INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO C#

Objetivos

- Apresentar a descrição da linguagem C#;
- Apresentar as estruturas básicas de controle em C#;
- Apresentar a forma de codificação em linguagem C#;
- Apresentar padrões de mapeamento para a linguagem C#.

Histórico

- 1999 – desenvolvimento da plataforma .NET em Cool, por Anders Hejlsberg e sua equipe;
- 2000 – proposta de padromização feita pela Microsoft, Hewlett-Packard e Intel Corporation;
- 2001 – aprovada a especificação ECMA-334, para a primeira edição;
- 2001 – aprovada a especificação ECMA-334, para a segunda edição;
- 2003 – padronização ISO/IEC 23270;
- 2005 – aprovada a especificação ECMA-334, para a terceira edição.

Descrição da linguagem

- Alfabeto

  Um programa em C# poderá conter os seguintes caracteres:

  - as vinte e seis (26) letras do alfabeto inglês:
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
    a b c d e f g h i j k l m n o p q r s t u v w x y z

  - os dez (10) algarismos:
    0 1 2 3 4 5 6 7 8 9

  - os símbolos:

<	menor	( )	parênteses
>	maior	[ ]	colchete
.	ponto	{ }	chaves
,	vírgula	+	soma
:	dois pontos	-	subtração
;	ponto-e-vírgula	*	asterisco
=	igualdade	/	barra
!	exclamação	#	sustenido
?	interrogação	“	aspas
&	ampersete ("e" comercial)	‘	apóstrofo
^	circunflexo	%	porcento
|	barra em pé	~	til

- Pontuação
  - Ponto-e-vírgula é usado para separar comandos, a menos que outro separador seja necessário;
  - Em alguns casos de operadores, convém o uso de espaços em branco antes, e depois.

- Observação:
  Em C# utilizam-se, ***obrigatoriamente***, as letras minúsculas para os comandos próprios da linguagem.

Tipos de dados

- Tipos básicos

Algoritmo	C#
inteiro	**int**
real	**double**
caractere	**char**
lógico	**bool**

- Outros tipos:

short	inteiros "*curtos*"
long	inteiros "*longos*": [-2.147.483.648,2.147.483.647]
byte	inteiros sem sinal: [0, 65535]
float	reais com precisão simples
decimal	reais para valores monetários (e sem arredondamento)

- Especificação de classe de armazenamento:

const	constante
static	alocação em memória durante a execução
extern	não aloca memória (declaração externa)

Exemplos:

```
const long  ZERO = 0L;
static int  x;
float       i, j, k;
double      a, b, c;
char        letra;
```

- Constantes
  - Constante inteira

    Exemplos:
    10, 532, -10567
    0L, 130000000L, 3241L (inteiro longo)
    0x41, 0xFFFF, 0xA0 (hexadecimal)

  - Constante real

    Exemplos:
    10.465 -5.61 +265. 0.0 .731 .37e+2 -3.e-1

    Observações:
    A vírgula decimal pode ser representada por ponto decimal.

  - Constante literal

    Exemplos:
    Caractere : '1', ' ', '*', 'A', 'a', '?'
    Cadeia : "BRANCO", "CEU AZUL"

    Observações:
    O tamanho da cadeia é limitado.
    As cadeias são terminadas pelo símbolo especial '\0'.

  - Constante lógica

    Exemplos:
    true e **false**

- Caracteres predefinidos:

'\0'	nulo (fim de cadeia de caracteres)
'\n'	passa para a próxima linha
'\t'	passa para a próxima coluna de tabulação (9,17, ...)
'\b'	retorna o cursor uma coluna
'\r'	posiciona o cursor no inicio da linha
'\f'	limpa a tela ou passa para a próxima página
'\\'	barra invertida
'\''	apóstrofo
'\xnn'	representação de um ***byte***, em hexadecimal

- Declaração de constantes

  Formas gerais:

  const <tipo> <NOME> = <valor>;

  Exemplos:
  const int zero = 0;
  const bool V = **true**, F = **false;**
  const double PI = 3.1415926;

- Variáveis
  - Nome de variável

    a) O nome de uma variável tem tamanho determinado;
    b) O primeiro caractere é uma letra ou travessão (_);
    c) Outros caracteres podem ser letra, algarismo ou
       travessão (_).

    Exemplos:
      Nomes válidos: I, a, de, V9a, Lista_Notas

      Nomes inválidos: x+, t.6, 43x, so 5

  - Definição de variáveis
    - Variáveis simples

      Forma geral:

      <tipo 1> <lista de nomes 1>;
      <tipo 2> <lista de nomes 2>;
        ...
      <tipo N> <lista de nomes N>;

      Exemplos:
      char fruta;
      int i, j, k;
      double p, DELTA;

      A definição de variáveis pode ser usada atribuir valores iniciais.

      Exemplos:
      int x = 10,
      y = 20;

- Variáveis agrupadas
  - Homogêneas

    Forma geral:

    ```
    <tipo 1> <lista de nomes 1> [índice];
    <tipo 2> <lista de nomes 2> [índice 1] [índice 2];
      ...
    <tipo N> <lista de nomes N> [índice] ... ;
    ```

    Exemplos:

    ```
    char [ ] frutas = new char [10];
    char [ ] letras = new char [ ] {'a','b','c'};
    string [ ] nomes  = new string [3]
                        {
                         "Alfredo",
                         "Jose",
                         "Mario"
                        };
    int [ ]  v = new int [5] {1,2,3,4,5};
    int [ , ] j = new int[4,4];
    ```

    Observações:
    O primeiro elemento tem índice igual a zero.
    O tipo **string** serve para indicar cadeias de caracteres.

  - Heterogêneas

    Forma geral:

    enum {<lista de valores>} <declaração de nomes>;

    struct <nome> {<campos>} <lista de nomes>;

    Exemplos:

    ```
    enum frutas {banana,laranja,abacaxi};

    frutas fruta = frutas.banana;

    struct pessoa
    {
      char nome, endereco;
      int    rg, cpf, titulo_eleitoral;
    };

    struct pessoa funcionario, operário;
    ```

    Observação:
    O acesso aos campos de uma estrutura ou classe pode ser feito por

    ```
    nome.membro
    ```

- Tipos de operadores
  - Aritméticos

Algoritmo	C#
* / ***mod***	* / %
+ -	+ -

Observações:

O operador ***div*** (divisão inteira) é a própria barra (/), quando os operandos forem inteiros.

Existem formas compactas para incremento e decremento:

<variável inteira>++	pós-incremento
++<variável inteira>	pré-incremento
<variável inteira>--	pós-decremento
--<variável inteira>	pré-decremento

  - Relacionais

Algoritmo	C#
< ≤ > ≥	< <= > >=
= ≠	== !=

Observação:

O resultado de uma comparação de dois valores pode ser falso (**false**) ou verdadeiro (**true**).

  - Lógicos (bit a bit)

Algoritmo	C#
complemento de um	~
e	&
ou-exclusivo	^
ou	|
deslocamento à direita	>>
deslocamento à esquerda	<<

Observação:
O resultado de uma operação lógica é um valor cujos bits são operados um a um de acordo com a álgebra de proposições.

  - Conectivos lógicos

Algoritmo	C#
não	!
e	&&
ou	||

- Literal
  Expressões literais podem ser concatenadas com o sinal (+).

  Exemplo:

  string s = “\n” + “palavra” + “texto com palavras”;

- Prioridade de operadores

Operador	Associação
( ) [ ] ->	à esquerda
! ~ ++ -- + - (tipo) * &	à direita
* / %	à esquerda
+ -	à esquerda
>> <<	à esquerda
< <= >= >	à esquerda
== !=	à esquerda
&	à esquerda
^	à esquerda
| |	à esquerda
&&	à esquerda
|	à esquerda
?:	à direita
= += -= *= /= %=	à direita
>>= <<= &= |= ^=	à direita
,	à esquerda

- Funções intrínsecas

  As regras usadas na formação dos nomes dessas funções intrínsecas são as mesmas utilizadas para os nomes das variáveis.

  Exemplo:

  a = Math.Sin (b);

  a - nome da variável que receberá o resultado da função;
  Math.Sin (x) - função (seno) predefinida do C# ;
  b - nome da variável que vai ser o argumento da função.

Nome (argumento)	Tipo de argumento	Descrição
Math.Sin (X)	**double**	seno (em radianos)
Math.Cos (X)	**double**	cosseno (em radianos)
Math.Atan(X)	**double**	arco tangente
Math.Sqrt(X)	**double**	raiz quadrada
Math.Exp (X)	**double**	exponencial de "e"
Math.Abs (X)	**int**	valor absoluto inteiro
Math.Log (X)	**double**	logaritmo neperiano
Math.Log10 (X)	**double**	logaritmo neperiano
Math.Pow(X,Y)	**double, double**	elevar X a Y

  A linguagem C# dispõe de uma significativa biblioteca básica, com diversas funções além das aritméticas citadas acima.

- Expressões
  - Aritmética

    Exemplos:

Algoritmo	C#
10 + 15	10 + 15
543.15/3	543.15/3
(x + y + z)*a/z	((x + y + z) * a)/z

  - Lógica

    Exemplos:

Algoritmo	C#
A = 0	A == 0
$a \neq 1$	a != 1
$(A \geq 0)$ & $(a \leq 1)$	(A >= 0) && (a <= 1)

    Observação:
    Para efeito de clareza, ou para mudar a precedência de operadores, pode-se separar as proposições por parênteses.

- Estrutura de programa

```
/// definições para documentação automática

// definição de escopo
using <escopo>;
// definições de classe
class <nome da classe>
{
 // declarações de atributos e métodos
   <tipo> <nome1>;
   <tipo> <nome2> (<lista de parâmetros>)
   {
      // definições locais
      // comandos
   } // fim do método

// ação principal
   public static void Main (<lista de parâmetros>)
   {
     // definições locais
     // comandos
   } // fim da ação principal

// declarações de outros métodos (OPCIONAL)
  <tipo> <nome3> (<lista de parâmetros>)
   {
     // definições locais
     // comandos
   }
} // fim da classe
```

- Comentários

  Comentários são precedidos pelos sinais // e /// , ou /* */ envolvendo o texto.

  Exemplo:

```
public static acao1 ( )
{
  // Esta acao nao faz nada - comentario

  /*
    que também pode ser colocado assim
  */
} // fim da ação1
```

- Atribuição

  - Atribuição simples

    Forma geral:

    <variável> = <expressão>;

    Exemplo:

```
x      =  0  ;
a      = 1.57;
letra  =  'A' ;
```

  - Atribuição múltipla

    Forma geral:

    <variável 1> = <variável 2> = <expressão>;

    Exemplo:

```
x = y = 0;
```

    Observação:

    A execução inicia-se pela direita.

  - Atribuição composta

    Forma geral:

    <variável> <operador> = <expressão>;

    Exemplo:

```
i += 1    ou    i = i + 1
```

    Observação:

    Operadores permitidos: + - * / % >> << | & ^

- Atribuição condicional

  Forma geral:

  <variável> = <teste> ? <expressão 1>: <expressão 2>;

  Exemplo:
  x = (a < b) ? a: b;

- Descrição de entrada e saída

  - Entrada/Saída (padrão C#):

    Forma geral:

    <variável> = Console.**Read** ( );
    <variável> = (<tipo>) Console.**ReadLine** ( );
    Console.Write (<expressão>);
    Console.WriteLine (<expressão>);

    Observação:

    É necessário previamente a definição de escopo abaixo (ou similar):

    using System;

    As entradas de dados são normalmente feitas por caracteres ou valores inteiros (ou bytes); e poderão passar por conformações para serem atribuídas aos tipos adequados.

    Exemplos:
    int x = **Console.Read** ( );
    int y = **int.Parse** (**Console.ReadLine** ( ));
    char c = (**char**) **Console.Read** ( );
    string s = **Console.ReadLine** ( );
    double a = **double.Parse** (**Console.ReadLine**);

    As saídas poderão ser convertidas e formatadas (ver adiante) para caracteres mediante uso do método **ToString** ( ) .

    Exemplo:
    Console.WriteLine (x.**ToString** ( ) + “ “ +a.**ToString** (“P”));

  - Saída formatada (padrão C#):

    Forma geral:

    Console.Write (<formato>, <lista de itens>);
    Console.WriteLine (<formato>, <lista de itens>);

    Observação:

    É necessário previamente a definição de escopo abaixo (ou similar):

    using System;

- Especificação de formatos:

  Forma geral:

  {<item><,<largura>><:<especificação>>}

  onde:
  <item> - número do argumento
  <largura> - largura mínima do campo
  <especificação> - pode ser:

caractere	argumento	conversão
C, c	**double**	para valor monetário
D, d	**int**	para inteiro
X, x	**int**	para hexadecimal
E, e	**double**	para real com expoente
F, f	**double**	para real sem expoente
G, g	**double**	para real
P, p	**double**	para porcentagem

Exemplos:

{0}	- X do tipo caractere e com valor igual a 'A'					A
{0:D5}	- X do tipo inteiro e com valor igual a 100	0	0	1	0	0
{0,5:F2}	- X do tipo real e com valor igual a -1	-	1	.	0	0

Observação:

Se a largura (no exemplo, 5) não for suficiente para conter o número na sua forma de representação interna, o tamanho padrão para cada tipo será usado.

- Caracteres com funções especiais em formatos:

caractere	função
\0	fim da cadeia de caracteres
\n	fim de linha (LF)
\t	tabulação
\b	retrocesso (BS)
\r	retorno de carro (CR)
\f	avanço de carro (FF)
\\	barra invertida
\'	apóstrofo
\nnn	representação em octal
\xnn	representação em hexadecimal

Exemplo completo de programa:

```
using System;

class Exemplo
{
   public static void Main ( )
   {
     int i, j;

     Console.Write ( "Exemplo: " );
     Console.WriteLine ( );
     Console.Write "Digite um numero inteiro: ";
     j = int.Parse ( Console.ReadLine ( ) );
     i =   j * 2 + 10;
     Console.WriteLine ( "O resultado e\' igual a " + i );
   } // fim da acao principal
} // fim da classe
```

Se fornecido o valor 5 para a variável ***j*** , o resultado será:

```
O resultado e' igual a 20
```

- Estruturas de controle
  - Sequência simples

    Forma geral:

Algoritmo	C#
<comando>	<comando> ;
<comando>	<comando> ;

    Observação:
    Em C# todos os comandos são separados por ponto-e-vírgula.

- Estrutura alternativa
  - Alternativa simples

    Forma geral:

Algoritmo	C#

se <condição>
então
  <comandos>
fim se
	if (<condição>)
	{
	  <comandos> ;
	}

  - Alternativa dupla

    Forma geral:

Algoritmo	C#
se <condição> então <comandos 1> senão <comandos 2> fim se	**if** (<condição>) { <comandos 1> ; } **else** { <comandos 2> ; }

  - Alternativa múltipla

    Forma geral:

Algoritmo	C#

escolher <valor>
 	
<opção 1>:
  <comandos 1>
 	
<opção 2>:
  <comandos 2>
 	
. . .
<opção n-1>:
  <comandos N-1>
 	
senão
  <comandos N>
 	
fim escolher
	switch <valor>
	{
	  case 1:
	  <comandos 1> ;
	  break;
	  case 2:
	  <comandos 2> ;
	  break;
	. . .
	  case (n-1):
	  <comandos N-1> ;
	  break;
	default:
	  <comandos N>;
	  break;
	}

    Observações:

    A variável de decisão deve ser de tipo escalar.

    A indicação ***default*** é opcional.

- Estrutura repetitiva

  - Repetição com teste no início

    Forma geral:

Algoritmo	C#
repetir enquanto <condição>	**while** (<condição>)
	{
<comandos>	<comandos> ;
fim repetir	}

    Observação:
    A condição para execução é sempre verdadeira.

  - Repetição com teste no início e variação

    Forma geral:

Algoritmo	C#
repetir para	**for**(
<variável> = <valor inicial>	<valor inicial> ;
: <fim>	<teste de fim> ;
: <variação>	<variação>)
	{
<comandos>	<comandos> ;
fim repetir	}

    Observações:
    A condição para execução é sempre verdadeira.
    Em C# , qualquer um dos elementos, ou mesmo todos, podem ser omitidos. Entretanto, se tal for preciso, recomenda-se o uso de outra estrutura mais apropriada.
    Há uma forma própria e simplificada para dados em grupos (**foreach**).

- Repetição com teste no fim

    Forma geral:

Algoritmo	C#
repetir até <condição>	**do**
	{
<comandos>	<comandos> ;
fim repetir	} **while** (<condição>) ;

    Observação:
    A condição para execução é sempre verdadeira.

  - Interrupções

    Em C# , as repetições podem ser interrompidas, em sua sequência normal de execução através dos comandos: **break** e **continue** .
    O comando ***break*** serve para interromper completamente uma repetição, passando o controle ao próximo comando após a estrutura repetitiva.
    O comando ***continue*** interrompe a iteração atual, e segue do início da estrutura repetitiva.