



## UNIDADE VI

Linguagem de Programação C#



# ALGORITMO – C Sharp (C#)

## Sumário

ALGORITMO – C Sharp (C#)	1
1 UNIDADE 6 - C Sharp (C#)	3
1.1 Linguagem de programação	3
2 Comandos básicos do c#	3
2.1 Comentários	4
2.2 Indentação	4
2.3 Comando de Saída de Dados	5
2.5 Comando de Entrada de Dados	5
2.5 Exercícios de Fixação	6
3 VARIÁVEIS	7
3.1 Tipos básicos	8
3.2 String	9
3.3 Valores Literais	9
3.3.1 Null	9
3.3.2 Booleanos	9
3.3.3 Inteiros	9
3.3.4 Reais	10
3.3.5 Caracteres	11
3.3.6 Strings literais	11
3.4 Casting	11
3.5 Convenções de nomenclatura	13
3.5.1 Regras de nomenclatura	14
3.6 Keywords	15
3.8 Formatação	16
3.8.1 Formatação de Data e Hora (Conteúdo Extra)	17
3.9 Erros de programação	17
3.9.1 Erros de compilação	18
3.9.3 Erros de lógica	18
3.9.4 Erros comuns no C#	19
4 Exercícios de Fixação	20

## 1 UNIDADE 6 - C SHARP (C#)

Essa unidade irá mostrar uma visão geral do C Sharp ou C#, linguagem de programação que daremos foco no nosso curso preparatório. Dará ênfase nas funções de uma variável, suas convenções de nomenclatura e formatação de saída de dados.

### 1.1 Linguagem de programação

Devido à complexidade, escrever um programa em linguagem de máquina é inviável. Para tornar viável o desenvolvimento de programas, existem as linguagens de programação que tentam se aproximar das linguagens humanas. Confira um trecho de código escrito com a linguagem de programação C#:

```
Condicao = true;
while (Condicao)
{
    Console.WriteLine("Digite o seu primeiro nome:");
    PrimeiroNome = Console.ReadLine();

    if (PrimeiroNome!= "")
    {
        Condicao = false;
    }
    else
    {
        Console.WriteLine("Digite um nome válido!");
    }
}
```

Por enquanto, você não precisa se preocupar em entender o que está escrito no código acima. Apenas, observe que um programa escrito em linguagem de programação é bem mais fácil de ser lido do que um programa escrito em linguagem de máquina.

#### Mais sobre

A maioria das linguagens de programação são *case sensitive*. Isso significa que elas diferenciam as letras maiúsculas das minúsculas. Portanto, ao escrever o código de um programa, devemos tomar cuidado para não trocar uma letra maiúscula por uma letra minúscula ou vice-versa.

## 2 COMANDOS BÁSICOS DO C#

Esse capítulo mostrará os comandos básicos do C# e suas principais funções.

## 2.1 Comentários

Podemos acrescentar comentários no código fonte. Geralmente, eles são utilizados para explicar a lógica do programa. Os compiladores ignoram os comentários inseridos no código fonte. Portanto, no código de máquina gerado pela compilação do código fonte, os comentários não são inseridos.

No C# para comentar uma linha, devemos utilizar a marcação `//`.

```
//string texto = null;
```

E também é possível comentar um bloco com os marcadores `/*` e `*/`.

```
/*Comentário  
   Comentário  
   Comentário  
*/
```

## 2.2 Indentação

A organização do código fonte é fundamental para o entendimento da lógica de um programa. Cada linguagem de programação possui os seus próprios padrões de organização. Observe a organização padrão do código fonte na linguagem C#. É possível utilizar o comando: **CTRL+K+D**.

```
private static void Main(string[] args)  
{  
    string texto = null;  
  
    if (true)  
    {  
        if (true)  
        {  
        }  
    }  
}
```

## 2.3 Comando de Saída de Dados

Vamos escrever o nosso primeiro programa para entendermos como funciona o processo de escrita de código fonte, compilação e execução de um programa.

Você já aprendeu na unidade anterior como se cria um projeto no Visual Studio 2013, então crie uma solução como nome AulasDeCSharp e um projeto em ConsoleApplication como nome de ExemplosUnidade6. Em seguida vamos escrever as seguintes linhas de código no método principal Main. Como o exemplo abaixo:

```
1  using System;
2
3  namespace AulasDeCSharp
4  {
5      internal class Program
6      {
7          private static void Main(string[] args)
8          {
9              Console.WriteLine("Hello Word!");
10             Console.ReadLine();
11         }
12     }
13 }
```

O *Console* pertence à classe *System*, ou seja você deverá chamar o *using.System*; Como mostra a imagem na linha 1. Existe outro comando como *Console.Write*, que a única diferença que é que esse não pula para a próxima linha. Na linha 10 o *Console.ReadLine* está servindo para segurar a tela.

## 2.5 Comando de Entrada de Dados

Agora vamos ver como podemos ler um valor digitado. Esse comando existe algumas particularidades:

```

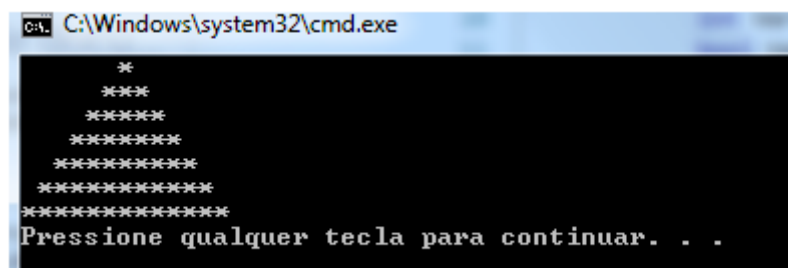
7 private static void Main(string[] args)
8 {
9     double Variavel2 = 0.0;
10    int Variavel3 = 0;
11    bool Variavel4 = false;
12
13    Console.WriteLine("Digite um valor:");
14    string Variavel1 = Console.ReadLine();
15
16    Console.WriteLine("Digite um valor:");
17    Variavel2 = double.Parse(Console.ReadLine());
18
19    Console.WriteLine("Digite um valor:");
20
21    Variavel3 = int.Parse(Console.ReadLine());
22
23
24    Console.WriteLine("Digite um valor:");
25
26    Variavel4 = bool.Parse(Console.ReadLine());
27
28 }

```

O comando de entrada de dados `Console.ReadLine`, interpreta sempre como caracter qualquer valor digitado. Sendo assim, temos que converter sempre para o tipo que queremos ler, como o exemplo acima. A variável local pode ser estanciada como a linha 14 mostra, é bem mais simples e economiza linhas.

## 2.5 Exercícios de Fixação

- 1) Crie um programa que mostre um triangulo sem nenhum comando de asteriscos como exemplo abaixo:



- 2) Agora com o comando “\n” crie o mesmo triangulo:
- 3) Escreva a sigla do CSharp:

```

*****  **  **
*****  **  **
**      *****
**      **  **
**      *****
**      *****
*****  **  **
*****  **  **

```

### 3 VARIÁVEIS

Para criar uma variável em C#, é necessário declará-la. Nessa linguagem de programação, para declarar uma variável é necessário informar o seu tipo e o seu nome (identificador).

No C# devemos informar, no código fonte, o tipo de dado que uma variável poderá armazenar. Por isso, essa linguagem são estaticamente tipadas, ou seja, os tipos das variáveis devem ser definidos em tempo de compilação.

Exemplos de declarações de variáveis locais:

```
11 private static void Main(string[] args)
12 {
13     int Variavel1; //Variável que armazena números inteiros
14     double Variavel2; //Variável que armazena números com vírgula
15     string Variavel3; //Variável que armazena uma cadeia de caracter
16     char Variavel4; //Variável que armazena um caracter
17     bool Variavel5 //Variável que armazena valor lógico
18 }
```

E agora as variáveis globais:

```
7 namespace AulasDeCSharp
8 {
9     0 references
10     class Program1
11     {
12         public static int Variavel1; //Variável que armazena números inteiros
13         public static double Variavel2; //Variável que armazena números com vírgula
14         public static string Variavel3; //Variável que armazena uma cadeia de caracter
15         public static char Variavel4; //Variável que armazena um caracter
16         public static bool Variavel5 //Variável que armazena valor lógico
17
18         0 references
19         private static void Main(string[] args)
20         {
21         }
22     }
```

Notem que elas são 'static'. Porque o método principal Main que é o ponto de entrada da aplicação é por padrão 'static' e todos os dados que ele irá chamar devem ser 'static'.

### 3.1 Tipos básicos

A linguagem C# possui tipos básicos de variáveis. Esses tipos são os mais utilizados e servem como base para a criação de outros tipos. A seguir, veja os tipos básicos da linguagem C#.

Tipo	Descrição	Tamanho (“peso”)
sbyte	Valor inteiro entre -128 e 127 (inclusivo)	1 byte
byte	Valor inteiro entre 0 e 255 (inclusivo)	1 byte
short	Valor inteiro entre -32.768 e 32.767 (inclusivo)	2 bytes
ushort	Valor inteiro entre 0 e 65.535 (inclusivo)	2 bytes
int	Valor inteiro entre -2.147.483.648 e 2.147.483.647 (inclusivo)	4 bytes
uint	Valor inteiro entre 0 e 4.294.967.295 (inclusivo)	4 bytes
long	Valor inteiro entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807 (inclusivo)	8 bytes
ulong	Valor inteiro entre 0 e 18.446.744.073.709.551.615 (inclusivo)	8 bytes
float	Valor com ponto flutuante entre $1,40129846432481707 \times 10^{-45}$ e $3,40282346638528860 \times 10^{38}$ (positivo ou negativo)	4 bytes
double	Valor com ponto flutuante entre $4,94065645841246544 \times 10^{-324}$ e $1,79769313486231570 \times 10^{308}$ (positivo ou negativo)	8 bytes
decimal	Valor com ponto flutuante entre $1,0 \times 10^{-28}$ e $7,9 \times 10^{28}$ (positivo ou negativo)	16 bytes
bool	true ou false	1 bit
char	Um único caractere Unicode de 16 bits. Valor inteiro e positivo entre 0 (ou ‘\u0000’) e 65.535 (ou ‘\uffff’)	2 bytes



## 3.2 String

Na linguagem C#, o tipo *string* é um dos mais importantes e mais utilizados. O tipo *string* é usado para o armazenamento de texto (sequência de caracteres). Observe, nos exemplos abaixo, que o texto que deve ser armazenado nas variáveis é definido dentro de aspas duplas.

```
string texto = "Curso NDDigital!";
```

O espaço utilizado por uma string depende da quantidade de caracteres que ela possui. Cada caractere ocupa 16 Bits. Portanto, a string “Curso NDDigital!” que possui 16 caracteres (o espaço também deve ser contabilizado) ocupa 256 Bits.

## 3.3 Valores Literais

Os valores inseridos diretamente no código fonte são chamados “valores literais”.

### 3.3.1 Null

Os valores inseridos diretamente no código fonte são chamados “valores literais”. Null. O valor nulo é representado pelo literal null em C#. Esse valor não pode ser utilizado para os tipos básicos numéricos e booleanos apresentados anteriormente (lembrando que o tipo char é um tipo numérico).

```
string texto = null;
```

### 3.3.2 Booleanos

Em C#, o valor “verdadeiro” é representado pelo valor literal **true** e o valor “falso” pelo valor literal **false**.

```
bool a = false;  
bool b = true;
```

### 3.3.3 Inteiros

Em C#, números inteiros podem ser definidos apenas em decimal ou hexadecimal.

- Se um número inteiro inicia com 0x ou 0X ele é hexadecimal;
- Caso contrário é decimal.

```
// 10 em decimal
int c = 10;
// 19 em hexadecimal
int d = 0x13;
```

Como vimos, variáveis do tipo **int** não armazenam valores maiores do que 2.147.483.647. Então, considere o valor inteiro 2.147.483.648. Esse valor não pode ser armazenado em variáveis do tipo **int** pois ultrapassa o limite de 2.147.483.647.

Por outro lado, o valor 2.147.483.648 pode ser armazenado em variáveis do tipo **long** já que esse tipo de variável aceita valores até 9.223.372.036.854.775.807.

```
// Vai dar erro
int c = 2147483648;
// Não vai dar erro
int d = 2147483647;
```

### 3.3.4 Reais

Em C#, valores literais reais são definidos com o separador de casas decimais “.”(ponto). Veja alguns exemplos:

```
double a = 19.19;
double b = 0.19;
double c = .19;
```

Por padrão, independentemente da grandeza, os valores literais reais são tratados como **double**. Por exemplo, considere o valor 19.09. Esse valor poderia ser tratado como **float** ou **double**. Contudo, por padrão, ele será tratado como **double**. Dessa forma, os códigos a seguir geram erros de compilação.

```
float a = 19.19;
float b = 0.19;
```

Para resolver esse problema, devemos utilizar o sufixo **F** ou **f**. Ao utilizar um desses sufixos, indicamos ao compilador que o valor literal real deve ser tratado como **float**.

```
float a = 19.19F;
float b = 0.19f;
```

### 3.3.5 Caracteres

Em C#, caracteres literais são definidos dentro de aspas simples. Veja alguns exemplos.

```
char a = 'a';  
char b = 'b';
```

### 3.3.6 Strings literais

Em C#, strings literais são definidas dentro de aspas duplas. Veja alguns exemplos.

```
string a = "Curso NDDigital!";
```

Determinados caracteres são especiais e não podem ser inseridos diretamente dentro das aspas duplas. Por exemplo, os códigos a seguir geram um erro de compilação pois utilizam o caractere especial “\”.

```
string a = "C:\Users\thiago.sartor\Documents\Visual Studio 2013";
```

Para resolvermos esse tipo de problema em C#, podemos utilizar o caractere @ no início das **strings**. Dessa forma, todos os caracteres especiais dentro das aspas duplas serão considerados caracteres normais.

```
string a = @"C:\Users\thiago.sartor\Documents\Visual Studio 2013";
```

## 3.4 Casting

Considere um valor dentro do intervalo de valores do tipo **int**. Esse valor pode ser armazenado em uma variável do tipo **long**, pois todos os valores que estão no intervalo do tipo **int** também estão no intervalo do tipo **long**.

Por causa disso, o C# permite que qualquer valor armazenado em uma variável do tipo **int** possa ser copiado para uma variável do tipo **long**. Veja o exemplo a seguir.

```
int a = 19;  
  
long b = a;
```

Agora, considere um valor dentro do intervalo de valores do tipo **long**. Não podemos garantir que esse valor possa ser armazenado em uma variável do tipo **int** porque o intervalo

do tipo **long** é mais abrangente do que o intervalo do tipo **int**. Por exemplo, o número **2147483648** está no intervalo do tipo **long** mas não está no intervalo do tipo **int**.

Por causa disso, o C# não permite que o valor de uma variável do tipo **long** seja copiado para uma variável do tipo **int**. A tentativa de realizar esse tipo de cópia gera erro de compilação mesmo que o valor armazenado na variável do tipo **long** seja compatível com **int**. Veja o exemplo a seguir.

```
long a = 19;
```

```
int b = a;
```

Nesses casos, podemos aplicar uma operação de conversão também chamada de operação de **casting**. Veja como essa operação é aplicada.

```
long a = 19;
```

```
int b = (int)a;
```

Operações de **casting** podem gerar resultados bem indesejados. Considere que uma variável do tipo **long** armazena o valor **2147483648**. Se uma operação de **casting** for aplicada para copiar esse valor para uma variável do tipo **int** ocorrerá perda de precisão e o valor obtido na cópia será **-2147483648**.

```
long a = 2147483648L;
```

```
int b = (int)a; // b = -2147483648
```

Em geral, quando há o risco de perder precisão, os compiladores exigem a operação de **casting**. Isso funciona como um alerta para o programador. Contudo, em alguns casos, mesmo com o risco de perder precisão, os compiladores não exigem a operação de **casting**. Considere os exemplos a seguir.

```
long a = 9223372036854775807L;
```

```
float b = a; // b = 9223372000000000000
```

Nos exemplos acima, a variável do tipo **long** armazena o valor **9223372036854775807**. Ao copiar o conteúdo dessa variável para uma variável do tipo **float**, há uma perda de precisão e o valor obtido é **9223372000000000000**.

### 3.5 Convenções de nomenclatura

Os nomes das variáveis são fundamentais para o entendimento do código fonte. Considere o exemplo a seguir:

```
int j;  
int f;  
int m;
```

Você consegue deduzir quais dados serão armazenados nas variáveis j, f e m? Provavelmente, não. Vamos melhorar um pouco os nomes dessas variáveis.

```
int jan;  
int fev;  
int mar;
```

Agora, talvez, você tenha uma vaga ideia. Vamos melhorar mais um pouco os nomes dessas variáveis.

```
int janeiro;  
int fevereiro;  
int marco;
```

Agora sim! Você já sabe para que servem essas variáveis? Se você parar para pensar ainda não sabe muita coisa. Então, é importante melhorar mais uma vez o nome dessas variáveis.

```
int numeroDePedidosEmJaneiro;  
int numeroDePedidosEmFevereiro;  
int numeroDePedidosEmMarco;
```

Finalmente, os nomes das variáveis conseguem expressar melhor a intenção delas. Consequentemente, a leitura e o entendimento do código fonte seria mais fácil.

Geralmente, bons nomes de variáveis são compostos por várias palavras como no exemplo a seguir.

```
int numeroDeCandidatosAprovados;
```

Quando o nome de uma variável é composto, é fundamental adotar alguma convenção para identificar o início e o término das palavras. A separação natural das palavras na língua portuguesa são os espaços. Contudo, os nomes das variáveis em C# não podem possuir espaços. Não adotar nenhuma convenção de nomenclatura para identificar o início e o término das palavras é como escrever um texto em português sem espaços entre as palavras. Em alguns casos, o leitor não saberia como separar as palavras. Considere o exemplo abaixo.

salamesadia

O que está escrito no texto acima? A resposta depende da divisão das palavras. Você pode ler como “sala mesa dia” ou “salame sadia”. Dessa forma, fica claro a necessidade deixar visualmente explícito a divisão das palavras.

Em algumas linguagens de programação, delimitadores são utilizados para separar as palavras que formam o nome de uma variável.

```
numero_de_candidatos_aprovados;  
numero-de-candidatos-aprovados;
```

Em outras linguagens de programação, letras maiúsculas e minúsculas são utilizadas para separar as palavras.

```
NumeroDeCandidatosAprovados; Global  
numeroDeCandidatosAprovados; Local
```

Em C#, a convenção de nomenclatura adotada para separar as palavras que formam o nome de uma variável é o CamelCase, que consiste em escrever o nome da variável com a primeira letra de cada palavra em maiúscula com exceção da primeira letra da primeira palavra.

```
int numeroDaConta;  
int NumeroDaConta; // não segue a convenção
```

Também devemos nos lembrar que as duas linguagens são Case Sensitive. Dessa forma, numeroDaConta e NumeroDaConta são consideradas variáveis diferentes pelo fato do nome da primeira começar com letra minúscula e o da segunda com maiúscula.

### 3.5.1 Regras de nomenclatura

A linguagem C# possui regras técnicas muito parecidas a respeito da nomenclatura das variáveis. O nome de uma variável:

1. Não deve começar com um dígito;
2. Não pode ser igual a uma palavra reservada;
3. Não pode conter espaço(s);
4. Pode ser uma palavra de qualquer tamanho;

5. Pode conter letras, dígitos e \_ (underscore).
6. Em Java, pode conter também o caractere \$ ao contrário do C#.

```
// válido
int numeroDaConta;
// inválido pois o nome de uma variável não pode começar com um dígito
int 2outraVariavel;
// inválido pois o nome de uma variável não pode ser igual a uma palavra reservada
double double;
// inválido pois o nome de uma variável não pode conter espaços
double saldo da conta;
// válido
int umaVariavelComUmNomeSuperHiperMegaUltraGigante;
// válido
int numeroDaContaCom8Digitos_semPontos;
// válido somente em Java
int valorDoProdutoEmR$;
// inválido pois o nome de uma variável não pode conter o caractere #
int #telefone;
```

O C# permite a criação de nomes de variáveis em qualquer idioma, pois elas aceitam qualquer caractere Unicode UTF-16. Portanto são válidas as variáveis escritas com as acentuações do português, assim como as variáveis escritas em japonês, por exemplo.

Apesar de ser possível o uso de caracteres especiais, assim como o uso dos caracteres \$ (cifrão) e \_ (underscore), não é recomendável utilizá-los. Não utilizar tais caracteres é uma boa prática de programação. Essa prática facilita a leitura do código fonte em qualquer editor de texto.

### 3.6 Keywords

Toda linguagem de programação possui um conjunto de palavras reservadas. Em geral, essas palavras representam os comandos da linguagem. Abaixo você pode visualizar as palavras reservadas do C#.

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	int	interface	in
internal	is	lock	long	namespace

new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

### 3.8 Formatação

Considere uma variável que armazena o preço de um produto. Geralmente, preços possuem casas decimais. Dessa forma, devemos escolher um tipo que permita o armazenamento de números reais. Por exemplo, podemos escolher o tipo `double` no C#. Veja os exemplos a seguir.

```
System.Random gerador = new System.Random();
double preco = gerador.NextDouble() * 100;
```

Nos exemplos anteriores, os preços dos produtos foram gerados aleatoriamente. Com alta probabilidade, esses valores possuirão mais do que duas casas decimais. Contudo, provavelmente, seria mais conveniente exibir os preços apenas com duas casas decimais. Isso pode ser feito facilmente em C# através das máscaras de formatação.

```
System.Console.WriteLine("{0:F2}", preco);
```

Em C#, os parâmetros são definidos com chaves (“{”}”).

```
System.Console.WriteLine("{0} tem {1:D} anos e pesa {2:F2}", "Jonas", 30, 49.459);
```

No exemplo, o trecho “{0}” indica onde o primeiro parâmetro deve ser inserido. Já o trecho “{1:D}” indica que o segundo parâmetro é um número inteiro. Por fim, o trecho “{2:F2}” indica que o terceiro parâmetro é um número real formatado com duas casas decimais.

**D** ou **d**: número inteiro decimal

**X** ou **x**: número inteiro hexadecimal



**F** ou **f**: número real

### 3.8.1 Formatação de Data e Hora (Conteúdo Extra)

Normalmente, o formato padrão para exibir data e hora varia de país para país ou de região para região. Por exemplo, os brasileiros estão mais acostumados com o formato de data “dia/mês/ano”. Por outro lado, os americanos costumam utilizar o formato “mês/dia/ano”. Em C#, podemos formatar data e hora facilmente. Na máscara de formatação, devemos utilizar os caracteres especiais para definir o formato desejado. Veja o que cada caractere indica.

**d**: dia

**M**:mês

**y**: ano

**H**: hora

**m**: minutos

**s**: segundos

Quando o caractere d é utilizado de forma simples na máscara de formatação, os dias de 1 até 9 são formatados com apenas um dígito. Quando utilizamos dd, os dias de 1 até 9 são formatados com apenas dois dígitos (01, 02, 03, ..., 09). Analogamente, para o mês, ano, hora, minutos e segundos. Agora, veremos a formatação de data e hora no C#. Veja o exemplo a seguir.

```
System.DateTime fundacaoNDD = new System.DateTime(2004, 1, 1, 8, 30, 00);  
string fundacaoNDDFormatada = fundacaoNDD.ToString("dd/MM/yyyy HH:mm:ss"); 3.9
```

### 3.9 Erros de programação

Nesta lição, você aprenderá sobre os diferentes tipos de erros que podem ocorrer ao escrever um programa. Mesmo os programadores mais experientes cometem erros, e saber como depurar um aplicativo e encontrar esses erros é uma parte importante da programação. Antes de saber sobre o processo de depuração, no entanto, isso ajuda a saber os tipos de erros que você precisará localizar e corrigir. Erros de programação se encaixam em três categorias: erros de compilação, erros em tempo de execução e erros de lógica. As técnicas para depurar cada um deles são abordadas nas próximas três lições.

### 3.9.1 Erros de compilação

Erros de compilação, também conhecido como erros de compilador, são erros que impedem seu programa de executar. Quando você pressionar F5 para executar um programa, C# compila o código em um idioma binário que o computador compreende. Se o compilador C# encontra código que ele não entende, ele emite um erro de compilador.

A maioria dos erros de compilador são causados por erros que você faz ao digitar o código. Por exemplo, você pode errar uma palavra-chave, deixar sem algumas pontuações necessárias ou tentar usar uma instrução.

### 3.9.2 Erros de Tempo de Execução

Erros em tempo de execução são erros que ocorrem enquanto o programa é executado. Elas normalmente ocorrem quando o programa tenta uma operação que é impossível executar.

Um exemplo disso é a divisão por zero. Suponha que você tinha a instrução a seguir:

`Speed = Miles / Hours`

Se a variável `Hours` possui um valor de 0, a operação de divisão falha e faz com que ocorra um erro em tempo de execução. O programa deve ser executado de modo a esse erro ser detectado, e se `Hours` contiver um valor válido, ele não ocorrerá.

Quando ocorrer um erro de tempo de execução, você pode usar as ferramentas de depuração no Visual Basic para determinar a causa. Você aprenderá a localizar e corrigir erros em tempo de execução na lição Não funciona! Localizando e eliminando erros em tempo de execução.

### 3.9.3 Erros de lógica

Erros lógicos são erros que impedem seu programa de fazer o que você pretendia fazer. Seu código pode ser compilado e executado sem erros, mas o resultado de uma operação pode produzir um resultado que você não esperava.

Por exemplo, você pode ter uma variável chamada `FirstName` que é inicialmente definida como uma sequência de caracteres em branco. Posteriormente no seu programa, você pode concatenar `FirstName` com outra variável chamada `LastName` para exibir um nome completo. Se você esqueceu atribuir um valor para `FirstName`, apenas o último nome seria exibido, não o nome completo como você pretendia.

Os erros lógicos são o mais difícil localizar e corrigir, mas o C# possui as ferramentas que facilitam, também este trabalho de depuração. Você aprenderá a localizar e corrigir os erros lógicos no O que? Não era para fazer isso! Localizando erros de lógica.

### 3.9.4 Erros comuns no C#

Agora que já sabemos quais são os principais tipos erros. Vamos ver os principais erros que encontramos utilizando o que foi mostrado nessa unidade.

#### **Erro: Variáveis com nomes repetidos:**

Um erro de compilação comum em C# ocorre quando duas ou mais variáveis são declaradas com nome repetido em um mesmo bloco. Veja um exemplo de programa em C# com esse problema.

```
int a = 10;  
int a = 5;
```

A descrição do erro abaixo é seguinte, A variável local chamada 'a' já está definida neste escopo({ }).



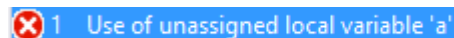
1 A local variable named 'a' is already defined in this scope

#### **Erro: Esquecer a inicialização de uma variável local:**

Outro erro de compilação comum em C# ocorre quando utilizamos uma variável local não inicializada. Veja um exemplo de programa em C# com esse problema.

```
int a;  
Console.WriteLine(a);
```

A descrição do erro abaixo é seguinte, o uso de variável local 'a' não atribuída.



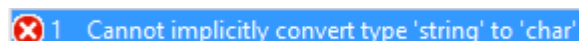
1 Use of unassigned local variable 'a'

#### **Erro: Trocar aspas simples por aspas duplas ou vice-versa:**

Mais um erro comum em C# ocorre quando utilizamos aspas simples onde deveria ser aspas duplas ou vice-versa. Veja um exemplo de programa em C# que utiliza aspas duplas onde deveria ser aspas simples.

```
char c = "A";
```

A descrição do erro abaixo é seguinte, não é possível converter implicitamente o tipo 'string' para 'char'.



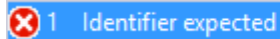
1 Cannot implicitly convert type 'string' to 'char'

#### **Erro: Utilizar o separador decimal errado:**

Outro erro de compilação comum em C# ocorre quando não utilizamos o separador decimal correto. Veja um exemplo de programa em C# com esse problema.

```
double d = 19,09;
```

A descrição do erro abaixo é um pouco genérica, veja abaixo. Interpretasse que ele espera um `'` ao invés de uma `;`:

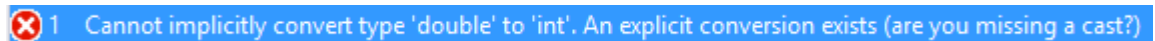
A screenshot of a blue error message box with a red 'X' icon. The text inside reads "1 Identifier expected".

### **Erro: Valores incompatíveis com os tipos das variáveis:**

```
int a = 19.09;
```

Também é um erro de compilação comum em C# atribuir valores incompatíveis com os tipos das variáveis. Veja um exemplo de programa em C# com esse problema.

A descrição do erro abaixo descreve e ajuda o programador a entender, vejam que ele mostra até uma dica. "Você não precisa de um Cast?"(Forçar a ser do tipo int).

A screenshot of a blue error message box with a red 'X' icon. The text inside reads "1 Cannot implicitly convert type 'double' to 'int'. An explicit conversion exists (are you missing a cast?)".

## **4 EXERCÍCIOS DE FIXAÇÃO**

Baixe no repositório do curso: <https://github.com/thiagosartor/CursoNDDigital>

**Bons estudos!**