SEMANAS:
- INICIO/APRESENTAÇÃO
- INTRODUÇÃO AOS ALGORITMOS E ESTRUTURAS DE DADOS
- ESTRUTURAS DE CONTROLO DE FLUXO EM C#
- CONCEITO DE CLASSE E OBJETO
- SEPARAÇÃO ENTRE INTERFACE PÚBLICA E IMPLEMENTAÇÃO PRIVADA
- ESTRUTURAS DE DADOS – INTRODUÇÃO
- ANÁLISE DE ALGORITMOS E COMPLEXIDADE
- ALGORITMOS DE ORDENAÇÃO
- ALGORITMOS DE ORDENAÇÃO (PARTE 2)
- PROVA DE AVALIAÇÃO
- LISTAS E FILAS
- TABELAS DE HASH
- TRATAMENTO DE COLISÕES
- ÁRVORES BINÁRIAS
- TIPOS GENÉRICOS E TRATAMENTO DE EXCEPÇÕES
- PROVA DE AVALIAÇÃO

Algoritmos e Estruturas de Dados

CTeSP - Tecnologias e Programação de Sistemas de Informação 2020/21

Docente: Ricardo Henriques.

Sumário:

- Estruturas de controlo de fluxo:
  - Estruturas de controlo de fluxo em C#
    - A cláusula "if then else"
    - A cláusula "if then else"
    - A cláusula "x ? a : b"
    - A cláusula "switch"
    - O ciclo "for"
    - O ciclo "while"
  - Exercício de aplicação e consolidação da matéria.

Estruturas de controlo de fluxo

Ao utilizarmos meios computacionais para resolver problemas humanos, acaba por ser natural que a forma de raciocínio e de decompor os problemas herdem muita da nossa forma de pensar, dado que é a esse nível de abstração que nós (humanos) pensamos.

Dessa forma as seguintes secções mais não são do que a apresentação dessas primitivas de raciocínio em C#. Elas traduzem-se em blocos básicos de raciocínio que expressamos em código para que os sistemas computacionais se comportem de forma semelhante à nossa forma de pensar.

A cláusula "if then else" em C#

A cláusula `if then else`, presente na generalidade das linguagens de programação, traduz a necessidade de expressarmos na linguagem máquina (ou seja de lhe explicarmos) que em determinadas situações é necessário optar por um caminho ou outro.

```
using System;

namespace aplicacao_ifthenelse {
    class Program {
        static void Main(string[] args) {
            int numeroA;
            int numeroB;

            Console.Write("numeroA --> ");
            numeroA = Convert.ToInt32(Console.ReadLine());
            Console.Write("numeroB --> ");
            numeroB = Convert.ToInt32(Console.ReadLine());

            if (numeroA > numeroB)
                Console.WriteLine("O número {0} é maior que o número {1}", numeroA, numeroB);
            else if (numeroA == numeroB)
                Console.WriteLine("O número {0} é igual ao número {1}", numeroA, numeroB);
            else Console.WriteLine("O número {0} é menor que o número {1}", numeroA, numeroB);
        }
    }
}
```

Exemplo: if-then-else

A cláusula "x ? a : b" em C#

É possível, à semelhança de outras linguagens de programação, como C e Java a utilização do operador condicional compacto, porém só é válido em instruções de incrementos, decrementos e atribuições. Veja a sua utilização no exemplo seguinte:

```
using System;

namespace ifTernario {
    class Program {
        static void Main(string[] args) {
            double numeroA;
            double numeroB;
            double menorDosNumeros;

            Console.WriteLine("Use a virgula como separador da parte inteira da decimal");
            Console.Write("número A: ");
            numeroA = Convert.ToDouble(Console.ReadLine());
            Console.Write("número B: ");
            numeroB = Convert.ToDouble(Console.ReadLine());

            menorDosNumeros = numeroA >= numeroB ? numeroA : numeroB;

            Console.WriteLine("O maior dos números é o {0}", menorDosNumeros);
        }
    }
}
```

Exemplo: x ? a : b

A cláusula "switch" em C#

O bloco de instruções switch permite ramificar as operações com base no valor de uma variável e torna-se mais simples de definir do que através da utilização de múltiplos blocos if-then-else. O exemplo seguinte mostra a sua utilização:

```
namespace Switch {
    class Program {
        static void Main(string[] args) {
            Console.Write("Escreva um número de 0 a 10:  ");
            int numero = Convert.ToInt32(Console.ReadLine());

            switch (numero) {
                case 0: Console.WriteLine("zero"); break;
                case 1: Console.WriteLine("um"); break;
                case 2: Console.WriteLine("dois"); break;
                case 3: Console.WriteLine("três"); break;
                case 4: Console.WriteLine("quatro"); break;
                case 5: Console.WriteLine("cinco"); break;
                case 6:
                case 7:
                case 8:
                case 9:
                case 10: Console.WriteLine("é um número de 6 a 10"); break;

                default: Console.WriteLine("oops... esse não sei traduzir!");
                    break;
            }
        }
    }
}
```

Exemplo: switch

Ciclos em C#

Muita computação é inerentemente repetitiva. Os ciclos são estruturas de controlo de fluxo de execução de um programa ditas de repetição. A sua forma usual, independentemente da linguagem em que estão codificadas, dividem-se em dois tipos.

- Iterativas - normalmente tomam a forma de um ciclo em que é aplicada uma transformação sobre elementos de um domínio, terminando quando todos os elementos do domínio foram percorridos.
- Condicionais - tomam a forma de um ciclo em que a transformação aplicada ocorre enquanto/até uma condição se verificar verdadeira/falsa.

O ciclo "foreach"

O ciclo `foreach` pertence ao primeiro tipo de estrutura cíclicas, onde o domínio pode ser vasto, podendo ser uma colecção (incluíndo arrays). As operações internas ao corpo do ciclo irão operar sobre todos os elementos do conjunto.

```
namespace CicloForEach {
    class Program {
        static void Main(string[] args) {
            int[] conjunto = { 2, 3, 20, 34, 71 };
            foreach (int n in conjunto) {
                Console.WriteLine("{0} ao quadrado = {1}", n, n*n);
            }

            string[] nomes = {"Ana", "Beatriz", "Carlos", "Damião",
                              "Estela", "Fábio", "Genaro"};

            foreach (string s in nomes) {
                Console.WriteLine("{0} em maiúscula -> {1}", s, s.ToUpper());
            }
        }
    }
}
```

Exemplo: ciclo "foreach"

O ciclo "for"

O ciclo `for` é um caso particular do `foreach` no qual a enumeração é conhecida e pode ser controlada por meio de uma equação sequencial/exponencial da variável de iteração. O seguinte exemplo constrói a tabuada do número passado como argumento do programa, ou por omissão, a tabuada do 1.

```
namespace Ciclo_For {
    class Program {
        static void Main(string[] args) {
            Console.Write("Qual o número para fazer a tabuada: ");
            double numero = Convert.ToDouble(Console.ReadLine());
            for (int i = 1; i <= 10; i++)
                Console.WriteLine("{0,5} x {1,5} = {2,5}", i, numero, i * numero);
        }
    }
}
```

Exemplo: ciclo "for"

O ciclo "while"

Este tipo de ciclo pode reproduzir o caso anterior, mas está mais vocacionado para controlar uma acção repetitiva enquanto determinada condição for verdadeira. A sua utilização aplica-se sobretudo nas situações em que o controlo da acção é feito com base num pressuposto, por exemplo, a leitura de caracteres ou linhas de um ficheiro enquanto houver mais linhas para receber (not eof).

```
namespace CicloWhile {
    class Program {
        static void Main(string[] args) {
            Console.Write("Dividendo: ");
            int dividendo = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Divisores inteiros de {0}", dividendo);

            int divisor = dividendo;

            while (divisor > 0) {
                if (( dividendo % divisor ) == 0) {
                    Console.WriteLine(divisor);
                }
                divisor--;
            }
        }
    }
}
```

Exemplo: ciclo "while"

O ciclo "do-while"

A diferença do ciclo anterior para este é que se a condição for à partida falsa nem sequer uma iteração é executada. No caso do `do-while` pelo menos uma (a primeira) iteração é (sempre) realizada.

```
namespace CicloDoWhile {
    class Program {
        const int valor = 16;
        static void Main(string[] args) {
            int valorLido;
            Console.WriteLine("Tente descobrir o número inteiro \"mistério\" entre [1..20]");

            do {
                Console.Write("--> ");
                valorLido = Convert.ToInt32(Console.ReadLine());
                if (( valorLido <= 20 ) && ( valorLido >= 1 )) {
                    if (valorLido > valor)
                        Console.WriteLine("...valor por excesso!");
                    else if (valorLido < valor)
                        Console.WriteLine("...valor por defeito!");

                } else {
                    Console.WriteLine("Deverá indicar um valor inteiro entre 1 e 20");
                }
            } while (valor != valorLido);

            Console.WriteLine("Parabéns! O número mistério é o {0}.", valor);
        }
    }
}
```

Exemplo: ciclo "do-while"

Exercício 1 (continuação, enunciado completo):

Suponha que deseja construir uma solução computacional para responder a duas questões:

1. Qual a média de idades da turma?
2. Qual (ou quais) o(s) nome(s) do(s) aluno(s) mais novos da turma?

Sugestão de análise de requisitos:

- Que tipo de informação nos interessa obter em cada questão?
- Será necessário alguma estrutura de dados para lidar com a informação de entrada ou de resposta?

Pense no papel! Implemente em C#.

Métodos e Parâmetros modificadores

Relembre que os métodos (funções) `static` podem ser invocados directamente sem ser necessário criar uma instância de class.

O que é uma instância? Conhece algum exemplo?

Embora a definição de um método seja bastante elementar em C# há um conjunto de *keywords* que podem auxiliar a controlar a forma como os argumentos são passados para o método em questão. Pode consultá-los na seguinte tabela:

Modificador	Significado em C#	Exemplo

nenhum	Se um parâmetro não está marcado com um modificador é assumido tratar-se de uma	`using System;`
		`using System.Collections.Generic;`
		`using System.Text;`
		
		`namespace exemplo_base {`

Modificador	Significado em C#	Exemplo

	passagem de argumento por valor, significando que o método recebe uma cópia dos dados originais.	`class Program {`
		`  static void Main(string[] args) {`
		`  int a = 12, b = 3;`
		`  int res = multiplica(a, b);`
		`  Console.WriteLine(res);`
		`  }`
		
		`  static int multiplica(int x, int y) {`
		`  return x * y;`
		`  }`
		`  }`
		`}`

out	Parâmetros de saída têm de ser atribuídos ao método e como tal são passados como referência. Se o método invocado falhar na atribuição do valor ao parâmetro de saída é gerado um erro de compilação.
	Repare que neste caso não é necessário inicializar as variáveis `multi` e `soma`. Além disso, repare que a passagem de parâmetros de saída permite um método devolver mais do que um resultado. Do ponto de vista formal deixou de ser uma função para ser uma operação.
		`using System;`
		`using System.Collections.Generic;`
		`using System.Text;`
		
		`namespace exemplo_out {`
		`  class Program {`
		`  static void Main(string[] args) {`
		`  int a = 12, b = 3;`
		`  int multi, soma;`
		`  multiplica(a, b, out multi, out soma);`
		`  Console.WriteLine("a * b = {0}", multi);`
		`  Console.WriteLine("a + b = {0}", soma);`
		`  }`
		
		`  static void multiplica(int x, int y,`
		`  out int ans1, out int ans2) {`
		`  ans1 = x * y;`
		`  ans2 = x + y;`
		`  }`
		`  }`
		`}`

ref	O valor é inicialmente definido no método invocador, podendo opcionalmente ser redefinido no método invocado (também aqui se trata de uma passagem por referência). Nenhum erro de compilação é gerado se o método falhar na atribuição do parâmetro em `ref`.
	O uso deste modificador é particularmente interessante quando o método operar
		`using System;`
		`using System.Collections.Generic;`
		`using System.Text;`
		
		`namespace exemplo_ref {`
		`  class Program {`
		`  static void Main(string[] args) {`
		`  string nome1 = "Júlio",`
		`  nome2 = "Sebastião";`
		`  Console.WriteLine("Antes, nome1 = {0},`
		`  nome2 = {1}",`
		`  nome1, nome2);`
		`  swap(ref nome1, ref nome2);`
		`  Console.WriteLine("Depois, nome1 = {0},`
		`  nome2 = {1}",`
		`  nome1, nome2);`
		`  }`
		
		`  private static void swap(ref string nome1,`
		`  ref string nome2) {`
		`  string aux = nome1;`
		`  nome1 = nome2;`
		`  nome2 = aux;`

Modificador	Significado em C#	Exemplo

	(modificar) vários elementos declarados no âmbito de quem a invoca, por exemplo uma função como o *swap*.	`  }`
		`  }`
		`}`

params	Este modificador permite enviar um número variável de parâmetros vistos logicamente como um só. Um método só pode ter um modificador deste tipo e deverá ser sempre o parâmetro final do método (o mais à direita).
	Repare que os elementos passados para o *array* variável de parâmetros são apenas separados por vírgulas, exactamente devido o número de elementos ser variável. Daí também se compreende a necessidade de apenas haver no método um `params` e que seja a variável de argumentos final (mais à direita).
		`using System;`
		`using System.Collections.Generic;`
		`using System.Text;`
		
		`namespace exemplo_params {`
		`  class Program {`
		`  static void Main(string[] args) {`
		`  Console.WriteLine(calculaMedia("notas testes",`
		`  1.2, 10, 12, 4.5));`
		`  }`
		
		`static string calculaMedia(string objectoMedia,`
		`  params double[] numeros) {`
		`  double media = 0;`
		`  foreach(double n in numeros) {`
		`  media += n;`
		`  }`
		`  media /= numeros.Length;`
		`  return string.Format("Média das {0} é de {1}",`
		`  objectoMedia, media);`
		`  }`
		`  }`
		`}`

Exercício 3:

Construa, à custa da operação da adição, a multiplicação entre dois números inteiros positivos.

- Garanta que funciona
- Se ainda não o fez imponha que a operação faça parte de um método, onde se isola a componente de cálculo da componente de interação com o utilizador.

Exercício 4:

Implemente o cálculo do fatorial de um número, sabendo que:

$$\text{fatorial}(n) = \begin{cases} 1, n = 1 \\ n * \text{fatorial}(n-1), n > 1 \end{cases}$$

- Implemente através de uma função iterativa;
- Implemente através de uma função recursiva.

MRH, 2020/21