



Programação e Sistemas de Informação

CURSO PROFISSIONAL TÉCNICO DE
GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

Estruturas de Dados Compostas

MÓDULO 5

Professor: João Martiniano

Introdução

- Neste módulo iremos analisar em detalhe dois componentes importantes da linguagem C#:
 - structs
 - enumerações
- A sua utilização num programa, deve resultar de uma reflexão, por parte do programador, sobre qual a melhor forma de organizar e representar os dados

|structs

Introdução

- Uma `struct` é uma estrutura de dados que permite **agrupar variáveis** que estão logicamente relacionadas entre si
- Imaginemos, por exemplo, um programa que necessita de manter informações sobre clientes de uma empresa
- Para cada cliente será necessário guardar os seguintes dados:
 - nome
 - morada
 - idade
 - telemóvel
 - email
- Se o programa não utilizar `structs` será necessário definir as seguintes variáveis:

```
string nome;  
string morada;  
int idade;  
int telemovel;  
string email;
```

Introdução

- Este programa apresenta, pelo menos, dois problemas:
 - apenas é possível guardar informações para um cliente: para guardar informações sobre mais clientes, seria necessário criar arrays para cada variável, o que diminuiria a legibilidade do programa e aumentaria a complexidade
 - não é possível definir mais variáveis com os mesmos nomes: representaria um problema caso também fosse necessário guardar informações semelhantes mas de outro contexto (por exemplo, informações sobre fornecedores ou colaboradores da empresa)
- Neste caso, a utilização de uma `struct` é a solução adequada

Declarar uma `struct`

- Sintaxe para declaração de uma `struct`:

```
struct nome  
{  
    membros  
}
```

- Em que:

`nome` O nome da `struct`

`membros` Os membros da `struct`, nomeadamente: campos, propriedades, construtor, métodos, eventos, etc.

- As `structs` têm algumas semelhanças com as classes
- São consideradas uma versão "light" das classes, partilhando algumas características com estas, mas sendo, comparativamente, mais limitadas
- Em alguns casos, são mais rápidas que as classes
- No entanto, na maior parte das vezes, as classes são a melhor escolha

Declarar uma `struct`

- Pegando no exemplo apresentado anteriormente, as variáveis que armazenam informações sobre um cliente...

```
string nome;  
string morada;  
int idade;  
int telemovel;  
string email;
```

- ... dão origem a uma `struct` destinada a armazenar dados pessoais:

```
public struct DadosPessoais  
{  
    public string Nome;  
    public string Morada;  
    public int Idade;  
    public int Telemovel;  
    public string Email;  
}
```

Instanciar uma `struct`

- Para declarar instâncias de uma `struct`, basta declarar novas variáveis com o tipo da `struct`:
- No seguinte exemplo é declarada uma variável chamada `cliente1` do tipo `DadosPessoais`:

```
DadosPessoais cliente1;
```


Utilizar uma `struct`

- Para declarar uma `struct` e aceder aos respetivos membros utiliza-se a seguinte sintaxe:

`instância.membro`

- Por exemplo, para utilizar a variável `cliente1`, atribuir dados e mostrá-los na consola:

```
// Atribuir dados à variável cliente1
cliente1.Nome = "Marco Alfredo Sousa";
cliente1.Morada = "Avenida Fernão de Magalhães, 107, 3000-001, Coimbra";
cliente1.Idade = 33;
cliente1.Telemove1 = 911234567;
cliente1.Email = "marco.sousa16@gmail.com";

// Mostrar os dados de cliente1 na consola
Console.WriteLine($"Dados do cliente {cliente1.Nome}\n");
Console.WriteLine($"Morada: {cliente1.morada}\nIdade:
{cliente1.idade}\nTelemóvel: {cliente1.telemove1}\nEmail: {cliente1.email}");
```

Utilizar uma struct

- Resultado:

```
Dados do cliente Marco Alfredo Sousa
```

```
Morada: Avenida Fernão de Magalhães, 107, 3000-001, Coimbra
```

```
Idade: 33
```

```
Telemóvel: 911234567
```

```
Email: marco.sousa16@gmail.com
```

Utilizar uma `struct`

- É importante saber que antes de utilizar uma `struct` é necessário atribuir valores a todos os seus membros
- O seguinte código irá provocar um erro: os restantes membros não têm valor atribuído


```
DadosPessoais cliente1;  
  
cliente1.Nome = "Marco Alfredo Sousa";  
  
// A seguinte instrução irá provocar um erro  
Console.WriteLine($"Nome: {cliente1.Nome}");
```

Construtor

- Também pode ser definido um construtor para inicializar uma `struct`
- Exemplo:

```
public struct DadosPessoais
{
    public string Nome;
    public string Morada;
    public int Idade;
    public int Telemovel;
    public string Email;

    public DadosPessoais(string nome, string morada, int idade, int telemovel, string email)
    {
        Nome = nome;
        Morada = morada;
        Idade = idade;
        Telemovel = telemovel;
        Email = email;
    }
}
```



Construtor da `struct`

Construtor

- Para declarar uma instância da `struct` é necessário utilizar a instrução `new`:

```
// Declarar e inicializar a variável cliente2
DadosPessoais cliente2 = new DadosPessoais("Joana Fernandes", "Avenida
Miguel Bombarda, 44, 1º Esq., 8500, Portimão", 47, 961234567,
"fernandesjoana@gmail.com");

// Mostrar os dados de cliente2 na consola
Console.WriteLine($"Dados do cliente {cliente2.Nome}\n");
Console.WriteLine($"Morada: {cliente2.Morada}\nIdade:
{cliente2.Idade}\nTelemóvel: {cliente2.Telemovei}\nEmail: {cliente2.Email}");
```

Construtor

- Também é possível declarar uma `struct` com o *default constructor*
- Neste caso, as propriedades são inicializadas com os valores por defeito do respetivo tipo
- Exemplo:

```
DadosPessoais cliente3 = new DadosPessoais();

Console.WriteLine($"Dados do cliente {cliente3.Nome}\n");
Console.WriteLine($"Morada: {cliente3.Morada}\nIdade:
{cliente3.Idade}\nTelemóvel: {cliente3.Telemovei}\nEmail: {cliente3.Email}");
```

- Resultado:

```
Dados do cliente
```

```
Morada:
```

```
Idade: 0
```

```
Telemóvel: 0
```

```
Email:
```

Métodos

- É possível definir métodos dentro de `structs`
- Um método permite implementar uma operação que está diretamente relacionada com os dados de uma `struct`
- Por exemplo, sabendo que a propriedade `morada` contém os seguintes componentes:

"Rua/Avenida/etc, número, andar, código postal, localidade (cidade/vila/etc.)"

- Exemplo:

"Avenida Miguel Bombarda, 44, 1º Esq., 8500, Portimão"

- Podemos acrescentar um método `ObterLocalidade()` para extrair e retornar a localidade, a partir da propriedade `morada`

Métodos

```
public struct DadosPessoais
{
    public string Nome;
    public string Morada;
    public int Idade;
    public int Telemovel;
    public string Email;

    public DadosPessoais(string nome, string morada, int idade, int telemovel, string email)
    {
        Nome = nome;
        Morada = morada;
        Idade = idade;
        Telemovel = telemovel;
        Email = email;
    }

    /// <summary>
    /// Extrair a localidade (cidade/vila/aldeia/lugar/etc.) a partir da propriedade Morada.
    /// </summary>
    /// <returns>Retorna a localidade.</returns>
    public string ObterLocalidade()
    {
        return Morada.Substring(Morada.LastIndexOf(',') + 1);
    }
}
```

Método ObterLocalidade()

Arrays de `structs`

- Uma forma de utilizar `structs` consiste em declarar arrays
- Continuando com o exemplo já analisado:
 - necessitamos de armazenar informações sobre clientes de uma empresa
 - para tal, iremos declarar um array chamado `Clientes`, com 10 elementos


```
DadosPessoais[] clientes = new DadosPessoais[10];  
  
// Atribuir dados ao primeiro elemento do array  
clientes[0].Nome = "Marco Alfredo Sousa";  
clientes[0].Morada = "Avenida Fernão de Magalhães, 107, 3000-001, Coimbra";  
clientes[0].Idade = 33;  
clientes[0].Telemovel = 911234567;  
clientes[0].Email = "marco.sousa16@gmail.com";
```

structs dentro de structs

- Uma `struct` pode conter campos do tipo de outras `structs`:

```
public struct Coordenada2D
{
    public int X;
    public int Y;
}

public struct Retangulo
{
    public Coordenada2D Coordenadas;
    public int Largura;
    public int Altura;
}
```



```
Retangulo r;
r.Coordenadas.X = 0;
r.Coordenadas.Y = 0;
r.Largura = 300;
r.Altura = 150;
```

structs como parâmetros de métodos

- Um método pode ter como parâmetro uma **struct**:

```
public void MostrarDados(DadosPessoais dados)
{
    Console.WriteLine($"Nome: {dados.Nome}");

    ...
}
```

- Importa saber que as **structs** são passados por valor
- Significa que:
 - o método irá receber uma cópia da struct
 - consoante o tamanho da **struct** poderá ser uma operação lenta
 - os dados da **struct** não podem ser modificados, dentro do método

structs como parâmetros de métodos

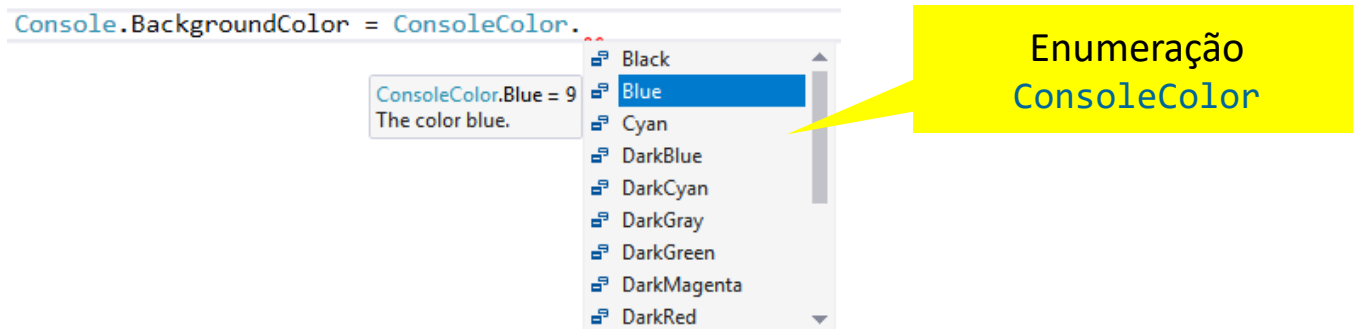
- Para poder modificar uma `struct`, o parâmetro deve ser especificado como sendo de referência:

```
public static void LimparDados(ref DadosPessoais dados)
{
    dados.Nome = string.Empty;
    dados.Morada = string.Empty;
    dados.Idade = 0;
    dados.Telemovei = 0;
    dados.Email = string.Empty;
}
```

|Enumerações

Introdução

- Uma enumeração é um tipo de dados composto por um conjunto de constantes que se traduzem em valores numéricos inteiros
- As duas grandes vantagens na utilização das enumerações são:
 - a nível semântico:
 - o código torna-se mais legível e mais fácil de compreender
 - é fácil perceber o conjunto de valores que uma variável pode aceitar
 - a nível da produtividade do programador:
 - a tecnologia Intellisense reconhece as enumerações e apresenta diretamente ao programador os vários elementos de uma enumeração, sempre que necessário



Declarar uma enumeração

- Sintaxe para definição de uma enumeração:

```
enum nome  
{  
    constantes  
}
```

- Em que:

`nome` O nome da enumeração

`constantes` As constantes definidas pela enumeração

- Por defeito, a lista de constantes tem início no valor 0 (zero)
- Segundo as *naming guidelines* da linguagem C# (definidas pela Microsoft):
 - deve-se atribuir a uma enumeração um nome no singular
 - utilizar PascalCasing

Declarar uma enumeração

- No exemplo seguinte é definida uma enumeração com nomes de cores:

```
public enum Cor
{
    Vermelho,
    Verde,
    Azul
}
```

- No exemplo seguinte é definida uma enumeração para diferentes categorias de produto:

```
public enum CategoriaProduto { Arrumacao, CasaBanho, Canalizacao,
    Cozinha, Decoracao, Ferragens, Ferramentas, Jardim }
```

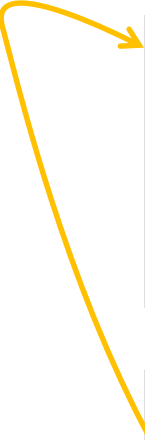
(nota: os elementos da enumeração estão dispostos na horizontal, por uma questão de espaço)

Utilizar uma enumeração

- Para aceder aos membros de uma enumeração, utiliza-se a *dot syntax*:

enumeração.membro

- Exemplo:



```
public enum Cor
{
    Vermelho,
    Verde,
    Azul
}
```

```
// Declarar a variável corProduto como sendo do tipo Cor
Cor corProduto = Cor.Verde;

// Verificar a cor do produto
if (corProduto == Cor.Vermelho)
{
    Console.WriteLine("O produto tem cor vermelha");
}
```

Utilizar uma enumeração

- Cada constante de uma enumeração equivale a um número inteiro
- Por defeito o primeiro elemento tem valor 0 (zero), os restantes incrementam em um, não sendo necessário atribuir um valor numérico de forma explícita:

```
public enum Cor
{
    Vermelho,
    Verde,
    Azul
}
```

equivale a

```
public enum Cor
{
    Vermelho = 0,
    Verde = 1,
    Azul = 2
}
```

- Pode ser atribuído um valor numérico diferente para as constantes de uma enumeração:

```
public enum Cor
{
    Vermelho = 5,
    Verde = 20,
    Azul = 99
}
```

Utilizar uma enumeração

- Caso um elemento de uma enumeração não tenha valor atribuído, assume o valor do elemento anterior, mais uma unidade

```
public enum Cor  
{  
    Vermelho,  
    Verde = 20,  
    Azul  
}
```

Valor: 0

Valor: 21

Utilizar uma enumeração

- Para obter o valor numérico de um membro, este deve ser explicitamente convertido para o tipo base da enumeração:

```
Console.WriteLine((int)Cor.Verde);
```

Utilizar uma enumeração

Exercício

- Crie uma enumeração para guardar os meses do ano
- O primeiro elemento deverá começar com o valor 1

Utilizar uma enumeração

Exercício: Resolução

- Crie uma enumeração para guardar os meses do ano
- O primeiro elemento deverá começar com o valor 1

```
public enum Mes
{
    Janeiro = 1,
    Fevereiro,
    Março,
    Abril,
    Maio,
    Junho,
    Julho,
    Agosto,
    Setembro,
    Outubro,
    Novembro,
    Dezembro
}
```

Utilizar enumerações em `structs`

- As enumerações são mais um tipo de dados possível de ser utilizado em `structs`
- Podemos, por exemplo, adicionar um campo na `struct DadosPessoais`, para armazenar as habilitações:

```
public enum Habilidade {
    EnsinoBasico = 1,
    Secundario,
    Universitario,
    Mestrado,
    Doutoramento
}

public struct DadosPessoais3
{
    public string Nome;
    public string Morada;
    public int Idade;
    public int Telemovel;
    public string Email;
    public Habilidade Habilidade;
}
```