Programador de Sistemas

Linguagem C#

Introdução e Definições de Tipos de Dados e Variáveis

Prof. Mauricio Wieler Orellana
mauricioow@gmail.com

Introdução

Em Junho de 2000 a Microsoft anunciou a Plataforma .NET e uma nova linguagem de programação chamada C# (se lê “C Sharp”).

C# é uma linguagem fortemente tipada e orientada a objetos, projetada para oferecer a melhor combinação de simplicidade, expressividade e performance.

A linguagem C# aproveita conceitos de muitas outras linguagens, mas especialmente de C++ e Java. Ela foi criada por Anders Hejlsberg (que já era famoso por ter criado o TurboPascal e o Delphi para a Borland) e Scott Wiltamuth.

Introdução

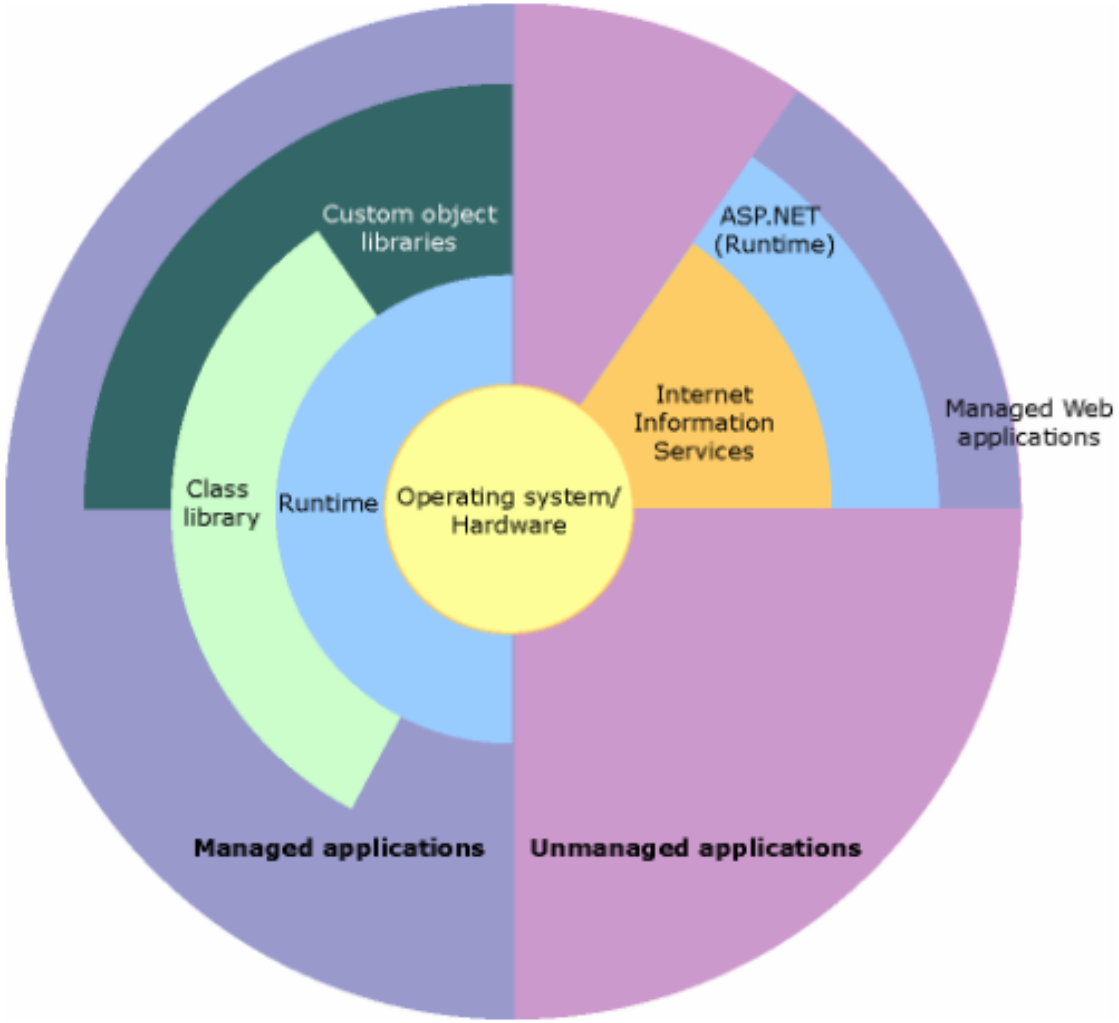


Ilustração 1 - Contexto do .NET Framework

A Plataforma .NET é centrada ao redor de uma Common Language Runtime (CLR, conceito similar ao da Java Virtual Machine, JVM) e um conjunto de bibliotecas que pode ser empregado em uma grande variedade de linguagens, as quais podem trabalhar juntas, já que todas são compiladas para uma mesma linguagem intermediária, a Microsoft Internediate Language (MSIL). Assim, é possível desenvolver aplicativos mesclando C# e Visual Basic ou qualquer outra

Introdução

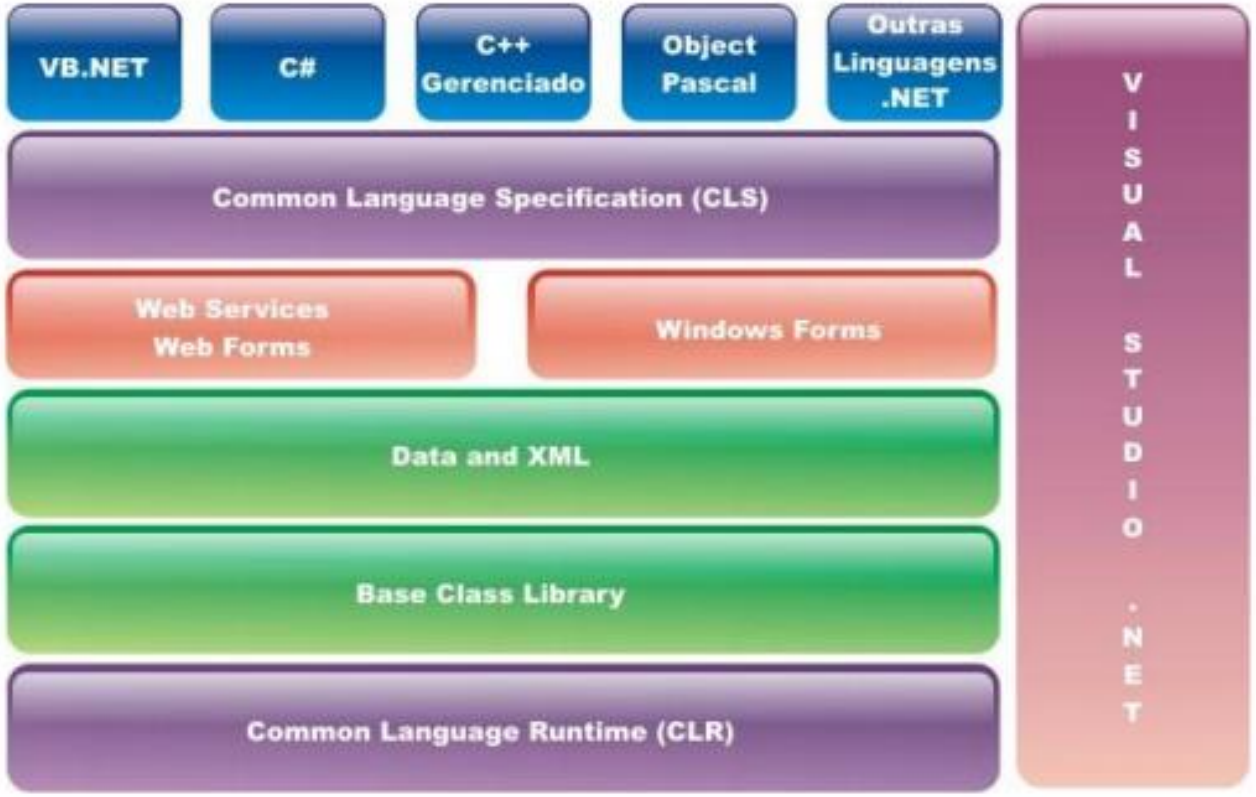
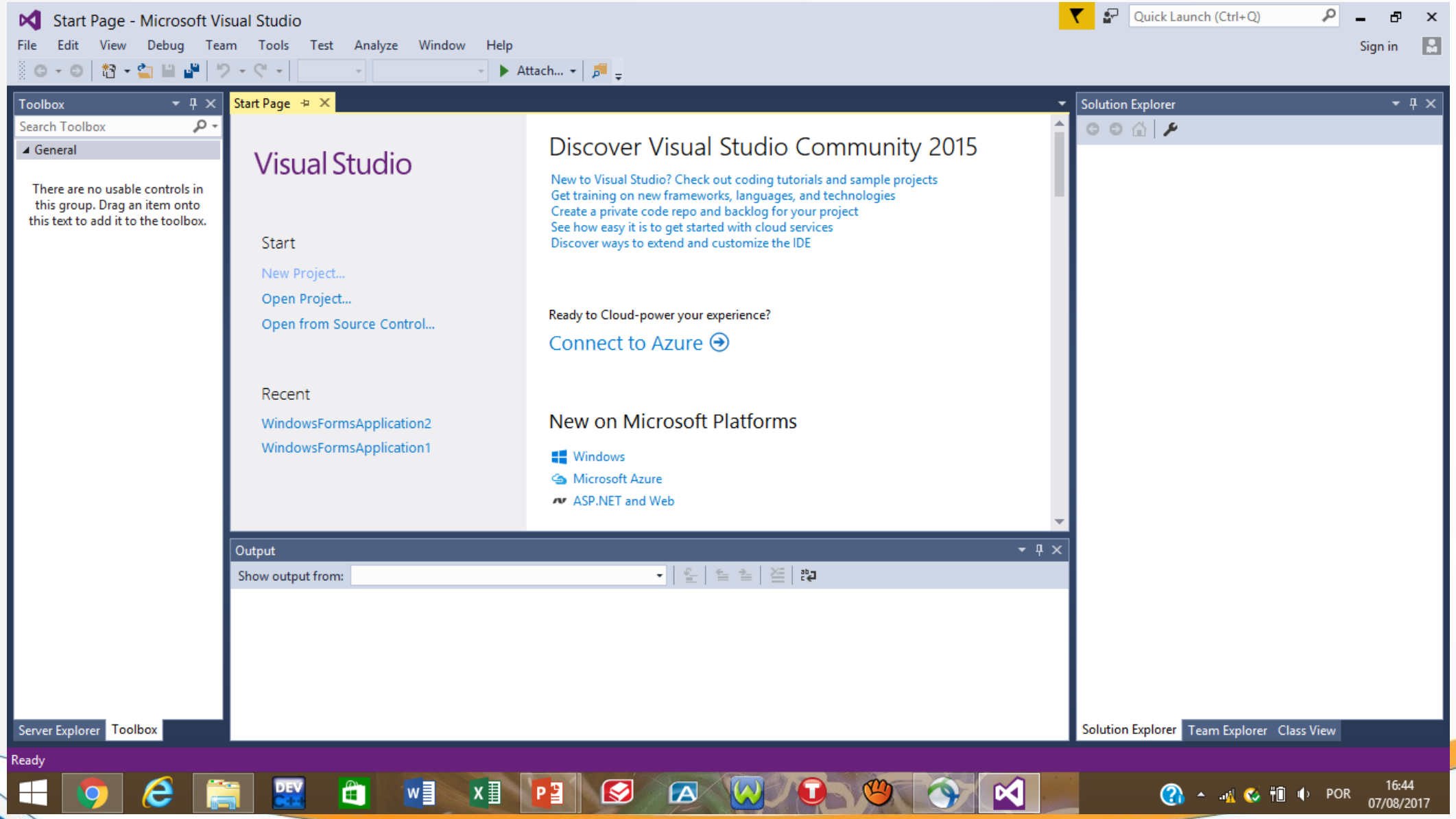


Ilustração 2 - Arquitetura do .NET

A sintaxe utilizada pelo C# é relativamente fácil, o que diminui o tempo de aprendizado. Todos os programas desenvolvidos devem ser compilados, gerando um arquivo com a extensão DLL ou EXE. Isso torna a execução dos programas mais rápida se comparados com as linguagens de script (VBScript , JavaScript) que atualmente utilizamos na internet.

O Ambiente de Desenvolvimento



Entrada e Saída (E/S) Básica

A entrada e saída de um programa C# é realizada pela biblioteca de classes do .NET Framework. Entre as classes presentes nesta biblioteca podemos citar a classe *System.Console*, que pode ser usada para escrever na tela e obter dados do teclado.

Imprimir na Tela em C# é uma atividade bastante simples. Para imprimir basta chamar o método System.Console.WriteLine(). Veja o exemplo:

```
System.Console.WriteLine("Olá, Mundo!");
```

Esta linha imprime a mensagem mostrada abaixo:

```
Olá, Mundo!
```

Entrada e Saída (E/S) Básica

Por vezes você quer apenas que o usuário pressione uma tecla qualquer antes que o programa proceda com uma dada tarefa. Isso é simples de fazer, conforme vemos abaixo:

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

O método acima retorna informações sobre a tecla pressionada, mas por enquanto vamos ignorar isto, pois por ora não nos interessa saber qual foi. Veremos mais sobre isso mais tarde.

Em outras situações iremos querer que o usuário nos dê alguma informação. A maneira mais fácil de se fazer isso é ler uma linha a partir do teclado, como mostra o exemplo:

```
string nome = System.Console.ReadLine();
```

O que quer que o usuário digitasse seria armazenado na variável chamada nome (veja mais sobre variáveis na próxima seção).

Por vezes será necessário converter o texto que o usuário digitou para um outro tipo de dados, então é aí que entram as conversões de tipos de dados. Veja mais sobre tipos de dados e conversões na próxima seção.

Variáveis e Tipos de Dados

As variáveis são utilizadas para armazenar informações na memória do computador enquanto o programa C# esta sendo executado. As informações contidas nas variáveis podem ser alteradas durante a execução do programa.

As variáveis devem possuir um nome para que possamos nos referenciar a elas mais tarde. Ao nomear uma variável devemos observar as seguintes restrições:

- O nome deve começar com uma letra ou _ .
- Não são permitidos espaços, pontos ou outros caracteres de pontuação, mas podemos usar números.
- O nome não pode ser uma palavra reservada do C#.
- O nome deve ser único dentro do contexto atual.

Além do nome, devemos também definir o tipo de dados e o escopo (local onde a variável estará acessível). O escopo é definido pelos modificadores de acesso (veremos mais sobre isso mais tarde)

Para declarar uma variável, primeiro você precisa indicar o seu tipo e depois o seu nome. Veja os exemplos:

```
string nome;
int telefone;
```

Tipos de Dados

- Como todas as linguagens de programação, o C# apresenta seu grupo de tipos de dados básico.
- Esses tipos são conhecidos como tipos primitivos ou fundamentais, por serem suportados diretamente pelo compilador e serão utilizados durante a codificação na definição de variáveis, parâmetros, declarações e até mesmo em comparações.
- Em C# todo o tipo de dados possui um correspondente na CLR (Common Language Runtime), por exemplo: int em C# refere-se a System.Int32 na plataforma .NET.

Tipos de Dados

Tipo C#	Tipo .NET	Descrição	Faixa de dados
bool	System.Boolean	Booleano	true ou false
byte	System.Byte	Inteiro de 8-bit com sinal	-127 a 128
char	System.Char	Caracter Unicode de 16-bit	U+0000 a U+ffff
decimal	System.Decimal	Inteiro de 96-bit com sinal com 28-29 dígitos	1,0 × 10-28 a 7,9 × 1028
double	System.Double	Flutuante IEEE 64-bit com 15-16 dígitos significativos	±5,0 × 10-324 a ±1,7 × 10308
float	System.Single	Flutuante IEEE 32-bit com 7 dígitos significativos	±1,5 × 10-45 a ±3,4 × 1038
int	System.Int32	Inteiro de 32-bit com sinal	-2.147.483.648 a 2.147.483.647
long	System.Int64	Inteiro de 64-bit com sinal	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
Object	System.Object	Classe base	
Sbyte	System.Sbyte	Inteiro de 8-bit sem sinal	0 a 255
Short	System.Int16	Inteiro de 16-bit com sinal	-32,768 a 32,767
String	System.String	String de caracteres Unicode	
Uint	System.UInt32	Inteiro de 32-bit sem sinal	0 a 4,294,967,295
Ulong	System.UInt64	Inteiro de 64-bit sem sinal	0 a 18,446,744,073,709,551,615
Ushort	System.UInt16	Inteiro de 16-bit sem sinal	0 a 65,535

Conversões de Tipos

- Uma tarefa muito comum no mundo da programação é a utilização de conversões. Por exemplo: podemos converter um inteiro para um longo. Quando o usuário digita um número qualquer numa TextBox, é necessária uma conversão de valores antes que um cálculo possa ser realizado.

Método	Descrição
System,Convert.ToBoolean()	Converte uma string para um valor booleano
System,Convert.ToByte()	Converte para o tipo byte
System,Convert.ToChar()	Converte para o tipo char
System,Convert.ToDateTime()	Converte para o tipo data e hora
System,Convert.ToDecimal()	Converte para o tipo decimal
System,Convert.ToDouble()	Converte para o tipo doublé
System,Convert.ToInt16()	Converte para o tipo short
System,Convert.ToInt32()	Converte para o tipo inteiro
System,Convert.ToInt64()	Converte para o tipo long
System,Convert.ToSingle()	Converte para o tipo single
System,Convert.ToString()	Converte para o tipo string

Vetores (Arrays)

- Array é uma coleção de elementos armazenados em sequência, acessíveis através de um índice numérico. No C#, o primeiro elemento de um array é o de índice zero (0).
- É importante notar que podemos criar arrays com uma ou mais dimensões.
- Para definir um array você seguirá esta estrutura:

```
<tipo-de-dados> [] <nome-do-array> = new <tipo-de-dados>[<tamanho>];
```

Você pode criar um array e não inicializá-lo:

```
string[] arr;
```

Vetores (Arrays)

- No entanto, para utilizá-lo em outras partes do código, precisamos inicializá-lo. Por exemplo, para inicializarmos o array anterior com 10 elementos:

```
arr = new string[10];
```

- Para armazenar informações num array use o índice para indicar o elemento desejado:

```
arr[0] = "Alfredo";
arr[1] = "Maria";
arr[2] = "Paulo";
arr[8] = "Beatriz";
```

Vetores (Arrays)

- Entretanto, podemos inicializá-lo, junto com a declaração:

```
arr = new string[4] {“Alfredo”, “Maria”, “Paulo”, Beatriz”};
```

Podemos também omitir o número de elementos:

```
int [] numeros = {1, 2, 3, 4, 5};
string [] nomes = {“Alfredo”, “Maria”, “Paulo”, “Beatriz”};
```

Para acessar um elemento de um array, você deve usar o índice do elemento desejado, lembrando que os índices começam sempre e o zero. Por exemplo, para acessar o segundo elemento do array nomes você usaria a seguinte instrução:

```
string saida = arr [1];   // Isto armazenaria “Maria” na
                                    // variável saida
```

Arrays Multidimensionais

- Num array multidimensional separamos as dimensões por vírgulas.

```
int [ , ] arr;
```

```
int [,] numeros = new int [3, 2] {{1, 2}, {3, 4}, {5, 6} };
string [,] nomes = new string [2, 2] {{"Mara", "Lotar"}, {"Mary", "José"} };
```

Essas linhas poderiam ser representadas por:

```
numeros [0, 0] = 1;
numeros [0, 1] = 2;
numeros [1, 0] = 3;
numeros [1, 1] = 4;
numeros [2, 0] = 5;
numeros [2, 1] = 6;
```

Arrays Multidimensionais

- E por:

```
nomes [0, 0] = "Mara";
nomes [0, 1] = "Mary";
nomes [1, 0] = "Lotar";
nomes [1, 1] = "José";
```

Expressões

- Um programa não será muito útil se não pudermos usá-lo para efetuar cálculos ou outro tipo de manipulações de dados. É para isso que surgem os operadores. Conforme veremos a seguir, existem diferentes tipos de operadores, dependendo do tipo de operação que desejamos realizar.

Categoria	Operadores
Aritmética	+ - * / %
Lógica (boolena e bitwise)	& | ^ ! ~ && || true false
Concatenação de string	+
Incremento e decremento	++ --
Shift	<< >>
Relacional	== != < > <= >=
Atribuição	= += -= *= /= %= &= |= ^= <<= >>=
Acesso a membro	.
Indexação	[]
Cast	()
Condicional	?:
Delegate (concatenação e remoção)	+ -
Criação de objeto	new
Informação de tipo	is sizeof typeof
Controle de excessão de overflow	checked unchecked
Indireção e endereço	* -> [] &

Estruturas de Controle

- A instrução **if** pode ser usada para seletivamente executar trechos de código. Você pode executar um código apenas se uma dada condição for verdadeira ou falsa. Existe ainda a possibilidade de se usar a cláusula opcional else, que inverte a condição testada. Seu corpo será executado se a condição não for verdadeira.

```
if (<condição>)
{
    // Código para quando a condição for verdadeira.
}
else
{
    // Código para quando a condição for falsa.
}
```

Estruturas de Controle

- Se a condição for satisfeita, todas as instruções do primeiro bloco de comando serão executadas; no entanto, se a condição não for satisfeita, serão executadas as instruções do segundo bloco.

```
int numero = 3;
if (numero % 2 == 0)
{
    System.Console.WriteLine("O número é par.");
}
else
{
    System.Console.WriteLine("O número é ímpar.");
}
```

Estruturas de Repetição - For

- Loops contadores executam uma tarefa um determinado número de vezes. A instrução for pode ser caracterizada como sendo loop contador, pois conhecemos os extremos que devem ser percorridos

```
for (<inicialização>; <teste>; <incremento>)
{
    // Código a ser executado enquanto o teste for
    // verdadeiro.
}
```

Estruturas de Repetição - While

- A instrução while executa os comandos dentro do loop enquanto a condição que lhe serve como parâmetro for verdadeira.

```
while (<condição>)
{
    // Código a ser executado enquanto a condição for
    // verdadeira.
}
```