os operandos forem verdadeiros
- caso contrário retorna `false`

`||` (OR / OU)

- retorna `true` se um dos operandos for verdadeiro
- caso contrário retorna `false`

A instrução `if`

- Exemplo:

```
int numero;  
  
Console.Write("Insira um mês: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
if ((numero >= 1) && (numero <= 12))  
{  
    Console.WriteLine("Obrigado");  
}
```

A instrução `if`

- Exemplo:

```
int numero;  
  
Console.Write("Insira um mês: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
if ((numero >= 1) && (numero <= 12))  
{  
    Console.WriteLine("Obrigado");  
}
```

1ª condição

`numero` deverá ser maior ou igual a 1

A instrução `if`

- Exemplo:

```
int numero;  
  
Console.Write("Insira um mês: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
if ((numero >= 1) && (numero <= 12))  
{  
    Console.WriteLine("Obrigado");  
}
```

2ª condição

`numero` deverá ser menor ou igual a 12

A instrução `if`

- Exemplo:

```
int numero;  
  
Console.Write("Insira um mês: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
if ((numero >= 1) && (numero <= 12))  
{  
    Console.WriteLine("Obrigado");  
}
```

Condição global

`numero` deverá estar entre 1 E 12
(ou seja: deverá ser maior ou igual a 1 E menor ou igual a 12)

A instrução `if`

- Exemplo:

```
int numero;  
  
Console.Write("Insira um mês: ");  
numero = Convert.ToInt32(Console.ReadLine());  
  
if ((numero >= 1) && (numero <= 12))  
{  
    Console.WriteLine("Obrigado");  
}
```

O operador `&&` obriga a que a 1ª e a 2ª condições sejam ambas **verdadeiras**

Ou seja, a mensagem "Obrigado" apenas surge se o número introduzido estiver entre 1 e 12

A instrução `if`

- Acrescentando uma instrução `else`:

```
int numero;

Console.Write("Insira um mês: ");
numero = Convert.ToInt32(Console.ReadLine());

if ((numero >= 1) && (numero <= 12))
{
    Console.WriteLine("Obrigado");
}
else
{
    Console.WriteLine("Introduziu um mês inválido!");
}
```

- **Questão:** o que irá acontecer se:
 - o utilizador inserir o número 1?
 - o utilizador inserir o número 17?

A instrução `if`

Exercício

- Crie um novo projeto C# no editor Visual Studio, com o nome `ExercicioCalculoIMC`
- O objetivo deste exercício é escrever um programa que:
 - recebe o peso e a altura do utilizador (através da consola)
 - calcula o Índice de Massa Corporal (IMC) utilizando a fórmula em baixo
 - mostrar ao utilizador (através da consola) o IMC e uma descrição sobre o IMC
- Para tal copie o código na página seguinte e complete-o no Visual Studio
- O IMC é calculado de acordo com a seguinte fórmula:

$$IMC = \frac{peso}{(altura \times altura)}$$

A instrução `if`

Exercício (continuação)

- Código a copiar e completar:

```
double peso = 0.0D, altura = 0.0D, imc = 0.0D;
string descricaoImc = "";

Console.Write("Introduza o peso: ");
peso = Convert.ToDouble(Console.ReadLine());

Console.Write("Introduza a altura: ");
altura = Convert.ToDouble(Console.ReadLine());

imc = ?;

/* Insira o resto do código a partir deste ponto */
```

A instrução `if`

Exercício (continuação)

- Além do valor numérico do IMC, deverá ser mostrada uma mensagem descritiva, de acordo com a seguinte tabela:

IMC	Descrição
$< 18,5$	Baixo peso
entre 18,5 e 24,9	Normal
entre 25 e 29,9	Pré-obesidade
entre 30 e 34,9	Obesidade grau I
entre 35 e 39,9	Obesidade grau II
≥ 40	Obesidade grau III

A instrução `if`

Exercício (continuação)

- Exemplo da execução do programa:

Introduza o peso: 67,3



Introduza o peso: 67,3
Introduza a altura: 1,70



Introduza o peso: 67,3
Introduza a altura: 1,70

O seu IMC é 23,3 - Normal

A instrução `if`

- Com o operador `||` basta que uma das condições seja verdadeira
- Exemplo:

```
string cidade = "";

Console.Write("Insira uma cidade: ");
cidade = Console.ReadLine();

if ((cidade == "Lisboa") || (cidade == "Coimbra"))
{
    Console.WriteLine("Portugal");
}
```

A instrução `if`

- Com o operador `||` basta que uma das condições seja verdadeira
- Exemplo:

```
string cidade = "";

Console.Write("Insira uma cidade: ");
cidade = Console.ReadLine();

if ((cidade == "Lisboa") || (cidade == "Coimbra"))
{
    Console.WriteLine("Portugal");
}
```

Se cidade é "Lisboa" **OU** "Coimbra"...

A instrução `if`

- Com o operador `||` basta que uma das condições seja verdadeira
- Exemplo:

```
string cidade = "";
```

```
Console.Write("Insira uma cidade: ");  
cidade = Console.ReadLine();
```

```
if ((cidade == "Lisboa") || (cidade == "Coimbra"))  
{  
    Console.WriteLine("Portugal");  
}
```

Se cidade é "Lisboa" OU "Coimbra"...

...mostrar esta mensagem

A instrução `if`

- Uma instrução `if` pode conter mais de 2 condições
- Exemplo:

```
string cidade = "";

Console.Write("Insira uma cidade: ");
cidade = Console.ReadLine();

if ((cidade == "Lisboa") || (cidade == "Coimbra") || (cidade == "Porto"))
{
    Console.WriteLine("Portugal");
}
```

A instrução `if`

Exercício

- Modifique o exemplo anterior, de modo a mostrar para as seguintes cidades, os respetivos países:

Cidade	País
Paris	França
Toulouse	
Roma	Itália
Turim	
Milão	
Oslo	Noruega
Bergen	
Stavanger	
Trondheim	

- Caso o utilizador introduza uma cidade diferente, mostre a mensagem "Cidade não reconhecida!"

A instrução `if`

Exercício

- Escreva, para cada alínea, a instrução `if` correspondente:
 - a) x é diferente de 8
 - b) x é diferente de 6 e 8
 - c) w é igual a 2 ou igual a 9
 - d) y está entre 100 e 200
 - e) preço deverá estar entre 1 e 5, caso contrário, deverá estar entre 5.01 e 100 ou igual a 500
 - f) idade deverá ser inferior a 12 ou superior a 23

A instrução `if`

Exercício: Resolução

- Escreva, para cada alínea, a instrução `if` correspondente:

a) `if (x != 8)`

b) `if ((x != 6) && (x != 8))`

c) `if ((w == 2) || (w == 9))`

d) `if ((y >= 100) && (y <= 200))`

e) `if ((preco >= 1) && (preco <= 5))`

`else if (((preco >= 5.01) && (preco <= 100)) || (preco == 500))`

f) `if ((idade < 12) || (idade > 23))`

A instrução `switch`

- A instrução `switch` é semelhante a um conjunto de instruções `if`
- Tem a vantagem de tornar o código mais fácil de ler
- Sintaxe:

```
switch (variável)
{
    case valor1:
        instruções;
        break;
    case valor2:
        instruções;
        break;
    ...
    [default:
        instruções;
        break;]
}
```

A instrução `break` **não** é opcional e serve para que a execução não continue no ramo `case` seguinte.

Opcional. Instruções que são executadas caso o valor não esteja contemplado acima.

A instrução `switch`

- Exemplo: comparação com a instrução `if`

```
if (a == 1)
{
    numero = "Um";
}
else if (a == 2)
{
    numero = "Dois";
}
else if (a == 3)
{
    numero = "Três";
}
else
{
    numero = "Outro número";
}
```

```
switch (a)
{
    case 1:
        numero = "Um";
        break;
    case 2:
        numero = "Dois";
        break;
    case 3:
        numero = "Três";
        break;
    default:
        numero = "Outro número";
        break;
}
```

A instrução `switch`

- É possível testar vários valores utilizando o seguinte estilo de programação:

```
switch (a)
{
    case 0:
    case 1:
    case 2:
        numero = "Menor ou igual a 2";
        break;
    case 3:
    case 4:
    case 5:
    case 6:
        numero = "Entre 3 e 6";
        break;
}
```

Se $a \geq 0$ e $a \leq 2$

Se $a \geq 3$ e $a \leq 6$

A instrução `switch`

Exercício

- Modifique o exercício efetuado anteriormente com a instrução `if`, utilizando a instrução `switch`

Cidade	País
Paris	França
Toulouse	
Roma	Itália
Turim	
Milão	
Oslo	Noruega
Bergen	
Stavanger	
Trondheim	

- Caso o utilizador introduza uma cidade diferente, mostre a mensagem "Cidade não reconhecida!"

O operador ? :

- Este operador é um operador ternário condicional (utiliza três operandos)
- Sintaxe:
`expressão1 ? expressão2 : expressão3;`
- Funciona da seguinte forma:
 - avalia a expressão `expressão1`
 - se o resultado de `expressão1` for verdadeiro, retorna o valor de `expressão2`
 - se o resultado de `expressão1` for falso, retorna o valor de `expressão3`
- Exemplo: verificar se uma variável contém um número par ou ímpar

```
int numero = 6;  
string parImpar = "";  
  
parImpar = ((numero % 2) == 0) ? "Número par" : "Número ímpar";
```

O operador ? :

- Este operador é um operador ternário condicional (utiliza três operandos)
- Sintaxe:
`expressão1 ? expressão2 : expressão3;`
- Funciona da seguinte forma:
 - avalia a expressão `expressão1`
 - se o resultado de `expressão1` for verdadeiro, retorna o valor de `expressão2`
 - se o resultado de `expressão1` for falso, retorna o valor de `expressão3`
- Exemplo: verificar se uma variável contém um número par ou ímpar

```
int numero = 6;  
string parImpar = "";  
parImpar = ((numero % 2) == 0) ? "Número par" : "Número ímpar";
```

O resto da divisão por 2 é zero?

O operador ? :

- Este operador é um operador ternário condicional (utiliza três operandos)
- Sintaxe:
`expressão1 ? expressão2 : expressão3;`
- Funciona da seguinte forma:
 - avalia a expressão `expressão1`
 - se o resultado de `expressão1` for verdadeiro, retorna o valor de `expressão2`
 - se o resultado de `expressão1` for falso, retorna o valor de `expressão3`
- Exemplo: verificar se uma variável contém um número par ou ímpar

```
int numero = 6;  
string parImpar = "";  
parImpar = ((numero % 2) == 0) ? "Número par" : "Número ímpar";
```

O resto da divisão por 2 é zero?

Se sim, retorna "Número par"

O operador ? :

- Este operador é um operador ternário condicional (utiliza três operandos)
- Sintaxe:
`expressão1 ? expressão2 : expressão3;`
- Funciona da seguinte forma:
 - avalia a expressão `expressão1`
 - se o resultado de `expressão1` for verdadeiro, retorna o valor de `expressão2`
 - se o resultado de `expressão1` for falso, retorna o valor de `expressão3`
- Exemplo: verificar se uma variável contém um número par ou ímpar

```
int numero = 6;  
string parImpar = "";  
parImpar = ((numero % 2) == 0) ? "Número par" : "Número ímpar";
```

O resto da divisão por 2 é zero?

Se sim, retorna "Número par"

Se não, retorna "Número ímpar"

O operador ? :

- O código equivalente, utilizando a instrução `if...else`:

```
int numero = 6;
string parImpar = "";

if ((numero % 2) == 0)
{
    parImpar = "Número par";
}
else
{
    parImpar = "Número ímpar";
}
```

INSTRUÇÕES REPETITIVAS

Instruções repetitivas

- As instruções repetitivas permitem executar instruções de forma **cíclica**
- A cada repetição dá-se o nome de **iteração**
- Existem 4 instruções repetitivas em C#:
 - a instrução `while`
 - a instrução `do...while`
 - a instrução `for`
 - a instrução `foreach`

A instrução `while`

- A instrução `while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**

- Sintaxe:

```
while (condição)
    instruções
```

- Exemplo:

```
int i = 1;

while (i <= 3)
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
```

A instrução `while`

- A instrução `while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**

- Sintaxe:

```
while (condição)
    instruções
```

- Exemplo:

```
int i = 1;

while (i <= 3)
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
```

Inicializar o contador com o valor 1

A instrução `while`

- A instrução `while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**

- Sintaxe:

```
while (condição)
    instruções
```

- Exemplo:

```
int i = 1;

while (i <= 3)
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
```

Enquanto o contador for
menor ou igual a 3...

A instrução `while`

- A instrução `while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**

- Sintaxe:

```
while (condição)
    instruções
```

- Exemplo:

```
int i = 1;

while (i <= 3)
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
```

Enquanto o contador for menor ou igual a 3...

...executar esta instrução

A instrução `while`

- A instrução `while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**

- Sintaxe:

```
while (condição)
    instruções
```

- Exemplo:

```
int i = 1;
```

```
while (i <= 3)
```

```
{
```

```
    Console.WriteLine("i = {0}", i);
```

```
    ++i;
```

```
}
```

Enquanto o contador for
menor ou igual a 3...

...executar esta instrução

Incrementar o contador
em uma unidade

A instrução `while`

Exercício

- Modifique o exemplo anterior e mostre os números de 1 a 10

A instrução `while`

Exercício: Resolução

- Modifique o exemplo anterior e mostre os números de 1 a 10

```
int i = 1;

while (i <= 10)
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
```

A instrução `while`

Exercício

- Mostre os números de 5 a -15

A instrução `while`

Exercício: Resolução

- Mostre os números de 5 a -15

```
int i = 5;

while (i >= -15)
{
    Console.WriteLine("i = {0}", i);
    --i;
}
```

A instrução `while`

Exercício

- Mostre os números entre 10 e 54, dois a dois
- Exemplo:

10

12

14

...

54

A instrução `while`

Exercício: Resolução

- Mostre os números entre 10 e 54, dois a dois
- Exemplo:

10

12

14

...

54

```
int i = 10;

while (i <= 54)
{
    Console.WriteLine(i);
    i += 2;
}
```


A instrução `do...while`

- À semelhança da instrução `while`, a instrução `do...while` permite executar uma ou mais instruções, **enquanto** uma determinada **condição** for **verdadeira**
- A diferença é que a condição é avaliada **no final** de cada iteração

- Sintaxe:

```
do
    instruções
while (condição)
```

- Exemplo:

```
int i = 1;

do
{
    Console.WriteLine("i = {0}", i);
    ++i;
}
while (i <= 3);
```

Executar esta instrução...

...enquanto o contador for
menor ou igual a 3

A instrução `do...while`

Exercício

- Utilizando a instrução `do...while`, mostre a tabuada de 1 a 10, para o número 8
- Exemplo:

```
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
...
8 x 10 = 80
```

A instrução `do...while`

Exercício: Resolução

- Utilizando a instrução `do...while`, mostre a tabuada de 1 a 10, para o número 8
- Exemplo:

```
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
...
8 x 10 = 80
```

- Resolução:

```
int i = 1;

do
{
    Console.WriteLine("8 x {0} = {1}", i, 8 * i);
    ++i;
} while (i <= 10);
```

A instrução for

- Sintaxe:

```
for ([inicialização]; [condição]; [incremento])  
    instruções
```

- A expressão de inicialização é executada no início (apenas uma vez)
- A condição é avaliada **no início** de cada iteração
- De seguida são executadas as instruções
- No final de cada iteração é executada a expressão de incremento
- Exemplo:

```
int i;  
  
for (i = 1; i <= 3; ++i)  
{  
    Console.WriteLine("i = {0}", i);  
}
```

A instrução `for`

Exercício

- Utilizando a instrução `for`, modifique a resolução do exercício anterior (instrução `do...while`) e mostre a tabuada de 1 a 16, para o número 7
- Exemplo:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 16 = 112
```

A instrução `for`

Exercício: Resolução

- Utilizando a instrução `for`, modifique a resolução do exercício anterior (instrução `do...while`) e mostre a tabuada de 1 a 16, para o número 7
- Exemplo:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 16 = 112
```

- Resolução:

```
int i;

for (i = 1; i <= 16; ++i)
{
    Console.WriteLine("7 x {0} = {1}", i, 7 * i);
}
```

A instrução `break`

- Esta instrução, além de ser utilizada na instrução `switch`, também pode ser utilizada em ciclos (`while`, `do...while`, `for`, `foreach`)
- Provoca o fim imediato de um ciclo
- Exemplo:
 - o ciclo seguinte deveria mostrar os números entre 1 e 10
 - no entanto, ao ser atingido o número 4, o ciclo é terminado

```
int i = 1;

while (i <= 10)
{
    Console.WriteLine("i = {0}", i);

    if (i == 4)
        break;

    ++i;
}
```

 Terminar o ciclo

A instrução `foreach`

- É uma instrução adequada para percorrer os elementos de arrays e coleções, e executar operações para cada um desses elementos

- Sintaxe:

```
foreach (tipo variável in expressão)
    instruções
```

- Exemplo:

```
string[] cidades = new string[] { "Paris", "Toulouse", "Oslo", "Bergen",
    "Stavanger", "Ljubljana", "Maribor", "Bled" };

foreach (string cidade in cidades)
{
    Console.WriteLine(cidade);
}
```


A instrução foreach

- Explicação do exemplo:

```
string[] cidades = new string[] { "Paris", "Toulouse", "Oslo", "Bergen",  
"Stavanger", "Ljubljana", "Maribor", "Bled" };
```

```
foreach (string cidade in cidades)  
{  
    Console.WriteLine(cidade);  
}
```

Em primeiro lugar é definido um array de strings,
com nomes de cidades

A instrução `foreach`

- Explicação do exemplo:

```
string[] cidades = new string[] { "Paris", "Toulouse", "Oslo", "Bergen",  
    "Stavanger", "Ljubljana", "Maribor", "Bled" };  
  
foreach (string cidade in cidades)  
{  
    Console.WriteLine(cidade);  
}
```

O ciclo `foreach` vai percorrer os elementos do array `cidades`:
cada elemento fica numa variável do tipo `string`, chamada `cidade`

A instrução `foreach`

- Explicação do exemplo:

```
string[] cidades = new string[] { "Paris", "Toulouse", "Oslo", "Bergen",  
    "Stavanger", "Ljubljana", "Maribor", "Bled" };  
  
foreach (string cidade in cidades)  
{  
    Console.WriteLine(cidade);  
}
```

O array tem 8 elementos, logo o ciclo `foreach` tem 8 iterações:
em cada iteração é mostrado valor da variável `cidade`

A instrução `foreach`

- Resultado do exemplo:

