

Programação C# + ASP.NET

Prof. Me. Daniel Menin Tortelli

e-mail: danielmenintortelli@gmail.com

Site: <http://sites.google.com/site/danielmenintortelli/home>

Escopo das Declarações de Variáveis

Escopo das Declarações de Variáveis

- O escopo de uma declaração é a parte do programa que pode referenciar a entidade (variável/objeto) declarada pelo seu nome.
- Diz-se que a entidade está **no escopo** para essa parte do programa.
- Um erro de compilação ocorre quando uma variável local (com o mesmo nome) é declarada mais de uma vez em um método.

Escopo das Declarações de Variáveis

- As regras básicas de escopo são:
 - O escopo de uma declaração de parâmetro é o corpo do método em que a declaração aparece.
 - O escopo de uma declaração de uma variável global vai do ponto que a declaração aparece até o final desse bloco.
 - O escopo de uma declaração de uma variável local que aparece na seção de inicialização do cabeçalho de uma instrução FOR é o corpo de uma instrução FOR com as outras expressões do cabeçalho.
 - O escopo de um método ou campo de uma classe é o corpo inteiro da classe.

Gerando Números Aleatórios

Gerando Números Aleatórios

- A geração de números aleatórios pode ser introduzido através da classe ***Random*** (inclusa no .NET Framework).
- Os objetos da classe ***Random*** podem produzir valores aleatórios do tipo ***int***.
- O intervalo de valores pode ser negativo.
- O intervalo de valores é calculado como:

(valMin, valMax – 1)

- Obs.: ***O valMax deve ser maior ou igual ao valMin.***

```
// Função Principal
static void Main(string[] args)
{
    // Cria objeto da classe Random
    Random random = new Random();

    // Declara e inicializa variável que irá armazenar
    // cada número aleatório gerado em tempo de execução
    int valor = 0;

    // Laço FOR que repete 30 vezes
    for (int i = 0; i < 30; i++)
    {
        // gera um número aleatório entre 0 e 100
        valor = random.Next(-10, 10);

        // Imprime cada valor gerado
        Console.Write("{0} ", valor);
    }

    Console.WriteLine();    // linha em branco

} // fim função Main
```

Gerando Números Aleatórios

Gerando Números Aleatórios – Exercício 1

- 1 – Utilizando números aleatórios, crie um programa em C# que simule o lançamento de um dado (de 6 faces), durante um quantidade **x** de vezes que será definida pelo usuário no início do programa.
 - Exiba o valor do dado a cada lançamento.
 - Faça um somatório dos valores de cada face a cada lançamento e exiba esse valor no final da execução do programa.

Gerando Números Aleatórios – Exercício 2

- Gerar 20 números aleatórios entre (-5, 6).
- Mostrar os valores gerados.
- Contar e mostrar os valores pares.
- Contar e mostrar os valores ímpares.
- Contar e mostrar os valores maiores que zero.
- Cuidar o caso do zero (não é par, nem ímpar).
- Obs.: Ao mostrar cada valor par ou ímpar, mostrar uma mensagem ao lado dizendo se o valor é positivo ou negativo.

Gerando Números Aleatórios – Exercício 3

- Faça um algoritmo que simule a geração de 6 números aleatórios que serão a aposta para a Mega Sena.
- O intervalo dos valores é de 1 a 60. Caso o valor gerado já tenha saído, faça com que o programa gere outro número até que sejam gerados 6 números distintos.
- No final, exiba os valores da aposta.

Arrays

Arrays

- Um **array** é um grupo de variáveis que contém valores que são todos do mesmo tipo.
- Os tipos podem ser divididos em duas categorias:
 - Tipos primitivos (int, float, string, double...);
 - Tipos por referência (objeto);

Declarando e criando Arrays

- Os arrays ocupam espaço na memória, assim como as variáveis.

```
// Declarando e inicializando um array com 5 elementos INT  
int[] array = new int[5] { 3, 5, 7, 10, 11 };
```

```
// Declarando um array de 5 elementos do tipo INT  
int[] array2 = new int[5];
```

```
// Inicializando o array com os elementos  
array2[0] = 3;  
array2[1] = 5;  
array2[2] = 7;  
array2[3] = 10;  
array2[4] = 11;
```

Declarando e criando Arrays

- Declarando 12 variáveis para usar no programa... mas, e se fossem 1000?!?!?!?

```
int c0 = -45;  
int c1 = 6;  
int c2 = 0;  
int c3 = 72;  
int c4 = 1543;  
int c5 = -89;  
int c6 = 0;  
int c7 = 62;  
int c8 = -3;  
int c9 = 1;  
int c10 = 6453;  
int c11 = 78;
```

- Melhor usar um **array**... ☺

```
int[] c = new int[12];
```

Nome do array (c) →

Índice (ou subscripto)
do elemento no array c

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

```
static void Main(string[] args)
{
    // Declarando um array e atribuindo 5 elementos
    // do tipo inteiro
    int[] array = new int[5] { 1, 5, 50, -400, 20 };

    // Declarando um array
    int[] array2 = new int[5];

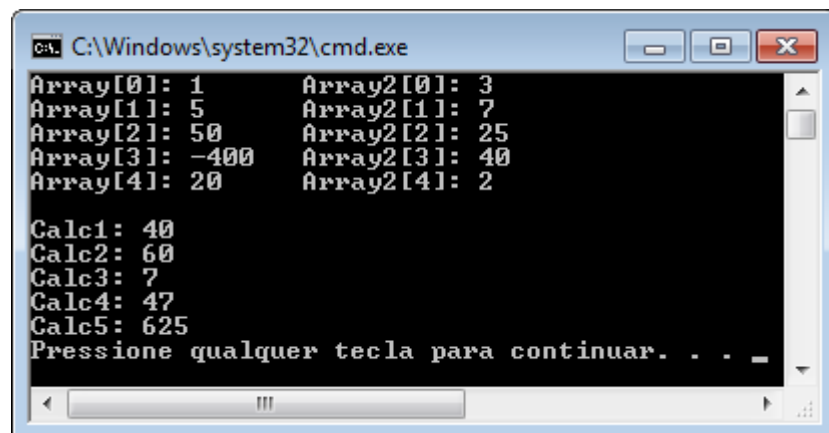
    // Atribuindo 5 elementos do tipo inteiro
    array2[0] = 3;
    array2[1] = 7;
    array2[2] = 25;
    array2[3] = 40;
    array2[4] = 2;

    // Fazendo alguns cálculos usando elementos de arrays
    int calc1 = 2 * array[4];
    int calc2 = array[2] + 10;
    int calc3 = array[1] + array2[4];
    int calc4 = array2[2] / array2[4] + 35;
    int calc5 = array2[2] * (array[4] + 5);
}
```

```
// Imprimindo Arrays
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("Array[{0}]: {1} \t Array2[{2}]: {3}", i, array[i], i, array2[i]);
}

Console.WriteLine(); // Imprime linha em branco

// Imprimindo resultados
Console.WriteLine("Calc1: {0}", calc1);
Console.WriteLine("Calc2: {0}", calc2);
Console.WriteLine("Calc3: {0}", calc3);
Console.WriteLine("Calc4: {0}", calc4);
Console.WriteLine("Calc5: {0}", calc5);
}
```



CA: C:\Windows\system32\cmd.exe

```
Array[0]: 1      Array2[0]: 3
Array[1]: 5      Array2[1]: 7
Array[2]: 50     Array2[2]: 25
Array[3]: -400   Array2[3]: 40
Array[4]: 20     Array2[4]: 2

Calc1: 40
Calc2: 60
Calc3: 7
Calc4: 47
Calc5: 625
Pressione qualquer tecla para continuar. . .
```


Declarando e criando Arrays

- Quando um array é criado, cada elemento do array recebe um **valor padrão**:
 - **Zero** para os elementos numéricos de tipo primitivo (int, float, double...);
 - **False** para elementos do tipo *boolean*;
 - **Null** para referências de objetos de classes (qualquer tipo não-primitivo).
- Um programa pode declarar arrays de qualquer tipo.
 - Por exemplo, todo elemento de um array **int** é um valor **int** e todo elemento de um array **String** é uma referência a um objeto **String**.

Declarando e criando Arrays

```
static void Main(string[] args)
{
    // Declarando e inicializando um array com 5 elementos INT
    int[] array = new int[5];

    // Solicita a inserção de 5 valores do tipo int
    // Armazena o valor digitado na posição do array
    for (int i = 0; i < 5; i++)
    {
        Console.Write("Digite um número para armazenar em array[{0}]: ", i);
        array[i] = Convert.ToInt32(Console.ReadLine());
    }

    Console.WriteLine(); // Imprime linha em branco
}
```

Declarando e criando Arrays

```
// Imprime o conteúdo do array
for (int i = 0; i < 5; i++)
{
    Console.WriteLine("array[{0}]: {1}", i, array[i]);
}

int soma = 0;    // Declara variável soma e inicializa com zero
for (int i = 0; i < 5; i++)
{
    // Soma elementos do array
    soma = soma + array[i];
}

// Imprime soma dos elementos do array
Console.WriteLine("A soma dos elementos do array é: {0}", soma);
}
```

Exercícios 1

Crie um programa em C# que solicite ao usuário a digitação de 10 números inteiros.

- Os números digitados devem ser armazenados em um array de inteiros.
- Antes de armazenar o número no array, certifique-se que o número digitado seja maior do que zero. Caso contrário, solicite ao usuário que insira um número válido novamente.
- No final mostre a média de todos os elementos do array.

Exercícios 2

Utilize um array unidimensional de inteiros de 5 elementos para resolver o seguinte problema:

- Escreva um aplicativo que solicite a inserção de números que estejam apenas no intervalo de 10 a 50.
- Enquanto cada número é inserido, armazene-o no array apenas se ele NÃO for um número duplicado dentro do array. Caso for, exiba uma mensagem mostrando a posição do array que encontra-se o número duplicado e solicite ao usuário a inserção de um novo valor.
- Repita esse processo até que o array contenha 5 elementos diferentes armazenados.

Exercícios 3

Escreva um programa em C# para ler um array **X** de 10 elementos inteiros.

Logo após copie os elementos do array **X** para um array **Y** fazendo com que o 1º elemento de **X** seja copiado para o 10º de **Y**, o 2º de **X** para o 9º de **Y** e assim sucessivamente.

Após o término da cópia, imprimir o array **Y**.

Exercícios 4

- Escreva um programa em C# para ler um array **A** de 10 elementos inteiros e um valor **X**.
- A seguir imprimir os índices do array **A** em que aparece um valor igual a **X**.

Exercícios 5

- Escreva um programa em C# para ler um array **A** de 10 elementos inteiros e um valor **X**.
- A seguir imprimir "ACHEI" se o valor **X** existir em **A** e "NÃO ACHEI" caso contrário.

Exercícios 6

- Escreva um programa em C# para ler um array **A** de 10 elementos e um valor X.
- Copie para um array **S** (sem deixar elementos vazios entre os valores copiados) os elementos de **A** que são maiores que X.
- Logo após imprimir o array **S**.

Exercícios 7

- Escreva um programa em C# para ler um array de 10 elementos inteiros.
- Excluir o 1º elemento do array deslocando os elementos subsequentes de uma posição para o início.
- Imprimir o array após a retirada do primeiro elemento.

Exercícios 8

- Escreva um programa em C# para ler um array **X** de 10 elementos e um valor **P** (aceitar apenas valores entre 0 e 9) que representa a posição de um elemento dentro do array **X**.
- Imprimir o valor do elemento que ocupa a posição informada.
- Logo após excluir esse elemento do array fazendo com que os elementos subsequentes (se houverem) sejam deslocados de 1 posição para o início.
- Imprimir o array **X** após a exclusão ter sido executada.

Exercícios 9

- Escreva um programa em C# para ler um array **R** (de 5 elementos) e um array **S** (de 10 elementos).
- Gere um array **X** que possua os elementos comuns a **R** e a **S**. Considere que pode existir repetição de elementos no mesmo array.
- Nesta situação somente uma ocorrência do elemento comum aos dois deve ser copiada para o array **X**.
- Após o término da cópia, escrever o array **X**.

Passando arrays para funções

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

namespace Arrays_02
{
    class Program
    {
        // Função que localiza e retorna o menor elemento do array
        static int AcharMenorValor(int[] array)
        {
            // Declara variável iMenor e atribui a ela o
            // primeiro elemento do array
            int iMenor = array[0];

            // Pesquisa no array e calcula o menor elemento
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i] < iMenor)
                    iMenor = array[i];
            }

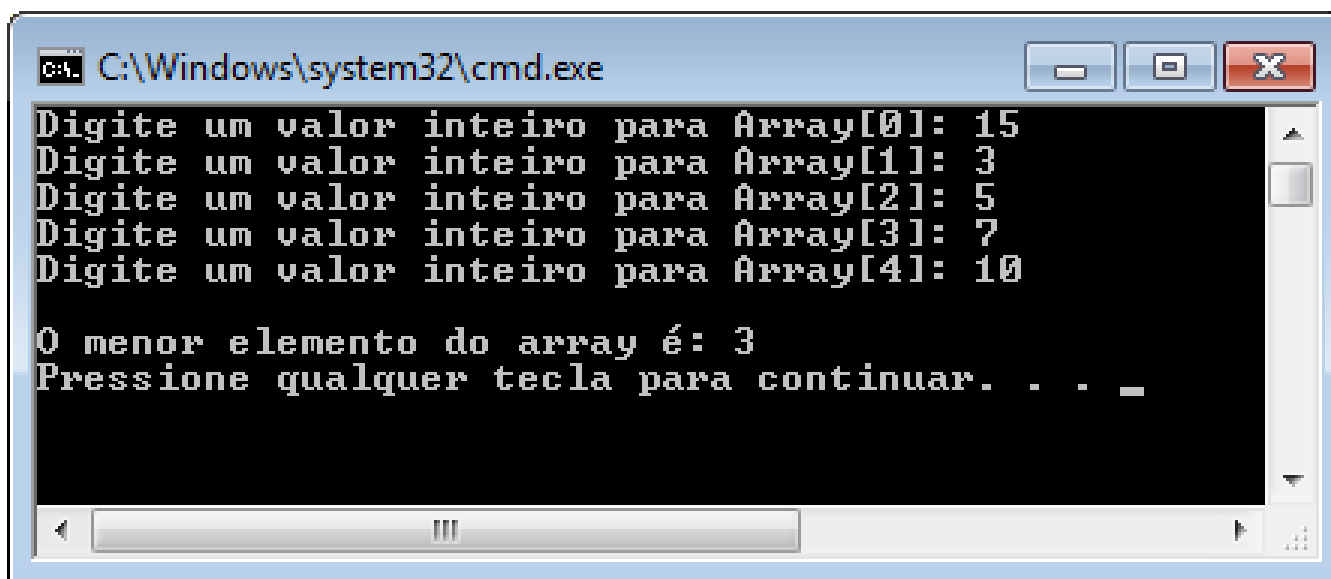
            return iMenor; // retorna o menor elemento encontrado no array
        }
    }
}
```

```
// Função Principal
static void Main(string[] args)
{
    // Declara um array de 5 elementos inteiros
    int[] array = new int[5];

    // Solicita a inserção de 5 elementos e
    // armazena cada elemento no array
    for (int i = 0; i < 5; i++)
    {
        Console.Write("Digite um valor inteiro para Array[{0}]: ", i);
        array[i] = Convert.ToInt32(Console.ReadLine());
    }

    // Chama função 'AcharMenorValor' e armazena o resultado
    // na variável iMenor
    int iMenor = AcharMenorValor(array);

    // Exibe o menor valor do array
    Console.WriteLine("\nO menor elemento do array é: {0} ", iMenor);
}
}
```



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

Digite um valor inteiro para Array[0]: 15
Digite um valor inteiro para Array[1]: 3
Digite um valor inteiro para Array[2]: 5
Digite um valor inteiro para Array[3]: 7
Digite um valor inteiro para Array[4]: 10

O menor elemento do array é: 3
Pressione qualquer tecla para continuar. . . _
```


Exercícios

Modifique o programa anterior:

- Crie a função “***AcharMaiorValorPar(int[] array)***”, cuja finalidade é retornar o maior elemento par encontrado dentro do array passado como parâmetro da função.
- Exiba uma mensagem mostrando o valor encontrado.

Alterando o Tamanho do Array

Alterando o Tamanho do Array

- Embora os arrays sejam declarados com um tamanho fixo, é possível modificar seu tamanho inicial utilizando o método `Resize`.
- O método cria um novo array com o novo tamanho especificado e copia todo o conteúdo do array antigo.
- Se o novo tamanho for menor do que o tamanho atual, parte do conteúdo será perdido.

```
int[] newArray = new int[ 5 ];  
Array.Resize( ref newArray, 10 );
```

Usando **foreach** para
percorrer e acessar os
elementos do array

Usando **foreach**

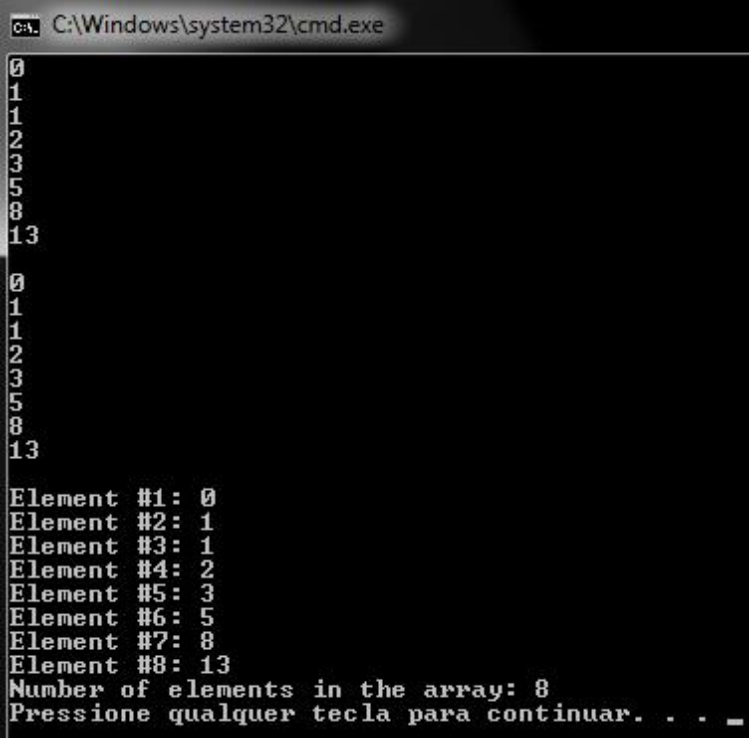
- A instrução **foreach** processa os elementos na ordem retornada pela matriz ou enumerador do tipo de coleção que normalmente vai do elemento 0 ao último.
- As instruções inseridas continuam a ser executadas para cada elemento no array ou na coleção.
- Após a iteração ter sido concluída para todos os elementos na coleção, o controle é transferido para a próxima declaração que segue o bloco de **foreach**.
- Em qualquer ponto dentro do bloco de **foreach**, você pode quebrar o loop usando a palavra-chave [break](#), ou pular para a próxima iteração no loop usando a palavra-chave [continue](#).

Usando foreach

```
static void Main(string[] args)
{
    int[] fibarray = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };
    foreach (int element in fibarray)
    {
        System.Console.WriteLine(element);
    }
    System.Console.WriteLine();

    // Compare the previous loop to a similar for loop.
    for (int i = 0; i < fibarray.Length; i++)
    {
        System.Console.WriteLine(fibarray[i]);
    }
    System.Console.WriteLine();

    // You can maintain a count of the elements in the collection.
    int count = 0;
    foreach (int element in fibarray)
    {
        count += 1;
        System.Console.WriteLine("Element #{0}: {1}", count, element);
    }
    System.Console.WriteLine("Number of elements in the array: {0}", count);
}
```



```
C:\Windows\system32\cmd.exe
0
1
1
2
3
5
8
13

0
1
1
2
3
5
8
13

Element #1: 0
Element #2: 1
Element #3: 1
Element #4: 2
Element #5: 3
Element #6: 5
Element #7: 8
Element #8: 13
Number of elements in the array: 8
Pressione qualquer tecla para continuar. . . _
```

Sobrecarga de Funções

```
// Imprime int array
static void ImprimeArray(int[] inputArray)
{
    foreach (int element in inputArray)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}

// Imprime double array
static void ImprimeArray(double[] inputArray)
{
    foreach (double element in inputArray)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}

// Imprime char array
private static void ImprimeArray(char[] inputArray)
{
    foreach (char element in inputArray)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}
```



```
static void Main(string[] args)
{
```

```
    // Cria arrays de int, double e char
```

```
    int[] intArray = { 1, 2, 3, 4, 5, 6 };
```

```
    double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
```

```
    char[] charArray = { 'H', 'E', 'L', 'L', 'O' };
```

```
    Console.WriteLine("Array intArray contém:");
```

```
    ImprimeArray(intArray); // Passando um array int para a função ImprimeArray
```

```
    Console.WriteLine("Array doubleArray contém:");
```

```
    ImprimeArray(doubleArray); // Passando um array double para a função ImprimeArray
```

```
    Console.WriteLine("Array charArray contém:");
```

```
    ImprimeArray(charArray); // Passando um array char para a função ImprimeArray
```

```
}
```

```
Array intArray contém:
```

```
1 2 3 4 5 6
```

```
Array doubleArray contém:
```

```
1,1 2,2 3,3 4,4 5,5 6,6 7,7
```

```
Array charArray contém:
```

```
H E L L O
```

```
Pressione qualquer tecla para continuar. . .
```


Introdução a Consultas **LINQ (C#)**

Consultas LINQ

- Uma **query** (consulta) é uma expressão que recupera dados de um fonte de dados.
- Em uma consulta LINQ, você está sempre trabalhando com objetos. Você usa a mesma codificação de padrões básicos para a query e transformar dados em documentos XML, bases de dados SQL, ADO.NET Datasets, coleções .NET, e qualquer outro formato para o qual um provedor LINQ esteja disponível.
- Todas as operações de consulta de LINQ consistem em três diferentes ações:
 1. Obtenha a fonte de dados.
 2. Criar a consulta.
 3. Executar a consulta.

Consultas LINQ

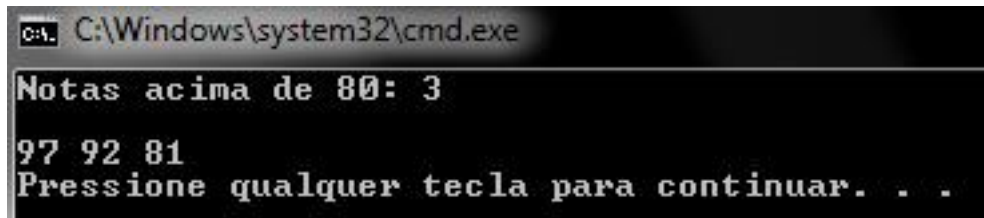
```
static void Main(string[] args)
{
    // Declara e inicializa um array
    int[] Notas = new int[] { 57, 68, 97, 92, 81, 60 };

    // Define a expressão de consulta
    IEnumerable<int> notasQuery =
        from nota in Notas
        where nota > 80
        select nota;

    // Exibe a quantidade de valores encontrados após a consulta
    Console.WriteLine("Notas acima de 80: {0}\n", notasQuery.Count());

    // Executa a consulta
    foreach (int i in notasQuery)
    {
        Console.Write(i + " ");
    }

    Console.WriteLine(); // Linha em branco
}
```



```
C:\Windows\system32\cmd.exe
Notas acima de 80: 3
97 92 81
Pressione qualquer tecla para continuar. . .
```

Consultas LINQ

```
static void Main(string[] args)
{
    // Declara e inicializa um array
    int[] Notas = new int[] { 57, 68, 97, 92, 81, 60 };

    // Define a expressão de consulta
    // Cria uma consulta que ordena as notas em ordem crescente
    IEnumerable<int> notasQuery =
        from nota in Notas
        orderby nota
        select nota;

    // Executa a consulta
    foreach (int i in notasQuery)
    {
        Console.Write(i + " ");
    }

    Console.WriteLine(); // linha em branco
}
```



C:\Windows\system32\cmd.exe
57 60 68 81 92 97
Pressione qualquer tecla para continuar. . .

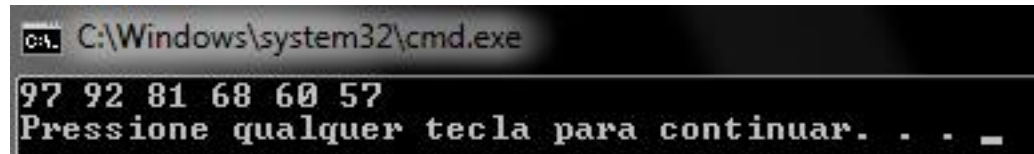
Consultas LINQ

```
static void Main(string[] args)
{
    // Declara e inicializa um array
    int[] Notas = new int[] { 57, 68, 97, 92, 81, 60 };

    // Define a expressão de consulta
    // Cria uma consulta que ordena as notas em ordem decrescente
    IEnumerable<int> notasQuery =
        from nota in Notas
        orderby nota descending
        select nota;

    // Executa a consulta
    foreach (int i in notasQuery)
    {
        Console.Write(i + " ");
    }

    Console.WriteLine(); // linha em branco
}
```



Desafio 1

Faça um programa que:

- 1) Solicite a entrada de dois números (limite inferior e limite superior) que representam o intervalo a ser usado na função random. O programa deve se certificar que o intervalo inferior seja menor que o intervalo superior. Caso contrário, o usuário deve digitar os limites novamente.
- 2) Em seguida, o programa solicita a quantidade de números aleatórios a serem guardados e gerados em um array de inteiros. A quantidade deve ser menor que os valores possíveis dentro do intervalo e devem ser únicos dentro do array.
- 3) Após, imprimir o array na ordem em que os números foram gerados, bem como, a quantidade de tentativas que foram necessárias para gerar os números.
- 4) Imprima também quantos números repetidos foram gerados.
- 5) Imprima o array em ordem crescente. (Use o Linq)
- 6) Imprima todos os elementos pares maiores que a metade do intervalo.
- 7) Crie uma função recursiva que calcule e mostre o fatorial do menor número do array.
- 8) Crie um novo array contendo os elementos do primeiro array em ordem decrescente. Em seguida, chame a função CalculaRaizes, que calcula e mostra a raiz quadrada de todos os ímpares do array.

Arrays

Multidimensionais

Arrays Multidimensionais

- Os arrays multidimensionais com duas dimensões costumam ser usados para representar **tabelas de valores** que consistem nas informações dispostas em **linhas** e **colunas**.
- Para identificar um elemento de uma tabela em particular, devemos especificar dois índices.
- Por convenção, o primeiro identifica a linha do elemento e o segundo, a coluna.

Arrays Multidimensionais

- Os arrays que requerem dois índices para identificar um elemento particular são chamados de ***arrays bidimensionais***.
- “Os arrays multidimensionais podem ter mais de duas dimensões.”
- Em geral, um array com **X** linhas e **Y** colunas é chamado de **array X por Y**.

Arrays Multidimensionais

- Cada elemento do array é identificado por uma expressão de acesso ao array da forma **a**[**linha**][**coluna**].
- Onde: **a** é o nome do array e **linha** e **coluna** são os índices que identificam unicamente cada elemento do array **a** por número de linha e coluna.

Arrays Multidimensionais

Array Unidimensional

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Array Bidimensional

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Linha 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Linha 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Observe que todos os nomes dos elementos da linha 0 tem o primeiro índice igual a 0; E todos os nomes dos elementos da coluna 3 tem o segundo índice igual a 3.

Declarando e Inicializando Arrays Multidimensionais

- Pode ser feitos de várias maneiras:

```
// Declarando um array bi-dimensional
int[,] array = new int[2,2];
```

```
// Inicializando array com elementos
array[0, 0] = 1;
array[0, 1] = 2;
array[1, 0] = 3;
array[1, 1] = 4;
```

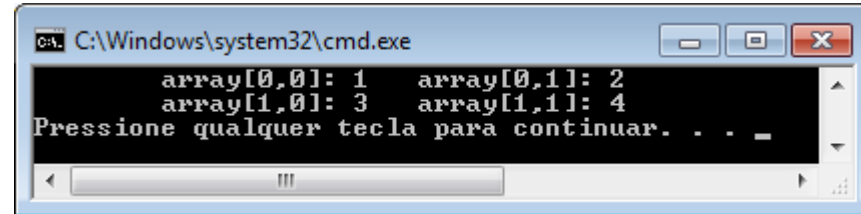
```
// Declarando um array bi-dimensional
int[,] array2;
array2 = new int[2,2];
```

```
// Declarando e Inicializando um array bi-dimensional
int[,] array3 = { { 1, 2 }, { 3, 4 } };
```

	Coluna 0	Coluna 1
Linha 0	a[0][0]	a[0][1]
Linha 1	a[1][0]	a[1][1]

	Coluna 0	Coluna 1
Linha 0	1	2
Linha 1	3	4

Declarando e Inicializando Arrays Multidimensionais



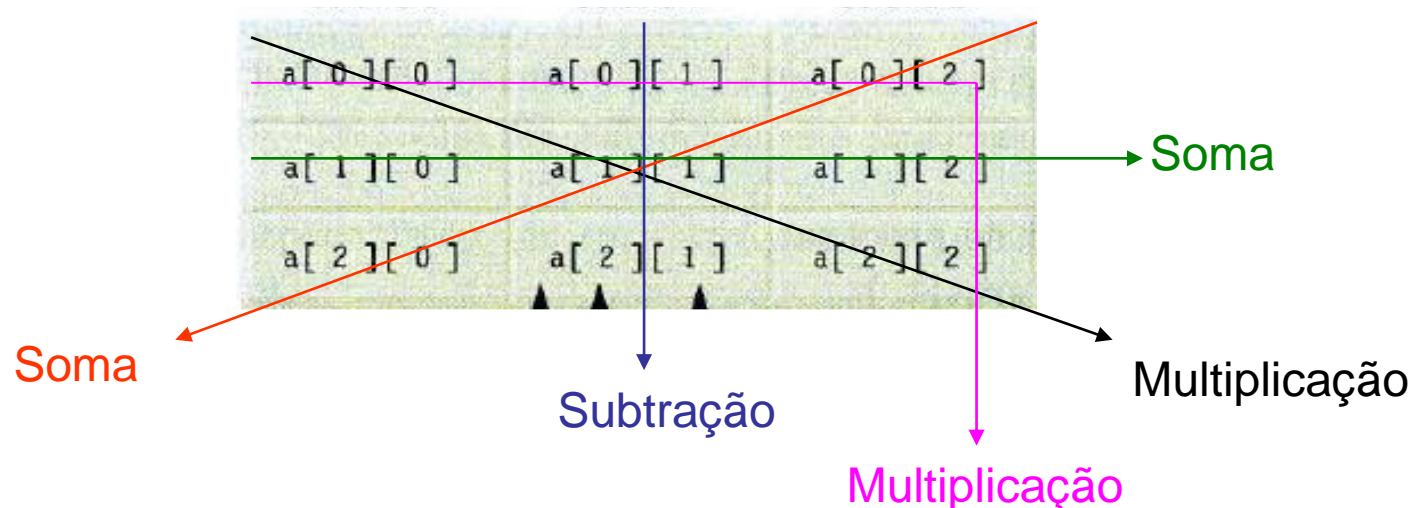
```
static void Main(string[] args)
{
    // Declarando e Inicializando um array bi-dimensional
    int[,] array = { { 1, 2 }, { 3, 4 } };

    // Estrutura de 2 FOR para acessar elementos de um
    // array bi-dimensional
    for (int linha = 0; linha < 2; linha++)
    {
        for (int coluna = 0; coluna < 2; coluna++)
        {
            Console.Write("\t array[{0},{1}]: {2}", linha, coluna, array[linha,coluna]);
        }
        Console.WriteLine();    // linha em branco
    }
}
```

Exercício 10

Crie um programa em C# que solicite ao usuário a inserção de 9 números inteiros que serão armazenados em um array **A[3,3]**.

- Exiba o array e a disposição dos números digitados.
- Em seguida, calcule e exiba o resultado das operações aritméticas sobre determinados elementos do array, de acordo como mostra a figura abaixo:



Exercício 11

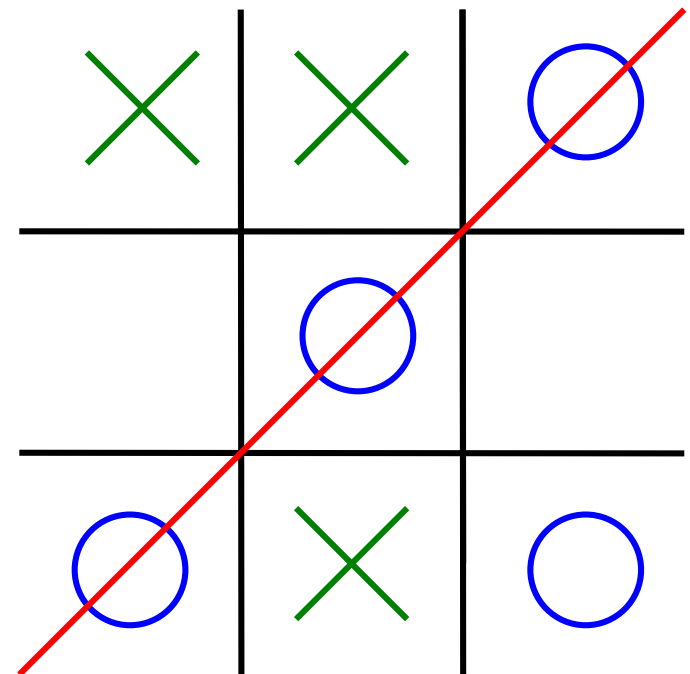
- Faça um algoritmo que solicite a inserção de valores em um array A 4x4.
- Em seguida mostre em quais posições do array estão todos os elementos pares. Faça a mesma coisa com os ímpares.
- Calcule e mostre também a soma de todos os elementos pares e dos elementos ímpares. Mostre também a quantidade de elementos pares e ímpares dentro do array.

Exercício 12 – Jogo da Velha

Neste jogo para dois jogadores, jogado sobre um tabuleiro de 3x3 casas, um dos jogadores escolhe uma casa e a marca com um círculo, e em seguida o outro escolhe outra casa e a marca com um xis.

Os jogadores continuam se alternando desta forma, até que uma linha com os mesmos símbolos seja formada, na vertical, horizontal ou diagonal, e neste caso o jogador que a fez, vence o jogo.

O jogo também acaba se não houver mais jogadas possíveis, e nesse caso é declarado empate.



Exercício 12 – Jogo da Velha

COMO FAZER – PARTE 1:

- Para este exercício deve ser implementado um programa Java que permita a duas pessoas jogarem Jogo da Velha entre si. Este exercício visa colocar em prática conhecimentos básicas de programação, como arrays, métodos, estruturas de controle de fluxo/repetição, classes, objetos e interação com o usuário.
- O tabuleiro deve ser implementado como um *array* 3x3, onde cada posição do *array* representa uma casa no jogo. Cada casa do tabuleiro deve acomodar três estados possíveis: vazio, círculo e xis. Portanto, o conteúdo do *array* pode ser *int* (0=vazio, 1=círculo, 2=xis), *char* ('_'=vazio, 'o'=círculo, 'x'=xis), ou qualquer outra coisa que permita representar estes três estados distintos.

Exercício 12 – Jogo da Velha

COMO FAZER – PARTE 2:

- Devem ser criados métodos para encapsular a lógica do jogo. Devem ser criados métodos para fazer uma jogada (indicando as coordenadas), checar se o jogo não acabou ainda, saber qual jogador ganhou (ou se deu empate), imprimir o tabuleiro na tela, e outros, conforme necessidade.
- Durante a execução do programa, cada jogador deve escrever sua jogada (coordenadas) na linha de comando, e o jogo deve imprimir o tabuleiro e esperar pela jogada do próximo jogador. O programa não deve permitir que o jogador tente marcar uma casa que já esteja marcada, nem que tente jogar em casas que não existam (coordenadas negativas, ou maior que o tamanho do tabuleiro).

Desafio 2

Faça um programa que:

- 1) Solicite ao usuário os valores referentes à quantidade de linhas e colunas de um array bidimensional. Os valores devem ser maiores do que zero e menores do que 10.
- 2) Utilize O objeto Random para gerar números aleatórios entre -100 e 100 e preencha a matriz.
- 3) Em seguida faça:
 - a) O somatório de todos os elementos das linhas e das colunas. Imprima a matriz com os somatórios ao lado/abaixo. (com cor diferente (a gosto menos branco))
 - b) Crie um array unidimensional e guarde todos os valores primos encontrados na matriz. Exiba o array no final do programa.

Coleções

Coleções

Para muitos aplicativos, você deseja criar e gerenciar grupos de objetos relacionados. Há duas maneiras para agrupar objetos: criando matrizes de objetos e criando coleções de objetos.

Matrizes são mais úteis para criar e trabalhar com um número fixo de objetos fortemente tipados.

Coleções fornecem uma maneira mais flexível de trabalhar com grupos de objetos.

Ao contrário dos arrays, o grupo de objetos com o qual você trabalha pode crescer e reduzir dinamicamente conforme as necessidades do aplicativo mudam.

Coleções

Uma coleção é uma classe, portanto, você deve declarar uma nova coleção antes de adicionar elementos a essa coleção.

Se sua coleção contiver elementos de apenas um tipo de dados, você pode usar uma das classes no namespace ***System.Collections.Generic*** e ***System.Collections***

Uma coleção genérica impõe a segurança de tipo para que nenhum outro tipo de dados possa ser adicionado a ela.

Quando você recupera um elemento de uma coleção genérica, não precisa determinar seu tipo de dados ou convertê-lo.

Coleções

Nome	Descrição
ArrayList	Uma simples coleção de objetos redimensionável e baseada em index.
SortedList	Uma coleção de pares nome/valor ordenada por chave.
Queue	Uma coleção de objetos First-in, First-out.
Stack	Uma coleção de objetos Last-in, First-out.
Hashtable	Uma coleção de pares de objeto nome/valor que podem ser acessados tanto por nome como por índice. Eficiente para grandes quantidades de dados.
BitArray	Coleção Compactas de valores Boolean.

Coleções

Nome	Descrição
StringCollection	Simple coleção redimensionável de strings.
StringDictionary	Uma coleção de pares de strings nome/valor que podem ser acessados tanto por nome como por index.
ListDictionary	Uma coleção eficiente para armazenar pequenas listas de objetos.
HybridDictionary	Trabalha como uma ListDictionary quando o número de itens a serem armazenados é pequeno e migra para Hashtable quando o número de itens aumenta.
NameValueCollection	Uma coleção de pares de strings nome/valor que podem ser acessados tanto por nome como por index. Podem ser adicionados vários valores em uma mesma chave.

ArrayList

Classe que permite armazenar qualquer tipo de informação

- Guarda tipos object
- Tipos por valor causa boxing

Tamanho é flexível, aumentando quando chega ao máximo atual

ArrayList

Métodos:

- **Add:** Adiciona o parâmetro no fim da coleção
- **AddRange:** Adiciona itens de outra coleção no fim desta coleção.
- **Insert:** Adiciona item em um índice indicado
- **InsertRange:** Adiciona itens de outra coleção a partir de um índice indicado.
- **Capacity:** Número máximo de elementos do ArrayList;
- **Count:** Número atual de elementos do ArrayList;
- **IndexOf:** busca um objeto e retorna o índice onde ele está
- **Remove:** Retira um objeto da lista

ArrayList

Ao trabalhar com ArrayList, cuidado com os tipos guardados.
É necessário fazer conversão.

```
ArrayList colecao = new ArrayList();  
colecao.Add("Nome");  
colecao.Add(3);  
colecao.Add(DateTime.Now);  
int valor = (int) colecao[1];  
if( colecao.Contains("Nome") )  
{  
    colecao.Remove("Nome");  
}
```

ArrayList – Iterando em coleções

Necessários para o foreach:

- IEnumerable: GetEnumerator()
- IEnumerator: Current() e MoveNext()

```
IEnumerator enumerator = coll.GetEnumerator();  
while (enumerator.MoveNext()) {  
    Console.WriteLine(enumerator.Current);  
}  
foreach (string item in coll) {  
    Console.WriteLine(item);  
}
```

Queue (Fila)

Queue (ou fila) é uma estrutura de dados do tipo (FIFO – First In First Out), que simula o comportamento de uma fila.

Fornece métodos (**Enqueue**): para inserir objeto no fim da fila e (**Dequeue**) para retirar objeto que se encontra no início da fila.

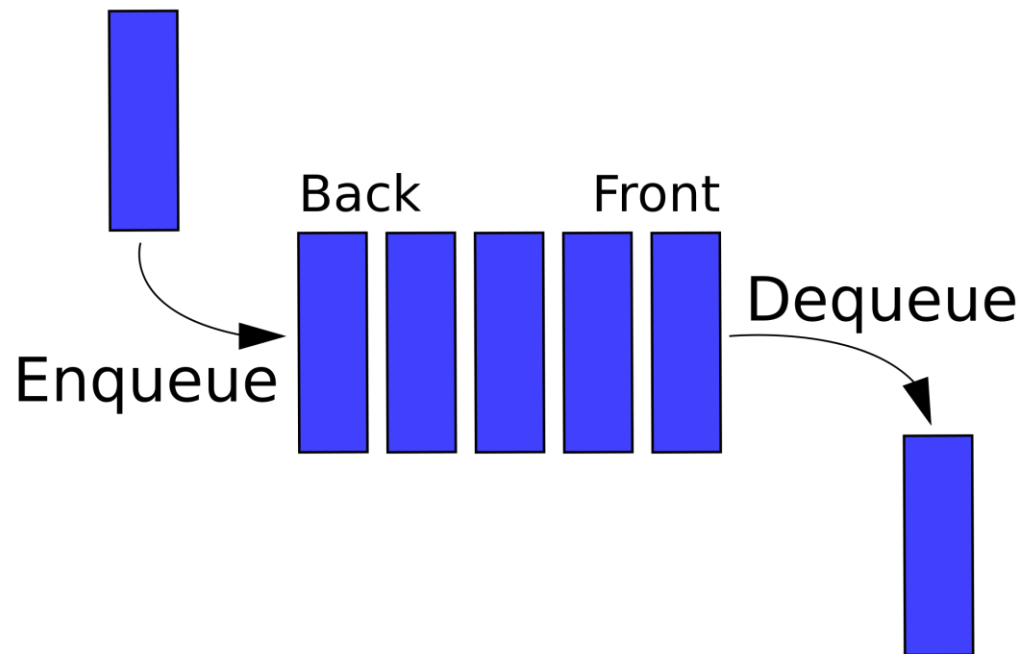
Um método (**Peek**) também pode ser usado para retornar o primeiro objeto da fila mas sem removê-lo.

O atributo **Count** pode ser usado para verificar o tamanho da fila. (i.e. quantidade de objetos na fila).

O método **ToArray** pode ser usado para converter a fila em um array simples.

A Queue (fila) é uma estrutura de dados dinâmica, ou seja, seu tamanho é modificado a medida que objetos são inseridos e removidos.

Queue (Fila)



Queue (Fila)

```
var q = new Queue<int>(); // Cria uma Fila

q.Enqueue(10); // Insere um valor no final da fila
q.Enqueue(20); // Insere um valor no final da fila

int[] data = q.ToArray(); // Converte uma fila em um array

Console.WriteLine(q.Count); // Retorna a quantidade de objetos na fila "2"

Console.WriteLine(q.Peek()); // Retorna qual o primeiro objeto na fila "10"

Console.WriteLine(q.Dequeue()); // Retira o primeiro objeto da fila "10"
Console.WriteLine(q.Dequeue()); // Retira o primeiro objeto da fila "20"

Console.WriteLine(q.Dequeue()); // causa uma exceção (fila vazia)
```

Stack (Pilha)

Stack (ou pilha) é uma estrutura de dados do tipo (LIFO – Last In First Out), que simula o comportamento de uma pilha.

Fornece métodos (**Push**): para inserir objeto no topo da pilha e (**Pop**) para retirar objeto que se encontra no topo da pilha.

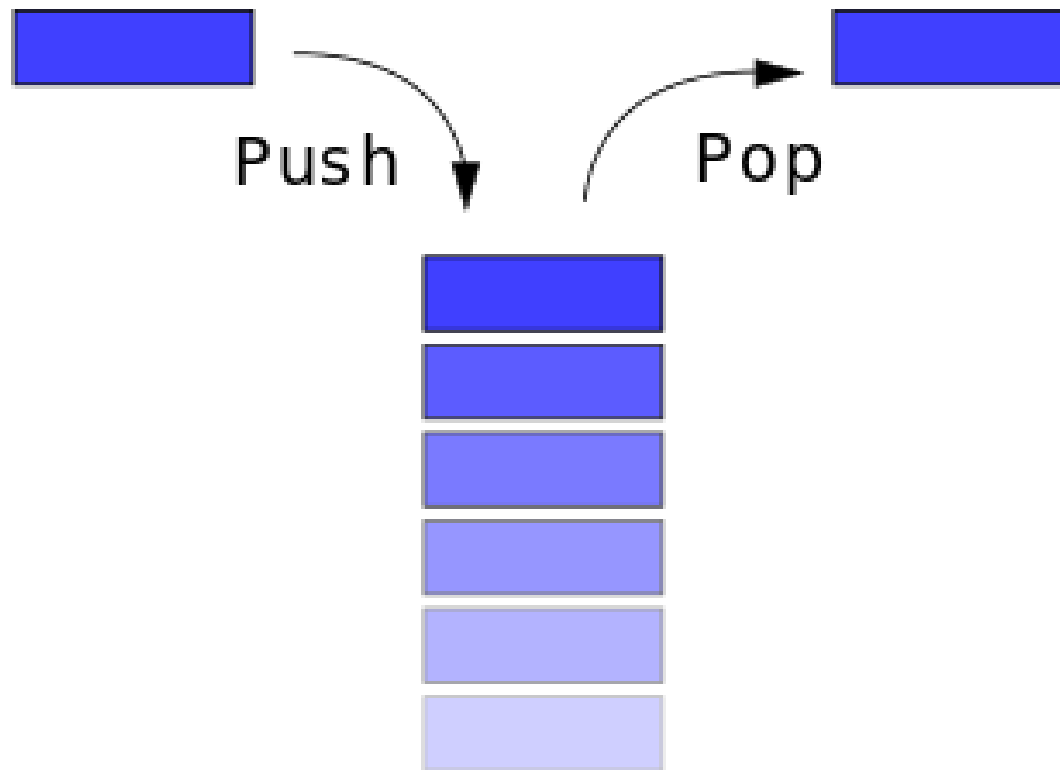
Um método (**Peek**) também pode ser usado para retornar o objeto do topo da pilha mas sem removê-lo.

O atributo **Count** pode ser usado para verificar o tamanho da pilha. (i.e. quantidade de objetos na fila).

O método **ToArray** pode ser usado para converter a pilha em um array simples.

A Stack (pilha) é uma estrutura de dados dinâmica, ou seja, seu tamanho é modificado a medida que objetos são inseridos e removidos.

Stack (Pilha)



Stack (Pilha)

```
var s = new Stack<int>(); // Cria uma Pilha

s.Push(1); // Insere um valor no topo da pilha = 1
s.Push(2); // Insere um valor no topo da pilha = 1,2
s.Push(3); // Insere um valor no topo da pilha = 1,2,3

Console.WriteLine(s.Count); // Retorna a quantidade de objetos na pilha 3

Console.WriteLine(s.Peek()); // Retorna qual o primeiro objeto na pilha 3, Stack = 1,2,3

Console.WriteLine(s.Pop()); // Retira o primeiro objeto da pilha 3, Stack = 1,2
Console.WriteLine(s.Pop()); // Retira o primeiro objeto da pilha 2, Stack = 1
Console.WriteLine(s.Pop()); // Retira o primeiro objeto da pilha 1, Stack = <vazia>
Console.WriteLine(s.Pop()); // causa uma exceção (pilha vazia)
```

SortedList (Lista Ordenada)

Representa uma coleção de pares chave/valor que são ordenados pelas chaves e são acessíveis por chave e por índice.

Fornece métodos (**Add**): para inserir objeto na lista e (**Remove/RemoveAt**) para retirar um objeto da lista.

Métodos (**Contains/ContainsKey/ContainsValue**) são usados para determinar se uma chave ou valor específicos se encontram na lista.

O atributo **Count** pode ser usado para verificar o tamanho da lista. (i.e. quantidade de objetos na lista).

O método **Clear** pode ser usado para esvaziar a lista.

SortedList (Lista Ordenada)

```
static void Main(string[] args)
{
    // Cria e inicializa uma SortedList.
    SortedList mySL = new SortedList();

    // Adiciona alguns elementos chave/valor
    mySL.Add("3c", "dog");
    mySL.Add("2c", "over");
    mySL.Add("1c", "brown");
    mySL.Add("1a", "The");
    mySL.Add("1b", "quick");
    mySL.Add("3a", "the");
    mySL.Add("3b", "lazy");
    mySL.Add("2a", "fox");
    mySL.Add("2b", "jumped");

    // Mostra os elementos e suas chaves
    Console.WriteLine("A SortedList contém inicialmente os valores:");
    PrintKeysAndValues(mySL);

    // Remove o elemento com a chave "3b".
    mySL.Remove("3b");

    // Mostra o estado atual da SortedList após a remoção do elemento "3b".
    Console.WriteLine("Depois de remover \"lazy\":");
    PrintKeysAndValues(mySL);

    // Remove o elemento que está no índice 5.
    mySL.RemoveAt(5);

    // Mostra o estado atual da SortedList após a remoção do elemento na posição 5.
    Console.WriteLine("Depois de remover elemento na posição 5:");
    PrintKeysAndValues(mySL);
}
```

```
public static void PrintKeysAndValues(SortedList myList)
{
    Console.WriteLine("\t-KEY-\t-VALUE-");
    for (int i = 0; i < myList.Count; i++)
    {
        Console.WriteLine("\t{0}:\t{1}", myList.GetKey(i), myList.GetByIndex(i));
    }
    Console.WriteLine();
}
```



```
C:\Windows\system32\cmd.exe
A SortedList contém inicialmente os valores:
-KEY- -VALUE-
1a: The
1b: quick
1c: brown
2a: fox
2b: jumped
2c: over
3a: the
3b: lazy
3c: dog

Depois de remover "lazy":
-KEY- -VALUE-
1a: The
1b: quick
1c: brown
2a: fox
2b: jumped
2c: over
3a: the
3c: dog

Depois de remover elemento na posição 5:
-KEY- -VALUE-
1a: The
1b: quick
1c: brown
2a: fox
2b: jumped
3a: the
3c: dog

Pressione qualquer tecla para continuar. . .
```

SortedList (Lista Ordenada)

```
static void Main(string[] args)
{
    // Cria e inicializa uma SortedList (Lista Ordenada).
    SortedList mySL = new SortedList();
    mySL.Add(2, "two");
    mySL.Add(4, "four");
    mySL.Add(1, "one");
    mySL.Add(3, "three");
    mySL.Add(0, "zero");

    // Mostra os valores e suas chaves na SortedList.
    Console.WriteLine("The SortedList contains the following values:");
    PrintIndexAndKeysAndValues(mySL);

    // Procura por uma chave específica
    int myKey = 2;
    Console.WriteLine("A Chave \"{0}\" {1}.", myKey, mySL.ContainsKey(myKey) ? "está na SortedList" : "NÃO está na SortedList");
    myKey = 6;
    Console.WriteLine("A Chave \"{0}\" {1}.", myKey, mySL.ContainsKey(myKey) ? "está na SortedList" : "NÃO está na SortedList");

    // Procura por um valor específico
    String myValue = "three";
    Console.WriteLine("O Valor \"{0}\" {1}.", myValue, mySL.ContainsValue(myValue) ? "está na SortedList" : "NÃO está na SortedList");
    myValue = "nine";
    Console.WriteLine("O Valor \"{0}\" {1}.", myValue, mySL.ContainsValue(myValue) ? "está na SortedList" : "NÃO está na SortedList");
}

public static void PrintKeysAndValues(SortedList myList)
{
    Console.WriteLine("\t-KEY-\t-VALUE-");
    for (int i = 0; i < myList.Count; i++)
    {
        Console.WriteLine("\t{0}:\t{1}", myList.GetKey(i), myList.GetByIndex(i));
    }
    Console.WriteLine();
}
```

A SortedList contém os seguintes valores:

-INDEX-	-KEY-	-VALUE-
[0]:	0	zero
[1]:	1	one
[2]:	2	two
[3]:	3	three
[4]:	4	four

A Chave "2" está na SortedList.
 A Chave "6" NÃO está na SortedList.
 O Valor "three" está na SortedList.
 O Valor "nine" NÃO está na SortedList.
 Pressione qualquer tecla para continuar. . .

Exercício 1

Faça um programa que solicite a digitação de 'N' númeos inteiros. Armazene todos em um List de inteiros chamado ListaDeNumeros. *A List não pode conter números repetidos.*

Quando o usuário digitar o valor 0 (zero) o programa deverá mostrar:

- 1) O somatório dos valores digitados.
- 2) Mostar a posição dentro do List aonde encontra-se o maior e o menor valor.
- 3) Crie um novo list chamado ListaDePares e mova todos os números pares do ListaDeNumeros para ele.
- 4) Se o tamanho dos arrays for igual, criar um novo array chamado ListaDasSommas, onde cada valor corresponde a soma dos elementos de ListaDeNumeros e ListaDePares localizados no mesmo índice.

Exercício 2

1) Faça um programa que gere 'n' números aleatórios entre -50 e 50 e guarde em uma Fila (Queue) chamada FilaDeNumeros.

A quantidade 'n' deve ser definida pelo usuário no início do programa. Se o valor zero for gerado, ignore-o.

2) Em seguida, armazene todos os valores ímpares em uma pilha (Stack) chamada PilhaDeImpares. Faça também uma PilhaDePares para guardar todos os valores pares.

3) Se o tamanho das pilhas for igual, criar um List chamado ListaDeNumeros, que conterà a soma dos elementos das duas pilhas respectivamente.

Senão, mostrar a raiz quadrada de todos os valores da PilhaDePares e o cosseno dos valores da PilhaDeImpares.

Desafio 3

O **Bubble Sort**, ou ordenação por flutuação (literalmente "por bolha"), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, a cada passagem fazendo “flutuar” para o topo o maior elemento da sequência.

Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

a) Crie um programa que usa o algoritmo de ordenação Bubble Sort, para ordenar um array de 10 valores inteiros em ordem crescente.

6 5 3 1 8 7 2 4

Desafio 3

O **Merge Sort**, ou ordenação por mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar.

Sua ideia básica consiste em Dividir (o problema em vários sub-problemas e resolver esses sub-problemas através da recursividade) e Conquistar (após todos os sub-problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas).

b) Crie um programa que usa o algoritmo de ordenação Merge Sort, para ordenar um array de 10 valores inteiros em ordem crescente.

6 5 3 1 8 7 2 4

Desafio 3

c) Crie um programa que crie uma matriz 5x5 e preencha com valores aleatórios entre -25 a 25.

Não devem haver elementos repetidos dentro da matriz.

Mostre a matriz.

Em seguida, ordene os valores das linhas da matriz como segue:

Linha 1: ordem crescente

Linha 2: ordem decrescente

Linha 3: pares antes dos ímpares

Linha 4: números negativos antes dos positivos

Linha 5: substitua os valores positivos pela sua raiz quadrada.

No final, mostre a matriz com as modificações.