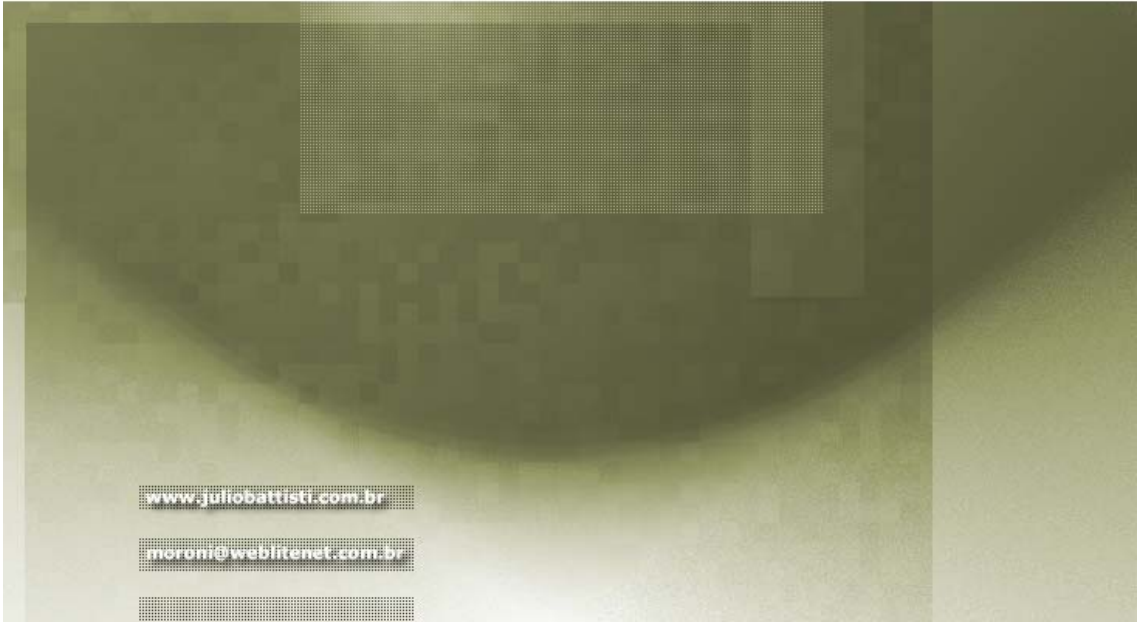




Herbert Moroni

Curso de Lógica de Programação com C# e VB.NET



www.juliobattisti.com.br

moroni@weblitenet.com.br

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Nota sobre direitos autorais:

Este e-book é de autoria de Herbert Moroni Cavallari da Costa Gois, sendo comercializado diretamente através do site www.juliobattisti.com.br e www.linhadecodigo.com.br ou através do site de leilões Mercado Livre: www.mercadolivre.com.br, mediante contato através do email: batisti@hotmail.com ou webmaster@juliobattisti.com.br, diretamente pelo autor ou por Júlio Battisti. No Mercado Livre, somente o usuário GROZA é que tem autorização para comercializar este e-book. **Nenhum outro usuário/email e/ou empresa está autorizada a comercializar este ebook.**

Ao adquirir este ebook você tem o direito de lê-lo na tela do seu computador e de imprimir quantas cópias desejar. É vetada a distribuição deste arquivo, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. Se você recebeu este ebook através do e-mail ou via ftp de algum site da Internet, ou através de um CD de Revista, saiba que você está com uma cópia pirata, ilegal, não autorizada, a qual constitui crime de Violação de Direito Autoral, de acordo com a Lei 5988. Se for este o caso entre em contato com o autor, através do e-mail webmaster@juliobattisti.com.br, para regularizar esta cópia. Ao regularizar a sua cópia você irá remunerar, mediante uma pequena quantia, o trabalho do autor e incentivar que novos trabalhos sejam disponibilizados. Se você tiver sugestões sobre novos cursos que gostaria de ver disponibilizados, entre em contato pelo e-mail: webmaster@juliobattisti.com.br.

Visite periodicamente o site www.juliobattisti.com.br para ficar por dentro das novidades:

- Cursos de informática.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- Guias de Estudo para os Exames de Certificação da Microsoft.
- Artigos e dicas sobre Certificações da Microsoft.
- Artigos sobre Carreira e Trabalho.
- Dicas de livros e sites sobre diversos assuntos.
- Simulados gratuitos, em português, para os exames da Microsoft.

- **ESTE E-BOOK NÃO PODE SER FORNECIDO EM UM CD OU DVD DE NENHUMA REVISTA**
- **SE VOCÊ OBTVEU UMA CÓPIA DESTE E-BOOK ATRAVÉS DO E-MULE, KAZAA, MORPHEUS OU OUTRO PROGRAMA DE COMPARTILHAMENTO, SAIBA QUE VOCÊ ESTÁ COM UMA CÓPIA ILEGAL, NÃO AUTORIZADA**
- **USAR UMA CÓPIA NÃO AUTORIZADA É CRIME DE VIOLAÇÃO DE DIREITOS AUTORAIS, COM PENA PREVISTA DE CADEIA**
- **VOCÊ SÓ PODE USAR ESTE E-BOOK SE VOCÊ COMPROU ELE DIRETAMENTE COM O AUTOR OU ATRAVÉS DOS SITES WWW.JULIOBATTISTI.COM.BR OU WWW.LINHADECODIGO.COM.BR**

PRÉ-REQUISITOS PARA O CURSO:

Para que você possa acompanhar as lições deste curso é necessário que você já tenha preenchido os seguintes pré-requisitos:

- Conhecimento básico do Windows 98, 2000 ou XP, tais como:
 - 📖 Criação de pastas e subpastas.
 - 📖 Utilização do mouse e do teclado.
 - 📖 Operações básicas com arquivos e pastas, usando o Windows Explorer.
 - 📖 Conhecer conceitos tais como ícones, área de trabalho, janelas do Windows, uso de menus e outras configurações básicas do Windows.

Palavras do autor:

A proposta desse curso é ensinar os conceitos sobre lógica de programação e a construção de algoritmos computacionais. Além disso apresentamos duas linguagens de programação e ensinamos a aplicar os conceitos aprendidos de lógica nas mesmas.

O VB.NET e C# junto com o Visual Studio .NET 2005 compõe uma ferramenta extremamente robusta e fácil de utilizar, com perfeito suporte a todas as novas ondas que rondam o mundo da informática e tecnologia.

Tanto a linguagem VB.NET e a C# trabalham de forma praticamente idêntica na plataforma .NET da Microsoft, com diferença de sintaxe, como você vai ver se acompanhar o conteúdo das duas linguagens.

O Visual Studio .NET 2005 é a melhor ferramenta de desenvolvimento de aplicações para a plataforma .NET. Com uma interface amigável e integrada com os ambientes e de fácil entendimento, proporciona aos desenvolvedores a criação de aplicações sofisticadas com todos os recursos existentes, sem ter que ficar criando parte de código em um aplicativo e o restante no outro. É possível com o Visual Studio gerenciar recursos da máquina e local e de um possível servidor, criar aplicações para Windows, web e dispositivos móveis.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Também estou a disposição para responder eventuais dúvidas sobre o conteúdo do curso, envie-me também suas sugestões para que possamos sempre melhorar o material proposto. Meu e-mail para contato é herbertmoroni@hotmail.com ou moroni@weblitenet.com.br.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Índice do Curso

CAPITULO 1.....	11
INTRODUÇÃO – CONCEITOS INICIAIS	11
CAPITULO 2.....	15
ALGORITMO	15
CAPITULO 3.....	28
VARIÁVEIS, TIPOS DE DADOS E CONSTANTES	28
3.1 – Tipos de dados	29
3.2 – Declarando Variáveis.....	31
3.3 – Constantes	33
3.4 – Case-sensitive.....	34
3.5 – Exercícios para fixação	35
CAPITULO 4.....	37
OPERADORES	37
4.1 – Operadores de atribuição	37
4.2 - Operadores aritméticos	38
4.3 – Operadores relacionais	39
4.4 – Operadores Lógicos.....	41
4.5 – Funções matemáticas	43
4.6 – Precedência dos operadores	44
4.7 – Tabela-verdade	46
4.8 – Exercícios para fixação	48
CAPITULO 5.....	49
COMANDO DE ENTRADA E SAÍDA	49
5.1 – Entrada de dados.....	49

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

5.2 – Saída de dados.....	50
CAPITULO 6.....	53
ESTRUTURAS DE CONTROLE	53
6.1 – Estruturas de decisão	53
6.1.1 – Estruturas de decisão simples	54
6.1.2 – Estruturas de decisão compostas.....	58
6.1.3 – Estruturas de decisão aninhadas	61
6.1.4 – Estruturas de decisão de múltipla escolha.....	63
6.1.4 – Exercícios para fixação.....	67
6.2 – Estruturas de repetição	68
6.2.1 – Estruturas de repetição com teste no inicio	69
6.2.2 – Estruturas de repetição com teste no fim.....	70
6.2.3 – Estruturas de repetição com variável de controle	73
6.2.4 – Exercícios para fixação.....	74
CAPITULO 7.....	75
ESTRUTURAS DE DADOS – VETORES E MATRIZES	75
7.1 – Vetores ou Arrays	75
7.2 – Matrizes	80
CAPITULO 8.....	84
PROCEDIMENTOS E FUNÇÕES	84
8.1 – Procedimentos	85
8.2 – Funções	91
8.3 – Escopo de variáveis	93
CAPITULO 9.....	95
LÓGICA DE PROGRAMAÇÃO COM C#	95
9.1 – Tipos de dados em C#	130
9.2 – Declarando Variáveis em C#	132
9.3 – Constantes em C#	133

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

9.4 – Operadores de atribuição em C#	134
9.5 - Operadores aritméticos em C#	135
9.6 – Operadores relacionais em C#	136
9.7 - Operadores Lógicos em C#	137
9.8 - Entrada e Saída de dados em C#	147
9.9 – Estruturas de decisão em C#	148
9.10 – Estruturas de repetição em C#	160
9.11 – Vetores e matrizes em C#	173
9.12 – Procedimentos e funções em C#	187
CAPITULO 10	206
LÓGICA DE PROGRAMAÇÃO COM VB.NET	206
10.1 – Tipos de dados em VB.NET	230
10.2 – Declarando Variáveis em VB.NET	231
10.3 – Constantes em VB.NET	233
10.4 – Operadores de atribuição em VB.NET	233
10.5 - Operadores aritméticos em VB.NET	234
10.6 – Operadores relacionais em VB.NET	235
10.7 - Operadores Lógicos em VB.NET	237
10.8 - Entrada e Saída de dados em VB.NET	238
10.9 – Estruturas de decisão em VB.NET	239
10.10 – Estruturas de repetição em VB.NET	245
10.11 – Vetores e matrizes em VB.NET	261
10.12 – Procedimentos e funções em VB.NET	280

CAPITULO 1

INTRODUÇÃO – CONCEITOS INICIAIS

Analise as seguintes afirmativas:

- O livro esta no armário.
- O armário esta fechado.
- É necessário primeiro abrir o armário para depois pegar o livro.

Esse exemplo comprova que sempre que pensamos ou fazemos algo a lógica nos acompanha. Podemos perceber a importância da lógica na nossa vida, não só na teoria, mas também na pratica, já que sempre que queremos pensar, falar, escrever e agir corretamente precisamos colocar “ordem no pensamento”, isto é utilizar a lógica.

Então podemos definir a lógica como técnica de encadear os pensamentos para atingir um determinado objetivo.

Usaremos a lógica de forma a organizar e resolver problemas que desejamos programar.

Perceba a importância do objetivo. Se você não tiver um objetivo não vai conseguir definir a seqüência necessária para alcançá-lo. Isso pode parecer básico, mas é muito importante e muitas vezes a falta de foco no objetivo atrapalha no desenvolvimento dos programas. Já vi muitos programadores “experientes” perderem um bom tempo no desenvolvimento de uma rotina exatamente por

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

perderem o foco no objetivo da mesma. Procure sempre deixar bem claro seu objetivo e definir a seqüência lógica antes de começar a programar, isso vai lhe poupar um bom tempo e vai aprimorar sua lógica. No começo ou para problemas difíceis pegue uma folha e anote a seqüência lógica, se necessário faça os diagramas conforme aprenderá neste curso.

Como você sabe, nosso raciocínio é algo abstrato e intangível, temos a capacidade de expressá-lo através das palavras e escrita que é baseado em um determinado idioma. Este idioma nada mais é do que um conjunto de regras e padrões (gramática). O que quero que você compreenda aqui é **que não importando o idioma sempre seguimos uma ordem de raciocínio, e o idioma é apenas a forma que utilizamos para expressar esse raciocínio.**

Da mesma forma ocorre na lógica da programação, temos varias formas de expressar nosso raciocínio em diversas linguagens de programação. Estas linguagens são muito atreladas a recursos dos computadores que pouco tem a ver com o raciocínio original, por isso utilizaremos durante o curso os **algoritmos** que nos ajudarão a expressar nosso **raciocínio independente da linguagem utilizada para se programar.**

O algoritmo pode ser definido como uma seqüência de passos necessários à execução de um objetivo.

Por exemplo, para fazer pipoca no microondas você precisa seguir os seguintes passos:

- Pegar o pacote de pipoca;
- Abrir o microondas;

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- Colocar o pacote no microondas;
- Fechar o microondas;
- Apertar o botão pipoca;

Esse conjunto de passos para fazer pipoca de microonda é um algoritmo. Um algoritmo para fazer pipoca de microondas.

Também fiquei com vontade de comer pipoca, mas voltando ao nosso assunto, com o exemplo da pipoca percebemos que embora pouco usuais, **os algoritmos são muito comuns em nossos programas e que podemos organizar de forma lógica e seqüencial toda tarefa que formos realizar desde que tenhamos um objetivo bem definido.**

Muitas vezes realizamos este tipo de atividade inconscientemente, sem percebermos os detalhes que nos levam a alcançar o objetivo proposto.

Existem muitas formas de se resolver um problema, afinal cada pessoa pensa e age de maneira diferente. Então podemos ter mais de uma solução correta para um problema, a solução correta é a que atinge o objetivo. O bom senso e a prática indicarão qual a solução mais apropriada, que com **menor esforço e maior objetividade** produz o resultado esperado.

Para você compreender melhor e fechar o conteúdo deste capítulo vamos fazer um algoritmo um pouco mais complexo, segue o problema:

Três jesuítas e três canibais precisam atravessar um rio usando **um barco** com capacidade para **duas pessoas**. Por segurança, não se deve deixar que em

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

nenhuma margem o número de canibais seja maior que o de jesuítas. (Senão os canibais podem comer o jesuíta sozinho.)

Vamos à solução:

- Atravessar um jesuíta em um canibal.
- Voltar o jesuíta com o barco.
- Atravessar dois canibais.
- Voltar um canibal com o barco.
- Atravessar dois jesuítas.
- Voltar um jesuíta e um canibal.
- Atravessar dois jesuítas.
- Voltar um canibal.
- Atravessar dois canibais.
- Voltar um canibal.
- Atravessar dois canibais.

Esse é o nosso algoritmo ele representa uma linha de raciocínio que pode ser descrita de varias maneiras, tanto gráficas como textuais. Até agora usamos a forma textual, no próprio português colonial, no entanto durante o curso você vai aprender uma técnica melhor para expressar os seus algoritmos, chamada de **PSEUDOCÓDIGO**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

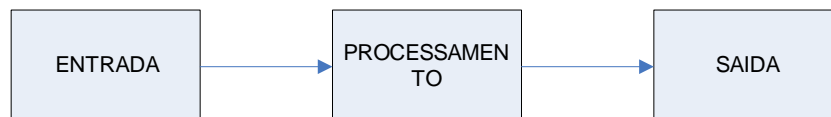
CAPITULO 2

ALGORITMO

Para escrever um algoritmo você precisa descrever a seqüência de instruções de maneira simples e objetiva. Para isso siga as seguintes instruções:

- Use somente um verbo por frase.
- Imagine que esta desenvolvendo um algoritmo para quem não trabalha com informática.
- Use frases curtas e simples.
- Seja objetivo.
- Não use palavras que tenham sentido dúbio ou duplo sentido.

Ao criar um algoritmo divida o problema apresentado em três fases:



Entrada representa a entrada de dados no algoritmo.

Processamento são os procedimentos utilizados para chegar ao resultado final.

Saída são os dados já processados.

Vamos a mais um exemplo, neste vamos fazer um algoritmo que calcule a média aritmética entre quatro números. Para isso primeiramente vamos fazer três perguntas:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

1 – Quais os dados de entrada?

2 – Qual será o processamento?

3 – Quais os dados de saída?

Os dados de entrada serão os quatro números que iremos utilizar para calcular a média. Chamaremos de N1, N2, N3, N4.

O processamento será a soma dos quatro números divididos por quatro.

O dado de saída será a média ou resultado final.

Vamos ao nosso algoritmo:

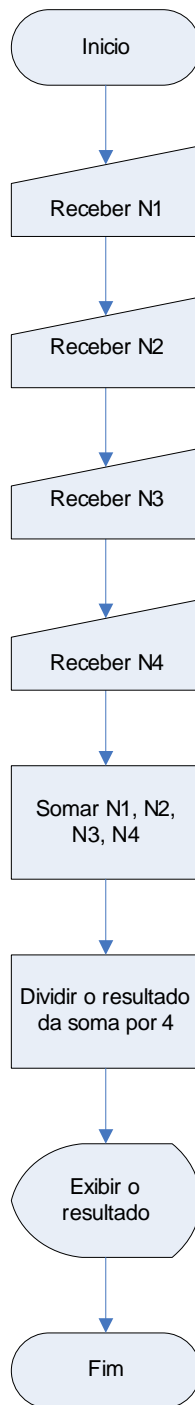
- Receber N1.
- Receber N2.
- Receber N3.
- Receber N4.
- Somar N1,N2,N3 e N4.
- Dividir o resultado da soma por quatro.
- Mostrar o resultado da divisão.

Verifique agora o seguinte diagrama:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"




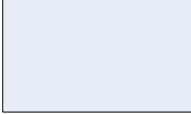

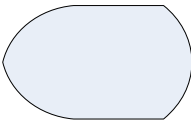
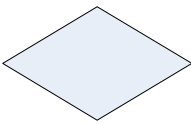
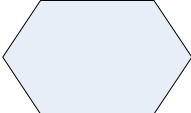

Com você já percebeu, este diagrama representa o algoritmo que acabamos de fazer. Esse tipo de diagrama é conhecido como **DIAGRAMA DE BLOCO** ou **FLUXOGRAMA**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

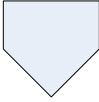
No quadro abaixo você vai conhecer os principais símbolos para a criação de um DIAGRAMA DE BLOCO ou FLUXOGRAMA e o que eles representam:

SÍMBOLO	NOME	FUNÇÃO
	TERMINAL	Indica o início ou fim de um algoritmo.
	PROCESSAMENTO	Representa um processamento
	ENTRADA MANUAL DE DADOS	Indica entrada de dados manual, através de um teclado por exemplo.
	EXIBIR	Representa a exibição dos dados e informações, através de um monitor por exemplo.
	DECISÃO	Representa um teste lógico que escolhe qual instrução será executado.
	PREPARAÇÃO	Representa uma ação de preparação para o processamento.
	CONECTOR	Utilizado para interligar partes de um fluxograma ou para desviar o fluxo

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		corrente para um determinado trecho do fluxograma.
	CONECTOR DE PÁGINAS	Utilizado para interligar partes do fluxograma em páginas distintas.

Como viu no diagrama que criamos somente o símbolo não significa muita coisa, precisamos sempre escrever algo nele que represente a sua função no nosso algoritmo.

Eu uso o **Microsoft Visio** para fazer meus diagramas. O **template** que uso é o **Basic Flowchart (Metric)**, que fica na categoria **Flowchart**.

Vou mostrar rapidamente como criar esse tipo de diagrama no Microsoft Visio, mas fique a vontade para criar os seus onde desejar, incluindo o Microsoft Word ou até mesmo uma folha de papel em branco. Muitas vezes eu crio meus diagramas rapidamente em uma folha em branco mesmo apenas para me certificar ou organizar o código que vou criar.

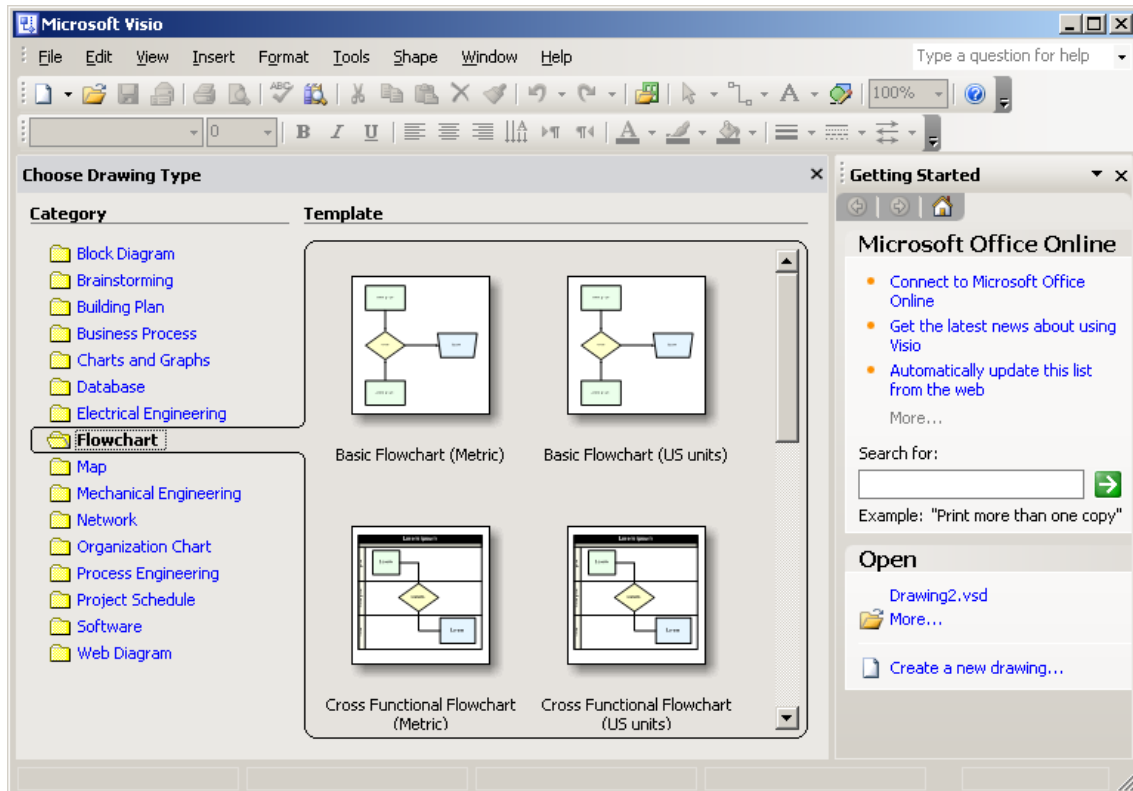
O Microsoft Visio não vem com o Office em sua versão padrão, ele é adquirido separado. No entanto, depois de instalado você pode encontrá-lo junto com os demais programas do Office. Para este exemplo vou usar o Microsoft Office 2003 assim como o Microsoft Visio da mesma versão.

Entre no Microsoft Visio, para isso clique em **Iniciar > Programas > Microsoft Office > Microsoft Visio**.

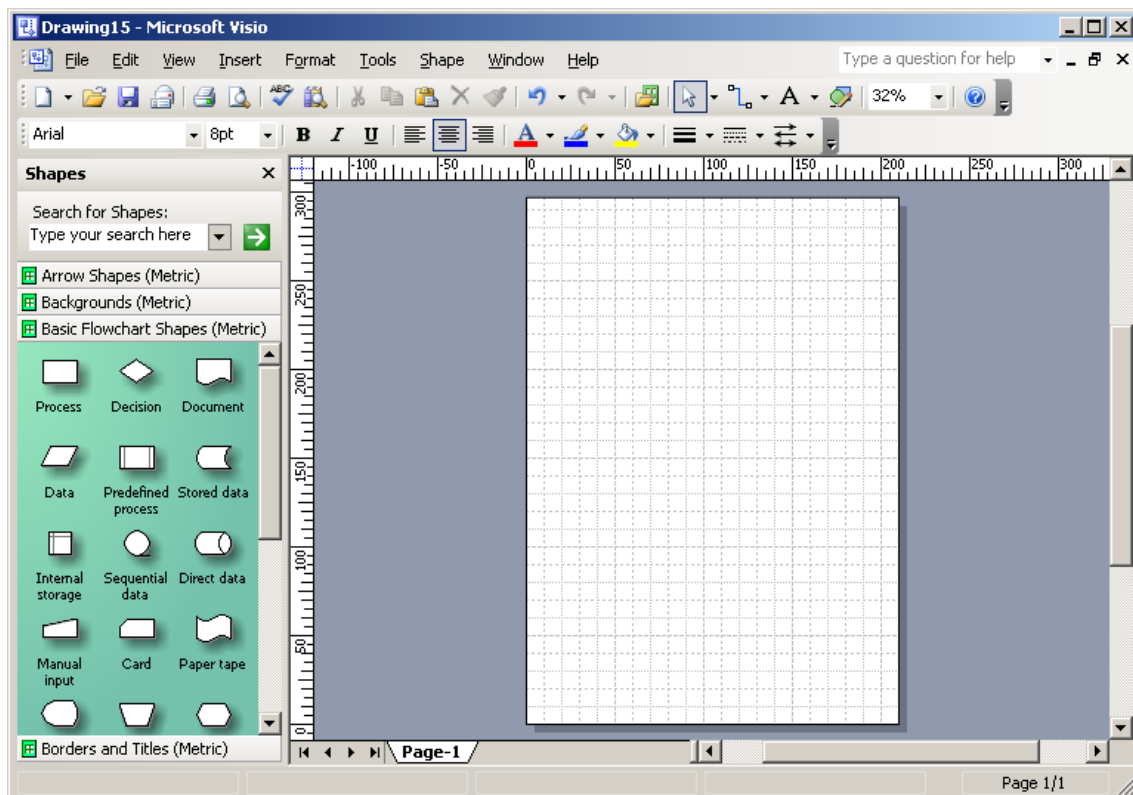
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Selecione **Flowchart** em **Category** como a figura acima e clique em **Basic Flowchart (Metric)**.



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Perceba na figura acima a janela a esquerda chamada **Shapes**, nela você encontra os símbolos que aprendeu logo acima. O nome deles na **Shapes** é **Terminator**, **Process**, **Manual Input** e **Display** respectivamente como a tabela.

Para montar seu diagrama basta arrastar os símbolos para o palco.

Vamos montar o diagrama do exemplo que fizemos logo acima. Arraste para o palco:

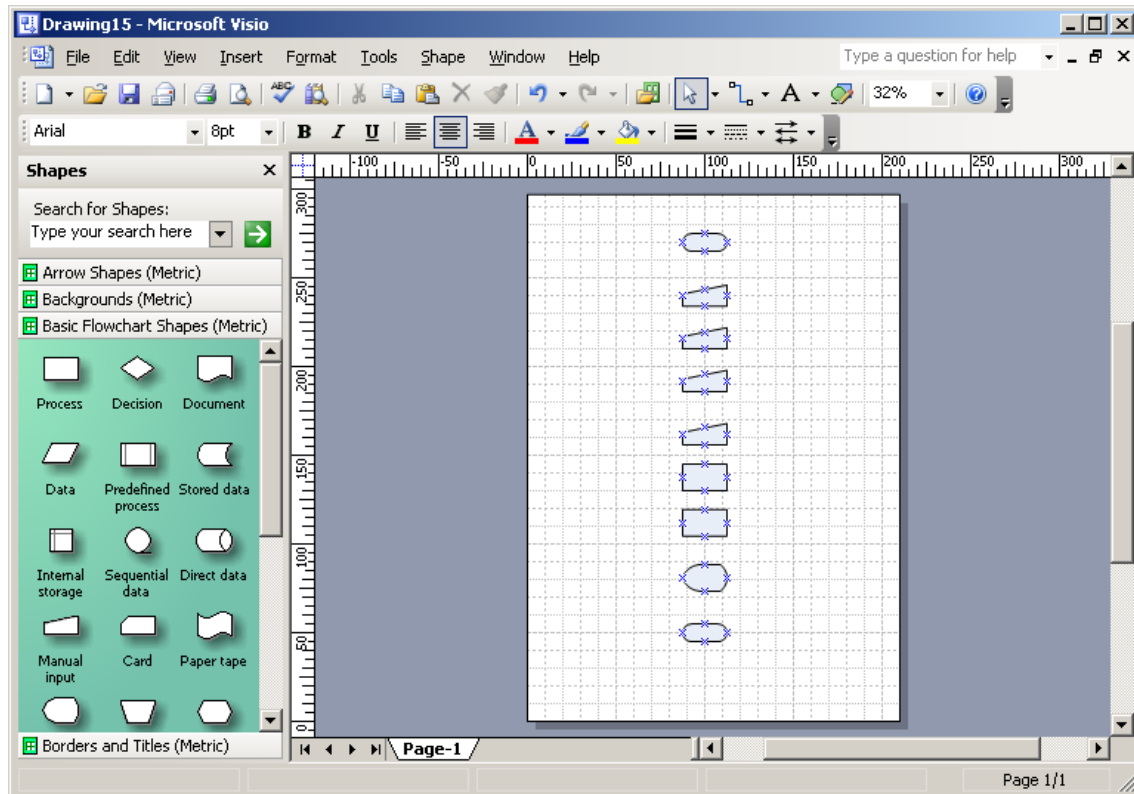
2 Terminal (terminator).

4 Entrada Manual de dados (manul input)

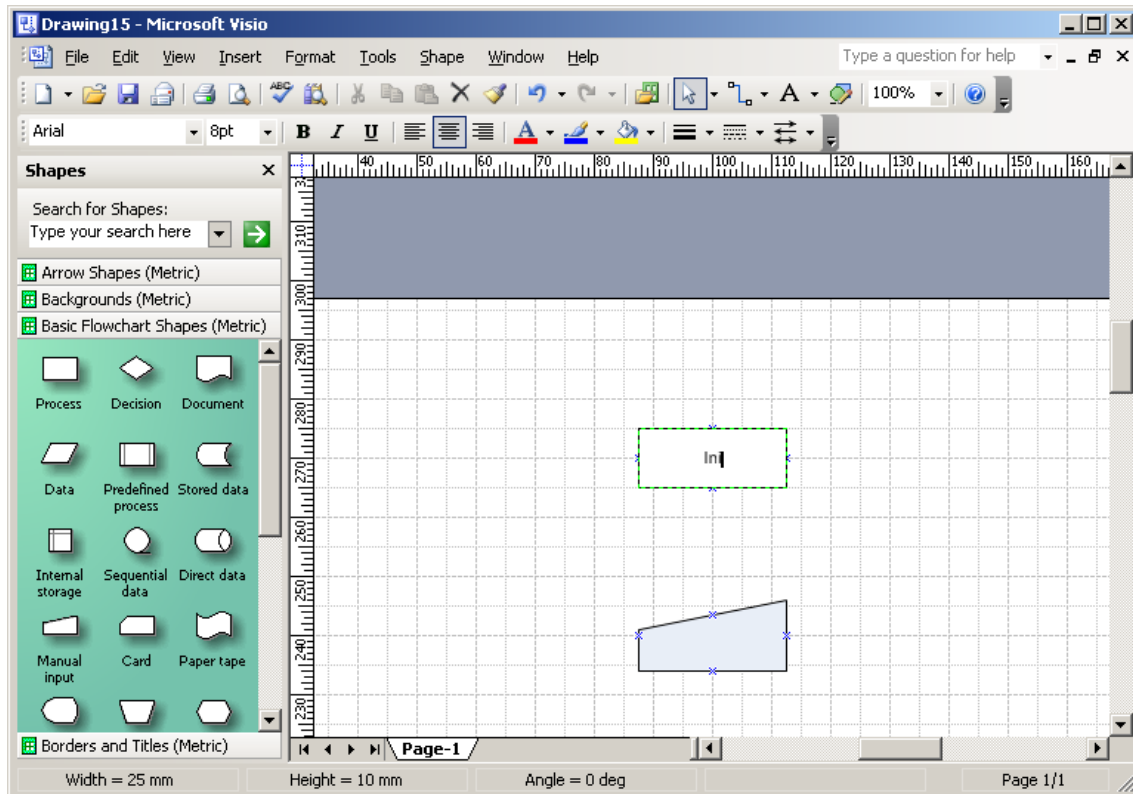
2 Processamento (process)

1 Exibir (Display)

Organize-os como a figura abaixo:

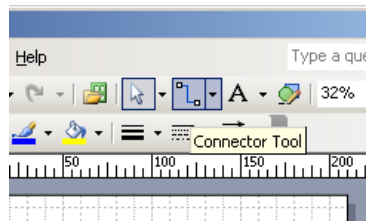


Para digitar um texto dentro do símbolo, de um clique duplo sobre o símbolo desejado. Veja a imagem abaixo, dei um clique duplo sobre o primeiro terminal.



Após digitar apenas clique fora do símbolo para o zoom voltar como estava.

Para você ligar um símbolo ao outro use a ferramenta **connector tool** que se encontra na barra de ferramentas. Como na imagem abaixo.

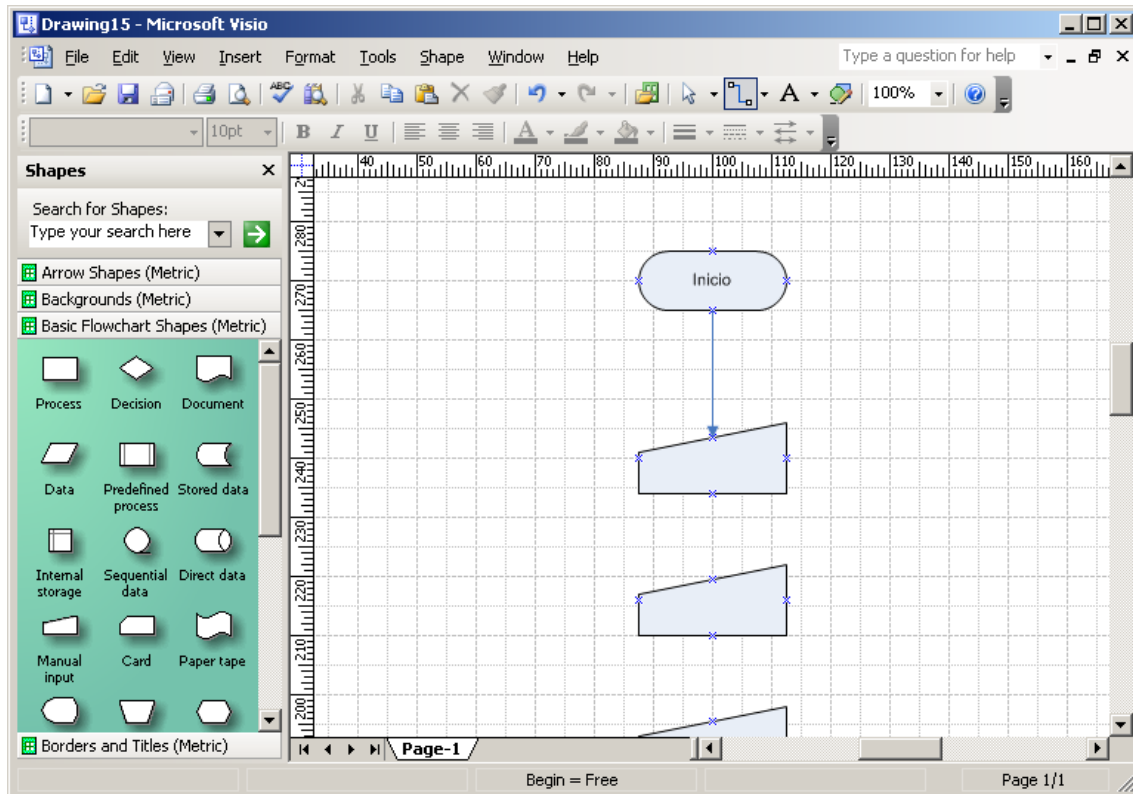


Após selecionar a ferramenta você pode ligar um símbolo ao outro, clicando sobre o primeiro e mantendo o botão pressionado arrastar até o outro. Veja a próxima imagem.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Agora que você já sabe como utilizar o Microsoft Visio para criar seus diagramas de bloco vamos a mais um exemplo para fixar o conteúdo até aqui. Vamos a um exemplo mais parecido com o que temos que fazer no mundo da programação embora ainda um problema bem “pequeno” comparado ao que enfrentamos no dia-a-dia, mas, vamos lá.

Considere o seguinte: precisamos fazer um algoritmo que calcule a comissão de venda de um vendedor. Para isso temos os seguintes dados:

- A comissão é de 5%.
- Você tem o código do vendedor.
- Tem também o código da venda, que recupera o valor total da mesma.

Nosso algoritmo:

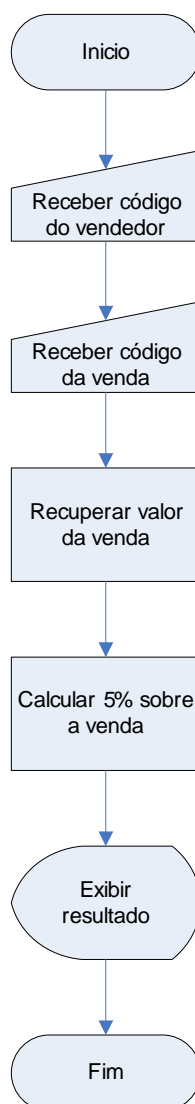
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

- Entrar com o código do vendedor.
- Entrar com o código da venda.
- Recuperar valor da venda.
- Calcular 5% do valor total da venda.
- Exibir resultado.

Vamos ao diagrama de bloco do nosso algoritmo:



Para finalizar este capítulo vamos aprender sobre PSEUDOCÓDIGO.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Visando a criação de um algoritmo com uma linguagem flexível, intermediária entre a linguagem natural e a linguagem de programação, utilizamos o PSEUDOCÓDIGO.

O exemplo a seguir mostra um algoritmo em PSEUDOCÓDIGO:

```
Algoritmo Exemplo1
Var
    codVendedor, codVenda: inteiro
    valorVenda, comissao: real
Inicio
    Mostrar ("Qual o código do Vendedor")
    Ler (codVendedor)
    Mostrar ("Qual o código da Venda")
    Ler (codVenda)
    valorVenda ← RetornaValorVenda(codVendedor, codVenda)
    comissao ← (5 * valorVenda) / 100
    Mostrar (comissao)
Fim.
```

Note que na primeira linha identificamos o algoritmo.

A segunda e terceira usamos para fazer a declaração de variáveis. Você sempre declara as variáveis que vai usar no seu algoritmo após a palavra-chave **var**.

Entre **Início** e **Fim** temos então o corpo do algoritmo.

Segue algumas regras para a criação de PSEUDOCÓDIGO:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- A primeira linha sempre deve identificar o algoritmo.
- Não se devem usar espaços entre as letras no nome do algoritmo, para um cadastro de cliente, por exemplo, você pode colocar `cadastroCliente` ou `cadastro_Cliente`.
- Não iniciar o nome com números.
- Não utilizar palavras reservadas, isto é, palavras usadas para representar ações específicas. Exemplo: **variáveis** (palavra que representa a área de declaração de variáveis).
- Não utilizar caracteres especiais como acentos, símbolos (? / : @ # etc.), ç, entre outros.
- Não utilizar nomes iguais para representar variáveis diferentes.
- Ser sucinto e utilizar nomes coerentes.

Nos próximos capítulos você vai aprender mais sobre a criação de PSEUDOCÓDIGO, e passaremos a utilizá-lo no nosso curso para representar de forma textual nossos algoritmos. O PSEUDOCÓDIGO ajuda a evitar erros de interpretação e comunicação que podem acontecer quando apenas descrevemos de forma textual, além impõe regras que nos ajudam a implementar melhor o algoritmo em nosso programa.

CAPITULO 3

VARIÁVEIS, TIPOS DE DADOS E CONSTANTES

Pense em uma caixa que você usa para guardar suas coisas, qualquer caixa, pode ser uma de papelão. Concorda comigo que a quantidade de coisas que você pode “guardar” ou armazenar nesta caixa é proporcional ao tamanho da caixa correto? Você não pode guardar na caixa algo maior do que ela. Concorda?

As variáveis de forma geral podem ser comparadas a caixas. Elas guardam informações na memória do computador. Já os tipos de dados são usados para dizer que tamanho terá a nossa variável ou quanto ela poderá armazenar.

Então sempre que você cria uma variável você na verdade esta dizendo ao computador que esta reservando ou alocando um espaço na memória para armazenar algo. Para que o computador saiba quanto ele precisa reservar de espaço você precisa informar o tipo de dado quando esta criando a sua variável. Você também precisa dar um nome único a sua variável para poder referenciar ela em seu programa, ou seja, para que possa atribuir e usar os valores dela.

Antes de aprender a sintaxe de como declarar suas variáveis vamos aprender sobre os tipos de dados.

3.1 – Tipos de dados

Dados são os valores que iremos usar nos nossos programas para resolver nossos problemas. Se você precisa fazer uma simples adição, como $45 + 20$, os valores 45 e 20 são **dados** que você precisa ter para realizar a operação.

Aproximando-nos da maneira que o computador manipula as informações, podemos dividir os dados nos seguintes tipos:

- **Inteiro** - é toda informação numérica que pertença ao conjunto de números inteiros relativos.
- **Real** - é toda informação numérica que pertença ao conjunto dos números reais.
- **Caracter** ou **Literal** - é toda informação composta por um conjunto de caracteres alfanuméricos: numéricos (0...9), alfabéticos (A...Z, a...z) e especiais (#,?,!,@).
- **Lógico** - é toda informação que pode assumir apenas duas situações, verdadeiro ou falso.

Os tipos de dados acima são conhecidos como **tipos primitivos**. Eles são os tipos básicos usados na construção de algoritmos.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Quando você precisar armazenar números inteiros em suas variáveis você as declara como **Inteiro**. Para armazenar valores monetários ou números com casas decimais você usa o tipo **Real**. Para armazenar textos você usa o **Literal** o **Caracter** e finalmente para armazenar apenas um valor lógico, ou seja, verdadeiro ou falso você usa o tipo **Lógico**.

Uma variável do tipo **Real** vai usar um espaço maior na memória do que uma variável do tipo **Inteiro**, porque o número que ela pode armazenar pode precisar de mais espaço do que o que é disponibilizado para uma variável do tipo inteiro. Isso quer dizer que ao usar tipos de dados errados você pode ou desperdiçar espaço na memória ou ter problemas na hora que for armazenar o dado por não caber no espaço alocado. Isso gera um erro no programa.

Outra informação importante. Cada linguagem de programação tem suas particularidade em relação aos tipos de dados primitivos, por isso vou mostrar no curso o que cada variável pode armazenar para cada tipo de dado em sua respectiva linguagem. De forma geral todas elas dividem os tipos em categorias, como as que você acabou de aprender.

Como os dados manipulados pelos computadores durante a execução dos programas são armazenados na memória, esses tipos de dados seguem as características de formato disponível nessa memória, ou seja, são organizados em bits e bytes, por exemplo, para representar um número inteiro poderiam ser usados dois bytes ou 16 bits que é a mesma coisa. Isso resultaria em 2^{16} combinações possíveis ou 65.536 possibilidades. Lembrando que os números podem assumir valores positivos ou negativos, nosso número inteiro poderia armazenar um valor que vai de -32.768 a 32.767.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

3.2 – Declarando Variáveis

Como você já aprendeu no início deste capítulo, quando declaramos uma variável estamos alocando um espaço na memória do computador para a mesma.

Para declarar uma variável você precisa saber qual tipo de dado ela vai armazenar e qual será o **identificador** dela. Além disso, não pode usar como nome de suas variáveis uma palavra reservada.

Identificador é o nome da sua variável. É através dele que você usa a sua variável.

Lembre-se destas três **regras** quando estiver definindo o nome da sua variável:

- Deve começar por um caractere alfabético;
- Podem ser seguidos por mais caracteres alfabéticos ou numéricos;
- Não devem ser usados espaços ou caracteres especiais;

Segue algumas **dicas** para você nomear suas variáveis melhor:

- Evite usar underline;
- Não crie variáveis que apenas se diferenciem apenas pela sua forma.
Exemplo: *minhaVariavel* e outra chamada *MinhaVariavel*;
- Procure iniciar o nome com uma letra minúscula;
- Evite usar todas as letras maiúsculas;
- Quando o nome tiver mais que uma palavra, a primeira letra de cada palavra após a primeira deve ser maiúscula (conhecido como notação camelCase);

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- Não use notação Húngara.

Você também pode usar as convenções **PascalCasing** e **camelCasing** na nomeação das suas variáveis.

Convenção PascalCasing: para usar a convenção **PascalCasing** para nomear suas variáveis, capitalize o primeiro caractere de cada palavra. Exemplo:

```
InicializandoDados
```

Recomenda-se usar o **PascalCasing** quando estiver nomeando procedimentos e funções, você vai aprender sobre elas no capítulo 8.

Convenção camelCasing: para usar esse tipo de convenção, capitalize a primeira letra de cada palavra menos da primeira. Como o exemplo:

```
precoMedioDoTerreno
```

Recomenda-se essa convenção na nomeação de variáveis que usam tipos primitivos.

Exemplos de identificadores válidos:

```
nome, x, hi123, k1, notas, media, M, ABC, FGTS
```

Exemplos de identificadores inválidos:

```
5x, e(1), a:g, x-y, preço/3, asd*, pp&hh
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Exemplo de declaração de variáveis em PSEUDOCÓDIGO:

```
idade: inteiro  
valor, x: real  
nome, telefone, endereco: literal  
pf: lógico
```

Perceba nos exemplos acima que podemos declarar mais de uma variável do mesmo tipo junto, basta separa-las por vírgula. **Em Pseudocódigo você também precisa criar suas variáveis no começo do algoritmo após a palavra-chave Var.**

3.3 – Constantes

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Constante é como uma variável, a diferença é que o seu valor não é alterado durante a execução do programa, você pode armazenar o valor da taxa de juros, por exemplo, e sempre que precisar desse valor você usa a constante.

Você também precisar dizer qual tipo de dado precisará ao declarar sua constante.

Em PSEUDOCÓDIGO as constantes devem ser declaradas como variáveis cujo valor atribuído permanecerá inalterado ao longo do programa.

3.4 – Case-sensitive

Para finalizar esse capítulo quero apenas que você compreenda mais este conceito:

O que é uma linguagem **Case-sensitive**?

Para compreender compare a diferença entre as seguintes palavras:

- Professora
- PROFESSORA
- ProFeSSora
- professora

Para nós as quatro palavras são iguais. Para uma linguagem que não é case-sensitive também. Mas para as linguagens que são case-sensitive as quatro palavras são interpretadas diferentes, isso porque **essas linguagens diferenciam letras maiúsculas de minúsculas para seus identificadores.**

Isso quer dizer que para uma linguagem case-sensitive se você declarasse quatro variáveis conforme as palavras acima, elas seriam quatro variáveis diferentes. Um exemplo de linguagem case-sensitive é o C#, uma variável chamada **nome** é diferente de outra chamada **NOME** em C#.

Já em VB.NET se você declara uma variável chamada nome você não pode declarar outra chamada NOME. Isso gera um erro, porque ele entende que você

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

esta declarando duas variáveis com o mesmo nome, isso porque o VB.NET não é case-sensitive.

3.5 – Exercícios para fixação

1) Classifique os dados de acordo com o seu tipo, sendo (**I** = inteiro, **R** = real, **C** = caracter, **L** = lógico).

- | | |
|-------------|--------------|
| () + 56 | () + 0,05 |
| () "+4567 | () ".V." |
| () F | () F |
| () 1 | () -1 |
| () + 35 | () + 234 |
| () "+6677" | () V |
| () 'F' | () -12 |
| () 0,0 | () "a" |
| () - 0,001 | () "abc" |
| () "-0,0" | () -1,9E123 |
| () ".V." | () '0' |
| () 0 | |

2) Assinale os identificadores válidos:

() abc

() Etc...

() AB/C

() ...a

() "João"

() xyz

() [x]

() Congresso_Internacional

() 123a

() A_B-C

() 080

() U2

() 1 a 3

() p{0}

() (x)

() A123

() #55

() A.

() AH!

CAPITULO 4

OPERADORES

Operadores são usados para representar expressões de **cálculo**, **comparação**, **condição** e **atribuição**. Portanto, temos os seguintes tipos de operadores:

- De atribuição;
- Aritméticos;
- Relacionais;
- Lógicos.

4.1 – Operadores de atribuição

Os operadores de atribuição são usados para expressar o armazenamento de um valor em uma variável. Esse valor pode ser predefinido e pode ser também o resultado de um processamento.

Exemplo:

```
nome ← "Fulano de tal"
```

Lê-se *nome* **recebe** *fulano de tal*. O exemplo acima representa a atribuição do valor ***Fulano de tal*** na variável *nome*.

Mais alguns exemplos:

```
valor ← 3
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

resultato $\leftarrow 45 + 30$

codigoCliente $\leftarrow 134$

4.2 - Operadores aritméticos

Os operadores aritméticos são usados na representação de cálculos matemáticos, são eles:

Operador	Representação em PSEUDOCÓDIGO	Exemplo
Incremento	Utiliza-se uma expressão. Exemplo: $a + 1$	$a \leftarrow a + 1$ Neste caso é somado 1 ao valor de a.
Decremento	Utiliza-se uma expressão. Exemplo $a - 1$	$a \leftarrow a - 1$ Neste caso é subtraído 1 do valor de a.
Multiplicação	*	$a * b$ Multiplica-se o valor de a por b.
Divisão	/	a / b Divide-se o valor de a por b.
Exponenciação	\wedge ou **	$2 \wedge 3$ Aqui representamos 2 elevado a 3 ou 2^3 .
Módulo	Mod	$a \text{ mod } b$

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		Retorna o resto da divisão de a por b.
Adição	+	a + b Soma de a com b.
Subtração	-	a - b Subtração de a com b.

4.3 – Operadores relacionais

Primeiramente vamos entender o que é uma EXPRESSÃO CONDICIONAL. Esta é uma expressão que sempre retorna um valor booleano, ou seja, VERDADEIRO ou FALSO.

Exemplo:

Preço é menor que 100.

Se preço for menor que 100 então o resultado da expressão acima é VERDADEIRO. Caso contrário o resultado é FALSO.

Você pode usar os seguintes operadores relacionais para fazer comparações:

Operador	Representação em PSEUDOCÓDIGO	Exemplo
Maior	>	a > b

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		Se o valor de a for maior que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Maior ou igual	\geq	$a \geq b$ Se o valor de a for maior ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Menor	$<$	$a < b$ Se o valor de a for menor que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Menor ou igual	\leq	$a \leq b$ Se o valor de a for menor ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Igual a	$=$	$a = b$ Se o valor de a for igual ao valor de b então o resultado dessa expressão

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		é verdadeiro senão é falso.
Diferente de	<>	a = b Se o valor de a for diferente do valor de b então o resultado dessa expressão é <u>verdadeiro</u> senão é falso.

4.4 – Operadores Lógicos

Antes de falar sobre operadores lógicos vamos entender o que é uma **expressão lógica**. Elas sempre retornam também um valor booleano, ou seja, verdadeiro ou falso, para compreender melhor vamos ver um exemplo:

Tiago tem 20 anos e Pedro tem 30 anos.

Perceba que o e une duas expressões condicionais. O resultado acima será verdadeiro de Tiago tiver 20 anos e Pedro 30. Senão será falso. Se tiago tiver 20 anos e Pedro 31 por exemplo o resultado será falso.

O e no nosso exemplo é conhecido como operador lógico. Você pode usar os seguintes operadores lógicos nos seus programas:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Operador	Representação em PSEUDOCÓDIGO	Exemplo
E	.e.	<p>$a = 5$.e. $b > 9$</p> <p>Caso o valor de a seja igual a cinco e o valor de b maior que nove o resultado é verdadeiro, senão é falso.</p>
Ou	.ou.	<p>$a = 5$.ou. $b > 9$</p> <p>Se o resultado de a for igual a cinco ou o valor de b for maior que nove então o resultado é verdadeiro. O resultado só será falso é as duas expressões retornarem falso.</p>
Não	.não.	<p>.não $a = 5$</p> <p>Se o resultado de a for igual a 5 então o resultado será falso, o operador não inverte o resultado da expressão.</p>

4.5 – Funções matemáticas

Além das operações aritméticas citadas anteriormente, podemos usar nas expressões aritméticas algumas funções da matemática.

Exemplo:

`a ← sen(x)`

No exemplo acima a variável `a` está recebendo como conteúdo o seno de `x`.

A tabela seguinte mostra as funções matemáticas que você usar em seus algoritmos, o valor de `x` pode ser um número, uma variável, expressão aritmética ou também outra função matemática.

Função	Observação
<code>sen(x)</code>	Seno de <code>x</code> .
<code>cos(x)</code>	Coseno de <code>x</code> .
<code>tg(x)</code>	Tangente de <code>x</code> .
<code>arctg(x)</code>	Arco cuja tangente é <code>x</code> .
<code>arccos(x)</code>	Arco cujo coseno é <code>x</code> .
<code>arcsen(x)</code>	Arco cujo seno é <code>x</code> .
<code>abs(x)</code>	Valor absoluto ou módulo de <code>x</code> .
<code>int(x)</code>	A parte inteira de um número fracionário.
<code>frac(x)</code>	A parte fracionaria de <code>x</code> .
<code>ard(x)</code>	Transforma por arredondamento um número fracionário em inteiro.
<code>sinal(x)</code>	Retorna <code>-1</code> , <code>+1</code> ou <code>0</code> conforme o valor de <code>x</code> seja

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

	negativo, positivo ou nulo.
rnd(x)	Valor randômico ou aleatório de x;

4.6 – Precedência dos operadores

Precedência de operadores nada mais é do que a ordem que as expressões são avaliadas quando mais do que um operador é usado em uma expressão, exemplo:

```

10 + 15 * 2 / 4 ^ 2
10 + 15 * 2 / 4 ^ 2
10 + 15 * 2 / 16
10 + 30 / 16
10 + 1.875
11.875

```

Percebe que a precedência aqui é parecida com a precedência em matemática, primeiro foi executado a exponenciação, depois a multiplicação, depois a divisão e então a soma.

Assim como em matemática você pode e deve usar os parênteses para orientar a execução, como no seguinte exemplo:

```

((8 - 5) * 3) ^ 2
((8 - 5) * 3) ^ 2

```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

$(3 * 3) ^ 2$

$9 ^ 2$

81

A seguinte tabela mostra a relação nos operadores e sua precedência:

Operador	Observação
(), []	Parênteses e colchetes são usados para agrupar expressões determinando a precedência, como acontece nas operações matemáticas.
^ ou **	Potenciação.
*, /	Operadores aritméticos de multiplicação e divisão.
+, -	Operadores aritméticos de adição e subtração.
←	Operador de atribuição.
=, <, >, <=, >=, <>	Operadores relacionais.
.não.	Operador lógico de negação.
.e.	Operador lógico e.
.ou.	Operador lógico ou.

4.7 – Tabela-verdade

A tabela-verdade representa o conjunto de todas as possibilidades existentes entre os valores de diversas variáveis ou expressões e operadores lógicos.

A	B	A .e. B	A .ou. B	.não. (A)
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

A tabela verdade acima mostra todos os valores possíveis para **A** e **B**.

Conforme a tabela acima, se o valor de **A** for verdadeiro e **B** for verdadeiro, então o resultado da expressão **A .e. B** será verdadeiro, o valor da expressão **A .ou. B** será verdadeiro e o valor da expressão **.não A** será falso, e assim por diante.

Vamos a um exemplo para que você compreenda melhor. Vamos fazer a tabela verdade da expressão: $a = 2$.e. $b = 5$.

			Operador		
Expressão em algoritmo	a = 2	b = 5	a = 2 .e. b = 5	a = 2 .ou. b = 5	.não. b = 5
Resultados possíveis	V	V	V	V	F
	V	F	F	V	F
	F	V	F	V	V
	F	F	F	F	V

Na tabela verdade podemos constatar que as expressões $a = 2$ e $b = 5$ podem assumir quatro possibilidades. Ou seja, ambas podem ser verdadeiras (primeira linha dos resultados possíveis), a primeira pode ser verdadeira e a segunda falsa, a primeira pode ser falsa e a segunda verdadeira ou ambas podem ser falsas. Essas combinações dependem portanto dos valores que estão em a e b .

Lembrando que uma expressão pode ter mais de um operador lógico aumentando as possibilidades, como a seguinte expressão:

$(a = 5 .e. b > 2) .ou. b <> 20$

4.8 – Exercícios para fixação

1) indique qual o resultado será obtido das seguintes expressões:

- a) $1 / 2$
- b) $1 \text{ DIV } 2$
- c) $1 \text{ MOD } 2$
- d) $(200 \text{ DIV } 10) \text{ MOD } 4$
- e) $5 * 2 + 3$
- f) $6 + 19 - 23$
- g) $3,0 * 5,0 + 1$
- h) $1/4 + 2$
- i) $29,0/7 + 4$
- j) $3/6,0 - 7$

2) Indique o resultado das seguintes expressões:

- a) $2 > 3$
- b) $(6 < 8) \text{ OR } (3 > 7)$
- c) $\text{NOT } (2 < 3)$

CAPITULO 5

COMANDO DE ENTRADA E SAÍDA

Os comandos de entrada e saída representam no nosso algoritmo as entradas e saídas de dados, você já aprendeu a representá-los usando **fluxogramas**, agora vai aprender como representamos os mesmos nos nossos algoritmos usando **pseudocódigo**.

5.1 – Entrada de dados

Para que o algoritmo possa receber os dados de que necessita adotaremos um **comando de entrada** de dados denominado **ler**. A entrada elementar de dados é feita por meio do teclado e representada por:

```
ler (variável)
```

Usamos como no seguinte exemplo:

```
Algoritmo Exemplo2  
Var  
    x: inteiro  
Inicio  
    ler (x)  
Fim.
```

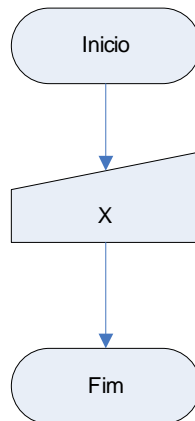
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Esse comando atribui o valor da leitura na variável x.

O Fluxograma do exemplo acima é este:



5.2 – Saída de dados

O dispositivo padrão de saída do computador é o monitor, representamos da seguinte forma:

```
Mostrar (variável)
```

Como no seguinte exemplo:

```
Algoritmo Exemplo3  
  
Var  
  
    x: inteiro  
  
Inicio
```

Autor: Herbert Moroni Cavallari da Costa Gois

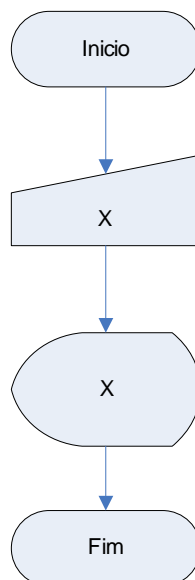
Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

```
x ← 2  
Mostrar (x)  
Fim.
```

Esse comando exibe no monitor o conteúdo da variável x, no caso do nosso exemplo o valor 2.

Segue o fluxograma:



Algoritmo Exemplo4

Var

nome: caracter

Inicio

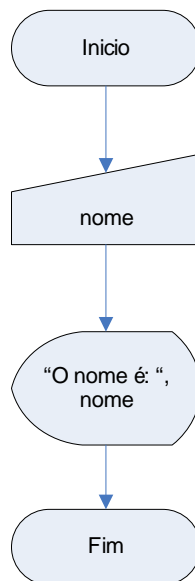
nome ← "Pedro"

Mostrar ("O nome é: ", nome)

Fim.

O exemplo acima imprime na tela a concatenação, ou seja, o agrupamento da string **O nome é:** com o conteúdo da variável **nome**. No caso do nosso exemplo seria exibido na tela **O nome é: Pedro**

Segue o fluxograma:



CAPITULO 6

ESTRUTURAS DE CONTROLE

Como o próprio nome diz, as estruturas de controle servem para controlar a execução dos nossos programas. Elas são divididas em:

- Estruturas de decisão ou seleção;
- Estruturas de repetição;

Portanto dividiremos esse capítulo em dois, a primeira parte tratando das estruturas de decisão e a segunda das de repetição.

6.1 – Estruturas de decisão

Imagine que você chegou à sua casa e vai abrir a porta da frente para entrar. Se a porta estiver aberta você entra correto?

Isto é uma estrutura condicional. O resultado da pergunta se a porta esta aberta pode ser verdadeira ou falsa. E você só vai entrar se a porta estiver aberta, correto? Isso é estrutura condicional simples. Você tem apenas uma pergunta, se for verdade então acontece algo, senão não acontece nada.

Mas como você sabe se a porta da sua casa não estiver aberta para você entrar, você vai tomar alguma outra atitude para entrar. Você pode bater na porta por

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

exemplo para ver se alguém abre ela para você. Nossa estrutura condicional ficaria assim então:

Se a porta estiver aberta então você entra, senão você bate na porta.

Conhecemos uma estrutura condicional como exemplo acima chama composta, porque tem dois blocos de instrução, um é o **entrar se a porta estiver aberta** e o outro é **bater na porta caso ela esteja fechada**.

No nosso dia-a-dia frequentemente tomamos esse tipo de decisão não é mesmo? Agora e se você bater na porta e ninguém abrir?

Neste caso nossa estrutura de decisão não resolveu o problema ainda. Você ainda não é capaz de entrar. O que fazemos geralmente nesse caso é utilizar estruturas condicionais aninhadas, ou seja, uma dentro da outra. Exemplo:

Se a porta estiver aberta então você entrar, senão você bate na porta, se ninguém abrir então você liga para ver se alguém atende ao telefone, senão você vai embora.

Os problemas mais complexos exigem varias estruturas de decisão aninhadas.

6.1.1 – Estruturas de decisão simples

Na introdução sobre estruturas de decisão você já aprendeu o que é uma estrutura de decisão simples. Aqui só quero salientar a sintaxe e os exemplos.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Sintaxe em Pseudocódigo:

```
Se (condição) então
    Conjunto de instruções
Fim-se
```

Exemplos:

```
Algoritmo Exemplo5
Var
    idade: integer
Inicio
    Ler(idade)
    Se (idade >= 18) então
        Mostrar("Maior de idade")
    Fim-se
Fim.
```

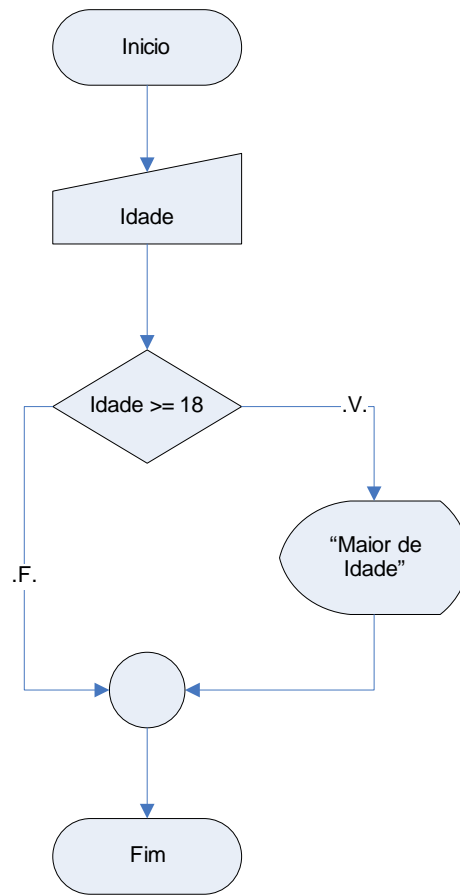
No exemplo acima de se o conteúdo da variável idade for **maior ou igual** a dezoito então será mostrado na tela o texto **Maior de idade**.

Segue o fluxograma:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Algoritmo Exemplo6

Var

 sexo: caracter

Inicio

 Leia(sexo)

 Se (sexo = "F") .ou. (sexo = "f") então

 Mostrar("Sexo feminino")

 Fim-se

Fim.

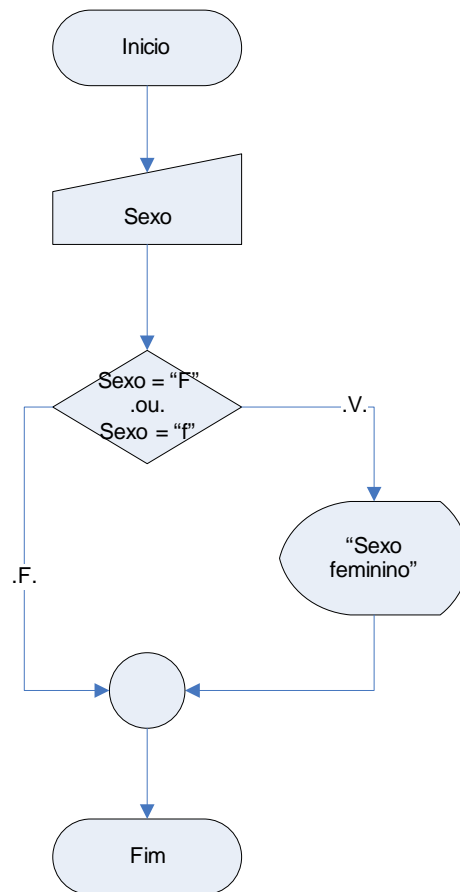
No exemplo acima de se o conteúdo da variável sexo for **igual** a F então será mostrado na tela o texto **Sexo feminino**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Segue o fluxograma:



6.1.2 – Estruturas de decisão compostas

Como você já aprendeu, a estrutura de decisão composta prevê uma condição com dois conjuntos de instruções para serem realizados de acordo com a avaliação da expressão condicional.

Sintaxe no Pseudocódigo:

```
Se (condição) então
    Conjunto de instruções que será realizado se o resultado da
    expressão condicional for verdadeiro
Senão
    Conjunto de instruções que será realizado se o resultado da
    expressão condicional for falso.
Fim-se
```

Exemplos:

```
Algoritmo Exemplo7
Var
    login: caracter
Inicio
    Ler(login)
    Se (login = "moroni") então
        mostrar("usuário válido")
    senão
```

Autor: Herbert Moroni Cavallari da Costa Gois

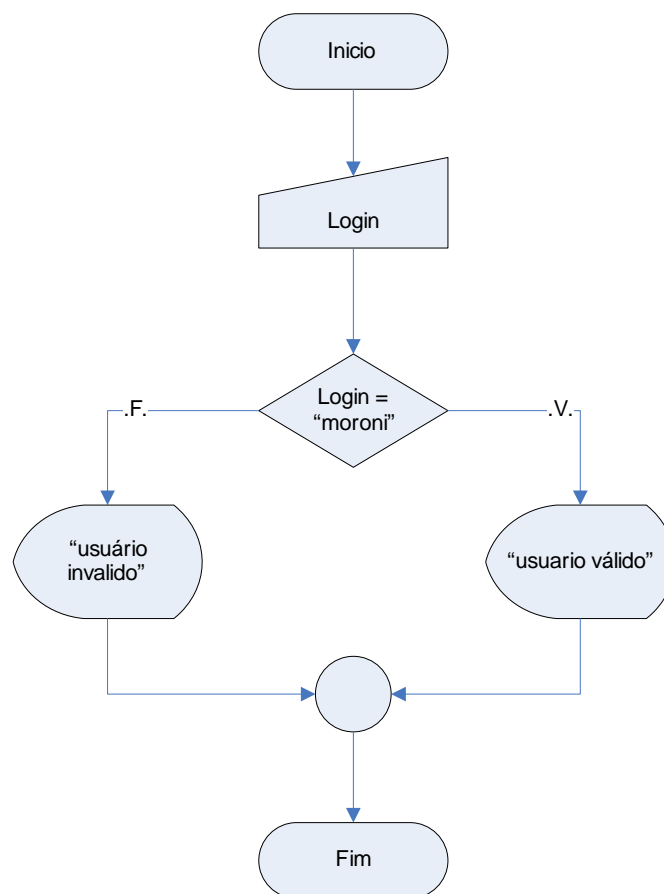
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        mostrar("usuário invalido")  
  
    Fim-se  
  
Fim.
```

No exemplo acima **se** o conteúdo da variável login for **igual** a moroni então vai ser exibido **usuário valido**, senão, ou seja, para qualquer outro valor diferente de moroni será exibido **usuário invalido**.

Segue o fluxograma:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Algoritmo Exemplo8

Var

 saldo: real

Inicio

 Ler (saldo)

 Se (saldo >= 0) então

 mostrar("Saldo positivo")

 senão

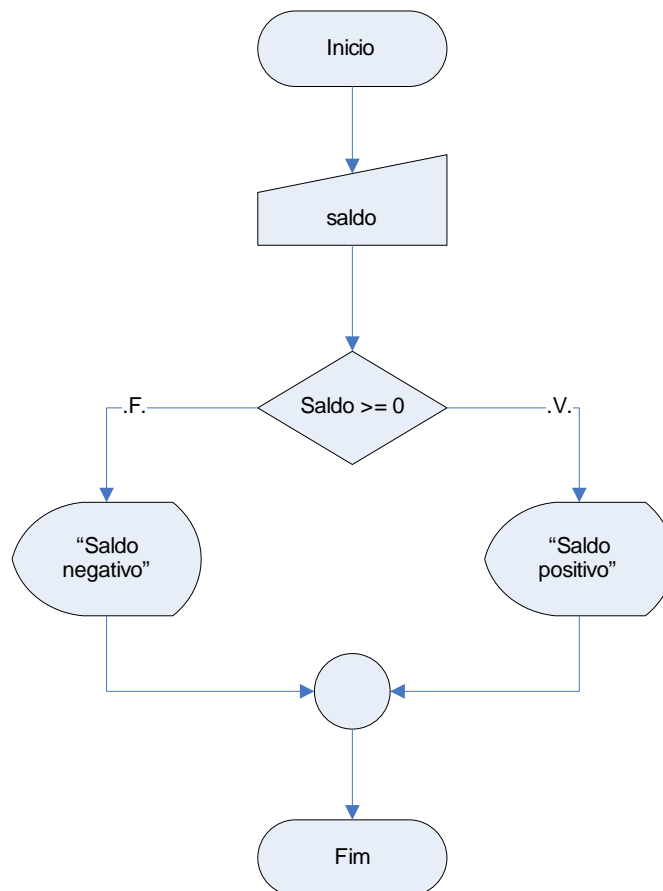
 mostrar("Saldo negativo")

 fim-se

Fim.

No exemplo acima se o conteúdo da variável saldo for **maior ou igual** a 0 então vai ser exibido **Saldo positivo**, senão será exibido **Saldo negativo**.

Segue o fluxograma:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

6.1.3 – Estruturas de decisão aninhadas

Temos estruturas de decisão aninhadas quando temos uma estrutura de decisão dentro de outra.

Estruturas de decisão aninhadas fazem uma seqüência de testes de seleção que poderão ser executados ou não de acordo com o resultado das expressões condicionais.

Sintaxe em pseudocódigo:

```
Se (condição_1) então
    Se (condição_2) então
        Conjunto de instruções A
    Senão
        Conjunto de instruções B
Fim-se
Senão
    Conjunto de instruções C
Fim-se
```

No modelo acima, se a condição_1 resultar verdadeiro, então será realizado o teste da condição_2. O importante aqui é que se o resultado da condição_1 for falso a condição_2 nem é testada, o código já é direcionado para o Senão da primeira estrutura de decisão.

Exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Algoritmo Exemplo9

Var

mês: integer

Início

Ler(mês)

Se (mês = 2 .ou. mês = 4 .ou. mês = 6 .ou. mês = 9 .ou. mês = 11)então

Se (mês = 2) então

Mostrar("Este mês tem 28 dias")

Senão

Mostrar("Este mês tem 30 dias")

Fim-se

Senão

Mostrar("Este mês tem 31 dias")

Fim-se

Fim.

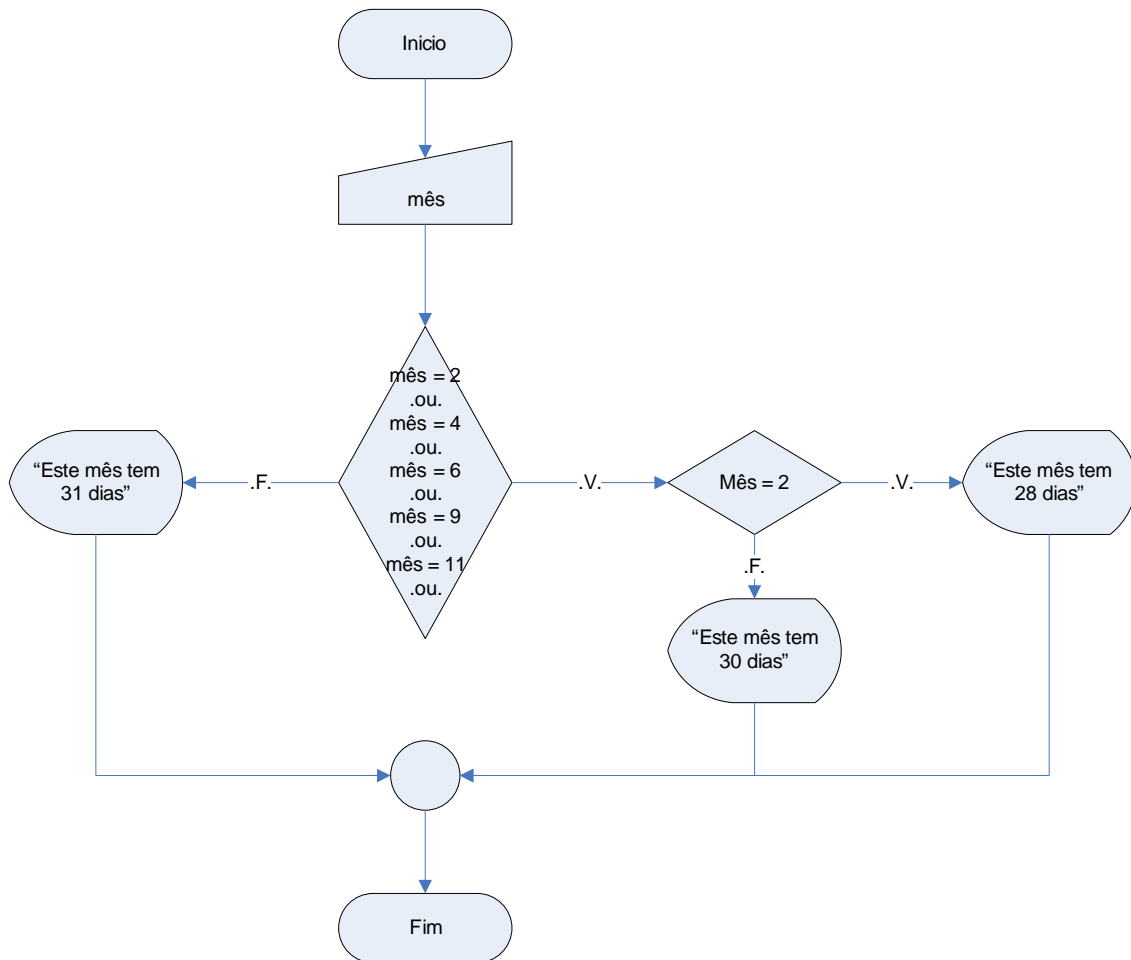
Como você já deve ter percebido a estrutura acima verifica quantos dias tem o mês atribuído a variável mês. O primeiro Se verifica se o mês é igual a 2 ou 4 ou 6 ou 9 ou 11, que são os meses que tem menos de 31 dias. Se o valor de mês não for nenhum desses valores a execução vai para senão e mostra que o mês tem 31 dias. Se o valor de mês for igual a um dos valores então o segundo Se é avaliado. Ele verifica se o mês é 2, ou seja, fevereiro. Se sim, o mês tem 28 dias, senão tem 30 dias. No nosso exemplo eu atribui 2 a variável mês então seria mostrado que o mês possui 28 dias.

Segue o fluxograma:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



6.1.4 – Estruturas de decisão de múltipla escolha

Uma estrutura de decisão de múltipla escolha funciona como um conjunto de opções para escolha.

Sintaxe em pseudocódigo:

Escolha variável

Caso tal_coisa_1

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        Faça instrução a
    Caso tal_coisa_2
        Faça instrução b
    Caso tal_coisa_3
        Faça instrução c
    Caso tal_coisa_4
        Faça instrução d
    Caso contrario
        Faça instrução e
Fim-escolha
```

Exemplo:

```
Algoritmo Exemplo10
Var
    idade: integer
Inicio
    Ler(idade)
    Escolha idade
        Caso idade < 18
            Mostrar("Menor de idade")
        Caso idade >= 18 .e. idade <= 55
            Mostrar("Adulto")
        Caso idade = 0
            Mostrar("Bebê")
        Caso contrario
            Mostrar("Idoso")
    Fim-escolha
Fim.
```

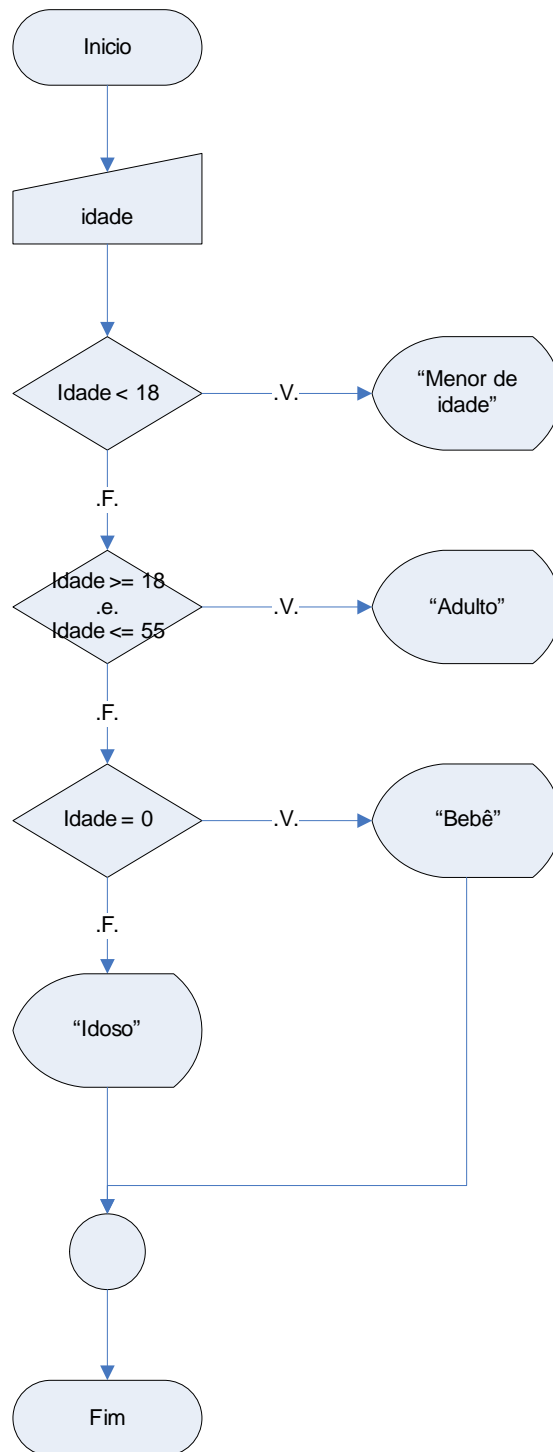
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Perceba que a estrutura de múltipla escolha tem finalidade parecida com a estrutura aninhada. No exemplo acima se idade for menor que 18 então será exibido **Menor de idade**, se idade for maior ou igual a 18 e menor ou igual a 55 então será exibido **Adulto**. Se o valor de idade for igual à zero, será exibido **Bebê**, se o valor de idade não corresponder a nenhuma das opções acima então será exibido **Idoso**. Como atribuí o valor 22 no nosso exemplo a variável idade então no exemplo acima será exibido Adulto.

Segue o fluxograma:



6.1.4 – Exercícios para fixação

- 1) Construa um algoritmo que verifique a validade de uma senha fornecida pelo usuário. A senha valida deve ser igual a "aai3115%".

- 2) Dados três números inteiros, colocá-los em ordem crescente.

6.2 – Estruturas de repetição

As estruturas de repetição usam as expressões condicionais para repetir um determinado bloco de código. Essas estruturas também são conhecidas como estruturas de **loop** ou **lopping**.

Exemplo:

```
Algoritmo Exemplo11
Var
    x: integer
Inicio
    x ← 1
    enquanto x < 5 faça
        mostrar (x)
        x ← x + 1
    fim-enquanto
Fim.
```

O bloco de código acima mostrará na tela os números 1, 2, 3 e 4.

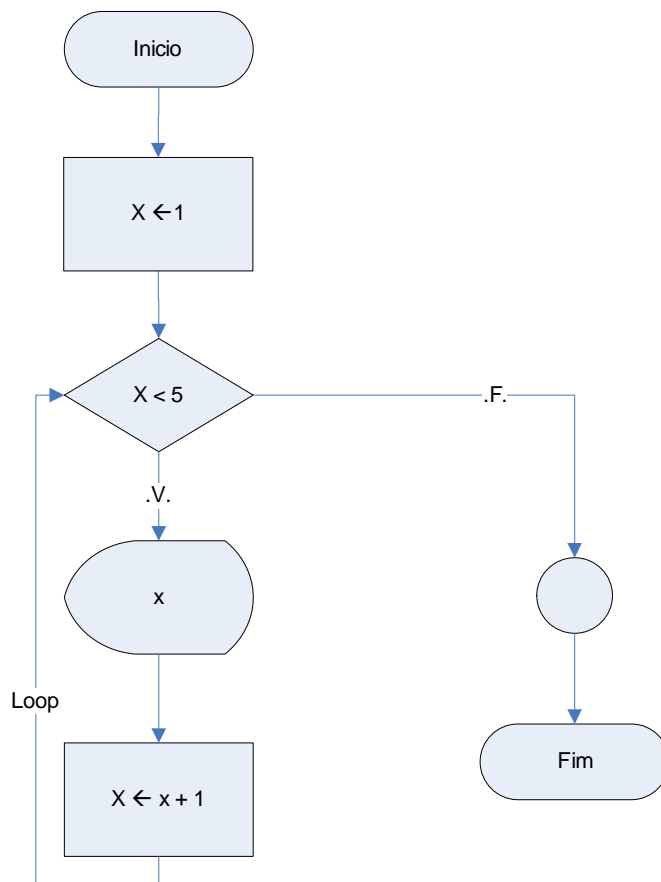
A expressão $x \leftarrow x + 1$ é denominada incremento. Isso significa que cada vez que for executado acrescentará 1 ao valor da variável x. O valor de **incremento** não precisa ser necessariamente 1, pode ser qualquer outro valor, inclusive negativo, neste caso chamamos de **decremento**.

Segue o fluxograma do nosso exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



6.2.1 – Estruturas de repetição com teste no início

O exemplo que acabamos de ver na introdução acima é uma estrutura de repetição com teste no início. Ela também é conhecida como estrutura **Enquanto**. Uma característica importante deste loop é que o teste é feito no início, ou seja, antes de entrar no bloco de código. Então, se logo na primeira vez que a estrutura enquanto for executada o resultado já for falso então o bloco de código de dentro da estrutura não será executado nenhuma vez.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Essa é uma característica importante das estruturas com teste no início.

Sintaxe em pseudocódigo:

```
Enquanto condição faça
    Bloco de instruções
Fim-enquanto
```

6.2.2 – Estruturas de repetição com teste no fim

A estrutura de repetição com teste no fim permite a um ou mais comandos serem executados repetidamente até uma condição específica tornar-se verdadeiro.

Essa estrutura é muito semelhante à estrutura anterior, a diferença é que as instruções são executadas antes da condição ser avaliada. Isso quer dizer que mesmo que o resultado da condição for falso as instruções serão executadas pelo menos uma vez.

Essa estrutura também é conhecida como **Repita**.

Sintaxe em pseudocódigo:

```
Repita
    Bloco de instruções
Até condição
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

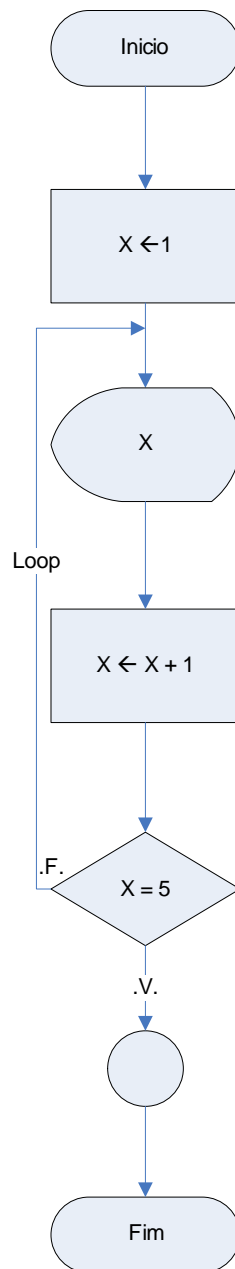
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Exemplos:

```
Algoritmo Exemplo12
Var
    x: integer
Inicio
    x ← 1
    Repita
        mostrar (x)
        x ← x + 1
    Até x = 5
Fim.
```

O exemplo acima vai exibir 1, 2, 3, 4 e 5.

Segue o fluxograma:



6.2.3 – Estruturas de repetição com variável de controle

Conhecida como **Para**, essa estrutura utiliza variáveis de controle que definem exatamente o número de vezes que a sequência de instruções será executada.

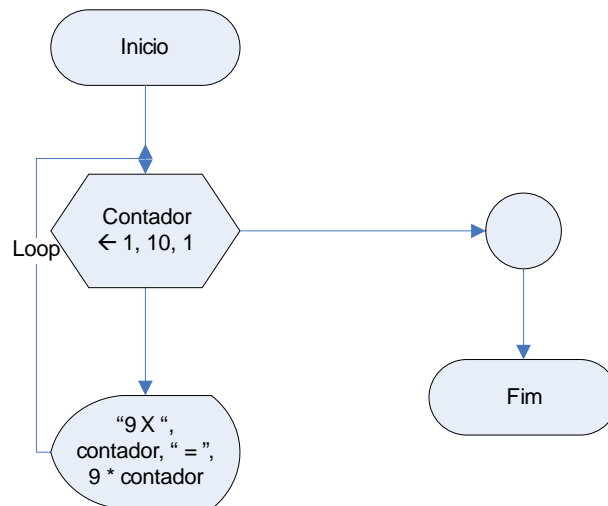
Usamos esta estrutura quando sabemos exatamente quantas vezes vamos executar o bloco de código.

Exemplo:

```
Algoritmo Exemplo13
Var
    contador: integer
Inicio
    Para contador = 1 até 10 passo 1 faça
        Mostrar ("9 X ", contador, " = ", 9 * contador)
    Fim-para
Fim.
```

O exemplo acima mostra na tela a tabuada do 9.

Segue fluxograma:



Sintaxe em pseudocódigo:

```
Para var = valor inicial até valor final Passo incremento faça  
    Bloco de instruções  
Fim-para
```

6.2.4 – Exercícios para fixação

- 1) Faça um algoritmo que leia 50 numeros fornecidos pelo usuario, calcule e exiba a média.
- 2) Faça um algoritmo que exiba a tabuada a partir de um numero fornecido pelo usuario.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

CAPITULO 7

ESTRUTURAS DE DADOS – VETORES E MATRIZES

Imagine que você tem que armazenar as faltas cometidas por cada jogador de um time em uma partida de futebol. Você precisaria armazenar o nome e o número de faltas que cada jogador cometeu. Como faria para armazenar isso em variáveis? Provavelmente você teria que declarar duas variáveis para cada jogador, uma para armazenar o nome e outra o número de faltas, chegando facilmente a 22 variáveis, isso, se não houver nenhuma substituição. Neste capítulo vamos aprender como gerenciar esse tipo de informação.

Vetores ou Arrays como também são conhecidos e Matrizes são usados para gerenciar grande quantidade de informação. Como variáveis, você precisa declarar seus arrays antes de usá-los.

7.1 – Vetores ou Arrays

Para que você possa compreender melhor o que são e como usar os arrays vamos fazer um exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Jogadores: vetor [1..11] de caracter
```

Acima declaramos um vetor chamado **Jogadores** do tipo **caracter**, ou seja, podemos usar esse vetor para armazenar os nomes dos jogadores. Nosso vetor permite que armazenemos 11 nomes conforme especificamos em sua declaração.

Os vetores são declarados no mesmo local que você declara as variáveis.

Agora como adicionamos o nome de um jogador no nosso vetor?

```
Jogadores[1] ← "Dida"
```

Desta forma atribuímos o valor **Dida** na primeira linha do nosso vetor.

Se fizermos assim:

```
Jogadores[6] ← "Dida"
```

O valor **Dida** do exemplo acima é inserido na linha 6 do nosso vetor. Você percebe assim que para inserir um valor no nosso vetor é como atribuir valor a uma variável, com a única diferença que precisa especificar o local ou linha onde o valor será armazenado. O numero que especifica isso é conhecido como **índice**, então 6 é um **índice** no nosso vetor no exemplo acima.

Para recuperar um valor de um vetor você faz da mesma forma que com as variáveis também, só que especifica o índice desejado.

```
Mostrar("Goleiro: ", jogadores[1])
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

O exemplo acima exibe o nome que está na posição 1 do vetor.

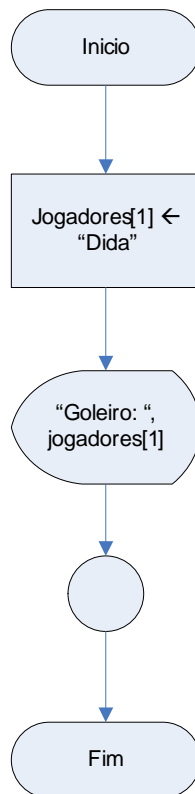
Você pode usar a estrutura de decisão **Para** a fim de varrer os elementos de um vetor.

Exemplos:

```
Algoritmo Exemplo14
Var
    Jogadores: vetor [1..11] de caracter
Inicio
    Jogadores[1] ← "Dida"
    Mostrar("Goleiro: ", jogadores[1])
Fim.
```

O algoritmo acima mostra como declaramos, atribuímos e lemos o valor de um vetor.

Segue o fluxograma:



Algoritmo Exemplo15

Var

num: vetor [1..6] de inteiro

soma, i: inteiro

Inicio

Soma \leftarrow 0

num[1] \leftarrow 2

num[2] \leftarrow 4

num[3] \leftarrow 5

num[4] \leftarrow 6

num[5] \leftarrow 8

num[6] \leftarrow 22

Para i \leftarrow 1 até 6 Faça

Soma \leftarrow soma + num[i]

Fim-Para

Autor: Herbert Moroni Cavallari da Costa Gois

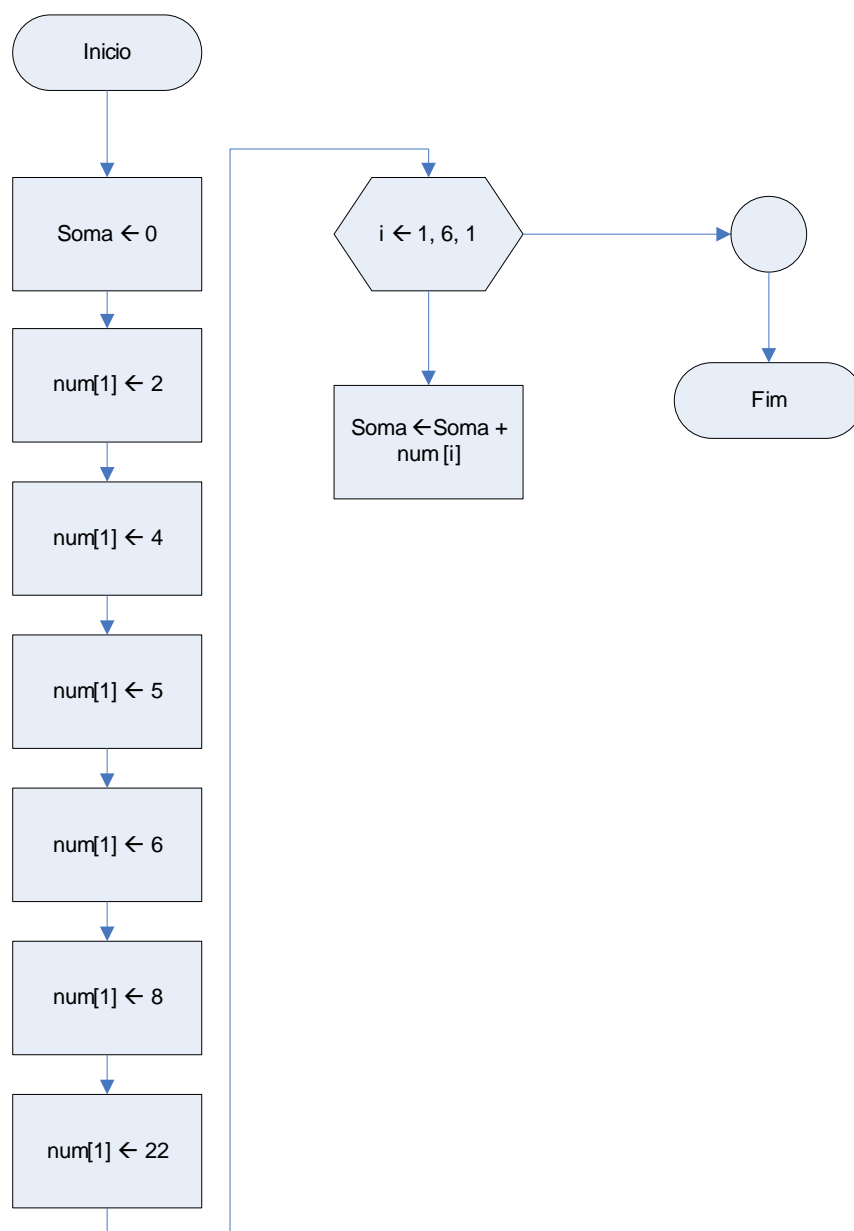
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Fim.

O algoritmo acima é um exemplo do uso da estrutura de repetição **Para** para somar todos os números entre os **índices** 1 e 6 do vetor **num**.

Segue o fluxograma:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

7.2 – Matrizes

Como você viu no item anterior, utilizamos um vetor para armazenar o nome dos jogadores.

No entanto como falei no principio deste capítulo nos precisamos armazenar o nome e o total de faltas de cada jogador em nosso problema.

Um vetor é como uma linha com varias colunas, para armazenar o total de faltas utilizando apenas vetores precisaríamos criar um novo vetor para armazenar as faltas ou utilizar matrizes.

Matrizes são mais parecidas com tabelas. Você pode ter varias “linhas”.

Novamente para que você entenda melhor vamos continuar com nosso exemplo dos jogadores.

```
Jogadores: vetor [1..11, 1..2] de character
```

Essa declaração é muito semelhante à declaração de vetor porque o vetor é uma matriz de uma dimensão.

Delimitadas entre colchetes temos duas declarações, a primeira representa a linha e o segundo a coluna. Visualmente nossa matriz seria assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

Uma tabela com 11 colunas e duas linhas.

Para armazenar um valor em uma matriz fazemos assim:

```
Jogadores[6,1] ← "Juninho"
```

No exemplo acima adicionamos o valor Juninho a primeira linha da matriz na coluna seis. No próximo exemplo adicionaremos um valor à segunda linha:

```
Jogadores[6,2] ← "22"
```

Perceba que coloquei 22 entre aspas porque minha matriz é do tipo caracter, então o 22 que estou armazenando nela é um texto e não um numero.

Neste ponto você já deve ter adivinhado como recuperar valores de suas matrizes, mas vamos ao exemplo:

```
Mostrar("Total de faltas do Juninho: ", jogadores[6,2])
```

Exemplos:

```
Algoritmo Exemplo16
```

```
Var
```

```
    Jogadores: vetor [1..11, 1..2] de caracter
```

```
Inicio
```

```
    Jogadores[6,1] ← "Juninho"
```

```
    Jogadores[6,2] ← "22"
```

Autor: Herbert Moroni Cavallari da Costa Gois

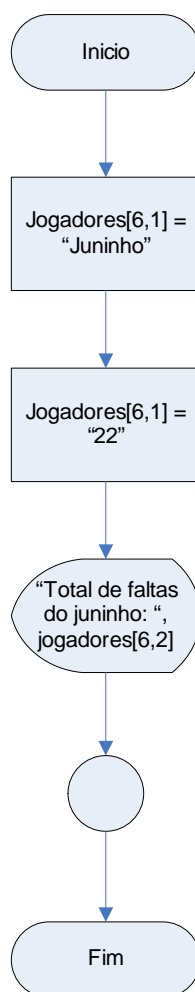
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Mostrar("Total de faltas do Juninho: ", jogadores[6,2])  
  
Fim.
```

O algoritmo acima mostra como declarar, atribuir e recuperar valores de uma matriz.

Segue o fluxograma:



O próximo exemplo mostra como fazer operações com matrizes. A partir de uma matriz de 6 colunas e duas linhas, calculamos e exibimos a media geometria dos

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

valores de cada uma das colunas. A media geométrica é calculada pela seguinte expressão: $\text{SQRT}(x_1 * x_2)$, que representa a raiz quadrada do resultado da multiplicação dos elementos da linha pelos da coluna.

Algoritmo Exemplo17

Var

V: vetor [1..6, 1..2] de inteiro

i, j: inteiro

prod: real

Início

Para i ← 1 até 6 Faça

Prod ←

Para i ← até 6 Faça

Ler (V [i,j])

Prod ← prod * V[i,j]

Fim-para

Mostrar ("Linha ", i, " = ", SQRT (prod))

Fim-para

Fim.

CAPITULO 8

PROCEDIMENTOS E FUNÇÕES

Quando você começar a escrever programas grandes e complexos, verá que seu programa será composto de vários algoritmos e que alguns algoritmos existentes em seu programa diferentes precisarão acessar as mesmas variáveis e rotinas.

Por padrão, variáveis são locais aos seus respectivos algoritmos. Entenda nesta introdução que uma variável local não pode ser acessada em todo lugar do programa - até agora criamos apenas variáveis locais.

Procedimento é um conjunto de códigos a serem executados para uma determinada finalidade. Quando você estiver escrevendo seus programas, vai notar que muito código digitado é igual, ou seja, que você precisa repetir código em vários locais.

Os procedimentos auxiliam nisso, você pode criar um procedimento e depois chamá-lo em seu programa sempre que for necessário.

Função é igual a um procedimento só que sempre vai retornar um valor. Um procedimento nunca vai retornar nada.

Os procedimentos e funções também ajudam na manutenção do programa.

Imagine que você precise fazer certo cálculo em vários locais do seu programa. E agora a fórmula do cálculo foi modificada, você terá que localizar no seu programa

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

cada local que tem a fórmula para alterá-la. Se você tiver um procedimento ou função que faz o cálculo, basta alterar o mesmo.

Tanto os procedimentos como as funções são “mini-algoritmos” que possuem variáveis e até mesmo outros procedimentos e funções dentro deles.

8.1 – Procedimentos

Um procedimento também conhecido como sub-rotina é um conjunto de instruções que realiza uma tarefa. Um algoritmo de procedimento é criado da mesma maneira que outro algoritmo qualquer. Você cria seus procedimentos antes de declarar as variáveis dentro do seu algoritmo.

Sintaxe em Pseudocódigo:

```
Procedimento nome_do_procedimento (lista de parâmetros)
Var
    Declaração das variáveis que pertencem a este procedimento
Inicio
    Instruções do procedimento
Fim Procedimento
```

A chamada de um procedimento é o momento em que o procedimento é acionado e seu código é executado. Você chama seu procedimento pelo nome passando os valores necessários nos parâmetros quando eles existirem. Os parâmetros são opcionais.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

Sintaxe de chamada de procedimento em Pseudocódigo:

```
nome_do_procedimento (valores de parâmetros)
```

Exemplo:

```
Algoritmo Exemplo18
    Procedimento CalcularAreaQuadrado (lado:real)
        Var
            Resultado: real
        Inicio
            Resultado := lado * lado
            Mostrar(Resultado)
        Fim Procedimento
    Var
        l: real
    Inicio
        Ler(l)
        CalcularAreaQuadrado (l)
    Fim.
```

O procedimento acima foi feito para mostrar a área de um quadrado. Para executar o procedimento você precisa passar o valor do lado do quadrado. Toda vez que você for precisar mostrar a área do quadrado na tela vai poder usar esse procedimento.

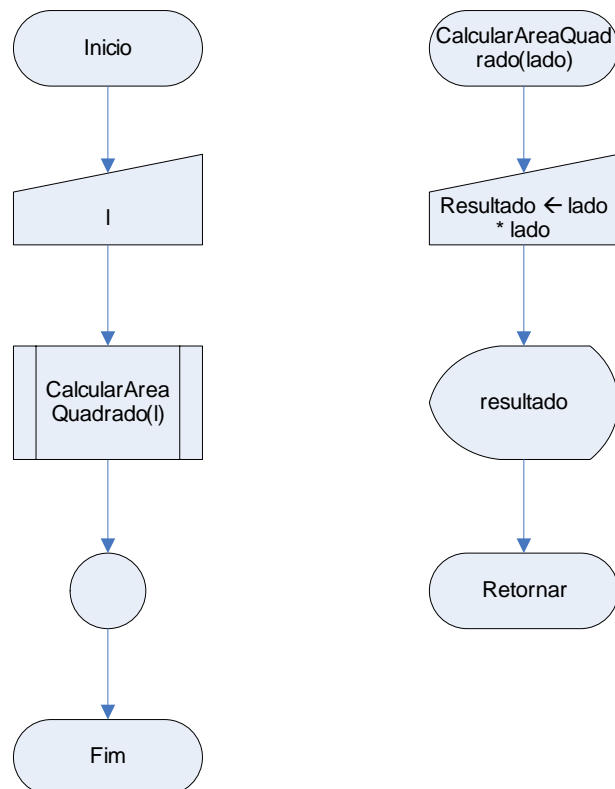
Você chama o procedimento da seguinte forma no seu programa supondo que o valor do lado seja 20.

```
CalcularAreaQuadrado (20)
```

Você também pode passar uma variável como parâmetro, desde que o valor nela seja do mesmo tipo de dado do parâmetro, veja o exemplo a seguir, suponha que a variável **ladoDoQuad** é do tipo real.

```
CalcularAreaQuadrado (ladoDoQuad)
```

Segue o fluxograma do exemplo acima:



Perceba que no fluxograma o procedimento é feito como se fosse um algoritmo separado, só que no lugar de início você coloca o nome do procedimento e no fim a instrução retornar que significa que o controle de fluxo de dados deverá retornar ao procedimento principal. Note também como representamos a chamada do procedimento em nosso algoritmo principal.

O próximo exemplo representa um algoritmo cujo procedimento recebe mais do que um parâmetro:

Algoritmo Exemplo19

```
Procedimento CalcularAreaRetangulo (lado:real, altura:real)
Var
    Resultado: real
Início
    Resultado := lado * altura
    Mostrar(Resultado)
Fim Procedimento

Var
    Lado, altura: real
Início
    Leia(Lado, altura)
    CalcularAreaRetangulo (Lado, altura)
Fim.
```

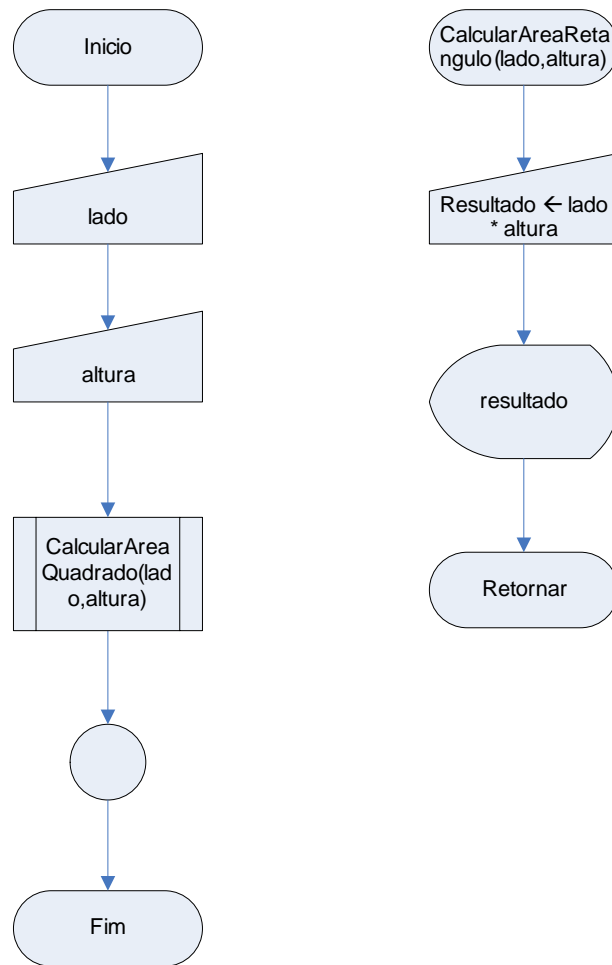
Para declarar mais de um parâmetro você os separa com vírgula como o exemplo acima.

Segue o fluxograma:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Se os tipos de dados dos parâmetros forem iguais você pode fazer assim:

Algoritmo Exemplo20

Procedimento CalcularAreaRetangula (lado, altura:real)

Var

Resultado: real

Inicio

Resultado := lado * altura

Mostrar(Resultado)

Fim Procedimento

Var

L1, a1: real

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Inicio
    Leia(L1, a1)
    CalcularAreaRetangula (L1, a1)
Fim.
```

O exemplo abaixo não recebe parâmetro nenhum:

```
Algoritmo Exemplo21
    Procedimento MostrarProprietario ( )
    Var
        proprietario: Caracter
    Inicio
        proprietario ← "Herbert Moroni"
        Mostrar(proprietario)
    Fim Procedimento
Var
Inicio
    MostrarProprietario()
Fim.
```

8.2 – Funções

Tudo que se aplica aos procedimentos é usado para as funções, elas são criadas e chamadas da mesma maneira. A diferença entre eles é que as funções podem ser utilizadas em expressões, como se fossem variáveis, pois as funções retornam valores que são associados ao seu nome. Por isso é necessário que ao criar uma função você especifique o tipo de dado que ela vai retornar.

Sintaxe em Pseudocódigo:

```
Função nome_da_função (lista de parâmetros): tipo_de_dado da função
Var
    Declaração das variáveis que pertencem a esta função
Início
    Instruções da função
Retornar(variável)
```

Exemplo:

```
Algoritmo Exemplo22
    Função CalcularAreaQuadrado (lado:real): real
    Var
        resultado: real
    Início
        resultado := lado * lado
    Retornar(resultado)
Var
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
    n1:real
    result:real

Inicio

    Ler(n1)

    Result ← CalcularAreaQuadrado(n1)

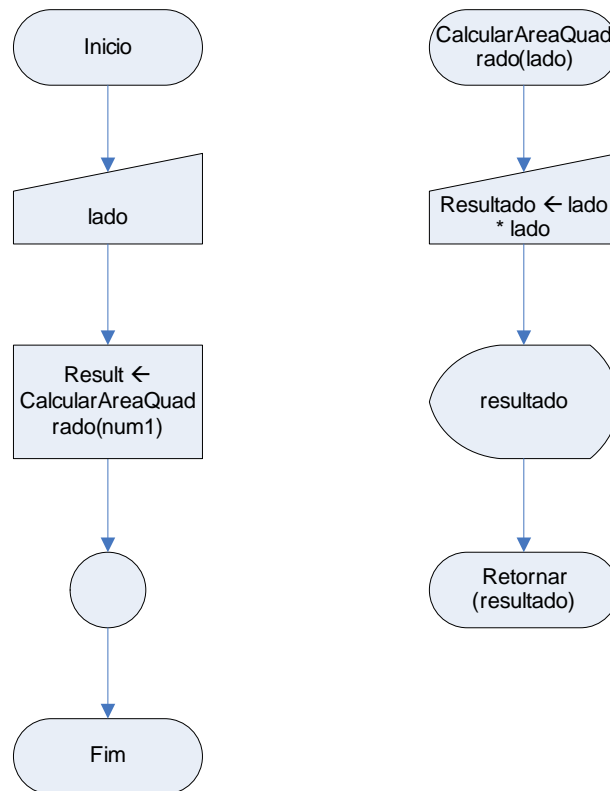
Fim.
```

Perceba que a função acima não usa um comando de saída para exibir o resultado, ela retorna o conteúdo da variável resultado através do comando **Retornar**. Por isso você não pode simplesmente chamar suas funções para usá-las. Segue um exemplo do uso da função do exemplo acima:

```
Result ← CalcularAreaQuadrado(n1)
```

No exemplo acima a variável **Result** recebe o resultado da função **CalcularAreaQuadrado**. O único detalhe é que a variável **Result** deve suportar o mesmo tipo de dado que a função retorna, no caso do nosso exemplo o tipo **real**.

Segue o fluxograma:



8.3 – Escopo de variáveis

Até agora, todos os exemplos utilizaram **variáveis locais**. Ou seja, variáveis que só podem ser utilizadas no escopo do algoritmo/programa no qual foram declaradas. Vamos analisar o seguinte exemplo para que você possa entender melhor sobre escopo de variáveis.

Algoritmo Exemplo23

Função Multiplicar (a, b:real):real

Var

 resultado: real

Início

 resultado ← a * b

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

```
Retornar(resultado)

Var

    numero1, numero2: real

Inicio

    Ler (numero1, numero2)

    Mostrar (Multiplicar(numero1,numero2))

Fim.
```

No exemplo acima a variável **resultado** foi declarada dentro da função **Multiplicar**. Essa variável só pode ser usada dentro desta função, você não pode atribuir ou consultar nenhum valor nela diretamente, somente através da função. Chamamos a variável **resultado** dentro da função **Multiplicar** de **variável local**. Já as variáveis **numero1** e **numero2** são globais, porque podem ser acessadas tanto pelo corpo do algoritmo quanto por qualquer procedimento ou função declarada dentro do algoritmo.

CAPITULO 9

LÓGICA DE PROGRAMAÇÃO COM C#

O C#, junto com o Visual Studio .NET 2005 compõe uma ferramenta extremamente robusta e fácil de utilizar, com perfeito suporte a todas as novas ondas que rondam o mundo da informática e tecnologia.

O Visual Studio .NET 2005 é a melhor ferramenta de desenvolvimento de aplicações para a plataforma .NET. Com uma interface amigável e integrada com os ambientes e de fácil entendimento, proporciona aos desenvolvedores a criação de aplicações sofisticadas com todos os recursos existentes, sem ter que ficar criando parte de código em um aplicativo e o restante no outro. É possível com o Visual Studio gerenciar recursos da máquina e local e de um possível servidor, criar aplicações para Windows, web e dispositivos móveis.

Meu objetivo neste capítulo é ajuda-lo a aplicar os conhecimentos de lógica de programação adquiridos até aqui na linguagem de programação C#.

Vamos primeiramente conhecer um pouco do Visual Studio.NET, a ferramenta que utilizaremos para desenvolver nossos aplicativos e criar nosso primeiro exemplo.

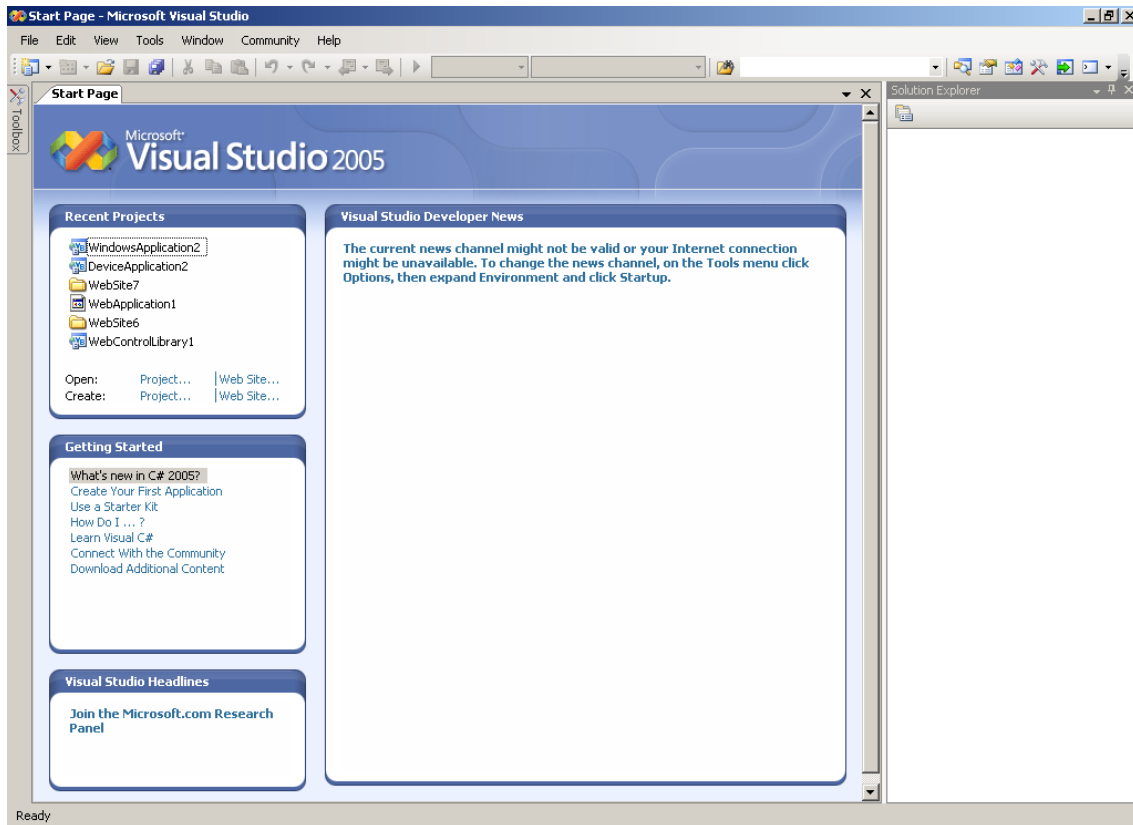
1 – Entre no Visual Studio.NET, eu estou usando a versão 2005, mas os exercícios funcionam em qualquer versão.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

Você pode entrar no Visual Studio.NET através do menu **Iniciar / Programas / Microsoft Visual Studio .NET / Microsoft Visual Studio .NET** , sinta-se a vontade para criar qualquer atalho para ele.



A imagem anterior mostra o Visual Studio .NET assim que o iniciamos, é exibida a pagina **Start Page** onde podemos abrir rapidamente os ultimos projetos criados através da caixa **Recent Projects**.

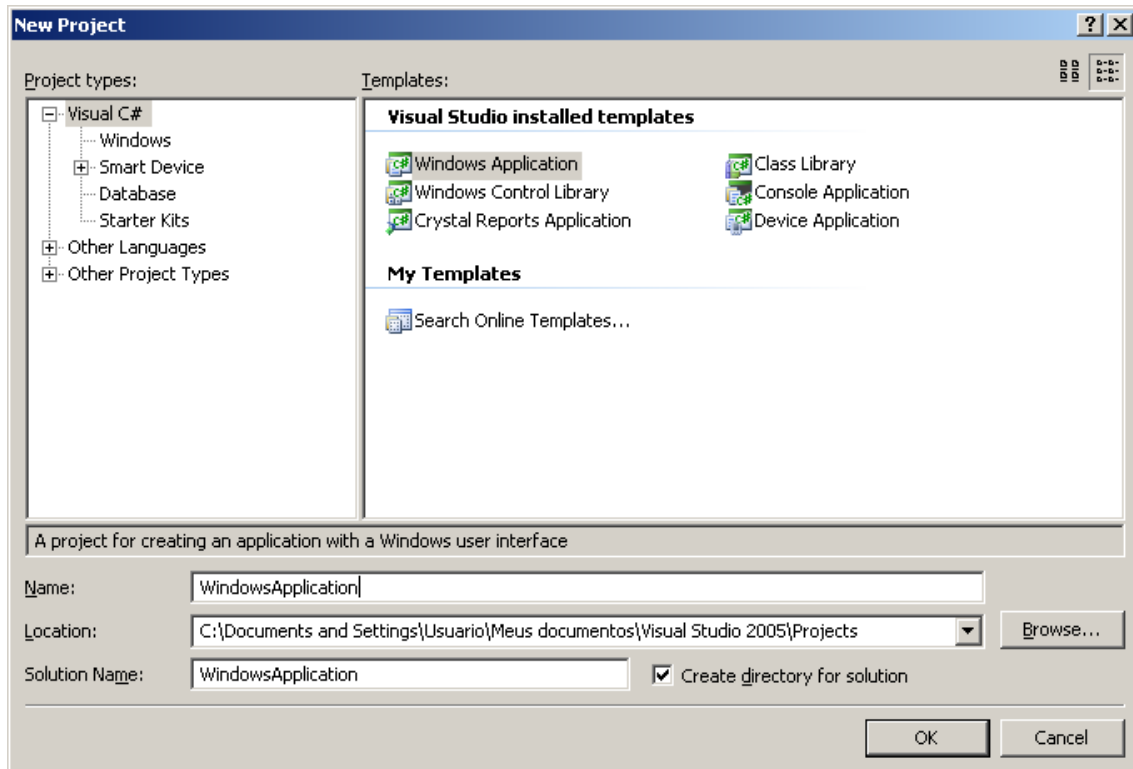
2 – No menu **File**, aponte em **New**, e clique em **Project**. (Ou clique Ctrl+Shift+N).

A caixa de dialogo **New Project** aparece. Ela permite que criemos um novo projeto usando vários **templates**, como um Windows Application, Class Library, Console Application e vários outros.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



3 – No painel **Project Type**, clique em **Visual C# Projects**, aqui estão todos os templates disponíveis para a linguagem C#.

4 – No painel **Templates** clique em **Console Application**.

5 – No campo nome digite, **ClassicoHelloWorld**.

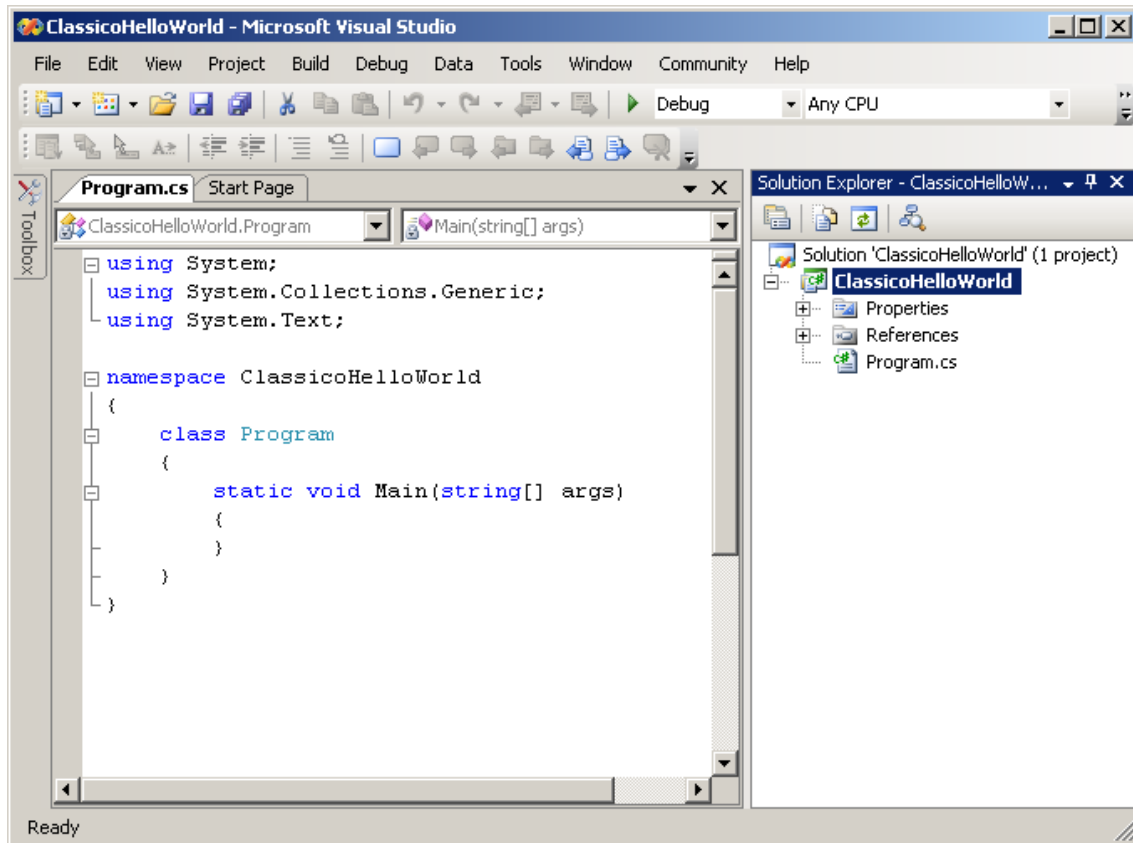
Perceba que você pode ainda alterar o caminho onde sua aplicação será salva e o nome da sua Solução.

6 – Clique em OK.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



A **barra de menus (menu bar)** possibilita o acesso aos comandos que você usa no ambiente de programação. Você pode usar o teclado ou o mouse para acessar o menu ou atalhos exatamente como usa em outros programas baseados em Windows.

A **barra de ferramentas (toolbar)** é localizada embaixo da barra de menus e disponibiliza botões que executam os comandos usados com mais frequência. Não confunda toolbar com toolbox.

A janela **Solution Explorer** mostra os nomes dos arquivos associados com o seu projeto. Você pode dar um clique duplo sobre o nome do arquivo para exibi-lo no **painel de código (Code pane)**. Vamos examinar os arquivos que o Visual Studio criou como parte do seu projeto:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- **ClassicoHelloWorld.sln** organiza os projetos de cada solução, uma solução pode ter vários projetos, seu nome é representado no primeiro item na janela Solution Explorer só que sem a extensão do arquivo.
- **ClassicoHelloWorld.csproj** este é o arquivo do projeto C#. Pode ser associado a vários arquivos de código. É reconhecido no Solution Explorer pelo nome do projeto apenas, no entanto é gravado no disco com a extensão .csproj.
- **Program.cs** é um arquivo de código do C#. Você vai escrever seu código neste arquivo. O Visual Studio já adicionou algum código nele automaticamente, examinaremos esse código mais adiante.

Aos poucos nós vamos explorando mais o Visual Studio, vamos ao nosso primeiro exemplo.

O arquivo **Program.cs** define uma classe chamada **Program** que contém um método chamado **Main**. Todos os métodos precisam ser definidos **dentro** de uma classe. O método **Main** é especial porque ele é o primeiro a ser executado quando o programa é iniciado, por isso ele precisa ser designado como **static** (estático), métodos e classes vão ser discutidos em detalhes mais adiante no curso.

Importante: O C# é **case-sensitive**, ele diferencia letras minúsculas de maiúsculas, um M é interpretado diferente de um m. Consequentemente Main é diferente de main.

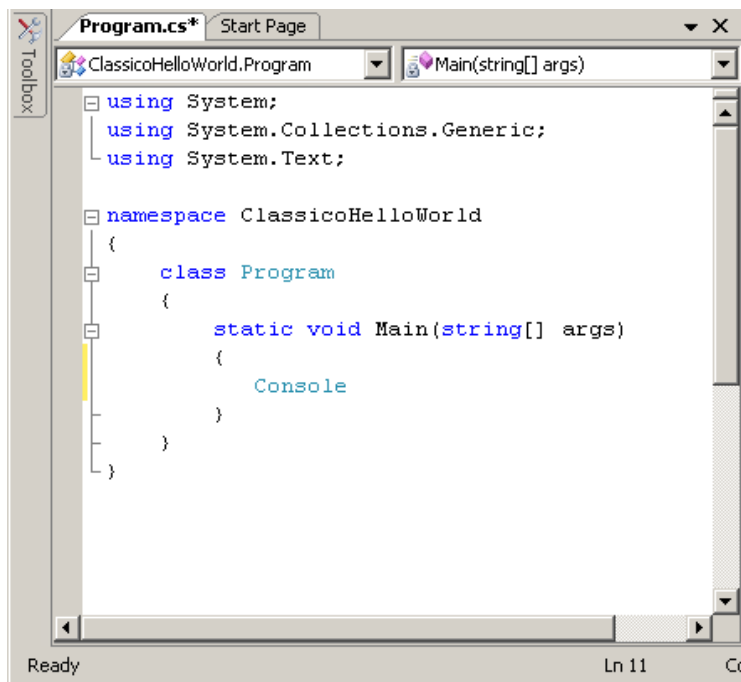
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Nosso primeiro exemplo é bem simples, e um clássico para quem está aprendendo qualquer linguagem, ele escreve **Hello World** no console.

7 – Dentro do método **Main**, entre os colchetes digite: **Console**



A classe **Console** contém os métodos para exibir mensagens na tela e pegar as entradas do teclado. Tudo que o usuário digita no teclado pode ser lido através da classe **Console**. A classe **Console** só é significativa para aplicações que rodam no prompt de comando como neste nosso primeiro exemplo.

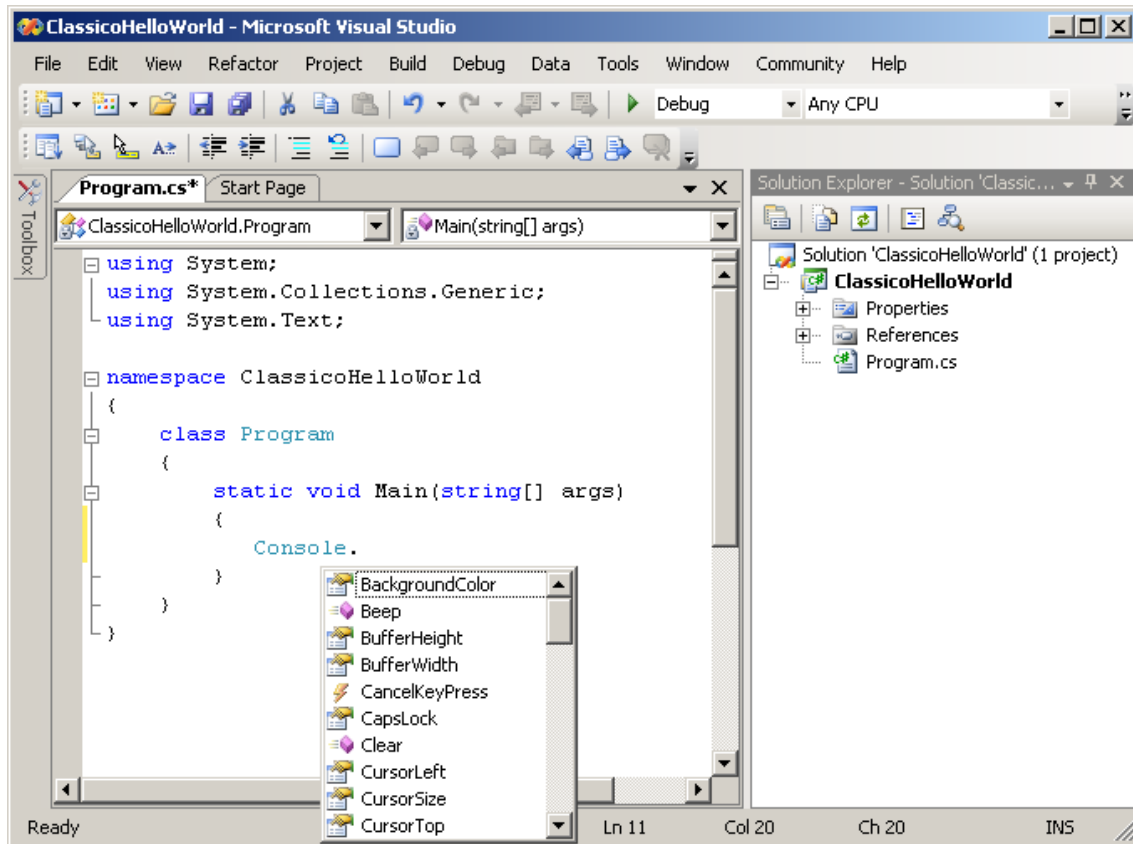
8 – Agora digite um ponto depois de **Console**.

Uma lista aparece, ela é chamada de **IntelliSense**, esse não é um recurso exclusivo do Visual Studio mas ajuda muito na programação principalmente em linguagens case-sensitive, como é o C#. O IntelliSense exibe todos os métodos, propriedades e campos da classe.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



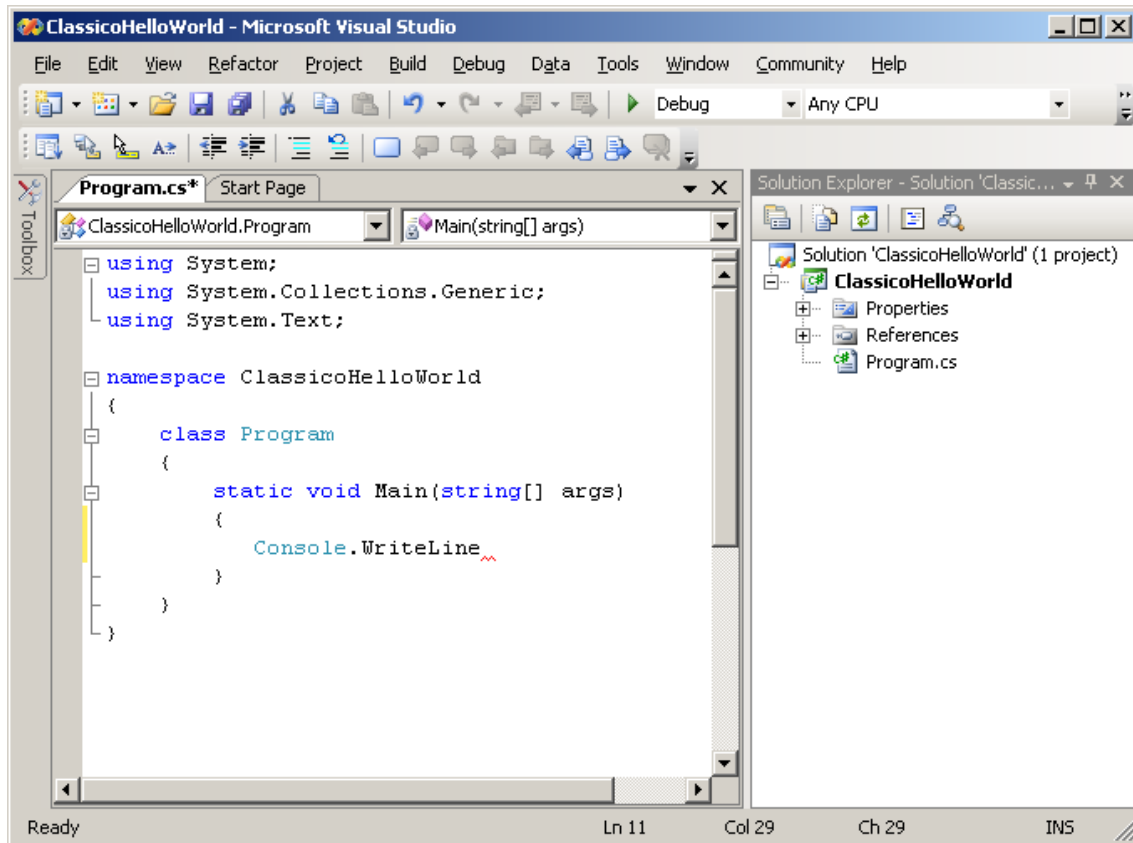
9 – Selecione **WriteLine**, você pode usar o Mouse ou o Teclado, tecla Enter ou dê um clique duplo sobre o WriteLine.

O IntelliSense é fechado e o método WriteLine é adicionado ao código. Como a seguinte imagem:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Quando o IntelliSense aparece você também pode pressionar W para ir direto para o primeiro membro do método Console que começa com w.

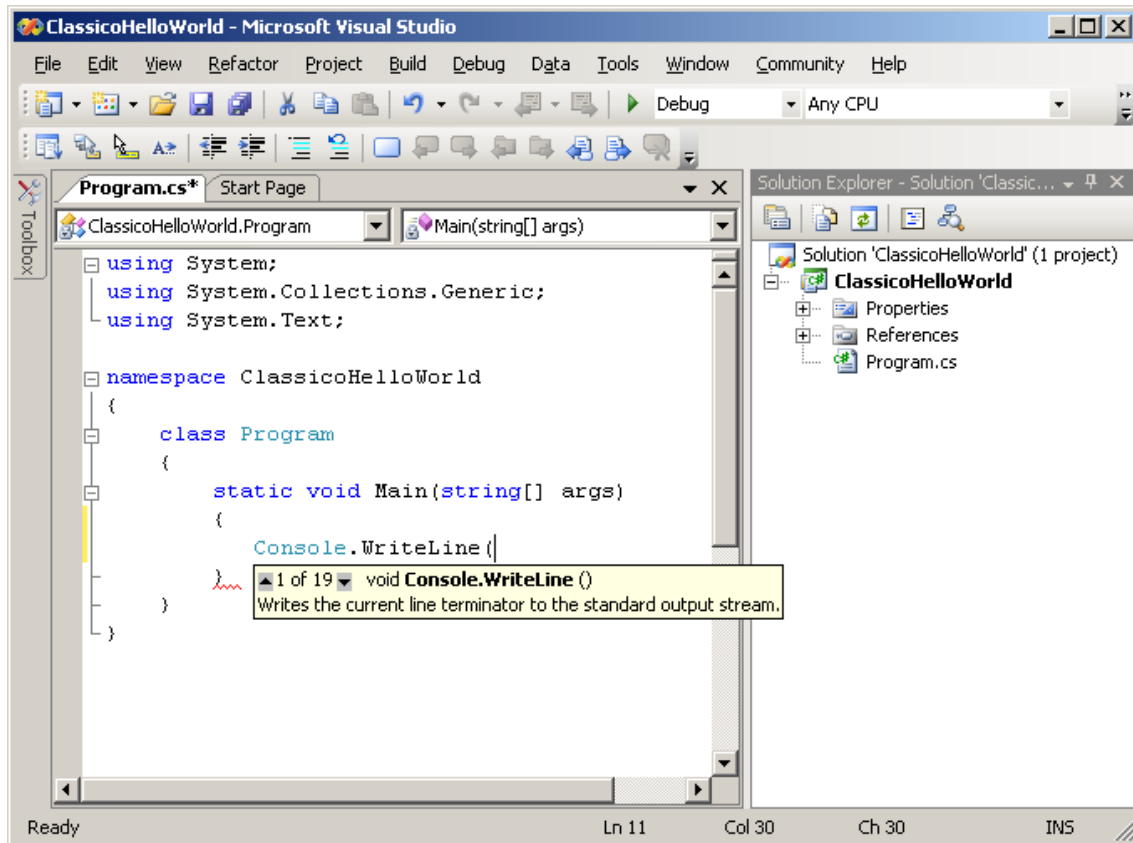
10 – Abra parênteses (

É mostrado uma outra forma do IntelliSense, esse mostra os parâmetros do método WriteLine. O método WriteLine tem o que chamamos de **Sobrecarga (Overload)**. Para cada sobrecarga do método WriteLine usamos parâmetros diferentes. Cada sobrecarga e seus respectivos parâmetros podem ser visualizados clicando com o mouse na seta do IntelliSense ou navegando pelas setas do teclado.

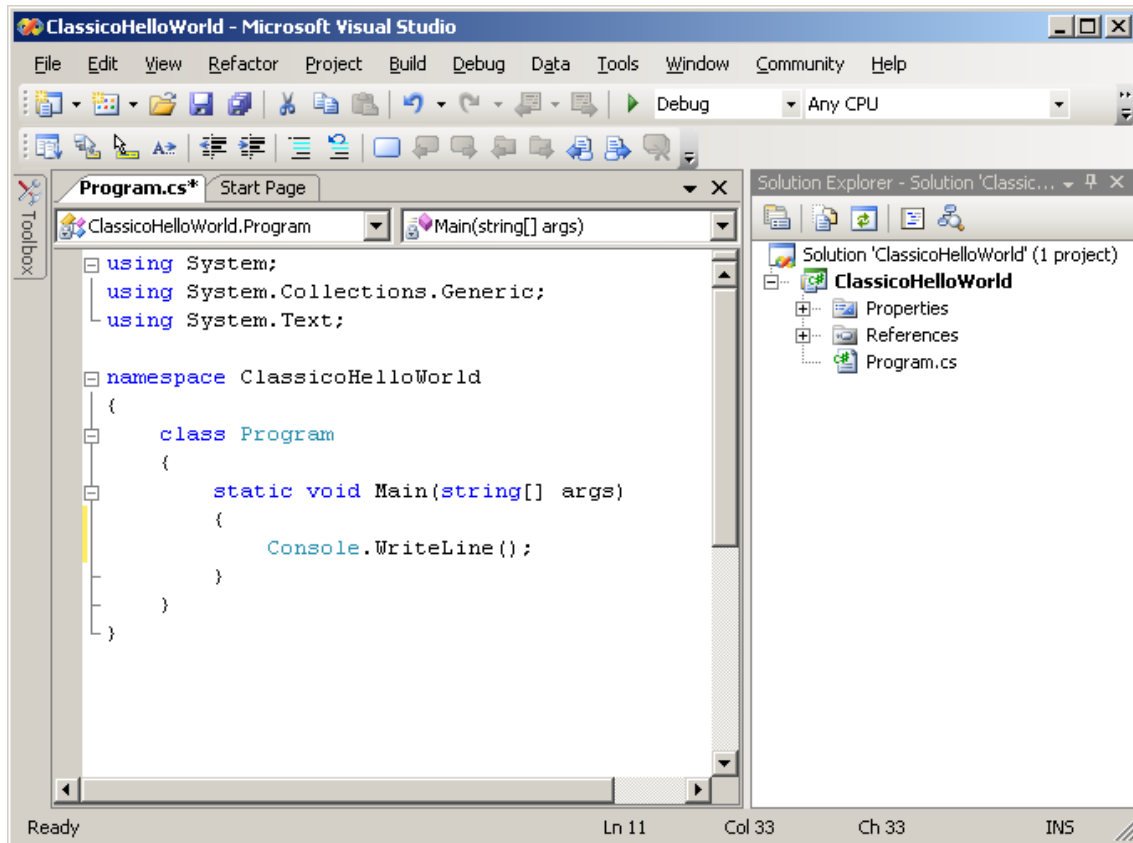
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

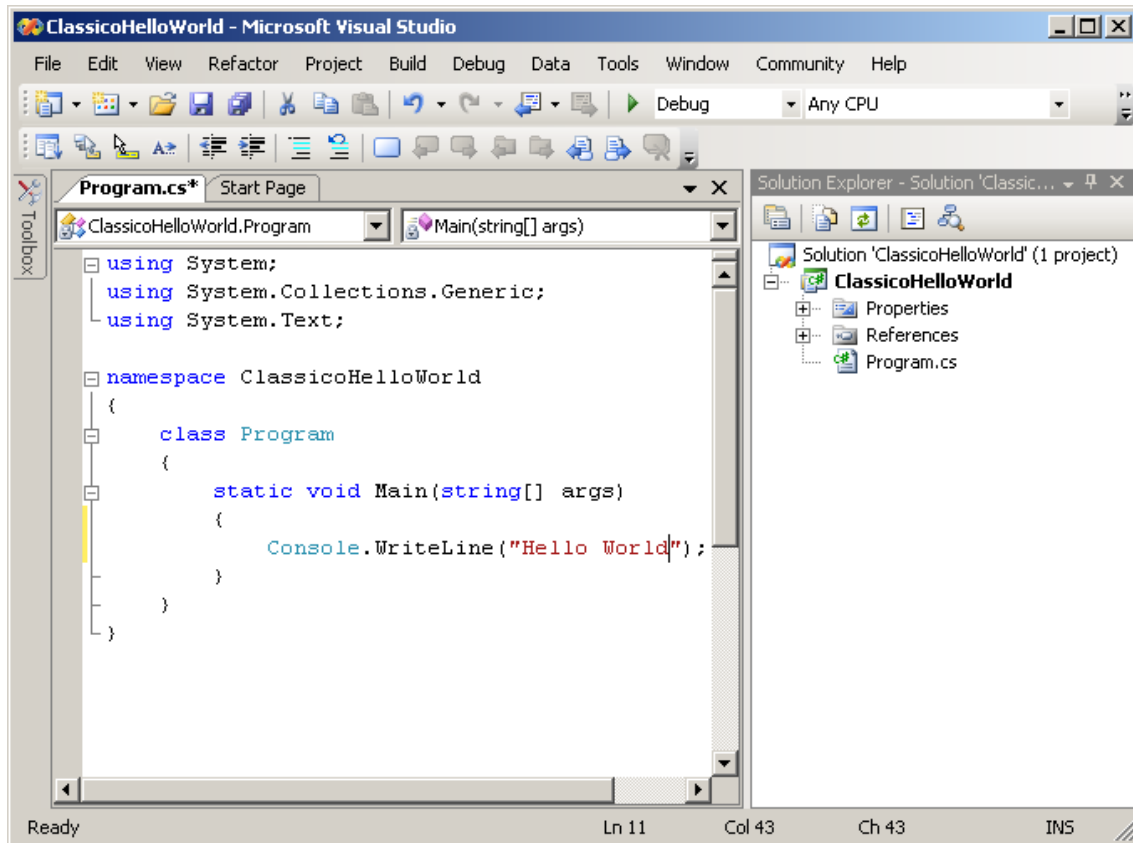
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



11 – Feche os parênteses) e digite ponto-e-virgula, vai ficar assim:



12 – Digite entre os parênteses a string **“Hello World”**, string deve ficar entre aspas. Vai ficar assim:



Pegue o habito de digitar os pares de caracteres juntos, como (e) e { e }, antes de entrar com seus respectivos conteúdos. Assim você evitará alguns erros por esquecer de fechar.

13 – Vamos agora compilar nossa aplicação. No menu **Build**, clique em **Build Solution**. Se tudo estiver certinho vai aparecer a seguinte linha na **janela Output**:

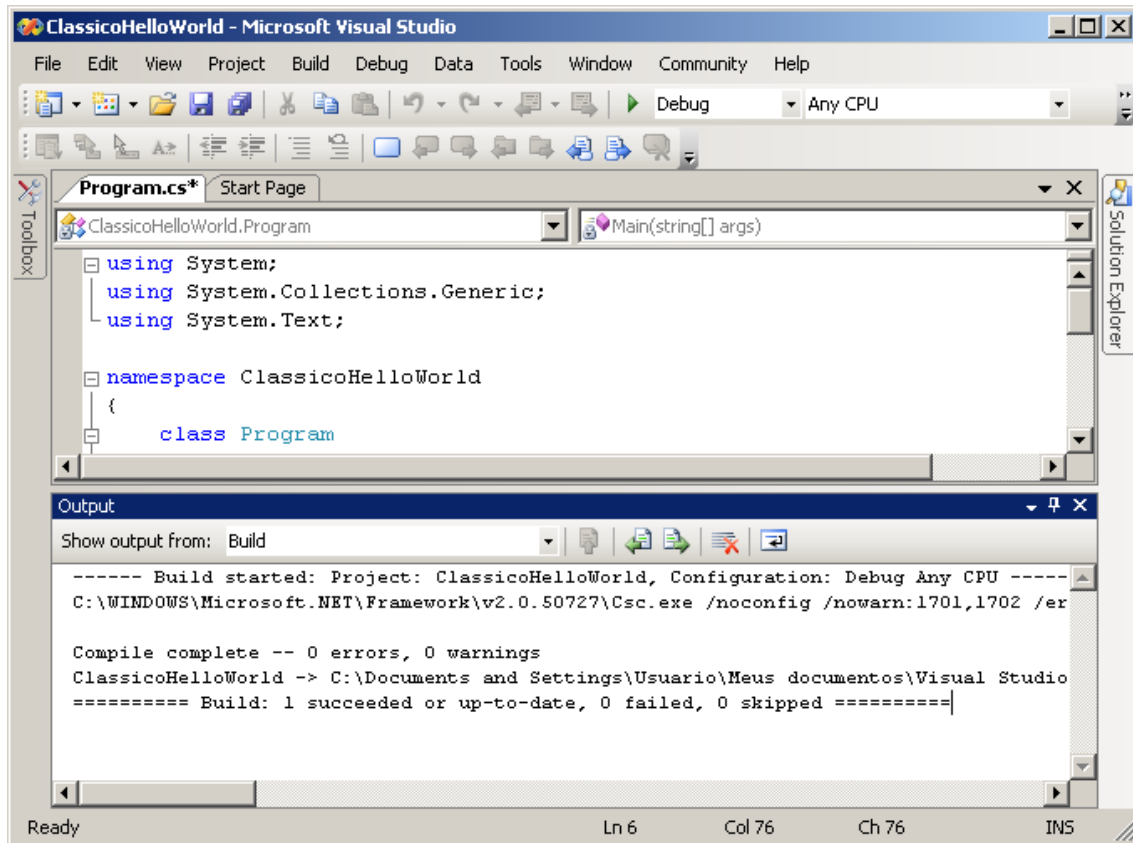
```
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
```

Para exibir a janela Output na barra de menus clique em View, Output ou pressione Ctrl+W+O.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



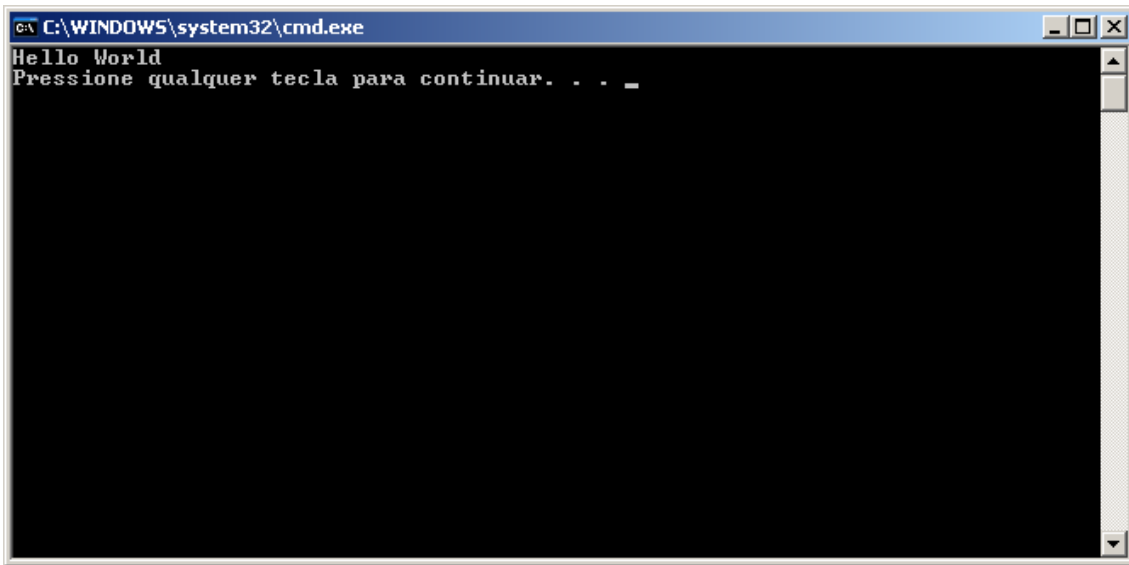
Um asterisco depois do nome do arquivo no painel de código indica que foram feitas modificações no código do respectivo arquivo e que essas alterações não foram salvas. Você pode salvar manualmente antes de compilar a aplicação, mas ao compilar o Visual Studio salva automaticamente todos os arquivos da aplicação.

14 – No menu **Debug**, clique em **Start Without Debugging** para executar o programa no prompt de commando.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



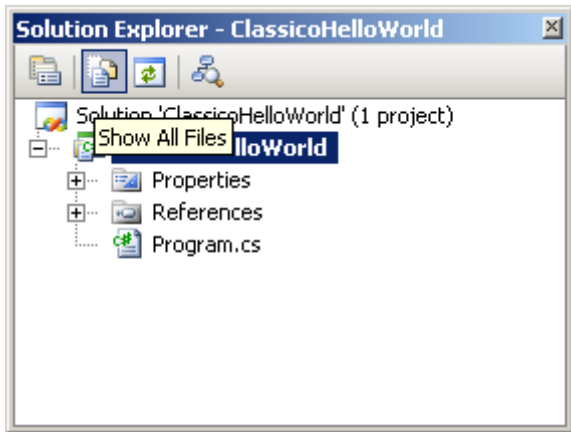
O programa vai escrever **Hello World** como a ilustração acima.

Nós escolhemos **Start Without Debugging** para forçar uma pausa no final da execução. Se clicássemos em **Start** ele iria executar o programa e fechar o prompt de comando logo após a execução, seria tão rápido que não conseguiríamos ver o que foi escrito, experimente.

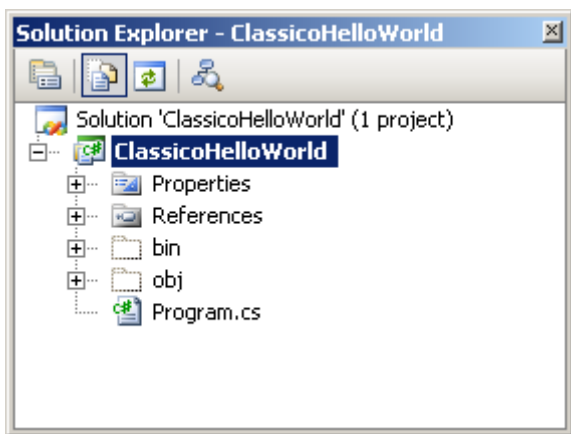
15 – Com o foco no prompt de comando pressione qualquer tecla.

A janela irá fechar e retornaremos para o Visual Studio.

16 - Na janela **Solution Explorer**, clique no botão **Show All Files**.



Aparecem os nomes **bin** e **obj** depois do nome do projeto. Esses dois correspondem a pastas com seus respectivos nomes. Essas pastas são criadas quando você executa a aplicação e contem uma versão executável do programa e outros arquivos necessários para depurar o programa.

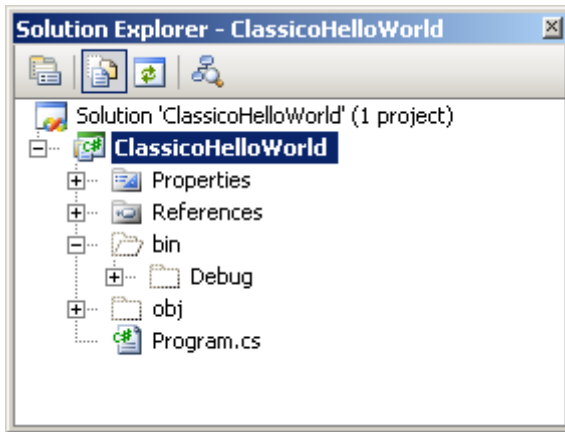


17 – Ainda na janela **Solution Explorer**, clique no sinal de + à esquerda do nome **bin**.

Autor: Herbert Moroni Cavallari da Costa Gois

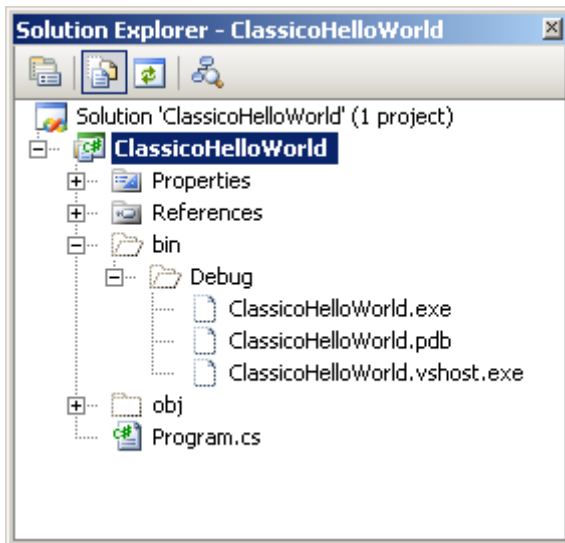
Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



Um outro nome aparece representando uma outra pasta chamada **debug**.

18 – Clique no sinal de + de **debug**.



Repare nos arquivos: **ClassicoHelloWorld.exe** e **ClassicoHelloWorld.pdb**.

O arquivo .exe é o executável da aplicação.

O arquivo .pdb armazena informações de depuração da aplicação.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

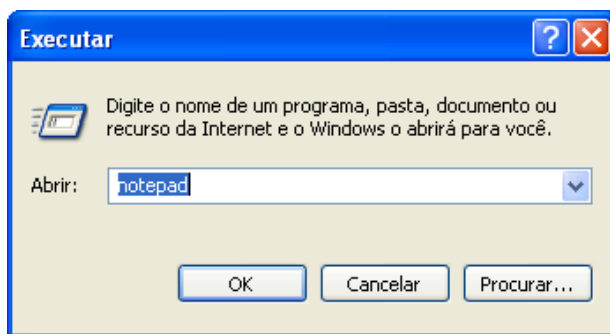
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

O arquivo **ClassicoHelloWorld.vshost.exe** só aparece no Visual Studio 2005, ele é usado para melhorar a performance da depuração de erros.

Como vimos até agora, o Visual Studio compilou automaticamente nosso programa e criou os arquivos necessários automaticamente, durante o processo de compilação. Em resumo a compilação é o processo que transforma seus arquivos fonte em um arquivo executável pelo sistema operacional, um **.exe** por exemplo.

Antes de prosseguir eu quero fazer mais um exemplo com você, agora vamos compilar nosso programa através do prompt de comando utilizando um utilitário chamado csc.

1 - Com o Visual Studio fechado, na barra de ferramentas do windows clique em **Iniciar, Executar**, digite **notepad** em Abrir e clique em OK.



2 - Digite o seguinte código no notepad.

```
using System;  
  
namespace txtHello  
{
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

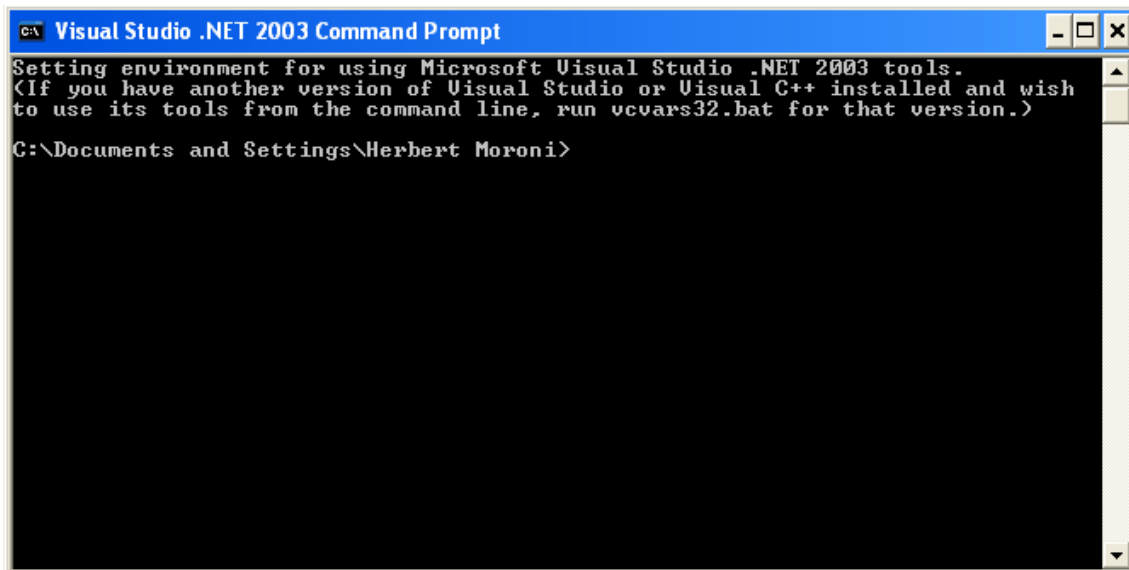
```
class Class1
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");
    }
}
```

3 – Salve o arquivo com o nome **teste.cs**

Para isso será necessário escolher Todos os arquivos na opção Salvar como tipo no notepad quando você clicar em **Arquivo \ Salvar**.

4 – Feche o notepad.

5 – No menu Iniciar, vá em **Programas / Visual Studio .NET / Visual Studio .NET Tools** e clique em **Visual Studio .NET Command Prompt**.



```
Visual Studio .NET 2003 Command Prompt
Setting environment for using Microsoft Visual Studio .NET 2003 tools.
(If you have another version of Visual Studio or Visual C++ installed and wish
to use its tools from the command line, run vcvars32.bat for that version.)
C:\Documents and Settings\Herbert Moroni>
```

Para compilarmos manualmente nosso programa são necessárias algumas mudanças no ambiente do sistema operacional como alterações em algumas variáveis de ambiente, PATH, LIB e INCLUDE. Essas mudanças incluem adicionar pastas contendo bibliotecas e utilitários .NET.

6 – No prompt, vá até a pasta que você salvou o arquivo **teste.cs**.

7 – Digite **dir** e tecla **enter**.

Ele vai listar os arquivos da pasta, no caso vai mostrar o arquivo **teste.cs**.

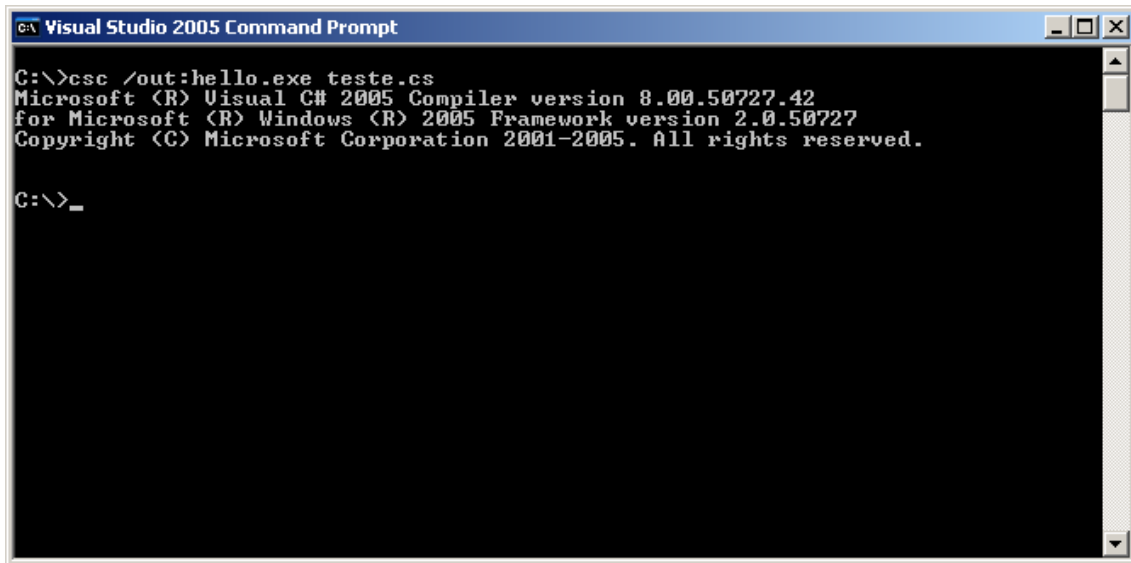
8 – Digite: `csc /out:hello.exe teste.cs`

Isso vai criar um executável chamado hello.exe.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



```
C:\>Visual Studio 2005 Command Prompt

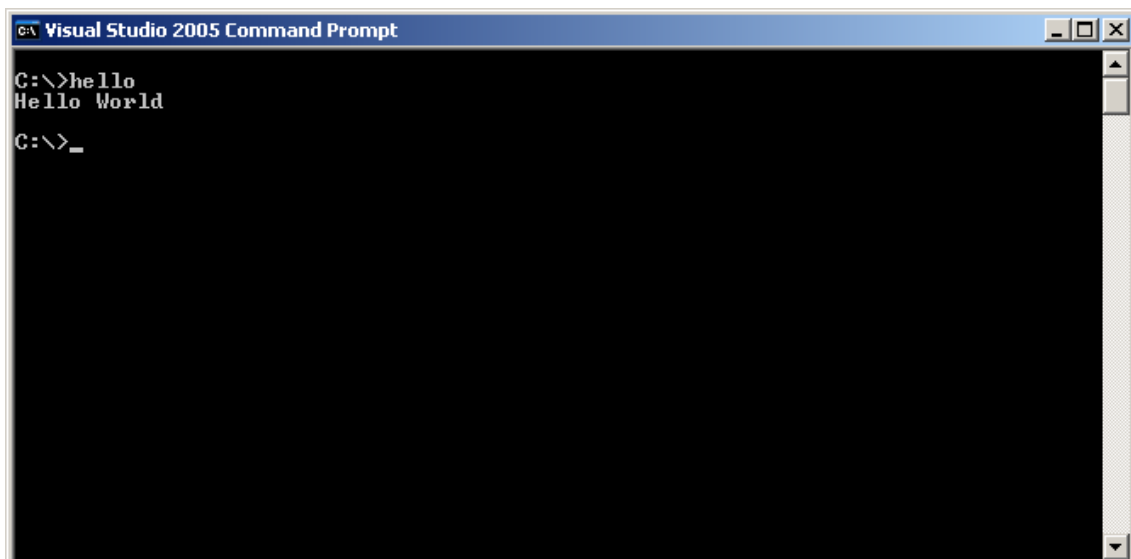
C:\>csc /out:hello.exe teste.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\>_
```

9 – Digite **dir** e tecla **enter**.

Agora você não vê apenas o arquivo **teste.cs**, foi adicionado um arquivo chamado **hello.exe**, ele é o resultado da compilação do seu arquivo fonte **teste.cs**.

10 – Digite **hello** e tecla **enter**.



```
C:\>Visual Studio 2005 Command Prompt

C:\>hello
Hello World

C:\>_
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

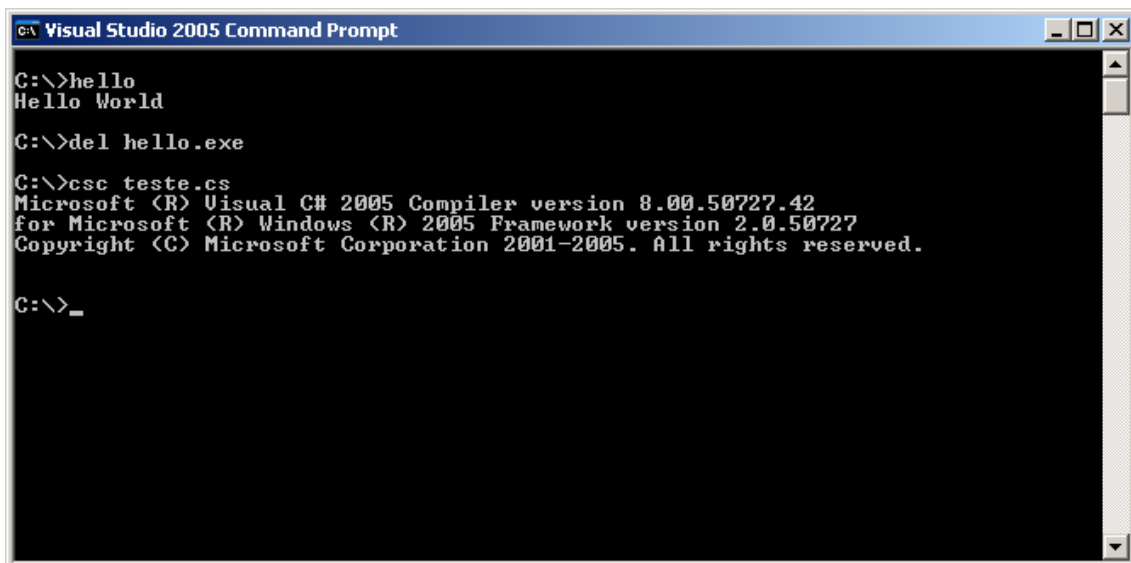
O programa será executado, no caso, será escrito a palavra "Hello World", no prompt semelhante ao que ocorreu com o exemplo da lição 1.

11 – Digite *del hello.exe* e tecele enter.

Isso apagará o arquivo executável.

12 – Digite *csc teste.cs*

Ao omitir o parâmetro */out* o compilador cria um arquivo executável com o mesmo nome do arquivo fonte. Será criado então o arquivo **teste.exe**, execute este arquivo para testá-lo.



```
Visual Studio 2005 Command Prompt
C:\>hello
Hello World
C:\>del hello.exe
C:\>csc teste.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.
C:\>_
```

13 – Abra novamente o arquivo **teste.cs**, pode ser no próprio notepad ou no Visual Studio .NET.

14 – Apague a primeira linha de código. Vai ficar assim:

```
namespace txtHello
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
{  
    class Class1  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World");  
        }  
    }  
}
```

15 – Compile e teste a aplicação, novamente fica a sua escolha, use o prompt de comando ou o Visual Studio .NET. Uma recomendação, compile manualmente pelo prompt para se familiarizar melhor com ele, já que não falaremos muito mais sobre ele no curso.

16 – Ao compilar ele vai emitir um erro. Como este:

teste.cs(7,4): error CS0246: The type or namespace name 'Console' could not be found (are you missing a using directive or an assembly reference?)

17 – Mude a linha com **Console.WriteLine("Hello World");** para **System.Console.WriteLine("Hello World");**

18 - Compile e teste novamente.

A compilação é concluída com sucesso e o programa funciona normalmente só que desta vez sem a diretiva **using System;**

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Como parte do Microsoft .NET Framework o C# pode fazer uso de uma série de classes de utilidades que executam uma gama de operações úteis. Essas classes são organizadas no que chamamos de **namespaces**, eles contem um conjunto de classes relacionadas e também podem conter outros namespaces.

System é um **namespace**. O namespace System é o mais importante porque contém as classes que a maior parte das aplicações utiliza para interagir com o sistema operacional.

A classe **Console** é uma classe do namespace System.

O método **Writeline** é um método da classe console que escreve uma mensagem no prompt de comando.

Os namespaces ajudam a reduzir a complexidade de um programa e a facilitar sua manutenção.

Podemos criar nossos próprios namespaces.

Programas pequenos e crianças pequenas têm uma coisa óbvia em comum, eles crescem. Com o crescimento de um programa surgem dois problemas:

1º - Quanto mais código maior a complexidade do programa e mais difícil sua manutenção.

2º - Mais código usualmente representa mais nomes de dados, funções, classes, etc. Facilitando os erros por conter nomes iguais.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

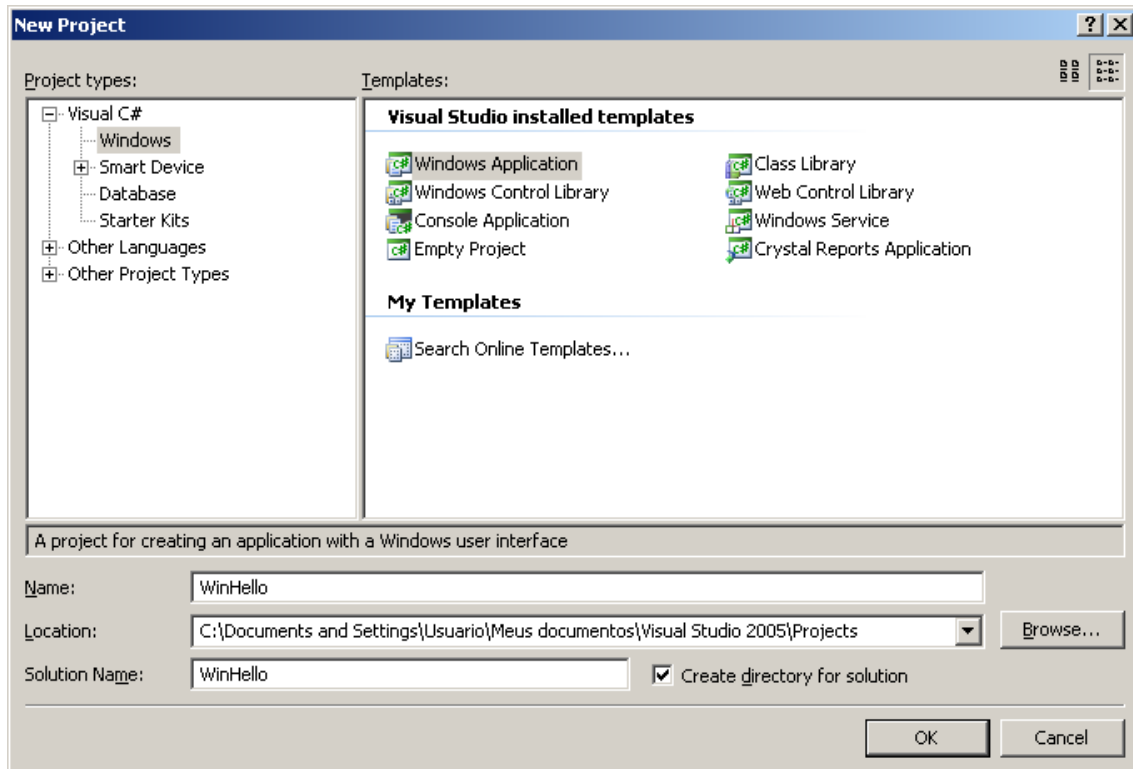
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Os namespaces tem a função de ajudar a solucionar esses dois problemas. Para usar um namespace lembre-se que é necessário fazer uma referencia ao mesmo através de uma diretiva **using** seguido do nome do namespace no **começo do bloco de código**. Você pode fazer referencia a vários namespaces no seu arquivo de código. Um em cada linha. Um após o outro.

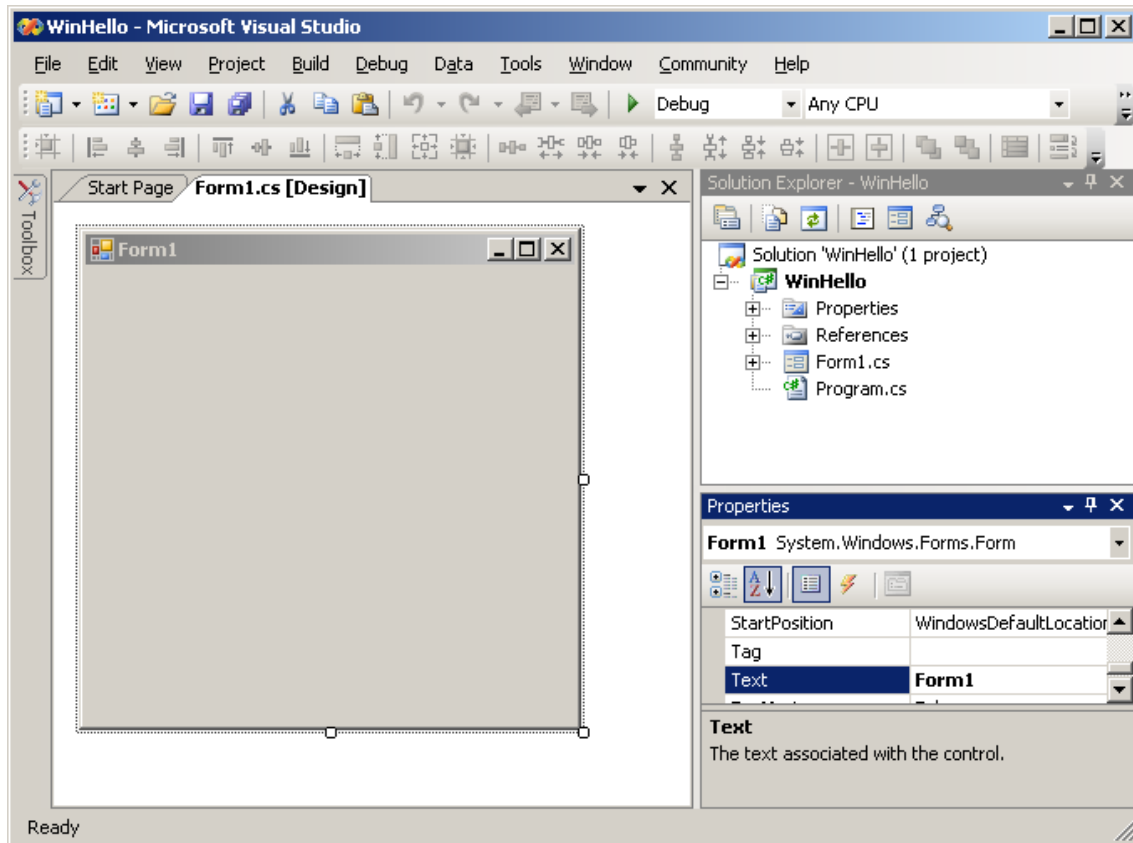
Até agora por motivo didático usamos somente o prompt de comando para criar os nossos exemplos. Como sabemos esse tipo de aplicação não é muito útil nos dias de hoje. O Visual Studio .NET conta com diversos recursos importantes para o desenvolvimento de aplicações Windows.

1 – Entre no Visual Studio .NET.

2 – Crie um novo projeto, só que desta vez do tipo **Windows Application**, chamado **WinHello**.



O Visual Studio .NET cria e mostra um formulário baseado em Windows no modo Design.



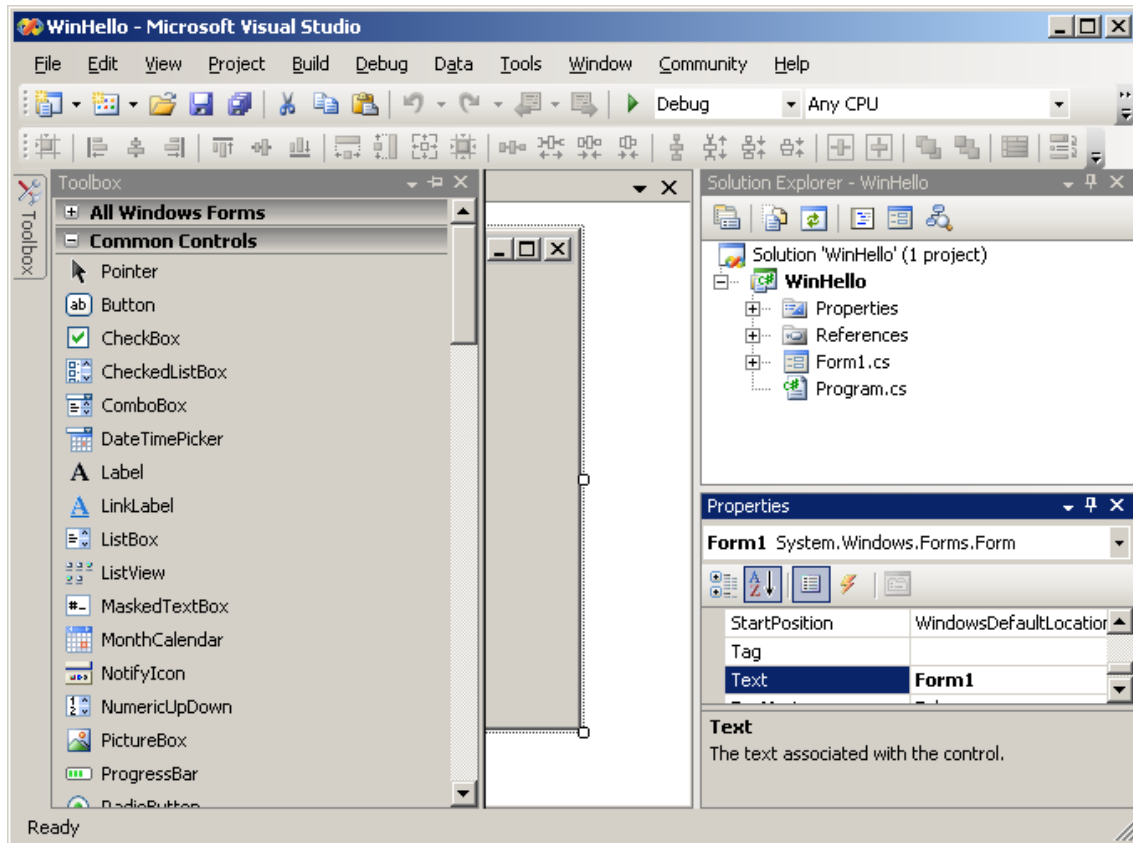
Vamos agora criar a nossa interface com o usuário.

3 – Na barra de ferramentas do Visual Studio .NET clique em **ToolBox**. O ícone da ToolBox aparece a esquerda do formulário. Você também pode localizar a ToolBox através do menu View > Toolbox.

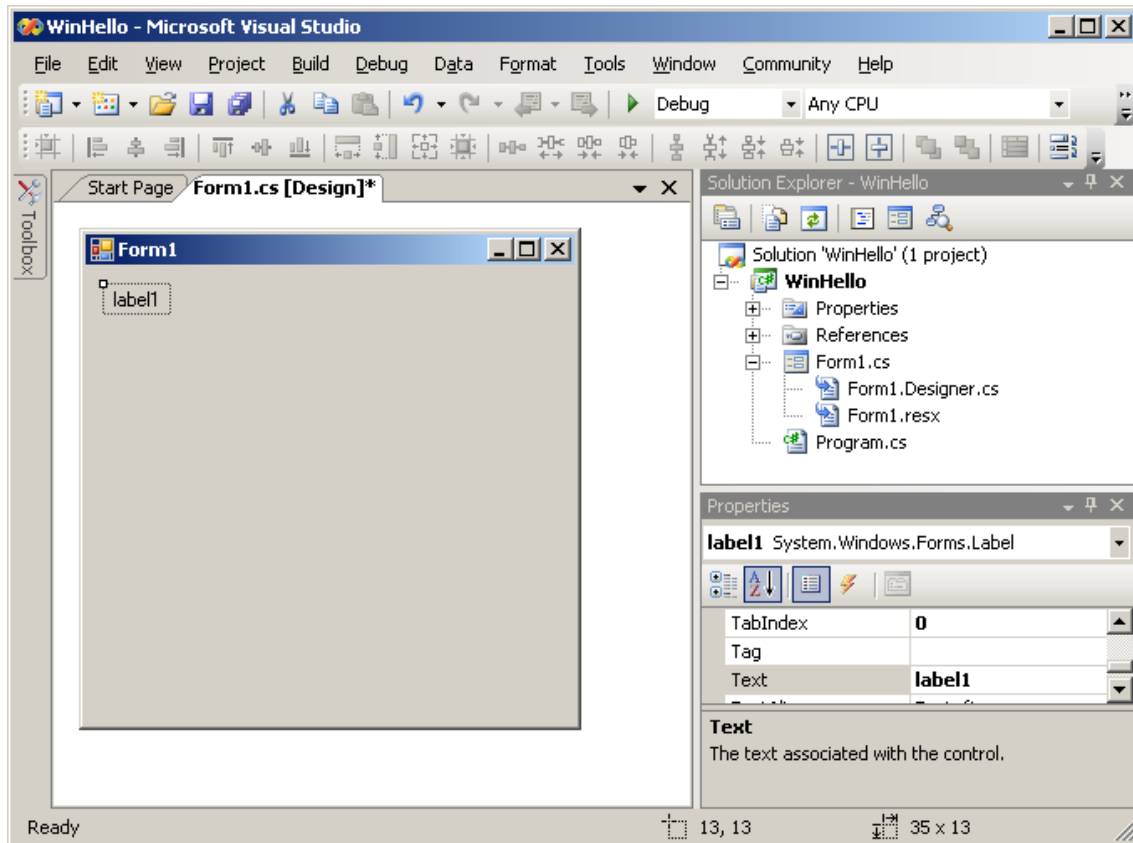
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

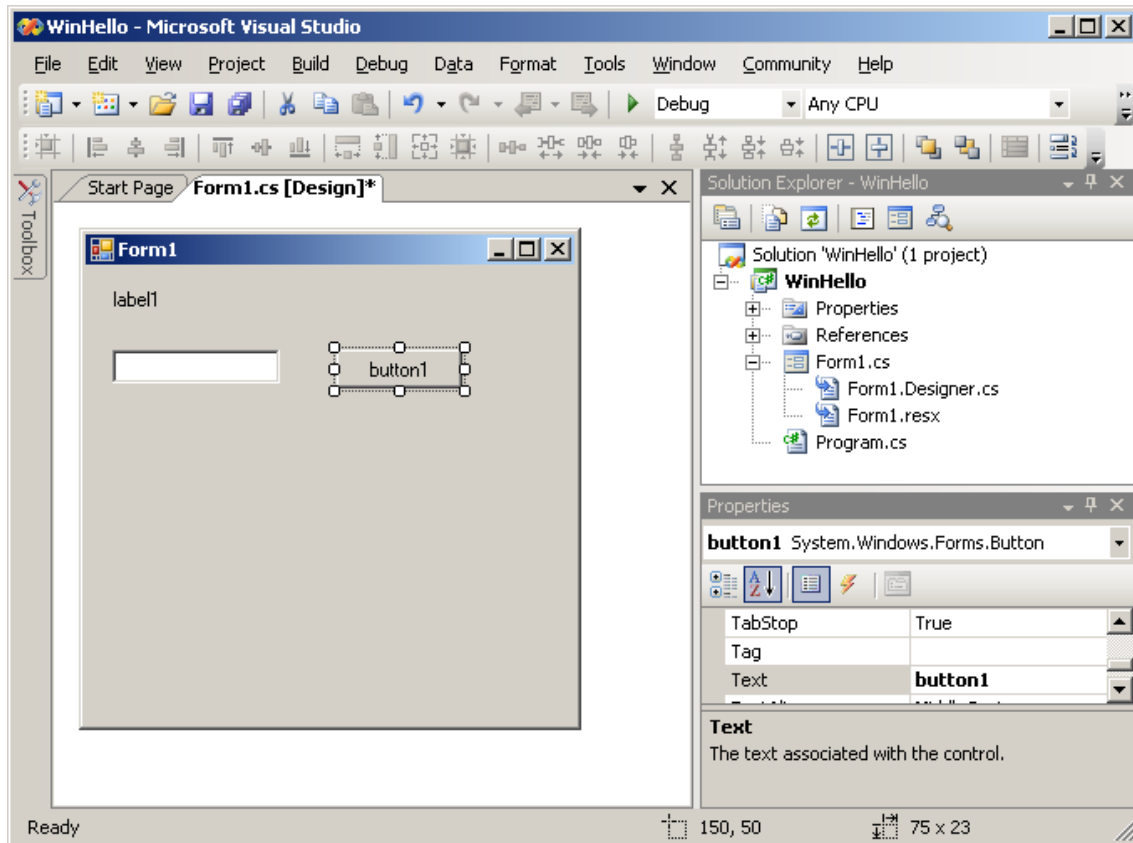


4 – Arraste da barra de ferramentas o controle **Label** e posicione-o no canto superior esquerdo do formulário.




Para colocar um controle no formulário você pode também dar um clique duplo sobre ele na barra de ferramentas ou clicar uma vez sobre ele na barra de ferramentas e depois clicar no formulário. O clique duplo posiciona o controle no canto superior esquerdo. A segunda opção coloca o controle no local onde você clicar.

5 – Coloque também no formulário um controle **TextBox** e um controle **Button**. Como na próxima ilustração:



6 – Na janela **Solution Explorer**, clique no botão **View Code**. 

O código do arquivo **Form1.cs** aparece.

Para voltar ao modo design, também na janela **Solution Explorer** clique em **View Design**. 

Form1.cs tem todo o código gerado automaticamente pelo Visual Studio .NET.

Note os seguintes elementos no código.

- As **diretivas** usadas no início do código referenciando aos **namespaces**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;
```

- O Visual Studio .NET usa o mesmo nome do projeto para criar o namespace principal.

```
namespace WinHello

{

...

}
```

- Uma classe chamada **Form1** dentro do namespace WinHello.

```
namespace WinHello

{

    public class Form1 ...

    {

        ...

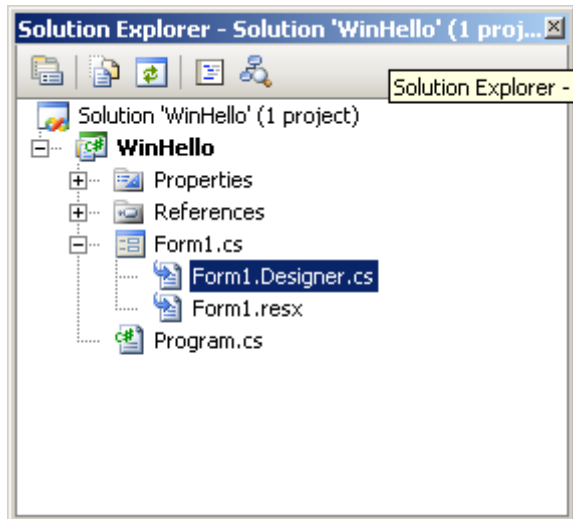
    }

}
```

- O **constructor** (construtor), é um método especial que tem o mesmo nome da classe. Ele é o primeiro método a ser executado quando o programa é iniciado.

```
public class Form1 ...  
  
{  
    ...  
  
    public Form1()  
    {  
        ...  
    }  
}
```

- Um método chamado **InitializeComponent**. O código dentro deste método configura as propriedades dos controles que adicionamos no modo Design. Atenção, não modifique o conteúdo do InitializeComponent diretamente no código, use a janela Properties no modo Design. Vamos aprender mais sobre os métodos nos próximos tutoriais. Este método está no arquivo Form1.designer.cs no Visual Studio .NET 2005.



```
private void InitializeComponent()  
{  
    this.label1 = new System.Windows.Forms.Label();  
    this.button1 = new System.Windows.Forms.Button();  
    this.textBox1 = new System.Windows.Forms.TextBox();  
    this.SuspendLayout();  
  
    //  
    // label1  
    //  
    this.label1.AutoSize = true;  
    this.label1.Location = new System.Drawing.Point(13, 13);  
    this.label1.Name = "label1";  
    this.label1.Size = new System.Drawing.Size(35, 13);  
    this.label1.TabIndex = 0;  
    this.label1.Text = "label1";  
  
    //  
    // button1  
    //  
    this.button1.Location = new System.Drawing.Point(150, 50);
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(75, 23);

this.button1.TabIndex = 1;

this.button1.Text = "button1";

this.button1.UseVisualStyleBackColor = true;

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(16, 50);

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(100, 20);

this.textBox1.TabIndex = 2;

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F,

13F);

this.AutoScaleMode =

System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(292, 273);

this.Controls.Add(this.textBox1);

this.Controls.Add(this.button1);

this.Controls.Add(this.label1);

this.Name = "Form1";

this.Text = "Form1";

this.ResumeLayout(false);

this.PerformLayout();

}...
```

Autor: Herbert Moroni Cavallari da Costa Gois


Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Vamos agora definir as propriedades dos controles que colocamos no Form.

7 – Volte para o modo Design.

Para voltar ao modo design, também na janela **Solution Explorer** clique em **View Design**. 

8 – De um clique sobre o **Button1** para selecioná-lo.

9 – Na janela **Properties**, altere a propriedade **Text** do **button1** para OK.

Se não localizar a janela **Properties**, clique em F4, ou no **menu View**, clique em **Properties Window**.

10 – Altere também a propriedade **Text** do **Label1** para *Digite o seu nome*

11 – Altere agora a propriedade **Text** do controle **textBox1** para *aqui*.

Note que as propriedades modificadas na janela **Properties** ficam em negrito. Assim você pode saber se elas estão com seu valor padrão ou não.

12 – Selecione o formulário clicando sobre ele.

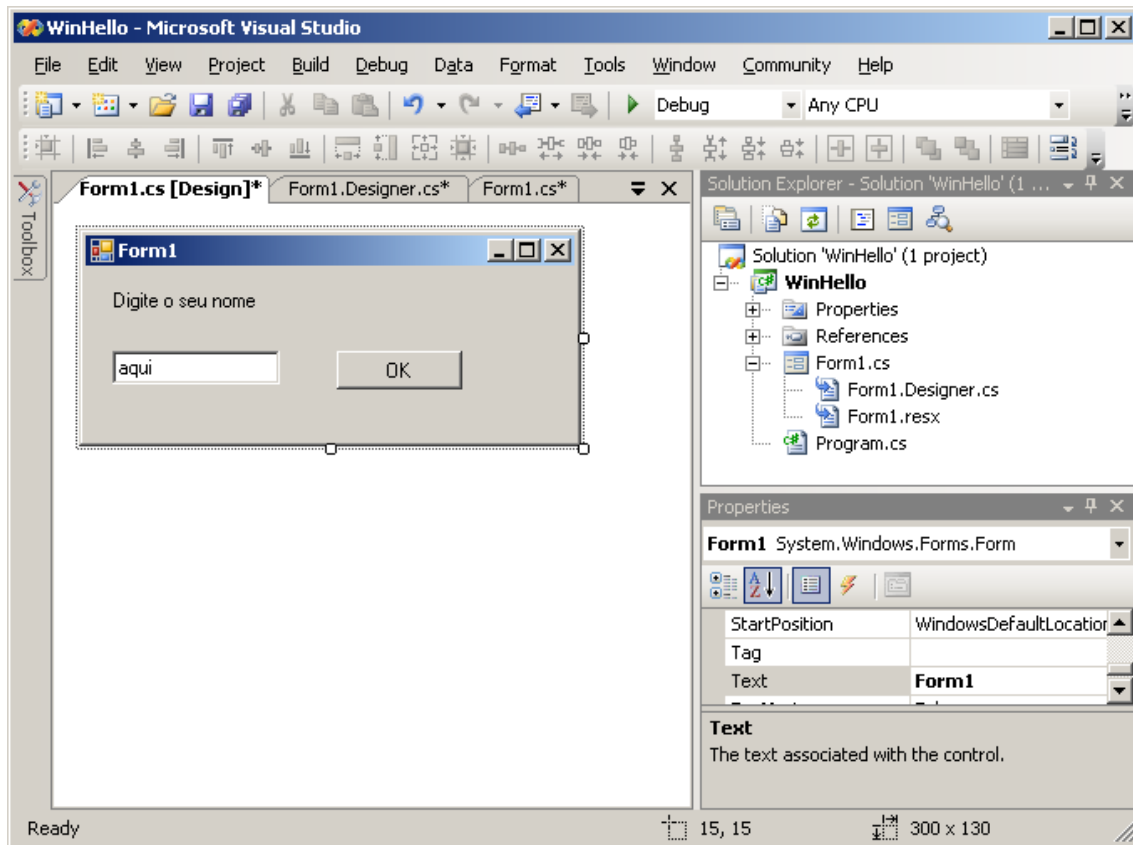
Note que aparecem alguns marcadores envolta do formulário. Eles ajudam a redimensionar o formulário.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

13 - Clique sobre o marcador central na parte de baixo do **Form1** e mantendo o botão pressionado arraste para cima.



Isso serve para os outros controles também, clique sobre os outros controles e note os marcadores.

Vamos agora escrever o código para o nosso programa.

14 – No painel de código de um clique duplo sobre o **Button1**.

Note que ele vai diretamente para o painel de código e é criado automaticamente o seguinte código.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
private void button1_Click(object sender, System.EventArgs e)
{

}
}
```

Tudo que for digitado dentro deste código será executado assim que o **Button1** for clicado quando o programa estiver executando.

15 - Digite o seguinte código:

Tenha atenção com esse código, ele deve ser digitado exatamente como se segue, lembre-se que o C# é case-sensitive. É necessário também o ponto-e-virgula no final da linha.

```
MessageBox.Show("Hello " + textBox1.Text);
```

Vai ficar assim:

```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Hello " + textBox1.Text);
}
}
```

16 – Execute o programa.

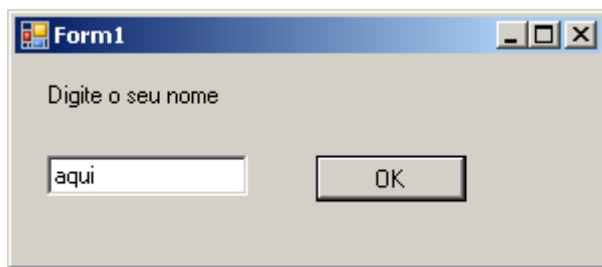
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

Para executar o programa você pode clicar e **F5**, ou no menu **Debug** clicar em **Start Debugging**.

Automaticamente o Visual Studio .NET salva o programa, compila e o executa. A seguinte janela aparece:



17 – Digite seu nome e clique em OK.

Uma janela aparece exibindo a mensagem "Hello seu-nome".



18 - Clique em **Ok** para fechar a janela com a mensagem.

19 – Na janela executando o **Form1** clique em fechar.

9.1 – Tipos de dados em C#

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Categoria	Tipo	Valores possíveis de se armazenar
Inteiro	byte	0 a 255 (8 bits)
Inteiro	sbyte	-128 a 127 (8 bits)
Inteiro	int	-2,147,483,648 a 2,147,483,647 (32 bits)
Inteiro	uint	0 a 4,294,967,295 (32 bits)
Inteiro	long	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (64 bits)
Inteiro	ulong	0 a 18,446,744,073,709,551,615 (64 bits)
Inteiro	short	-32,768 a 32,767 (16 bits)
Inteiro	ushort	0 a 65,535 (16 bits)
Real	decimal	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$ (128 bits)
Real	double	$\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$ (64 bits)
Real	float	$\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$ (32 bits)
Lógico	bool	Verdadeiro ou Falso (Valores booleanos)
Literal	char	Um caractere (16 bits)
Literal	string	Seqüência de caracteres (16 bits por caractere)

Em C# os tipos **uint**, **ulong** e **ushort**, ocupam o mesmo espaço que os tipos **int**, **long** e **short**. Só que só podem armazenar números positivos.

Os tipos **uint**, **ulong** e **ushort** são conhecidos como **Tipos não assinados**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

9.2 – Declarando Variáveis em C#

Antes de qualquer coisa segue a lista das palavras reservadas em C#, lembre-se de que elas não podem ser usadas na criação de identificadores, ou seja, não podem ser usadas como nome de variáveis.

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
in	int	interface	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

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Declaramos especificando o tipo de dado seguido do nome da variável como no exemplo:

```
int contador;
```

Esse exemplo declara uma variável chamada contador do tipo *integer*.

Podemos também declarar múltiplas variáveis de uma vez, fazemos isso da seguinte maneira:

```
int contador, numeroCarro;
```

Estamos declarando nesse exemplo duas variáveis do tipo integer, uma chamada contador e a outra numeroCarro.

9.3 – Constantes em C#

Para declarar uma constante em C# você usa o comando Const na declaração da variável que será a constante. Exemplo:

```
const double pi = 3.1415;
```

9.4 – Operadores de atribuição em C#

Depois de declarar sua variável você precisa atribuir um valor a ela. No C# você não pode usar uma variável antes de colocar um valor nela, isso gera um erro de compilação.

Exemplo de como atribuir um valor a uma variável:

```
int numeroFuncionario;
```

```
numeroFuncionario = 23;
```

Primeiro nós declaramos nossa variável do tipo integer. Depois atribuímos o valor 23 a ela. Entendemos pelo sinal de igual como recebe. Assim numeroFuncionario recebe 23.

Podemos também atribuir um valor a variável quando a declaramos, dessa forma:

```
int numeroFuncionario = 23;
```

Isso faz à mesma coisa que o exemplo anterior, só que tudo em uma linha.

Mais um exemplo:

```
char letraInicial = 'M';
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

9.5 - Operadores aritméticos em C#

Operador	Representação em C#	Exemplo
Incremento	++	a ++ Neste caso é somado 1 ao valor de a.
Decremento	--	a -- Neste caso é subtraído 1 do valor de a.
Multiplicação	*	a * b Multiplica-se o valor de a por b.
Divisão	/	a / b Divide-se o valor de a por b.
Módulo	%	a % b Retorna o resto da divisão de a por b.
Adição	+	a + b Soma de a e b.
Subtração	-	a - b Subtração de a com b.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

9.6 – Operadores relacionais em C#

Operador	Representação em C#	Exemplo
Maior	>	<p>$a > b$</p> <p>Se o valor de a for maior que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.</p>
Maior ou igual	>=	<p>$a \geq b$</p> <p>Se o valor de a for maior ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.</p>
Menor	<	<p>$a < b$</p> <p>Se o valor de a for menor que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.</p>
Menor ou igual	<=	<p>$a \leq b$</p> <p>Se o valor de a for menor ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.</p>
Igual a	==	$a == b$

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		Se o valor de a for igual ao valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Diferente de	!=	a != b Se o valor de a for diferente do valor de b então o resultado dessa expressão é <u>verdadeiro</u> senão é falso.

9.7 - Operadores Lógicos em C#

Operador	Representação em C#	Exemplo
E	&&	a == 5 && b > 9 Caso o valor de a seja igual a cinco e o valor de b maior que nove o resultado é verdadeiro, senão é falso.
Ou		a == 5 b > 9 Se o resultado de a for igual a cinco ou o valor de b for maior que nove então o resultado é verdadeiro. O resultado

Autor: Herbert Moroni Cavallari da Costa Gois

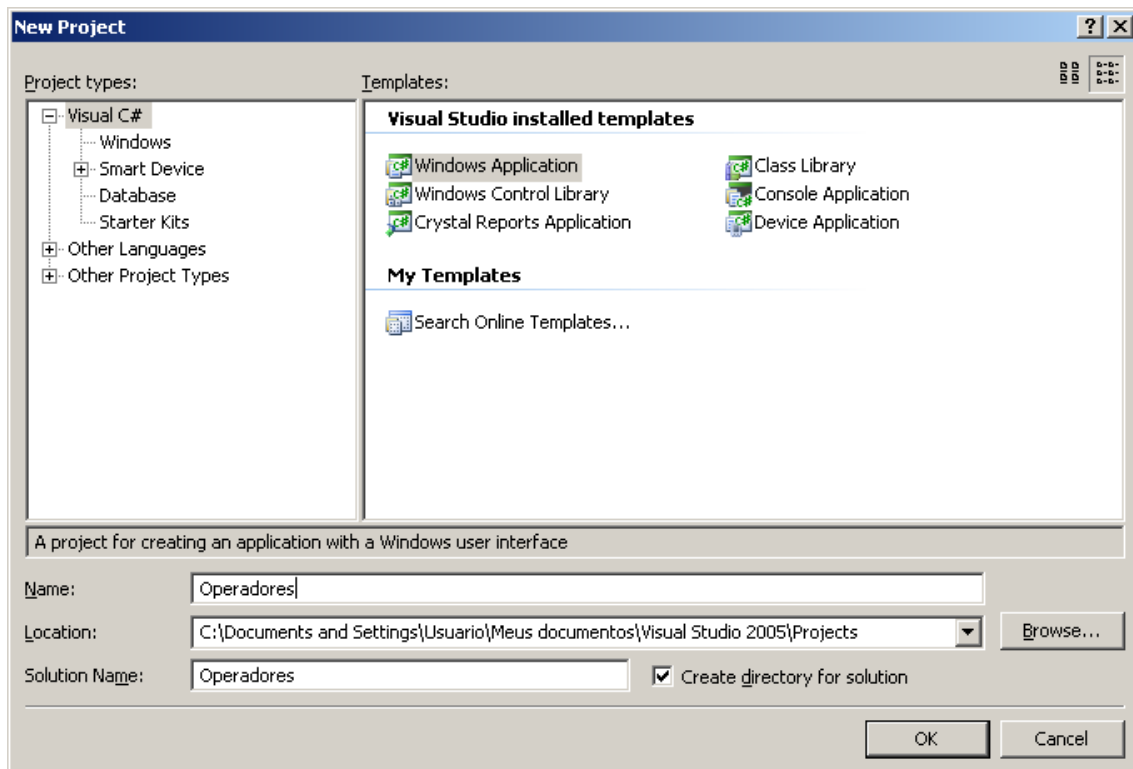
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		só será falso é as duas expressões retornarem falso.
Não	!	! a == 5 Se o resultado de a for igual a 5 então o resultado será falso, o operador não inverte o resultado da expressão.

Vamos fazer um exemplo pratico de como utilizar os operadores.

1 – Crie um novo projeto no Visual Studio do tipo Windows Application chamado **Operadores**.



Autor: Herbert Moroni Cavallari da Costa Gois

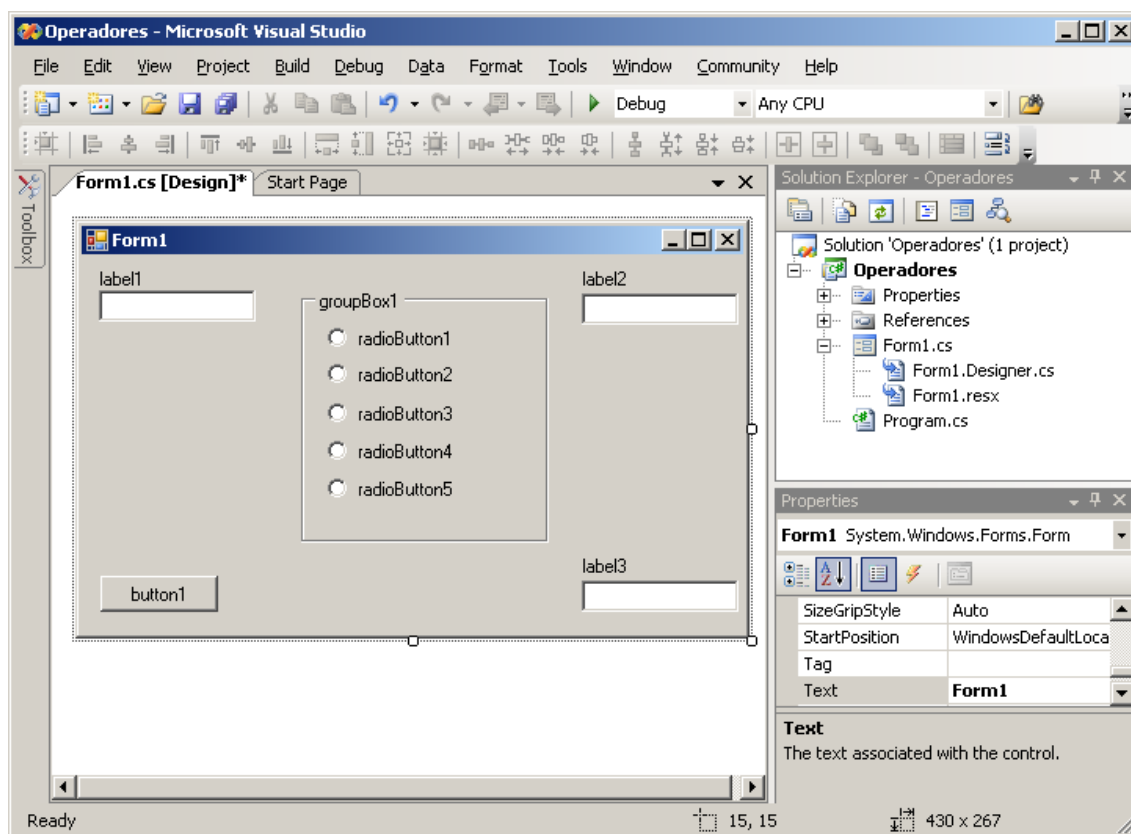
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

2 – Arraste para o **Form1** os seguintes controles:

- 3 Label
- 3 TextBox
- 1 Button
- 1 GroupBox
- 5 RadioButton

3 – Organize-os como a figura abaixo:



4 – Configure as propriedades dos controles conforme a tabela abaixo:

Controle	Propriedade	Valor
----------	-------------	-------

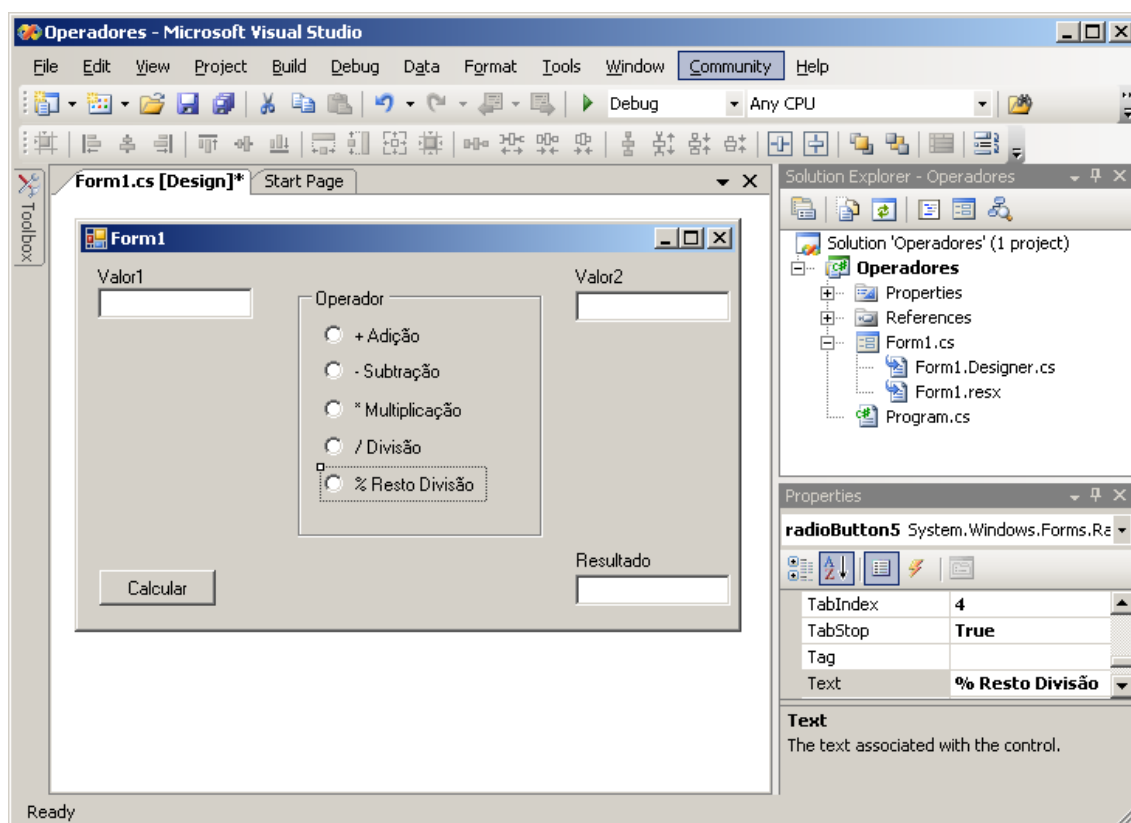
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Label1	Text	Valor1
Label2	Text	Valor2
Label3	Text	Resultado
Button	Text	Calcular
GroupBox	Text	Operador
RadioButton1	Text	+ Adição
RadioButton2	Text	- Subtração
RadioButton3	Text	* Multiplicação
RadioButton4	Text	/ Divisão
RadioButton5	Text	% Resto Divisão

Vai ficar como a figura abaixo:



5 – Execute a aplicação.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Clique na **TextBox1**, pressione a tecla **Tab** do teclado, verifique que conforme você clica na tecla ele passa de um controle a outro.

Clique sobre os operadores, veja que você só pode selecionar um, essa é uma característica do controle **RadioButton** que não tinha sido usado até aqui. A **GroupBox** agrupa todos os **RadioButtons** dentro dela de forma que apenas um deles pode ser selecionado.

6 – Finalize a execução, para isso você pode simplesmente fechar a janela do **Form1** ou clicar no botão **Stop Debugging** na barra de ferramentas.

7 - Vamos agora digitar o código que efetuará os cálculos, esse código vai ser executado quando o botão **Calcular** for clicado. De um clique duplo sobre o **Button1** para digitarmos o seguinte código. (Atenção na hora de digitar, lembre-se que o C# é case-sensitive ou seja, diferencia maiúsculas de minúsculas).

```
long primeiroValor, segundoValor, resultado;
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
primeiroValor = int.Parse(textBox1.Text);
segundoValor = int.Parse(textBox2.Text);

if (radioButton1.Checked)
{
    resultado = primeiroValor + segundoValor;
    textBox3.Text = resultado.ToString();
}
else if (radioButton2.Checked)
{
    resultado = primeiroValor - segundoValor;
    textBox3.Text = resultado.ToString();
}
else if (radioButton3.Checked)
{
    resultado = primeiroValor * segundoValor;
    textBox3.Text = resultado.ToString();
}
else if (radioButton4.Checked)
{
    resultado = primeiroValor / segundoValor;
    textBox3.Text = resultado.ToString();
}
else if (radioButton5.Checked)
{
    resultado = primeiroValor % segundoValor;
    textBox3.Text = resultado.ToString();
}
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
}

```

Digitamos nosso código dentro do procedimento:

```
private void button1_Click(object sender, System.EventArgs e)
{

}

```

Esse procedimento é executado sempre que o **Button1** é clicado. Sendo assim quando o botão é clicado, primeiro:

Declaramos as variáveis **primeiroValor**, **segundoValor** e **resultato** do tipo **Long**. O tipo **Long** é usado aqui porque armazena uma grande variedade de números. Perceba que declaramos as três variáveis em apenas uma linha de código.r.

```
long primeiroValor, segundoValor, resultado;

```

Depois atribuímos os valores das caixas de texto as variáveis **primeiroValor** e **segundoValor**.

```
primeiroValor = int.Parse(textBox1.Text);
segundoValor = int.Parse(textBox2.Text);

```

Como o valor que esta na caixa de texto é do tipo **string** convertemos para **int** usando **int.Parse**. Você pode usar o método Parse sempre que precisar converter um tipo de dados em outro. Para usa-lo digite o tipo do dado ponto Parte como o exemplo acima ou o seguinte exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
long.Parse(textBox2.Text);
```

Voltando ao nosso exemplo, depois de atribuir os valores as variáveis vamos verificar qual operação executar, para isto usamos a propriedade **checked** de cada **RadioButton**.

```
if (radioButton1.Checked)
```

ou

```
else if (radioButton2.Checked)
```

Se o valor da propriedade Checked do RadioButton for True quer dizer que ele esta selecionado, então executamos o calculo correspondente ao RadioButton selecionado e atribuímos o resultado a variável **resultado**.

Agora atribuímos o valor da variável resultado à propriedade **Text** do **textbox3** para que seja exibida na tela.

```
textBox3.Text = resultado.ToString();
```

Perceba que precisamos converter o valor da variável resultado para String. Fizemos isto utilizando o método ToString. Você pode usar o método ToString sempre que precisar converter um tipo de dados para string.

8 – Execute o programa.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

9 – Digite 9 para o valor1 e 2 para o valor2, selecione adição e clique em calcular.

Form1

Valor1: 9

Operador:

- ☒ + Adição
- ☐ - Subtração
- ☐ * Multiplicação
- ☐ / Divisão
- ☐ % Resto Divisão

Valor2: 2

Resultado: 11

Calcular

Faça testes com os outros operadores.

Subtração:

Form1

Valor1: 9

Operador:

- ☐ + Adição
- ☒ - Subtração
- ☐ * Multiplicação
- ☐ / Divisão
- ☐ % Resto Divisão

Valor2: 2

Resultado: 7

Calcular

Multiplicação:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Form1

Valor1: 9

Operador:

- ☐ + Adição
- ☐ - Subtração
- ☒ * Multiplicação
- ☐ / Divisão
- ☐ % Resto Divisão

Valor2: 2

Resultado: 18

Calcular

Divisão, perceba que ele retorna um resultado em inteiro, o numero inteiro mais próximo do resultado:

Form1

Valor1: 9

Operador:

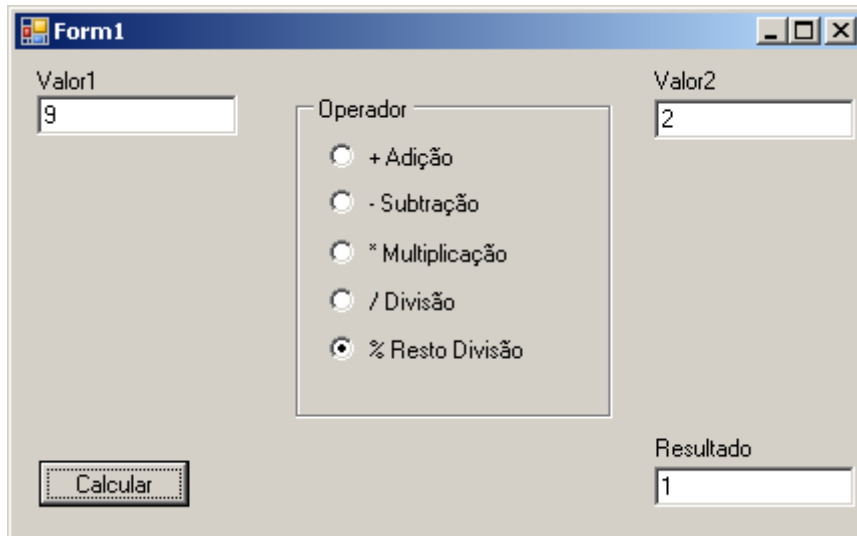
- ☐ + Adição
- ☐ - Subtração
- ☐ * Multiplicação
- ☒ / Divisão
- ☐ % Resto Divisão

Valor2: 2

Resultado: 4

Calcular

Resto da Divisão, perceba que ele retorna o resto da divisão, 9 dividido por dois resulta em 4 com resto 1.



10 – Pare a aplicação.

9.8 - Entrada e Saída de dados em C#

Para entrada de dados em C# usamos o comando **ReadLine**, e para a Saída de dados o comando **WriteLine**.

Segue um exemplo:

```
string nome;  
  
Console.WriteLine("Digite seu nome");  
  
nome = Console.ReadLine();  
  
Console.WriteLine("Seu nome é: " + nome);
```

No exemplo acima, primeiro declaramos uma variável do tipo **string** chamada **nome**, depois usamos o comando de saída **WriteLine** para imprimir **Digite seu nome na tela**. Na terceira linha atribuímos o valor que foi digitado à variável

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

nome usando o comando de entrada **ReadLine**. Por ultimo exibimos na tela o nome digitado novamente usando o comando **WriteLine**. O operador mais foi usado para concatenar, ou seja, para unir o texto **Seu nome é** com o conteúdo da variável **nome**.

9.9 – Estruturas de decisão em C#

A primeira estrutura de decisão que iremos estudar é o **IF**, veja a sintaxe a seguir:

```
if ( expressãocondicional )
{
    bloco-de-codigo1
}
else
{
    bloco-de-codigo2
}
```

Se o resultado da expressão condicional for VERDADEIRO então o bloco-de-codigo1 será executado, se a expressão for FALSO, então o bloco-de-codigo2 será executado.

Se o bloco-de-codigo1 e bloco-de-codigo2 for apenas uma linha de código não é necessário usar os colchetes, como nos exemplos abaixo.

Veja o seguinte exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
double Salario = 1200;
```

```
if (Salario > 1000)  
    Salario *= 1.1;
```

Existe uma variável chamada Salario que está definida e contém o valor 1200. Como **toda condição retorna apenas verdadeiro ou falso**, É verificado se o valor é maior que 1000, e caso seja verdadeiro será somado 10% ao Salario. Se Salario for menor que 1000 nada será executado.

Vamos a mais alguns exemplos para facilitar o entendimento.

```
double Salario = 1200;
```

```
if (Salario < 500)  
    Salario += 50;  
else  
    Salario += 100;
```

Aqui é verificado se o valor é menor que 500 e dependendo da condição é somado 50 ou 100, pois há o Else. Então se a variável for menor que 500 adicionamos nela o valor 50, senão adicionamos 100.

```
double Salario = 1200;
```

```
if (Salario < 500)
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
    Salario += 50;

else if ((Salario >= 500) && (Salario < 2000))

    Salario += 100;

else

    Salario += 250;
```

Já no exemplo acima existem 3 condições, onde o primeiro IF soma 50, o segundo 100 e o ELSE 250. **O Else sempre é executado quando nenhuma expressão é verdadeira.**

É possível avaliar diversos **else ifs** com uma determinada expressão. Como no exemplo abaixo:

```
double Salario = 1200;

if (Salario < 500)

    Salario += 50;

else if ((Salario >= 500) && (Salario < 600))

    Salario += 100;

else if ((Salario >= 500) && (Salario < 700))

    Salario += 110;

else if ((Salario >= 500) && (Salario < 800))

    Salario += 120;

else

    Salario += 250;
```

Perceba que no bloco de código acima usamos o &&.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

O && é usado quando precisamos testar mais de uma expressão condicional.

Exemplo:

```
else if ((Salario >= 500) && (Salario < 600))
```

Nesta linha testamos se o Salario é maior ou igual a 500 e o Salario é menor que 600.

Então se o Salario for 553 a expressão acima é VERDADEIRA, caso contrario é FALSA.

A segunda estrutura de decisão que iremos estudar é conhecida como **switch**.

Essa estrutura de decisão seleciona o código a ser executado baseado no valor de uma expressão.

A sintaxe é a seguinte:

```
switch (testexpression)
{
    case constant-expression:
        statements
        break:
    default:
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
statements  
  
}
```

O funcionamento ocorre da seguinte maneira: a expressão é obtida no **switch** e para cada **Case** existe uma condição a ser validada. Caso o **Case** seja verdadeiro, então a linha ou o bloco de código é executado. Se nenhum dos Cases for válido, então **default** é executado. O **default** é opcional e você pode ter quantos Cases for necessário. A cada **Case** é preciso declarar o **break**, senão o programa continua avaliando todos os **Cases**.

Veja o exemplo:

```
int diaDaSemana = 3;  
  
switch (diaDaSemana)  
{  
    case 1:  
        MessageBox.Show("Domingo");  
        break;  
    case 2:  
        MessageBox.Show("Segunda-Feira");  
        break;  
    case 3:  
        MessageBox.Show("Terça-Feira");  
        break;  
    case 4:  
        MessageBox.Show("Quarta-Feira");  
        break;
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
case 5:
    MessageBox.Show("Quinta-Feira");
    break;
case 6:
    MessageBox.Show("Sexta-Feira");
    break;
case 7:
    MessageBox.Show("Sabado");
    break;
}
```

Primeiro criamos uma variável do tipo **int** e atribuímos a mesma o valor 3.

A seguir iniciamos a estrutura colocando como expressão de teste o valor da variável.

O primeiro **Case** verifica se o valor da variável ou expressão de teste é 1. Se for o código que escreve na tela o texto *Domingo* é executado e a estrutura é finalizada não verificando os outros Cases. Caso contrario ele vai pro próximo case e segue como o primeiro até o fim da estrutura. Se nenhum dos cases se enquadrar no valor da expressão de teste nenhum código será executado, para isso que serve o **default**. No próximo exemplo vamos apenas programar o uso do **default**:

```
int diaDaSemana = 3;

switch (diaDaSemana)
{
    case 1:
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        MessageBox.Show( "Domingo" );

        break;

    case 2:

        MessageBox.Show( "Segunda-Feira" );

        break;

    case 3:

        MessageBox.Show( "Terça-Feira" );

        break;

    case 4:

        MessageBox.Show( "Quarta-Feira" );

        break;

    case 5:

        MessageBox.Show( "Quinta-Feira" );

        break;

    case 6:

        MessageBox.Show( "Sexta-Feira" );

        break;

    case 7:

        MessageBox.Show( "Sabado" );

        break;

    default:

        MessageBox.Show( "Dia Inválido" );

        break;

}
```

Aqui se o resultado da expressão de teste não for verdadeiro em nenhum **Case**, então será executado o **default**, escrevendo na tela que o valor informado é

invalido, o que não é o caso para o nosso exemplo, já que o valor da expressão se enquadra no **Case 3** escrevendo *Terça-Feira*.

Lembre-se do seguinte a respeito do switch:

- O **switch** deixa o código mais estruturado em relação ao uso de vários **else if** seguidos;
- Você só pode usar o **switch** com tipos de dados primitivos, como: (**int, long, float, double, decimal, string, char, bool**). Para usar outros tipos de dados use o **if**;
- A expressão do **case** precisa ser uma expressão única, em outras palavras, não é permitido escrever duas expressões em um mesmo case.
- Você precisa repetir a sintaxe do **case** para cada valor individual que você quer avaliar, mesmo que ele execute o mesmo bloco de código, veja o ultimo exemplo.
- Você deve usar o **break** após cada bloco de código senão ele vai continuar executando os outros **cases**.

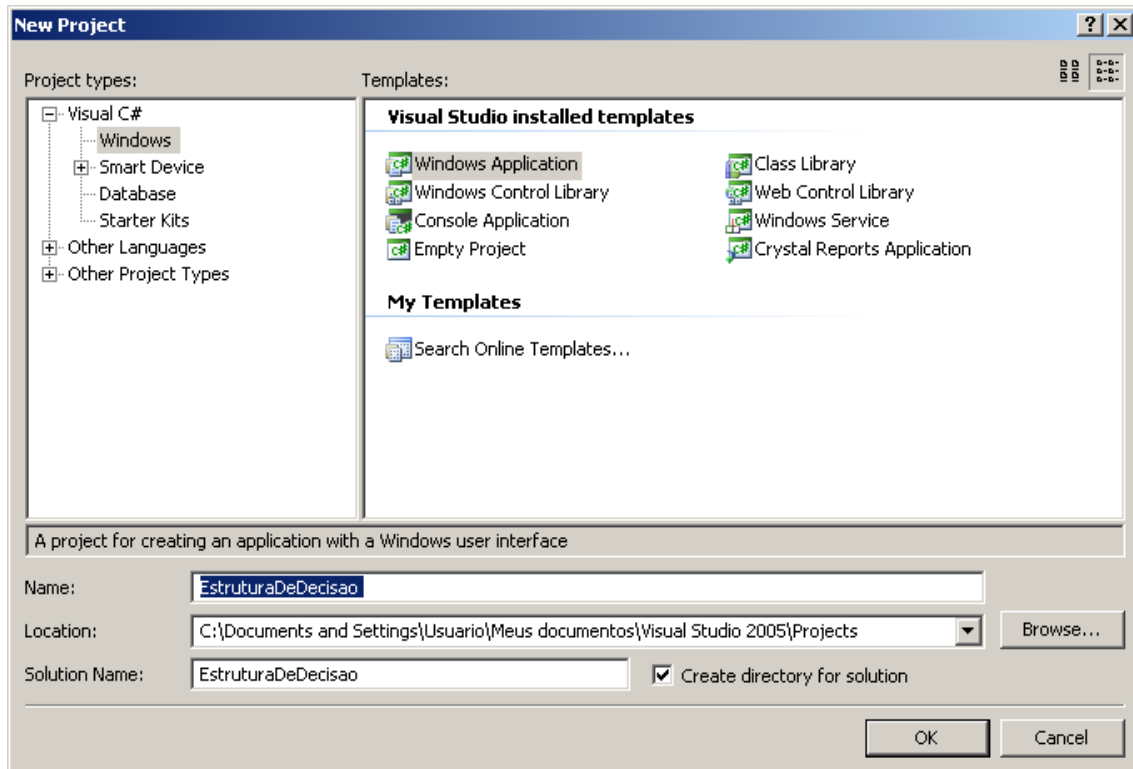
Vamos fazer um programinha que valida o usuário, verificando se é ele mesmo através do seu nome e senha.

1 – Entre no Visual Studio e crie uma aplicação chamada EstruturaDeDecisao do tipo Windows Application.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



2 – Arraste para o Form os seguintes controles:

- 2 - Label
- 2 – TextBox
- 1 – Button

3 – Mude as propriedades dos controles como a tabela abaixo:

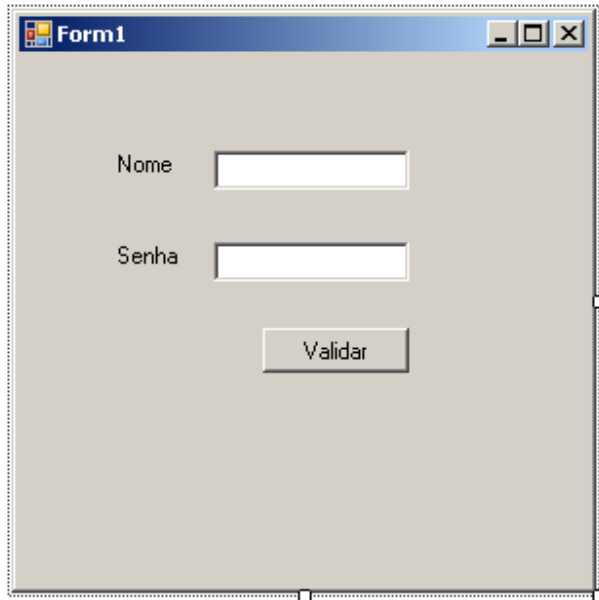
Controle	Propriedade	Valor
Label1	Text	Nome
Label2	Text	Senha
Textbox1	Text	
Textbox2	Text	
Button1	Text	Validar

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

4 – Organize-os para ficarem assim:

A screenshot of a Windows application window titled "Form1". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main area of the form is light gray. It contains two text input fields. The first field is preceded by the label "Nome" and the second by "Senha". Below these two fields is a single button with the text "Validar".

5 – De um clique duplo no **button1** e digite o seguinte código:

```
string Nome = textBox1.Text;
string Senha = textBox2.Text;

if ((Nome == "Moroni") && (Senha == "123"))
    MessageBox.Show("Bem Vindo Moroni");
else
{
    MessageBox.Show("Usuario Invalido");
    Close();
}
```

Autor: Herbert Moroni Cavallari da Costa Gois

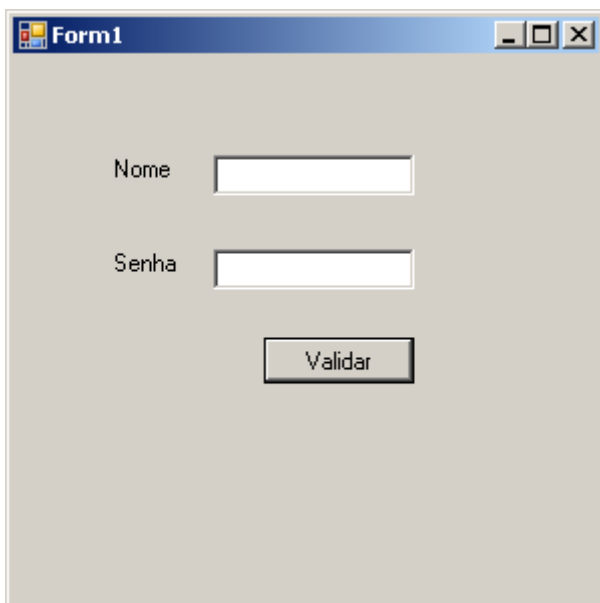
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Você pode substituir o valor do nome para o seu nome e sua senha.

Perceba que tivemos que abrir e fechar colchetes ({ }) depois do else, sempre temos que fazer isso quando tivermos mais que uma linha de código, isso se aplica ao else, e também depois do if, se tivéssemos mais de uma linha de código no bloco de código acima teríamos que fazer o mesmo.

6 – Execute o programa.

A screenshot of a Windows application window titled "Form1". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main area of the form is light gray. It contains two text input fields. The first is labeled "Nome" and the second is labeled "Senha". Below these fields is a button labeled "Validar".

7 – Digite Moroni em nome (ou seu nome) e a senha 123.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



8 – Clique em validar.

É executado o seguinte código:

```
MessageBox.Show("Bem Vindo Moroni");
```

Fazendo com que a seguinte mensagem apareça.



9 – Clique em Ok.

10 – Digite outro valor qualquer na caixa de texto e clique em OK.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

E executado o seguinte código:

```
MessageBox.Show("Usuario Invalido");  
Close();
```

Fazendo com que a seguinte mensagem apareça.



E que o programa seja encerrado.

Em resumo:

Declaramos duas variáveis e demos as elas os valores dos controles de texto. Depois testamos se o valor do nome era Moroni e a Senha 123, você pode substituir o valor para o seu nome e sua senha. Se o resultado da expressão for verdadeira então ele da a mensagem de bem vindo, caso contrario ele da a mensagem de usuário invalido e sai do programa.

9.10 – Estruturas de repetição em C#

O looping **While** é executado sempre associado a uma condição, ou seja, a cada passagem pelo looping a condição é avaliada. Veja a sintaxe a seguir:

Autor: Herbert Moroni Cavallari da Costa Gois

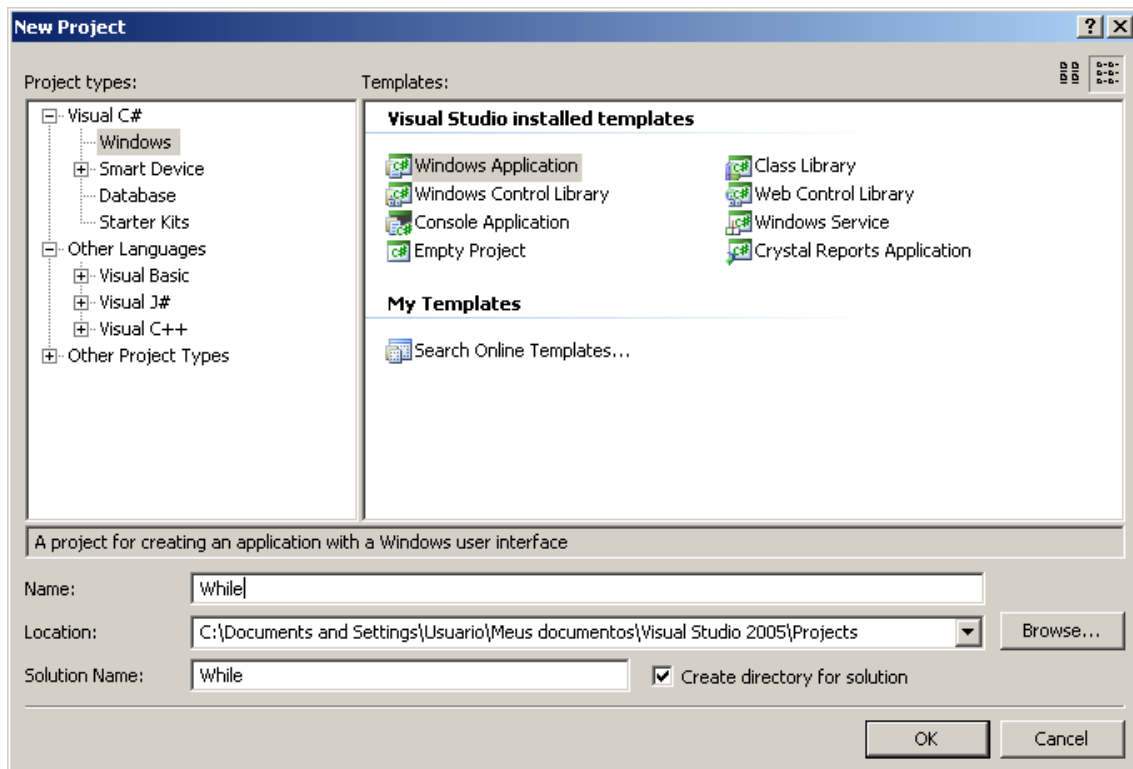
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
while ( booleanExpression )  
    statement
```

Vamos fazer um exemplo para você compreender melhor o funcionamento do while.

1 - Crie um novo projeto no Visual Studio, novamente do tipo **Windows Application**, chame o projeto de **while**.



2 - Adicione no **Form1** os seguintes controles:

- 1 Button
- 1 ListBox

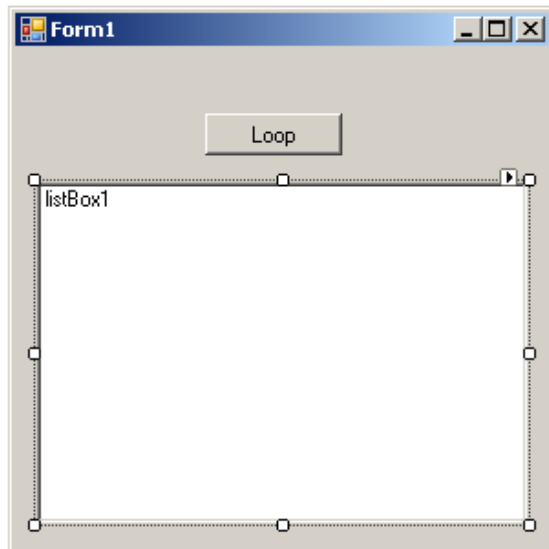
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

3 - Mude a propriedade **Text** do **Button1** para "Loop".

4 - Organize os controles como a figura abaixo:



5 - De um clique duplo sobre o **Button1** e digite o seguinte código:

```
int contador = 0;

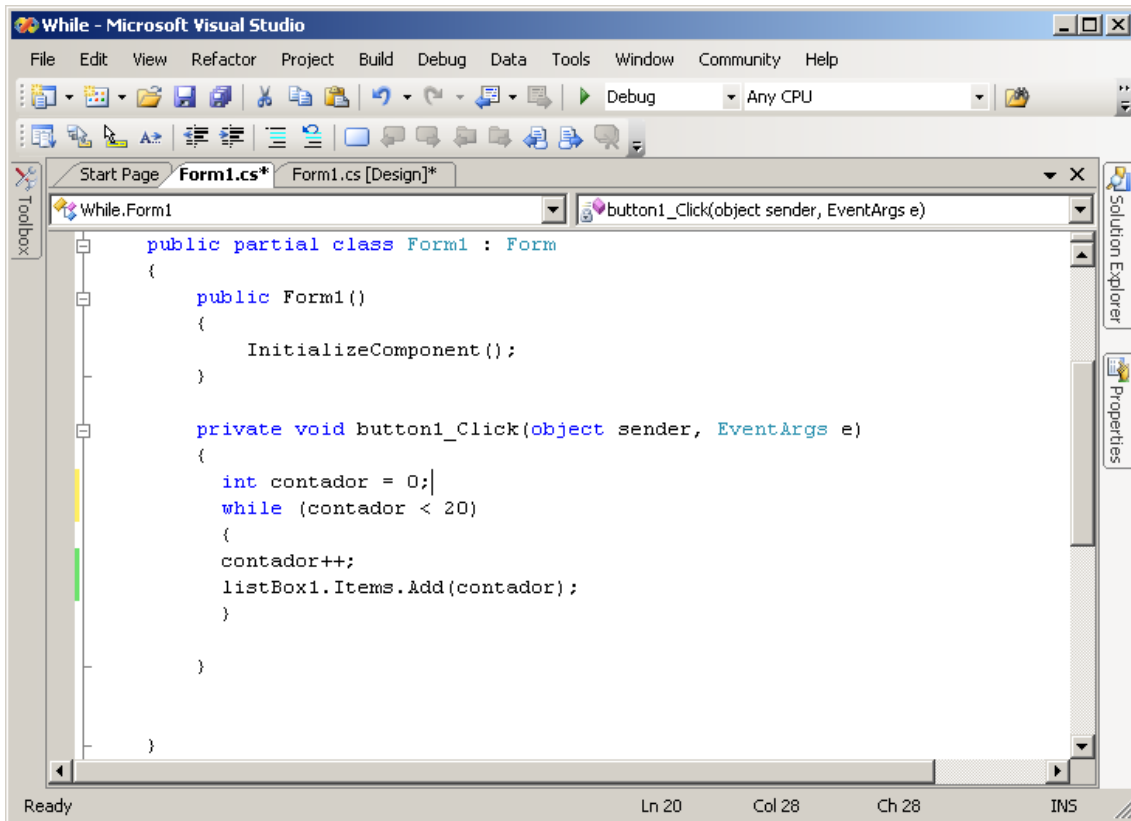
while (contador < 20)
{
    contador++;
    listBox1.Items.Add(contador);
}
```

Vai ficar assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Esse é nosso código, vamos estudá-lo um pouco:

```
int contador = 0;

while (contador < 20)
{
    contador++;

    listBox1.Items.Add(contador);
}
```

Na primeira linha declaramos uma variável chamada **contador** do tipo **integer** e atribuímos a essa variável o valor 0.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

A segunda linha começa o **While** com a condição. Essa linha quer dizer enquanto contador for menor que 20, ou seja ele vai executar o loop ou o código que esta entre os colchetes até que o conteúdo da variável **contador** seja igual ou maior que 20.

A linha quatro soma 1 ao conteúdo da variável contador, a linha a seguir tem o mesmo significado:

```
contador = contador + 1
```

Entretanto da forma que fizemos no nosso exemplo é mais elegante porque não precisamos repetir o nome da variável, se apenas fizéssemos assim:

```
contador = 1
```

Ele atribuiria 1 ao conteúdo da variável e o nosso looping se tornaria um loop infinito ou seja nunca ia parar de rodar porque nunca entraria na condição que o faz parar. Cuidado com esse tipo de loop, seus loops nunca devem ser infinitos.

A linha 5 atribui ao **ListBox** o valor da variável.

Fique atento para o seguinte quando estiver utilizando o WHILE:

- A expressão condicional deve retornar sempre um valor Boolean ou seja, verdadeiro o falso.
- A expressão condicional também deve estar sempre entre parênteses.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- Se na primeira passagem o resultado da expressão condicional for falso, o código do While não será executado.
- Se você for usar mais de uma linha de código dentro do while precisa colocá-lo entre colchetes, senão não.

O **Do** é semelhante ao while, ou seja, é executado sempre associado a uma condição, novamente a cada passagem pelo looping a condição é avaliada. Só que no **Do** a condição é colocada no final, fazendo com que mesmo que o resultado seja falso da expressão booleana o código seja executado pelo menos uma vez. Caso precise abandonar o looping, use o **break**, que é opcional. Veja a sintaxe (não esqueça do ponto-e-virgula no final):

```
do
    statement
while (booleanExpression);
```

Vamos fazer uma modificação no exercício do item anterior para compreender como funciona o **break**.

1 - Se o exemplo anterior não estiver aberto no Visual Studio, abra-o.

2 - Vá para o código do **button1**, ele deve ser o seguinte:

```
int contador = 0;

while (contador < 20)
{
    contador++;
}
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
listBox1.Items.Add(contador);  
}
```

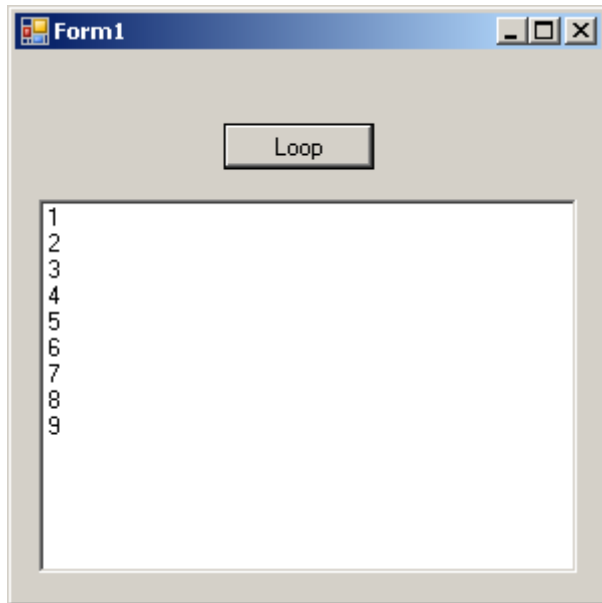
3 - Mude para o seguinte:

```
int contador = 0;  
while (contador < 20)  
{  
    contador++;  
    if (contador == 10)  
    {  
        break;  
    }  
    listBox1.Items.Add(contador);  
}
```

Perceba que as modificações foram mínimas, apenas inserimos um **If** que verifica se o conteúdo da variável **contador** é 10, se for ele executa o **break** que finaliza imediatamente o loop.

4 - Execute a aplicação:

5 - Clique em Loop.



Perceba que ele nem chega a escrever o número 10 por que a linha de código que é responsável por isso está depois do **If** que finalizou o loop.

6 – Agora mude o valor inicial da variável 0 para 20.

7 – Execute a aplicação e clique em Loop.

Perceba que não aparece nada, porque o resultado da expressão condicional começa como false, já que o valor não é menor que 20, e sim é igual a 20.

8 – Vamos usar o Do agora, mude o código para o seguinte:

```
int contador = 20;

do
{
    contador++;

    listBox1.Items.Add(contador);

    if (contador == 10)
```

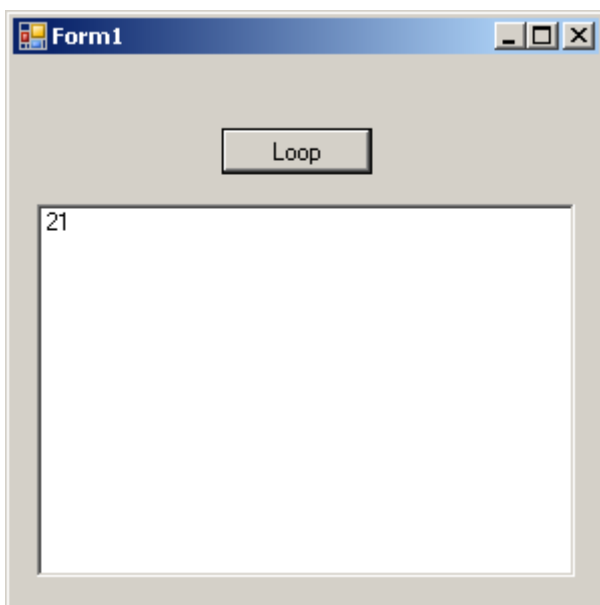
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

```
        {  
            break;  
        }  
    }  
while (contador < 20);
```

9 - Execute a aplicação e clique em Loop.



Se a expressão estivesse encima nada apareceria, mas agora aparece o 21, isso porque ele executou uma vez o loop antes de avaliar a expressão, algumas vezes esse tipo de loop pode ser necessário. Fique atento.

O **For** é usado quando sabemos o número de vezes que iremos executar o Loop. O For precisa de um contador que normalmente é uma variável que controla o número de vezes que o loop será executado..

A variável contador pode ser inicializada antes do For ou na própria declaração.

Autor: Herbert Moroni Cavallari da Costa Gois

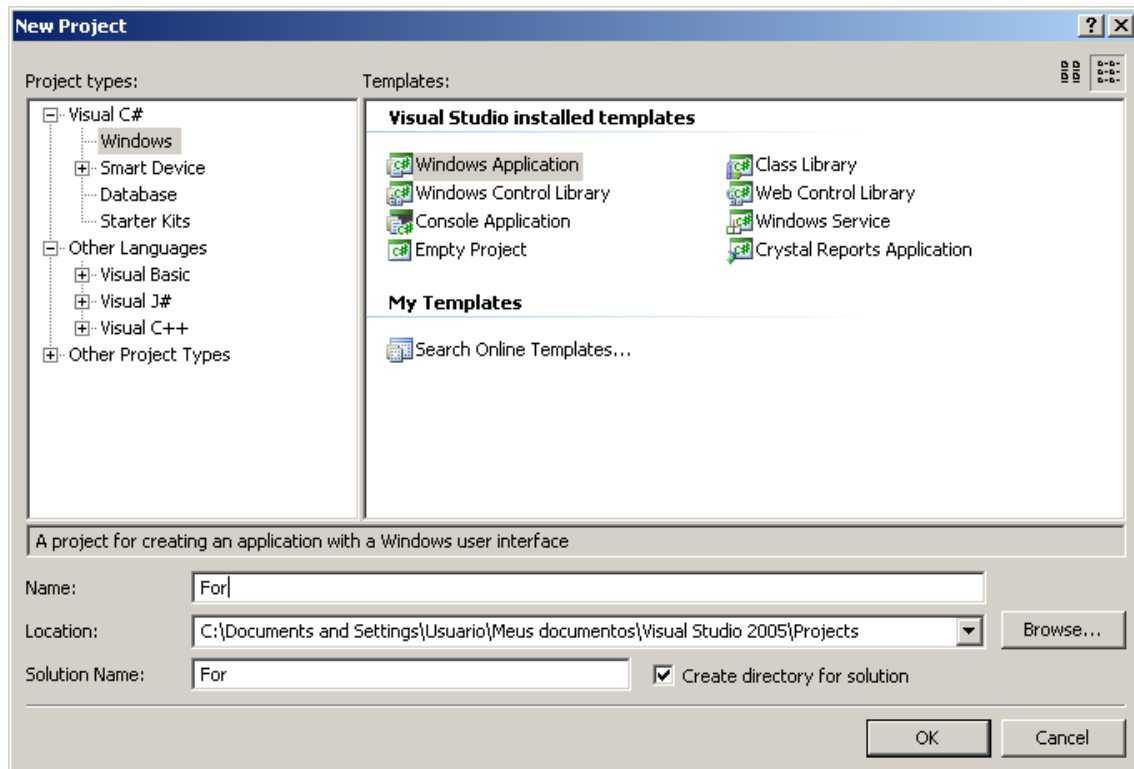
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Se durante o processamento você quiser abandonar o looping, terá que usar o **break**. A sintaxe do For é a seguinte:

```
for (initialization; booleanExpression; updateControlVariable)
    statement
```

1 - Crie uma nova aplicação do tipo **Windows Application** com nome **For**.



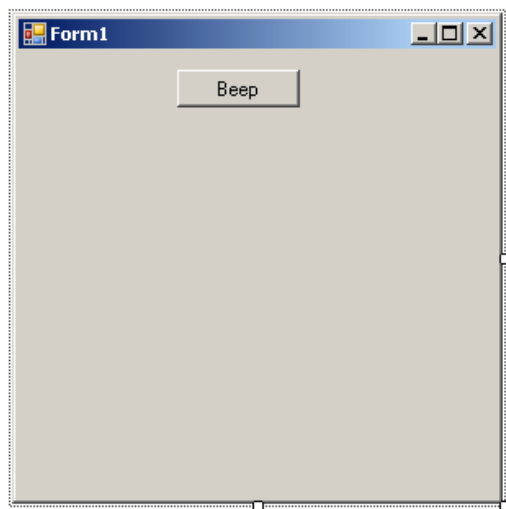
2 - Arraste para o **Form1** um **Button**.

3 - Mude a propriedade **Text** do **Button1** para "Beep".

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

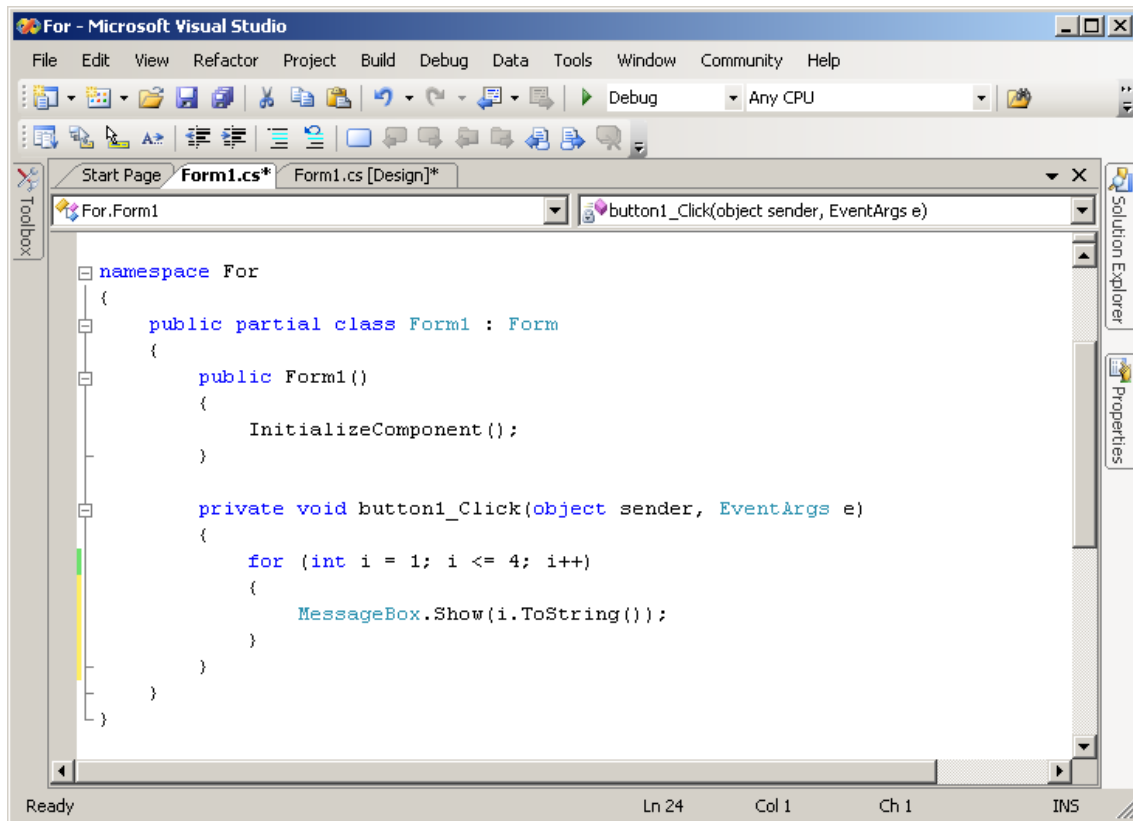
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



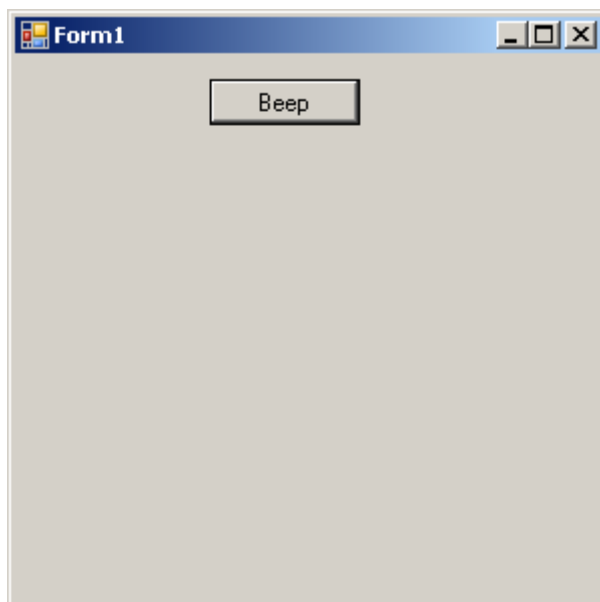
4 - De um clique duplo sobre o botão e no digite o seguinte código para o evento do botão:

```
for (int i = 1; i <= 4; i++)  
{  
    MessageBox.Show(i.ToString());  
}
```

Vai ficar assim:



5 - Execute a aplicação.

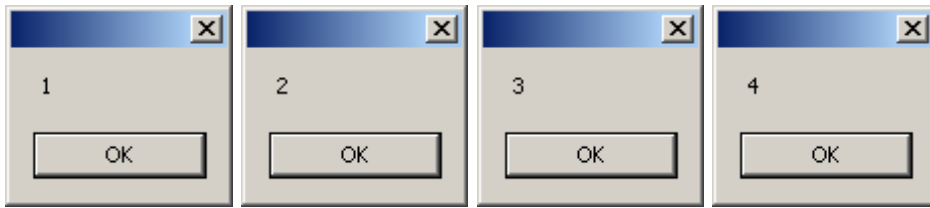


6 - Clique no botão Beep.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Vamos dar uma avaliada no código:

```
for (int i = 1; i <= 4; i++)  
{  
    MessageBox.Show(i.ToString());  
}
```

Quando inicializamos o for, primeiro iniciamos uma variável do tipo integer e damos o valor 1 para ela, damos o nome de i para essa variável. Depois colocamos uma expressão booleana que será responsável por avaliar o número de vezes que o for será executado, nossa expressão diz que o for deve ser executado enquanto o conteúdo da variável i for menor ou igual a 4. Para finalizar a inicialização do For dizemos que a cada passada o conteúdo da variável i deve ser acrescentado em 1.

7 - Mude o código do botão para o seguinte:

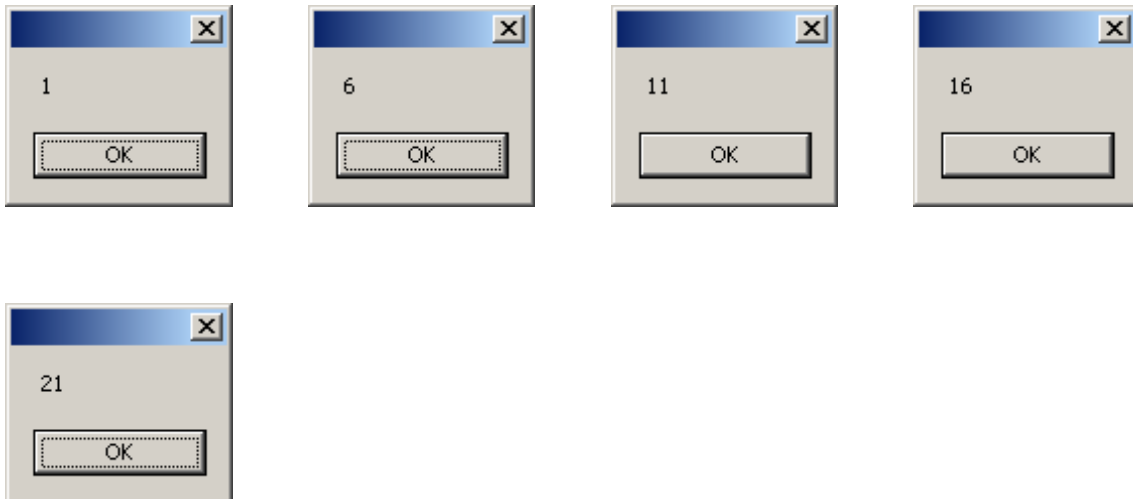
```
for (int i = 1; i <= 25; i+=5)  
{  
    MessageBox.Show(i.ToString());  
}
```

8 - Execute a aplicação e clique em Beep.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Perceba que o `i += 5` fez com que o número adicionado a variável a cada passada do loop é 5 ao invés de 1.

9.11 – Vetores e matrizes em C#

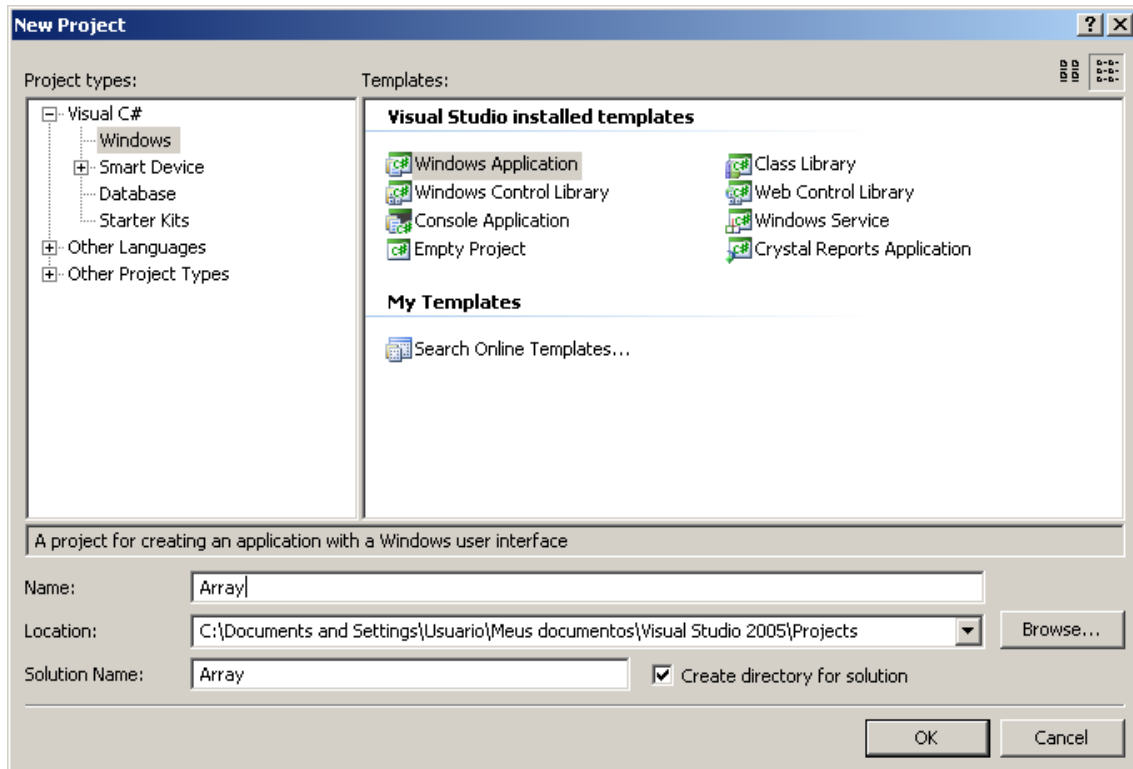
Vamos fazer um exemplo, nele vamos criar o programa que armazenará os dados nos nomes dos jogadores e as faltas cometidas por eles.

1 - Entre no Visual Studio e crie um novo projeto do tipo **Windows Application** chamado **Arrays**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



2 - Arraste para o Form1 os seguintes controles:

- 3 Button
- 1 ListBox
- 2 Labels
- 2 TextBoxes

3 - Altere a propriedade Text do Button1 para "Inserir Jogador".

4 - Altere a propriedade Text do Button2 para "Inserir Falta"

5 - Altere a propriedade Text do Button3 para "Visualizar Array"

6 - Altere a propriedade Text do Form1 para "Array".

Autor: Herbert Moroni Cavallari da Costa Gois

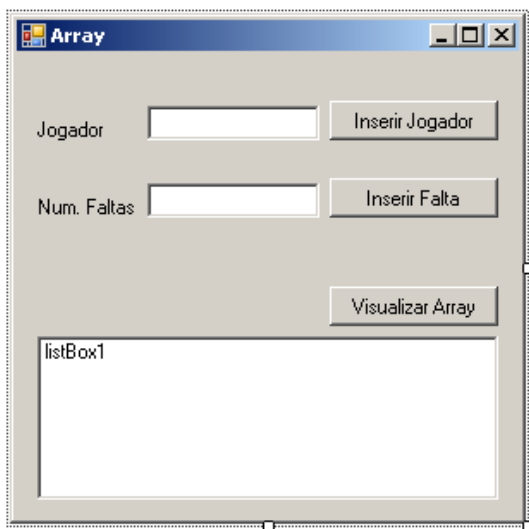
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

7 – Altere a propriedade Text do Label1 para “Jogador”.

8 – Altere a propriedade Text do Label2 para “Num. Faltas”

9 - Organize-os como a próxima figura:



10 - Na janela **Solution Explorer** clique no botão **View Code** para ir para o Painel de código.

11 - Digite dentro da classe do **Form1** o seguinte código:

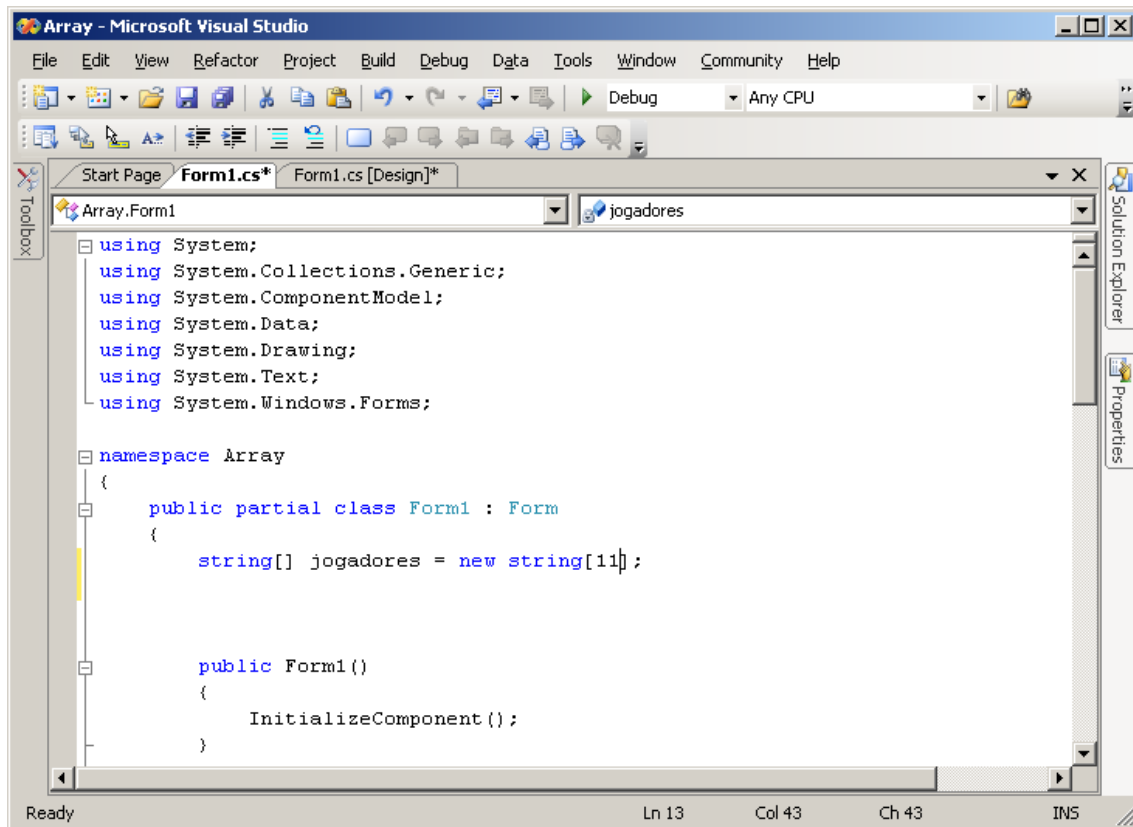
```
string[] jogadores = new string[11];
```

Vai ficar assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



Nesta linha declaramos um Array de tamanho fixo. O array que criamos é como uma tabela com 11 linhas, o suficiente para armazenar o nome do time inteiro, 11 jogadores.

12 - Na janela **Solution Explorer** clique em **View Designer** para voltar para o **Form1** no modo Design, você pode também clicar na aba **Form1.vb[Design]** ao lado da aba **Start Page**, à essa altura do curso você já deve estar familiarizado com o Visual Studio.

13 - De um clique duplo sobre o **Button1** e digite o seguinte código:

```
jogadores[0] = textBox1.Text;
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Isso insere "Marcos" na primeira linha do Array, se colocássemos 6 ao invés de 0, por exemplo, seria na linha 7. O primeiro registro no array é sempre o 0, isso porque os índices sempre começam com 0.

14 - De um clique duplo sobre o **Button3** (Visualizar Array) e digite o seguinte código:

```
listBox1.Items.Add(jogadores[0]);
```


Isso insere na **ListBox1** o conteúdo da primeira linha do array Jogadores, se colocássemos 6, exibiria o conteúdo da linha 7, lembre-se que de o primeiro registro do array tem sempre o índice 0 e assim sucessivamente.

Não se assuste com o código para inserir no **ListBox1**, se fosse em uma **TextBox** ou **Label** você poderia usar a seguinte linha de código:

```
textBox1.Text = jogadores[0];
```

Só que para inserir um item no **ListBox** precisamos usar seu método **Add**.

15 – Execute a aplicação.



The screenshot shows a Windows-style application window titled "Array". It contains two text input fields: "Jogador" and "Num. Faltas". To the right of the "Jogador" field is a button labeled "Inserir Jogador". To the right of the "Num. Faltas" field is a button labeled "Inserir Falta". Below these is a button labeled "Visualizar Array". At the bottom of the window is a large, empty rectangular list box.

16 – Digite Marcos no textBox1 e clique em inserir Jogador.

17 – Clique em Visualizar Array.



This screenshot shows the same "Array" application window after the user has entered data. The "Jogador" text box now contains the name "Marcos". The "Inserir Jogador" button remains visible. The "Num. Faltas" field is still empty. The "Visualizar Array" button is also visible. The large list box at the bottom now contains the name "Marcos", indicating that the data has been successfully added to the array.

Agora que você já compreende como inserir e ler dados do **array** vamos avançar no exemplo.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

18 – Pare a execução.

19 - Digite dentro da classe do **Form1** o seguinte código:

```
int contador = 0;
```

Isto cria uma variável que estará disponível para todos os procedimentos dentro da classe, ou seja, seu escopo é a classe. Também já atribuímos o valor 0 nela. Essa variável será usada para controlar o índice ou local do array onde vamos inserir cada jogador.

20 - Volte para o código do procedimento do **Button1**.

21 - Substitua o código pelo seguinte:

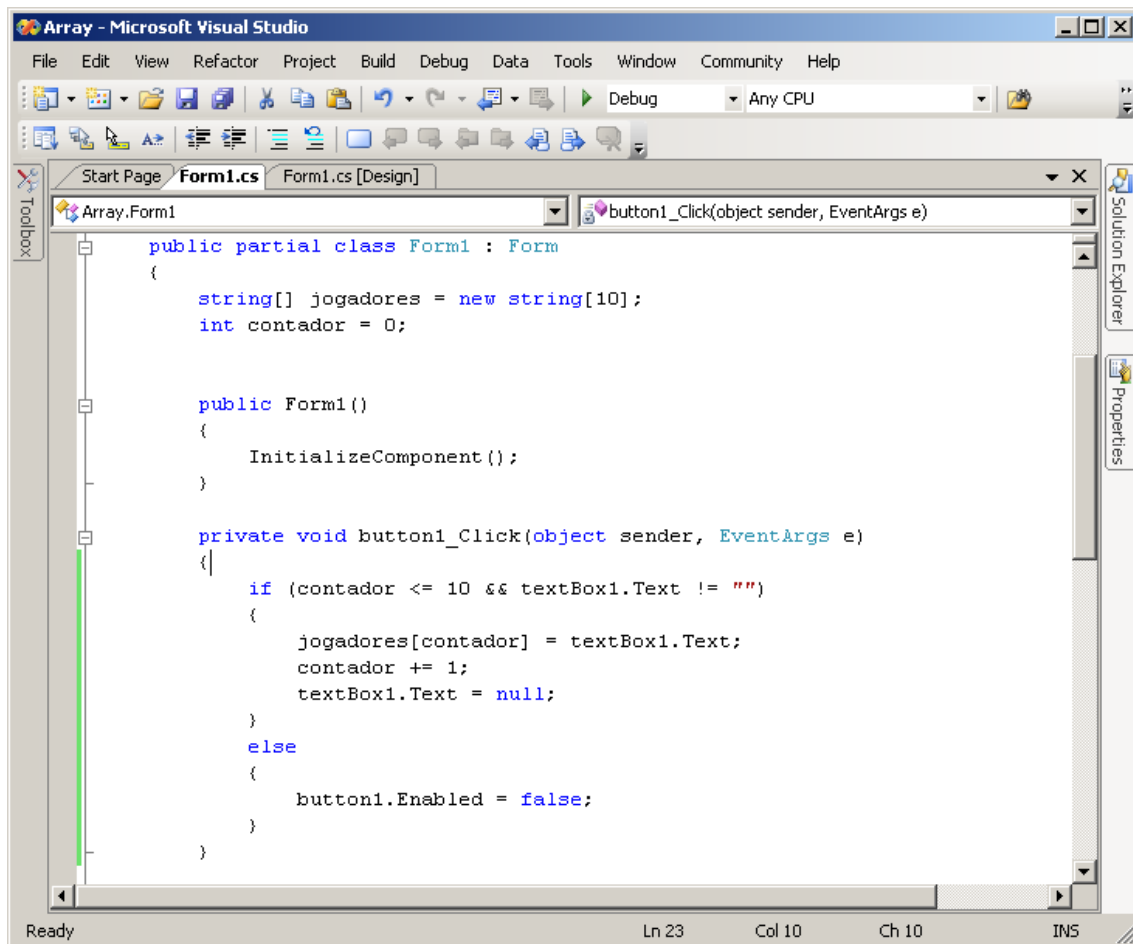
```
if (contador <= 10 && textBox1.Text != "")
{
    jogadores[contador] = textBox1.Text;
    contador += 1;
    textBox1.Text = null;
}
else
{
    button1.Enabled = false;
}
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Vai ficar assim:



Não podemos adicionar jogadores no array se ele estiver cheio e se o conteúdo do textBox não tiver nada. Para isso usamos o if.

Depois adicionamos o conteúdo do textBox1 no array. A posição é informada pela variável contador.

Adicionamos 1 a variável contador, senão o array nunca vai para a próxima posição.

Para finalizar limpamos o textBox1.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

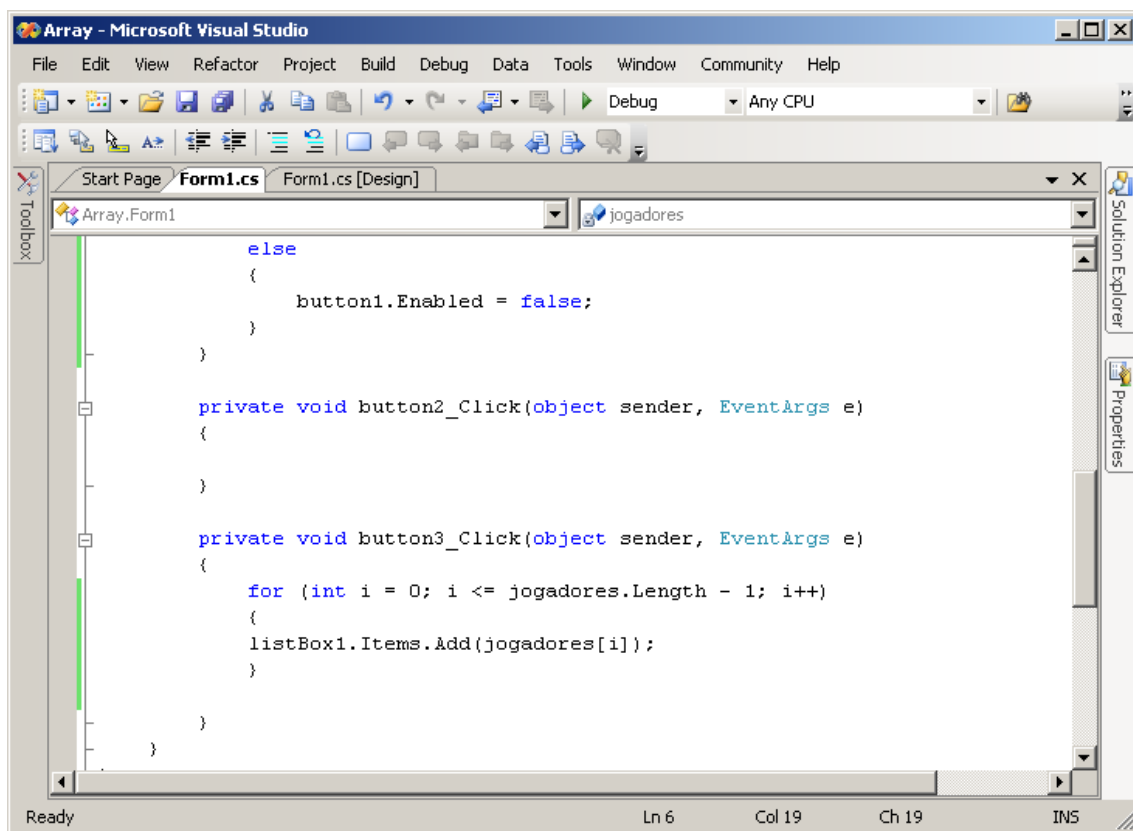
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

No else apenas desabilitamos o **Button1** caso o array esteja vazio ou não tenha nada nele digitado, ou seja, caso a expressão lógica do if retorne false.

22 - Mude o código do **Button3** pelo seguinte:

```
for (int i = 0; i <= jogadores.Length - 1; i++)
{
    listBox1.Items.Add(jogadores[i]);
}
```

Vai ficar assim:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Aqui usamos o loop `for` para passar por cada item do Array e adiciona-lo no **ListBox1**.

A propriedade **Length** retorna o numero de registros do array, que é 11, poderíamos simplesmente ter feito assim:

```
for (int i = 0; i <= 10; i++)
```

Colocamos o -1 depois do Length porque o Length retorna 11, lembre-se que o índice do primeiro elemento do array é 0, então o ultimo é 10.

A cada passada pelo For o `i` assume o valor de uma linha do array até chegar ao final.

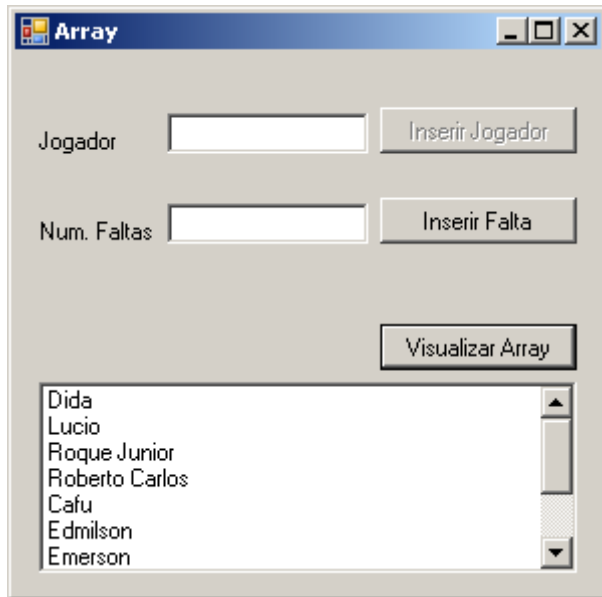
23 - Execute a aplicação.

24 – Digite o nome de um Jogador e clique no botão Inserir Jogador.

Faça isso sucessivamente até que o botão seja desabilitado, ou seja, que o array seja totalmente preenchido.

25 - Clique no botão Visualizar Array.

O nome dos 11 jogadores é listado.



Como você pode ver, trabalhamos facilmente com os nomes dos 11 jogadores em nosso array, podemos modificar os dados do nosso array facilmente, bastando para isso indicar qual linha deve ser alterada e o novo dado, como no exemplo:

```
jogadores[10] = "Robinho";
```

Um array pode conter mais de um coluna. Vamos entender como isso funciona implementando o código que vai armazenar o número de faltas de cada jogador.

Como um Array só pode conter um tipo de dado e nosso Array já é do tipo String, no nosso exemplo vamos adicionar o número de faltas como string mesmo, mas, a maneira correta seria criar um outro Array do tipo integer para armazenar as faltas. Vamos optar pela primeira opção para você aprender a trabalhar com duas colunas.

26 - Altere o código que declara o Array para o seguinte:

```
string[,] jogadores = new string[11,2];
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Isso cria um Array com 11 linhas e 2 colunas.

27 - Localize no procedimento do evento do Button1, as seguintes código:

```
jogadores[contador] = textBox1.Text;
```

28 - Mude-o para o seguinte:

```
jogadores[contador,0] = textBox1.Text;
```

Isso apenas indica que vamos adicionar o nome na linha que o loop indicar, na coluna 1, de índice 0.

29 - Digite dentro da classe do **Form1** o seguinte código:

```
int contadorFaltas = 0;
```

Esta variável fará o mesmo que a contador, só que para as faltas.

30 - Dê um clique duplo sobre o Button2 e digite o seguinte código:

```
if (contadorFaltas <= 10 && textBox2.Text != "")  
{  
    jogadores[contadorFaltas, 1] = textBox2.Text;  
    contadorFaltas += 1;  
    textBox2.Text = null;  
}
```

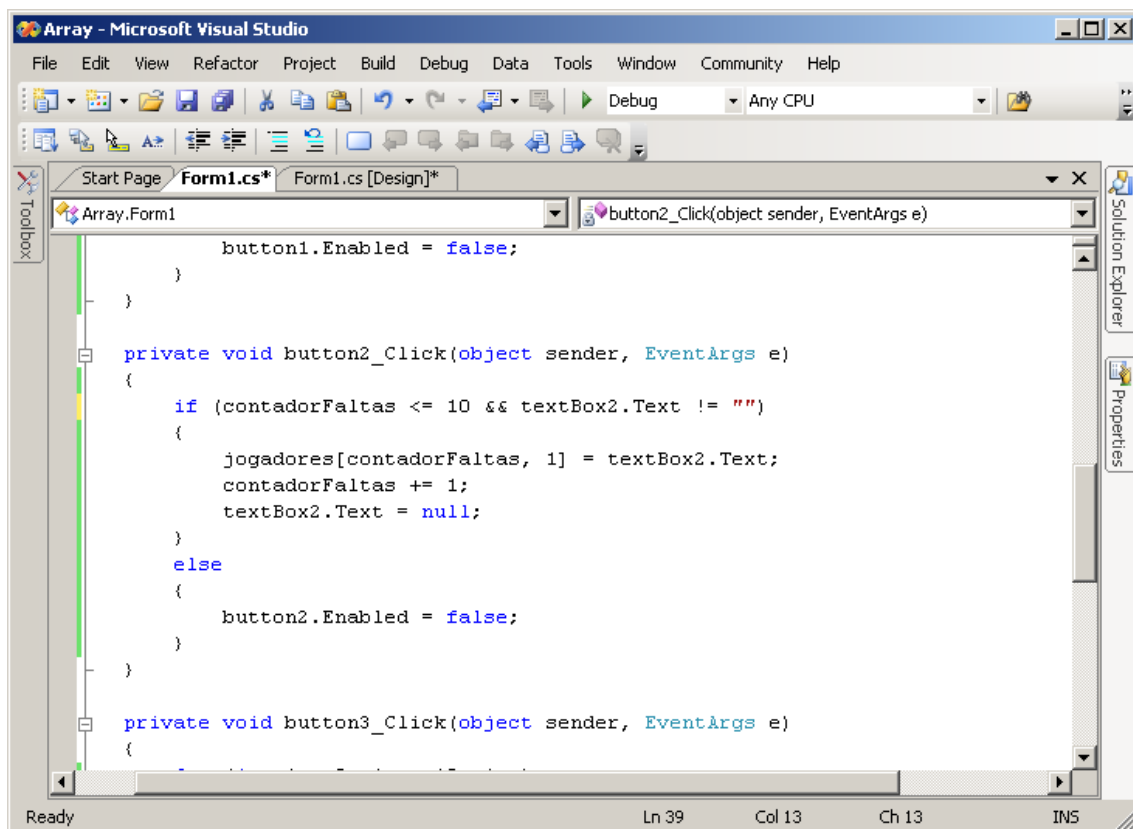
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
else
{
    button2.Enabled = false;
}
```

Vai ficar assim:



Isso é semelhante ao código do Button1. Só que usamos a variável contadorFaltas, o textBox2 e o button2.

Note também que agora atribuímos os valores à segunda coluna, como mostra o código:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
jogadores[contadorFaltas, 1] = textBox2.Text;
```

O índice 1 se refere a segunda coluna.

Agora precisamos mudar o código do button3, que exibe os dados, para listar as faltas também.

31 - Mude o código do Button3 para o seguinte:

```
for (int i = 0; i <= 10; i++)
{
    listBox1.Items.Add(jogadores[i,0] + " - Total de faltas: " +
    jogadores[i,1]);
}
```

O que fizemos foi concatenar os dados da coluna 1 com os da 2, e fazer um loop por cada linha do array de 0 a 10 ou seja pelas 11 linhas.

32 - Execute sua aplicação:

33 – Digite o nome de um Jogador e clique no botão Inserir Jogador.

Faça isso sucessivamente até que o botão seja desabilitado, ou seja, que o array seja totalmente preenchido.

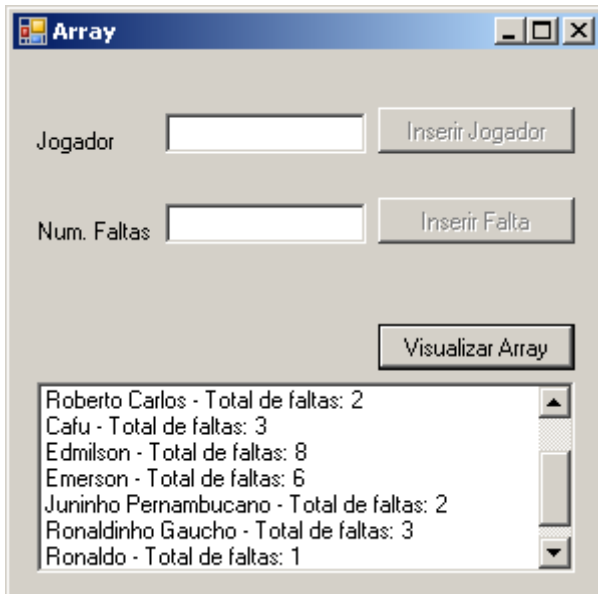
34 – Digite uma quantidade de faltas para cada jogador como fez com o nome dos jogadores até o botão ser desabilitado.

35 – Clique no botão Visualizar array

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Jogador	Total de faltas
Roberto Carlos	2
Cafu	3
Edmilson	8
Emerson	6
Juninho Pernambucano	2
Ronaldinho Gaucho	3
Ronaldo	1

9.12 – Procedimentos e funções em C#

Em C# tanto os procedimentos quanto as funções são conhecidos como **métodos**.

Em C# os métodos podem ou não retornar valores. Quando eles retornam valores trabalham de forma idêntica a uma função, quando não retornam valores são como procedimentos.

Método é uma sequência nomeada de instruções. Cada método tem um nome e um corpo. O corpo contém as instruções que vão ser executadas quando o método for chamado. O nome do método deve ajudar a identificar seu propósito, ex: *CalcularImpostoVenda*.

A maioria dos métodos recebe dados, processa-os e retorna a informação ou o resultado do processamento. Método é um mecanismo fundamental e poderoso.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Especificando a sintaxe de declaração de um método:

```
tipoDeRetorno nomeDoMetodo ( listaDeParametros opcional )  
  
{  
  
    // corpo do método  
  
}
```

- *tipoDeRetorno* – é o tipo de dado que vai ser retornado pelo método após sua execução. Pode ser o nome de qualquer tipo como *int* ou *string*. Se o seu método não for retornar valor algum, você precisa usar a palavra reservada *void* aqui, especificando que o método não retorna nada.
- *nomeDoMetodo* – é o nome que vai ser usado para chamar o método. Este nome deve seguir as mesmas recomendações usadas nos nomes das variáveis. Procure usar notação camelCasing para nomear os métodos e também procure utilizar um verbo na primeira palavra do nome, para ajudar a identificar o que o método faz.
- *ListaDeParametros* – descreve o tipo e os nomes das informações opcionais que o método recebe. Você escreve os parâmetros dentro dos parênteses como se fosse declarar variáveis: nome do tipo seguido pelo nome do parâmetro. Se o seu método tiver mais que um parâmetro, separe-os com vírgula.
- *Corpo do método* – linhas de código que vão ser executadas quando o método é chamado.

O C# não suporta os métodos globais, utilizados por programadores Visual Basic, C e C++.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Abaixo um exemplo de um método chamado *AdicionarValores* que retorna um número inteiro (*int*) e recebe dois números inteiros como parâmetros.

```
int adicionarValores (int numeroPequeno, int numeroGrande)
{
    //...
    // corpo do método deve ser feito aqui
    //...
}
```

Abaixo um segundo exemplo, de um método chamado *mostrarResultado* que não retorna nenhum valor, e recebe um simples parâmetro chamado pergunta do tipo *int*.

```
void mostrarResultado (int pergunta)
{
    // ...
}
```

Lembre-se de usar a palavra reservada *void* quando o método não for retornar nada.

Escrevendo declarações que retornam valores:

Se você quer que seu método retorne uma informação (em outras palavras que retorne um tipo e não um *void*), você precisa escrever um código de retorno dentro do método.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Para isso você usa a palavra reservada *return*, seguida da expressão que calcula o valor a ser retornado. Atenção, porque o resultado da expressão deve ser do mesmo tipo que o especificado como *tipoDeRetorno* do método, senão o programa não vai compilar.

Por exemplo:

```
int adicionarValores (int numeroPequeno, int numeroGrande)
{
    //...
    return numeroPequeno + numeroGrande;
}
```

Lembre-se do ponto-e-vírgula ao final de cada instrução.

O retorno do método deve estar no final do método porque ele causa a finalização do método. Qualquer código depois da linha que faz o retorno não vai ser executado.

Se você não quer que seu método retorne informação alguma (do tipo *void*), você pode uma variação da palavra reservada *return* para causar o encerramento imediato do método, para isso digite *return* seguido de ponto-e-vírgula.

Por exemplo:

```
void mostrarResultado (int pergunta)
{
    ...
    if (...)
        return;
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
    ...  
}
```

Se o seu método não precisar retornar nada você pode simplesmente omitir o *return*, porque o método vai ser finalizado automaticamente quando a última linha de código do corpo for executada.

Chamando métodos:

Métodos existem para serem chamados. Você chama um método pelo seu nome para ele executar uma tarefa. Se esse método precisa de informações para realizar sua tarefa (parâmetros), você precisa enviar essas informações pra ele. Se o método retornar uma informação, você precisa ter uma forma de receber essa informação, como uma variável, por exemplo.

Especificando a sintaxe para se chamar um método:

nomeDoMetodo (*listaDeArgumentos* opcional)

- *nomeDoMetodo* – precisa ser exatamente igual ao nome do método que você está chamando, lembre-se que o C# é case-sensitive ou seja, diferencia maiúsculas de minúsculas.
- *listaDeArgumentos* – informações adicionais que o método aceita, você precisa passar um valor para cada parâmetro e este valor precisa ser compatível o tipo correspondente ao parâmetro. Se o método que você está chamando tiver dois ou mais parâmetros, você precisa separar os valores com vírgula.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Importante: Você precisa escrever os parênteses ao final de cada método, mesmo que lê não tiver parâmetro algum.

Por exemplo, lembre-se do método *adicionarValores*:

```
int adicionarValores (int numeroPequeno, int numeroGrande)
{
    //...
    return numeroPequeno + numeroGrande;
}
```

Este método tem dois parâmetros entre os parênteses, para chamá-lo, faça assim:

```
adicionarValores(39,3)
```

Esta é a maneira correta de chamar o método, se você tentar das seguintes formas não vai conseguir:

```
adicionarValores                // Falta parênteses
adicionarValores()              // Falta argumentos
adicionarValores(39)            // Falta um argumento
adicionarValores("39", "3")    // Tipos errados
```

O método *adicionarValores* retorna um numero inteiro. Este valor inteiro pode ser usado em qualquer lugar que uma variavel int puder ser usada.

Por exemplo:

```
resultado = adicionarValores(39,3);
mostrarResultado(adicionarValores(39,3));
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

No primeiro exemplo atribuímos o retorno a uma variável chamada *resultado*.

No segundo atribuímos o resultado a outro método, que vai utilizar este resultado como parâmetro para sua execução.

Você viu que pode criar uma variável dentro de um método. A variável é criada na código que a define, e outros código no mesmo método que vêm depois podem usar a variável. Em outras palavras, uma variável pode ser usada só em certos lugares depois de que ter sido criada. Uma vez que o método terminou, a variável desaparece completamente.

Se uma variável pode ser usada em um local particular em um programa, ela parece estar no escopo daquele local. O escopo de uma variável é simplesmente a região do programa em que ela é utilizável. O escopo se aplica a métodos como também a variáveis. O escopo de uma variável é ligado ao local da sua declaração que introduz a mesma no programa, como você aprenderá agora.

Criando escopo local com um método.

As chaves determinam onde começa e onde termina o corpo do método. Elas também determinam o escopo do método. Qualquer variável criada dentro do corpo do método faz parte do escopo do método. Estas variáveis são chamadas de **variáveis locais** porque são locais ao método onde são declaradas. Elas não podem ser usadas no escopo nenhum outro método, por isso você não pode usar variáveis locais para armazenar informações entre métodos. Quando um método acaba sua execução ele finaliza as variáveis que ele criou.

Por exemplo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
class Exemplo
{
    void método()
    {
        int variavel;
        ...
    }
    void outroMetodo()
    {
        variavel = 42; // isso gera um erro de compilação
    }
}
```

O erro mostrado acima é porque a variável foi criada dentro de um método diferente da qual esta sendo usada.

Criando escopo de classe com uma classe:

As chaves determinam onde começa e onde termina o corpo da classe e determinam seu escopo. Assim sendo, qualquer variável que você criar dentro do corpo da classe (mas que não estejam dentro do método), fazem parte do seu escopo. Em C# o nome correto desse tipo de variável é **campo**. Em contraste as variáveis locais você pode usar os campos para armazenar informações entre os métodos.

Por exemplo:

```
class Exemplo
{
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
int campo;

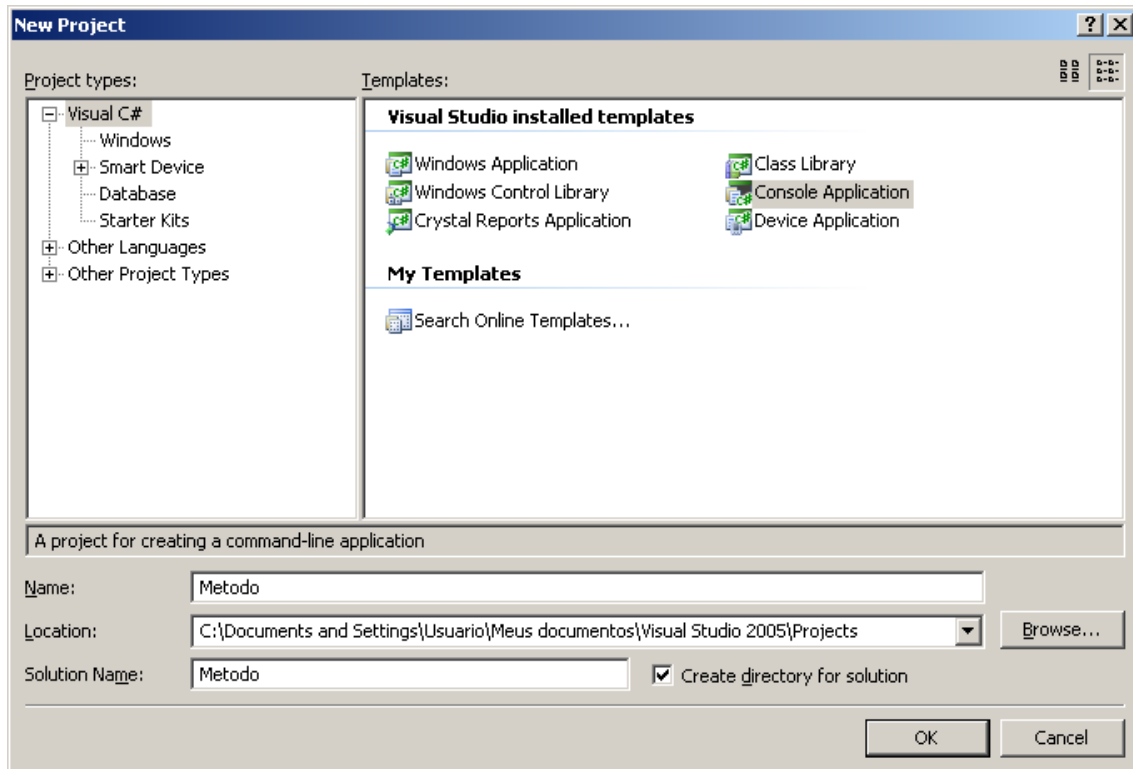
    void método()
    {
        campo = 42;
        ...
    }
    void outroMetodo()
    {
        campo = 44;
    }
}
```

Perceba que criamos a variável dentro da classe.

Em um método você precisa declarar uma variável antes de usá-la. Campos são diferentes, um método pode usar um campo antes de ser definido algum valor para ele.

Exemplo:

1 – Crie uma aplicação no Visual Studio .NET do tipo **Console Application** chamada **Método**.



2 – No painel de código crie digite o seguinte código que cria o método **lerDouble** dentro da classe **Program**.

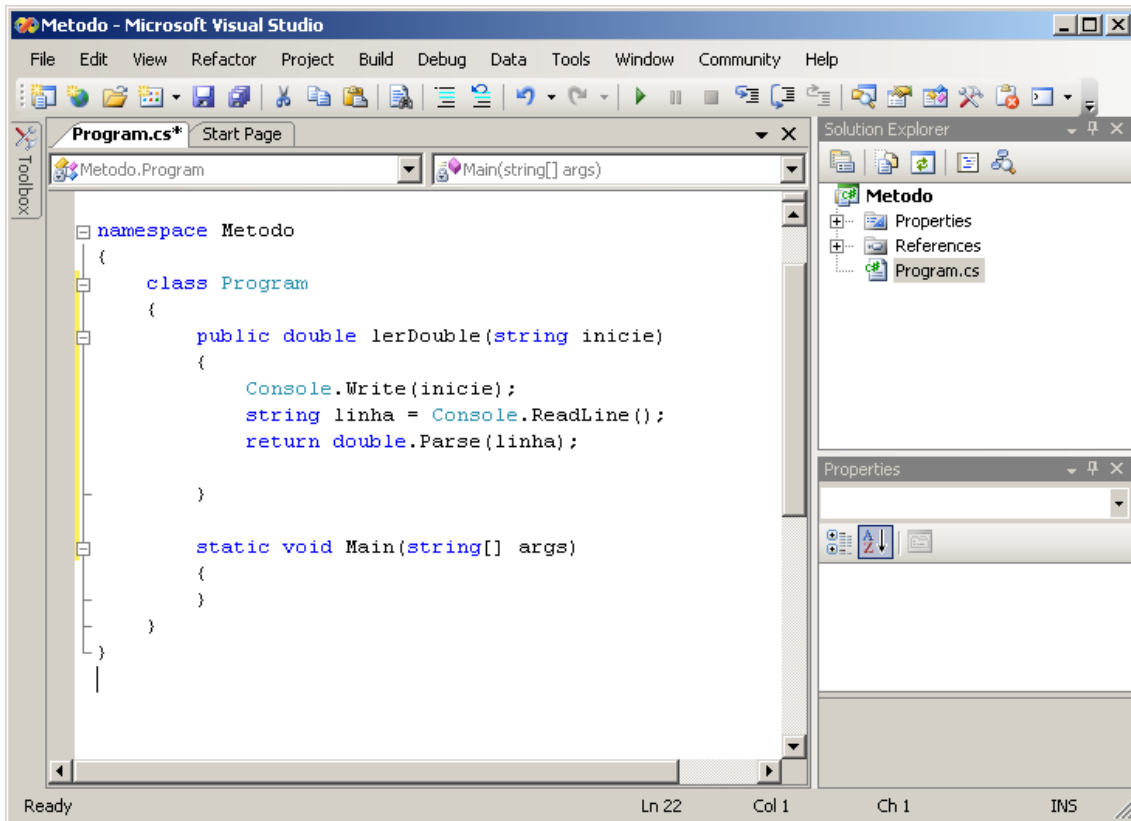
```
public double lerDouble(string inicio)
{
    Console.WriteLine(inicio);
    string linha = Console.ReadLine();
    return double.Parse(linha);
}
```

Vai ficar assim nosso método:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Esse método escreve no console o texto que é enviado para ele como parâmetro. Depois armazena na variável **linha** o numero digitado pelo usuário e o retorna.

3 – No menu **Build** clique em **Build Metodo** ou pressione **Ctrl+Shift+B**.

Verifique se compila sem nenhum erro.

4 - Vamos criar agora um outro método com as seguintes características:

Digite o seguinte código, você pode digitá-lo em qualquer local desde que esteja dentro da Classe no nosso caso a **Program**.

```
public int lerInteiro(string inicio)
{
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        Console.Write(inicio);

        string linha = Console.ReadLine();

        return int.Parse(linha);
    }
```

Esse método faz a mesma coisa que o método **lerDouble**, só que retorna uma **integer** (inteiro).

Esta ficando assim nosso código:

```
using System;

using System.Collections.Generic;

using System.Text;

namespace Metodo
{
    class Program
    {
        public double lerDouble(string inicio)
        {
            Console.Write(inicie);

            string linha = Console.ReadLine();

            return double.Parse(linha);
        }

        public int lerInteiro(string inicio)
        {
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        Console.WriteLine(inicio);

        string linha = Console.ReadLine();

        return int.Parse(linha);
    }

    static void Main(string[] args)
    {
    }
}
```

Verifique se os dois métodos estão dentro do escopo da classe Program.

5 – Novamente menu **Build** clique em **Build Metodo** ou pressione **Ctrl+Shift+B**. E verifique se ocorre erros, se ocorrer verifique seu código comparando-o ao acima. Lembre-se novamente que o C# é case-sensitive, ou seja, diferencia maiúsculas de minúsculas.

6 – Vamos criar mais um método, este método possui mais de um parâmetro: Segue as características do nosso método:

```
public double calcular(double taxaDiaria, int dias)
{
    return taxaDiaria * dias;
}
```

Esse método retorna a multiplicação dos dois parâmetros que são enviados para ele.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

7 – Nosso ultimo método tem as seguintes especificações:

```
public void escrever(double taxa)
{
    Console.WriteLine("A taxa é: {0}", taxa * 1.1);
}
```

Lembre que os métodos do tipo **void** não retornam nada.

Esse método escreve no console a multiplicação do parâmetro que é enviado para ele com 1.1

Esta ficando assim nosso código:

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

namespace Metodo
{
    class Program
    {
        public double lerDouble(string inicio)
        {
            Console.Write(inicio);

            string linha = Console.ReadLine();

            return double.Parse(linha);
        }
    }
}
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
    }

    public int lerInteiro(string inicio)
    {
        Console.Write(inicio);

        string linha = Console.ReadLine();
        return int.Parse(linha);
    }

    public double calcular(double taxaDiaria, int dias)
    {
        return taxaDiaria * dias;
    }

    public void escrever(double taxa)
    {
        Console.WriteLine("A taxa é: {0}", taxa * 1.1);
    }

    static void Main(string[] args)
    {
    }
}
}
```

8 – Clique em **Build Metodo** no menu **Build** para verificar se compila corretamente. Tudo certo? Vamos usar os métodos que criamos.

Chamando Métodos

Podemos chamar métodos dentro de métodos, como faremos a seguir.

9 – Crie um novo método que não retorna nada, ou seja, do tipo **void**. De o nome dele de **Executar**

O código do nosso novo método deve ficar assim:

```
public void executar()  
{  
  
    double taxadiaria = lerDouble("Digite a taxa diaria: ");  
    int dias = lerInteiro("Digite o numero de dias: ");  
    escrever(calcular(taxadiaria,dias));  
}
```

Na primeira linha de código criamos uma variável do tipo **double** e atribuímos a ela o método **lerDouble**.

Na segunda linha criamos uma variável do tipo **int** e atribuímos a ela o método **lerInteiro**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Na terceira linha de código chamamos o método **escrever** e passamos para ele como parâmetro o método **calcular**.

10 – No método **Main** (primeiro método executado pelo programa) digite o seguinte código:

```
(new Class1()).executar();
```

Isso vai executar o método **executar** assim que o programa for iniciado

O método **Main** vai ficar assim:

```
static void Main(string[] args)
{
    (new Program()).executar();
}
```

11 – Pressione **Ctrl+F5** para executar o programa.

O Visual Studio compila o programa e o executa. Uma janela console aparece.

12 – Para taxa diária digite 315 e pressione **Enter**.

13 – Para numero de dias digite 15 e pressione **Enter**.

O programa vai escrever a seguinte mensagem no console:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

A taxa é: 5197,5

14 – Pressione qualquer tecla para retornar ao Visual Studio.

Vou digitar o código fonte todo utilizado para você poder verificar caso haja algum problema e vamos depurar nosso programa para entender melhor como ele funciona.

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

namespace Metodo
{
    class Program
    {
        public double lerDouble(string inicio)
        {
            Console.Write(inicio);

            string linha = Console.ReadLine();

            return double.Parse(linha);
        }

        public int lerInteiro(string inicio)
        {
            Console.Write(inicio);

            string linha = Console.ReadLine();
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
        return int.Parse(linha);
    }

    public double calcular(double taxaDiaria, int dias)
    {
        return taxaDiaria * dias;
    }

    public void escrever(double taxa)
    {
        Console.WriteLine("A taxa é: {0}", taxa * 1.1);
    }

    public void executar()
    {
        double taxadiaria = lerDouble("Digite a taxa diaria: ");
        int dias = lerInteiro("Digite o numero de dias: ");
        escrever(calcular(taxadiaria, dias));
    }

    static void Main(string[] args)
    {
        (new Program()).executar();
    }
}
```

CAPITULO 10

LÓGICA DE PROGRAMAÇÃO COM VB.NET

O VB.NET, junto com o Visual Studio .NET 2005 compõe uma ferramenta extremamente robusta e fácil de utilizar, com perfeito suporte a todas as novas ondas que rondam o mundo da informática e tecnologia.

Tanto a linguagem VB.NET e a C# trabalham de forma praticamente idêntica na plataforma .NET da Microsoft, com diferença de sintaxe, como você vai ver se acompanhar o conteúdo das duas linguagens.

O Visual Studio .NET 2005 é a melhor ferramenta de desenvolvimento de aplicações para a plataforma .NET. Com uma interface amigável e integrada com os ambientes e de fácil entendimento, proporciona aos desenvolvedores a criação de aplicações sofisticadas com todos os recursos existentes, sem ter que ficar criando parte de código em um aplicativo e o restante no outro. É possível com o Visual Studio gerenciar recursos da máquina e local e de um possível servidor, criar aplicações para Windows, web e dispositivos móveis.

Meu objetivo neste capítulo é ajuda-lo a aplicar os conhecimentos de lógica de programação adquiridos até aqui na linguagem de programação VB.NET.

Vamos primeiramente conhecer um pouco do Visual Studio.NET, a ferramenta que utilizaremos para desenvolver nossos aplicativos e criar nosso primeiro exemplo.

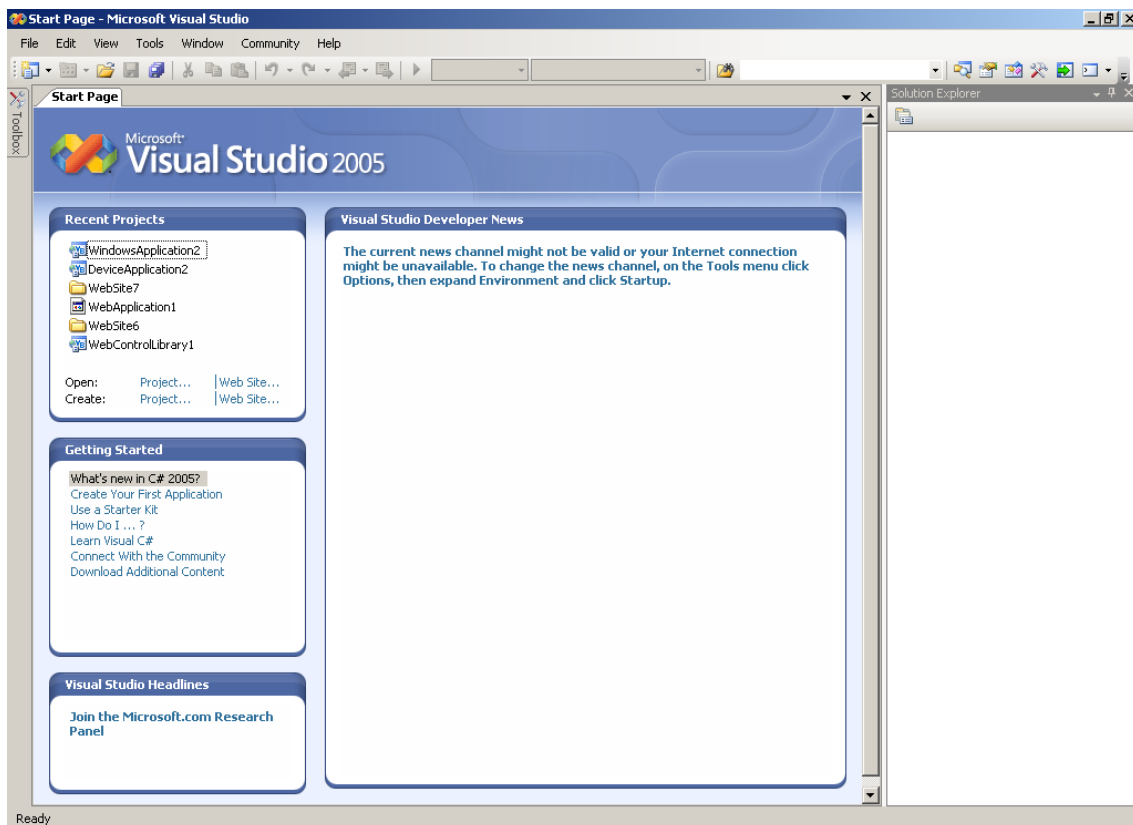
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

1 – Entre no Visual Studio.NET, eu estou usando a versão 2005, mas os exercícios funcionam em qualquer versão.

Você pode entrar no Visual Studio.NET através do menu **Iniciar / Programas / Microsoft Visual Studio .NET / Microsoft Visual Studio .NET** , sinta-se a vontade para criar qualquer atalho para ele.



A imagem anterior mostra o Visual Studio .NET assim que o iniciamos, é exibida a pagina **Start Page** onde podemos abrir rapidamente os ultimos projetos criados através da caixa **Recent Projects**.

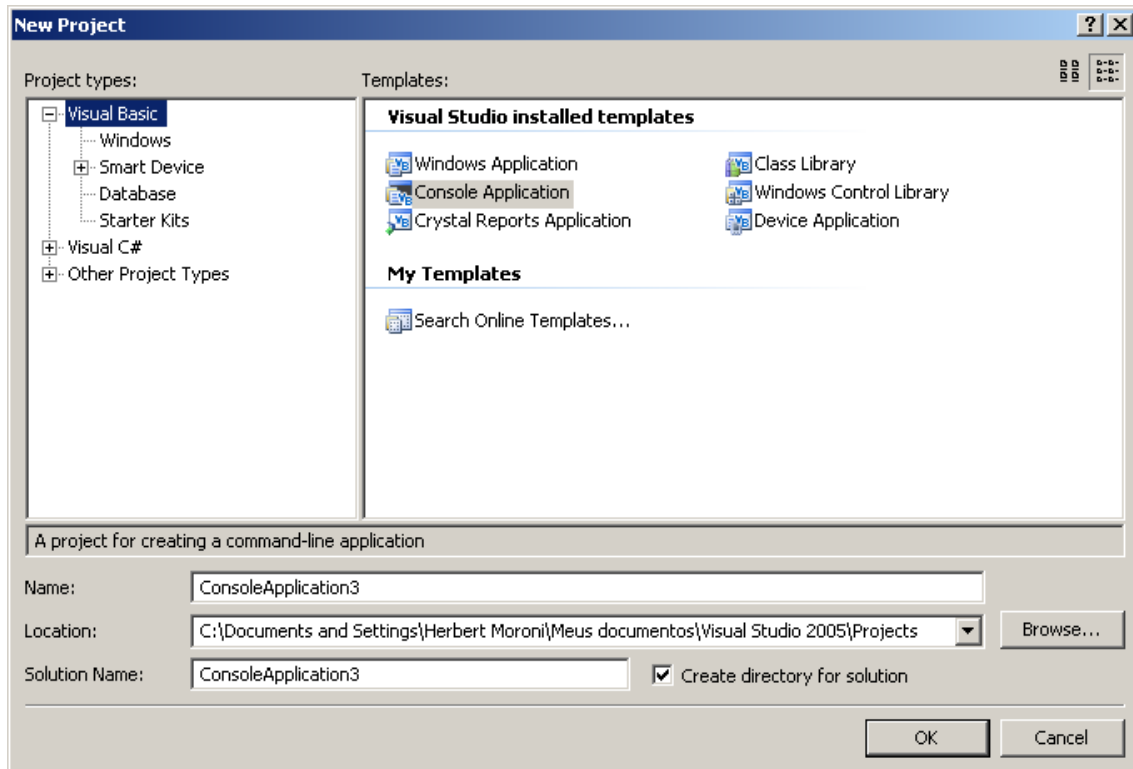
2 – No menu **File**, aponte em **New**, e clique em **Project**. (Ou clique Ctrl+Shift+N).

A caixa de dialogo **New Project** aparece. Ela permite que criemos um novo projeto usando vários **templates**, como um Windows Application, Class Library, Console Application e vários outros.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



3 – No painel **Project Type**, clique em **Visual C# Projects**, aqui estão todos os templates disponíveis para a linguagem C#.

4 – No painel **Templates** clique em **Console Application**.

5 – No campo nome digite, **ClassicoHelloWorld**.

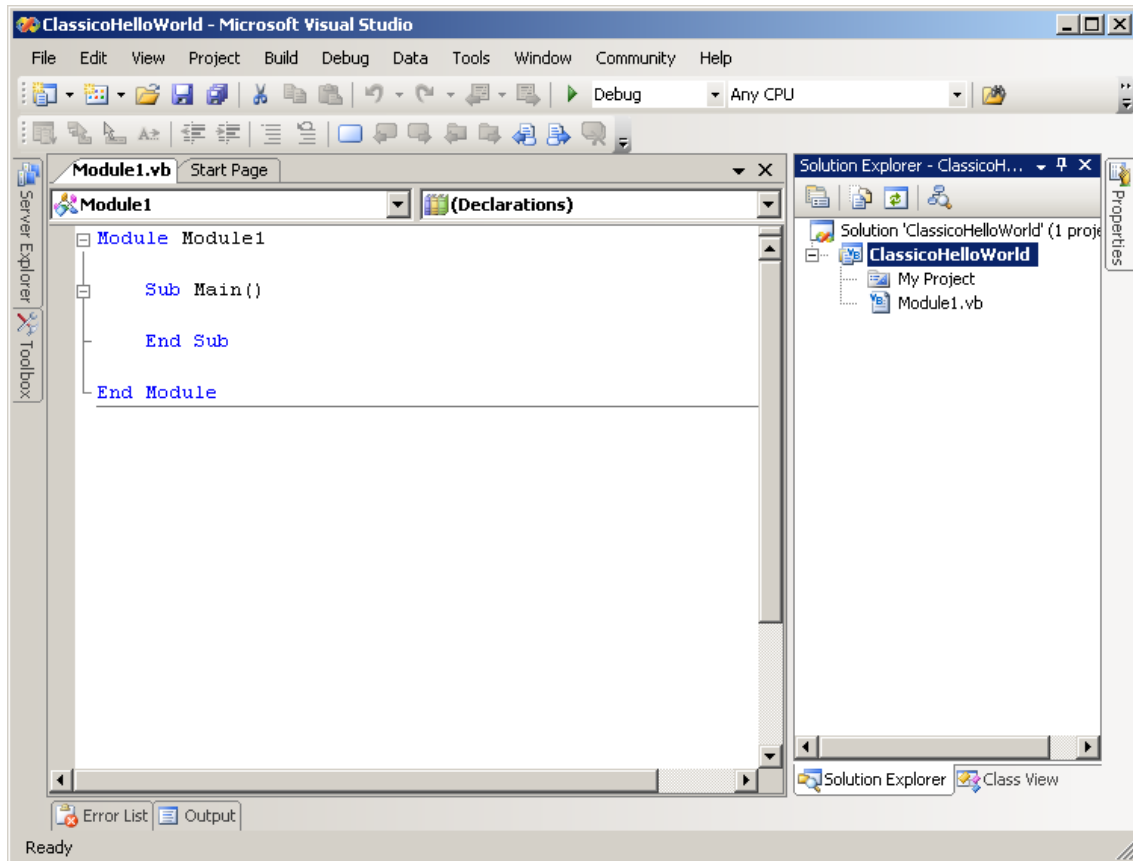
Perceba que você pode ainda alterar o caminho onde sua aplicação será salva e o nome da sua Solução.

6 – Clique em OK.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



A **barra de menus (menu bar)** possibilita o acesso aos comandos que você usa no ambiente de programação. Você pode usar o teclado ou o mouse para acessar o menu ou atalhos exatamente como usa em outros programas baseados em Windows.

A **barra de ferramentas (toolbar)** é localizada embaixo da barra de menus e disponibiliza botões que executam os comandos usados com mais frequência. Não confunda toolbar com toolbox.

A janela **Solution Explorer** mostra os nomes dos arquivos associados com o seu projeto. Você pode dar um clique duplo sobre o nome do arquivo para exibi-lo no **painel de código (Code pane)**. Vamos examinar os arquivos que o Visual Studio criou como parte do seu projeto:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

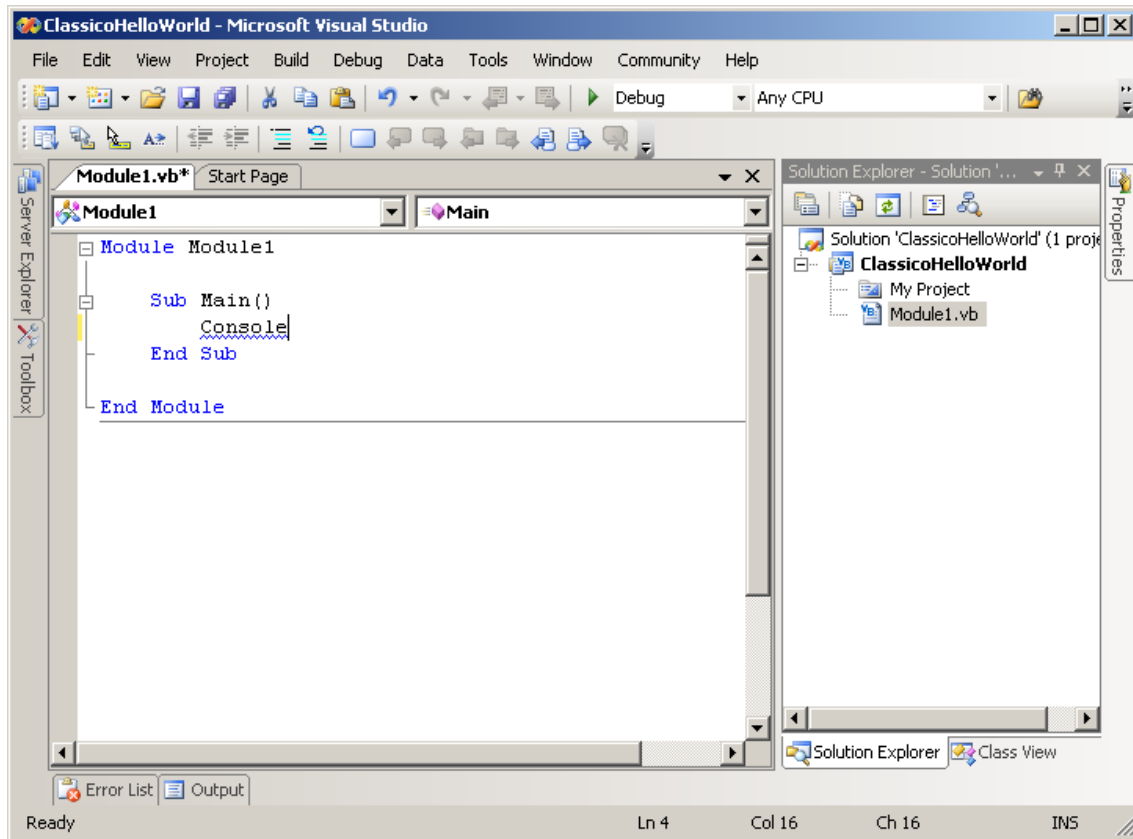
Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- **ClassicoHelloWorld.sln** organiza os projetos de cada solução, uma solução pode ter vários projetos, seu nome é representado no primeiro item na janela Solution Explorer só que sem a extensão do arquivo.
- **ClassicoHelloWorld.vbproj** este é o arquivo do projeto VB.NET. Pode ser associado a vários arquivos de código. É reconhecido no Solution Explorer pelo nome do projeto apenas, no entanto é gravado no disco com a extensão .vbproj.
- **Module1.vb** é um arquivo de código do VB.NET. Você vai escrever seu código neste arquivo. O Visual Studio já adicionou algum código nele automaticamente, examinaremos esse código mais adiante.

Aos poucos nós vamos explorando mais o Visual Studio, vamos ao nosso primeiro exemplo.

Nosso primeiro exemplo é bem simples, e um clássico para quem esta aprendendo qualquer linguagem, ele escreve **Hello World** no console.

7 – Dentro da Sub **Main**, digite: **Console**



A classe **Console** contém os métodos para exibir mensagens na tela e pegar as entradas do teclado. Tudo que o usuário digita no teclado pode ser lido através da classe **Console**. A classe **Console** só é significativa para aplicações que rodam no prompt de comando como neste nosso primeiro exemplo.

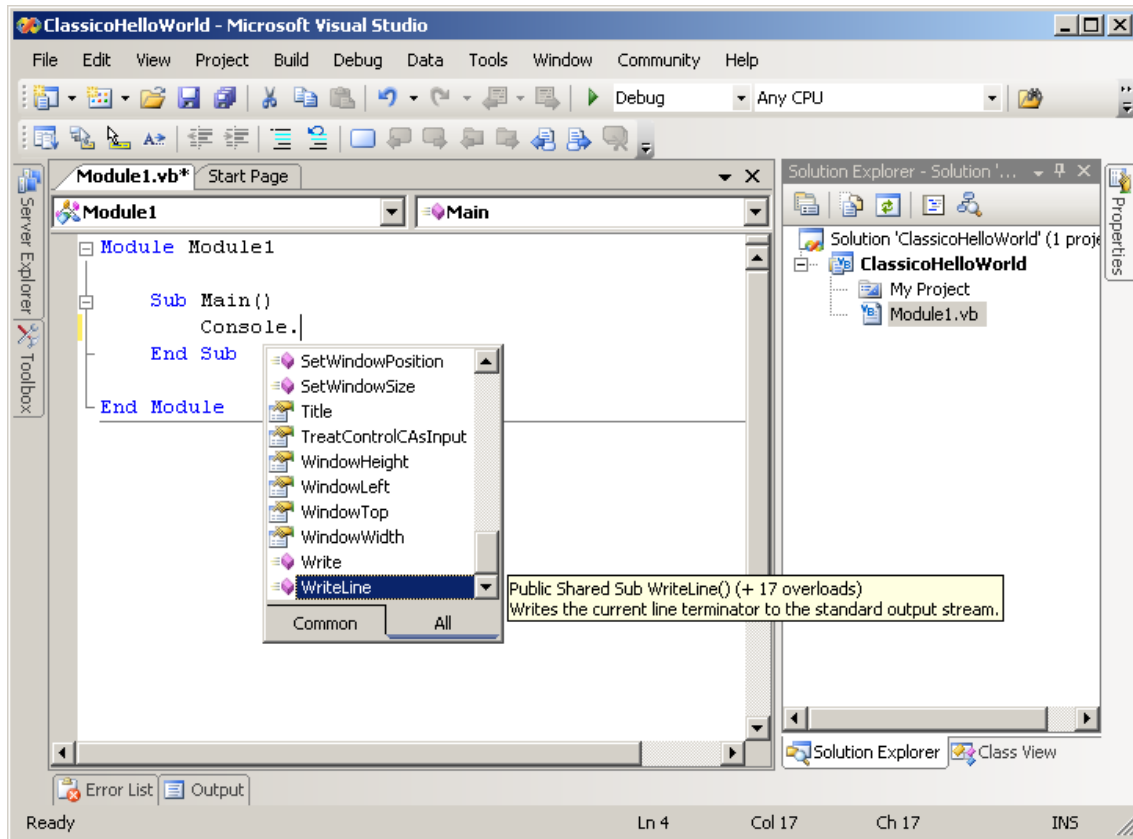
8 – Agora digite um ponto depois de **Console**.

Uma lista aparece, ela é chamada de **IntelliSense**, esse não é um recurso exclusivo do Visual Studio mas ajuda muito na programação. O IntelliSense exibe todos os métodos, propriedades e campos da classe.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



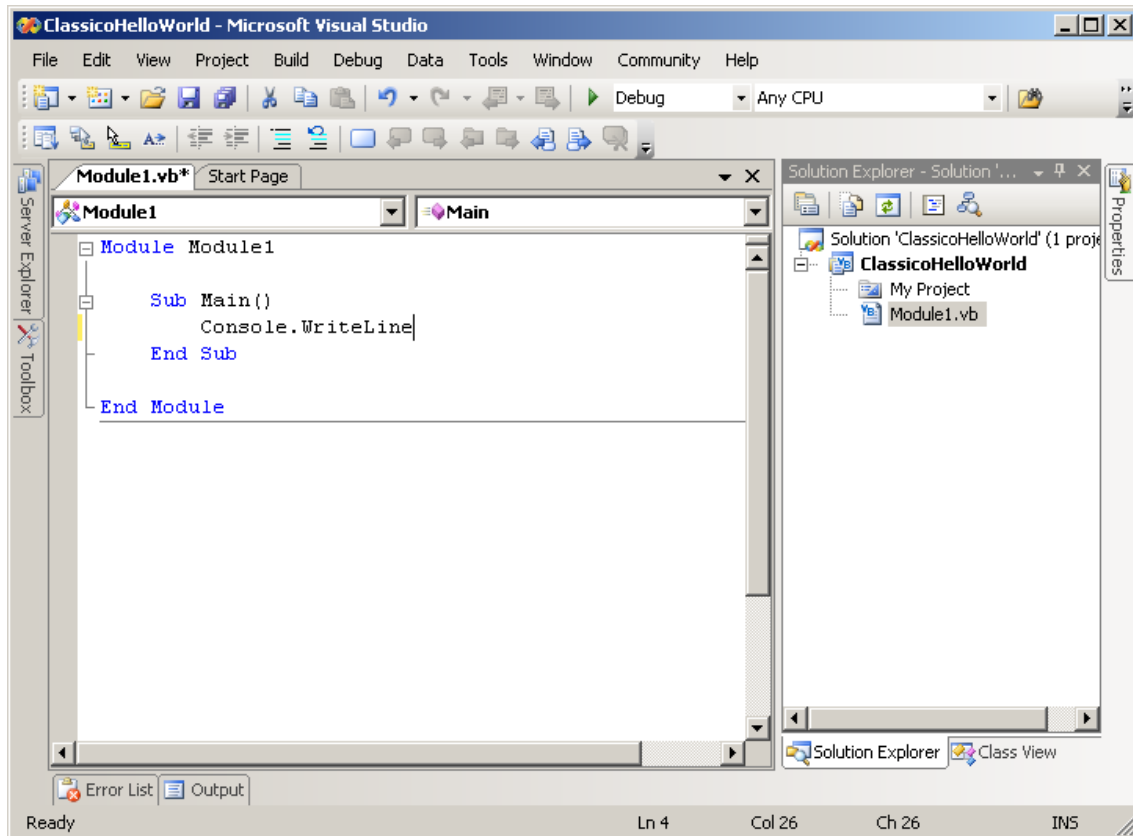
9 – Selecione **WriteLine**, você pode usar o Mouse ou o Teclado, tecele Enter ou dê um clique duplo sobre o WriteLine.

O IntelliSense é fechado e o método WriteLine é adicionado ao código. Como a seguinte imagem:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Quando o IntelliSense aparece você também pode pressionar W para ir direto para o primeiro membro do método Console que começa com w.

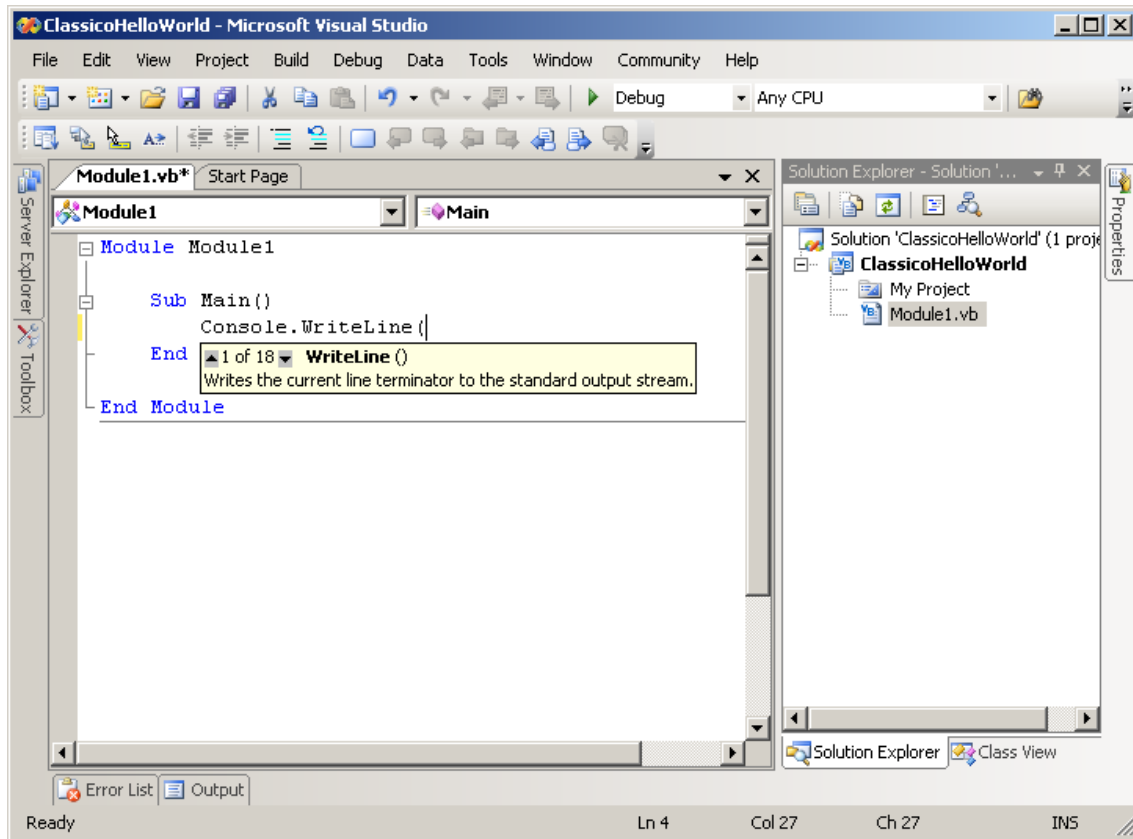
10 – Abra parênteses (

É mostrado uma outra forma do IntelliSense, esse mostra os parâmetros do método WriteLine. O método WriteLine tem o que chamamos de **Sobrecarga (Overload)**. Para cada sobrecarga do método WriteLine usamos parâmetros diferentes. Cada sobrecarga e seus respectivos parâmetros podem ser visualizados clicando com o mouse na seta do IntelliSense ou navegando pelas setas do teclado.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

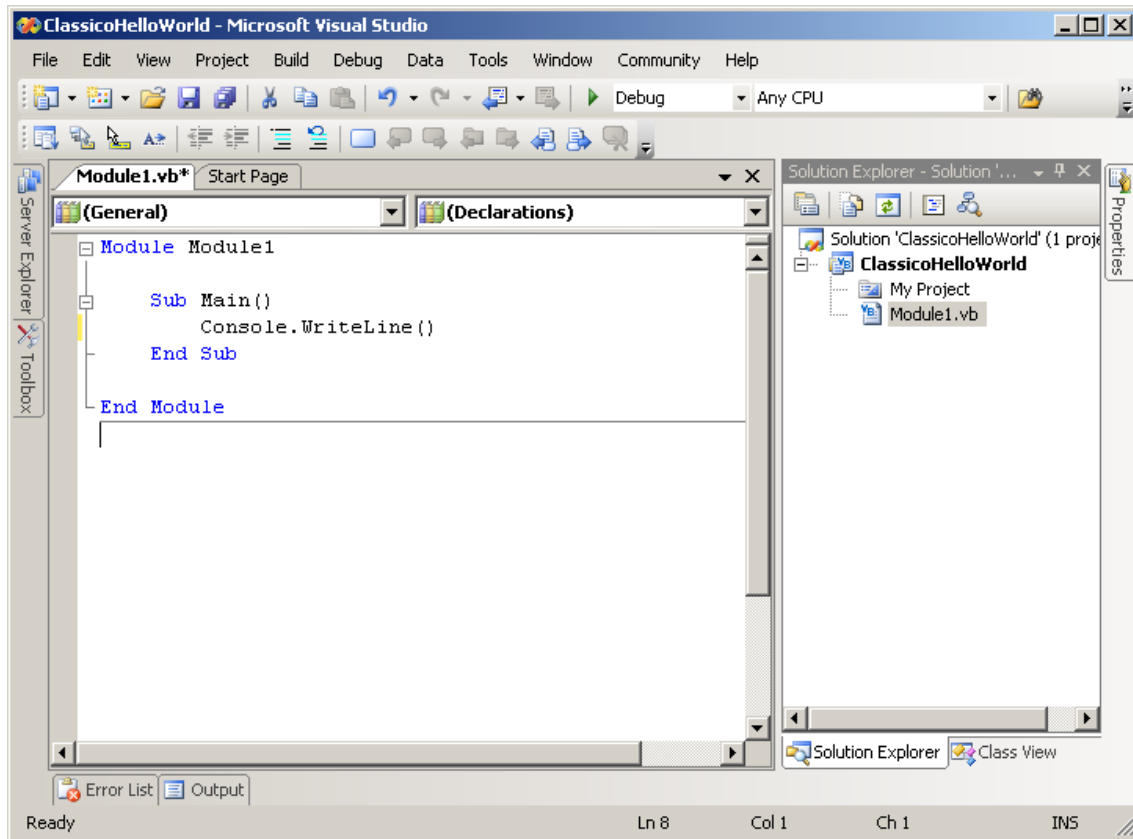


11 – Feche os parênteses, vai ficar assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

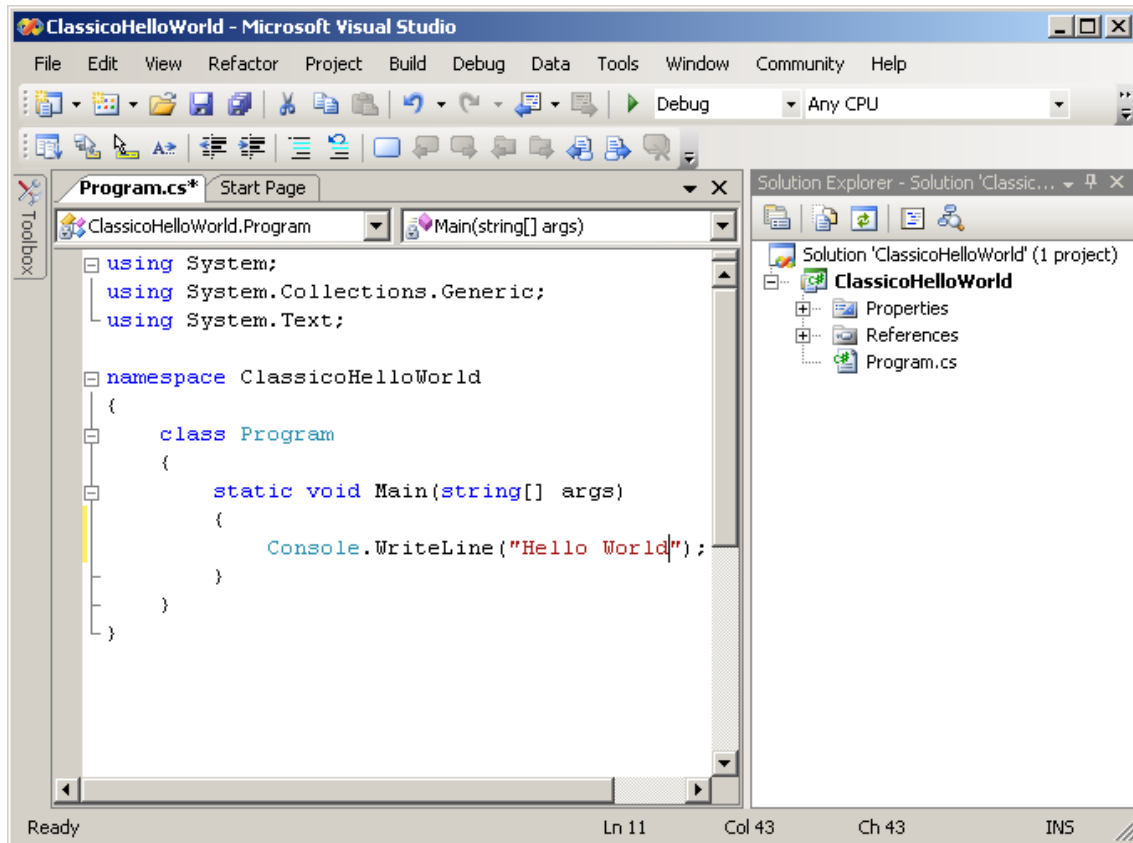


12 – Digite entre os parênteses a string **“Hello World”**, string deve ficar entre aspas. Vai ficar assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



13 – Vamos agora compilar nossa aplicação. No menu **Build**, clique em **Build Solution**. Se tudo estiver certinho vai aparecer a seguinte linha na **janela Output**:

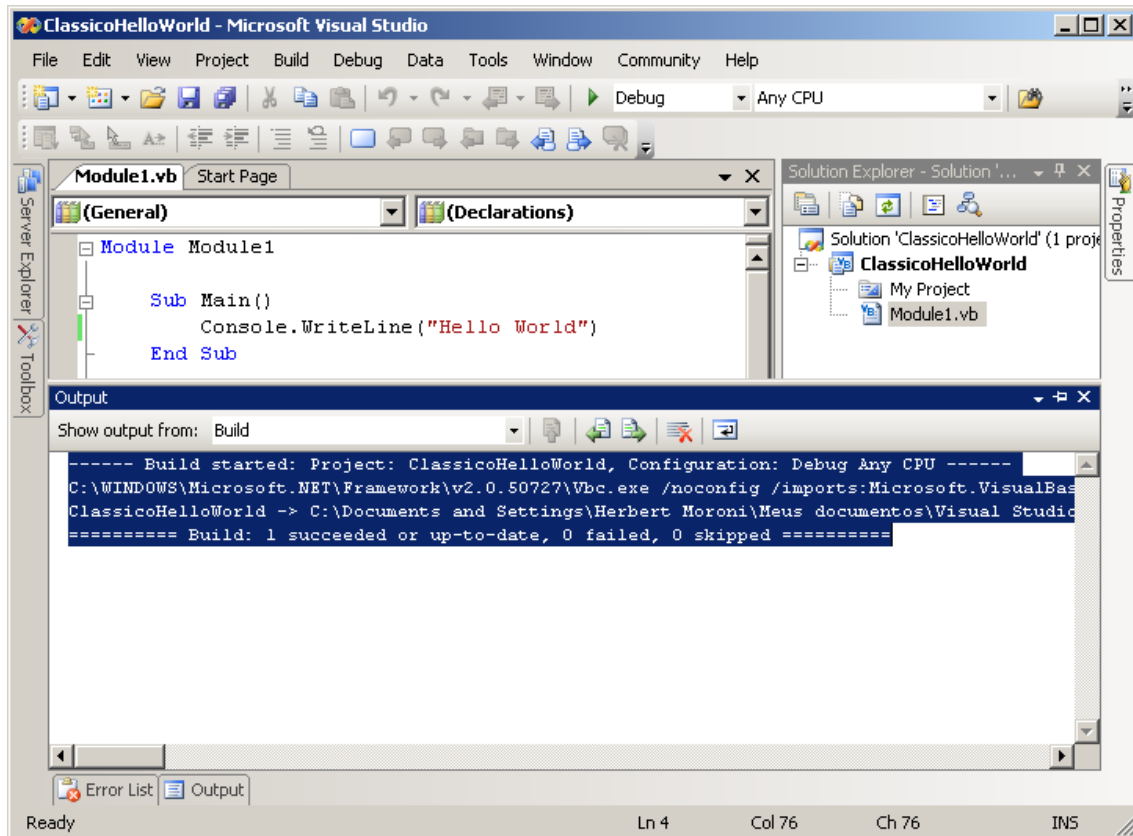
```
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
```

Para exibir a janela Output na barra de menus clique em View, Output ou pressione Ctrl+W+O.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



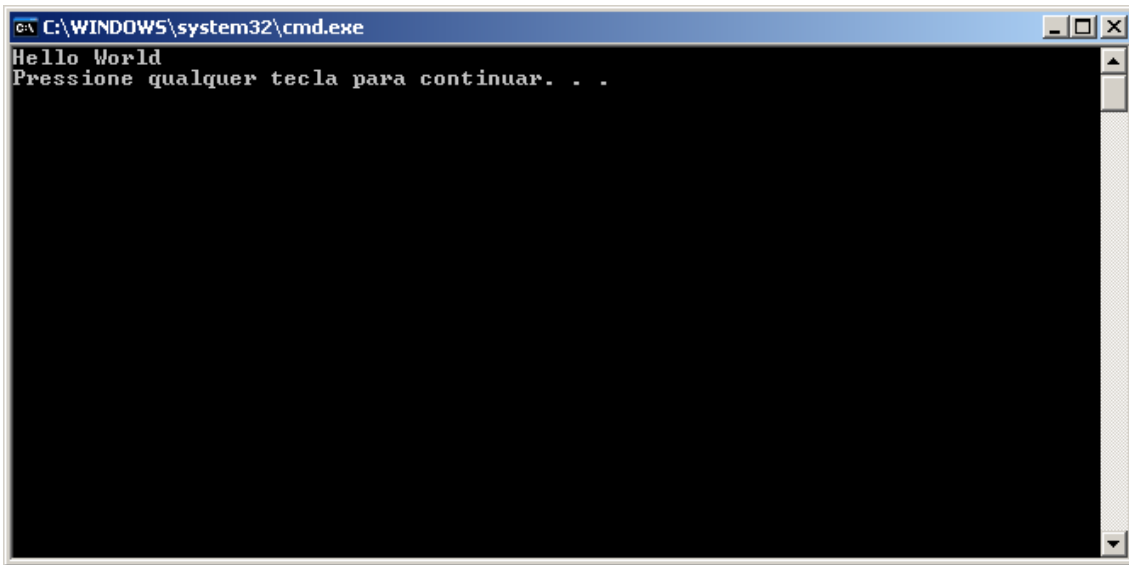
Um asterisco depois do nome do arquivo no painel de código indica que foram feitas modificações no código do respectivo arquivo e que essas alterações não foram salvas. Você pode salvar manualmente antes de compilar a aplicação, mas ao compilar o Visual Studio salva automaticamente todos os arquivos da aplicação.

14 – No menu **Debug**, clique em **Start Without Debugging** para executar o programa no prompt de commando.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



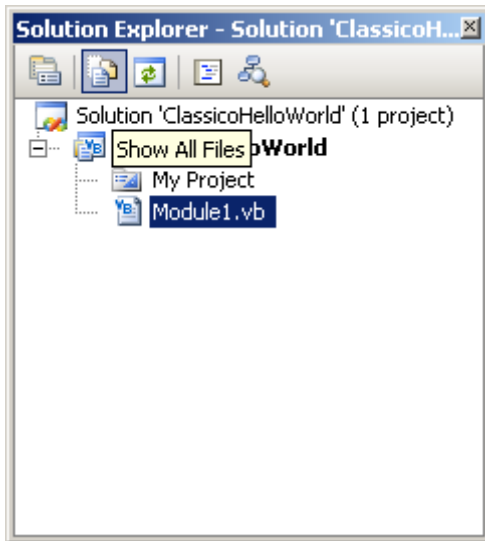
O programa vai escrever **Hello World** como a ilustração acima.

Nós escolhemos **Start Without Debugging** para forçar uma pausa no final da execução. Se clicássemos em **Start** ele iria executar o programa e fechar o prompt de comando logo após a execução, seria tão rápido que não conseguiríamos ver o que foi escrito, experimente.

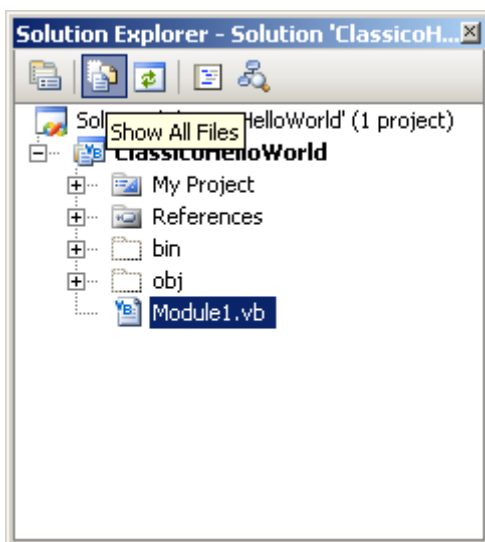
15 – Com o foco no prompt de comando pressione qualquer tecla.

A janela irá fechar e retornaremos para o Visual Studio.

16 - Na janela **Solution Explorer**, clique no botão **Show All Files**.



Aparecem os nomes **bin** e **obj** depois do nome do projeto. Esses dois correspondem a pastas com seus respectivos nomes. Essas pastas são criadas quando você executa a aplicação e contem uma versão executável do programa e outros arquivos necessários para depurar o programa.

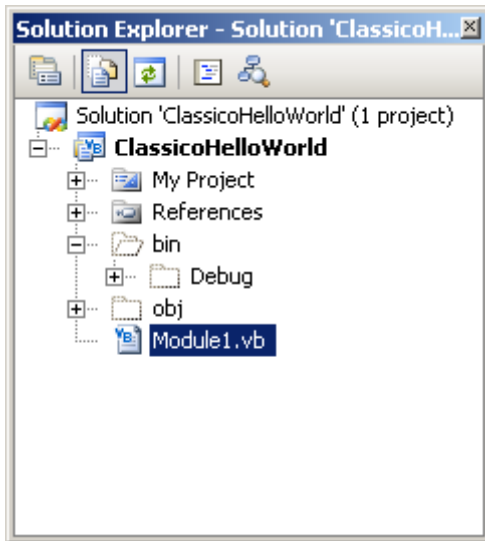


17 – Ainda na janela **Solution Explorer**, clique no sinal de + à esquerda do nome **bin**.

Autor: Herbert Moroni Cavallari da Costa Gois

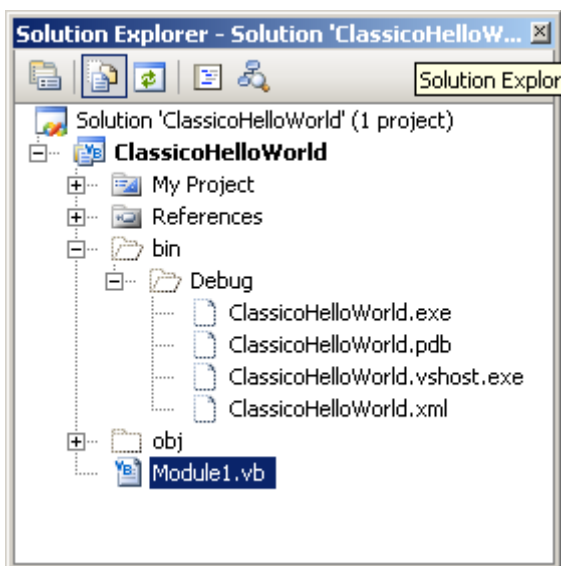
Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”



Um outro nome aparece representando uma outra pasta chamada **debug**.

18 – Clique no sinal de + de **debug**.



Repare nos arquivos: **ClassicoHelloWorld.exe** e **ClassicoHelloWorld.pdb**.

O arquivo .exe é o executável da aplicação.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

O arquivo .pdb armazena informações de depuração da aplicação.

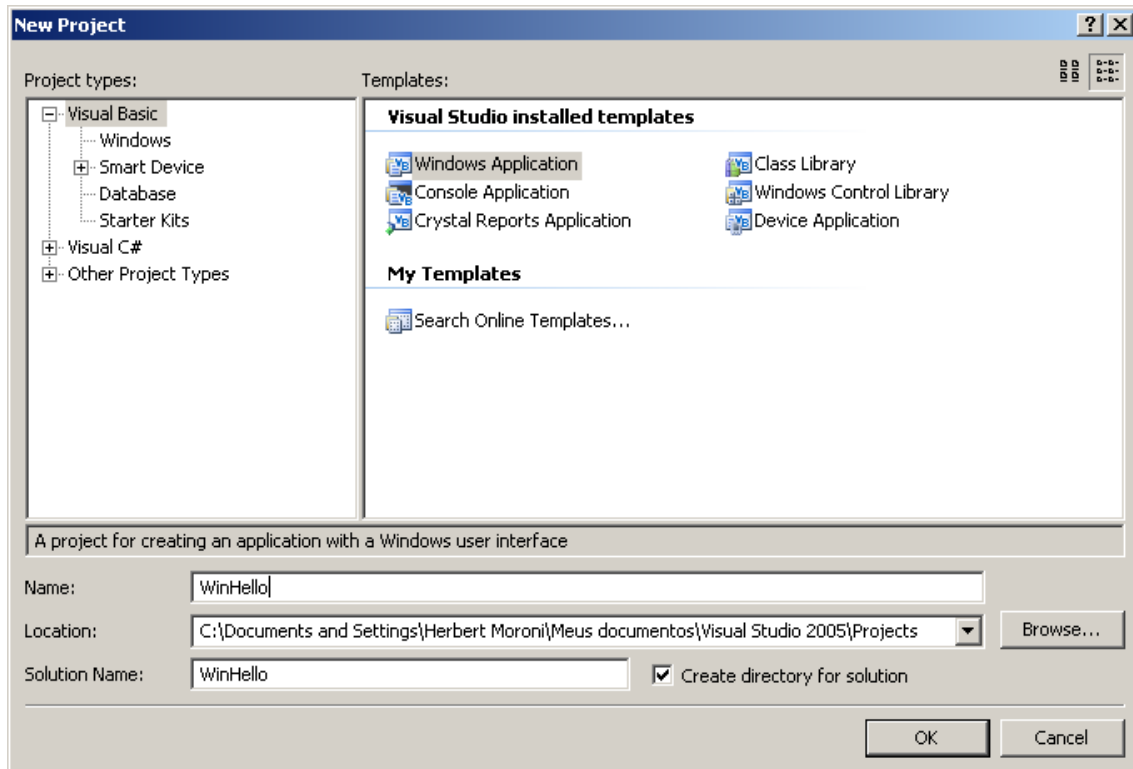
O arquivo **ClassicoHelloWorld.vshost.exe** só aparece no Visual Studio 2005, ele é usado para melhorar a performance da depuração de erros.

Como vimos até agora, o Visual Studio compilou automaticamente nosso programa e criou os arquivos necessários automaticamente, durante o processo de compilação. Em resumo a compilação é o processo que transforma seus arquivos fonte em um arquivo executável pelo sistema operacional, um **.exe** por exemplo.

Até agora por motivo didático usamos somente o prompt de comando para criar os nossos exemplos. Como sabemos esse tipo de aplicação não é muito útil nos dias de hoje. O Visual Studio .NET conta com diversos recursos importantes para o desenvolvimento de aplicações Windows.

1 – Entre no Visual Studio .NET.

2 – Crie um novo projeto, só que desta vez do tipo **Windows Application**, chamado **WinHello**.

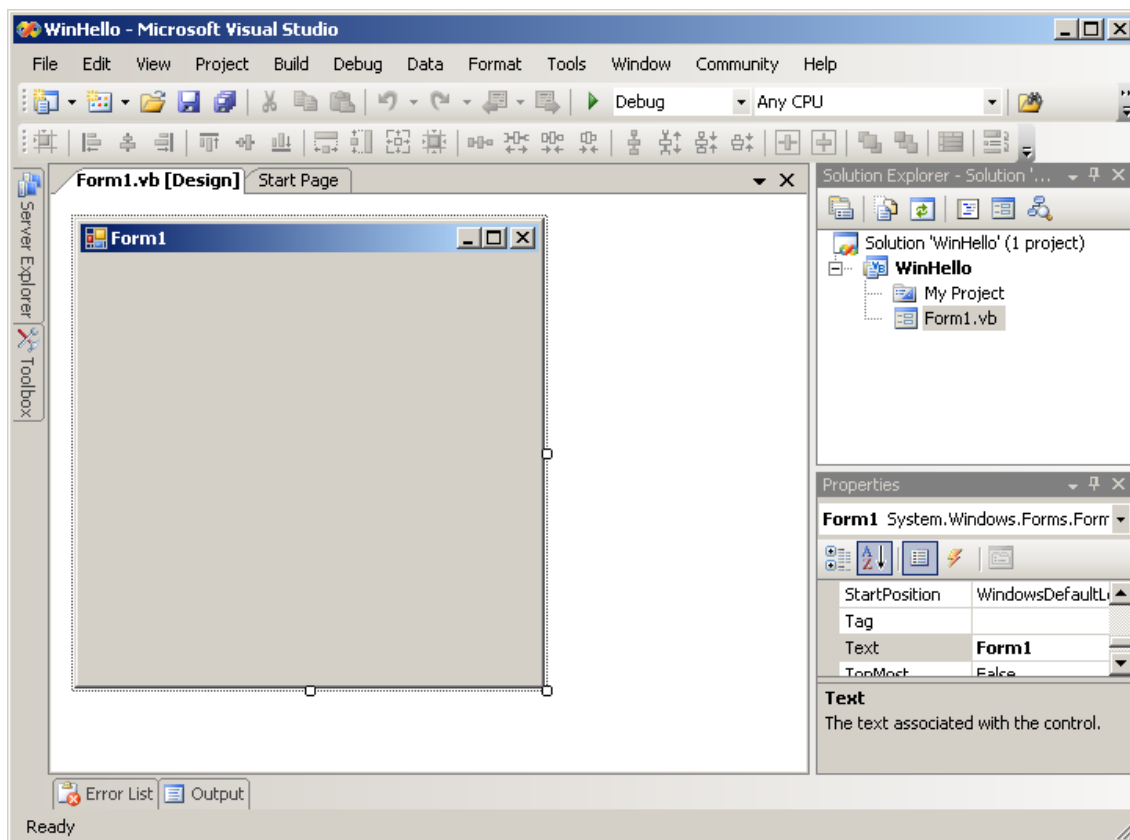


O Visual Studio .NET cria e mostra um formulário baseado em Windows no modo Design.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



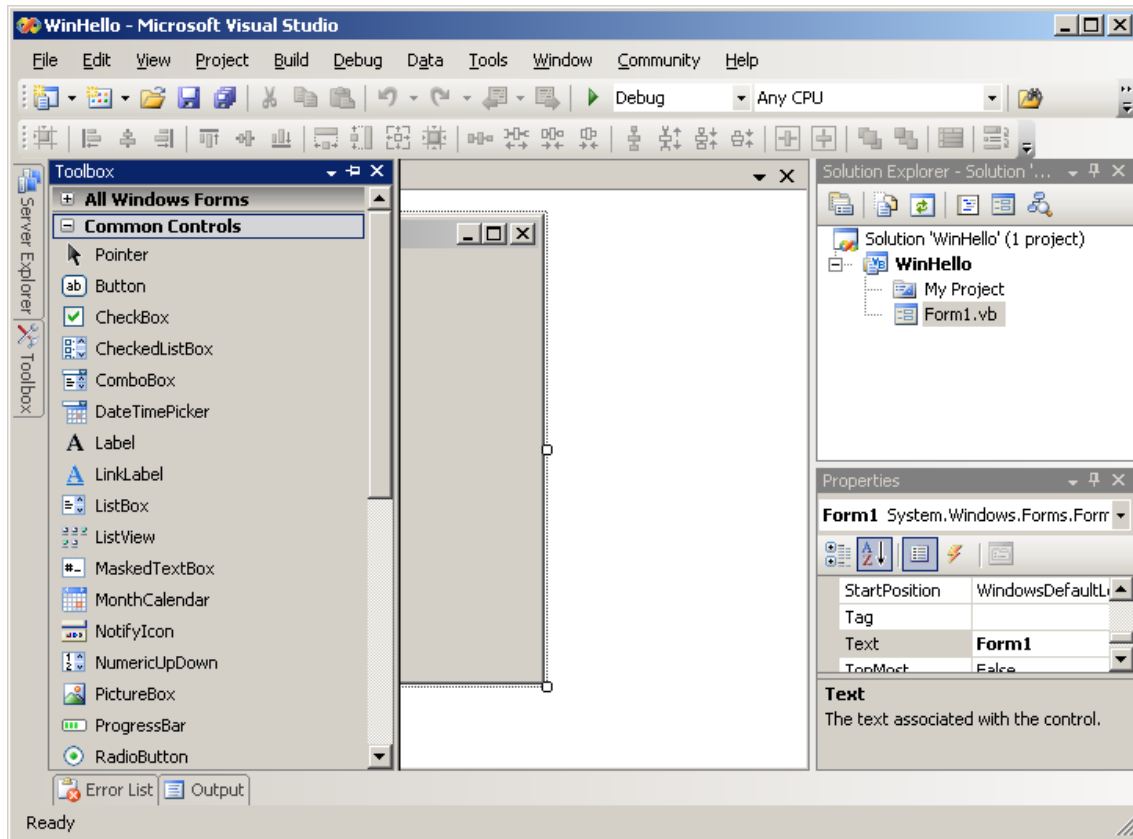
Vamos agora criar a nossa interface com o usuário.

3 – Na barra de ferramentas do Visual Studio .NET clique em **ToolBox**. O ícone da ToolBox aparece a esquerda do formulário. Você também pode localizar a ToolBox através do menu View > Toolbox.

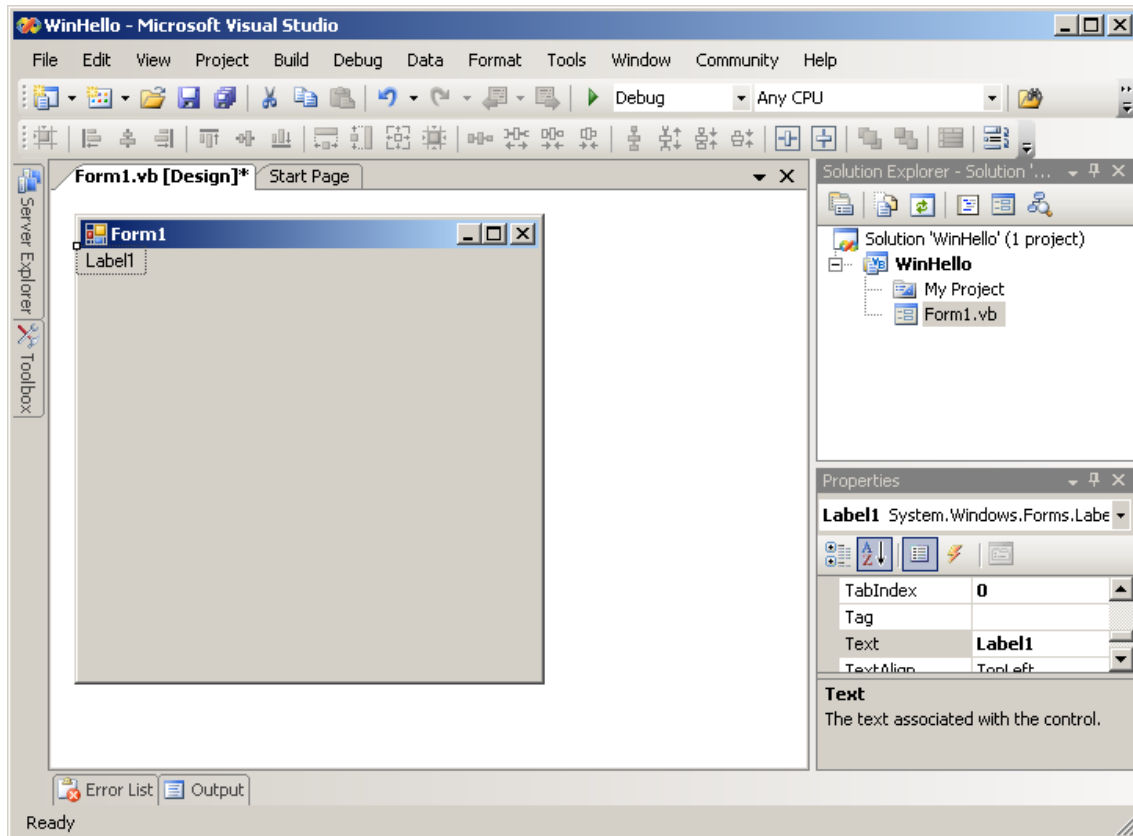
Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

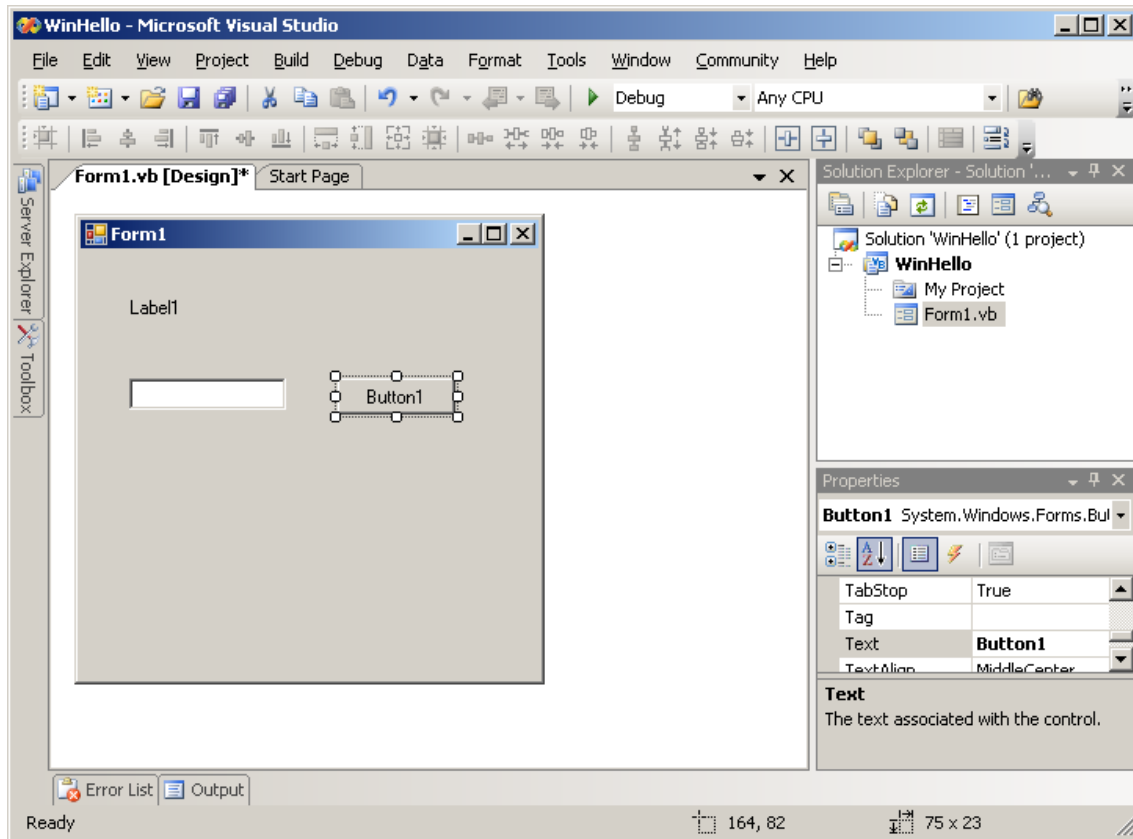


4 – Arraste da barra de ferramentas o controle **Label** e posicione-o no canto superior esquerdo do formulário.



Para colocar um controle no formulário você pode também dar um clique duplo sobre ele na barra de ferramentas ou clicar uma vez sobre ele na barra de ferramentas e depois clicar no formulário. O clique duplo posiciona o controle no canto superior esquerdo. A segunda opção coloca o controle no local onde você clicar.

5 – Coloque também no formulário um controle **TextBox** e um controle **Button**. Como na próxima ilustração:



6 – Na janela **Solution Explorer**, clique no botão **View Code**.



O código do arquivo **Form1.vb** aparece.

Para voltar ao modo design, também na janela **Solution Explorer** clique em **View**

Design.



Form1.vb tem todo o código gerado automaticamente pelo Visual Studio .NET.

Note os seguintes elementos no código uma classe chamada **Form1**.


Vamos agora definir as propriedades dos controles que colocamos no Form.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: “Programando com VB.NET” e “Programando com C# e o Visual Studio .NET 2005”

7 – Volte para o modo Design.

Para voltar ao modo design, também na janela **Solution Explorer** clique em **View Design**. 

8 – De um clique sobre o **Button1** para selecioná-lo.

9 – Na janela **Properties**, altere a propriedade **Text** do **button1** para OK.

Se não localizar a janela **Properties**, clique em F4, ou no **menu View**, clique em **Properties Window**.

10 – Altere também a propriedade **Text** do **Label1** para *Digite o seu nome*

11 – Altere agora a propriedade **Text** do controle **textBox1** para *aqui*.

Note que as propriedades modificadas na janela **Properties** ficam em negrito. Assim você pode saber se elas estão com seu valor padrão ou não.

12 – Selecione o formulário clicando sobre ele.

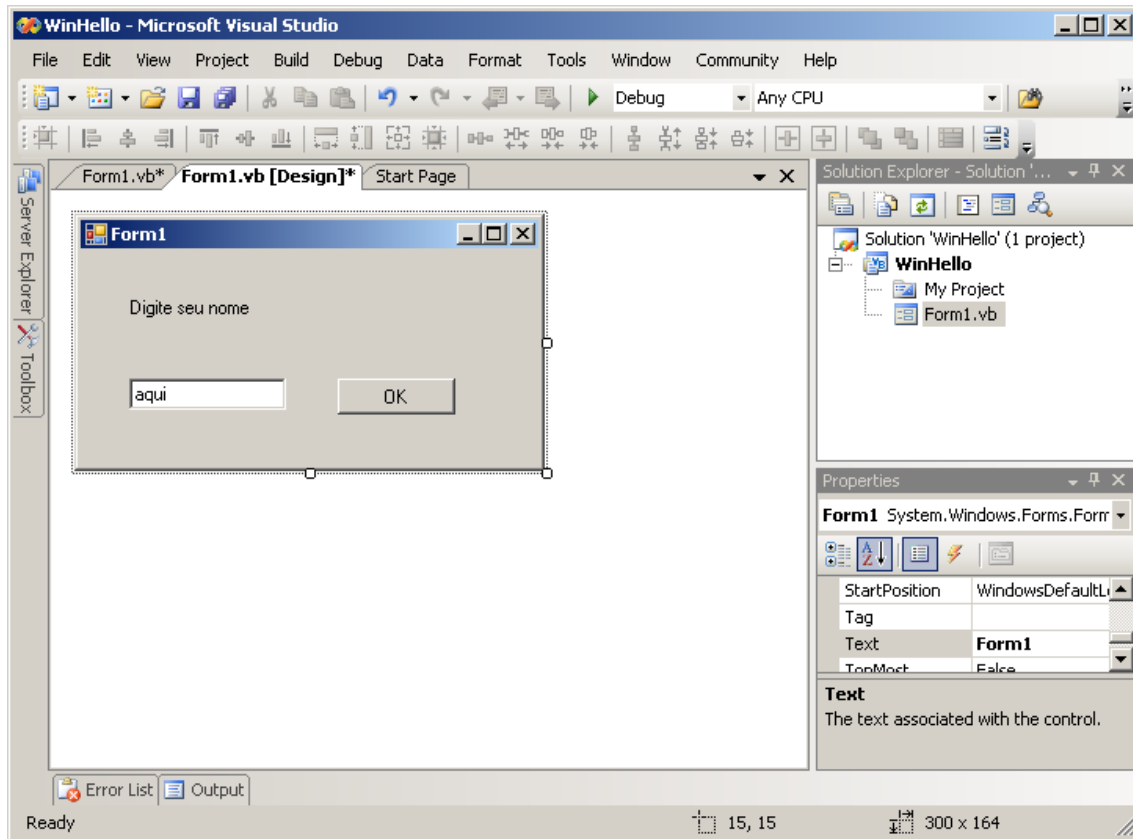
Note que aparecem alguns marcadores envolta do formulário. Eles ajudam a redimensionar o formulário.

13 - Clique sobre o marcador central na parte de baixo do **Form1** e mantendo o botão pressionado arraste para cima.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Isso serve para os outros controles também, clique sobre os outros controles e note os marcadores.

Vamos agora escrever o código para o nosso programa.

14 – No painel de código de um clique duplo sobre o **Button1**.

Note que ele vai diretamente para o painel de código e é criado automaticamente o seguinte código.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
  
End Sub
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Tudo que for digitado dentro deste código será executado assim que o **Button1** for clicado quando o programa estiver executando.

15 - Digite o seguinte código:

```
MessageBox.Show("Hello " + textBox1.Text)
```

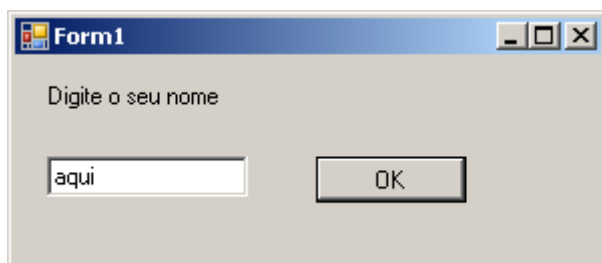
Vai ficar assim:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
    MessageBox.Show("Hello " + TextBox1.Text)  
End Sub
```

16 - Execute o programa.

Para executar o programa você pode clicar e **F5**, ou no menu **Debug** clicar em **Start Debugging**.

Automaticamente o Visual Studio .NET salva o programa, compila e o executa. A seguinte janela aparece:



17 - Digite seu nome e clique em OK.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Uma janela aparece exibindo a mensagem "Hello seu-nome".



18 - Clique em **Ok** para fechar a janela com a mensagem.

19 - Na janela executando o **Form1** clique em fechar.

10.1 – Tipos de dados em VB.NET

Categoria	Tipo	Valores possíveis de se armazenar
Inteiro	Byte	0 a 255 (8 bits)
Inteiro	SByte	-128 a 127 (8 bits)
Inteiro	Integer	-2,147,483,648 a 2,147,483,647 (32 bits)
Inteiro	UInteger	0 a 4,294,967,295 (32 bits)
Inteiro	Long	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (64 bits)
Inteiro	ULong	0 a 18,446,744,073,709,551,615 (64 bits)
Inteiro	Short	-32,768 a 32,767 (16 bits)
Inteiro	UShort	0 a 65,535 (16 bits)
Real	Decimal	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$ (128 bits)

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Real	Double	$\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$ (64 bits)
Real	Single	$\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$ (32 bits)
Lógico	Boolean	Verdadeiro ou Falso (Valores booleanos)
Literal	Char	Um caractere (16 bits)
Literal	String	Seqüência de caracteres (16 bits por caractere)

Em VB.NET os tipos **UInteger**, **ULong** e **UShort**, ocupam o mesmo espaço que os tipos **Integer**, **Long** e **Short**. Só que só podem armazenar números positivos.

Os tipos **UInteger**, **ULong** e **UShort** são conhecidos como **Tipos não assinados**.

10.2 – Declarando Variáveis em VB.NET

Segue a lista de palavras-reservadas do VB.NET, lembre-se que elas não podem ser usadas como nome de variáveis.

AddHandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar	CDate
CDec	CDBl	Char	CInt	Class
CLng	CObj	Const	CShort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	ElseIf	End	Enum
Erase	Error	Event	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Function	Get	GetType	Goto
Handles	If	Implements	Imports	In
Inherits	Integer	Interface	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads	Overridable
Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw
To	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	eval	extends
instanceof	package	var		

Em VB.NET declaramos usando o comando **Dim** como segue:

Dim contador As Integer

Sempre depois de **As** vem o tipo de dado.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Esse exemplo declara uma variável chamada contador do tipo *integer*.

Podemos também declarar múltiplas variáveis de uma vez, fazemos isso da seguinte maneira:

Dim contador, numeroCarro as Integer

Estamos declarando nesse exemplo duas variáveis do tipo integer, uma chamada contador e a outra numeroCarro.

10.3 – Constantes em VB.NET

Para declarar uma constante em VB.NET você usa o comando Const na declaração da variável ao invés de Dim. Exemplo:

Const pi as Double = 3.1415

10.4 – Operadores de atribuição em VB.NET

Exemplo de como atribuir um valor a uma variável em VB.NET.

Dim numeroFuncionario as Integer

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
numeroFuncionario = 23
```

Primeiro nós declaramos nossa variável do tipo integer. Depois atribuímos o valor 23 a ela. Entendemos pelo sinal de igual como recebe. Assim numeroFuncionario recebe 23.

Podemos também atribuir um valor a variável quando a declaramos, dessa forma:

```
Dim numeroFuncionario as Integer = 23
```

Isso faz à mesma coisa que o exemplo anterior, só que tudo em uma linha.

Valores do tipo String ou Character devem ser inseridos entre aspas duplas, exemplo:

```
Dim nome As String = "Moroni"
```

10.5 - Operadores aritméticos em VB.NET

Operador	Representação em VB.NET	Exemplo
Incremento	++	a ++ Neste caso é somado 1 ao valor de a.
Decremento	--	a --

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		Neste caso é subtraído 1 do valor de a.
Multiplicação	*	$a * b$ Multiplica-se o valor de a por b.
Divisão	/	a / b Divide-se o valor de a por b.
Exponenciação	^	$2 ^ 3$ Aqui representamos 2 elevado a 3 ou 2^3 .
Módulo	Mod	$a \text{ mod } b$ Retorna o resto da divisão de a por b.
Adição	+	$a + b$ Soma de a com b.
Subtração	-	$a - b$ Subtração de a com b.

10.6 – Operadores relacionais em VB.NET

Operador	Representação em VB.NET	Exemplo
Maior	>	$a > b$ Se o valor de a for maior que o valor de b então o resultado dessa expressão

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		é verdadeiro senão é falso.
Maior ou igual	\geq	$a \geq b$ Se o valor de a for maior ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Menor	$<$	$a < b$ Se o valor de a for menor que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Menor ou igual	\leq	$a \leq b$ Se o valor de a for menor ou igual que o valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Igual a	$=$	$a = b$ Se o valor de a for igual ao valor de b então o resultado dessa expressão é verdadeiro senão é falso.
Diferente de	\neq	$a \neq b$

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		Se o valor de a for diferente do valor de b então o resultado dessa expressão é <u>verdadeiro</u> senão é falso.
--	--	--

10.7 - Operadores Lógicos em VB.NET

Operador	Representação em VB.NET	Exemplo
E	And	a = 5 And b > 9 Caso o valor de a seja igual a cinco e o valor de b maior que nove o resultado é verdadeiro, senão é falso.
Ou	Or	a = 5 Or b > 9 Se o resultado de a for igual a cinco ou o valor de b for maior que nove então o resultado é verdadeiro. O resultado só será falso é as duas expressões retornarem falso.
Não	Not	Not a = 5 Se o resultado de a for

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

		igual a 5 então o resultado será falso, o operador não inverte o resultado da expressão.
--	--	--

10.8 - Entrada e Saída de dados em VB.NET

Para entrada de dados em VB.NET usamos o comando **ReadLine**, e para a Saída de dados o comando **WriteLine**. A linguagem VB.NET e C# tem vários comandos similares, já que ambas foram desenvolvidas para serem utilizadas na plataforma .NET da Microsoft. No entanto a sintaxe do C# é mais parecida com a do C, por ser uma linguagem que vem do C.

Segue um exemplo em VB.NET:

```
Dim nome As String

Console.WriteLine("Digite seu nome: ")

nome = Console.ReadLine

Console.WriteLine("Seu nome é: " & nome)
```

No exemplo acima, primeiro declaramos uma variável do tipo **string** chamada **nome**, depois usamos o comando de saída **WriteLine** para imprimir **Digite seu nome na tela**. Na terceira linha atribuímos o valor que foi digitado à variável **nome** usando o comando de entrada **ReadLine**. Por último exibimos na tela o

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

nome digitado novamente usando o comando **WriteLine**. O operador & foi utilizado para concatenar o texto **Seu nome é** com o conteúdo da variável **nome**.

10.9 – Estruturas de decisão em VB.NET

A primeira estrutura de decisão que iremos estudar é o **IF**, veja a sintaxe a seguir:

```
If condição Then
    código
ElseIf condição Then
    código
Else
    código
End If
```

IF condição **THEN** então faça algo; caso contrário **Else** ou ainda você pode testar outra condição **ElseIf** até que o programa encontre o resultado correto.

Veja o seguinte exemplo:

```
Dim Salario As Double = 1200
If Salario > 1000 Then
    Salario *= 1.1
End If
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Existe uma variável chamada Salario que está definida e contém o valor 1200. Como **toda condição retorna apenas verdadeiro ou falso**, É verificado se o valor é maior que 1000, e caso seja verdadeiro será somado 10% ao Salario.

Vamos a mais alguns exemplos para facilitar o entendimento.

```
Dim Salario As Double = 1200

If Salario < 500 Then

    Salario += 50

Else

    Salario += 100

End If
```

Aqui é verificado se o valor é menor que 500 e dependendo da condição é somado 50 ou 100, pois há o Else. Então se a variável for menor que 500 adicionamos nela o valor 50, senão adicionamos 100.

```
Dim Salario As Double = 1200

If Salario < 500 Then

    Salario += 50

ElseIf Salario >= 500 And Salario < 2000 Then

    Salario += 100

Else

    Salario += 250

End If
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Já no exemplo acima existem 3 condições, onde o primeiro IF soma 50, o segundo 100 e o ELSE 250. **O Else sempre é executado quando nenhuma expressão é verdadeira.**

É possível avaliar diversos ElseIfs com uma determinada expressão. Como no exemplo abaixo:

```
Dim Salario As Double = 1200

If Salario < 500 Then
    Salario += 50
ElseIf Salario >= 500 And Salario < 600 Then
    Salario += 100
ElseIf Salario >= 600 And Salario < 700 Then
    Salario += 110
ElseIf Salario >= 700 And Salario < 800 Then
    Salario += 120
Else
    Salario += 250
End If
```

Perceba que no bloco de código acima usamos o **And**.

O **And** é usado quando precisamos testar mais de uma expressão condicional.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Exemplo:

```
ElseIf Salario >= 500 And Salario < 600 Then
```

Nesta linha testamos se o salario é maior ou igual a 500 e o salario é menor que 600.

Então se o salario for 553 a expressão acima é VERDADEIRA, caso contrario é FALSA.

A segunda estrutura de decisão que iremos estudar é conhecida como **Select Case**.

Essa estrutura de decisão seleciona o código a ser executado baseado no valor de uma expressão.

A sintaxe é a seguinte:

```
Select Case expressão de teste
    Case item de teste
        código
    Case Else
        código
End Select
```

O funcionamento ocorre da seguinte maneira: a expressão é obtida no **Select Case** e para cada **Case** existe uma condição a ser validada. Caso o **Case** seja verdadeiro, então a linha ou o bloco de código é executado. Se nenhum dos Cases for válido,

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

you can use the **Case Else**. The **Case Else** is optional. You can have as many Cases as necessary.

See the example:

```
Dim diaDaSemana As Integer = 3

Select Case diaDaSemana
    Case 1
        MsgBox( "Domingo" )
    Case 2
        MsgBox( "Segunda-Feira" )
    Case 3
        MsgBox( "Terça-Feira" )
    Case 4
        MsgBox( "Quarta-Feira" )
    Case 5
        MsgBox( "Quinta-Feira" )
    Case 6
        MsgBox( "Sexta-Feira" )
    Case 7
        MsgBox( "Sabado" )
End Select
```

First we create a variable of type **integer** and assign it the value 3.

Next we start the structure by putting the value of the variable as the test expression.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

O primeiro **Case** verifica se o valor da variável ou expressão de teste é 1. Se for o código que escreve na tela o texto *Domingo* é executado e a estrutura é finalizada não verificando os outros Cases. Caso contrario ele vai pro próximo **case** e segue como o primeiro até o fim da estrutura. Se nenhum dos cases se enquadrar no valor da expressão de teste nenhum código será executado, para isso que serve o **Else**. No próximo exemplo vamos apenas programar o uso do **Else**:

```
Dim diaDaSemana As Integer = 3

Select Case diaDaSemana
    Case 1
        MsgBox( "Domingo" )
    Case 2
        MsgBox( "Segunda-Feira" )
    Case 3
        MsgBox( "Terça-Feira" )
    Case 4
        MsgBox( "Quarta-Feira" )
    Case 5
        MsgBox( "Quinta-Feira" )
    Case 6
        MsgBox( "Sexta-Feira" )
    Case 7
        MsgBox( "Sabado" )
    Case Else
        MsgBox( "Numero Inválido" )
End Select
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

- Aqui se o resultado da expressão de teste não for verdadeiro em nenhum **Case**, então será executado o **Case Else**, escrevendo na tela que o valor informado é inválido, o que não é o caso para o nosso exemplo, já que o valor da expressão se enquadra no **Case 3** escrevendo *Terça-Feira*.

10.10 – Estruturas de repetição em VB.NET

O looping **While** é executado sempre associado a uma condição, ou seja, a cada passagem pelo looping a condição é avaliada. Veja a sintaxe a seguir:

```
While condição  
    código  
End While
```

Vamos fazer um exemplo para você compreender melhor o funcionamento do while.

- 1 - Crie um novo projeto no Visual Studio, novamente do tipo **Windows Application**, chame o projeto de **while**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

2 - Adicione no **Form1** os seguintes controles:

- 1 Button
- 1 TextBox

3 - Mude a propriedade **Text** do **Button1** para "Loop".

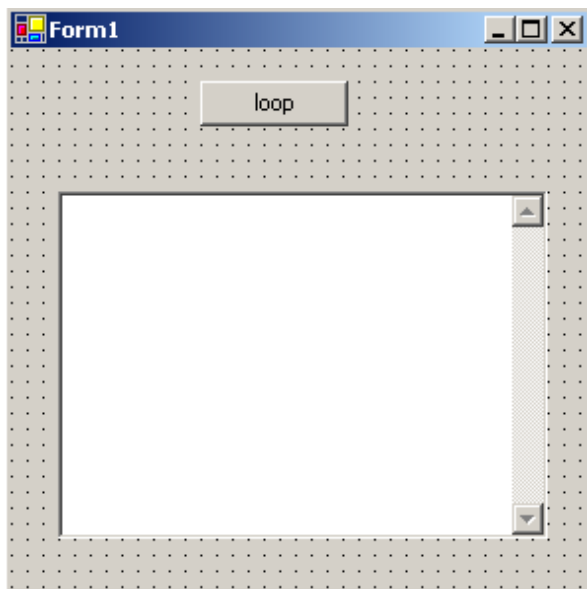
4 - Mude as seguintes propriedades do **TextBox1**:

Text = "vazio"

Multiline = True

ScrollBars = Vertical

5 - Organize os controles como a figura abaixo:



6 - De um clique duplo sobre o **Button1** e digite o seguinte código:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Dim Contador As Integer = 0

Dim Quebra As String = Chr(13) & Chr(10)

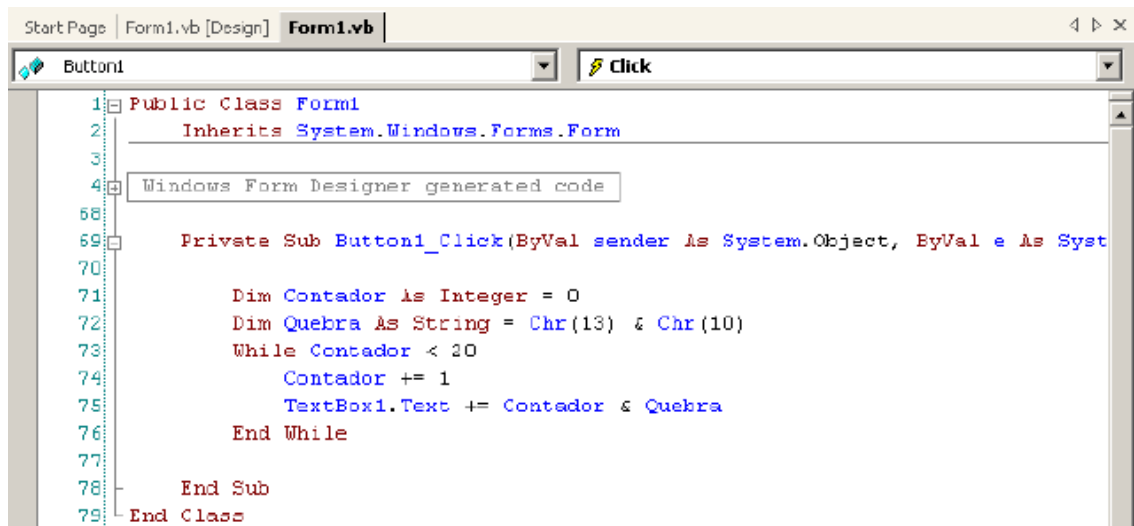
While Contador < 20

    Contador += 1

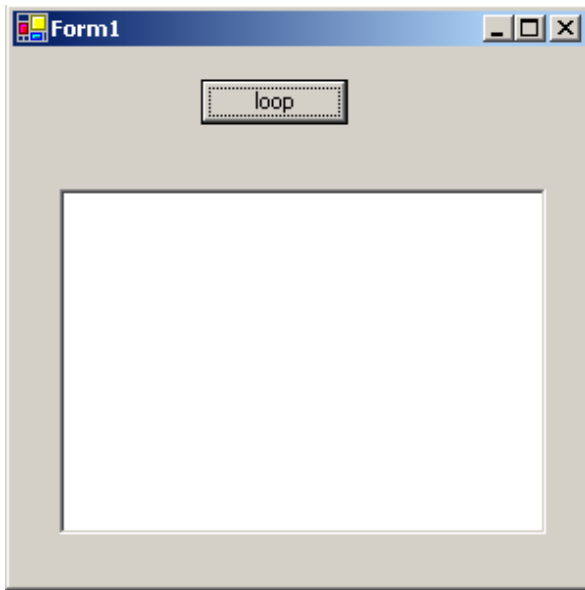
    TextBox1.Text += Contador & Quebra

End While
```

Vai ficar assim:

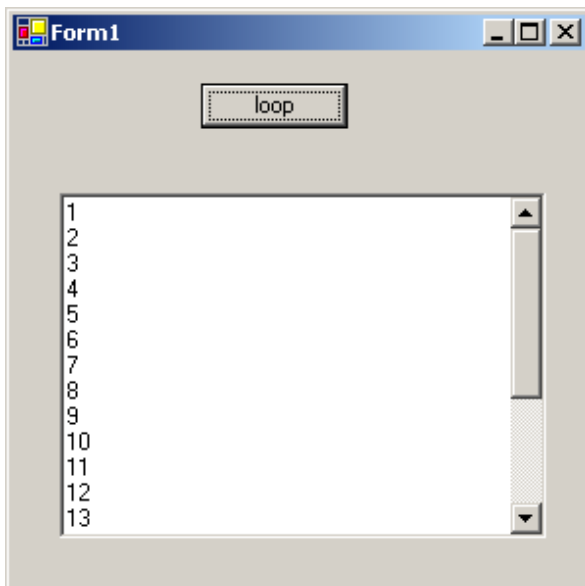


7 - Execute a aplicação.



8 - Clique no botão loop.

Verifique que o nosso programa escreve os números de 1 a 20, um embaixo do outro.



Esse é nosso código, vamos estudá-lo um pouco:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"


```
Dim Contador As Integer = 0

Dim Quebra As String = Chr(13) & Chr(10)

While Contador < 20

    Contador += 1

    TextBox1.Text += Contador & Quebra

End While
```

Na primeira linha declaramos uma variável chamada **contador** do tipo **integer** e atribuímos a essa variável o valor 0.

Na segunda declaramos uma segunda variável chamada **quebra** do tipo **String**, e armazenamos nela uma string que representa o pressionamento da tecla Enter, isso é responsável por fazer com que o próximo numero seja exibido na linha abaixo, sem isso os números ficariam um seguido do outro, faça o teste.

A terceira linha começa o **While** com a condição. Essa linha quer dizer enquanto contador for menor que 20, ou seja ele vai executar o loop ou o código que esta entre While e End While até que o conteúdo da variável **contador** seja igual ou maior que 20.

A linha 4 soma 1 ao conteúdo da variável contador, a linha a seguir tem o mesmo significado:

```
Contador = Contador + 1
```

Entretanto da forma que fizemos no nosso exemplo é mais elegante porque não precisamos repetir o nome da variável, se apenas fizéssemos assim:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Contador = 1
```

Ele atribuiria 1 ao conteúdo da variável e o nosso looping se tornaria um loop infinito ou seja nunca ia parar de rodar porque nunca entraria na condição que o faz parar. Cuidado com esse tipo de loop, seus loops nunca devem ser infinitos.

A linha 5 atribui ao **TextBox** o valor da variável contador concatenado com o valor da variável **quebra** que simula o enter como especificado acima.

A linha 6 finaliza o While.

Repare que após o numero 20 no programa tem uma linha em branco. Isso é porque sempre é inserida uma linha no **TextBox** quando chamada a variável **Quebra**.

O **Do...Loop** é semelhante ao while, ou seja, é executado sempre associado a uma condição, novamente a cada passagem pelo looping a condição é avaliada. Só que o **Do...Loop** permite que dependendo da situação, a condição possa ser colocada no início ou no final do looping. Se for ao início, é avaliada logo na primeira vez; se for no final, o looping é executado pelo menos a primeira vez, pois a condição será avaliada no final da primeira passagem pelo looping. Caso precise abandonar o looping, use o **Exit Do**, que é opcional. Veja a sintaxe tanto com a condição no começo como no final:

```
Do { While | Until } condição  
    código  
Exit Do
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Loop

ou

Do

código

Exit Do

Loop { While | Until } *condição*

Depois do **Do**, você pode usar o **While** ou o **Until**, a diferença é que o **While** faz com que o loop seja executado até que a condição seja (falsa), e o **Until** até que a condição seja True (Verdadeira).

Vamos fazer uma modificação no exercício anterior para compreender como funciona o **Exit Do**.

1 - Se o exemplo anterior não estiver aberto no Visual Studio, abra-o.

2 - Vá para o código do **button1**, ele deve ser o seguinte:

```
Dim Contador As Integer = 0

Dim Quebra As String = Chr(13) & Chr(10)

While Contador < 20

    Contador += 1

    TextBox1.Text += Contador & Quebra

End While
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

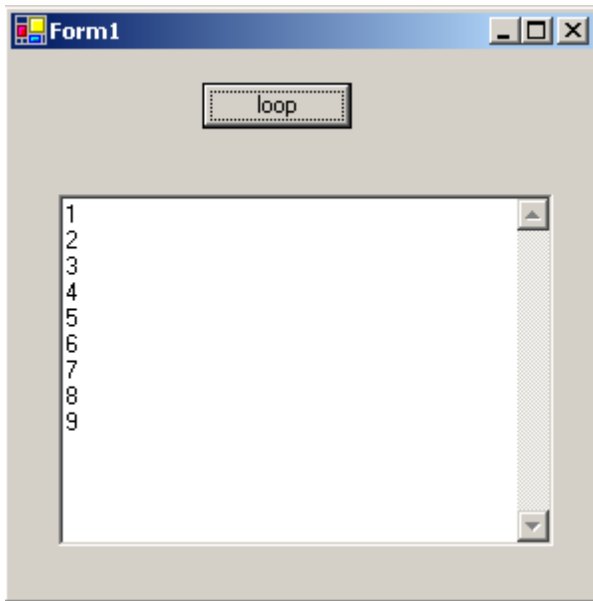
3 - Mude para o seguinte:

```
Dim Contador As Integer = 0
Dim Quebra As String = Chr(13) & Chr(10)
Do While Contador < 20
    Contador += 1
    If Contador = 10 Then
        Exit Do
    End If
    TextBox1.Text += Contador & Quebra
Loop
```

Perceba que as modificações foram mínimas, apenas colocamos o **Do** antes do **While**, substituímos o **End While** por **Loop**, transformando assim o simples **While** em um **Do...Loop**. Após inserimos um **If** que verifica se o conteúdo da variável **contador** é 10, se for ele executa o **Exit Do** que finaliza imediatamente o loop.

4 - Execute a aplicação:

5 - Clique em Loop.



Perceba que ele nem chega a escrever o numero 10 por que a linha de código que é responsável por isso esta depois do **If** que finalizou o loop.

Se você fizer a seguinte modificação ele escreve o 10 antes de finalizar.

```
Dim Contador As Integer = 0

Dim Quebra As String = Chr(13) & Chr(10)

Do While Contador < 20

    Contador += 1

    TextBox1.Text += Contador & Quebra

    If Contador = 10 Then

        Exit Do

    End If

Loop
```

6 - Agora mude o comando **While** para **Until** como a linha abaixo:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Do Until Contador < 20
```

7 - Execute a aplicação.

8 - Clique em Loop.

Nada acontece, não é mesmo? Isso porque o **Until** executa até que a expressão seja **True**. Como ela já é true nem chega a executar o loop.

9 - Você pode negar a expressão para ela executar, faça isso, mude a linha do Do Until... para a seguinte:

```
Do Until Not Contador < 20
```

10 - Execute a aplicação novamente e clique em Loop.

Agora executa normalmente.

11 - Agora tire o **not** e coloque a expressão no final do loop como o código a seguir:

```
Dim Contador As Integer = 0  
  
Dim Quebra As String = Chr(13) & Chr(10)  
  
Do  
  
    Contador += 1  
  
    If Contador = 10 Then  
  
        Exit Do
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
End If  
  
TextBox1.Text += Contador & Quebra  
  
Loop Until Contador < 20
```

12 - Execute a aplicação.

Se a expressão estivesse acima nada apareceria, como no exemplo acima, mas agora aparece o 1, isso porque ele executou uma vez o loop antes de avaliar a expressão, algumas vezes esse tipo de loop pode ser necessário. Fique atento.

O **For...Next** é usado quando sabemos o número de vezes que iremos executar o Loop. O For Next precisa de um contador que normalmente é uma variável e o looping vai do início (Start) até (TO).

A variável contador pode ser inicializada antes do For ou na própria declaração. Em certos loopings você pode usar o **Step** que é o incremento do looping, podendo ser positivo ou negativo.

Se durante o processamento você quiser abandonar o looping, terá que usar o **Exit For** que é semelhante ao Exit Do. A sintaxe do For...Next é a seguinte:

```
For contador = start To end Step  
  
    statements  
  
Exit For  
  
Next
```

Autor: Herbert Moroni Cavallari da Costa Gois

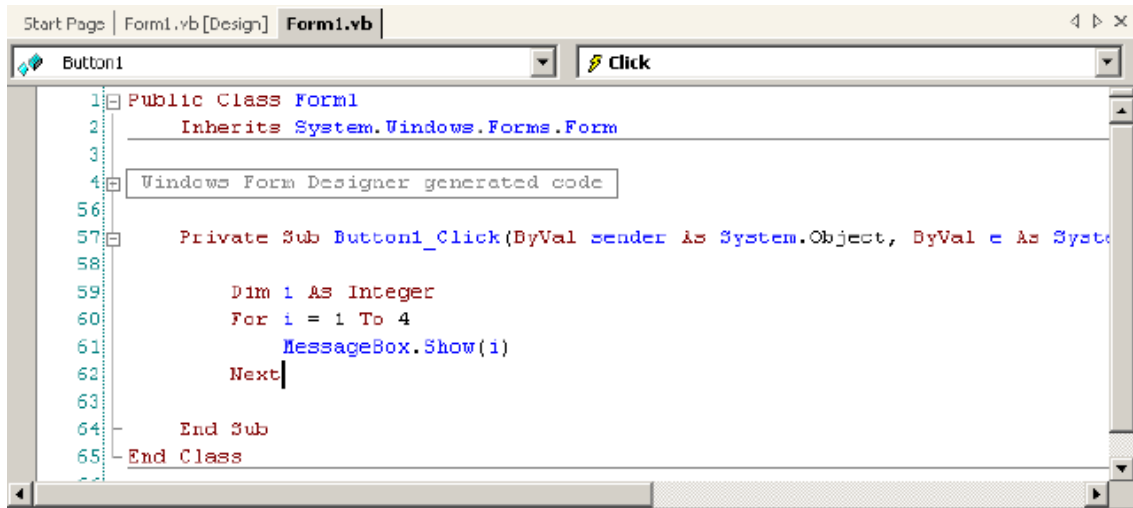
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

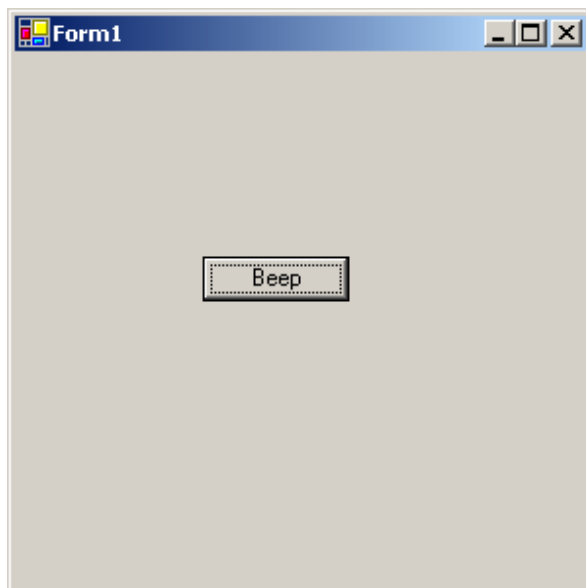
- 1 - Crie uma nova aplicação do tipo Windows Application com nome ForNext.
- 2 - Arraste para o **Form1** um **Button**.
- 3 - Mude a propriedade **Text** do **Button1** para "Beep".
- 4 - De um clique duplo sobre o botão e no digite o seguinte código para o evento do botão:

```
Dim i As Integer  
For i = 1 To 4  
    MessageBox.Show(i)  
Next
```

Vai ficar assim:



5 - Execute a aplicação.



6 - Clique no botão Beep.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Vamos dar uma avaliada no código:

```
Dim i As Integer  
For i = 1 To 4  
    MessageBox.Show(i)  
Next
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

A primeira linha cria uma variável do tipo **Integer**.

A segunda linha inicia o **For**, ele executará de 1 a 4.

A linha 3 é responsável por abrir uma caixa de diálogo na tela com o valor da variável **i**.

A linha 4 joga novamente para a linha 2.

A cada passada no For é adicionada 1 ao conteúdo da variável **i**, quando chega no 4 ele finaliza o loop.

7 - Mude o código do botão para o seguinte:

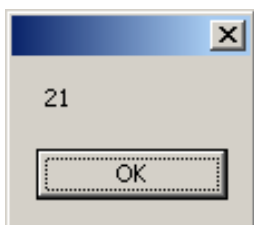
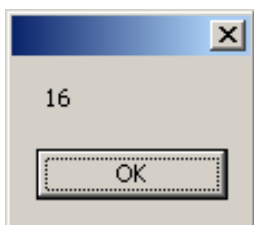
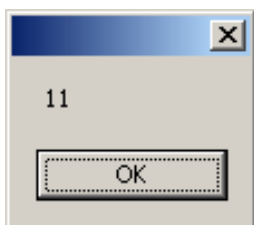
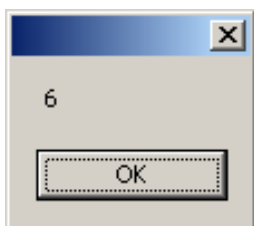
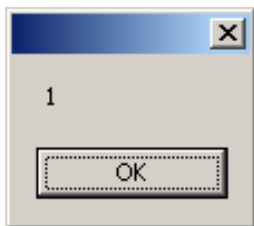
```
Dim i As Integer
For i = 1 To 25 Step 5
    MessageBox.Show(i)
Next
```

8 - Execute a aplicação e clique em Beep.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Perceba que o Step 5 fez com que o número adicionado a variável a cada passada do loop é 5 ao invés de 1.

10.11 – Vetores e matrizes em VB.NET

Vamos fazer um exemplo, nele vamos criar o programa que armazenará os dados nos nomes dos jogadores e as faltas cometidas por eles.

1 - Entre no Visual Studio e crie um novo projeto do tipo **Windows Application** chamado **Arrays**.

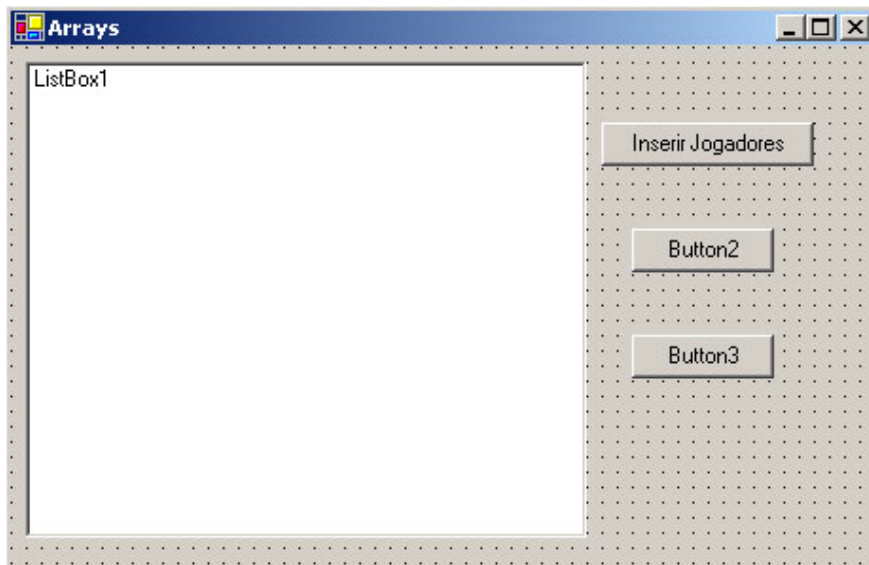
2 - Arraste para o Form1 os seguintes controles:

- 3 Button
- 1 ListBox

3 - Altere a propriedade Text do Button1 para Inserir Jogadores.

4 - Altere a propriedade Text do Form1 para Arrays.

5 - Organize-os como a figura abaixo:

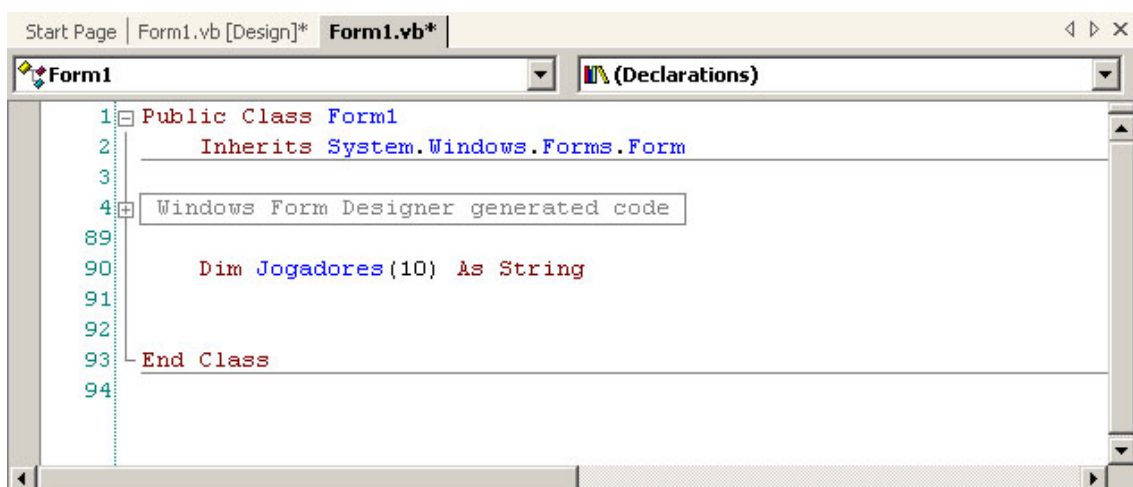


6 - Na janela **Solution Explorer** clique no botão **View Code** para ir para o Painel de código.

7 - Digite dentro da classe do **Form1** o seguinte código:

```
Dim Jogadores(10) As String
```

Vai ficar assim:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Nesta linha declaramos um Array de tamanho fixo. Perceba que a declaração é bem semelhante a de uma variável, com exceção do (10). O array que criamos é como uma tabela com 11 linhas, isso porque o array começa sempre no 0 e acaba no número que especificamos, no caso 10, o suficiente para armazenar o nome do time inteiro, 11 jogadores.

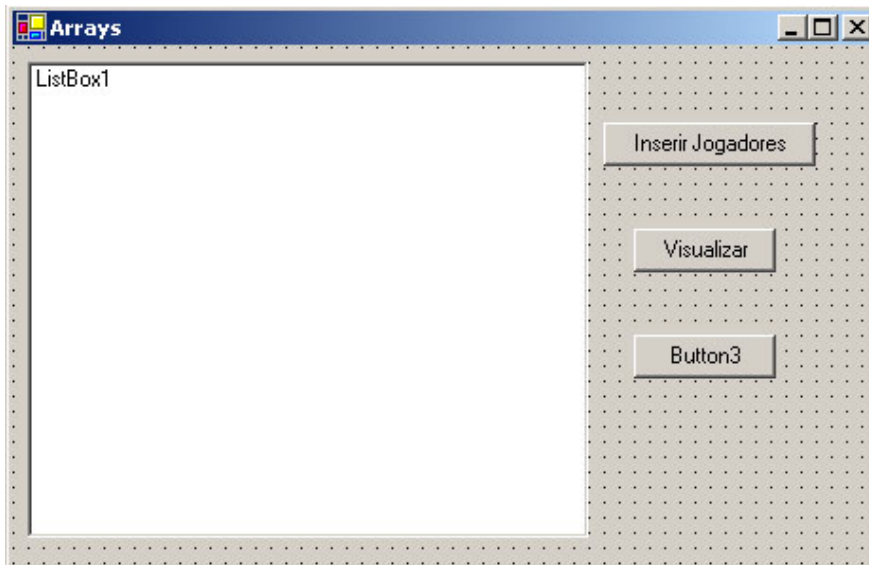
8 - Na janela **Solution Explorer** clique em **View Designer** para voltar para o **Form1** no modo Design, você pode também clicar na aba **Form1.vb[Design]** ao lado da aba **Start Page**, à essa altura do curso você já deve estar familiarizado com o Visual Studio.

9 - De um clique duplo sobre o **Button1** e digite o seguinte código:

```
Jogadores(0) = "Marcos"
```

Isso insere "Marcos" na primeira linha do Array, se colocássemos 6 ao invés de 0, por exemplo, seria na linha 7.

10 - Mude a propriedade **Text** do **Button2** para Visualizar.



11 - De um clique duplo sobre o **Button2** e digite o seguinte código:

```
ListBox1.Items.Add(Jogadores(0))
```

Isso insere na **ListBox1** o conteúdo da primeira linha do array Jogadores, se colocássemos 6, exibiria o conteúdo da linha 7, lembre-se que de que todo array começa com 0, obrigatoriamente.

Não se assuste com o código para inserir no **ListBox1**, se fosse em uma **TextBox** ou **Label** você poderia usar a seguinte linha de código:

```
TextBox1.Text = Jogadores(0)
```

Só que para inserir um item no **ListBox** precisamos usar seu método **Add**.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Agora que você já compreende como inserir e ler dados do **array** vamos avançar no exemplo.

12 - Volte para o código do procedimento do **Button1**.

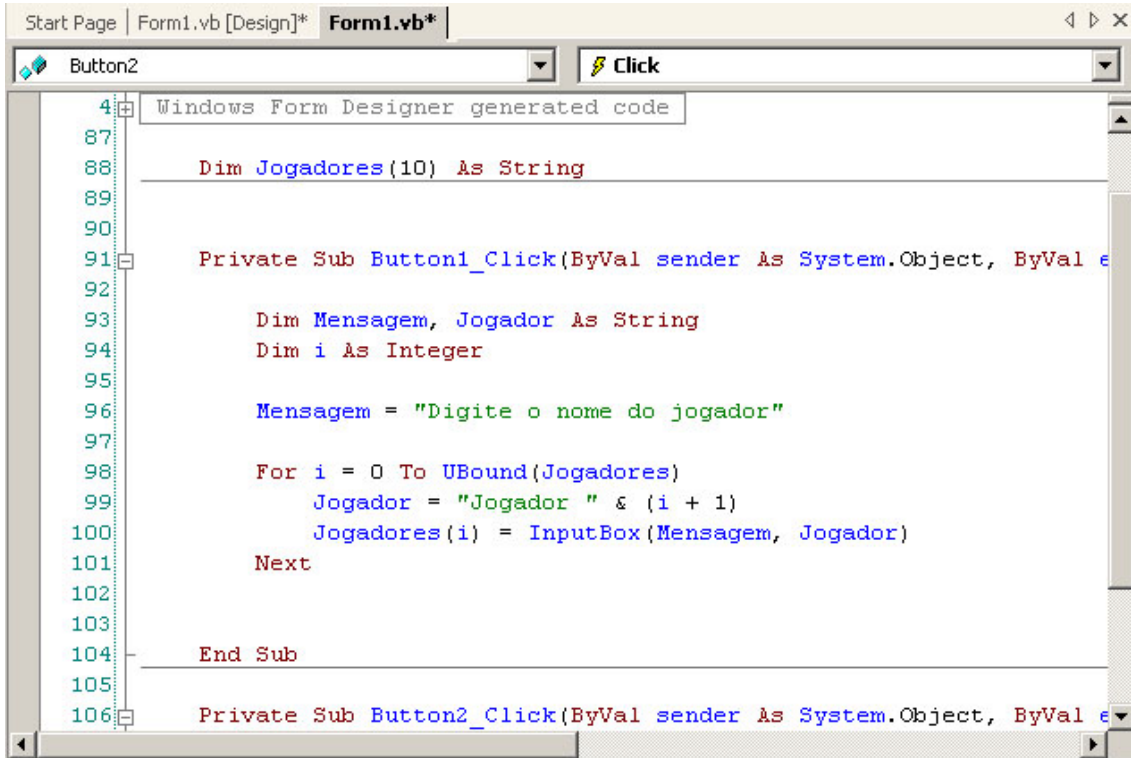
13 - Substitua o código pelo seguinte:

```
Dim Mensagem, Jogador As String
Dim i As Integer

Mensagem = "Digite o nome do jogador"

For i = 0 To UBound(Jogadores)
    Jogador = "Jogador " & (i + 1)
    Jogadores(i) = InputBox(Mensagem, Jogador)
Next
```

Vai ficar assim:



```
Start Page | Form1.vb [Design]* | Form1.vb*
Button2 Click
Windows Form Designer generated code
87
88 Dim Jogadores(10) As String
89
90
91 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
92
93     Dim Mensagem, Jogador As String
94     Dim i As Integer
95
96     Mensagem = "Digite o nome do jogador"
97
98     For i = 0 To UBound(Jogadores)
99         Jogador = "Jogador " & (i + 1)
100         Jogadores(i) = InputBox(Mensagem, Jogador)
101     Next
102
103
104 End Sub
105
106 Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
```

Com este código vamos solicitar ao usuário que digite o nome dos 11 jogadores. Para isso fazemos um loop que vai de 0 até o total de itens no array.

Como sabemos que o Array possui 11 linhas poderia ter usado o seguinte código, que seria mais simples:

```
For i = 0 To 11
```

No entanto a técnica que utilizamos é mais flexível por prever que o array pode aumentar.

Agora temos que mudar o código que exibe os dados também, para que possa exibir todos os dados cadastrados.

14 - Mude o código do **Button2** pelo seguinte:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

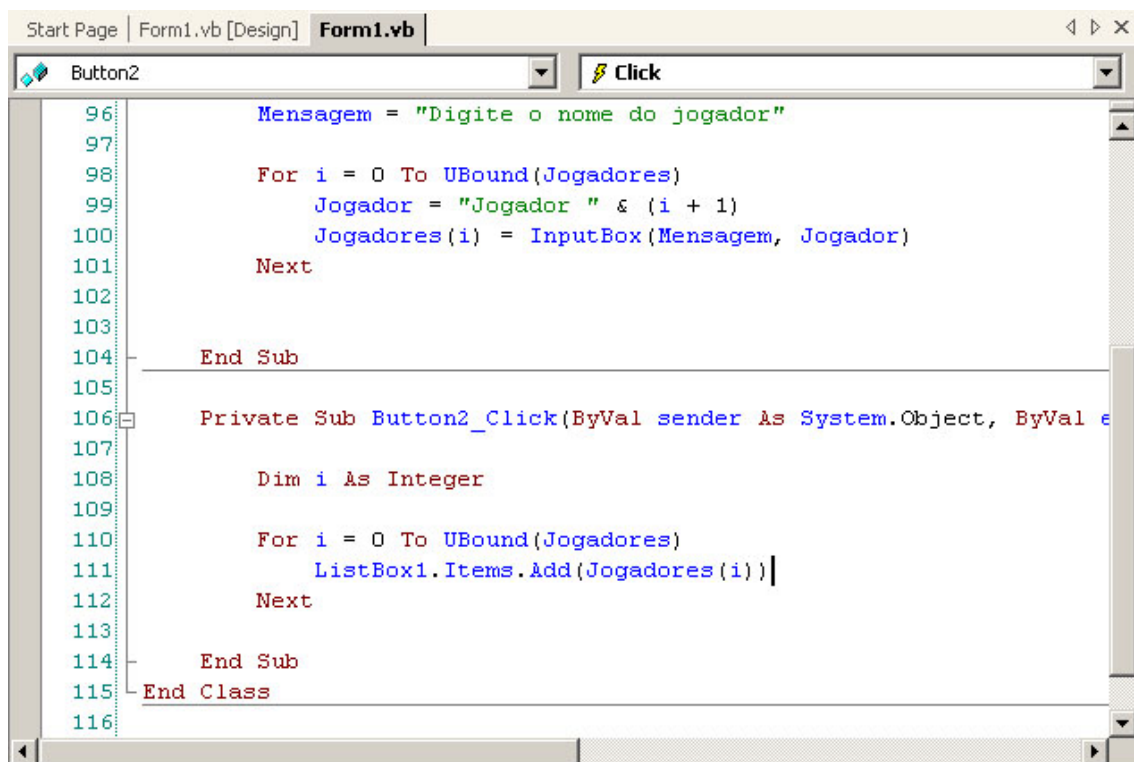
```
Dim i As Integer

For i = 0 To UBound(Jogadores)

    ListBox1.Items.Add(Jogadores(i))

Next
```

Vai ficar assim:



```
Start Page | Form1.vb [Design] | Form1.vb | Click
Button2
96     Mensagem = "Digite o nome do jogador"
97
98     For i = 0 To UBound(Jogadores)
99         Jogador = "Jogador " & (i + 1)
100        Jogadores(i) = InputBox(Mensagem, Jogador)
101    Next
102
103
104 End Sub
105
106 Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
107
108     Dim i As Integer
109
110     For i = 0 To UBound(Jogadores)
111         ListBox1.Items.Add(Jogadores(i))
112     Next
113
114 End Sub
115 End Class
116
```

Aqui usamos o loop For também para passar por cada item do Array e adicioná-lo no **ListBox1**.

Lembrando que a cada passada pelo For o i assume de uma linha do array até chegar ao final.

15 - Execute a aplicação.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

16 - Clique no botão Inserir Jogadores.



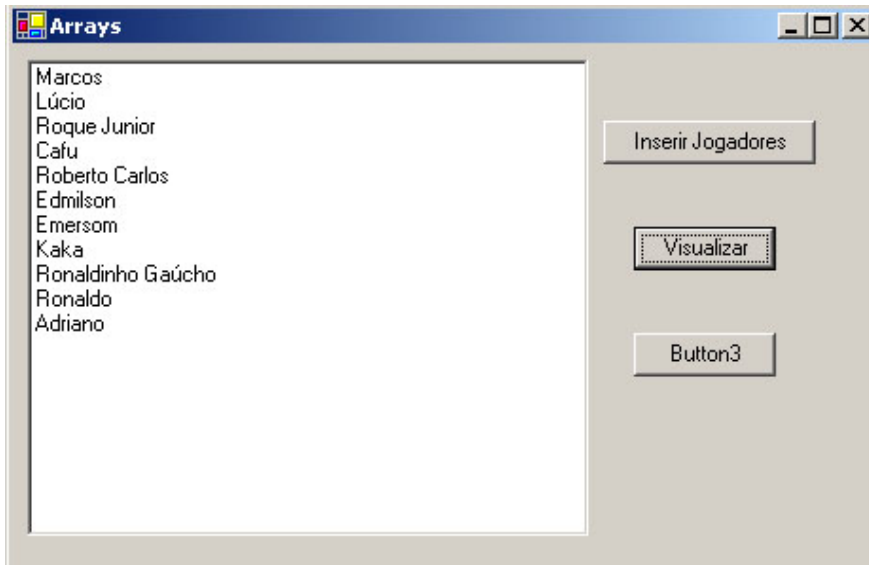
Aparece uma caixa pedindo para que digite o nome do Jogador 1.

17 - Digite um nome para o Jogador 1 e clique em OK.

Ele pede para inserir o nome do Jogador 2.

18 - Digite um nome para o Jogador 2 e clique em OK. Faça isso para os 11 jogadores.

19 - Clique no botão Visualizar.



O nome dos 11 jogadores é listado.

Como você pode ver, trabalhamos facilmente com os nomes dos 11 jogadores em nosso array, podemos modificar os dados do nosso array facilmente, bastando para isso indicar qual linha deve ser alterada e o novo dado, como no exemplo:

```
Jogadores(10) = "Robinho"
```

Um array pode conter mais de um coluna. Vamos entender como isso funciona implementando o código que vai armazenar o número de faltas de cada jogador.

Como um Array só pode conter um tipo de dado e nosso Array já é do tipo String, no nosso exemplo vamos adicionar o número de faltas como string mesmo, mas, a maneira correta seria criar um outro Array do tipo integer para armazenar as faltas.

20 - Altere o código que declara o Array para o seguinte:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
Dim Jogadores(10, 1) As String
```

Isso cria um Array com 11 linhas e 2 colunas. Um Array pode ter várias colunas.

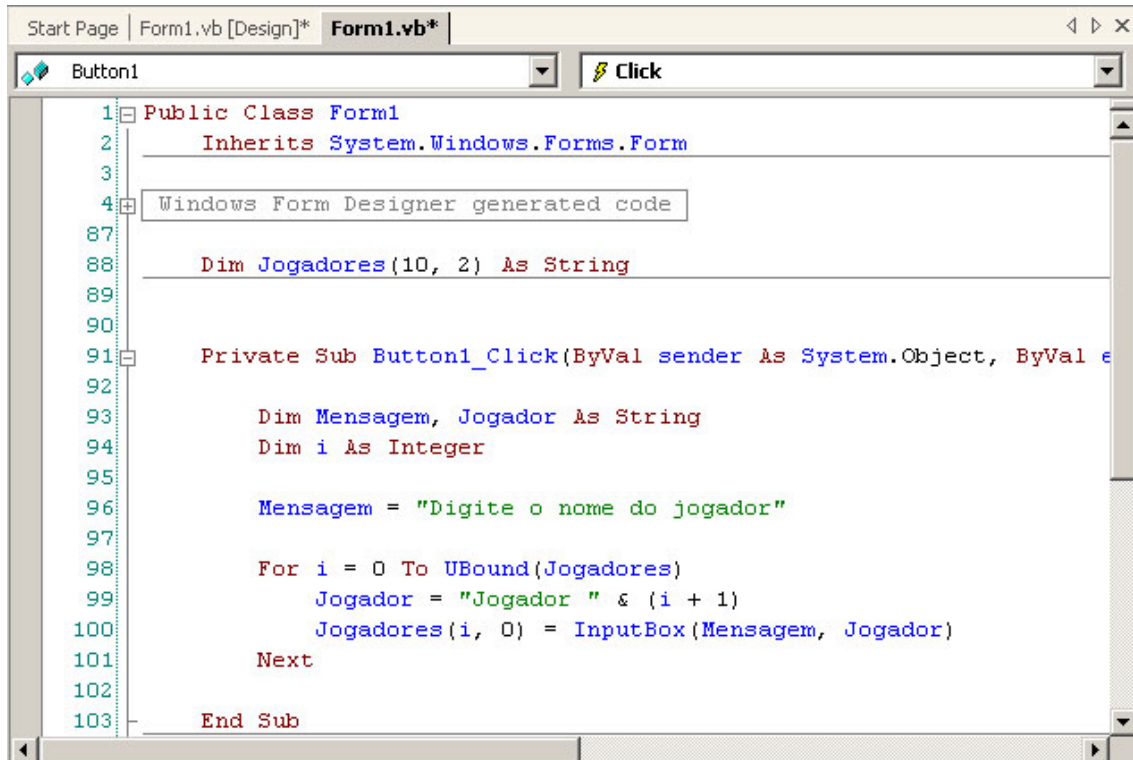
21 - Localize no procedimento do evento do Button1, o seguinte código:

```
Jogadores(i) = InputBox(Mensagem, Jogador)
```

22 - Mude-o para o seguinte:

```
Jogadores(i, 0) = InputBox(Mensagem, Jogador)
```

Isso apenas indica que vamos adicionar o nome na linha que o loop indicar, na coluna 1, lembrando que os Arrays sempre iniciam com índice 0.

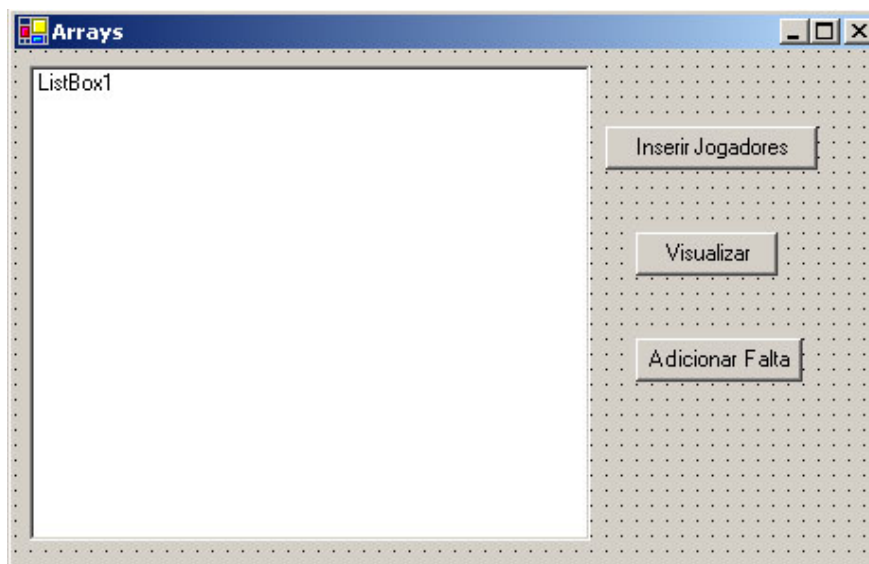


```

1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3
4     Windows Form Designer generated code
5
6
7
87
88     Dim Jogadores(10, 2) As String
89
90
91     Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
92
93         Dim Mensagem, Jogador As String
94         Dim i As Integer
95
96         Mensagem = "Digite o nome do jogador"
97
98         For i = 0 To UBound(Jogadores)
99             Jogador = "Jogador " & (i + 1)
100             Jogadores(i, 0) = InputBox(Mensagem, Jogador)
101         Next
102
103     End Sub

```

23 - Mude a propriedade Text do Button3 para Adicionar Faltas.



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

24 - Dê um clique duplo sobre o Button3 e digite o seguinte código:

```
Dim Mensagem As String

Dim i As Integer

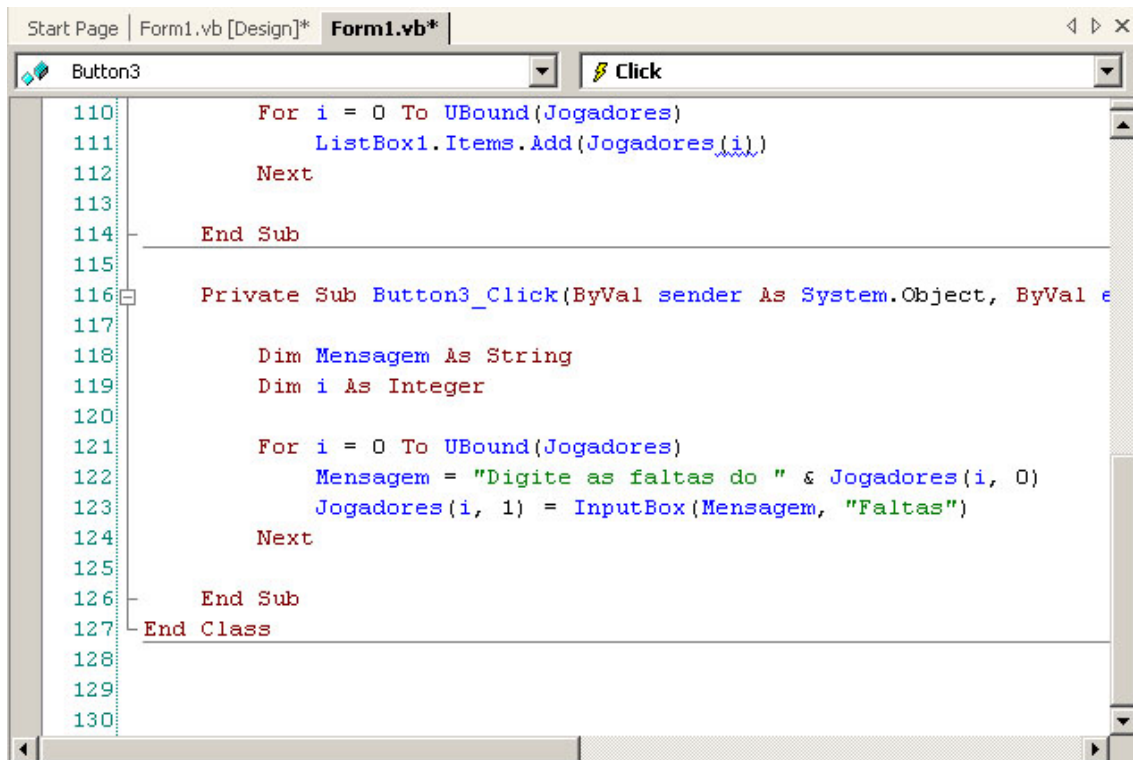
For i = 0 To UBound(Jogadores)

    Mensagem = "Digite as faltas do " & Jogadores(i, 0)

    Jogadores(i, 1) = InputBox(Mensagem, "Faltas")

Next
```

Vai ficar assim:



Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Isso é semelhante ao código do Button1, com algumas modificações, como as mensagens que serão inseridas, a cada loop vamos exibir o nome do jogador que esta recebendo os dados das faltas, a linha de código responsável por isso pe a seguinte:

```
Mensagem = "Digite as faltas do " & Jogadores(i, 0)
```

Note que agora atribuímos os valores à segunda coluna, como mostra o código:

```
Jogadores(i, 1) = InputBox(Mensagem, "Faltas")
```

Agora precisamos mudar o código do button2, que exibe os dados.

25 - Mude o código do Button2 para o seguinte:

```
Dim i As Integer

For i = 0 To UBound(Jogadores)

    ListBox1.Items.Add(Jogadores(i, 0) & " - Total de faltas: " &
Jogadores(i, 1))

Next
```

O que fizemos foi concatenar os dados da coluna 1 com os da 2.

26 - Execute sua aplicação:

Autor: Herbert Moroni Cavallari da Costa Gois

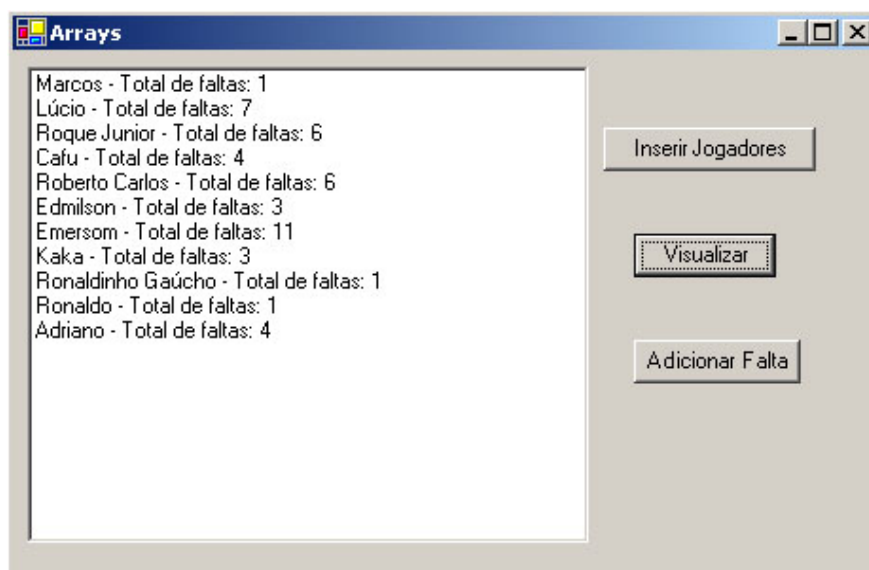
Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

27 - Clique no botão Inserir Jogadore e adicione o nome dos 11 jogadores.

28 - Clique no botão Adicionar Falta e adicione um número de faltas para cada jogador.

29 - Clique no botão Visualizar.



Como você sabe bem, tanto em um jogo de futebol, como em muitas outras situações, nós não sabemos com certeza o número de linhas que iremos precisar no nosso Array. Em um jogo de futebol podem haver até 3 substituições, o que aumentaria em 3 o número de linhas no nosso Array. No entanto, não é certeza que o treinador fará as 3 substituições que lhe cabem por direito.

Para esse tipo de situações, o Visual Basic oferece um Array especial, conhecido como Array Dinâmico. Um Array Dinâmico pode ser redimensionado durante a execução do programa.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

ARRAY DINÂMICO

Para se criar um Array Dinâmico você precisa de alguns cuidados extras.

1 - Declare o Array Dinâmico sem especificar o número de elementos, como no seguinte exemplo:

```
Dim Jogadores() As String
```

2 - Antes de inserir dados no seu Array Dinâmico você precisa saber qual será o tamanho dele, você deve implementar código para isso, como por exemplo, uma caixa de diálogo que pergunta ao usuário quantos jogadores participaram da partida.

3 - Use o comando Redim para redimensionar seu Array em tempo de execução, usando para isso a informação do número de elementos que descobriu.

Exemplo:

```
ReDim Jogadores(13)
```

Você não pode usar o Redim para mudar o tipo de dados do seu array, isso não é possível.

Atenção quando usar o Redim. Lembre-se que o Array começa com 0.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Agora você já pode usar normalmente seu Array.

Vamos implementar um Array Dinâmico no nosso exemplo.

30 - Mude a declaração do Array Jogadores para o seguinte:

```
Dim Jogadores(,) As String
```

Usamos a virgula porque nosso Array possui duas colunas. Perceba que não especificamos o número de linhas, nem o número de colunas. Com isso atendemos a regra 1 sobre os Arrays Dinâmicos.

31 - Para atender a regra 2, descobrindo quantas linhas serão necessárias vamos implementar no nosso programa mais uma caixa de diálogo que pergunta quantos jogadores participaram do jogo, para isso altere o código do Button1 para o seguinte:

```
Dim Mensagem, Jogador As String  
Dim i As Integer  
Dim NumeroJogadores As Integer  
  
NumeroJogadores = InputBox("Digite o número de jogadores que  
participou da partida", "Jogadores")  
  
ReDim Jogadores(NumeroJogadores - 1, 1)  
  
Mensagem = "Digite o nome do jogador"
```

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
For i = 0 To UBound(Jogadores)

    Jogador = "Jogador " & (i + 1)

    Jogadores(i, 0) = InputBox(Mensagem, Jogador)

Next
```

Tudo que fizemos foi criar uma variável do tipo integer que armazena o número de jogadores que participaram da partida. Adicionamos a essa variável o número digitado na caixa de diálogo através da seguinte linha de código:

```
NumeroJogadores = InputBox("Digite o número de jogadores que  
participou da partida", "Jogadores")
```

Neste ponto sabemos o número de jogadores então usamos o Redim para redimensionar o array. Perceba que passamos como número de linhas o número de jogadores menos 1, isso porque a primeira linha do array é sempre 0, se não fizessemos isso e o usuário digitasse 2 por exemplo, teríamos 3 linhas no array, a 0, 1, 2. Como sabemos que o número de colunas desejado é 2, colocamos 1 no Redim.

```
ReDim Jogadores(NumeroJogadores - 1, 1)
```

32 - Execute sua aplicação.

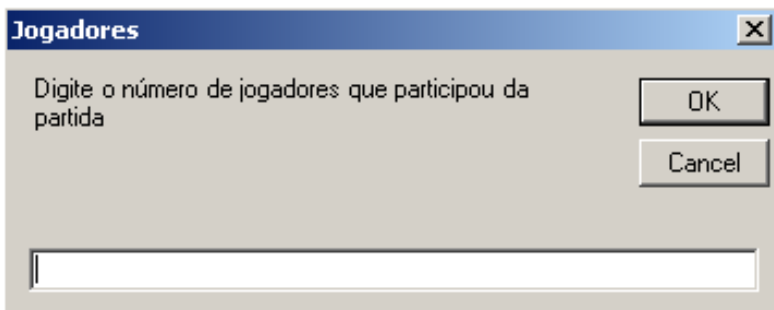
33 - Clique em Inserir Jogadores.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Agora, antes de exibir a caixa que pergunta o nome dos jogadores, aparece a seguinte caixa perguntando a quantidade. Digite a quantidade e clique em OK.



O restante da aplicação funciona normalmente, só que agora com o número de jogadores informado.

Uma observação importante, é que quando você executa o **Redim** o que estiver armazenado no seu array é perdido. Dependendo das necessidades do seu programa isso pode lhe trazer muitas dores de cabeça. Para solucionar esse problema, você pode usar o **Redim Preserve**, que mantém os dados do array mesmo depois do seu redimensionamento.

O seguinte exemplo ilustra o uso do **Redim Preserve**:

```
Dim Jogadores(10) As String
```

Declaramos um array que pode armazenar 11 jogadores, no entanto agora precisamos armazenar 13, só que sem perder o nome dos jogadores que já estão armazenados. Para isso usamos o Redim Preserve como se segue:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

```
ReDim Preserve Jogadores(12)
```

O problema com o **Redim Preserve** é que se temos um Array com duas dimensões, como o seguinte exemplo:

```
Dim Jogadores(10,1) As String
```

Só podemos redimensionar o tamanho da última dimensão, ou seja, só podemos mudar o número de colunas.

Para finalizar o assunto Arrays, vamos apenas entender o que é **LBound** e **UBound**. Lembre-se que durante nossos exemplos usamos o **UBound**, junto com o Loop For, como se segue:

```
For i = 0 To UBound(Jogadores)
```

O **Ubound** como você pode perceber sempre retorna o número máximo de registros que podem ser armazenados no Array iniciando em 0. O **LBound** retorna o número mínimo, geralmente 0. A sintaxe dos dois são:

UBound (*NomedoArray*)

LBound (*NomedoArray*)

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Lembre-se que Arrays são usados para gerenciar grande quantidade de informação, como variáveis, você precisa declarar seus arrays antes de usá-los. Os arrays também respeitam as mesmas regras de escopo que as variáveis, ou seja, se declaradas dentro de procedimentos, ficam disponíveis apenas para o procedimento em questão.

10.12 – Procedimentos e funções em VB.NET

Procedimentos em VB.NET são conhecidos como Sub.

Procedimentos Sub são tipicamente usados para pegar informações do usuário, exibir ou imprimir informações e manipular propriedades associadas a condições.

Sua sintaxe é a seguinte:

```
Sub NomedoProcedimento(argumentos)  
    Código  
End Sub
```

NomedoProcedimento é o nome que você quer dar a **Sub** que esta criando. Procure dar sempre um nome que ajude a identificar a utilidade da sua Sub.

Argumentos são uma lista de argumentos opcionais - separados por vírgula se forem mais de um - que serão usados na sua Sub. Cada argumento precisa ter um tipo específico.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

Código é o bloco de código que faz o trabalho do procedimento.

Funções em VB.NET são declaradas como **Function**.

A Function (função) é muito parecida com a Sub, as regras aplicadas para as Subs se aplicam aqui, principalmente as que se referem ao escopo e aos argumentos. A diferença é que a Function vai sempre retornar um valor.

A sintaxe da Function é a seguinte:

```
Function NomedaFunção(argumentos) as Type  
    Código  
    Return valor  
End Function
```

NomedaFunção é o nome que você quer dar a função que esta criando. Como com as Subs, procure dar sempre um nome que ajude a identificar a utilidade da sua função. Não esqueça do Type que é o tipo de dados que sua função vai retornar.

Argumentos são uma lista de argumentos opcionais - separados por vírgula se forem mais de um - que serão usados, como nas Subs.

Código é o bloco de código que faz o trabalho da função.

Return é a instrução que permite retornar o valor. Após o **Return** ser executado a função é finalizada.

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

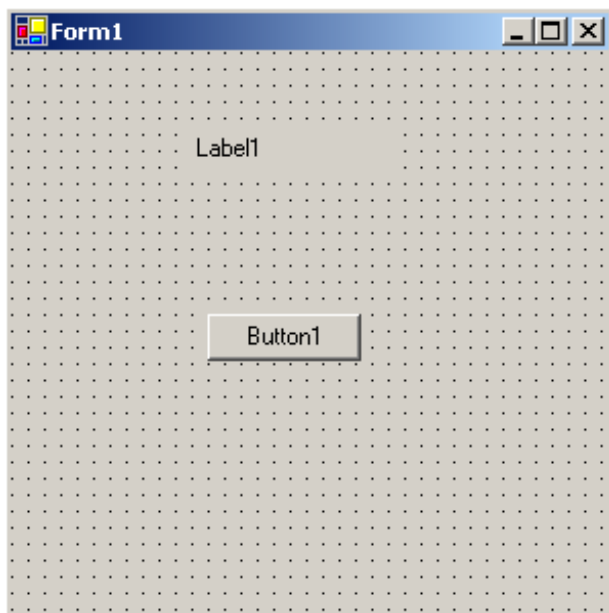
Para entender melhor vamos fazer um exemplo bem simples.

1 - Crie um projeto no Visual Studio do tipo **Windows Application**, chamado **ProdFunc.**

2 - Arraste para o **Form1** os seguintes controles:

- 1 Button
- 1 TextBox

3 - Organize-os como a imagem:



4 - Na classe do **Form1**, crie o seguinte procedimento Sub:

Autor: Herbert Moroni Cavallari da Costa Gois

Site: www.juliobattisti.com.br

Confira também o curso: "Programando com VB.NET" e "Programando com C# e o Visual Studio .NET 2005"

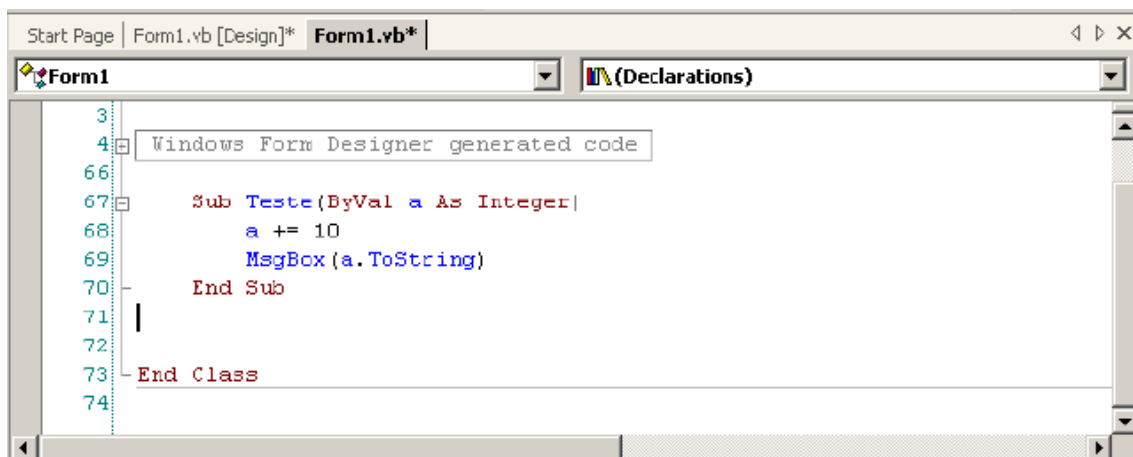
```
Sub Teste(ByVal a As Integer)

    a += 10

    MsgBox(a.ToString)

End Sub
```

Nossa sub simplesmente soma 10 ao valor que foi passado a ela como parâmetro e mostra uma caixa de diálogo com o valor. Deve ficar assim no painel de código:



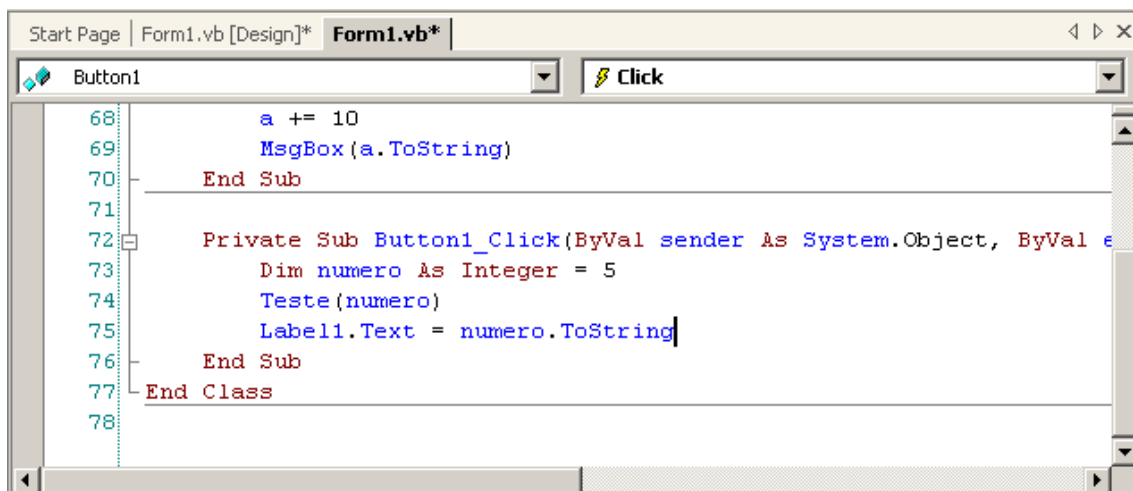
5 - Agora no modo Design, de um clique duplo sobre o **Button1** e digite o seguinte código:

```
Dim numero As Integer = 5

Teste(numero)

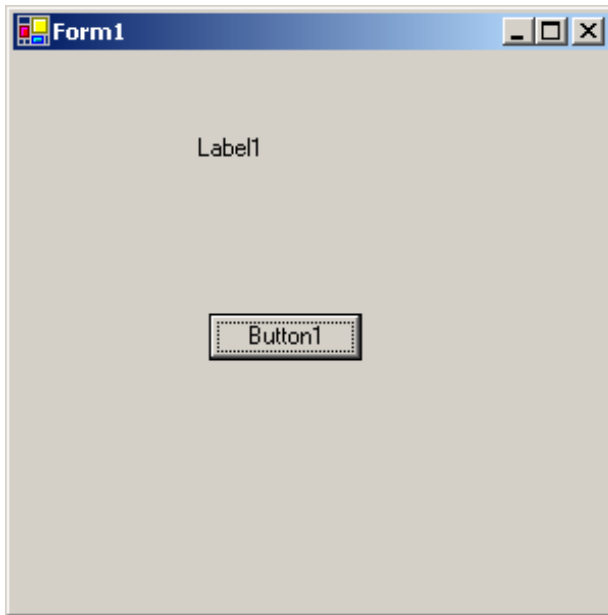
Label1.Text = numero.ToString
```

A primeira linha declara uma variável chamada numero do tipo Integer e atribui 5 a ela. A segunda faz a chamada ao procedimento Teste que criamos. A terceira linha simplesmente pega o conteúdo da variável numero e exibe no Label1. Deve ficar assim no painel de código:



```
68      a += 10
69      MsgBox(a.ToString)
70  End Sub
71
72  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
73      Dim numero As Integer = 5
74      Teste(numero)
75      Label1.Text = numero.ToString
76  End Sub
77 End Class
78
```

6 - Execute a aplicação.

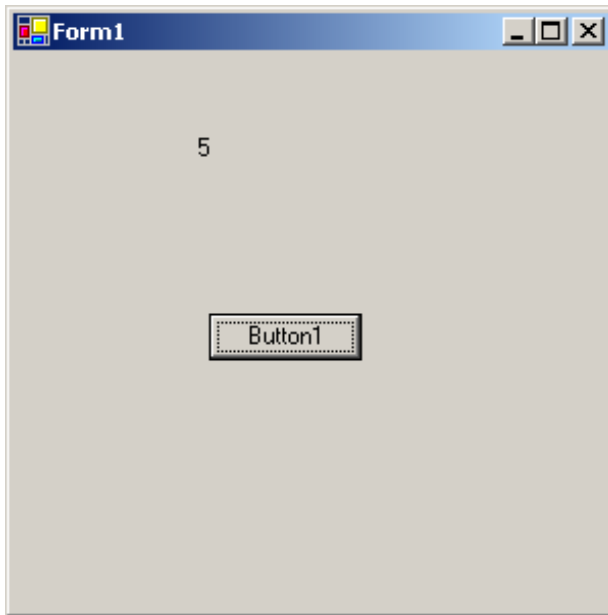


7 - Clique no **Button1**.

É exibida a caixa de diálogo da Sub com o valor.

Como passamos a variável numero como parâmetro e o conteúdo dela é 5, esse valor somado a 10 na Sub é o valor exibido pela caixa de diálogo.

8 - Clique em OK.



Depois que ele executa a Sub ele pega o valor da variável e exibe na **Label1**, percebe que o valor permanece sendo 5, ou seja ele efetuou a soma na Sub mais não alterou o valor da variável número.

9 - Pare a execução do programa.