



Programação e Sistemas de Informação

CURSO PROFISSIONAL TÉCNICO DE
GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

Estruturas de Dados Estáticas: **Arrays**

MÓDULO 4

Professor: João Martiniano

Introdução

- Neste módulo iremos analisar em detalhe dois componentes importantes da linguagem C#:
 - o tipo de dados `string`
 - arrays

ARRAYS UNIDIMENSIONAIS

Introdução

- Um array permite armazenar vários valores numa única variável
- Existem várias situações em que é necessário guardar vários valores que estão relacionados
- Por exemplo:
 - as classificações, de uma turma com 20 alunos
 - sem arrays seria necessário ter 20 variáveis (uma para cada aluno)
- Utilizando um array apenas é necessário ter uma variável, a qual irá guardar as classificações de todos os alunos da turma

Introdução

- Exemplo 1: sem array, são necessárias 20 variáveis

```
int classificacaoAluno1 = 10;  
int classificacaoAluno2 = 12;  
int classificacaoAluno3 = 14;  
int classificacaoAluno4 = 16;  
....  
int classificacaoAluno20 = 10;
```

Classificação
do 1º aluno

Classificação
do 2º aluno

Classificação
do 3º aluno

Classificação
do último aluno

Introdução

- Exemplo 1: sem array, são necessárias 20 variáveis

```
int classificacaoAluno1 = 10;  
int classificacaoAluno2 = 12;  
int classificacaoAluno3 = 14;  
int classificacaoAluno4 = 16;  
....  
int classificacaoAluno20 = 10;
```

- Exemplo 2: com um array, apenas é necessária uma variável

```
int[] classificacoesAlunos = { 10, 12, 14, 16, ..., 10 };
```

Classificação
do 1º aluno

Classificação
do 2º aluno

Classificação
do 3º aluno

Classificação
do último aluno

Índices de elementos

- Tal como numa string, cada elemento de um array tem um **índice**
- O primeiro elemento tem índice 0

```
int[] classificacoesAlunos = { 10, 12, 14, 16, 18, 20, 10, 12 }
```

	10	12	14	16	18	20	10	12
Índice →	0	1	2	3	4	5	6	7

Aceder e modificar os elementos de um array

- É possível aceder a cada elemento de um array utilizando a seguinte sintaxe:

`variável[índice]`

- Exemplo: mostrar os dois primeiros elementos do array

```
Console.WriteLine("{0}, {1}", classificacoesAlunos[0], classificacoesAlunos[1]);
```

- Exemplo: modificar o terceiro elemento do array

```
classificacoesAlunos[2] = 15;
```


Declaração de arrays

- Ao declarar um array é sempre necessário especificar o tipo de dados que este irá conter
- Sintaxe para declaração de um array:

`tipo[] nome`

- Em que:

`tipo` O tipo de dados do array

`nome` O nome do array

- Exemplo:

```
string[] cidades = { "Lisboa", "Coimbra", "Viana do Castelo", "Faro" };
```

Declaração de arrays

- Existem várias formas de declarar arrays
- Exemplo 1: declarar um array, especificando o número de elementos (5) mas sem inicializar com valores

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

- Exemplo 2: declarar e inicializar um array com 5 elementos

```
int[] valores = new int[5] { 30, 33, 7, 18, 45 };
```

- Exemplo 3: forma alternativa de declarar e inicializar um array

```
int[] valores = { 30, 33, 7, 18, 45 };
```

- Como atribuir valores ao array do exemplo 1?

```
int[] valores = new int[5];  
valores[0] = 10;  
valores[1] = -14;  
...
```

Declaração de arrays

Exercício

- Declare e inicialize um array denominado `diasSemana`, que guarda os nomes dos dias da semana

Declaração de arrays

Exercício: Resolução

- Declare e inicialize um array denominado `diasSemana`, que guarda os nomes dos dias da semana

```
string[] diasSemana = new string[7] { "segunda-feira", "terça-feira",  
"quarta-feira", "quinta-feira", "sexta-feira", "sábado", "domingo" };
```

Declaração de arrays

- Os arrays podem ser declarados sem inicialização e sem a *keyword* `new`:

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

- No entanto, é depois necessário utilizar esta *keyword* quando o array for inicializado:

```
int[] valores;  
valores = new int[] { 30, 33, 7, 18, 45 };
```

- Tentar inicializar o array sem utilizar a *keyword* `new`, irá originar um erro:

```
int[] valores;  
  
// A seguinte linha de código irá provocar um erro  
valores = { 30, 33, 7, 18, 45 };
```

Nº de elementos de um array: Propriedade `Length`

- Na linguagem C# os arrays são objetos
- Como tal, têm acesso a várias propriedades e métodos
- Uma dessas propriedades, `Length`, retorna o número total de elementos de um array
- Exemplo:

```
string[] cidades = { "Lisboa", "Coimbra", "Viana do Castelo", "Faro" };  
Console.WriteLine("O array 'cidades' tem {0} elementos", cidades.Length);
```

- Resultado:

```
O array 'cidades' tem 4 elementos
```

Percorrer um array sequencialmente

- Muitas vezes existe a necessidade de percorrer um array, seja para ler ou modificar os elementos
- Para tal utiliza-se normalmente uma instrução repetitiva (`while`, `do...while`, `for`, `foreach`)

Percorrer um array sequencialmente

- Exemplo 1: percorrer um array utilizando a instrução `while`

```
int i = 0;
while (i < cidades.Length)
{
    Console.WriteLine("cidades[{0}] = {1}", i, cidades[i]);
    ++i;
}
```

- Exemplo 2: percorrer um array utilizando a instrução `do...while`

```
i = 0;
do
{
    Console.WriteLine("cidades[{0}] = {1}", i, cidades[i]);
    ++i;
} while (i < cidades.Length);
```


Percorrer um array sequencialmente

- Exemplo 3: percorrer um array utilizando a instrução `for`

```
for (i = 0; i < cidades.Length; ++i)
{
    Console.WriteLine("cidades[{0}] = {1}", i, cidades[i]);
}
```

- A instrução `foreach` é a instrução recomendada para iterar arrays (e outras estruturas)
- Permite percorrer um array de forma eficiente e sem preocupação de verificar se foi atingido o último elemento
- Exemplo 4: percorrer um array utilizando a instrução `foreach`

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

Arrays como parâmetros de métodos

- Os arrays podem ser passados como parâmetros de métodos
- Para tal a definição do parâmetro deve ser semelhante a:
- Sintaxe para declaração de um array:

`método(tipo[] nome)`

- Em que:

`tipo` O tipo de dados do array

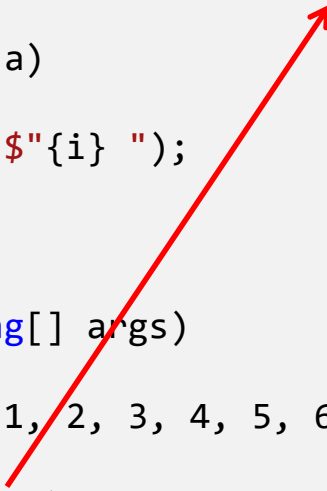
`nome` O nome do parâmetro

Arrays como parâmetros de métodos

- No exemplo seguinte, o método `MostrarArray()` recebe um array unidimensional, do tipo `int`, como parâmetro e mostra o conteúdo do mesmo na consola:

```
public static void MostrarArray(int[] a)
{
    foreach (int i in a)
    {
        Console.Write($"{i} ");
    }
}

static void Main(string[] args)
{
    int[] valores = { 1, 2, 3, 4, 5, 6 };
    MostrarArray(valores);
}
```



Arrays como parâmetros de métodos

- Também é possível declarar, inicializar e passar um array como parâmetro, num único passo:

```
public static void MostrarArray(int[] a)
{
    foreach (int i in a)
    {
        Console.WriteLine($"{i} ");
    }
}

static void Main(string[] args)
{
    MostrarArray(new int[] { 1, 2, 3, 4, 5, 6 });
}
```

Arrays como parâmetros de métodos

- Há que ter em atenção que os arrays são sempre passados por referência pelo que é possível alterá-los dentro de um método
- No seguinte exemplo, o método `MultiplicarArray()` multiplica por 2 cada elemento do array passado por parâmetro:

```
public static void MultiplicarArray(int[] a)
{
    for (int i = 0; i < a.Length; ++i)
    {
        a[i] *= 2;
    }
}

static void Main(string[] args)
{
    int[] valores = { 1, 2, 3, 4, 5, 6 };

    MultiplicarArray(valores);
}
```

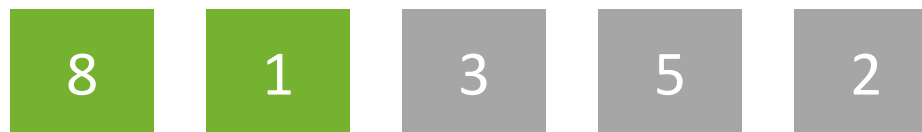
ORDENAÇÃO DE ARRAYS

Ordenação de arrays

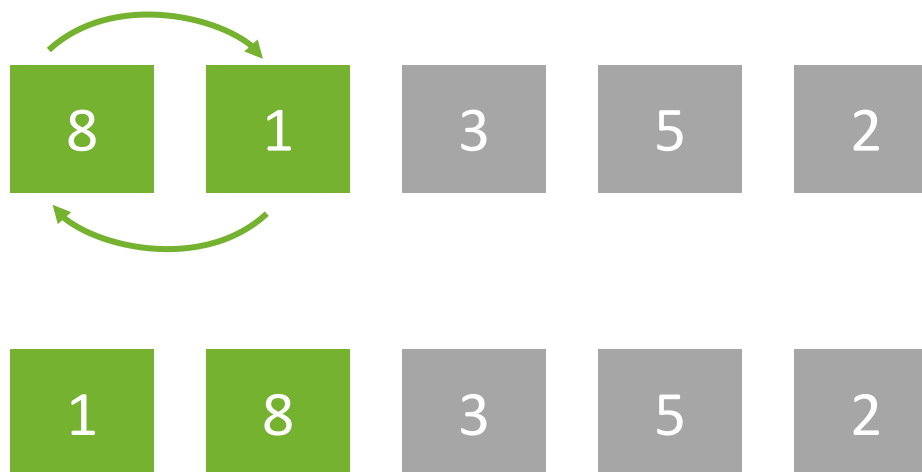
- Um algoritmo de ordenação coloca os dados de uma estrutura numa determinada ordem
- É uma área da informática que vem sendo estudada e trabalhada há já muito
- Existem vários algoritmos para ordenação de dados
- Alguns exemplos:
 - Bubble sort
 - Quicksort
 - Merge sort
 - Insertion sort
 - etc.
- Diferentes algoritmos têm diferentes características:
 - uns são mais ou menos rápidos, outros ocupam mais ou menos memória, outros ainda são adequados para muitos ou poucos dados, etc.
- Ou seja, o tipo de algoritmo a utilizar depende de vários fatores

Ordenação de arrays: Bubble sort

- Neste algoritmo, são comparados pares de elementos
- Se o elemento n é maior do que o elemento $n + 1$:



- Ambos trocam de posição:



Ordenação de arrays: Bubble sort

- O processo continua com o par seguinte:



- É necessário efetuar várias passagens até atingir o resultado final:



- O seguinte vídeo ilustra o funcionamento deste algoritmo:
<https://www.youtube.com/watch?v=nmhjrl-aW5o>

Ordenação de arrays: Bubble sort

- Como implementar este algoritmo?
- Primeiro é preciso saber como trocar dois elementos:



```
// Declarar e inicializar o array
int[] x = { 8, 1, 3, 5, 2 };

// Armazenar o valor de um dos elementos numa variável temporária
int temporario = x[0];

// Trocar
x[0] = x[1];
x[1] = temporario;
```

Ordenação de arrays: Bubble sort

- Depois é necessário percorrer o array desde o início até ao fim:

```
// Declarar e inicializar o array
int[] x = { 8, 1, 3, 5, 2 };

// Percorrer o array
for (int i = 0; i < x.Length; ++i)
{
    // Verificar se o elemento seguinte é maior
    if (x[i] > x[i + 1])
    {
        // Se sim, trocar os elementos
        int temporario = x[i];
        x[i] = x[i + 1];
        x[i + 1] = temporario;
    }
}
```

Ordenação de arrays: Bubble sort

- Mas o código anterior apenas percorre o array uma vez
- É necessário continuar a percorrer até que todos os elementos estejam ordenados

```
// Declarar e inicializar o array
int[] x = { 8, 1, 3, 5, 2 };

// Percorrer o array, desde o último elemento até ao início
for (int j = x.Length - 1; j > 0; --j)
{
    // Percorrer o array, do início até ao fim
    for (int i = 0; i < j; ++i)
    {
        // Verificar se o elemento seguinte é maior
        if (x[i] > x[i + 1])
        {
            // Se sim, trocar os elementos
            int temporario = x[i];
            x[i] = x[i + 1];
            x[i + 1] = temporario;
        }
    }
}
```

Ordenação de arrays: Insertion sort

- O algoritmo bubble sort é bastante ineficiente não se recomendando a sua utilização
- Por sua vez, o algoritmo insertion sort, não sendo dos mais rápidos, constitui uma alternativa simples de efetuar ordenação
- Funciona da seguinte forma:
 - o array é percorrido do início até ao fim, sendo analisado cada elemento
 - se um elemento está fora do lugar (ou seja se é menor do que o elemento anterior) o elemento é trocado com o elemento anterior até que esteja no local apropriado
- O seguinte vídeo ilustra o funcionamento deste algoritmo:
<https://www.youtube.com/watch?v=OGzPmgsl-pQ>

Ordenação de arrays: Insertion sort

- Exemplo do algoritmo insertion sort em C#:

```
// Declarar e inicializar o array
int[] y = { 8, 1, 3, 5, 2 };

// Percorrer o array, desde o segundo elemento até ao fim
for (int i = 1, j; i < y.Length; ++i)
{
    j = i;
    // Se necessário, trocar com o elemento anterior até que esteja na
    // ordem correta
    while ((j > 0) && (y[j - 1] > y[j]))
    {
        int temp = y[j];
        y[j] = y[j - 1];
        y[j - 1] = temp;

        --j;
    }
}
```

Ordenação de arrays

- Analise e execute o código no Moodle, ficheiro [AlgoritmosOrdenação.txt](#), para ver uma comparação da velocidade:
 - do algoritmo bubble sort
 - do algoritmo insertion sort
 - do método [Array.Sort\(\)](#)
- Regra geral a melhor escolha para ordenar arrays é o método [Array.Sort\(\)](#)

| A CLASSE
| ARRAY

A classe `Array`

- A classe `Array` fornece recursos para trabalhar com arrays, facilitando o trabalho do programador
- Destes, os mais importantes são:
 - ordenação de arrays
 - pesquisa de valores
- A classe `Array` está preparada para trabalhar com arrays de diferentes tipos

A classe `Array`: Ordenação de arrays

- A ordenação de arrays é efetuada utilizando o método `Sort()`
- Sintaxe:

`Array.Sort(array)`

A classe `Array`: Ordenação de arrays

- No exemplo seguinte é ordenado um array do tipo `String`:

```
string[] cidades = { "Lisboa", "Almada", "Coimbra", "Aveiro", "Viana do  
Castelo", "Tomar", "Faro" };  
  
Console.WriteLine("Array cidades antes da ordenação:");  
foreach (string cidade in cidades)  
{  
    Console.Write($"{cidade} ");  
}  
  
// Ordenar o array  
Array.Sort(cidades);  
  
Console.WriteLine("\n\nArray cidades após a ordenação:");  
foreach (string cidade in cidades)  
{  
    Console.Write($"{cidade} ");  
}
```

A classe `Array`: Ordenação de arrays

- Resultado:

Array cidades antes da ordenação:

Lisboa Almada Coimbra Aveiro Viana do Castelo Tomar Faro

Array cidades após a ordenação:

Almada Aveiro Coimbra Faro Lisboa Tomar Viana do Castelo

A classe `Array`: Ordenação de arrays

- A ordenação de um array do tipo `int` é igualmente simples:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };

Console.WriteLine("Array valores antes da ordenação: ");
foreach (int valor in valores)
{
    Console.WriteLine($"{valor} ");
}

// Ordenar o array
Array.Sort(valores);

Console.WriteLine("\nArray valores após a ordenação: ");
foreach (int valor in valores)
{
    Console.WriteLine($"{valor} ");
}
```

- Resultado:

```
Array valores antes da ordenação: 30  -33  10  7  18  -9  45  15  0
Array valores após a ordenação: -33  -9  0  7  10  15  18  30  45
```

A classe `Array`: Pesquisa de dados

- A pesquisa de dados é efetuada utilizando o método `BinarySearch()`
- Importa salientar que o array a pesquisar deverá estar ordenado
- Este método retorna:
 - o índice do elemento encontrado
 - ou um número negativo caso o elemento não seja encontrado

- Sintaxe:

```
Array.BinarySearch(array, valor)
```

```
Array.BinarySearch(array, início, comprimento, valor)
```

- Em que:

`array` O array a pesquisar

`valor` O valor a pesquisar

`início` O índice a partir do qual a pesquisa tem início

`comprimento` A quantidade de elementos a pesquisar (a partir do índice `início`)

A classe `Array`: Pesquisa de dados

- No exemplo seguinte, é efetuada uma pesquisa num array do tipo `String`:

```
string[] cidades = { "Lisboa", "Almada", "Coimbra", "Aveiro", "Viana do  
Castelo", "Tomar", "Faro" };  
  
// Ordenar o array  
Array.Sort(cidades);  
  
// O array, ordenado, fica da seguinte forma:  
//   Almada, Aveiro, Coimbra, Faro, Lisboa, Tomar, Viana do Castelo  
  
// Pesquisar valor "Viana do Castelo"  
int i = Array.BinarySearch(cidades, "Viana do Castelo");  
Console.WriteLine($"\\n\\nÍndice do valor \\\"Viana do Castelo\\\": {i}");  
  
// Pesquisar valor "Porto" (não vai ser encontrado)  
i = Array.BinarySearch(cidades, "Porto");  
Console.WriteLine($"\\n\\nÍndice do valor \\\"Porto\\\": {i}");
```

- Resultado:

```
Índice do valor "Viana do Castelo": 6  
Índice do valor "Porto": -6
```

A classe `Array`: Pesquisa de dados

- No exemplo seguinte, é efetuada uma pesquisa apenas num determinado conjunto de valores do array do tipo `int`:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };  
  
// Ordenar o array  
Array.Sort(valores);  
  
// O array, ordenado, fica da seguinte forma:  
//   -33, -9, 0, 7, 10, 15, 18, 30, 45  
  
// Pesquisar valor 10, num intervalo dos índices 1 a 6  
i = Array.BinarySearch(valores, 1, 6, 10);  
Console.WriteLine($"Índice do valor 10: {i}");
```

- Resultado:

```
Índice do valor 10: 4
```


A classe `Array`: Pesquisa de dados

- O método `IndexOf()` permite a pesquisa de dados dentro de um array, sem que esta tenha de estar ordenado
- Retorna o índice da primeira ocorrência de um valor
- Diferentes formatos deste método:

```
Array.IndexOf(array, valor)
```

```
Array.IndexOf(array, valor, início)
```

```
Array.IndexOf(array, valor, início, númeroElementos)
```

- Em que:

`array`

O array a pesquisar

`valor`

O valor a pesquisar

`início`

O índice a partir do qual é efetuada a pesquisa

`númeroElementos`

Número de elementos a pesquisar

A classe `Array`: Pesquisa de dados

- Exemplo 1: pesquisar no array `valores` o valor `-9`

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };  
  
int i = Array.IndexOf(valores, -9); // Devolve o índice 5
```

- Exemplo 2: pesquisar no array `idades` o valor `"Faro"`, a partir do índice `3`

```
string[] cidades = { "Lisboa", "Almada", "Coimbra", "Aveiro", "Viana do  
Castelo", "Tomar", "Faro" };  
  
int i = Array.IndexOf(cidades, "Faro", 3); // Devolve o índice 6
```

- Exemplo 3: pesquisar no array `idades`, entre os índices `2` e `4`, o valor `"Faro"`

```
string[] cidades = { "Lisboa", "Almada", "Coimbra", "Aveiro", "Viana do  
Castelo", "Tomar", "Faro" };  
  
int i = Array.IndexOf(cidades, "Faro", 2, 2); // Devolve o valor -1
```

A classe `Array`: Pesquisa de dados

- O método `LastIndexOf()` retorna o índice da última ocorrência de um valor
- Diferentes formatos deste método:

```
Array.LastIndexOf(array, valor)
```

```
Array.LastIndexOf(array, valor, fim)
```

```
Array.LastIndexOf(array, valor, fim, númeroElementos)
```

- Em que:

`array`

O array a pesquisar

`valor`

O valor a pesquisar

`fim`

O índice a partir do qual é efetuada a pesquisa (em sentido inverso: do fim para o início)

`númeroElementos`

Número de elementos a pesquisar

A classe `Array`: Pesquisa de dados

- Exemplo 1: pesquisar no array `valores2` a última ocorrência do valor `-9`

```
int[] valores2 = { 15, -33, 10, 15, 18, -9, 45, 15, 0 };  
  
int i = Array.LastIndexOf(valores, 15); // Devolve o índice 7
```

- Exemplo 2: pesquisar no array `idades2` a última ocorrência do valor `"Lisboa"`, desde o início até ao índice 4

```
string[] cidades2 = { "Lisboa", "Almada", "Coimbra", "Lisboa", "Viana do  
Castelo", "Tomar", "Lisboa" };  
  
int i = Array.LastIndexOf(cidades2, "Lisboa", 4); // Devolve o índice 3
```

- Exemplo 3: pesquisar no array `idades2`, a última ocorrência do valor `"Lisboa"`, 4 elementos a partir do índice 6

```
string[] cidades2 = { "Lisboa", "Almada", "Coimbra", "Lisboa", "Viana do  
Castelo", "Tomar", "Lisboa" };  
  
int i = Array.LastIndexOf(cidades2, "Lisboa", 6, 4); // Devolve o índice 6
```

Inverter uma string: O método `Reverse()`

- O método `Reverse()` permite inverter a ordem de uma string
- Sintaxe:

`Array.Reverse(array)`

Inverter uma string: O método `Reverse()`

- Exemplo:

```
string[] cidades = { "Lisboa", "Almada", "Coimbra", "Aveiro", "Viana do  
Castelo", "Tomar", "Faro" };  
Console.WriteLine("Array cidades antes da inversão:");  
foreach (string cidade in cidades)  
{  
    Console.Write($"{cidade} ");  
}  
  
Array.Reverse(cidades);  
  
Console.WriteLine("\nArray cidades após inversão:");  
foreach (string cidade in cidades)  
{  
    Console.Write($"{cidade} ");  
}
```

- Resultado:

```
Array cidades antes da inversão:  
Lisboa Almada Coimbra Aveiro Viana do Castelo Tomar Faro  
  
Array cidades após a inversão:  
Faro Tomar Viana do Castelo Aveiro Coimbra Almada Lisboa
```

A tecnologia Linq e arrays

- A tecnologia Linq (na realidade um conjunto de tecnologias) foi desenvolvida pela Microsoft e permite trabalhar de forma fácil e rápida com conjuntos de dados
- Esta tecnologia fornece recursos que facilitam a utilização de arrays (e não só)

Como utilizar

- Para utilizar a tecnologia Linq é necessário importar o namespace `System.Linq`
- Para tal, no início do ficheiro deverá estar presente a diretiva `using System.Linq`

A tecnologia Linq e arrays

Métodos

- Alguns dos métodos disponíveis:

<code>Min()</code>	Retorna o valor mais baixo no array
<code>Max()</code>	Retorna o valor mais alto no array
<code>Sum()</code>	Retorna a soma de todos os elementos do array
<code>Average()</code>	Retorna a média dos elementos do array

- Exemplo:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };

Console.WriteLine($"Valor mínimo: {valores.Min()}");
Console.WriteLine($"Valor máximo: {valores.Max()}");
Console.WriteLine($"Soma: {valores.Sum()}");
Console.WriteLine($"Média: {valores.Average()}");
```


A tecnologia Linq e arrays

- O verdadeiro poder da tecnologia Linq reside na capacidade de criar *querys* (consultas) aos dados
- Por exemplo, para obter todos os números pares no array **valores**:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };
```

```
var pares = (from n in valores  
            where (n % 2 == 0)  
            select n);
```

```
Console.WriteLine("Números pares: ");  
foreach (int i in pares)  
{  
    Console.WriteLine($"{i}, ");  
}
```

A tecnologia Linq e arrays

- Para obter todos os números negativos no array `valores`:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };  
  
var negativos = (from n in valores  
                 where n < 0  
                 select n);  
  
Console.WriteLine("Números negativos: ");  
foreach (int i in negativos)  
{  
    Console.WriteLine($"{i}, ");  
}
```

A tecnologia Linq e arrays

- Também podemos estabelecer condições mais complexas
- Por exemplo, para obter apenas os números entre 0 e 10:

```
int[] valores = { 30, -33, 10, 7, 18, -9, 45, 15, 0 };

// Obter os números entre 0 e 10
var intervalo = (from n in valores
                 where (n >= 0 && n <= 10)
                 select n);

Console.WriteLine("Números entre 0 e 10: ");
foreach (int i in intervalo)
{
    Console.WriteLine($"{i}, ");
}
```

ARRAYS MULTIDIMENSIONAIS

Arrays multidimensionais

- Até agora analisámos arrays unidimensionais, ou seja, arrays simples, com apenas uma dimensão
- No entanto, os arrays podem conter várias dimensões: são os arrays **multidimensionais**
- Enquanto que um array unidimensional é equivalente a uma lista ou vetor, um array bidimensional é equivalente a uma tabela (ou a uma matriz), na qual os elementos se encontram organizados em linhas e colunas.

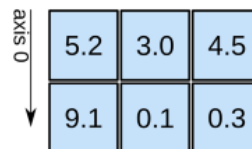
1D array



axis 0 →

shape: (4,)

2D array

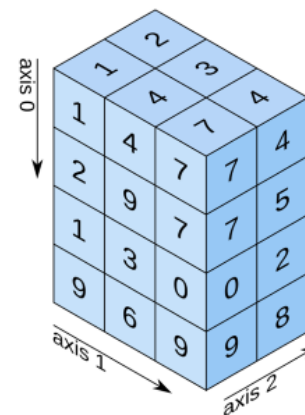


axis 0 ↓

axis 1 →

shape: (2, 3)

3D array



axis 0 ↓

axis 1 →

axis 2 ↗

shape: (4, 3, 2)

Declaração de arrays bidimensionais

- Declaração de um array bidimensional (2 dimensões), com:
 - 2 linhas
 - 3 colunas

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

- Forma alternativa de declarar o array:

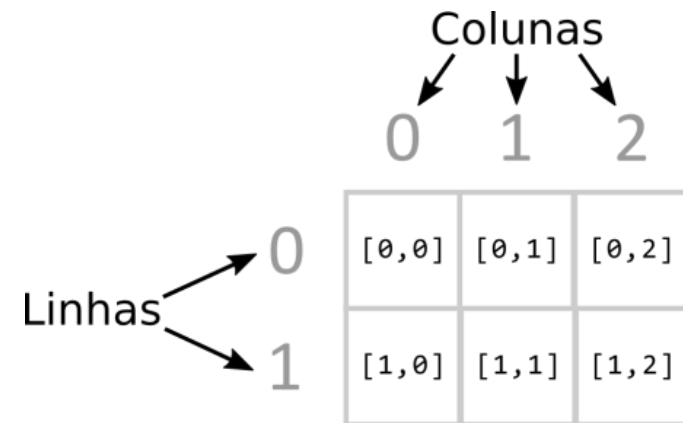
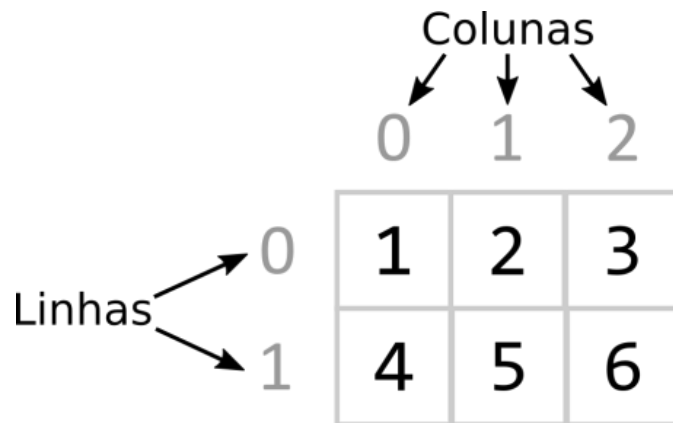
```
int[,] valores = { { 1, 2, 3 }, { 4, 5, 6 } };
```

1	2	3
4	5	6

Índices de elementos

- Cada dimensão tem um índice
- O primeiro elemento, de cada dimensão, tem índice 0
- Tomando como exemplo, o array declarado anteriormente:

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



Arrays bidimensionais

Exercício

- Declare e inicialize um array denominado `temperaturas`, que guarda um conjunto de anos e respetivas temperaturas médias, de acordo com a seguinte tabela:

Ano	Temperatura
1990	30
1991	31
1992	30
1993	33
1994	32
1995	32

Arrays bidimensionais

Exercício: Resolução

- Declare e inicialize um array denominado `temperaturas`, que guarda um conjunto de anos e respetivas temperaturas médias, de acordo com a seguinte tabela:

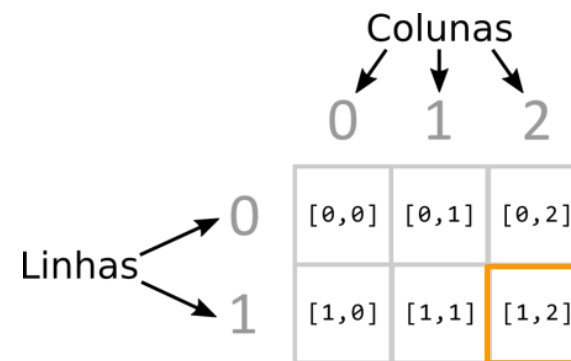
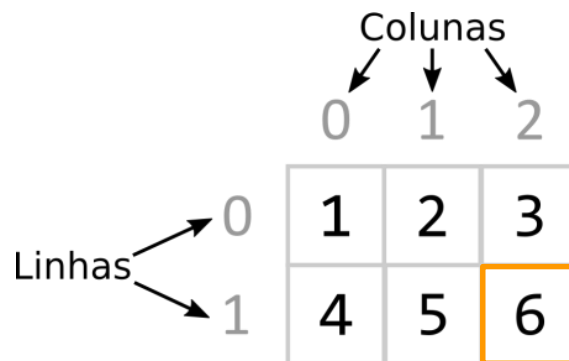
```
int[,] temperaturas = { { 1990, 30 }, { 1991, 31 }, { 1992, 30 }, { 1993, 33 },  
{ 1994, 32 }, { 1995, 32 } };
```

Aceder e modificar os elementos de um array bidimensional

- É possível aceder a cada elemento de um array utilizando a seguinte sintaxe:

`variável[índice linha, índice coluna]`

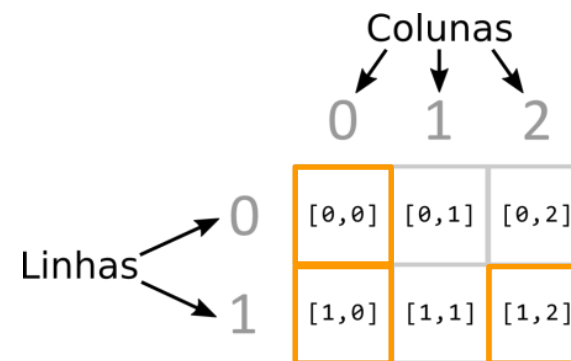
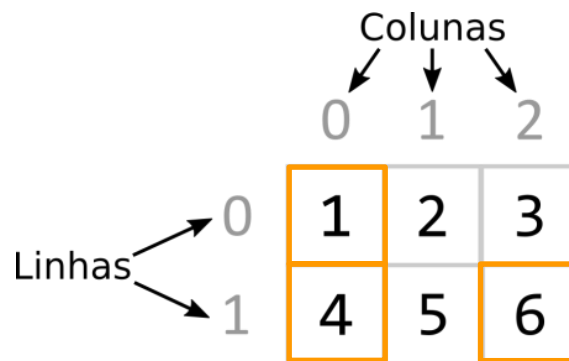
- Exemplo: mostrar o último elemento do array



```
int[,] valores = new int[2, 3] { { 1, 2, 3 }, { 4, 5, 6 } };
Console.WriteLine("{0}", valores[1, 2]);
```

Aceder e modificar os elementos de um array bidimensional

- Exemplo: mostrar a primeira coluna de ambas as linhas do array **valores** e modificar o último elemento do array



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

Console.WriteLine("{0}, {1}", valores[0, 0], valores[1, 0]);

valores[1, 2] = 99;
```

Aceder e modificar os elementos de um array bidimensional

Exercício

- Dado o array `temperaturas`, mostre o ano de 1995 e a respetiva temperatura

Ano	Temperatura
1990	30
1991	31
1992	30
1993	33
1994	32
1995	32

```
int[,] temperaturas = { { 1990, 30 },  
                        { 1991, 31 },  
                        { 1992, 30 },  
                        { 1993, 33 },  
                        { 1994, 32 },  
                        { 1995, 32 } };
```

Aceder e modificar os elementos de um array bidimensional

Exercício

- Dado o array `temperaturas`, mostre o ano de 1995 e a respetiva temperatura

```
int[,] temperaturas = { { 1990, 30 }, { 1991, 31 }, { 1992, 30 }, { 1993, 33 },  
{ 1994, 32 }, { 1995, 32 } };
```

```
Console.WriteLine("Ano de {0} = {1} Celsius", temperaturas[5, 0], temperaturas[5, 1]);
```

Métodos e propriedades

- As seguintes propriedades e método retornam dados úteis para a manipulação de arrays multidimensionais:

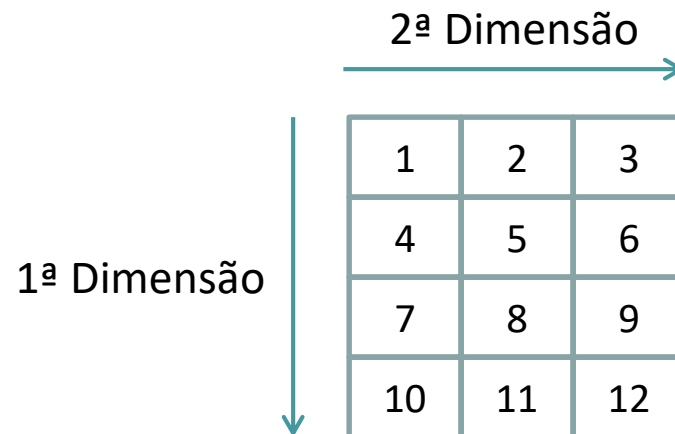
Nome	Tipo	Descrição
<code>Rank</code>	Propriedade	Retorna o número de dimensões de um array
<code>Length</code>	Propriedade	Retorna o número total de elementos de um array
<code>GetLength(int dimensão)</code>	Método	Retorna o número de elementos numa dada dimensão de um array

Métodos e propriedades

- Dado o seguinte array bidimensional, com 4 linhas 3 colunas...
- ... a propriedade **Rank** irá indicar que o array tem 2 dimensões

```
int[,] valores = { { 1, 2, 3 },
                   { 4, 5, 6 },
                   { 7, 8, 9 },
                   { 10, 11, 12 } };

// Propriedade Rank: o array tem 2 dimensões
Console.WriteLine("O array valores tem {0} dimensões", valores.Rank);
```



Métodos e propriedades

- A propriedade `Length` irá indicar que o array tem 12 elementos

```
int[,] valores = { { 1, 2, 3 },
                  { 4, 5, 6 },
                  { 7, 8, 9 },
                  { 10, 11, 12 } };

// Propriedade Length: o array tem 12 elementos
Console.WriteLine("O array valores tem {0} elementos", valores.Length);
```

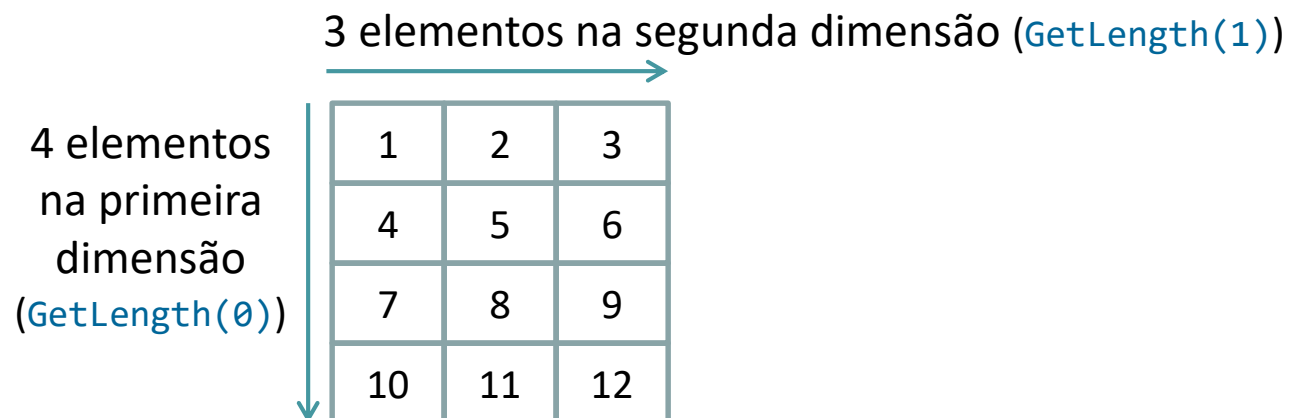
1	2	3
4	5	6
7	8	9
10	11	12

Métodos e propriedades

- O método `GetLength()` irá indicar que o array tem 4 elementos na primeira dimensão e 3 elementos na segunda dimensão

```
int[,] valores = { { 1, 2, 3 },
                   { 4, 5, 6 },
                   { 7, 8, 9 },
                   { 10, 11, 12 } };
```

```
Console.WriteLine("O array valores tem {0} elementos na primeira
dimensão", valores.GetLength(0));
Console.WriteLine("O array valores tem {0} elementos na segunda
dimensão", valores.GetLength(1));
```



Percorrer um array bidimensional sequencialmente

- Exemplo 1: percorrer o array `valores` utilizando a instrução `for`

```
int[,] valores = { { 1, 2, 3 }, { 4, 5, 6 } };  
int i, j;  
  
for (i = 0; i < valores.GetLength(0); ++i)  
{  
    for (j = 0; j < valores.GetLength(1); ++j)  
    {  
        Console.Write("{0} ", valores[i, j]);  
    }  
    Console.WriteLine();  
}
```

- Resultado:

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

Percorrer um array bidimensional sequencialmente

- A instrução `foreach` é mais fácil para percorrer um array multidimensional
- No entanto oferece menos flexibilidade que a instrução `for`
- Exemplo 2: percorrer o array `valores` utilizando a instrução `foreach`

```
int[,] valores = { { 1, 2, 3 }, { 4, 5, 6 } };  
  
foreach (int valor in valores)  
{  
    Console.WriteLine("{0} ", valor);  
}
```

- Resultado:

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