(Parte I) – Introdução ao C#

Embora se trate de uma nova (recente) linguagem, quem tiver alguma experiência em programação numa outra linguagem, não terá grande dificuldade em atingir o essencial do C#. A dificuldade poderá advir do facto do C# ser um paradigma **Orientado a Objetos** com algumas novidades curiosas.

O C# é uma linguagem de Programação Orientação a Objetos que deriva essencialmente dos princípios que estão por trás da do C, C++ e JAVA. Foi criada para ser suportada pela arquitetura .NET.

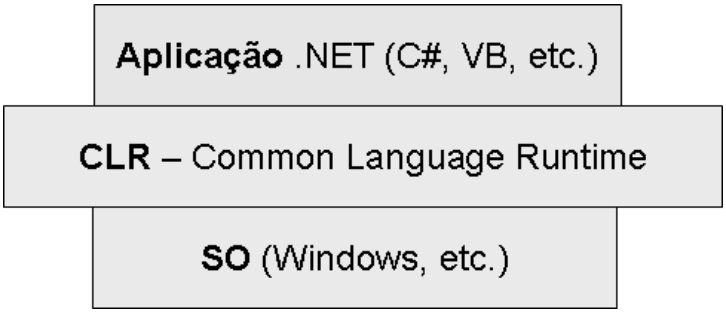


Figura 1 – Arquitectura .NET

Um pequeno exemplo escrito em C# (leia-se c-sharp)

```
using System;
namespace Porto{
 class Exemplo
 {
public static void Main(String[] args)
{
Console.WriteLine("Msg1: " + "O Porto");
Console.WriteLine("Msg2: " + "é o maior");
}
}
```

Código 1 – Exemplo de Programa em C#

Requisitos para programar em C#:

No mínimo é essencial ter instalado o .NET Framework[1]. Não é obrigatório ter instalado o Visual Studio.NET.

Editor para escrever o código no :

- Visual Studio .NET (com C# instalado)

ou

- Visual Studio C# Express

Estrutura base de um programa em C#

```
using System;                          //opcional
namespace nomeNamespace        //opcional
{
class nomeClasse
{
        public static void Main(String[] args)
        {
        }
}

}
```

using	Permite ter acesso a todas as classes definidas no namespace System (embora na realidade não o seja, pode ser visto como os packages do JAVA). Não é obrigatório.
Namespace	Usado para evitar conflito de nomes. Num programa em C# não é possível haver duas classes com o mesmo nome. Um conjunto de classes relacionadas deve ser colocadas no mesmo namespace. Dentro de um namspace podem ser definidos: classes,eventos, exceções, delegates, e namespaces internos. Não é obrigatório.
Class	São conjuntos de dados e métodos que descrevem uma entidade. Num programa C# deve existir pelo menos uma classe que contém um método Main().
Main()	Primeiro método a utilizar na execução do programa. Num mesmo namespace podem existir mais do que um método Main().

Comentários

```
//      isto é um comentário
/*
        Isto também é um comentário
*/
```

Variáveis (Camel Notation) e Tipos de Dados

No C# convencionou-se que:

- Nome das variáveis deve ser feito em **notação Camel**

  Palavras iniciadas com letra minúscula. Restantes palavras iniciadas com letra maiúscula
  Exemplos: (Camel Case) myName, average, bestValue

- Nome dos métodos deve ser feito em **notação Pascal**

  Pascal Notation: Palavras iniciadas com letra maiúscula: Average(), Add(), Main()

Declaração de Variáveis

```
int aux = 20;

long l = 12; short tot;

double myArea = 2.5;

float x = 3.7f
```

Conversão explícita de tipos (cast)

```
int aux = (int) (a+1);

float origem = 3.764f;

int destino = (int) origem;
```

Outros tipos:

```
bool    b;              // true ou false

char    ch ='\n';       //Carcter que representa nova linha "Enter"

string s="Porto";
```

Operações sobre strings:

Concatenação	"luís"+ "ferreira"
Tamanho	nome.Length
Comparação	"lufer"=="lufer"
Parte de String	nome.Substring(0,15)

Operadores

Existem vários tipos de operadores. Abordamos somente os Aritméticos, Relacionais e Lógicos.

Aritméticos

```
int x=2,y=6;

x++;   //incrementa x de uma unidade

x--;   //decrementa x de uma unidade

--x;   //decrementa x de uma unidade

++x;   //incrementa x de uma unidade

 x = x*y;            //multiplicação

x = x-y;             //Subração

x = x+y;             //Multiplicação

x = x/y;             //Divisão

x = x%y;             //Resto da Divisão
```

Em C# podemos escrever as 5 expressões anteriores de uma forma reduzida:

```
x *= y;    x -= y;    x += y;    x /= y;    x %= y;
```

Relacionais

```
if (x == y){ };        //igualdade

if (x != y){ };        //diferença

 if (x > y){ };        //maior

if (x < y){ };         //menor

if (x >= y){ };        //maior ou igual

if (x <= y){};         //menor ou igual
```

Lógicos

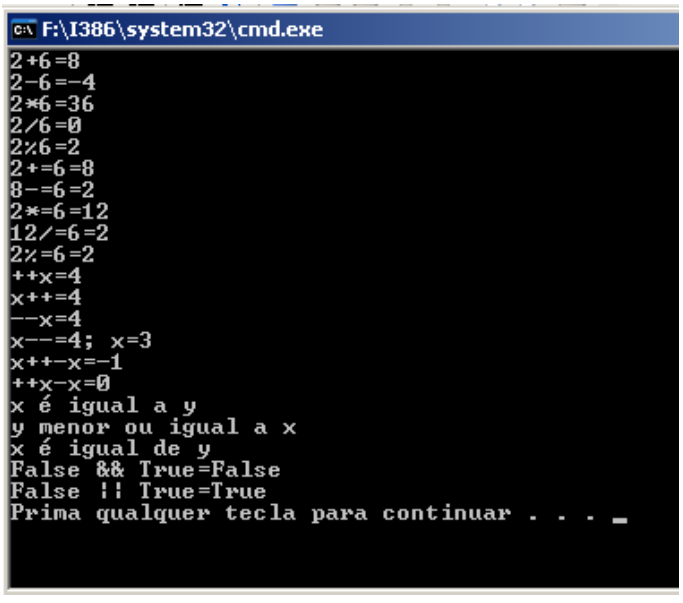
```
bool b=false, c=true;

b=(b && c);          // "and"; b=false

b=(b || c);          // "or"; b=true

b=!b;                          // negação; b=true
```

O exemplo seguinte mostra a aplicabilidade de variáveis, operadores e estruturas de controlo

```
using System;
namespace Operadores
{
/// <summary>
/// Variáveis, Operadores e Estruturas de controlo
/// </summary>
class Program
{
/// <summary>
/// Este programa mostra a utilização de variáveis e operadores
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
        int x=2,y=6;
        int soma, diff, mult, div, rest;

        soma = x + y;
        diff = x - y;
        mult = y * y;
         div = x / y;
        rest = x % y;   //resto da divisão

        //Apresenta o resultado na forma x+y=z
        Console.WriteLine("{0}+{1}={2}", x, y, soma);
         Console.WriteLine("{0}-{1}={2}", x, y, diff);
         Console.WriteLine("{0}*{1}={2}", x, y, mult);
         Console.WriteLine("{0}/{1}={2}", x, y, div);
        Console.WriteLine("{0}%{1}={2}", x, y, rest);
        //Apresenta o resultado na forma x += y
        Console.WriteLine("{0}+={1}={2}", x, y, (x += y));
        Console.WriteLine("{0}-={1}={2}", x, y, (x -= y));
        Console.WriteLine("{0}*={1}={2}", x, y, (x *= y));
        Console.WriteLine("{0}/={1}={2}", x, y, (x /= y));
        Console.WriteLine("{0}%={1}={2}", x, y, (x %= y));

        x = 3;      Console.WriteLine("++x="+ ++x);          //escreve 4; x=4
        Console.WriteLine("x++="+ x++);          //escreve 4; x=5
        Console.WriteLine("--x="+ --x);          //escreve 4; x=4
        Console.WriteLine("x--={0}; x={1}", x--,x);//escreve 4; x=3; Console.WriteLine("x++-x={0}", ((x++)-x)); //escreve -1; x=4
        Console.WriteLine("++x-x={0}", ++x - x); //escreve 0; x=5
```

```
            //operadores relacionais  em if...else
            x = 3; y = x;
            if (x < y)
            {
                        Console.WriteLine("x é menor que y");
            }
            else if (x > y)
                  {
                        Console.WriteLine("y é menor que x");
                  }
                  else
                  {
                        Console.WriteLine("x é igual a y");
                  }
            // operadores relacionais  com o operador " ? ... : "      forma geral c ? a1 : a2
            Console.WriteLine((x < y) ? "x é menor que y" : "y menor ou igual a x");

            //operadores lógicos
            //operadores && (and)  e || (or)
            bool a=false, b=!a;
            Console.WriteLine("{0} && {1}={2}", a, b, a&&b);
            Console.WriteLine("{0} || {1}={2}", a, b, a || b);
}
}
}
```

Resultado da execução do programa



Estruturas de controlo

Condicionais

if-else

```
if (x < y)
{
        Console.WriteLine("x é menor que y");

}

else
    if (x > y)
    {

        Console.WriteLine("y é menor que x");

     }
     else
          {
               Console.WriteLine("x é igual a y");
           }
```

switch

```
int valor=2;

switch (valor)
{
case 1: Console.WriteLine("Um");
                break;
        case 2:
        case 3: Console.WriteLine("Dois ou Três");
                break;
        default: Console.WriteLine("Nenhum deles");
}
```

Ciclos

Todos os ciclos apresentados mostram o resultado: 2,4,6,8

While

```
int k = 2;
while (k<10)
{
        Console.WriteLine("k= {0}", k);
        k+=2;
}
```

do-while

```
int k = 2;
do{
        Console.WriteLine("k= {0}", k);
        k+=2;

}while (k<10)
```

for

Deve ser utilizado sempre que se conhece o número exato de vezes que o ciclo irá ser executado.

```
for (int k = 2; k <= 7; k += 2)
{
        Console.WriteLine("k= {0}", k);
}
```

Como cada uma das seccões do "for" é opcional, é possível utilizar o ciclo for seguinte forma:

```
for(;;)          // ciclo infinito –  não realiza nada - dentro do ciclo

for(;k<=7;k+=2)  // não tem inicialização

for(;k<=7;)      // não tem inicialização e pós execução depois de executado o corpo

for(int k=2;;)   // tem inicialização com ciclo infinito
```

foreach

Utilizado essencialmente para analisar o conteúdo de arrays e outras estruturas de dados.

Será analisado com mais detalhe aquando a abordagem de arrays.

```
int[] elementos={10,20,30,40};

foreach (int i in elementos)

{

        Console.WriteLine("i= {0}", i); // mostra um a um os elementos do vetor elementos

}
```

Controlo dos ciclos: break e continue

Pode acontecer que não seja necessário executar o ciclo no número de vezes previsto, ou seja, o ciclo pode ser interrompido. Pode também interessar alterar a sequência da execução do ciclo. Isto faz-se com as instruções: **break e continue:**

- A instrução **break** pára o ciclo.
- A instrução **continue** força a nova iteração do ciclo.

Analisemos o ciclo definido anteriormente mas alterado com um continue:

```
int k = 2;
while (k<10)
{
        if (k%2==0)
        {
           k++;
           continue;
        }
        Console.WriteLine("k= {0}", k);
        k+=2;
}
```

Neste caso, o resultado apresentado seria 3,5,7,9. Procure analisar porquê!

Agora com break:

```
int k = 2;
while (k<10)
{
        if (k%2==0)
        {
            k++;
            break;
        }
        Console.WriteLine("k= {0}", k); k+=2;
}
```

Neste caso, não apresenta nenhum resultado. Porquê?

Resumo:

- C# é case-sensitive . int é diferente de Int
- Método Main() represente o "arranque" da aplicação
- Espaços, tabuladores e CR são ignorados
- O nome do ficheiro não tem que possuir o mesmo nome que a classe que contém o Main() Podem existir múltiplos métodos Main() num programa
- Blocos de código representam-se com chavetas {}
- A palavra-chave using permite utilizar bibliotecas externas (namespaces)
- O namespace não é obrigatório
- O método Main não necessita de ter argumentos.
- Todas as instruções terminam com ";".
- Existem essencialmente três tipos de operadores: aritméticos, lógicos e condicionais
- As variáveis têm de ser declaradas antes de serem utilizadas
- Existem várias estruturas de controlo:
  Condicionais: if..else e switch;
  Ciclos: for(); do..while, while.
- As estruturas de controlo podem ser controladas com break e continue.

Recomendações CLS:

- Nomear variáveis com notação CamelCase (myName). Começar com letra minúscula.
- Nomear Métodos com notação Pascal (Main)
- Nomes devem começar com caracteres não dígitos (_ é um carácter)
- Nomes de variáveis não devem usar "_"
- Nomes de variáveis não devem diferir somente pelo facto de ser maiúsculas ou minúsculas. Por exemplo, não declarar variáveis myName e MyName.