

Abordaremos os seguintes tópicos na aula de hoje:

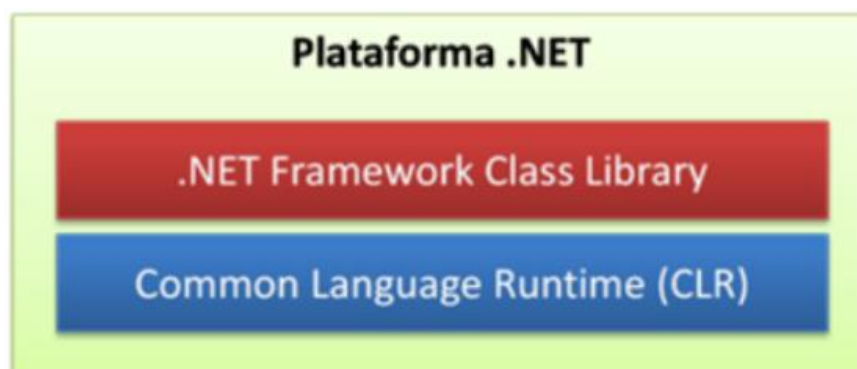
- 1 - A plataforma .Net
- 2 - Sintaxe básica C# usando console
- 3 - Estrutura de um programa C#
- 4 – Variáveis e Constantes
- 5 - Tipos de dados
- 6 - Condicionais

1 - Na plataforma .Net abordaremos os tópicos:

- 1.1 - Definição e benefícios
- 1.2 - Interoperabilidade entre linguagens
- 1.3 - Compilação e execução
- 1.4 - Common Type System (CTS)
- 1.5 - Common Language Specification (CLS)

1.1 - Definição e Benefícios na plataforma .Net

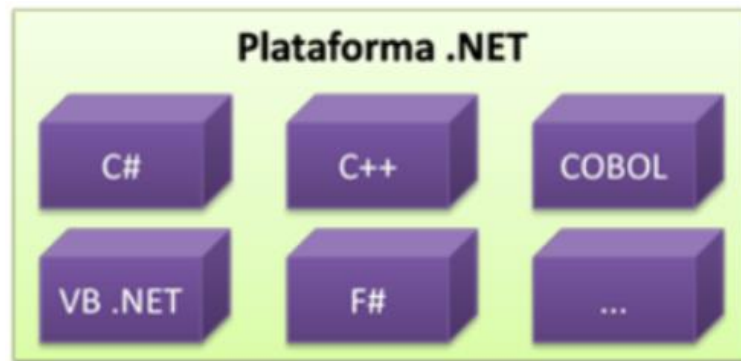
- Foi criada em 2002 pela Microsoft.
- É um ambiente de execução gerenciado
 - Fornece uma série de serviços e aplicações
- Composta por dois principais componentes
 - Common Language Runtime (CLR) – Uma máquina virtual responsável por executar o código e coordenar os serviços que serão utilizados a aplicação.
 - .NET Framework Class Library – Um conjunto de bibliotecas que auxiliam no desenvolvimento de aplicações.



1.2 - Interoperabilidade entre linguagens

A arquitetura da plataforma .NET permite a interoperabilidade entre linguagens de programação.

- É possível “misturar linguagens de programação na mesma aplicação.”

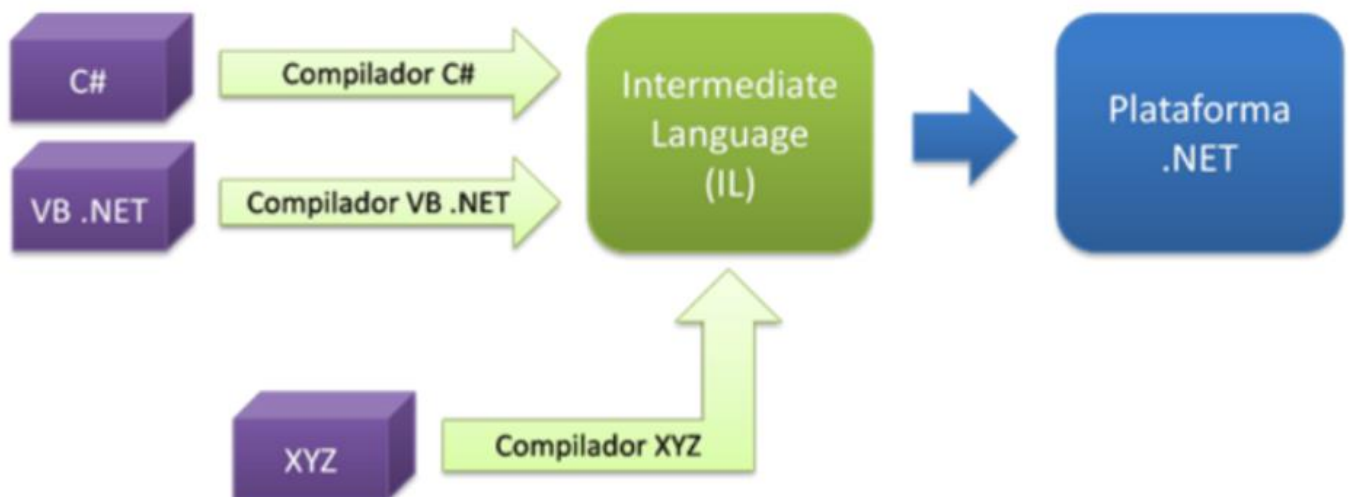


Para que a interoperabilidade funcione, todas as linguagens são compiladas para outra linguagem, chamada IL – Intermediate Language.

Quando você compila um código C# ou VB .NET é gerada uma linguagem intermediária IL para ser executada dentro da plataforma .NET

IL ou CIL – Common Intermediate Language ou MSIL – Microsoft Intermediate Language.

O .NET executa código compilado em IL.



Se surgir uma outra linguagem XYZ, será criado um outro compilador XYZ que gerará a IL.

1.3 - Compilação e execução

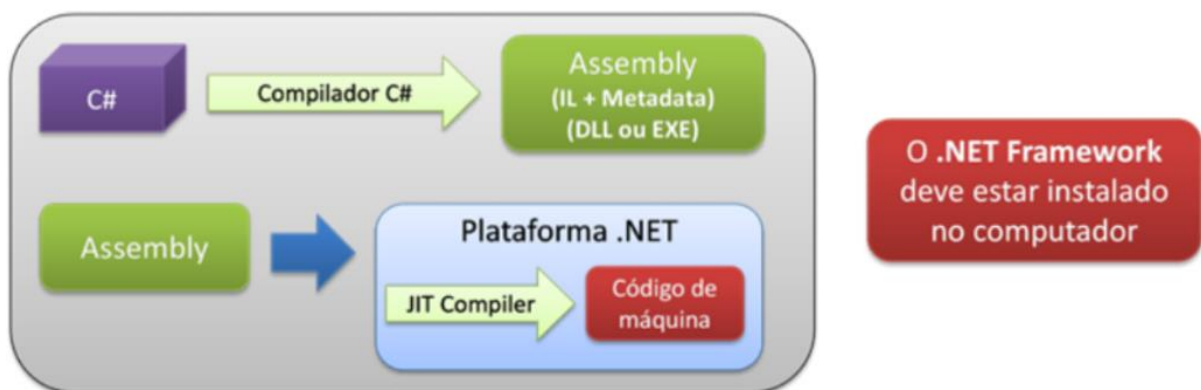
Modelo não gerenciado

Nesse modelo é compilado o código e então é gerado um arquivo DLL ou um EXE em código de máquina para ser executado no processador.



Modelo gerenciado

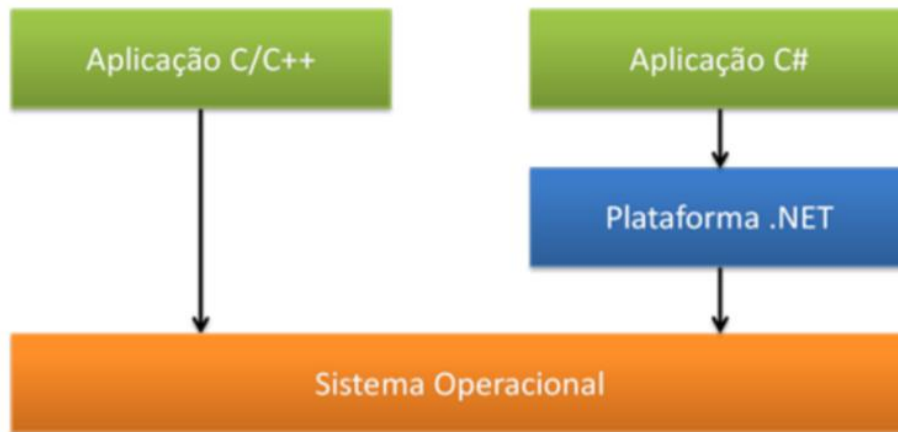
A linguagem C# é compilada e gerada a IL + Metadata. Esse conjunto é chamado de Assembly que é um arquivo DLL ou EXE.



Dentro da plataforma .NET o Assembly é compilado pelo JIT Compiler – Just In Time. Ele compila durante a execução da aplicação para o Código de máquina.

É necessário que tenha instalado no computador o .NET Framework para executar as aplicações.

As aplicações não gerenciadas são executadas pelo sistema operacional. Nas aplicações gerenciadas, no caso do C#, a plataforma .NET é que executa a aplicação.



Por esse motivo é que a plataforma consegue fornecer serviços para a aplicação, pois ela tem o controle do que está sendo executado, como segurança e gerenciamento de memória.

1.4 - Common Type System (CTS)

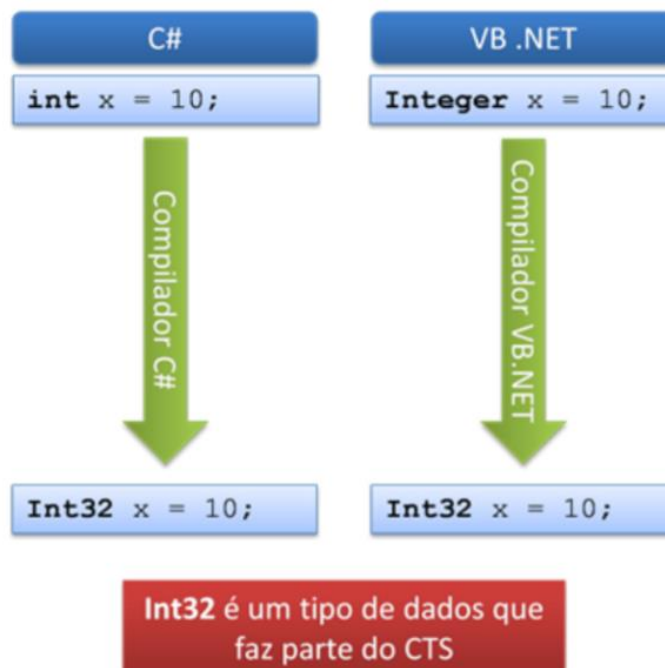
Se cada linguagem de programação definisse seus próprios tipos de dados, a interoperabilidade seria impossível.



Para resolver esse problema, a .NET possui o CTS (Common Type System)

- Os tipos de dados são definidos pela .NET.
- As linguagens mapeiam seus próprios tipos para tipos definidos no CTS

Na figura abaixo, a IL é que faz a conversão dos dados `int x` e `Integer x` para `Int32` que é um tipo de dado do CTS.



Outros exemplos.

C#	VB .NET	CTS
int	Integer	Int32
short	Short	Int16
string	String	String
double	Double	Double
...

1.5 - Common Language Especification (CTS)

Para que as linguagens de programação possam “conversar” na plataforma .NET, elas devem seguir regras:

- Uso de orientação a objetos.
- Herança simples entre classes (Não pode ser herança múltipla).
- Tipos de dados numéricos com sinal

Uma linguagem pode ter elementos que fogem da especificação da CLS. Caso isso aconteça, a interoperabilidade com outras linguagens pode ficar prejudicada.

2 - Sintaxe básica C# usando console

Tópicos do ambiente de desenvolvimento e primeira aplicação

- 2.1 - .NET Framework
- 2.2 - Como desenvolver aplicações C#
- 2.3 - Visual Studio Community
- 2.4 - Preparando o Ambiente de Desenvolvimento
- 2.5 - Sua primeira aplicação C#
- 2.6 - Explorando o Visual Studio

2.1 - .NET Framework

Para que o desenvolvimento de aplicações em C# seja possível, o **.NET Framework** deve ser instalado:

<https://dotnet.microsoft.com/download/dotnet-framework>

O .NET Framework contém as bibliotecas necessárias para execução de código e também o CLR (Common Language Runtime).

Ele contém o compilador do C#

- csc.exe

2.2 - Como desenvolver aplicações C#

Para desenvolver aplicações C# você pode usar:

- Qualquer editor de texto.
 - Ex. bloco de notas, Notepad++
 - Depois de criado o código fonte da aplicação, o compilador C# é usado para gerar o assembly
- Ambientes integrados de desenvolvimento (IDE - Integrated Development Environment).
 - Ex. Visual Studio – principal ferramenta para C#
 - Essas ferramentas facilitam o processo de escrita do código fonte e de execução.
 - Possuem outros recursos muito úteis, como debug de código, code completion (auto complete - sugestão de código)

2.3 - Visual Studio Community

- Versão simplificada do Visual Studio
- Voltada para entusiastas e estudantes
- Gratuito

2.4 - Preparando o Ambiente de Desenvolvimento

Baixando e instalando o Visual Studio Community.

Vá até a página do Visual Studio:

<https://visualstudio.microsoft.com/pt-br/vs/older-downloads/>

Escolha a opção 2017 e clique em baixar.



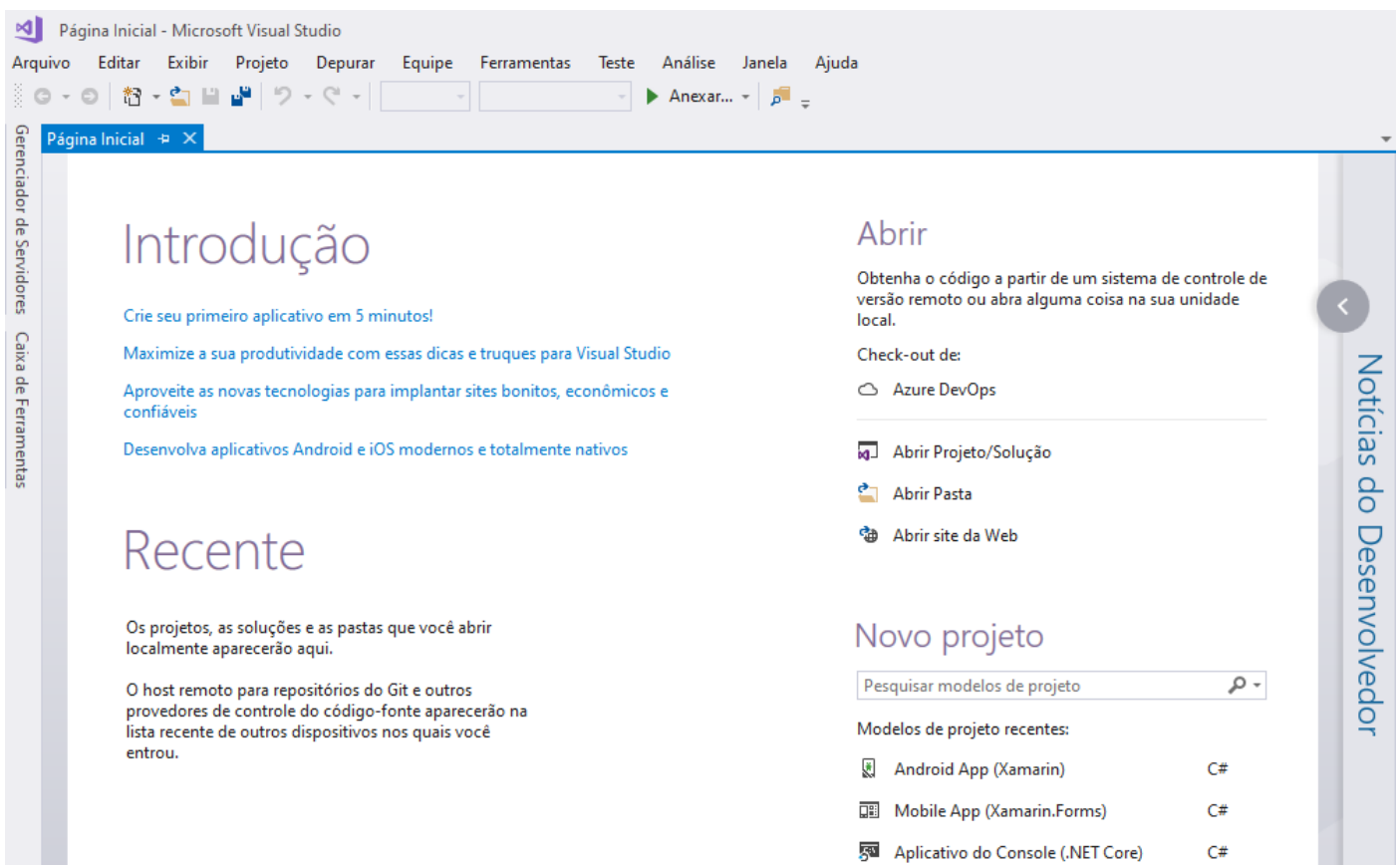
Role a página até o item Visual Studio Community 2017 e clique em Download.



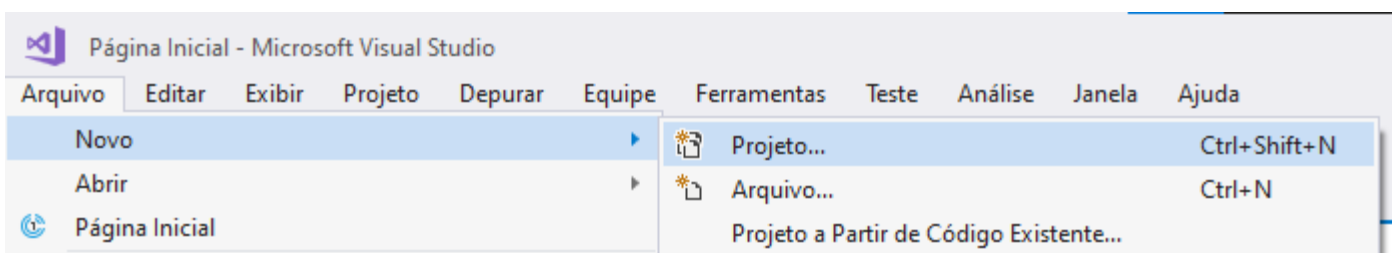
A partir daí é só executar o programa baixado e seguir as instruções.

2.5 - Sua primeira aplicação C#

Vamos abrir o Visual Studio.

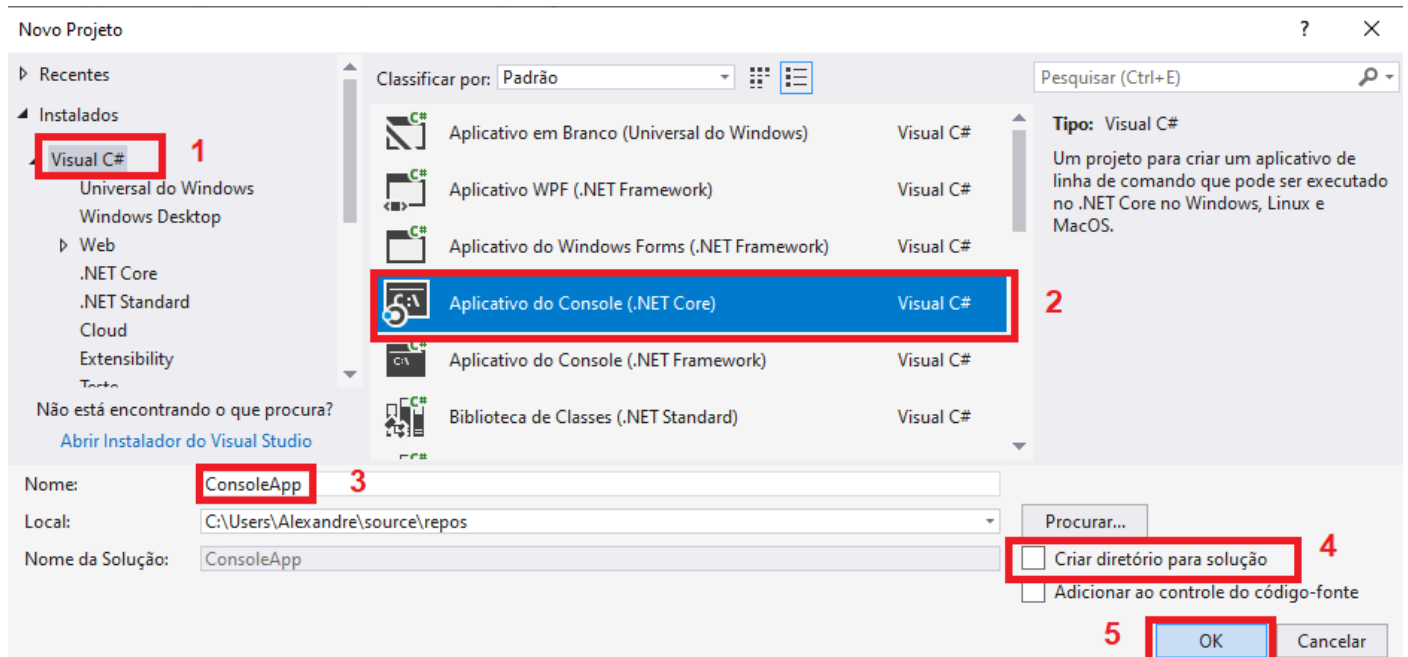


E iniciar um novo projeto em Arquivo > Novo > Projeto...

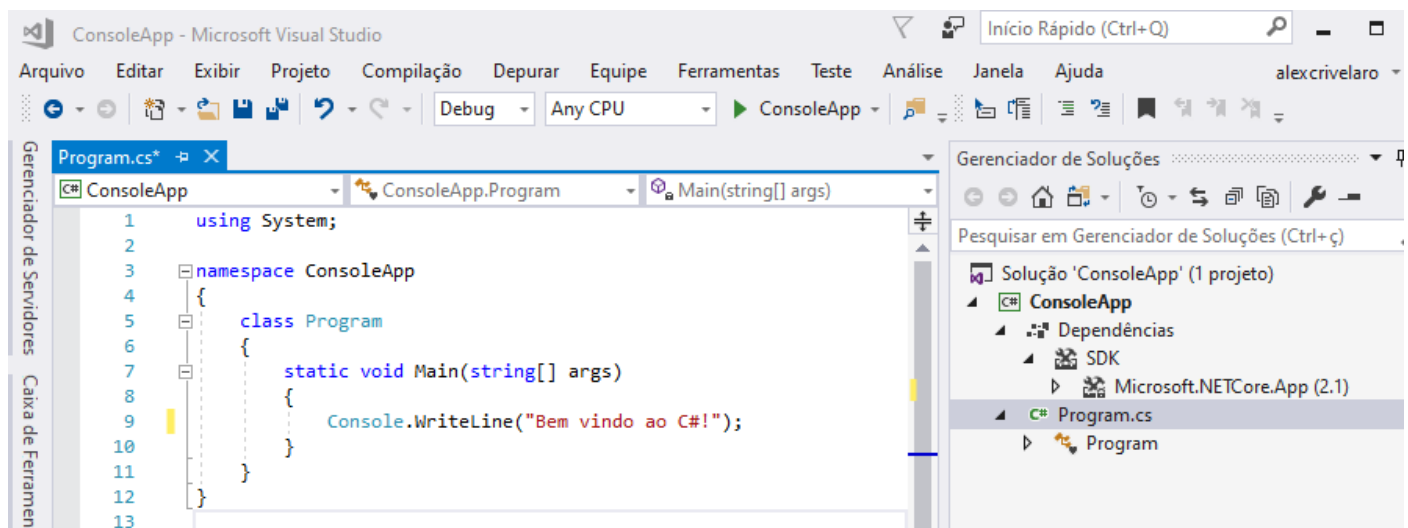


Clicaremos em **Visual C# > Aplicativo do Console (.NET Core)**

Renomear para **ConsoleApp** e desmarcar **Criar diretório para solução** pois aqui iremos utilizar uma única solução e finalizar com **OK**.

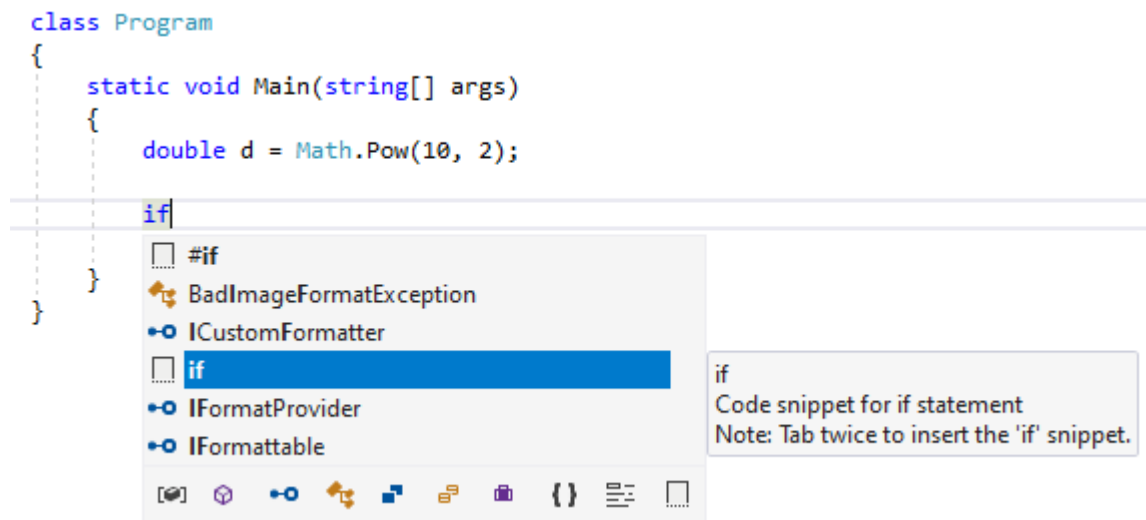


Nossa aplicação terá a seguinte aparência após colocarmos a mensagem de "Bem-vindo ao C#!"



2.6 - Explorando o Visual Studio

Existe o recurso auto complete. Um exemplo é quando utilizamos a função if e ele pode completar o código pressionando a tecla <TAB> duas vezes.



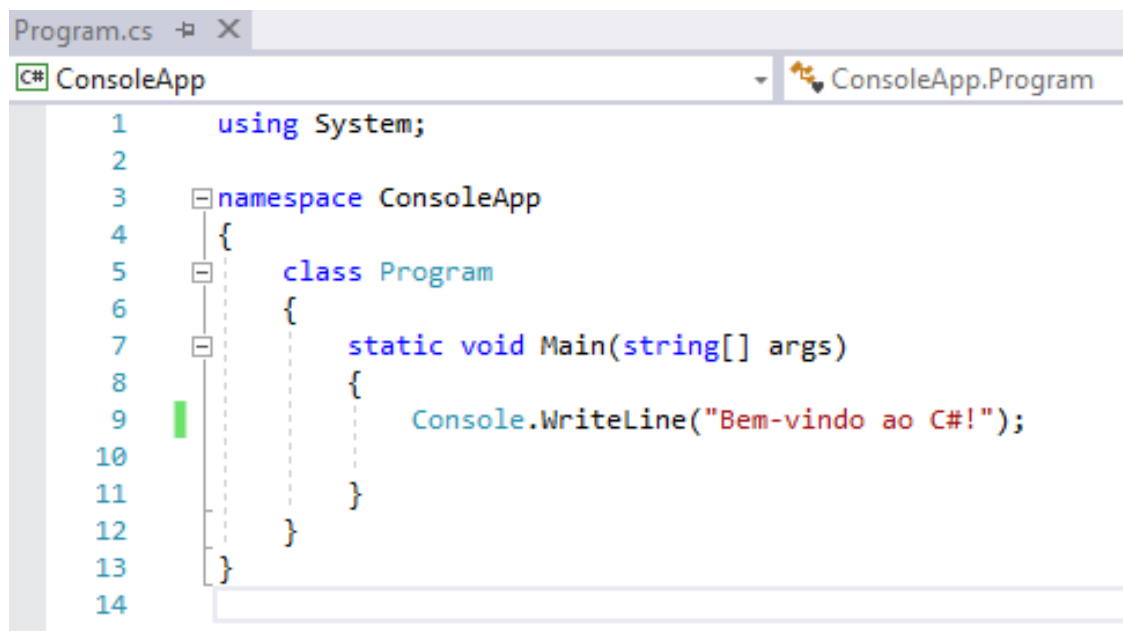
```
class Program
{
    static void Main(string[] args)
    {
        double d = Math.Pow(10, 2);

        if (true)
        {
        }
    }
}
```

Funciona também para outras funções como o for.

3 - Estrutura de um programa C#,

Vamos entender as partes do programa:



```
Program.cs - X
C# ConsoleApp ConsoleApp.Program
1  using System;
2
3  namespace ConsoleApp
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Bem-vindo ao C#!");
10         }
11     }
12 }
13
14
```

`using System;` - Biblioteca para utilizar a instrução

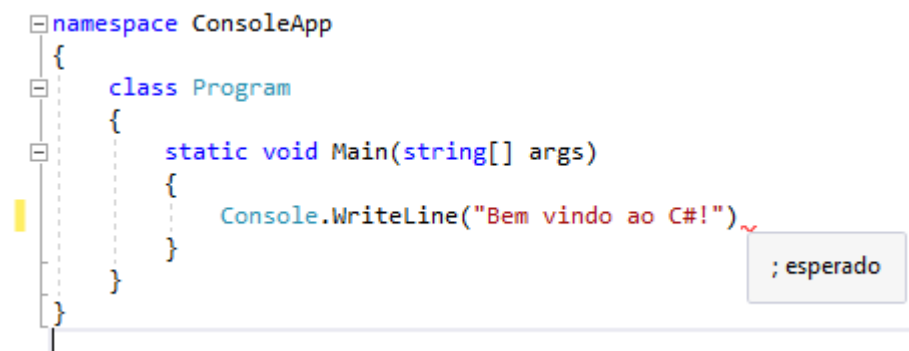
`Console.WriteLine("Bem-vindo ao C#!");` que mostra a mensagem na tela.

`namespace ConsoleApp` – é usada para declarar um escopo (abrangência, que envolve) que contém um conjunto de objetos relacionados.

`class Program` - Classe do programa.

`static void Main(string[] args)` - Main é o programa principal. O ponto de entrada da execução do programa.

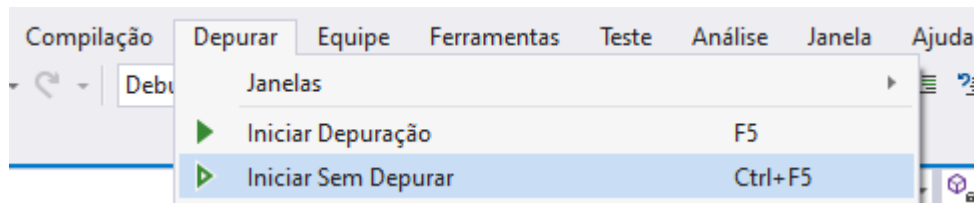
Se eu esquecer um ponto e vírgula (;) no final da instrução o programa irá acusar esse erro.



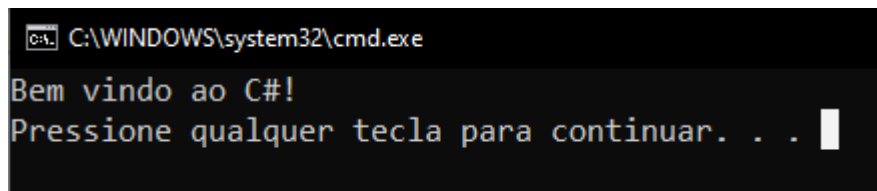
```
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Bem vindo ao C#!")
        }
    }
}
```

; esperado

Podemos então executar nossa aplicação em Depurar > Iniciar Sem Depurar



Esse processo automático envolve: salvar o projeto, compilar e executar.



Você pode notar que todos os menus estão em português.

É uma boa prática deixar tudo em inglês, pois dessa forma facilita o aprendizado quando você consultar outras fontes de informação em inglês.

Para mudar o idioma do seu Visual Studio, abra o Visual Studio Installer e clique no botão modificar.

Feche o Visual Studio e abra o Visual Studio Installer.



Clique em Pacote de Idiomas e marque a opção Inglês e depois clique no botão Modificar.

Você pode adicionar pacotes de idiomas adicionais à instalação do Visual Studio.

- ☐ Chinês (Tradicional)
- ☐ Coreano
- ☐ Espanhol
- ☐ Francês
- ☒ Inglês
- ☐ Italiano
- ☐ Japonês
- ☐ Polonês
- ☒ Português (Brasil)
- ☐ Russo
- ☐ Tcheco
- ☐ Turco

Detalhes da instalação

- ▶ Editor de núcleo do Visual Studio
- ▶ Desenvolvimento para desktop com
- ▶ Desenvolvimento com a Plataforma...
- ▶ ASP.NET e desenvolvimento Web
- ▶ Processamento e armazenamentos d...
- ▶ Desenvolvimento de extensão do Vis...
- ▼ .NET Core cross-platform developme...
 - ▼ Incluído
 - ✓ .NET Core 2.1 development tools (out of s...
 - ✓ Ferramentas de desenvolvimento do .NET...
 - ✓ Pré-requisitos de ASP.NET e de ferramenta...
 - ▼ Opcional
 - ▼ Ferramentas de desenvolvimento para dispositivos móveis

Localização

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community

Ao continuar, você concorda com a licença [licença](#) da edição do Visual Studio selecionada. Também oferecemos a possibilidade de baixar outro software com o Visual Studio. Esse software é licenciado separadamente, conforme [3rd Party Notices](#) ou na licença que o acompanha. Ao continuar, você também concorda com essas licenças.

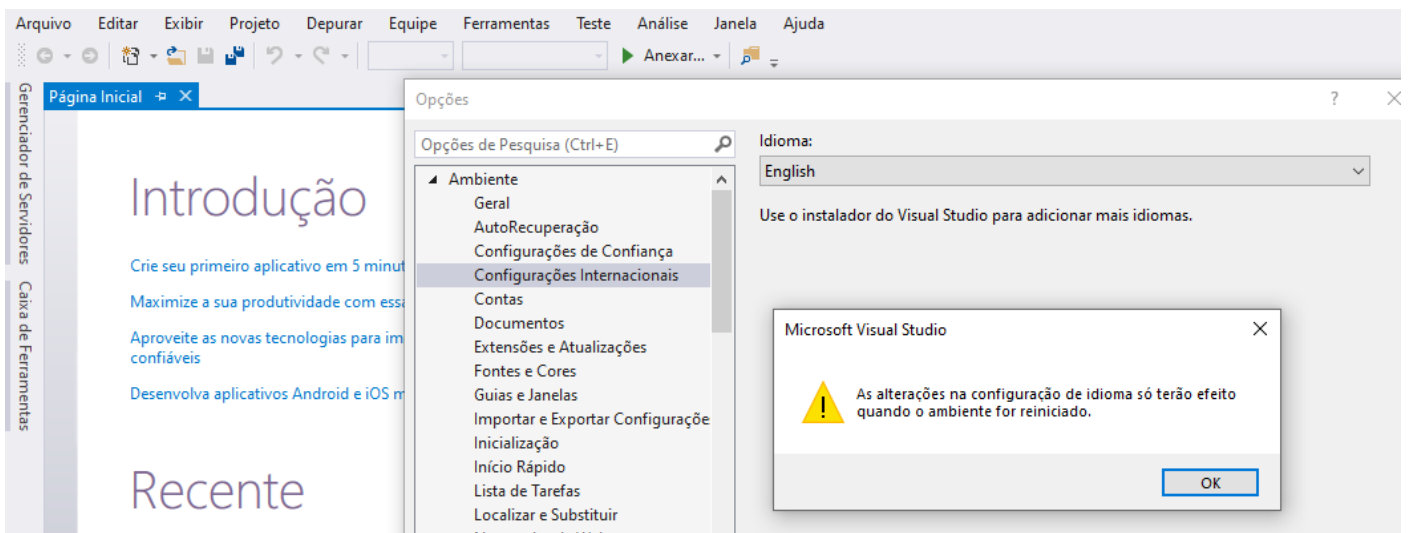
Espaço total necessário 348 MB

Instalar durante o download

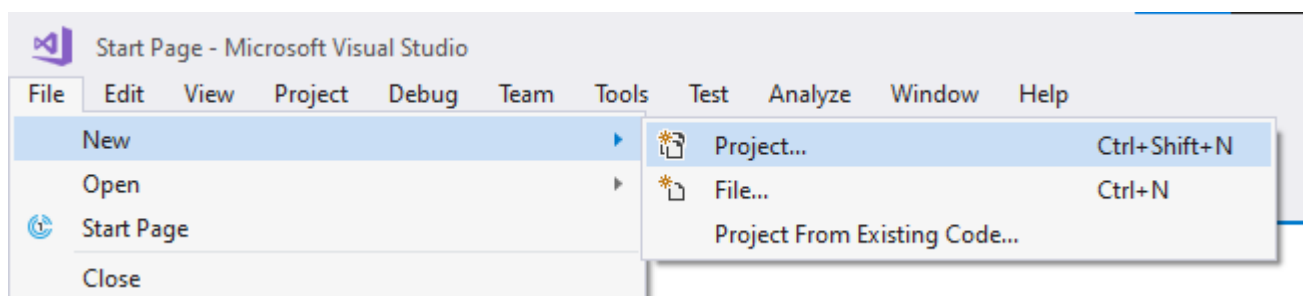
Modificar

Aguarde fazer o download.

Abra novamente o Visual Studio, vá em Ferramentas > Opções > Configurações Internacionais > Idioma > English.



Reinicie o Visual Studio e verifique a mudança.



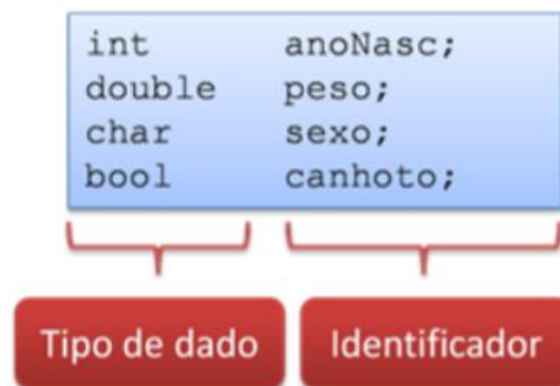
4 – Variáveis e Constantes

- 4.1 – Declaração de variáveis
- 4.2 - Inicialização
- 4.3 - Mudança de valor
- 4.4 – Definição de constante
- 4.5 – Tipos de Dados
- 4.6 – Value Types
- 4.7 – Conversão de tipos de dados
- 4.8 – Operadores

4.1 – Declaração de variáveis

Variáveis são área de memória onde você pode guardar dados.

Em C# as variáveis devem possuir um tipo e um identificador (nome).



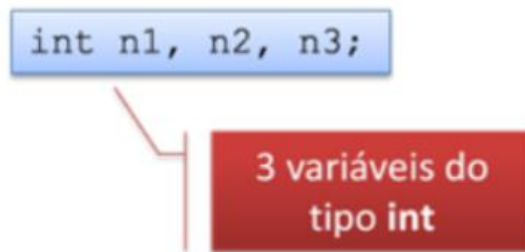
Sempre definimos um tipo de dado e ao lado um identificador (nome).

Aqui temos:

- o **int** que é um número inteiro,
- o **double** um número decimal,
- o **char** apenas uma letra,
- e o **bool** que é uma variável booleana true ou false.

Em C# todas as variáveis são tipadas, isto é, que tipo de dado a variável pode armazenar.

Podemos declarar múltiplas variáveis, basta separar as variáveis por vírgula. Nesse caso estou criando 3 variáveis do tipo **int** (variáveis do mesmo tipo).



Os identificadores de variáveis devem seguir algumas regras:

- O primeiro caractere deve ser uma letra ou '_' (underscore ou underline ou o sublinhado).
- A partir do segundo caractere, podem ser usados:
 - Letras
 - Números
 - ' '
 - '_'
- Se o identificador for igual a uma palavra-chave do C#, esse identificador deve iniciar com '@'.

Vejamos alguns exemplos de nomes de variáveis.

abc;	OK	1abc;	ERRO!
_abc;	OK	\$abc	ERRO!
ab1c1;	OK	abc#	ERRO!
_a_b_c	OK	a-b	ERRO!

Palavras reservadas devem começar com o '@'.

class	ERRO!	int	ERRO!
@class	OK	@int	OK

4.2 - Inicialização

O primeiro passo na criação de variáveis é a declaração de variáveis, o tipo de dado (valor) que aquela variável pode armazenar.

O segundo passo é inicializar a variável, ou seja, atribuir um valor à ela, armazenar um valor nessa variável.

A atribuição é feita pelo operador '='

Para compreender melhor essa atribuição, substitua o igual (=) pela palavra receba.

Exemplos

```
anoNasc = 1980; (anoNasc recebe 1980)
peso = 65.7; (peso recebe 65,7)
sexo = 'M'; (sexo recebe 'M')
```

Declaração e inicialização simultânea.

Double altura = 1.8;

Declaração e inicialização múltipla.

Int n1 = 10, n2 = 20, n3 = 30.

Todas as variáveis devem ser inicializadas antes de serem utilizadas.

- Se isso não for feito gera erro de compilação.

4.3 - Mudança de valor

Depois que você cria uma variável e inicializa ela com algum valor, você pode alterar o valor dessa variável.

```
int contador = 20;
int novoContador = contador + 1;
```

novoContador = 21

```
int x = 15;
x = x + 1;
```

x = 16

```
int y = x + x - 10;
```

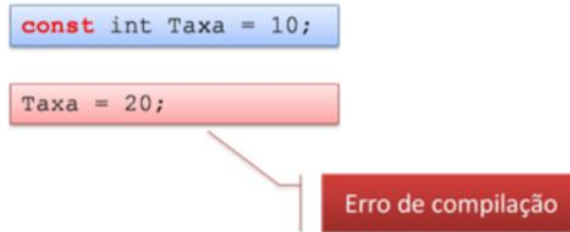
y = 22

4.4 – Definição de constante

Uma constante é uma variável que não pode ter seu valor alterado após a inicialização. O compilador garante isso. Se você tentar mudar o valor dará erro.

- Auxiliam o entendimento do código.
- Evitam erros na programação.

O modificador **const** é utilizado.



4.5 – Tipos de Dados e Operadores

O C# possui uma série de tipos de dados.

- Todos eles têm um mapeamento no CTS (Common Type System) da plataforma .NET.

Os tipos de dados estão divididos em:

- Value Types
- Reference Types

Esta divisão indica como os dados são armazenados na memória.

4.6 - Value Types

Números inteiros

C#	CTS	Descrição
sbyte	System.SByte	8 bits, com sinal
byte	System.Byte	8 bits, sem sinal
short	System.Int16	16 bits, com sinal
ushort	System.UInt16	16 bits, sem sinal
int	System.Int32	32 bits, com sinal
uint	System.UInt32	32 bits, sem sinal
long	System.Int64	64 bits, com sinal
Ulong	System.UInt64	64 bits, sem sinal

Os tipos inteiros **byte**, **short**, **int** e **long** aderem ao CLS (Common Language System) e tem a interoperabilidade na plataforma .NET com outras linguagens.

Número decimais.

	C#	CTS	Descrição
Aderem ao CLS	float	System.Single	32 bits, ponto flutuante
	double	System.Double	64 bits, ponto flutuante
	decimal	System.Decimal	128 bits, alta precisão

Uso do tipo **decimal** pode ocasionar queda de performance

Booleano

	C#	CTS	Descrição
Adere ao CLS	bool	System.Boolean	<i>verdadeiro ou falso</i>

Caractere

	C#	CTS	Descrição
Adere ao CLS	char	System.Char	16 bits, 1 caractere

Exemplo do uso de Value Types

```
int x = 90;
```

90 é um número inteiro, portanto pode ser atribuído a uma variável do tipo **int**

```
System.Int32 x = 90;
```

Idêntico ao exemplo anterior, mas utilizando o tipo CTS **System.Int32**

```
bool x = true;
```

true é um valor booleano, portanto pode ser atribuído a uma variável do tipo **bool**

```
double x = 2.5;
```

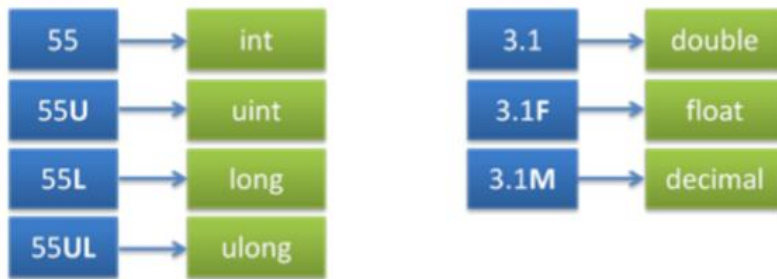
2.5 é um valor do tipo **double**, portanto pode ser atribuído a uma variável **double**

```
char x = 'A'
```

'A' é um caractere, portanto pode ser atribuído a uma variável do tipo **char**

```
short x = 90;
```

90 é um número inteiro, portanto pode ser atribuído a uma variável do tipo **short**

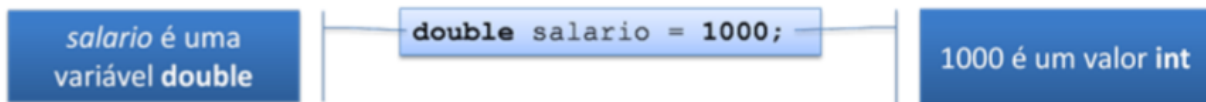


É possível utilizar os qualificadores com letras maiúsculas ou minúsculas

Apesar do usual ser letra maiúscula, o compilador irá compilar tanto com letra minúscula como maiúscula

4.7 - Conversão de tipos de dados

Algumas vezes precisamos atribuir um valor de um tipo a uma variável de outro tipo.



A conversão pode ser feita

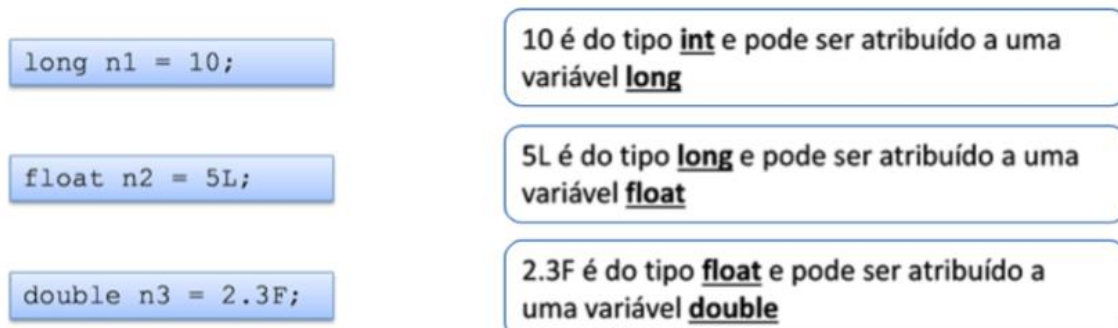
- De um tipo “menor” para um tipo maior
Ex.: **int** para **double**
- De um tipo “maior” para um tipo menor
Ex.: **long** para **int**

Esse processo é chamado de **casting**.

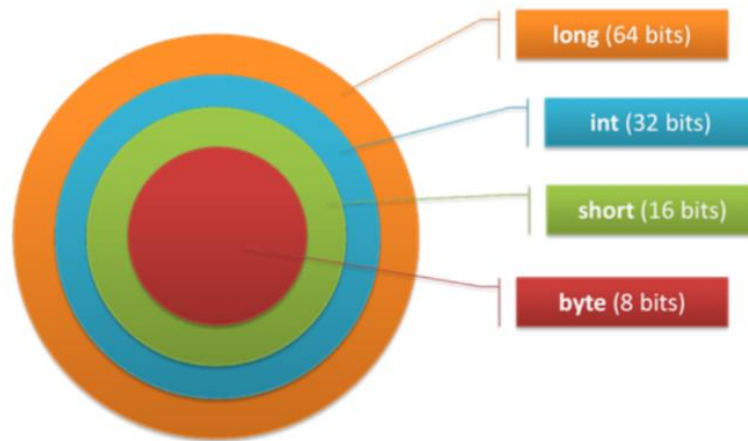
Casting Implícito

O compilador cuida de todo o processo de conversão.

Ocorre quando é preciso converte um tipo “menor” para um tipo “maior”
Não ocorre perda de informação.



Alguns exemplos, lembrando que existem outros tipos de casting em C#.



Casting Explícito

O programador é responsável pela conversão de dados.

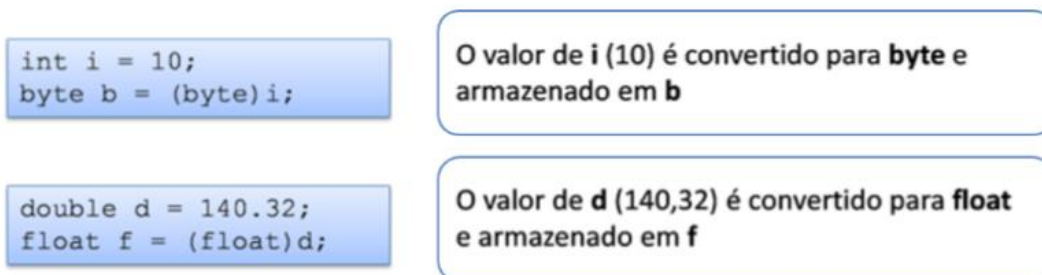
Necessário quando é preciso converter de um tipo “maior” para um tipo “menor”.

Pode ocorrer perda de informação.

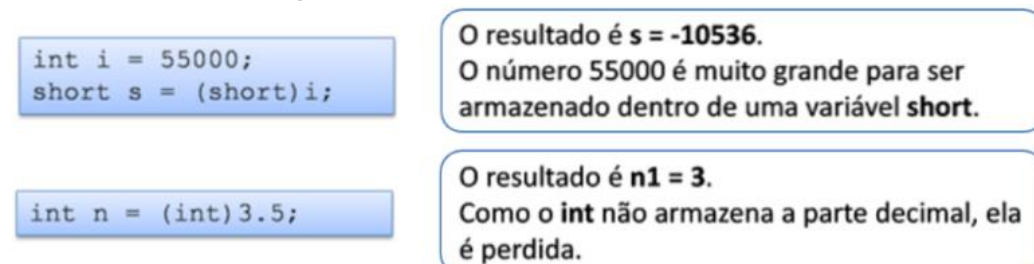


Se tentarmos compilar o código da esquerda ocorrerá erro.

Para fazer o casting explícito é preciso fazer como está o código da direita, colocando o tipo de dado que será armazenado entre parênteses ao lado da variável.



Cuidado com o casting explícito



4.8 – Operadores

Operadores Aritméticos

Operador	Descrição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo

O % (por cento) é o resto da divisão.

Operadores de Comparação

Operador	Descrição
==	Igual
!=	Diferente
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

O resultado dessa operação sempre é u valor booleano, ou seja, true or false, verdadeiro ou falso.

Operadores Lógicos

Operador	Descrição
!	Negação
	OU
&&	E

Outros Operadores Importantes

Operador	Descrição	Exemplo
++	Incremento	x++
--	Decremento	x--
+=	Soma com valor e atribui o resultado à própria variável	x += 2
-=	Subtrai do valor e atribui o resultado à própria variável	x -= 5
*=	Multiplica pelo o valor e atribui o resultado à própria variável	x *= 3
/=	Divide pelo valor e atribui o resultado à própria variável	x /= 4

Prioridades Entre Operadores



Tipo de Operador	Operadores
Parênteses	()
Unário	! casting
Multiplicação / Divisão	* / %
Soma / Subtração	+ -
Relacional	< > <= >= is as
Comparação	== !=
AND	&&
OR	
Atribuição	= += -= *= /=

Para expressões complexas, use parênteses para especificar prioridade

6 – Condicionais – Estruturas de Controle

Sintaxe básica

```
if (<condição_booleana>)  
{  
    <código_se_condição_verdadeira>;  
}
```

Opcionalmente, pode existir uma cláusula **else**

```
if (<condição_booleana>)  
{  
    <código_se_condição_verdadeira>;  
}  
else  
{  
    <código_se_condição_falsa>;  
}
```

A condição booleana pode ser qualquer expressão lógica.

- O resultado deve ser **true** ou **false**.

```
int x = 50;

if (x > 30)
{
    Console.WriteLine("Número maior que 30");
}
else
{
    Console.WriteLine("Número menor que 30");
}
```

Para blocos compostos por uma linha, não é necessário usar { e }, abre e fecha chaves.

```
int x = 20, y;

if (x > 10)
    y = x * 2;
else
    y = x * 3;
```

O recomendado ainda é utilizar { e }, abre e fecha chaves.

Outra possibilidade é utilizar o **operador ternário** em substituição ao if-else

<pre>int x = 50; int y; if (x > 30) { y = 1; } else { y = -1; }</pre>	<pre>int x = 50; int y = x > 30 ? 1 : -1;</pre> <div><div>Resultado, se verdadeiro</div><div>Resultado, se falso</div></div>
-----------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------