Linguagem de Programação Visual

5º Semestre – Sistemas de Informação

Tipos básicos de dados em C#

C# Type	.Net Framework Type	Signed	Bytes	Possible Values
sbyte	System.Sbyte	Yes	1	-128 to 127
short	System.Int16	Yes	2	-32768 to 32767
int	System.Int32	Yes	4	-2^{31} to $2^{31}-1$
long	System.Int64	Yes	8	-2^{63} to $2^{63}-1$
byte	System.Byte	No	1	0 to 255
ushort	System.Uint16	No	2	0 to 65535
uint	System.Uint32	No	4	0 to $2^{32}-1$
ulong	System.Uint64	No	8	0 to $2^{64}-1$
float	System.Single	Yes	4	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 significant figures
double	System.Double	Yes	8	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15 or 16 significant figures
decimal	System.Decimal	Yes	12	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28 or 29 significant figures
char	System.Char	N/A	2	Any Unicode character
bool	System.Boolean	N/A	1/2	true or false

Tipos básicos de dados em C# (Tipos referência)

Tipo C#	Tipo .NET	Descrição
string	System.String	Uma cadeia de caracteres Unicode **IMUTÁVEL** *(segurança, simplicidade, thread safe)*
object	System.Object	Um objeto genérico (toda classe em C# é subclasse de object) GetType Equals GetHashCode ToString

Restrições e convenções para nomes

- Não começar com dígito: use letra ou _
- Não usar acentos ou til
- Não ter espaço em branco
- Sugestão: use nomes que tenham um significado.

Errado:

```
int 5minutos;
int salário;
int salario do funcionario;
```

Correto:

```
int _5minutos;
int salario;
int salarioDoFuncionario;
```

Restrições e convenções para nomes

- Camel Case: lastName (parâmetros de métodos, variáveis dentro de métodos).
- Pascal Case: LastName (namespace, classe, properties e métodos).
- Padrão _lastName (atributos "internos" da classe).

Operadores aritméticos

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão

- * / % tem precedência maior que + e -.
- Exemplos: 3 + 4 * 2 = 11, (3 + 4) * 2 = 14.

Operadores de atribuição

Operador	Exemplo	Significado
=	a = 10;	a recebe 10
+=	a += 2;	a recebe a + 2
-=	a -= 2;	a recebe a - 2
*=	a *= 2;	a recebe a * 2
/=	a /= 2;	a recebe a / 2
%=	a %= 3;	a recebe a % 3

Operadores aritméticos / atribuição

Operador	Exemplo	Significado
++	a++; ou ++a;	a = a + 1;
--	a--; ou –-a;	a = a – 1;

```
int a = 10;
a++;
Console.WriteLine(a);

SAÍDA:
11
```

```
int a = 10;
int b = a++;
Console.WriteLine(a);
Console.WriteLine(b);

SÁIDA:
11
10
```

```
int a = 10;
int b = ++a;

Console.WriteLine(a);
Console.WriteLine(b);

SAÍDA:
11
11
```

Operadores comparativos

Operador	Significado
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Diferente

Operadores lógicos

Operador	Significado
&&	E
||	OU
!	NÃO

1. Precedência: **!** > **&&** > **||**
2. Usar parêntesis.
3. Exemplos:

2 > 3 || 4 != 5 Resultado: true

!(2>3) && 4 != 5 Resultado: true

C1	C2	C1 E C2
F	F	F
F	V	F
V	F	F
V	V	V

C1	C2	C1 OU C2
F	F	F
F	V	V
V	F	V
V	V	V

Estruturas de controle

- Estrutura de sequência: executa as instruções uma após a outra, na ordem em que aparecem no programa.
- Estrutura de seleção (condição): if, if/else, switch.
- Estrutura de repetição: while, do/while, for e foreach.

If

- A estrutura condicional if identifica qual instrução executar com base no valor de uma expressão booleana.
- Caso a condição seja verdadeira, as instruções serão executadas.
- Sintaxe:

```
if (condição)
{
    instrução;
}
```

If/else

- A estrutura condicional if/else, se a condição for verdadeira, a instrução do if será executada. Se a condição for falsa, a instrução do else será executada.

- Sintaxe:

```
if (condição)
{
    instrução 1;
}
else
{
   instrução 2;
}
```

If/else

- Pode ser constituído por uma única instrução ou várias instruções entre chaves {}.
- Para uma única instrução, as chaves são opcionais, mas recomendadas.

Else if

- Estrutura de decisão condicional aninhada.
- A instrução else if especifica uma nova condição se a primeira for falsa.
- Sintaxe:

```
if (condição 1)
{
    Instrução se a condição 1 é verdadeira;
}
else if (condição 2)
{
    Instrução se a condição 1 é falsa e condição 2 é verdadeira;
}
else
{
   Instrução executada se a condição 1 é falsa e condição 2 é falsa;
}
```

If/else

```
// Simples
if (condition) {
    statement 1;
    statement 2;
}
```

```
//Composta
if (condition) {
    statement 1;
    statement 2;
}
else {
    statement 1;
    statement 2;
}
```

```
// Encadeamentos
if (condition) {
    statement 1;
    statement 2;
}
else if (condition) {
    statement 3;
    statement 4;
}
else if (condition) {
    statement 5;
    statement 6;
}
else {
    statement 7;
    statement 8;
}
```

Else if

```
double media = 8;
if (media >= 7)
{
   Console.WriteLine("Aluno aprovado");
}
else if (media < 7 && media >= 5)
{
   Console.WriteLine("Aluno em recuperação");
}
else
{
  Console.WriteLine("Aluno reprovado");
}
```

Operador ternário

- O operador condicional ternário ?: tem objetivo similar ao if/else mas é codificado em apenas uma linha de código.
- Sintaxe:
- **expressão booleana ? instrução 1 : instrução 2;**
- Caso a expressão retorne verdadeiro, é executado a instrução 1, caso contrário será executado a instrução 2.

Operador ternário

```
double media = 8;

string resultado = "O aluno foi ";

resultado += media >= 7 ? "aprovado" : "reprovado";

Console.WriteLine(resultado);
```

Switch/case

- Switch/case é uma estrutura de condição que define o código a ser executado com base em uma comparação de valores.

- Sintaxe:

```
switch (expressão)
{
    case valor1:
         // bloco de código
         break;
    case valor2:
        // bloco de código
        break;
    default:
        // bloco de código
        break;
}
```

Switch/case

- O comando **break** encerra o bloco switch quando um case é executado.
- Caso não seja declarado, os cases subsequentes serão executados.

Switch/case

- O operador **default** é utilizado para definir um fluxo alternativo para situações em que o valor do switch não seja atendido por nenhum case.
- Caso o valor do switch não seja igual a um dos valores nos **cases**, o código do bloco default será executado.

Switch/case

```
char letra = 'b';
string resposta = " ";
switch(letra) {
      case 'a':
         resposta = "Resposta errada!";
         break;
      case 'b':
         resposta = "Resposta correta!";
         break;
      case 'c':
          resposta = "Resposta errada!";
          break;
      default:
          resposta = "Resposta inválida!";
          break;
}
Console.WriteLine(resposta);
```

While

- Na estrutura de repetição while, antes da execução do laço a condição é testada, enquanto a condição for verdadeira o bloco de código será executado.
- Sintaxe:

```
while (condição)
{
    // bloco de código
}
```

condição: expressão que conterá em que condições o laço será executado.

While

```
int i = 1;
while (i <= 10)
{
        Console.WriteLine(i);
        i++;
}
```

Do/while

- A estrutura de repetição do/while é semelhante ao while, porém, o código dentro do laço é executado pelo menos uma vez.
- A condição é declarada após o bloco de código.
- O do/while inicia a execução do laço, e depois realiza o teste da condição para continuar ou não a execução.
- Sintaxe:

```
do
{
    // bloco de código
} while (condição);
```

Do/while

```
int cont = 0;
do
{
    Console.WriteLine(cont++);
} while (cont <= 10);
```

No while e no do/while deve ser implementada alguma lógica que torne a condição falsa, para não criar um loop infinito.

Break/Continue

- Utilizam-se os comandos break/continue para os laços de repetição for, while e do/while.
- Um loop só é finalizado quando a condição de parada é atendida.
- Com o comando **break** é possível interromper a execução de um laço de repetição.

Break/Continue

```
int i = 1;
while (i <= 10)
{
    if(i == 5)
        break;
    Console.WriteLine(i);
    i++;
}
```

O laço é executado enquanto a variável i for menor que 10. Caso i seja igual a 5 o **break** interrompe o loop.

As instruções seguintes do loop não serão executadas.

Break/Continue

- O comando **continue** também modifica o fluxo de execução do loop.
- Esse comando faz com que a execução do laço seja interrompida e direciona o fluxo de execução para o teste de condição. A condição é novamente avaliada e o laço será executado novamente.

Break/Continue

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

O código deveria ser executado enquanto i fosse menor que 10. Com o **continue** o loop é desviado, executando o while novamente, caso i seja divisível por 2.

As instruções seguintes ao continue não são executadas.

Imprime os valores entre 1 e 10 que não são múltiplos de 2 (ímpares).

For

- A estrutura de repetição for é usada quando sabemos a quantidade de repetições a serem executadas.
- Em sua estrutura:
- declarar e inicializar uma variável de controle da execução do loop;
- a condição para o laço ser executado, enquanto a condição for verdadeira continuará executando;
- incremento ou decremento da variável de controle.

For

- Sintaxe:

```
for (definição/inicialização variável; condição;
incremento/decremento variável)
{
    // bloco de código
}
```

For

```
for (int i = 0; i <= 10; i++)
{
    Console.WriteLine(i);
}
```

int i = 0 define e inicializa a variável que é o contador de quantas vezes o laço será executado.

i <= 10 condição para execução do laço. A condição é analisada antes do início da execução do laço e após cada iteração. Enquanto for verdadeira o laço é executado.

i++ após a execução do laço e se a condição for verdadeira, é incrementado ou decrementado a variável contadora.