

## Linguagem de Programação C#

“C Sharp”, é uma linguagem de programação, da Plataforma .NET, derivada de C/C++, simples, moderna e orientada à objetos.

Possui o poder do C/C++ aliado com a alta produtividade do Visual Basic.

É distribuído juntamente com Microsoft Visual Studio.NET, e tem acesso a toda a plataforma do Next Generation Windows Services (NGWS), que incluem uma poderosa biblioteca de classes e um mecanismo de execução comum.

É a linguagem nativa para .NET Common Language Runtime (CLR), mecanismo de execução da plataforma .NET. Isso possibilita a convivência com várias outras linguagens especificadas pela Common Language Subset (CLS).

## Linguagem de Programação C#

Para o desenvolvimento deste material, estaremos utilizando a versão Microsoft Visual C# 2010 Express, que é gratuita para estudo e atenderá aos nossos propósitos.

A Linguagem C# é *sensitive case*, ou seja, faz a diferença de letras minúsculas com maiúsculas.

Por exemplo: prova é diferente de Prova, que é diferente de PrOVa, e assim por diante.

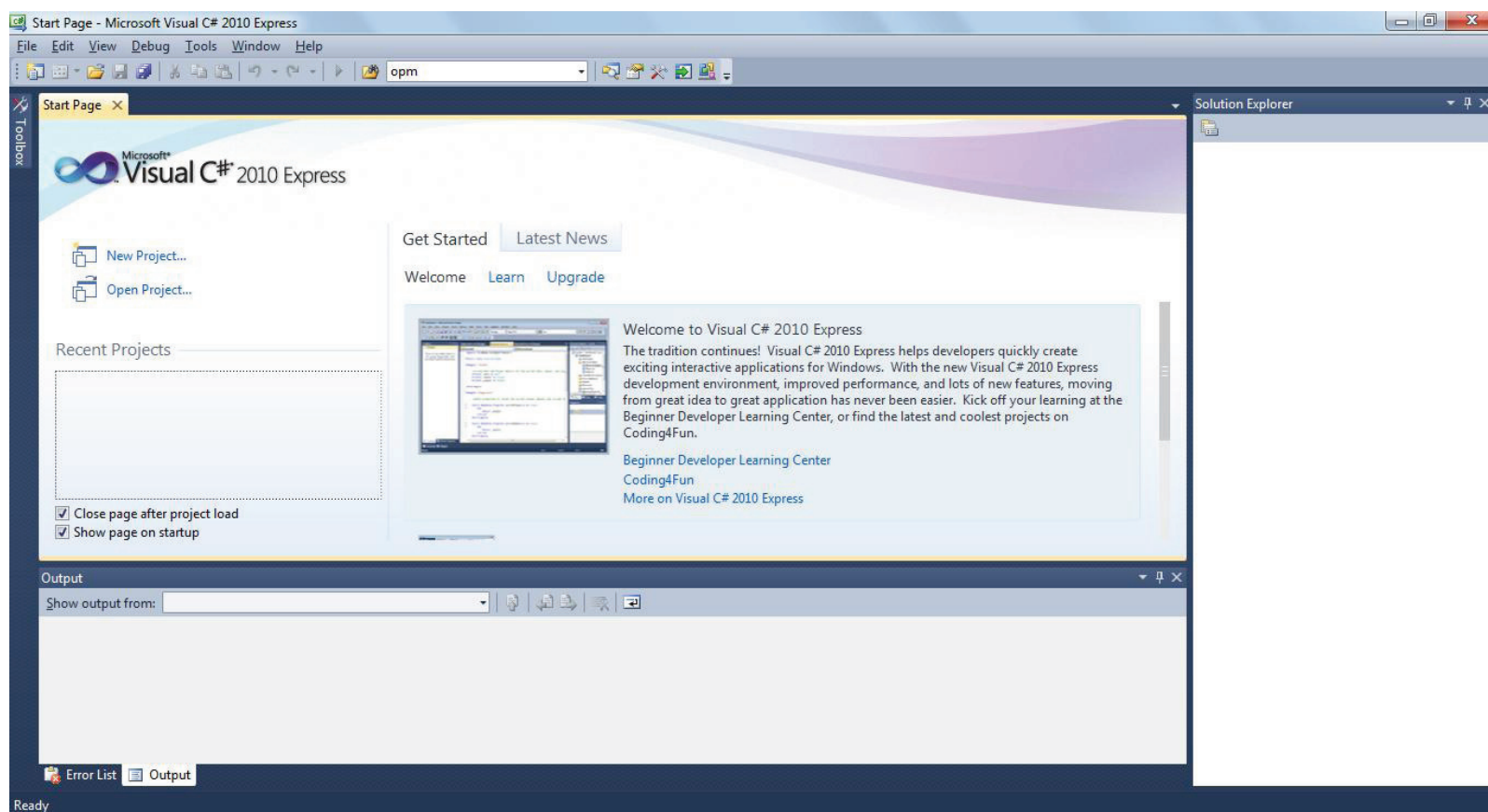
Link para download do programa Microsoft Visual C# 2010 Express:

<http://www.microsoft.com/visualstudio/ptb/downloads#d-2010-express>

Após o download, instale o programa para dar continuidade na aula.

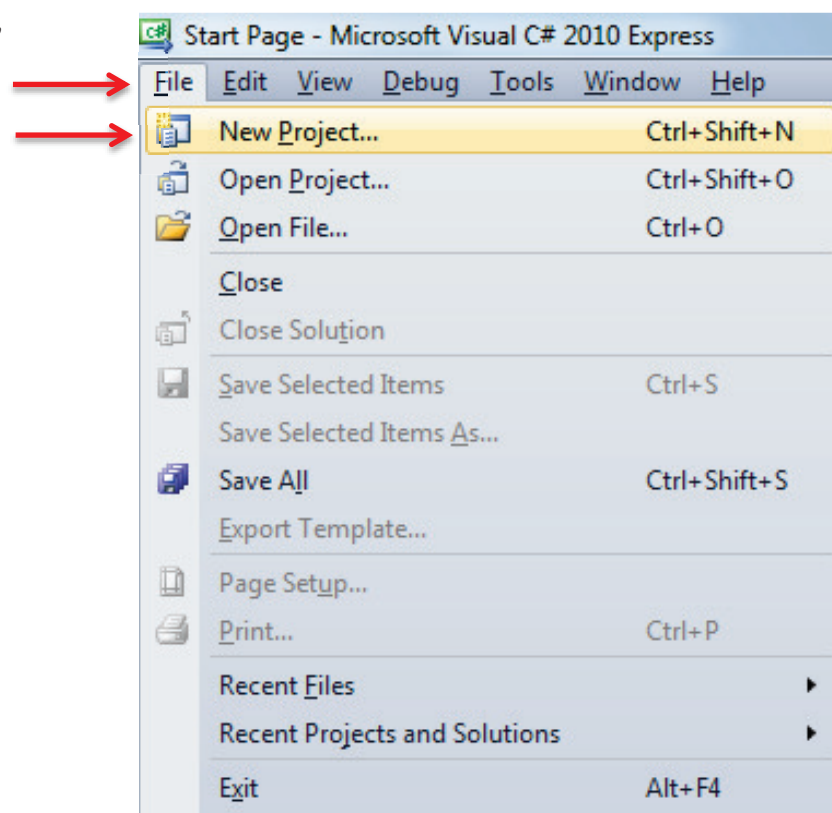
## Tela Inicial do Microsoft Visual C# 2010 Express

Logo após a instalação do programa, ao inicia-lo, será exibida a seguinte tela:



## Criando um projeto

Para iniciarmos o desenvolvimento temos que criar um novo projeto. Para isso, clique em **File → New Project**.

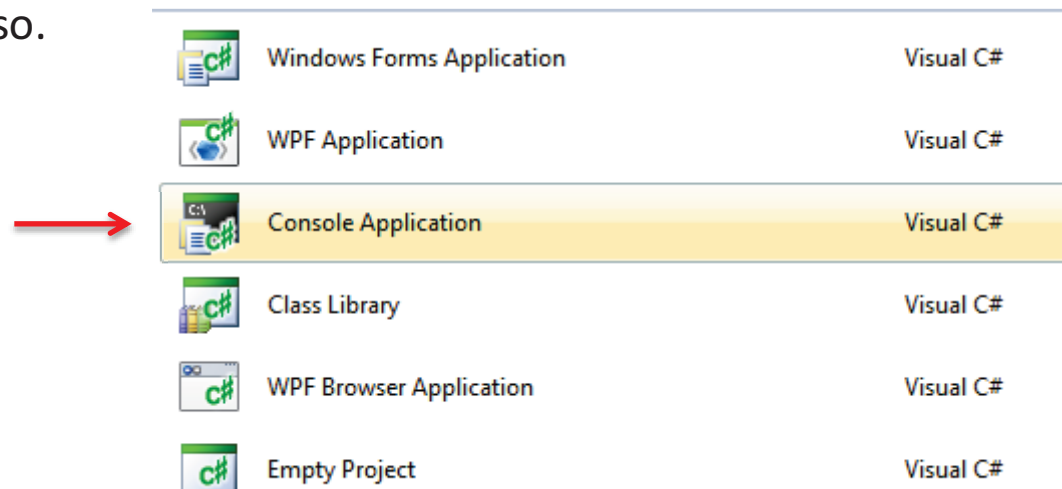


## Criando um projeto

Logo após selecionar a opção de novo projeto, será exibida uma janela para escolher o tipo de projeto que deseja desenvolver.

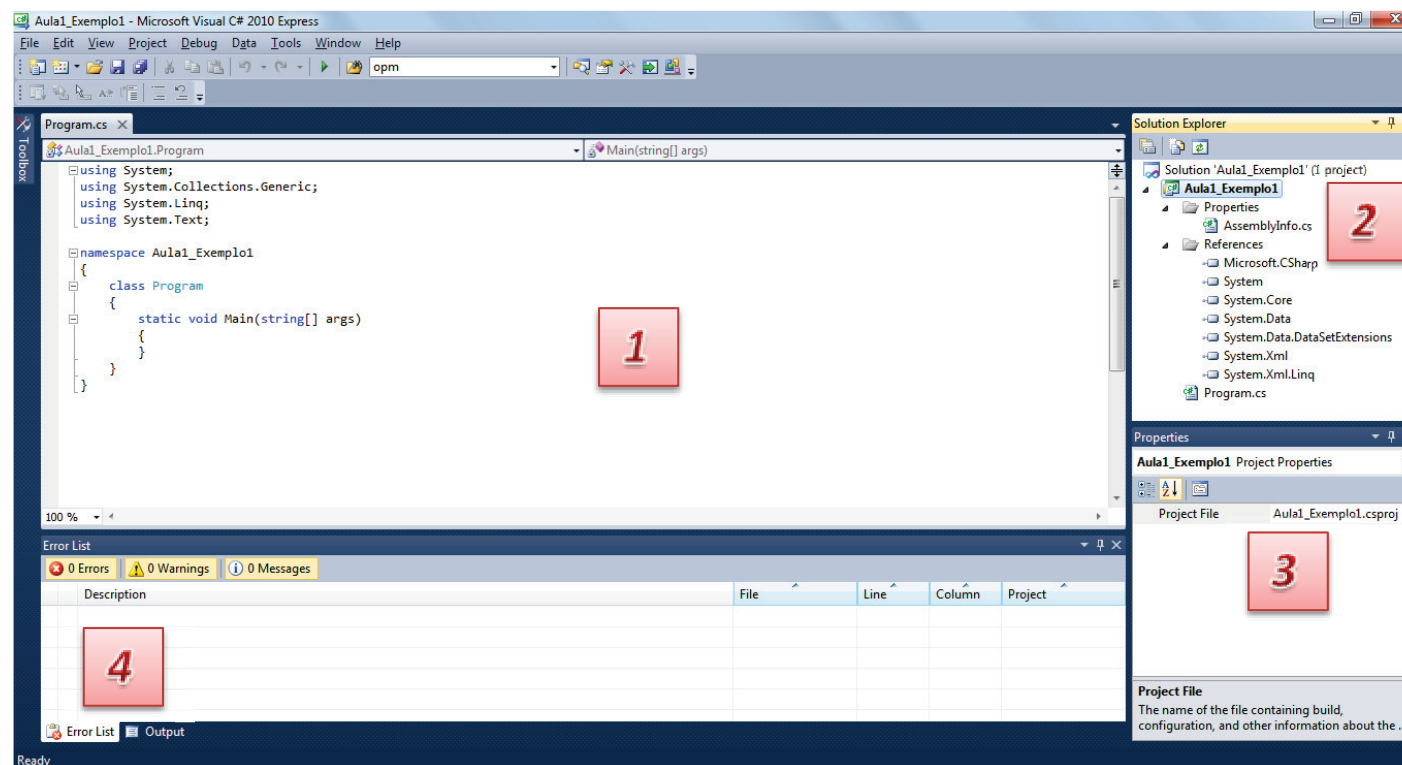
Para nossos estudos neste curso, estaremos utilizando o tipo de projeto para aplicação console. Para isso basta selecionar **Console Application**, colocar um nome para o projeto **Aula1\_Exemplo1** (no lugar de **ConsoleApplication1** na parte inferior da janela) e clicar em **OK**.

Estaremos adotando este padrão aos nomes dos projetos, para facilitar a identificação dos mesmos durante o curso.



## Ambiente para Desenvolvimento

O ambiente então estará preparado para o desenvolvimento do projeto, onde teremos na região central da tela a área de desenvolvimento (1), ao lado direito na tela o ***Solution Explorer*** (2) e ***Properties*** (3), e na região inferior ***Error List*** e ***Output*** (4) .



## Ambiente para Desenvolvimento

Em ***Solution Explorer*** é mostrada a estrutura de pastas e arquivos do projeto, sendo que o arquivo *Program.cs* será o arquivo que estaremos editando para o desenvolvimento do nosso primeiro exemplo.

Em ***Properties*** é possível visualizar e alterar as configurações do projeto, como por exemplo, o nome do projeto, framework, tipo de saída e algumas outras propriedades.

Em ***Error List*** será o local onde aparecerão os erros de compilação que vierem a ocorrer durante o desenvolvimento dos projetos. O processo de compilação tem a função de verificar se o código fonte desenvolvido, possui algum erro. Caso tenha algum erro, o mesmo será listado neste espaço e consequentemente facilitando assim o entendimento do erro para que possamos solucioná-lo.



## Ambiente para Desenvolvimento

Em **Output** será um dos locais possíveis para saída de dados na execução do programa, ou seja, todos os comandos que utilizarmos para exibir informações para os usuários podem ser exibidos neste local. O padrão que estaremos utilizando como saída de dados é uma janela de Console. Para alterar a saída de dados, temos que configurar as propriedades do projeto.

Na região de desenvolvimento do código fonte, quando criamos um projeto com um tipo específico, o software já trás algumas linhas de código já preenchidas. Este código é a estrutura padrão básica para o desenvolvimento de um projeto deste tipo.



## Estrutura básica padrão

**using** serve para carregar as bibliotecas do sistema, ou seja, muitas funções e comandos já estão prontas para serem utilizadas, portanto temos que carregá-las no projeto para que possamos usufruir.

**namespace** é o nome dado ao ambiente de desenvolvimento do projeto. Em um projeto

podemos criar vários arquivos e com isso para cada arquivo criado, utilizaremos o mesmo namespace, facilitando assim a organização do projeto.

**class** diretiva utilizada em linguagem orientada a objetos para identificar o tipo de objeto a ser modelado e o arquivo deverá ter sempre o mesmo nome da classe.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Aula1_Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

## Estrutura básica padrão

**static void main (string[] args)** é o cabeçalho do método principal do projeto, ou seja, no momento da execução do projeto, será executada a sequência lógica de comandos que estiverem escritas a partir do início do desenvolvimento deste método. O nome deste método é **main**.

As chaves “ {} ” servem para delimitar um bloco de instruções, o início e término de métodos, classes, namespaces ou comandos.

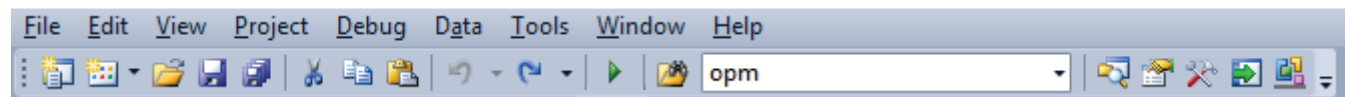
Iremos observar também que o ponto-e-vírgula “ ; ” é fundamental para indicar o final de muitas instruções (comandos). Poderíamos escrever um programa com várias instruções em apenas uma linha, visto que o ponto-e-vírgula faria as delimitações entre as instruções. Mas para um bom entendimento e organização do código fonte, geralmente é usada uma instrução por linha.

## Primeiro exemplo “Hello World”

Para nosso primeiro exemplo, estaremos desenvolvendo um programa que exiba na tela uma mensagem clássica para todos que iniciam um curso de programação, o famoso “Hello World!”. E para isso estaremos acrescentando duas linhas de código dentro do método principal (main). “Dentro” significa estar entre as chaves do método main.

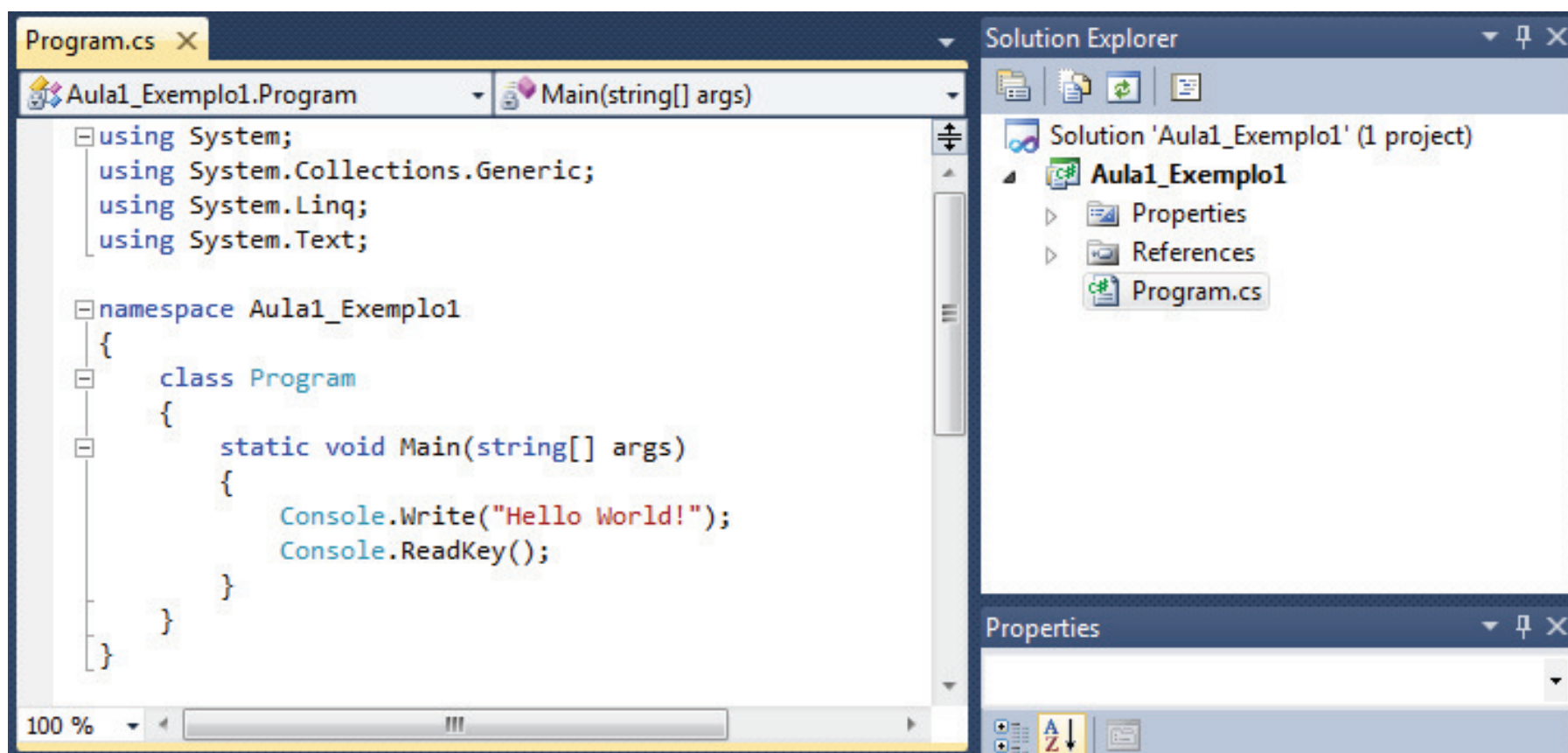
```
Console.WriteLine("Hello World!");  
Console.ReadKey();
```

Depois de incluir estas duas linhas, temos que verificar se o código digitado está correto, e estando correto, executar o programa. Para isso temos que clicar em **Debug → Start Debugging** ou pressionar a tecla F5 ou ainda clicar no símbolo indicado abaixo, na barra de ferramentas.



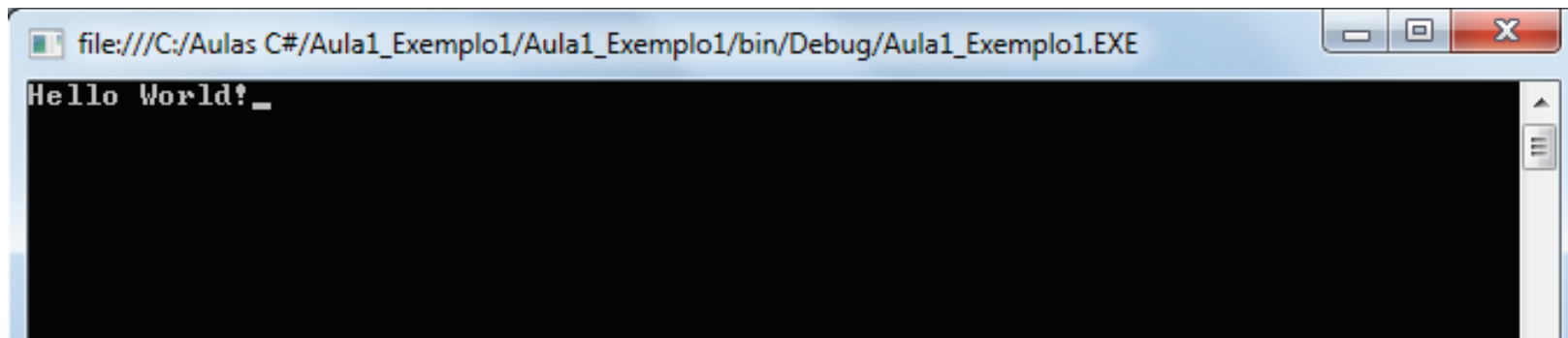
## Primeiro exemplo “Hello World” – Tela

Logo após digitar as instruções indicadas anteriormente, o código fonte fica assim:



## Primeiro exemplo “Hello World” – Resultado

Logo aparecerá uma janela de console, com a mensagem “Hello World!”.



Observe que no título desta janela possui um endereço à um arquivo executável gerado após a execução, ou seja, foi criado o arquivo executável do projeto.

Vá até a pasta indicada e dê um clique no arquivo ***Aula1\_Exemplo1.exe***.

Nota: O arquivo executável é o produto (software) que será disponibilizado para os usuários que utilizarão o programa, e o código fonte não poderá ser disponibilizado.

## Primeiro exemplo “Hello World” – Comandos

Duas instruções foram utilizadas para que obtivéssemos este resultado, que foram:

***Console.Write(“Hello World!”) ;***

Responsável por exibir uma mensagem da tela.

***Console.ReadKey();***

Responsável por pausar a execução do programa.

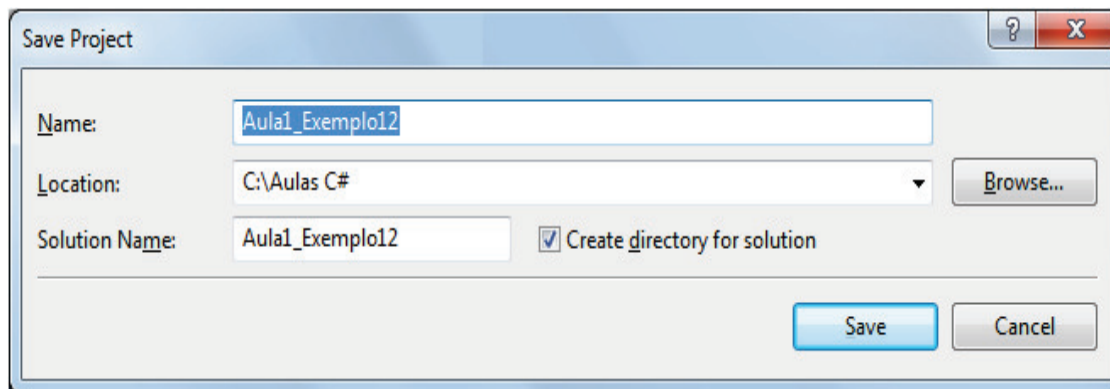
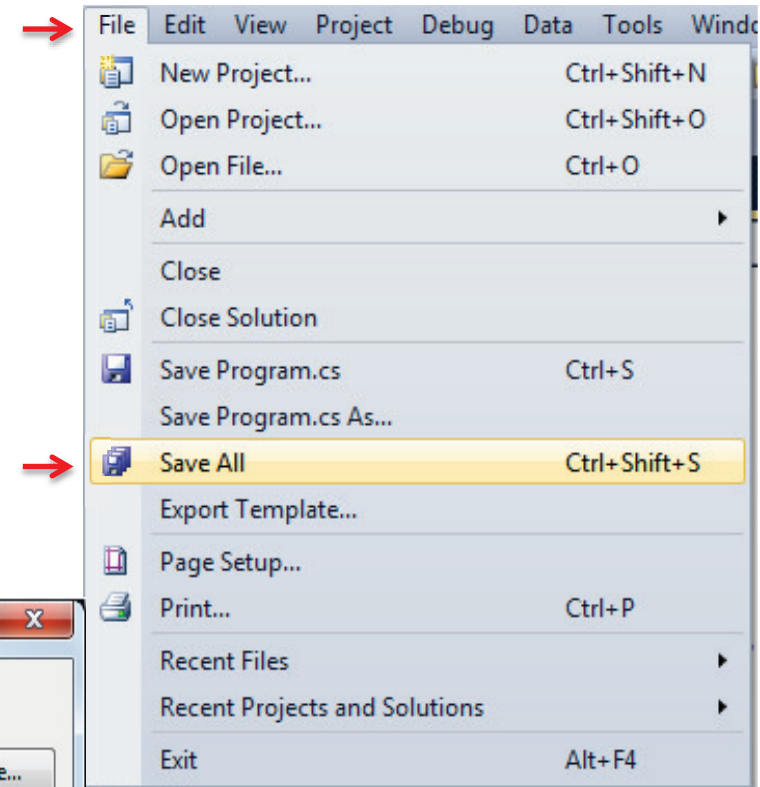
Muitas outras instruções serão vistas durante o curso, mas não todas, pois como será percebido, existem muitas bibliotecas com muitos recursos prontos para serem utilizados.

## Salvando o projeto pela primeira vez

Para salvar o projeto para clicar em **File** → **Save All**

Na sequencia irá aparecer uma janela, com alguns campos para serem preenchidos (Nome do Projeto, Local onde deseja salvar e Nome para a Solução).

Feito isso basta clicar em **Save**.





## Erros em tempo de Compilação

Caso a sintaxe (regra) de alguma instrução for escrita errada, na própria linha de comando irá aparecer um sublinhado em vermelho, alertando que algo não está escrito corretamente, aí basta analisar.

Caso não perceba que têm algo de errado, e tentar executar o programa, o compilador acusará um erro. E só conseguirá executá-lo depois que fizer as correções necessárias.

Por exemplo, se esquecer o ponto-e-vírgula do final da instrução, ou escrever o comando em letras minúsculas, ou esquecer de colocar aspas para exibir o texto, entre outras mais que acontecerão durante o curso.

```
Console.Write ("Hello world!")  
console.write ("Hello world!");  
console.write (Hello world!);
```

## Erros de compilação

```

Program.cs* X
Aula1_Exemplo1.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Aula1_Exemplo1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.write("Hello World!");
            Console.ReadKey();
        }
    }
}
100 %

```

Ao compilar/executar este exemplo (F5), aparecerá uma janela perguntando se quer continuar e executar a versão anterior antes da modificação (opção **Sim**), ou se não quer executar e verificar o que está de errado no código atual (opção **Não**). Vamos então clicar sempre em **Não**, para que possamos analisar os erros e corrigi-los.

Neste exemplo, quando compilar o programa, os erros encontrados serão listados. Então, basta realizar as correções sugeridas e compilar/executar novamente.

Error List				
<div> <div>2 Errors</div> <div>0 Warnings</div> <div>0 Messages</div> </div>				
Description	File	Line	Column	Project
1 'System.Console' does not contain a definition for 'write'	Program.cs	12	21	Aula1_Exemplo1
2 ; expected	Program.cs	12	42	Aula1_Exemplo1

## **Erros em tempo de Execução**

Este tipo de erro é gerado quando por exemplo, o usuário está utilizando o programa (executável) e ele digita uma letra em um campo onde era para ser digitado um número.

Neste caso, a execução do programa será encerrada e aparecerá uma informação com o possível erro ocorrido. Sendo assim, como o usuário não tem acesso ao código fonte, terá que solicitar ao programador que analise este erro e solucione o problema.

Na Linguagem C#, existem mecanismos para tratamento de erros em tempo de execução. Este assunto iremos tratar posteriormente.

## Tipos de Dados

Antes de continuarmos com o desenvolvimento de projetos, precisamos saber e entender algumas particularidades da estrutura da linguagem C#, como os tipos de dados, comando de atribuição, operadores aritméticos, operadores relacionais e operadores lógicos.

Toda linguagem de programação possui seus tipos de dados primitivos, ou seja, são os dados suportados diretamente pelo compilador. Esses dados possuem palavras chaves para facilitar sua utilização durante a implementação de um sistema.

Por exemplo, a palavra chave ***string*** mapeia diretamente o tipo ***System.String***. Inclusive saber/conhecer estes tipos, pode auxiliar no desempenho do sistema a ser desenvolvido.

## Tipos Primitivos em Linguagem C#

Tipo	Descrição	Faixa de dados
bool	Booleano	True ou false
byte	Inteiro de 8 bits com sinal	-127 a 128
char	Caracter Unicode de 16 bits	U+0000 a U+ffff
decimal	Inteiro de 96 bits com sinal com 28-29 dígitos significativos	$1,0 \times 10^{-28}$ a $7,9 \times 10^{28}$
double	Flutuante IEEE 64 bits com 15-16 dígitos significativos	$\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$
float	Flutuante IEEE 32 bits com 7 dígitos significativos	$\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$
int	Inteiro de 32 bits com sinal	-2.147.483.648 a 2.147.438.647
long	Inteiro de 64 bits com sinal	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
object	Classe base	
sbyte	Inteiro de 8 bits sem sinal	0 a 255
short	Inteiro de 16 bits com sinal	-32.768 a 32.767
string	String de caracteres Unicode	
uint	Inteiro de 32 bits sem sinal	0 a 4.294.967.295
ulong	Inteiro de 64 bits sem sinal	0 a 18.446.744.073.709.551.615
ushort	Inteiro de 16 bits sem sinal	0 a 65.535

## Declaração de variáveis

Para armazenar informações em um programa, é necessário reservar espaço na memória. E para isso temos que declarar variáveis, para que estas possam receber informações e assim, estas poderem ser utilizadas durante a execução do programa.

Como o próprio nome já diz, uma variável pode ter seu conteúdo alterado durante a execução de um programa, ou seja, ela pode iniciar valendo 5 e terminar valendo 322.

Exemplos de declaração de variáveis:

<code>int A, B, C;</code>	<i>//Declaração de três variáveis do tipo int (inteiro)</i>
<code>double valor;</code>	<i>//Declaração de uma variável do tipo double (real)</i>
<code>string nome;</code>	<i>//Declaração de uma variável do tipo string (sequencia de caracteres)</i>

## Definição de nomes para variáveis

Visando facilitar o entendimento do código fonte desenvolvido, é importante dar nome para as variáveis, de acordo com o papel que ela irá exercer no programa.

Por exemplo, no caso do programa precisar armazenar vários dados de um cadastro de cliente, e dentre elas está o número do Cadastro de Pessoa Física. Para criar uma relação que faça sentido para quem está desenvolvendo o código fonte, uma sugestão seria criar uma variável chamada **CPF**, ao invés de criar uma variável chamada **N**, **NUM**, **X** ou outra qualquer que não faça o mínimo sentido para com seu objetivo.

Lembrando também que variáveis não podem ter seus nomes com acentuação, caracteres especiais (exceto “\_”), espaço em branco e iniciar com números.

Aproveitando, o exemplo anterior, poderia ser criada a variável **Cadastro de Pessoa Física** em vez da variável **CPF**? (Resposta: Não. Pois possui espaços em branco)



## Comando de Atribuição

Para armazenar alguma informação em uma variável, usamos o símbolo "=", efetuando assim a atribuição de um valor em uma variável.

Exemplos de atribuição de dados em variáveis:

`A = 10;`                `//Está sendo atribuído o número 10 na variável A`

`valor = 5.5;`           `//Está sendo atribuído o valor 5.5 na variável valor`

`nome = "Tiago";` `//Está sendo atribuído Tiago na variável nome`

*Observem que quando trabalhamos com strings o conteúdo a ser armazenado deve estar delimitado entre aspas.*

## Operadores Aritméticos

Para a realização de operações envolvendo cálculos matemáticos, os operadores utilizados na Linguagem C# são:

<i><b>Operador</b></i>	<i><b>Operação</b></i>
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão

Exemplos:

$A = 10 + 2;$

$B = A - 5;$

$C = 587 * A + B;$

$\text{valor} = 5.5 / 2;$

$C = 34 \% 3;$

Para exemplificar estes operadores aritméticos, vamos testar usando exemplos práticos e analisar os resultados apresentados.

## Exemplo 2

Para este exemplo vamos criar um novo projeto com o nome **Aula1\_Exemplo2** e digitar o código a seguir:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace Aula1_Exemplo2  
{ //início namespace  
    class Program  
    { //início class
```

## Exemplo 2 - *continuação*

```
static void Main(string[] args)
{    //início Main

    //Declaração de variáveis
    int a, b, total;

    //Atribuição de valores
    a = 20;
    b = 10;

    /*Concatenação de valores a serem exibidos na tela
       Calculo da operação sendo realizado entre parênteses */
    Console.WriteLine(a + " + " + b + " = " + (a + b));
```

## Exemplo 2 - *continuação*

```
//Resultado do cálculo da operação sendo atribuído em uma variável
total = a - b;
Console.WriteLine(a + " - " + b + " = " + total);

Console.WriteLine(a + " * " + b + " = " + (a * b));

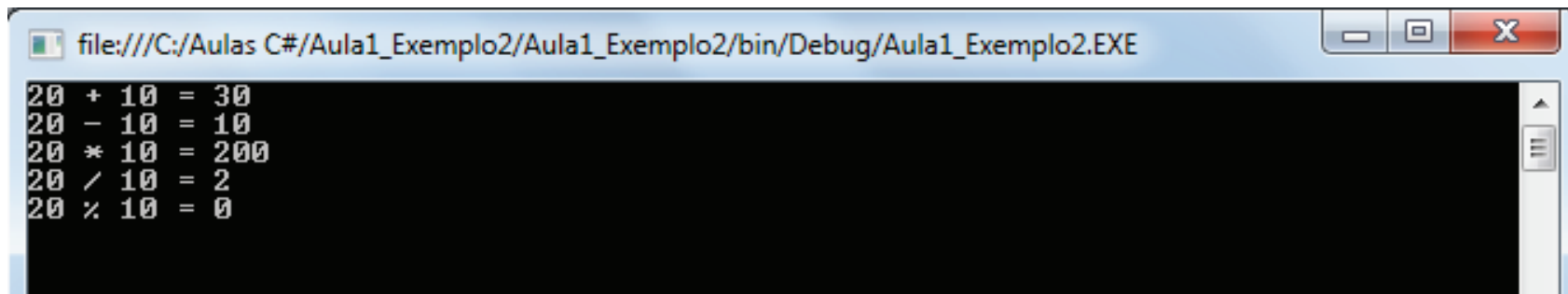
Console.WriteLine(a + " / " + b + " = " + (a / b));

Console.WriteLine(a + " % " + b + " = " + (a % b));

Console.ReadKey(); //Instrução utilizada para dar uma pausa para visualização
} //fim Main
} //fim class
} //fim namespace
```

## Exemplo 2 – Resultado

O resultado gerado na execução do código anterior é mostrado abaixo:



```
file:///C:/Aulas C#/Aula1_Exemplo2/Aula1_Exemplo2/bin/Debug/Aula1_Exemplo2.EXE
20 + 10 = 30
20 - 10 = 10
20 * 10 = 200
20 / 10 = 2
20 % 10 = 0
```

Observem que todo o conteúdo que está escrito em verde no código fonte não foi mostrado na tela. Isto significa que podemos usar comentários durante o desenvolvimento do código fonte. Como visto podemos usar das seguintes maneiras:

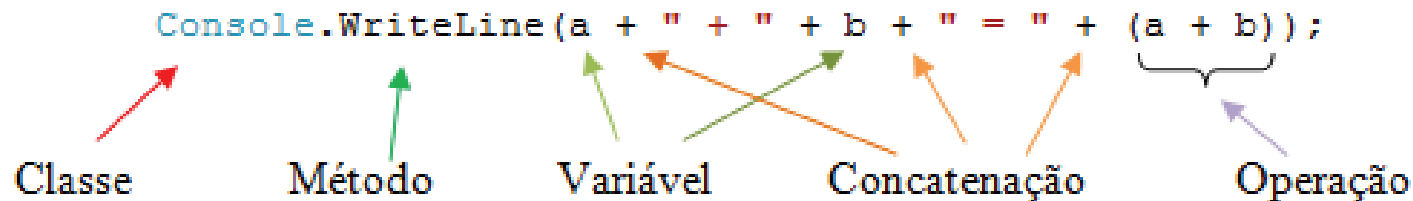
- // - para comentar o que estiver escrito a direita na mesma linha
- /\* ... \*/ - para comentar um trecho de código

## ***Console.WriteLine()***

Responsável por exibir uma mensagem da tela e logo em seguida pular uma linha.

Observem que no primeiro exemplo o cursor ficou piscando ao fim da mensagem, na mesma linha. No exemplo 2 o cursor ficou piscando na linha abaixo da última mensagem.

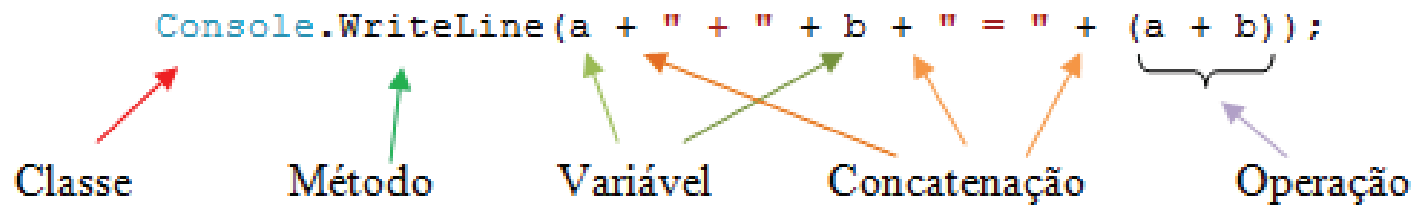
Nestes métodos para exibição de dados na tela, notem que o que está entre aspas é o que exatamente saiu na tela, e o que está fora das aspas, foi exibido o valor que está armazenado na variável. Para compreender melhor a sintaxe desta linha de comando:





## Entendendo melhor a instrução

Nesta linha de comando, vamos entender cada um dos itens:



**Classe:** É um tipo de objeto que contém vários métodos (funções) prontos para serem utilizados, ou seja, que já fazem parte do pacote de bibliotecas da Linguagem C#.

Toda linha de comando que estiver utilizando a classe Console, significa que estará executando uma ação sob a janela de console (tela preta) onde é gerado o resultado da execução do programa.

## Entendendo melhor a instrução

**Métodos:** São funções que pertencem as classes, ou seja, para exibir dados na tela podemos utilizar os métodos Write() ou WriteLine(), para dar uma pausa no programa utilizamos o ReadKey(), para capturar um texto digitado pelo usuário e armazenar em uma variável utilizamos ReadLine() e assim por diante. Cada classe possui um conjunto de métodos pré-definidos.

**Variável:** Como já visto anteriormente nesta mesma aula, são nomes que são dados aos espaços de memória para que sejam armazenados valores, que posteriormente possam ser utilizados durante a execução o programa. No caso as variáveis ***a*** e ***b*** possuem valores do tipo inteiro.

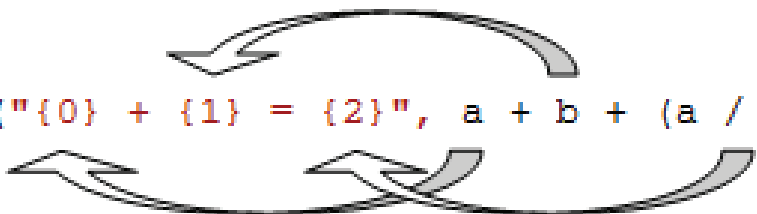
## Entendendo melhor a instrução

**Concatenação:** O símbolo de adição “+” é usado para concatenar, ou seja, montar uma mensagem para que seja exibida uma mensagem organizada na tela. Podemos definir como “juntar” texto com variáveis para que seja exibida a mensagem na tela da forma desejada.

**Operação:** Neste caso está realmente sendo realizada uma operação de adição (soma) de duas variáveis. O que difere da concatenação são exatamente os parênteses que faz o compilador interpretar que representa uma operação aritmética.

## Outra forma de exibição

Uma outra forma de exibir dados da tela utilizando os mesmos comandos (***Console.Write*** e ***Console.WriteLine***) é a de formatação de saída. Como é mostrado no exemplo a seguir:



```
Console.WriteLine("{0} + {1} = {2}", a + b + (a / b));
```

Neste caso, o valor que está armazenado na variável ***a*** será exibido no lugar da **{0}**, o valor armazenado na variável ***b*** será exibido no lugar da **{1}**, e o valor gerado pelo resultado da operação será exibido no lugar da **{2}**.

Notem que o resultado da divisão é um número real, e para que sejam exibidas duas casas decimais deste resultado é necessário formatar a saída de dados, e umas das formas é usando **{2:N2}**, ou se quiser três casas decimais **{2:N3}**.

## Precedência de Operadores

Operações dentro dos parênteses são sempre executadas primeiro, como nas operações matemáticas.

Em C#, operadores multiplicativos (\*, /, %) tem precedência sobre os aditivos (+, -).

Observem a ordem de precedência das seguintes expressões:

Exemplo 1:

$2 + 3 * 4$

$2 + \mathbf{3 * 4}$

$2 + 12$

14

Exemplo 2:

$(2 + 3) * 4$

$\mathbf{(2 + 3) * 4}$

$5 * 4$

20

Perceba que a ordem de precedência altera o valor do resultado, por isso devemos ter atenção com a precedência.

## Constantes

São nomes dados a valores que não poderão ter seu conteúdo alterado durante a execução de um programa, ou seja, diferente das variáveis as constantes manterão o mesmo valor do início ao fim da execução.

Para a definição de constantes utilizaremos a sintaxe:

***const <tipo\_de\_dado> <nome\_da\_constante> = <valor>;***

Exemplos de definição de constantes:

```
const int ano = 2013;
```

```
const string linguagem = "C Sharp";
```

```
const double pi = 3.14159265358979323846264338327950;
```

*Obs.: Nos próximos exemplos será mostrado apenas o conteúdo do **Main**, mas continue criando um novo projeto para cada exemplo, conforme já demonstrado.*

## Exemplo 3: Constante e Formatação

```
static void Main(string[] args)
{
    //definição do constante pi
    const double pi = 3.14159265358979323846264338327950;

    //declaração de variável e atribuição de valor na variável
    double raio = 10;
    double area;

    Console.WriteLine("Calculo da área de um Circulo \n");
    Console.WriteLine("Fórmula: area = pi * raio² \n");

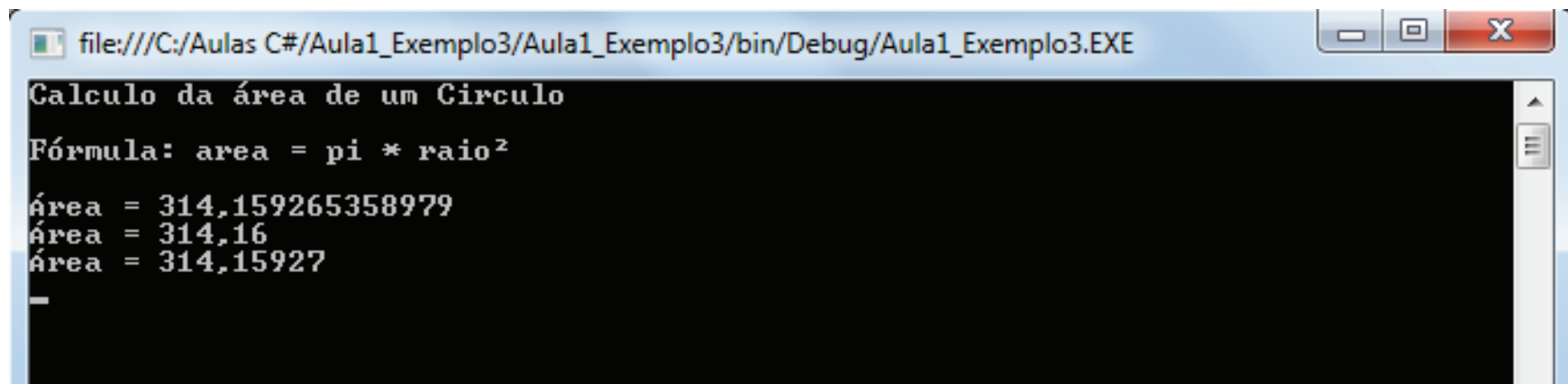
    //Math.Pow: exponenciação
    area = pi * Math.Pow(raio, 2);    //sem usar método Pow seria: area = pi * raio* raio;

    Console.WriteLine("Área = {0}", area);    //exibe resultado sem formatação de casas decimais
    Console.WriteLine("Área = {0:N}", area);    //exibe resultado com formatação de duas casas decimais
    Console.WriteLine("Área = {0:N5}", area);    //exibe resultado com formatação de cinco casas decimais

    Console.ReadKey();
}
```



- Exemplo 3: Resultado



```
file:///C:/Aulas C#/Aula1_Exemplo3/Aula1_Exemplo3/bin/Debug/Aula1_Exemplo3.EXE
Calculo da área de um Circulo
Fórmula: area = pi * raio²
Área = 314,159265358979
Área = 314,16
Área = 314,15927
_
```

Para o cálculo da exponenciação, a Linguagem C# possui uma classe chamada Math que fornece várias funções matemáticas já prontas para serem utilizadas.

A seguir serão listadas algumas funções fornecidas por esta classe.

## Biblioteca Math

Algumas funções matemáticas da classe Math:

Função	Finalidade	Exemplo
<b>Abs(x)</b>	Valor absoluto de x	$\text{Abs}(3.5) = 3,5$ ; $\text{Abs}(-3.5) = 3,5$ ;
<b>Acos(x)</b>	Arco cosseno de x	$\text{Acos}(0.3) = 1,26\dots$
<b>Asin(x)</b>	Arco seno de x	$\text{Asin}(0.6) = 0,64\dots$
<b>Atan(x)</b>	Arco tangente de x	$\text{Atan}(0.5) = 0,46\dots$
<b>Ceiling(x)</b>	Arredonda x para cima	$\text{Ceiling}(0.1) = 1$ ; $\text{Ceiling}(-0.1) = 0$ ;
<b>Cos(x)</b>	Cosseno de x	$\text{Cos}(0.5) = 0,87\dots$
<b>Exp(x)</b>	Exponencial (e elevado na x)	$\text{Exp}(2.0) = 7,38\dots$
<b>Floor(x)</b>	Arredonda x para baixo	$\text{Floor}(0.9) = 0$ ; $\text{Floor}(-0.9) = -1$ ;
<b>Log(x)</b>	Logaritmo de x na base natural e	$\text{Log}(2.71) = 0,99\dots$

## Biblioteca Math - continuação

Função	Finalidade	Exemplo
<b>Max(x,y)</b>	Maior valor entre dois números	$\text{Max}(1.5, 1.7) = 1,7;$ $\text{Max}(-0.5, -0.7) = -0,5;$
<b>Min(x)</b>	Menor valor entre dois números	$\text{Min}(1.5, 1.7) = 1,5;$ $\text{Min}(-0.5, -0.7) = -0,7;$
<b>PI</b>	Valor de $\pi$	$\text{PI} = 3,14\dots$
<b>Pow(x,y)</b>	Valor de x elevado na y	$\text{Pow}(2, 4) = 16$
<b>Round(x,y)</b>	Arredonda x para y casas decimais	$\text{Round}(5.98765, 3) = 5,988$
<b>Sin(x)</b>	Seno de x	$\text{Sin}(2) = 0,9\dots$
<b>Sqrt(x)</b>	Raiz quadrada de x	$\text{Sqrt}(81) = 9$
<b>Tan(x)</b>	Tangente de x	$\text{Tan}(0.5) = 0,54\dots$

## Exemplo 4: Tratando Strings

O tipo string possui alguns métodos interessantes e bem particulares. No exemplo 4 a seguir serão mostrados alguns métodos muito utilizados.

```
static void Main(string[] args)
{
    string nome = "Tiago", sobrenome = "Souza", completo;
    completo = nome + " " + sobrenome;

    Console.WriteLine("1) " + nome + sobrenome);

    Console.WriteLine("2) " + completo);

    Console.WriteLine("3) " + nome + " possui " + nome.Length + " caracteres");

    Console.WriteLine("4) " + sobrenome + " possui " + sobrenome.Length + " caracteres");

    Console.WriteLine("5) " + completo + " possui " + completo.Length + " caracteres - incluindo espaço em branco");

    Console.WriteLine("6) " + nome + " em minuscuro " + nome.ToLower());

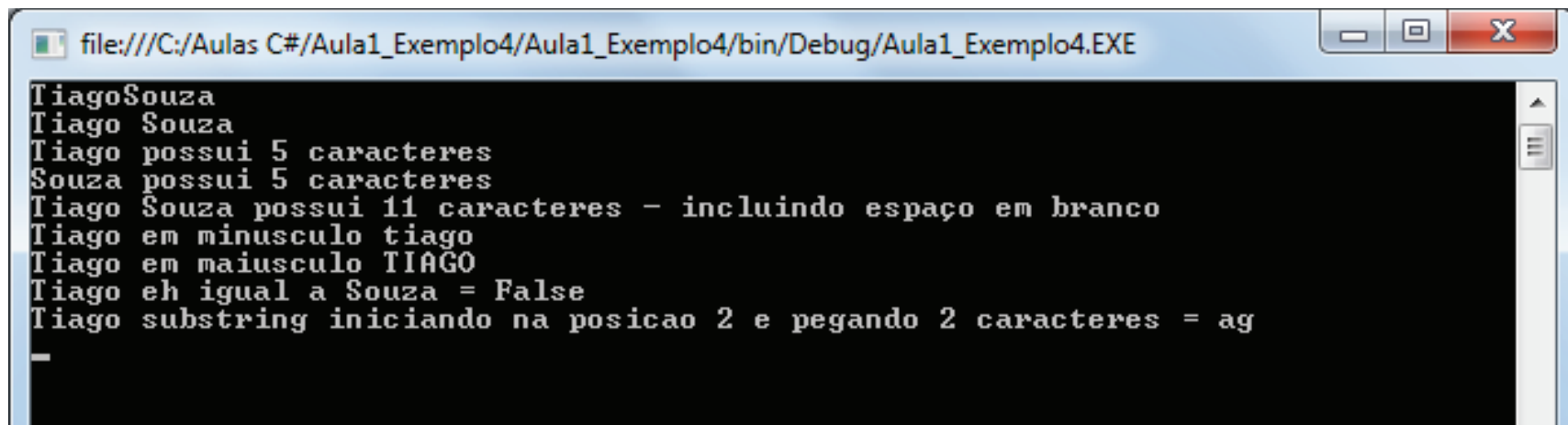
    Console.WriteLine("7) " + nome + " em maiusculo " + nome.ToUpper());

    Console.WriteLine("8) " + nome + " eh igual a " + sobrenome + " = " + nome.Equals(sobrenome));

    Console.WriteLine("9) " + nome + " substring iniciando na posicao 2 e pegando 2 caracteres = " + nome.Substring(2, 2));

    Console.ReadKey();
}
```

## Exemplo 4 – Resultado



```
file:///C:/Aulas C#/Aula1_Exemplo4/Aula1_Exemplo4/bin/Debug/Aula1_Exemplo4.EXE
TiagoSouza
Tiago Souza
Tiago possui 5 caracteres
Souza possui 5 caracteres
Tiago Souza possui 11 caracteres - incluindo espaço em branco
Tiago em minusculo tiago
Tiago em maiusculo TIAGO
Tiago eh igual a Souza = False
Tiago substring iniciando na posicao 2 e pegando 2 caracteres = ag
-
```

### Métodos usados:

nome.Length – retorna a quantidade de caracteres armazenadas na variável

nome.ToLower() – converte o conteúdo da string em caracteres minúsculos

nome.ToUpper() – converte o conteúdo da string em caracteres maiúsculos

nome.Equals(sobrenome) – realiza a comparação entre duas strings

## Entrada de Dados

Até o momento nos exemplos apresentados, não houve nenhuma interação com o usuário. Sendo que os valores foram definidos em tempo de programação, ou seja, no desenvolvimento do código fonte.

Agora será apresentada uma maneira de fazer com que o usuário digite a informação e esta informação fique armazenada em uma variável. E para isso iremos utilizar o método ***Console.ReadLine()***.

Este método é responsável por aguardar o usuário digitar uma informação e pressionar <enter>. Fazendo isso, a informação digitada pelo usuário será armazenada em uma variável. Exemplo:

```
string <variável> = Console.ReadLine();
```

Esta informação sempre será capturada no formato de ***string***.

## Exemplo 5 - Entrada de Dados

```
static void Main(string[] args)
{
    string nome;
    string cidade;

    Console.Write("Digite seu nome: ");
    nome = Console.ReadLine();

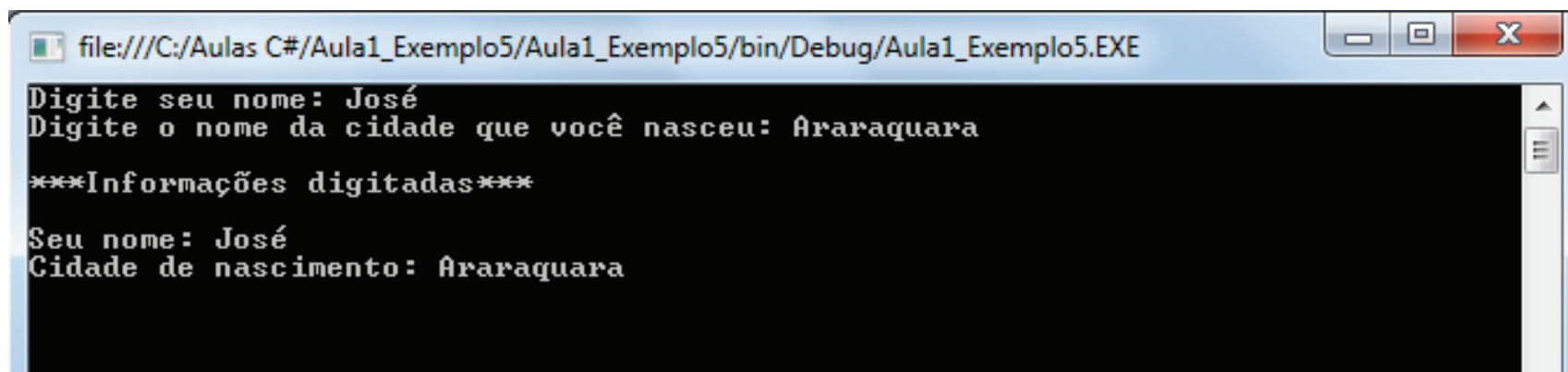
    Console.Write("Digite o nome da cidade que você nasceu: ");
    cidade = Console.ReadLine();

    Console.WriteLine("\n***Informações digitadas*** \n");
    Console.WriteLine("Seu nome: " + nome);
    Console.WriteLine("Cidade de nascimento: " + cidade);

    Console.ReadKey();
}
```

Podemos usar o “\n” para pular linha, ou seja, neste exemplo irá pular uma linha antes e outra depois que for exibida a mensagem “\*\*\**Informações digitadas*\*\*\*”.

## Exemplo 5 - Resultado



```
file:///C:/Aulas C#/Aula1_Exemplo5/Aula1_Exemplo5/bin/Debug/Aula1_Exemplo5.EXE
Digite seu nome: José
Digite o nome da cidade que você nasceu: Araraquara

***Informações digitadas***

Seu nome: José
Cidade de nascimento: Araraquara
```

Neste exemplo o nome digitado “José” foi armazenado na variável *nome* e a **cidade** informada “Araraquara” foi armazenada na variável **cidade**. Ambas variáveis são do tipo *string*.

Na sequência foram exibidas as informações que foram digitadas.



## Armazenar outros tipos de dados

Conforme já vimos anteriormente, toda a informação capturada pelo ***Console.ReadLine()*** é reconhecida como uma string. Portanto, para que a informação seja armazenada em uma variável de outro tipo (diferente de ***string***), temos que aplicar a conversão de tipo adequada a cada necessidade.

Exemplo de conversão:

```
int N = int.Parse(Console.ReadLine()); ou int n = Convert.ToInt32(Console.ReadLine());
```

O ***Console.ReadLine()*** será responsável por capturar a informação que o usuário digitar. Na sequência esta informação será convertida para inteiro pelo método ***int.Parse()*** ou pelo método ***Convert.ToInt32()***. E por fim será armazenada na variável ***N*** que é do tipo inteiro.

## Mais alguns exemplos de conversões

`double V = double.Parse(Console.ReadLine());`    *//Conversão de string para double*  
ou    `double V = Convert.ToDouble(Console.ReadLine());`

`bool R = bool.Parse(Console.ReadLine());`    *//Conversão de string para boolean*  
ou    `bool R = Convert.ToBoolean(Console.ReadLine());`

`char L = char.Parse(Console.ReadLine());`    *//Conversão de string para char*  
ou    `char L = Convert.ToChar(Console.ReadLine());`

Existem ainda outros métodos para conversão para outros tipos de dados, veremos durante o curso, de acordo com as necessidades.

## Exemplo 6 – Conversão de Tipos de Dados

```
static void Main(string[] args)
{
    string nome;
    int idade;
    double peso;

    Console.Write("Digite seu nome: ");
    nome = Console.ReadLine();

    Console.Write(nome + ", digite sua idade: ");
    idade = int.Parse(Console.ReadLine());    //ou idade = Convert.ToInt32(Console.ReadLine());

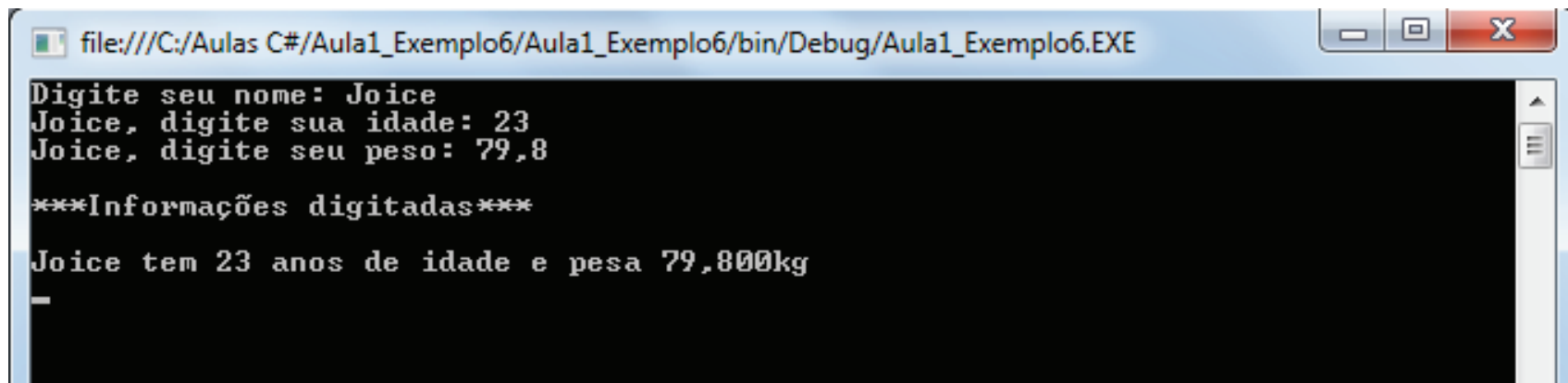
    Console.Write(nome + ", digite seu peso: ");
    peso = double.Parse(Console.ReadLine());    //ou peso = Convert.ToDouble(Console.ReadLine());

    Console.WriteLine("\n***Informações digitadas*** \n");
    Console.WriteLine("{0} tem {1} anos de idade e pesa {2:N3}kg", nome, idade, peso);

    Console.ReadKey();
}
```

Neste exemplo a variável nome é do tipo string e por isso não necessita de conversão. Já para as demais variáveis, devemos aplicar a conversão, conforme demonstrado.

## Exemplo 6 - Resultado



```
file:///C:/Aulas C#/Aula1_Exemplo6/Aula1_Exemplo6/bin/Debug/Aula1_Exemplo6.EXE
Digite seu nome: Joice
Joice, digite sua idade: 23
Joice, digite seu peso: 79,8

***Informações digitadas***
Joice tem 23 anos de idade e pesa 79,800kg
_
```

Logo após o usuário digitar as informações , será exibida a mensagem de acordo com a formatação pré-definida.

Nota: Quando trabalhamos com números reais, devemos usar a vírgula “,” para identificar as casas decimais.

## Exemplo 7 – Adição de dois números inteiros

Neste exemplo, o usuário deverá digitar dois números inteiros, na sequência será realizada a adição dos números digitados e o resultado será armazenado na variável *c* e por fim será exibido o resultado da operação.

```
static void Main(string[] args)
{
    int a, b, c;
    Console.WriteLine("Digite um numero: ");
    a = Convert.ToInt32(Console.ReadLine());

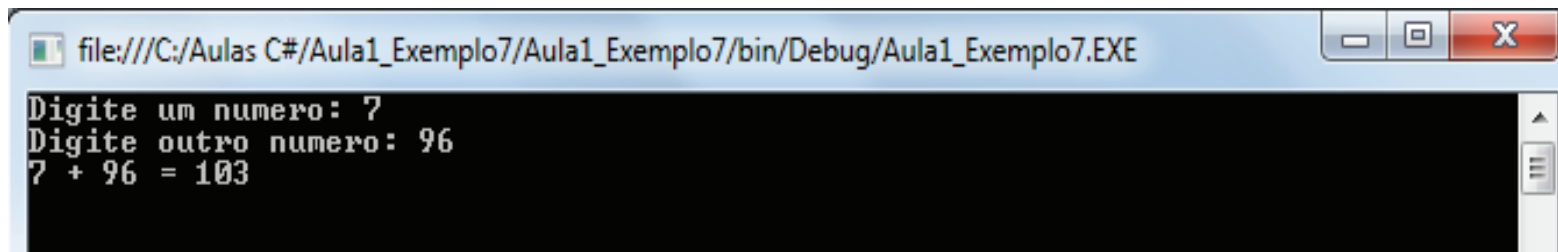
    Console.WriteLine("Digite outro numero: ");
    b = Convert.ToInt32(Console.ReadLine());

    c = a + b;

    Console.WriteLine(a + " + " + b + " = " + c);

    Console.ReadKey();
}
```

## Resultado



```
file:///C:/Aulas C#/Aula1_Exemplo7/Aula1_Exemplo7/bin/Debug/Aula1_Exemplo7.EXE
Digite um numero: 7
Digite outro numero: 96
7 + 96 = 103
```

## Operadores Relacionais

Em programação, é de suma importância o uso destes operadores, tendo em vista que a maioria dos problemas computacionais envolve comparações entre valores.

Como são operadores que realizarão testes comparativos entre valores, o resultado do uso destes operadores será sempre um valor lógico, verdadeiro (true) ou falso (false).

Segue tabela com os operadores relacionais usados na linguagem C# e suas ações:

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

## Exemplo 8 - Operadores Relacionais

Neste exemplo, o usuário deverá digitar dois números inteiros e será exibido o resultado da relação entre estes dois números com cada operador relacional.

```
static void Main(string[] args)
{
    int a, b;

    Console.WriteLine("Exemplo - Operadores Relacionais");
    Console.WriteLine("***** \n\n");

    Console.Write("Digite um número: ");    a = int.Parse(Console.ReadLine());
    Console.Write("Digite outro número: "); b = int.Parse(Console.ReadLine());
    Console.WriteLine();

    Console.WriteLine("{0} é igual a {1}?      Resposta: {2}", a, b, (a == b));
    Console.WriteLine("{0} é diferente de {1}?  Resposta: {2}", a, b, (a != b));
    Console.WriteLine("{0} é maior que {1}?     Resposta: {2}", a, b, (a > b));
    Console.WriteLine("{0} é menor que {1}?     Resposta: {2}", a, b, (a < b));
    Console.WriteLine("{0} é maior ou igual a {1}? Resposta: {2}", a, b, (a >= b));
    Console.WriteLine("{0} é menor ou igual a {1}? Resposta: {2}", a, b, (a <= b));

    Console.ReadKey();
}
```

## Exemplo 8 – Resultado

```
file:///C:/Aulas C#/Aula1_Exemplo8/Aula1_Exemplo8/bin/Debug/Aula1_Exemplo8.EXE
Exemplo - Operadores Relacionais
*****

Digite um número: 5
Digite outro número: 13

5 é igual a 13?           Resposta: False
5 é diferente de 13?      Resposta: True
5 é maior que 13?         Resposta: False
5 é menor que 13?        Resposta: True
5 é maior ou igual a 13?  Resposta: False
5 é menor ou igual a 13?  Resposta: True
```

Observem que as respostas são apresentadas como valores lógicos (**boolean**).

Estes operadores serão muito utilizados em trechos do programa, para tomadas de decisões.



## Operadores Lógicos

Os operadores lógicos são usados para relacionar expressões e a partir destas comparações, os resultados obtidos serão também de um tipo lógico (***boolean***).

São usados somente em expressões lógicas, e são descritos na tabela a seguir:

Operador	Ação
&&	AND (E)
	OR (OU)
!	NOT (NÃO)

O operador lógico **!** (NOT) têm a finalidade de inverter o valor lógico, ou seja, se temos um valor lógico *Falso*, aplicando o operador resultará em *Verdadeiro*, e vice-versa.

## Operadores Lógicos – Tabela Verdade

Para auxiliar no entendimento, a seguir é mostrado o resultado da operação entre valores lógicos para cada operador lógico.

p	q	p && q	p    q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

Exemplo 1:  $72 > 30$  &&  $32 \leq 10$   
Verdadeiro && Falso  
Falso

Exemplo 2:  $9 == 3$  ||  $10 \leq 10$   
Falso || Verdadeiro  
Verdadeiro

## Exemplo 9 – Operadores Lógicos

Neste exemplo, o usuário deve digitar dois valores lógicos, e na sequência será exibido o resultado de cada operador lógico.

```
static void Main(string[] args)
{
    bool i, j;

    Console.WriteLine("Exemplo - Operadores Lógicos");
    Console.WriteLine("*****\n\n");
    Console.WriteLine("Digite true(verdadeiro) ou false(falso)\n ");

    Console.Write("Valor Lógico 1: ");    i = Boolean.Parse(Console.ReadLine());
    Console.Write("Valor Lógico 2: ");    j = Boolean.Parse(Console.ReadLine());

    Console.WriteLine("\nResultados testando os operadores lógicos\n");
    Console.WriteLine("{0} AND {1} resulta {2}", i, j, i && j);
    Console.WriteLine("{0} OR {1} resulta {2}", i, j, i || j);
    Console.WriteLine("Valor 1: NOT {0} resulta {1}", i, !i);
    Console.WriteLine("Valor 2: NOT {0} resulta {1}", j, !j);

    Console.ReadKey();
}
```

## Exemplo 9 – Resultado

```
file:///C:/Aulas C#/Aula1_Exemplo9/Aula1_Exemplo9/bin/Debug/Aula1_Exemplo9.EXE
Exemplo - Operadores Lógicos
*****

Digite true(verdadeiro) ou false(falso)
Valor Lógico 1: true
Valor Lógico 2: false

Resultados testando os operadores lógicos
True AND False resulta False
True OR False resulta True
Valor 1: NOT True resulta False
Valor 2: NOT False resulta True
-
```

A utilização destes operadores pode reduzir consideravelmente o número de linhas do código fonte de um programa, evitando fazer várias comparações em linhas de instruções diferentes. Veremos durante o curso, outras aplicações utilizando estes operadores.

## Operadores Ternários

É composto por três operandos (expressões/valores) separados pelos sinais **?** e **:**.  
Têm o objetivo de atribuir o valor a uma variável de acordo com o resultado de um teste lógico. Veja a sintaxe dele abaixo:

**teste lógico ? valor se verdadeiro : valor se falso;**

Onde:

- teste lógico é qualquer valor ou expressão, que pode ser verdadeiro ou falso.
- valor se verdadeiro é atribuído o valor **true**;
- valor se falso é atribuído o valor **false**.

## Exemplo 10 - Operadores Ternários

Neste exemplo, foi realizado o teste para saber se o primeiro número digitado é maior que o segundo número digitado, sendo que a resposta será o valor da variável verdade ou da variável falso.

```
static void Main(string[] args)
{
    int a, b;
    string verdade = "sim", falso = "não";

    Console.Write("Digite um numero inteiro: ");
    a = int.Parse(Console.ReadLine());
    Console.Write("Digite outro numero inteiro: ");
    b = int.Parse(Console.ReadLine());
    Console.WriteLine();
    Console.Write("{0} > {1} ? Resposta: ", a, b);
    Console.WriteLine(a > b ? verdade : falso);

    Console.ReadKey();
}
```

## Resultado

```
file:///C:/Aulas C#/Aula1_Exemplo10/Aula1_Exemplo10/bin/Debug/Aula1_Exemplo10.EXE
Digite um numero inteiro: 7
Digite outro numero inteiro: 2
7 > 2 ? Resposta: sim
```

## Outros Operadores de Atribuição

São formas usadas para otimizar uma linha de instrução, que usa o valor da própria variável (antes o operador) para realizar o cálculo e atribuir nela mesma o novo resultado.

Operador	Descrição
=	Atribuição Simples (já visto)
+=	Atribuição Aditiva
-=	Atribuição Subtrativa
*=	Atribuição Multiplicativa
/=	Atribuição de Divisão
%=	Atribuição de Resto da Divisão

## Exemplo 11 - Operadores de Atribuição

```
static void Main(string[] args)
{
    int num1=10, num2=2;

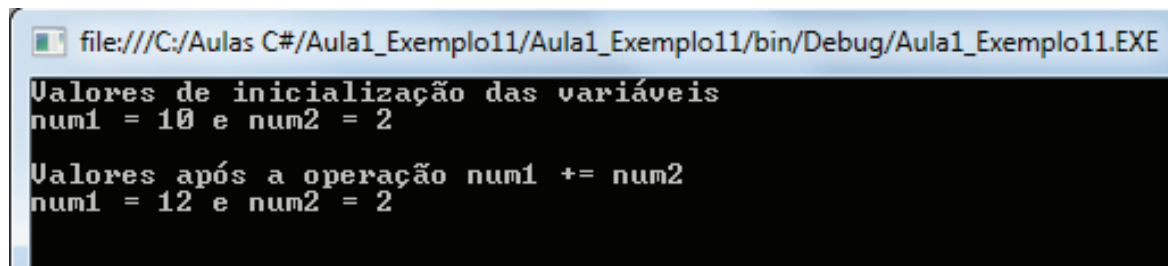
    Console.WriteLine("Valores de inicialização das variáveis");
    Console.WriteLine("num1 = {0} e num2 = {1}", num1, num2);

    num1 += num2;          //é o mesmo que: num1 = num1 + num2;
    Console.WriteLine();
    Console.WriteLine("Valores após a operação num1 += num2");
    Console.WriteLine("num1 = {0} e num2 = {1}", num1, num2);

    Console.ReadKey();
}
```

### Resultado

O mesmo é válido para os outros operadores.





## Incremento e Decremento

Incrementar um número significa adicionar 1 ao valor atual.

Decrementar um número significa subtrair 1 de um número atual.

Operador	Descrição
++	Incremento
--	Decremento

Exemplo 1:

```
int X = 1;    //Atribuindo o valor 1 na variável X
X++;         //Incrementando 1 na mesma variável, que passa a valer 2
```

Exemplo 2:

```
int X = 8;    //Atribuindo o valor 8 na variável X
X--;         //Decrementando 1 na mesma variável, que passa a valer 7
```

## Prefixo e Sufixo

Tanto o operador de incremento quanto o de decremento tem duas formas de serem aplicados: prefixo e sufixo.

**Prefixo:** O operador está antes do nome da variável. Ex: `int X = ++Y;`

Quando estamos atribuindo um valor a uma variável usando prefixo, significa que primeiro deve ser feito o incremento na variável **Y** e depois atribuído o novo valor na variável destino, que no exemplo é a variável **X**.

**Sufixo:** O operador está depois do nome da variável. Ex: `int X = Y++;`

No caso de usarmos o sufixo, primeiro o valor de **Y** seria passado para a variável **X**, e somente depois disso é que haveria o incremento em **Y**.

## **Bibliografia**

- Manzano, José Augusto N. G., **Estudo Dirigido de Microsoft Visual C# 2010 Express**. São Paulo, SP, Editora Érica, 2010.
- MSDN, Microsoft. **Guia de Programação C#**. Disponível: [http://msdn.microsoft.com/pt-br/library/67ef8sbd\(v=vs.80\).aspx](http://msdn.microsoft.com/pt-br/library/67ef8sbd(v=vs.80).aspx). Acesso em 31 jan 2013
- Dantas, Cleber. **Operadores em C Sharp**. Disponível: <http://www.linhadecodigo.com.br/artigo/1007/serie-aprenda-csharp-operadores-em-csharp.aspx#ixzz2JW53NQkC>. Acesso em 31 jan 2013.
- Balbo, Wellington. **Estrutura da Linguagem e Operadores em C Sharp**. Disponível: <http://www.devmedia.com.br/relacionais-logicos-e-ternario-operadores-do-c-estrutura-da-linguagem-parte-3/18875>. Acesso em 31 jan 2013