



Colégio
Pedro II

PROGRAMAÇÃO O.O.

(C#)



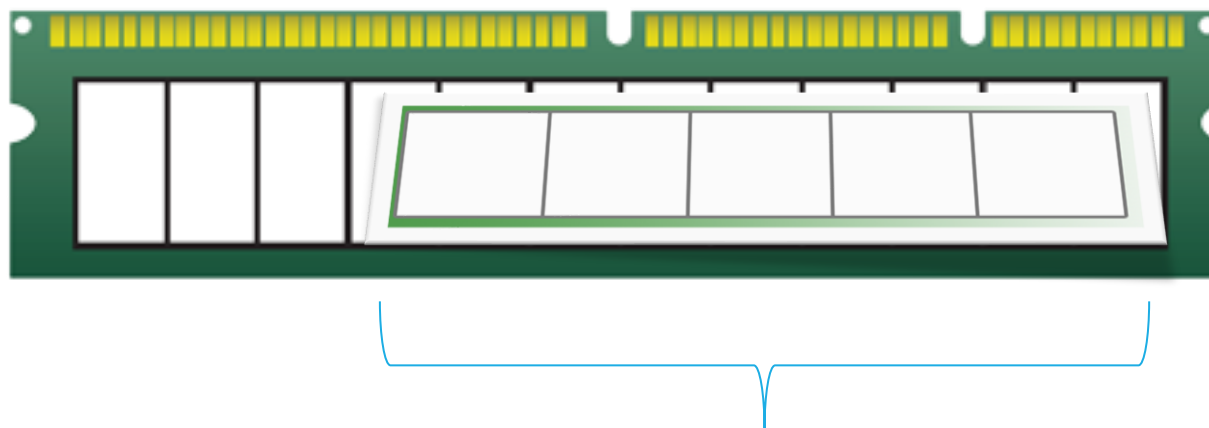
Vetores/Arrays x Listas: inserindo, percorrendo e removendo
Professor: João Luiz Lagôas



Array



- Uma array (ou vetor) é um grupo de posições de memória adjacentes que tem o mesmo nome e tipo.

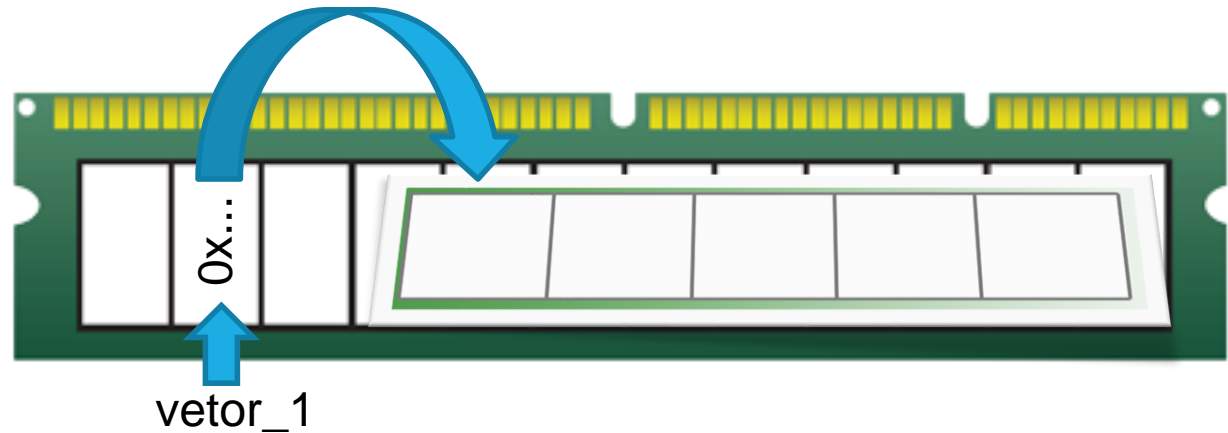


Array ou vetor

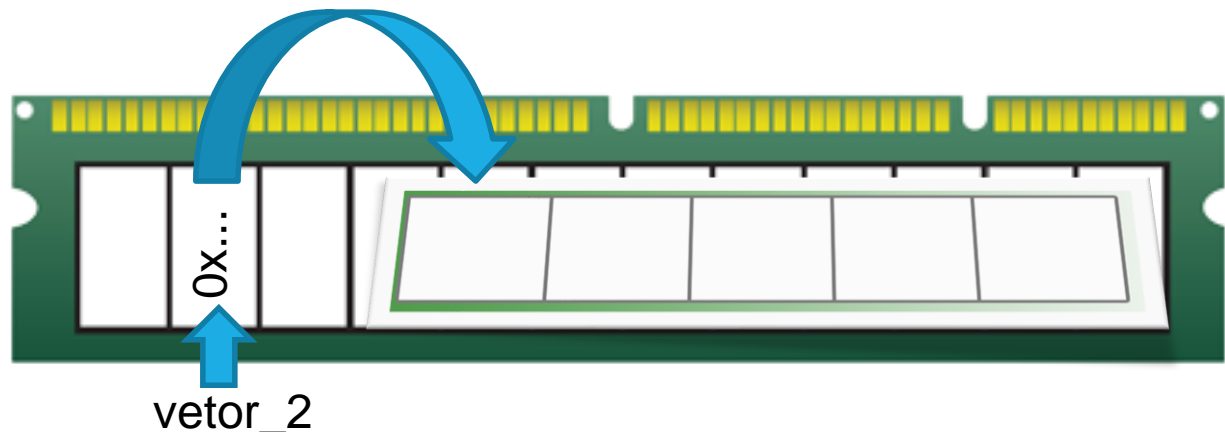
Declarando e Criando uma array

Podemos criar uma array de duas formas diferentes, mas em ambos os casos **devemos determinar o seu tamanho.**

```
int[] vetor_1 = new int[5];
```



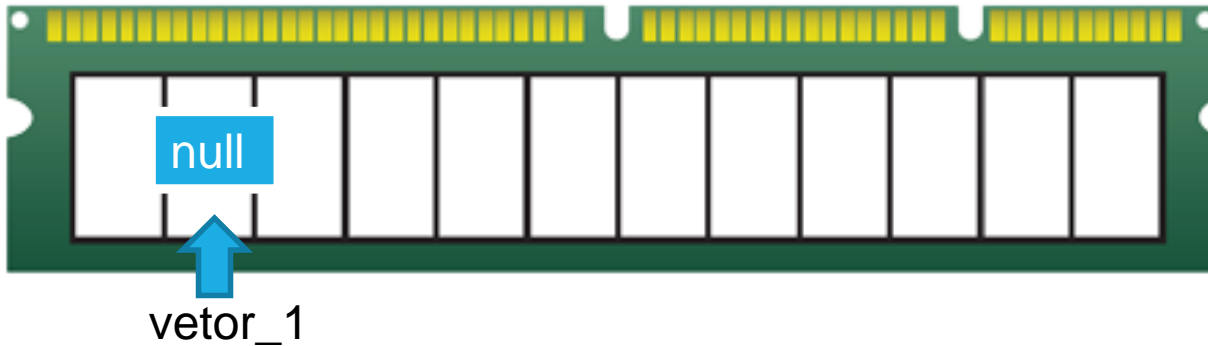
```
int[] vetor_2 = { 1, 4, 1, 2, 4 };
```



Declarando e Criando uma array

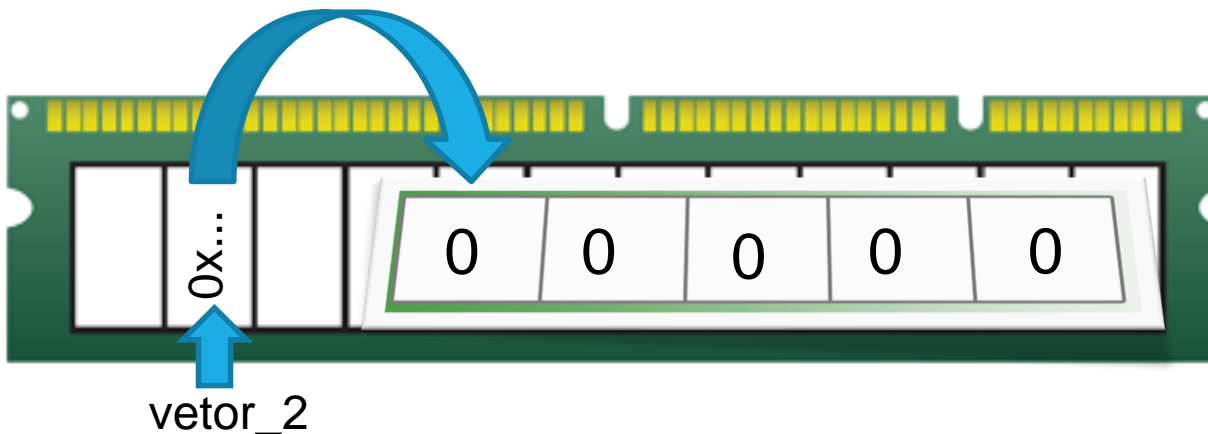
```
int[] vetor_1;
```

← VARIÁVEL QUE É DO TIPO VETOR DE INTEIROS.



```
vetor_1 = new int[5];
```

← CRIAÇÃO DO VETOR NA MÉMORIA

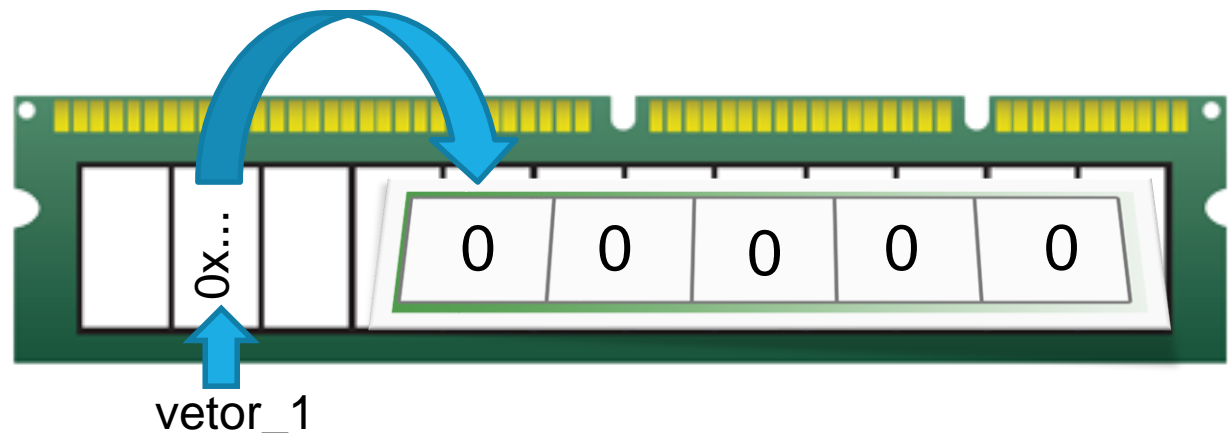


Valores default de uma array



- Quando os arrays são alocados, os elementos são inicializados com zero para variáveis numéricas, com false para variáveis booleanas e null para variáveis por referência.

```
int[] vetor_1 = new int[5];
```




Tamanho da array

- Toda array em C# conhece seu próprio tamanho, basta escrever o comando `nome_da_array.Length`.

```
int[] vetor_1 = new int[5];  
Console.WriteLine(vetor_1.Length);
```

Irá imprimir 5 no Console

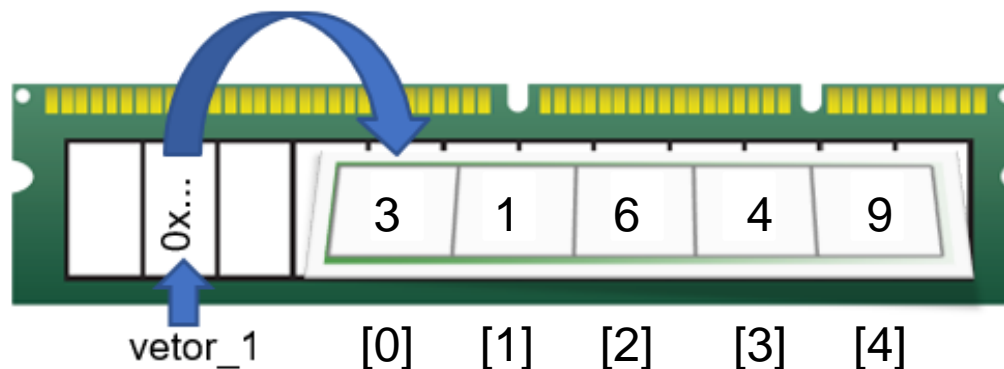


Acessando os elementos



- Para nos referirmos a uma posição ou elemento particular na array, especificamos o nome da array e um **índice**.

```
int[] vetor_1 = new int[5];  
  
vetor_1[0] = 3;  
vetor_1[1] = 1;  
vetor_1[2] = 6;  
vetor_1[3] = 4;  
vetor_1[4] = 9;  
  
Console.WriteLine(vetor_1[2]);
```



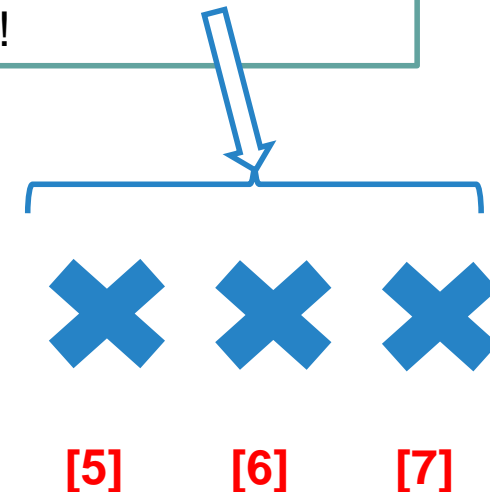
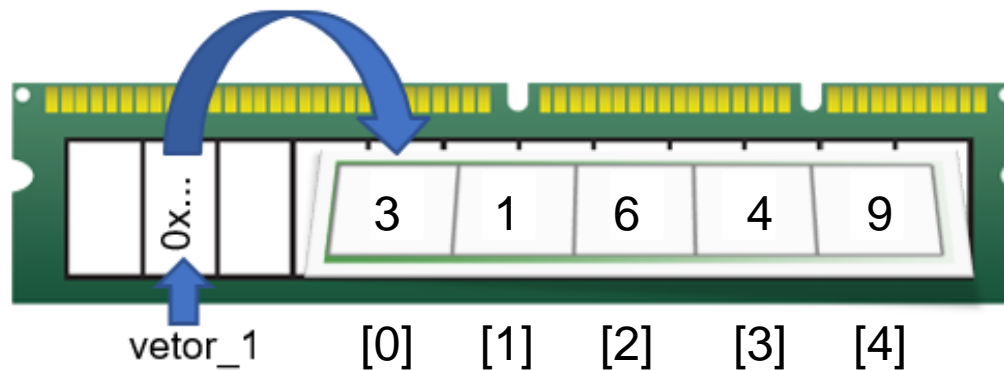
Acessando os elementos



- Se uma posição de memória for acessada no vetor mas nunca foi alocada (ultrapassar os limites da array), uma **exceção** será gerada.

```
int[] vetor_1 = new int[5];  
  
vetor_1[0] = 3;  
vetor_1[1] = 1;  
vetor_1[2] = 6;  
vetor_1[3] = 4;  
vetor_1[4] = 9;  
  
Console.WriteLine(vetor_1[7]);
```

Essas posições nunca foram alocadas na criação do vetor!

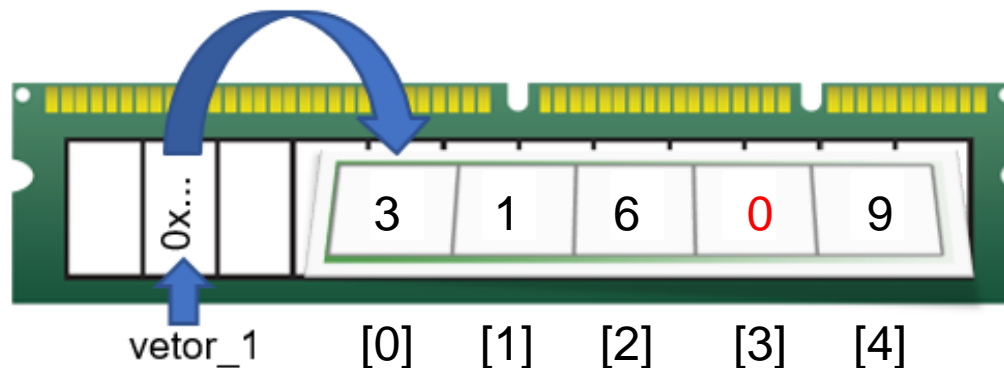


Acessando os elementos



- Não é possível remover elementos do vetor, apenas substituir por novos valores. O tamanho do vetor ainda é 5.

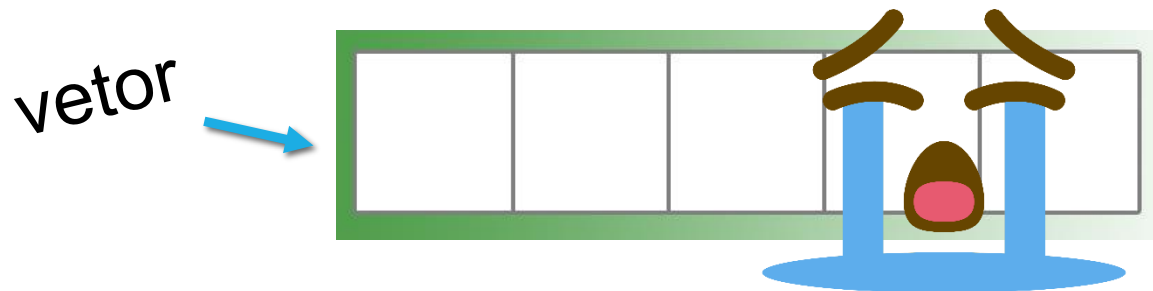
```
int[] vetor_1 = new int[5];  
  
vetor_1[0] = 3;  
vetor_1[1] = 1;  
vetor_1[2] = 6;  
vetor_1[3] = 0;  
vetor_1[4] = 9;  
  
Console.WriteLine(vetor_1[2]);
```



Problemas com vetores...



- Apesar de ser comumente usada, arrays/vetores têm algumas limitações.
 - O tamanho deve ser especificado e é fixo na hora de se criar um novo vetor;
 - Não é possível remover elementos do vetor e eliminar a alocação da posição de memória.



Coleções







- C# provê várias classes conhecidas como *collections*.
- Essas classes são usadas para armazenar conjuntos de dados de um mesmo tipo (assim como arrays).
- A vantagem de se utilizar classes *collections* é que muitas delas **resolvem problemas que existem em vetores entre outras características positivas.**

Coleções



- O Framework .NET tem diversas classes de coleções que lidam com as situações complicadas encontradas pelo uso de vetores.
- Algumas dessas classes de coleções são:

	Queue	Representa uma coleção em formato de fila. O primeira a entrar é o primeiro a sair.
	Stack	Representa uma coleção em formato de pilha. O última a entrar é o primeiro a sair.
	HashTable	Representa uma coleção de pares chave-valor organizados com base no código hash da chave.
	List	Representa uma lista fortemente tipada de objetos que podem ser acessados por índice.

namespace
Collections

Listas ao invés de Arrays/Vetores



Colégio
Pedro II

- A classe mais popular e comum das coleções C# é denominada **List**.
- Um objeto do tipo List é semelhante com um objeto do tipo Array (vetor): é usado para armazenar vários elementos de dados.
- No entanto, objetos List apresentam diversas facilidades que não são encontradas em vetores.
- Uma vez que você cria um objeto do tipo List, é **fácil** adicionar, remover, observar e mesmo mover itens de um lugar da lista para o outro.



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();
```



```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```

Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();
```

```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

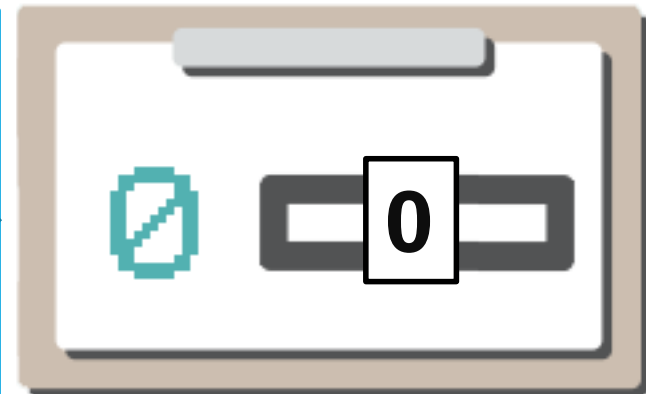


```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();
```

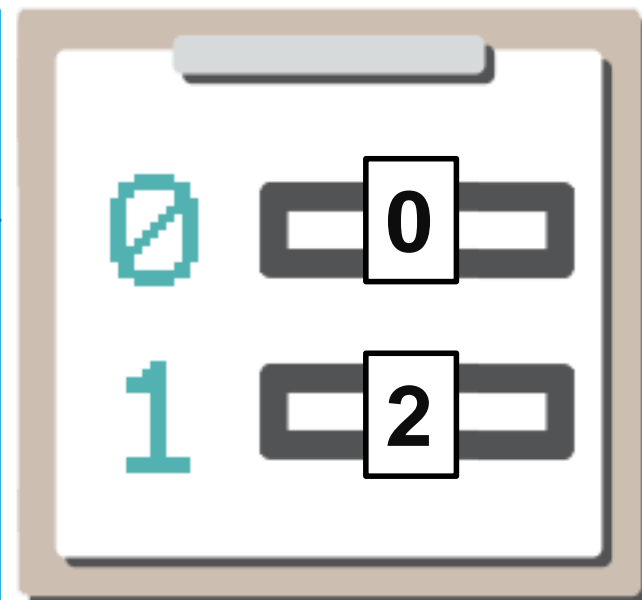
```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

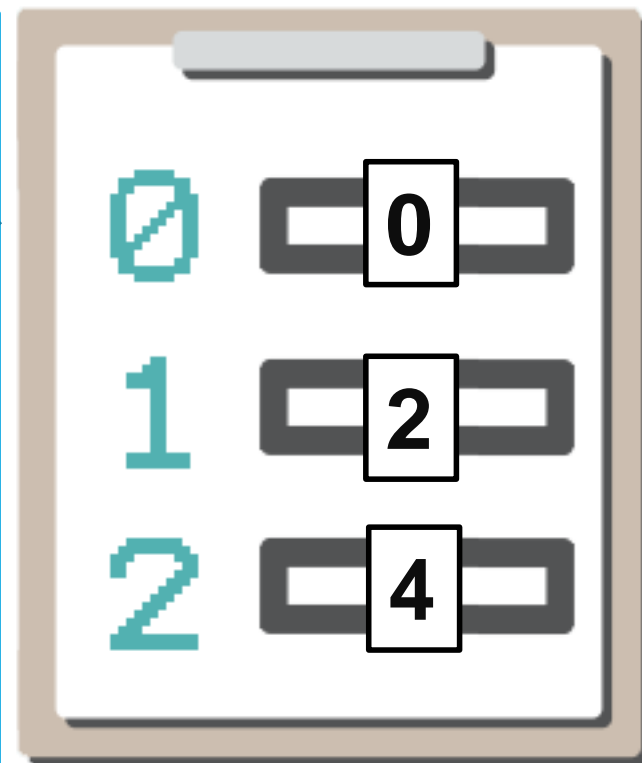
```
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

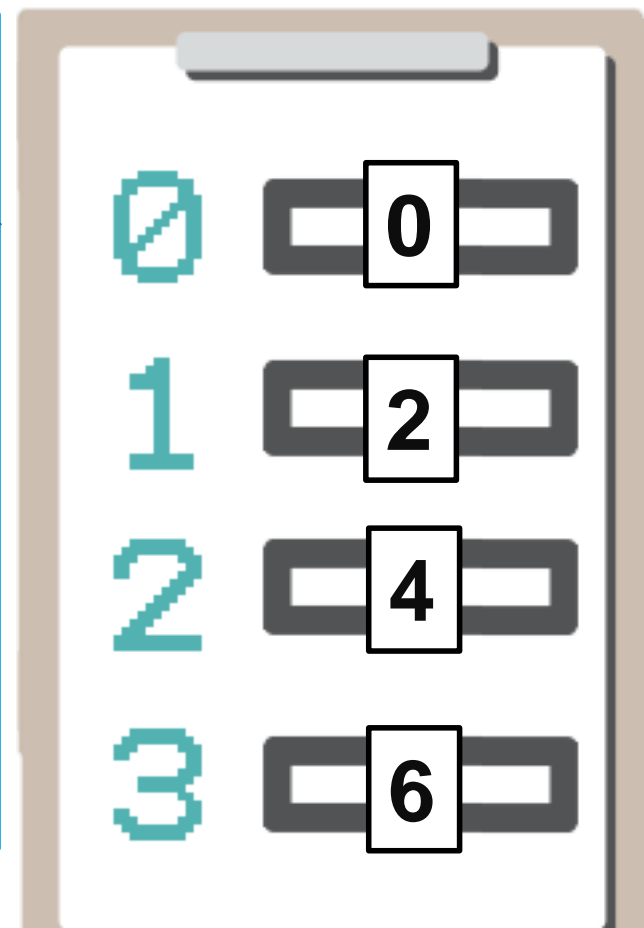
```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```





Exemplo

Console

0

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```

0

0

1

2

2

4

3

6

Exemplo



Colégio
Pedro II

Console

```
0  
2
```

```
List<int> minhaLista = new List<int>();
```

```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```

0

0

1

2

2

4

3

6



Exemplo

Console

```
0  
2  
4
```

```
List<int>
```

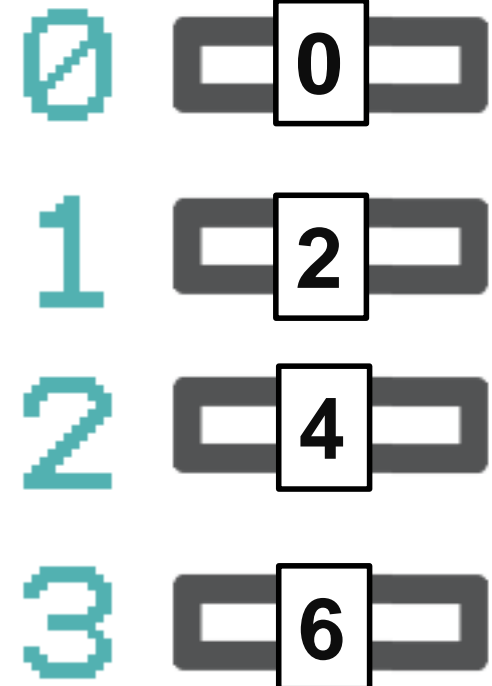
```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```





Exemplo

Console

```
0  
2  
4  
6
```

```
List<int>
```

```
for (int i = 0; i < 5; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```

0

0

1

2

2

4

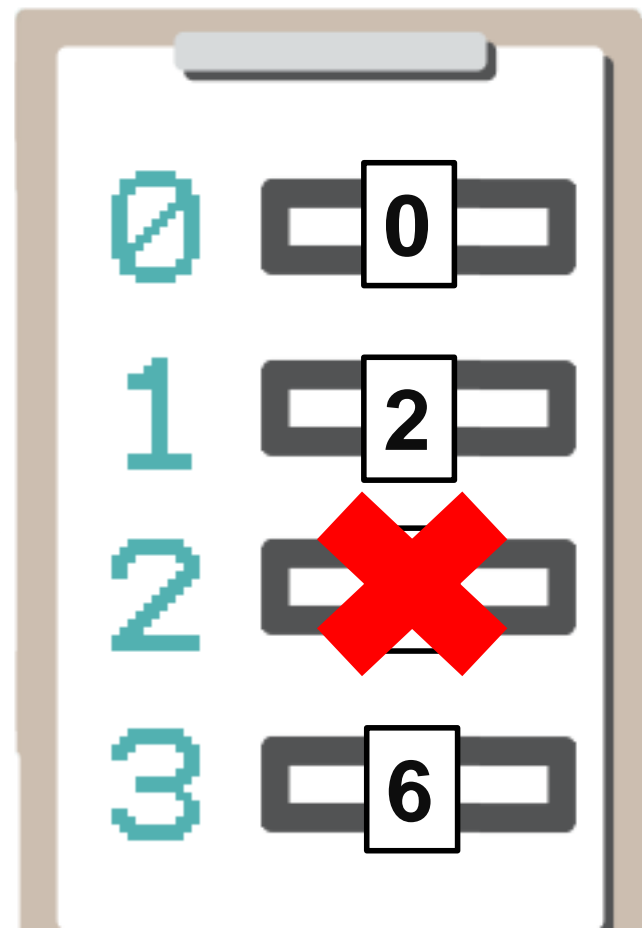
3

6

Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

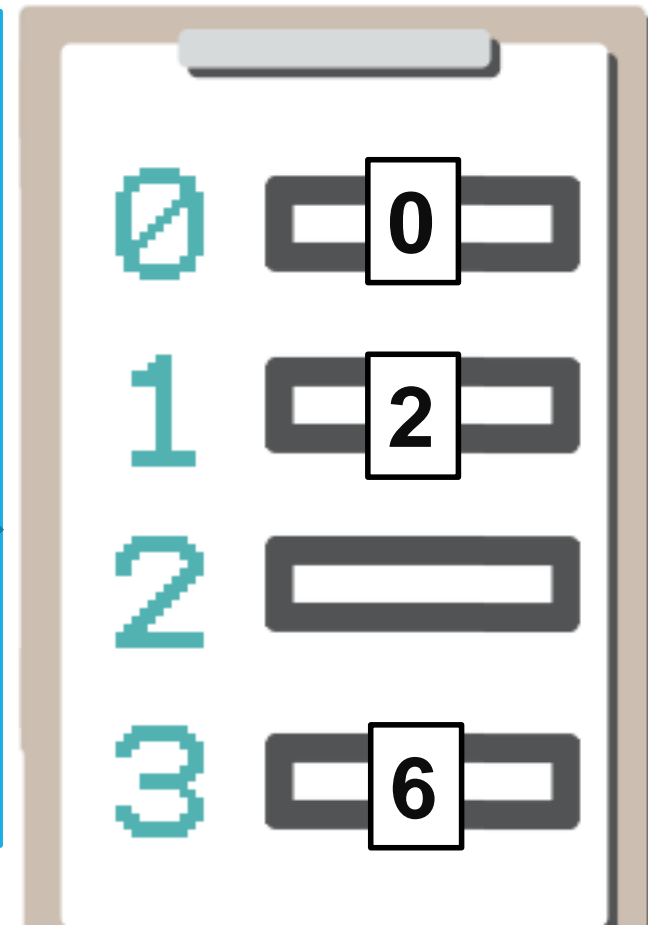
```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

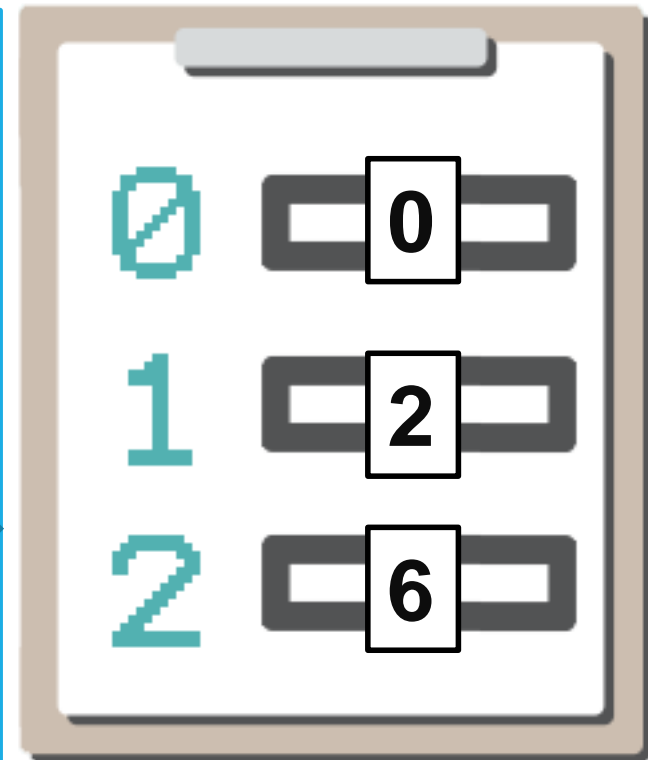
```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```



Exemplo

- Tente entender o que os comandos estão fazendo... Eles são intuitivos!

```
List<int> minhaLista = new List<int>();  
  
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
minhaLista.RemoveAt(2);  
  
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);  
  
Console.ReadLine();
```





Exemplo

Console

0

```
List<int> minhaLista = new List<int>();
```

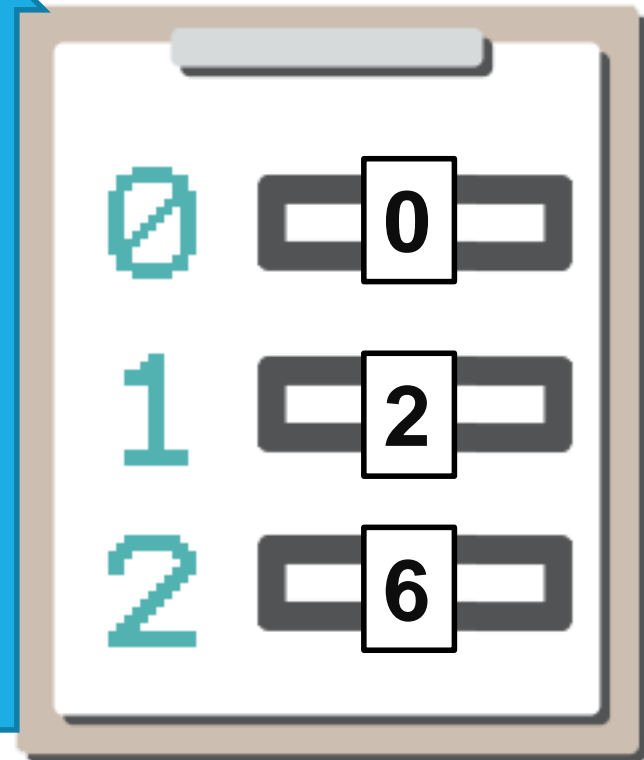
```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Exemplo



Console

```
0  
2
```

```
List<int> minhaLista = new List<int>();
```

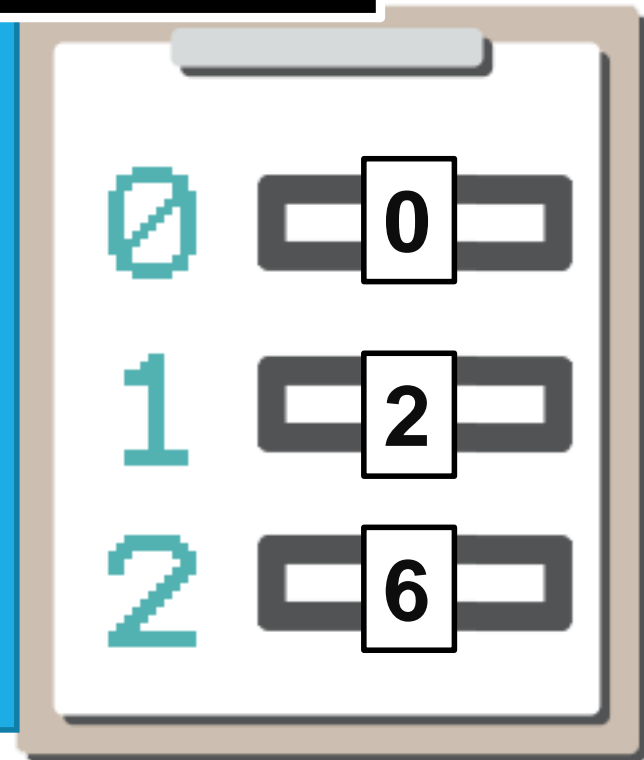
```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Exemplo



Colégio
Pedro II

Console

```
0
```

```
2
```

```
6
```

```
List<int>
```

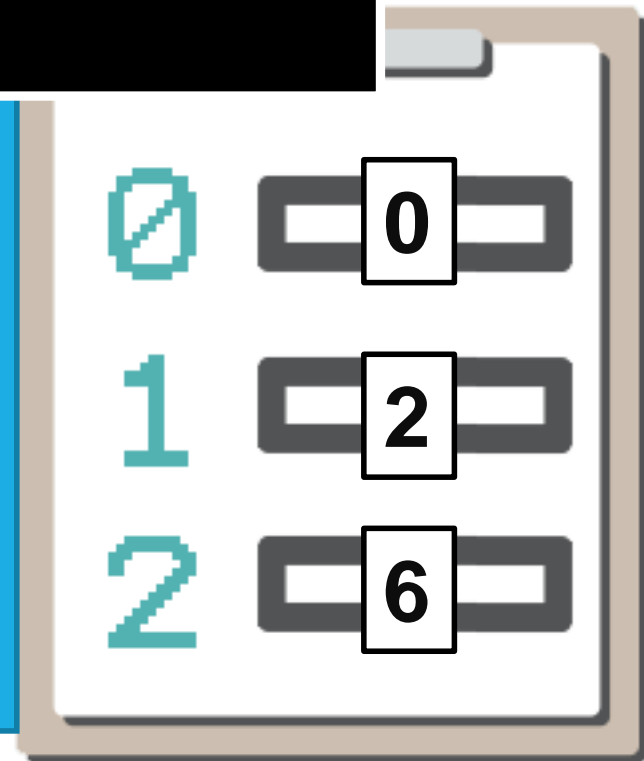
```
for (int i = 0; i <= 3; i++)  
    minhaLista.Add(i * 2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
minhaLista.RemoveAt(2);
```

```
for (int i = 0; i < minhaLista.Count; i++)  
    Console.WriteLine(minhaLista[i]);
```

```
Console.ReadLine();
```



Criando uma lista

- Exemplo:

```
List<int> minhaLista;  
minhaLista = new List<int>();
```

Declara uma List que pode armazenar inteiros.

Cria um objeto List<int> chamando o seu construtor.

```
List<string> minhaLista;  
minhaLista = new List<string>();
```

Declara uma List que pode armazenar strings.

Cria um objeto List<string> chamando o seu construtor.

Criando uma lista

Que "tag" é essa em frente ao nome da classe?



```
List<int> minhaLista;  
minhaLista = new List<int>();
```

Generics é um conceito criado em programação para especificar quando tipos de dados são utilizados como uma espécie de parâmetro.




Dessa forma, é possível a criação de classes que operam em dados de tipos diferentes.

O uso mais comum deste conceito é encontrado nas classes de coleção.

Listas encolhem e aumentam de tamanho dinamicamente










- Uma grande vantagem da Lista em relação a uma Array, é que você não precisa saber o tamanho dela quando você a cria.
- Ela pode aumentar e diminuir automaticamente para se adequar a necessidade do programador.
- Isso é feito através do uso dos métodos prontos e resolve os DOIS PROBLEMAS que enunciamos nas arrays!:

	Add(T)	Adiciona um elemento ao final da List<T>.
	RemoveAt(Int)	Remove o elemento no índice especificado do List<T>.
	Count	Retorna a quantidade de elementos que estão dentro da lista.

O que mais posso fazer com listas?

- Outros Atributos/Propriedades e Métodos

Atributos/
Propriedades

	Capacity	Indica o número de elementos que a lista pode suportar antes de ter que se redimensionar.
	Count	Retorna a quantidade de elementos que estão dentro da lista.
	Clear()	Remove todos os elementos da lista tornando-a uma lista vazia.
	Contains(T)	Retorna true se um determinado elemento está dentro da lista ou false caso contrário.
	IndexOf(T)	Retorna o índice de onde um determinado elemento se encontra na lista.
	Insert(Int, T)	Insere um elemento na lista na posição especificada como parâmetro.
	Sort()	Ordena os elementos da lista de acordo com um comparador padrão.

Métodos

Para os
curiosos...

Para saber mais sobre a classe List<T>, confira o site:
[https://msdn.microsoft.com/pt-br/library/6sh2ey19\(v=vs.110\).aspx](https://msdn.microsoft.com/pt-br/library/6sh2ey19(v=vs.110).aspx)