

Informática para Engenharia



Visual C# 2008

Prof. José Marcelo Traina Chacon

Sumário

Introdução	01
Iniciando o Visual C#	02
Criando uma Aplicação	04
Adicionando Componentes	06
Formulários	08
Rótulos	10
Caixa de Texto	12
Caixa de Texto com Máscara	14
Botão de Comando	15
Botões de opção	17
Caixa de verificação	17
Caixa de Lista	18
Caixa Combinada	19
Imagem	22
Caixa de grupo	25
Itens Fundamentais	26
Operadores	33
Estruturas Condicionais	38
Estruturas de Repetição	43
Funções	48
Matrizes	50
Estruturas	56
Bibliografia	59

Introdução

Introdução

O Visual C# é um ambiente visual, orientado a objetos que tem por finalidade desenvolver aplicações rapidamente para o Windows XP e Vista. Estas aplicações podem ser de propósitos gerais. Usando o Visual C#, você pode criar eficientes aplicações Windows com o mínimo de codificação manual.

O Visual C# disponibiliza uma extensa biblioteca de componentes reutilizáveis e um ambiente de ferramentas RAD (Desenvolvimento de Aplicações Rápida).

Quando você inicia o Visual C#, você é imediatamente colocado com o ambiente de programação visual. É este ambiente que disponibiliza todas as ferramentas que você necessita para criar, desenvolver, testar, e debugar suas aplicações.

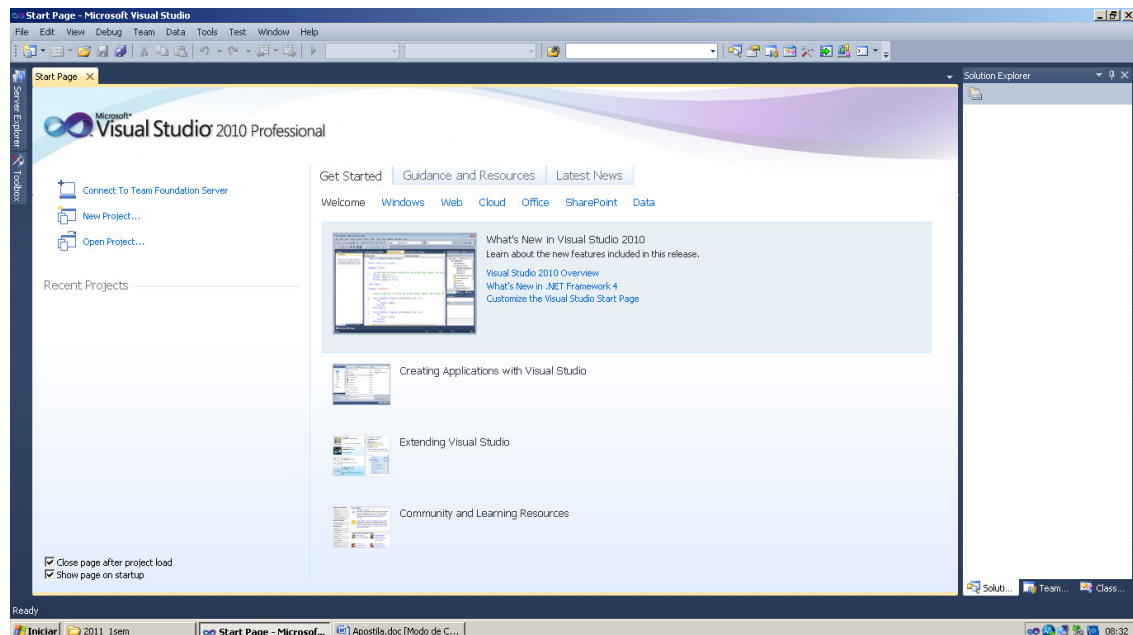
Iniciando o Visual C#

Iniciando Visual C#

A melhor maneira de aprender é rodando o Visual C#. Você poderá rodar o C# da mesma maneira que roda qualquer aplicação Windows, como segue abaixo:

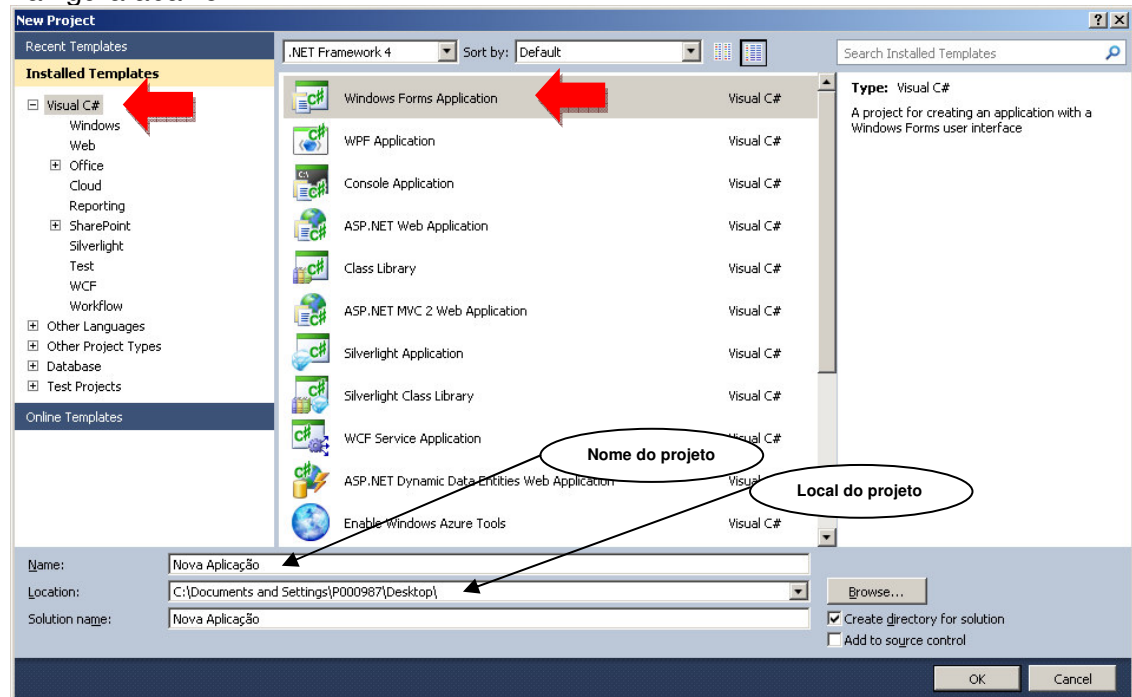
- Dando um duplo-clique no ícone do Visual C#
- Usando o Windows Explorer. Localize e dê um duplo-clique no arquivo DEVENV.EXE
- Escolha Executar a partir do menu Iniciar, e especifique o path para o Microsoft Visual Studio 2010, Microsoft Visual Studio 2010

A figura abaixo mostra como o Visual Studio 2008 irá aparecer quando for rodado pela primeira vez:



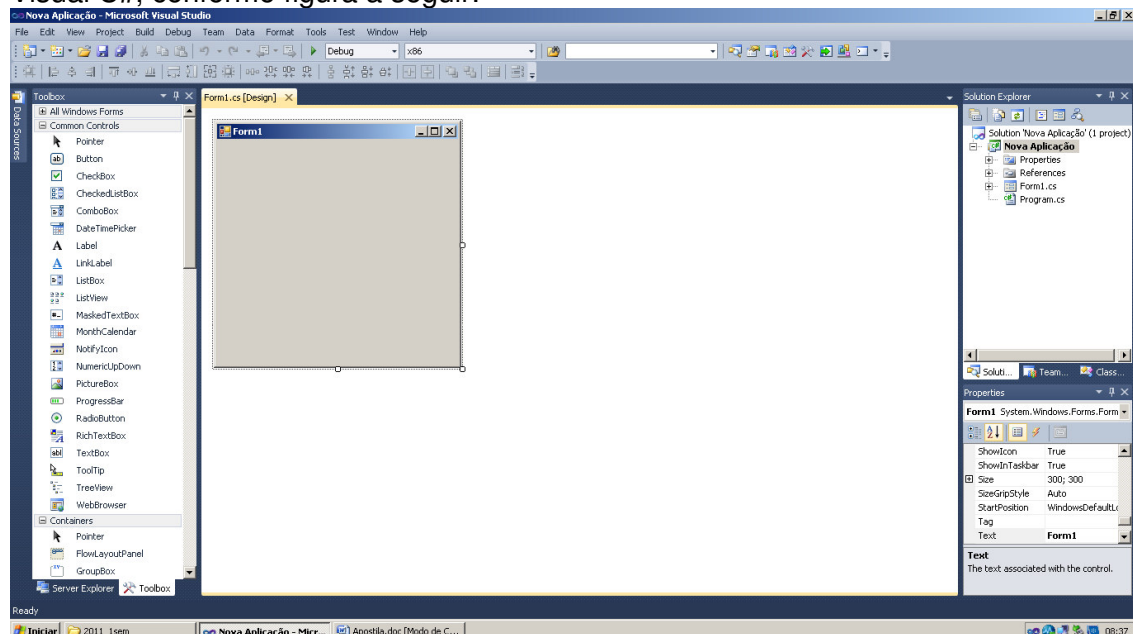
Para criarmos uma aplicação no Visual C# podemos clicar na opção **New Project** contida na tela **Start Page**. A seguir deveremos escolher o programa C#, o Windows

Forms Application, o nome e o local do projeto a ser criado, conforme é apresentado na figura abaixo:



Obs: Escolha o local a ser criado o projeto com o botão Browse. Ex: Desktop.

Estamos prontos para desenvolver nossas aplicações visuais na tela do programa Visual C#, conforme figura a seguir:



Obs: Se não aparecerem as guias ToolBox e Solution Explorer clicar no menu view ToolBox e view Solution Explorer. Para aparecer a guia Properties clicar em form1.cs e no botão de atalho Properties.

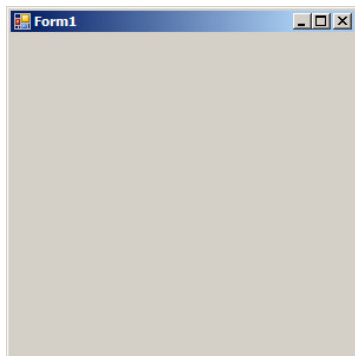
Criando uma Aplicação

Criando uma Aplicação

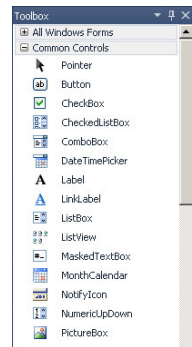
O Visual Studio C# imediatamente apresenta as ferramentas necessárias para que você inicie a criação da aplicação.

- Uma janela em branco, conhecida como formulário, na qual você criará a interface do usuário
- Uma extensiva biblioteca de objetos, chamada de toolbox
- Uma maneira fácil de alterar as propriedades de um objeto e eventos é utilizando a janela de propriedades
- Acesso direto ao código do programa utilizando a tecla F7.

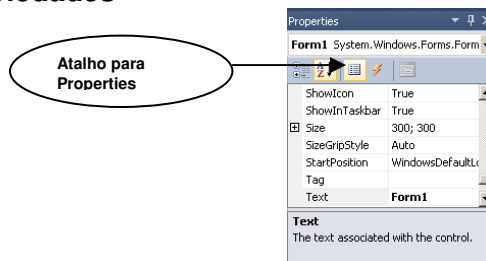
Formulário



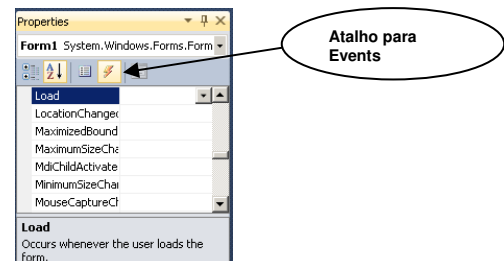
ToolBox



Propriedades

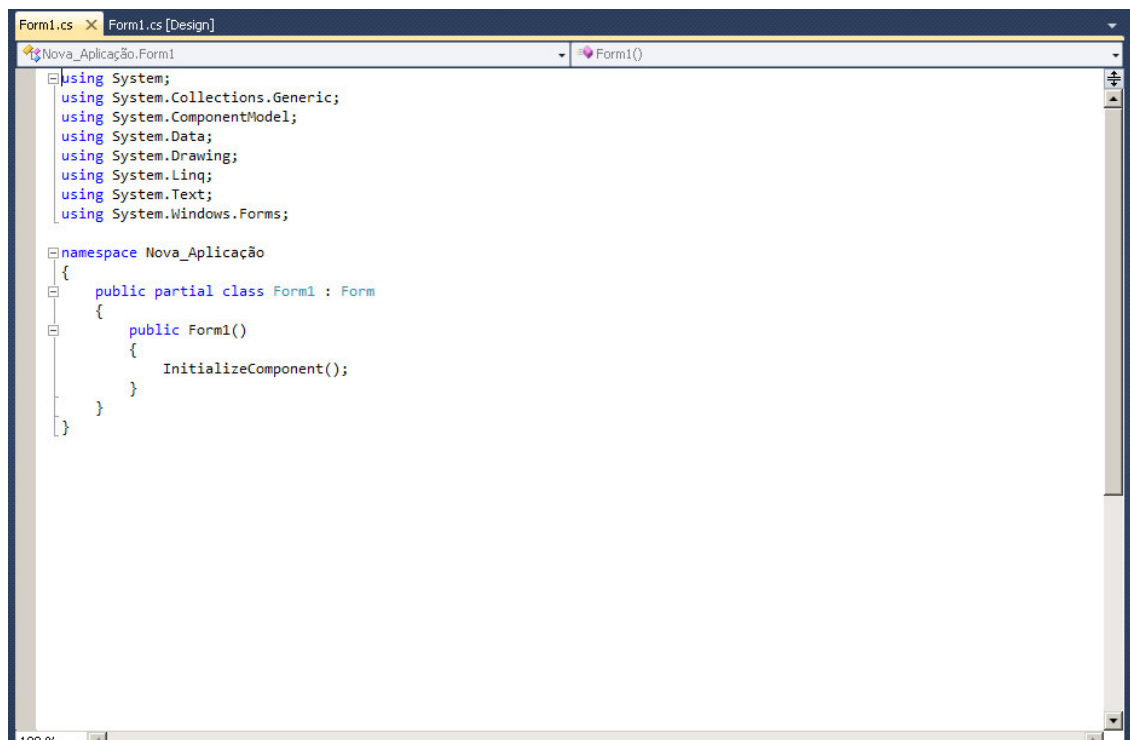


Propriedades



Eventos

Editor de Códigos

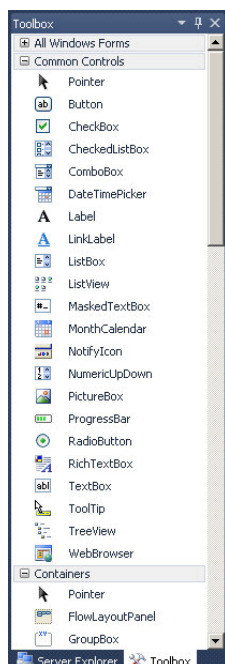


Você pode usar o Visual C# para criar aplicações para Windows XP ou Vista. Desde aplicações de propósitos gerais, até sofisticados programas de acesso a banco de dados.

Adicionando Componentes

Adicionando Componentes

Componentes são elementos que você usa para construir suas aplicações em Visual C#. Eles incluem todas as partes visíveis de sua aplicação tais como, box de dialogo e botões; bem como as partes não visíveis enquanto a aplicação estiver rodando, como os Timers.

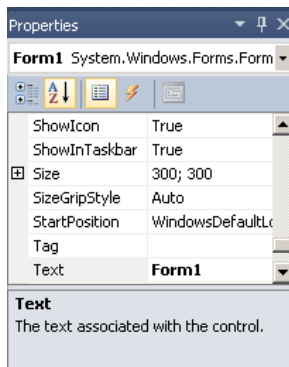


Muitos componentes visuais são disponibilizados no ambiente de desenvolvimento no TollBox. Você seleciona os componentes da paleta e arrasta-o no formulário que você esta desenvolvendo. Uma vez que os componentes estejam no formulário você poderá ajustar as suas propriedades como, posição, tamanho entre outras.

O Visual C# agrupa os componentes, em diferentes grupos, de acordo com a sua funcionalidade.

Alterando o comportamento do componente

Você pode facilmente personalizar a aparência e o comportamento de sua aplicação usando a janela de propriedades. Quando um componente tem o foco no formulário, suas propriedades e eventos são mostrados na janela propriedade.



Usando a janela de Propriedades você ajustará os valores iniciais, dos componentes, com o qual sua aplicação irá iniciar. Você usará a página Eventos para navegar através dos eventos que o componente pode ter.

Dando um duplo clique em um evento em particular, o Visual C# irá gerar o código para este evento. Você apenas deverá completar o código de acordo com a ação desejada.

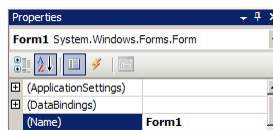
Formulários

Formulários (Forms) são objetos especializados no qual você coloca componentes. Forms geralmente se parecem como janelas e caixas de diálogo em uma aplicação que esta rodando. Você interage com o forms lendo e fixando suas propriedades e respondendo aos seus eventos.

Segue uma lista das principais propriedades de um formulário.

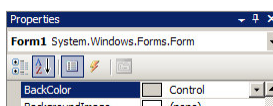
Propriedades – Características de um componente

Name – Nome do formulário

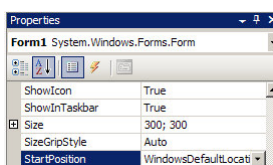


Normalmente em nossas aplicações não alteraremos os nomes dos objetos inseridos no programa. Portanto o nome que utilizaremos para o Formulário será Form1.

BackColor – Determina qual a cor de fundo do formulário

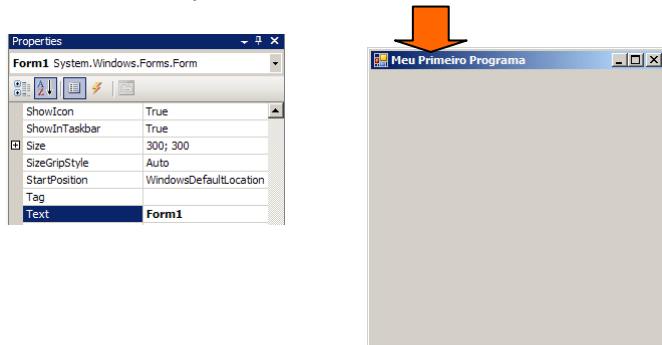


StartPosition – Determina o local que o formulário irá aparecer quando rodarmos o programa.



- WindowsDefaultLocation – Local padrão do Windows (Canto superior esquerdo)
- CenterScreen – Centro da tela do computador (o programa aparecerá centralizado na tela do computador).

Text – Texto que será mostrado na barra de título do formulário



Rótulo

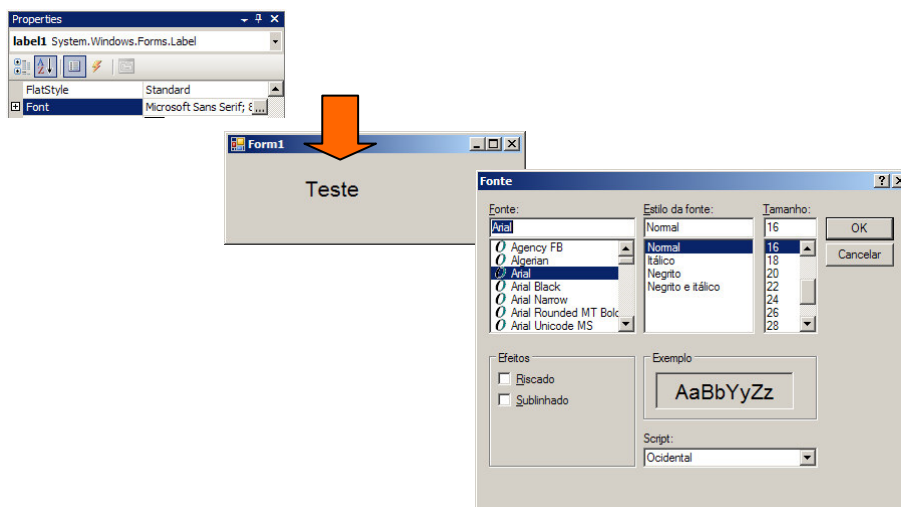
Rótulos (label) são objetos no qual você pode apresentar uma informação textual.

A grande parte das propriedades para o objeto rótulo é comum as propriedades explicadas no objeto form.

Segue uma lista das principais propriedades de um rótulo.

Propriedades – Características de um componente

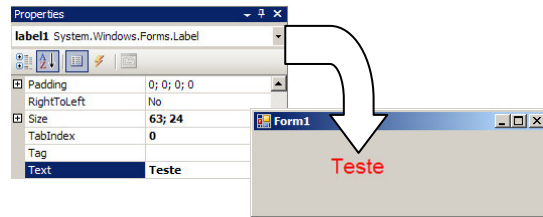
Font – Define características de padrão de letra do texto que será escrito na tela.



ForeColor – Altera a cor da letra do objeto label



Text – Define o texto que será escrito na tela.



Exemplo de um projeto com rótulo

Programa Hello World!



Objeto	Propriedades	Valor
form1	Text	Meu primeiro programa em Visual C#
	StartPosition	CenterScreen
label1	Text	Hello World!!!
	Font	Arial, 20, Negrito

Caixa de Texto

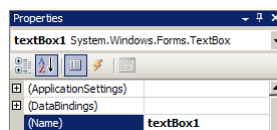
Caixa de Texto (TextBox) são objetos no qual você pode introduzir informação textual. Podemos compará-lo a um comando de entrada.

A grande parte das propriedades para o objeto caixa de texto é comum as propriedades explicadas anteriormente.

Segue uma lista das principais propriedades de um caixa de texto.

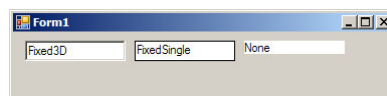
Propriedades – Características de um componente

Name – Define o nome da caixa de texto no programa



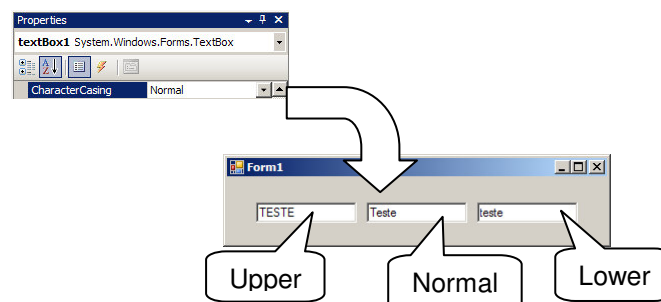
BorderStyle – Define o estilo da borda da caixa de texto

- Fixed3D – Borda em 3D
- FixedSingle – Borda simples
- None – Sem Borda



CharacterCasing – Define o tipo de caracteres para a caixa de texto.

- Lower – Letras minúsculas
- Normal – Letras minúsculas e maiúsculas
- Upper – Letras maiúsculas

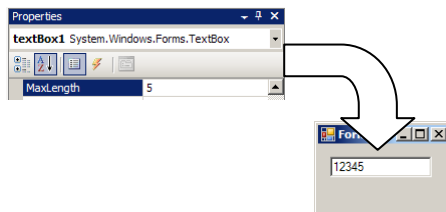


Enabled – Ativa ou desativa uma caixa de texto.

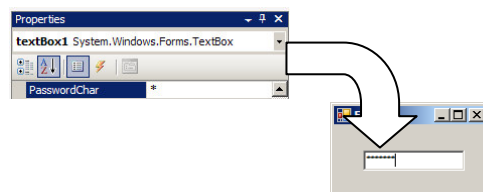
- true – Habilita uma caixa de texto
- false – Desabilita uma caixa de texto – não aceitará digitação de valores e aparecerá levemente acinzentado



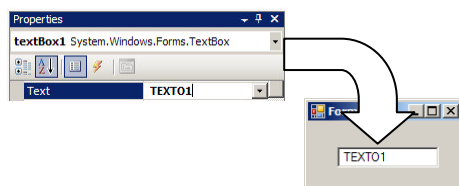
MaxLength – Limita o número de caracteres a ser digitado



PasswordChar – Campo Senha - Define o caracter a ser mostrado



Text – Texto que será exibido na caixa de textos



Caixa de Texto com Máscara

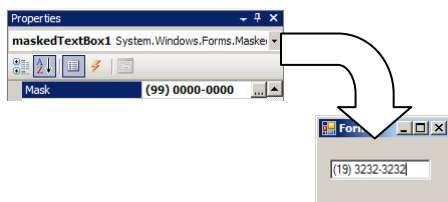
Caixa de Texto com Máscara (MaskedTextBox) são objetos no qual você pode introduzir informação textual com formatação de texto desejada. Podemos compará-lo a um comando de entrada formatado.

A grande parte das propriedades para o objeto caixa de texto é comum as propriedades explicadas anteriormente na caixa de texto.

Segue uma lista das principais propriedades de um caixa de texto com máscara.

Propriedades – Características de um componente

Mask – Define a máscara a ser utilizada na caixa de texto



Botões de Comando

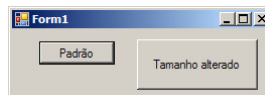
Botões de Comando (Button) são componentes geralmente usados em programas para disparar um processo quando acionados pelo clique do mouse.

A grande parte das propriedades para o objeto Botão de comando é comum as propriedades explicadas anteriormente.

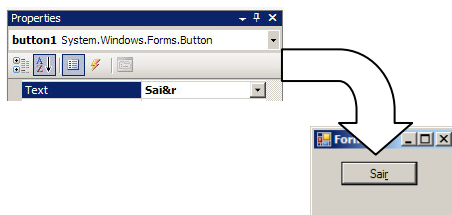
Segue uma lista das principais propriedades de um Botão de comando.

Propriedades – Características de um componente

Size – Permite alterar o tamanho do objeto Button, o primeiro número representa a largura e o segundo número a altura do objeto.



Text – O texto que será escrito no botão. Se acrescentarmos o caractere & antes de uma letra esta se tornará uma tecla de atalho do botão



Programa Temperatura

Objeto	Propriedades	Valor
form1	Text StartPosition	Conversão de Temperaturas CenterScreen
label1	Text	Temperatura em Celsius:
label2	Text	Temperatura em Kelvin:
label3	Text	Temperatura em Fahrenheit:
textBox1	MaxLength	3
textBox2	MaxLength	3
textBox3	Enabled	False
textBox3	MaxLength	3
textBox3	Enabled	False
button1	Text	&Converter
button2	Text	Sai&r

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Temperaturas
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                //Declaração de Variáveis
                int tcel, tkel, tfah;
                string entrada;

                //Faz a leitura da temp. Celsius
                entrada = textBox1.Text;
                //Converte de Texto para Int
                tcel = int.Parse(entrada);

                //Calcula as temperaturas
                tkel = tcel + 273;
                tfah = (((tcel * 9) / 5) + 32);

                //Mostra na tela os valores calculados
                textBox2.Text = tkel.ToString();
                textBox3.Text = tfah.ToString();
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO ***** ",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                textBox1.Text = "";
                textBox2.Text = "";
                textBox3.Text = "";
                textBox1.Focus();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Botões de Opção e Caixas de verificação

Botões de Opção (radioButton) – são componentes que são apresentados ao usuário em grupo para a escolha de apenas uma opção. As opções não escolhidas ficam desabilitadas.

Caixa de Verificação (checkBox) – são componentes que são apresentados ao usuário em grupo para a escolha de uma ou mais opções.

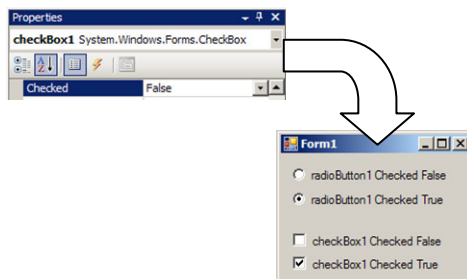
A grande parte das propriedades para os objetos Botão de opção e Caixa de verificação é comum as propriedades explicadas anteriormente.

Segue uma lista das principais propriedades de um Botão de opção e Caixa de verificação.

Propriedades – Características de um componente

Checked – Indica se o componente está ou não selecionado.

- true – indica selecionado
- false – indica não selecionado



Caixas de listas

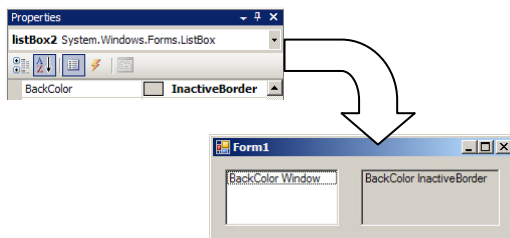
Caixa de Lista (listBox) – são componentes que dão ao usuário a opção de escolhas, esta escolhas são feitas através do mouse.

A grande parte das propriedades para o objeto Caixa de Lista é comum as propriedades explicadas anteriormente.

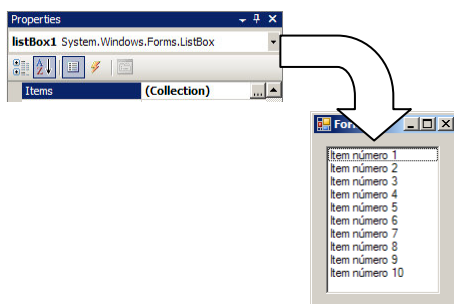
Segue uma lista das principais propriedades de um Caixa de Lista.

Propriedades – Características de um componente

- **BackColor** – Define uma cor para o fundo da caixa de lista.



- **Items** – Define uma lista inicial de opções na caixa de lista.



Caixa Combinada

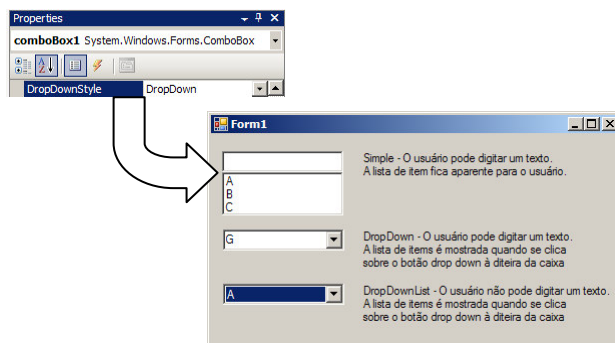
Caixa combinada (comboBox) – são componentes que dão ao usuário a opção de escolhas, esta escolhas são feitas através do mouse. Combina caixa de texto com Caixa de lista

A grande parte das propriedades para o objeto Caixa Combinada é comum as propriedades explicadas anteriormente.

Segue uma lista das principais propriedades de um Caixa Combinada.

Propriedades – Características de um componente

- **DropDownStyle** – Determina o tipo de caixa combinada e seu comportamento



Programa Lista de Usuários

Objeto	Propriedades	Valor
form1	Text StartPosition	Lista de usuários CenterScreen
label1	Text	Usuários
textBox1	Text	-
listBox1	-	-
radioButton1	Text	
radioButton2	Text	
comboBox1	Items	Windows, Office, AutoCad, Visual Studio, PhotoShop
button1	Text Enabled	&Adicionar False
button2	Text	&Remover
button3	Text	&Sair

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Lista_de_usuários
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            if (textBox1.Text != "")
                button1.Enabled = true;
            else
                button1.Enabled = false;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            /*-----
               Adicionar Usuário
            -----*/
            //Verifica o tipo de Usuário
            if (radioButton1.Checked)
                textBox1.Text =
                    textBox1.Text + " M " + comboBox1.Text;
            else
                if (radioButton2.Checked)
                    textBox1.Text = textBox1.Text + " N " + comboBox1.Text;
            //Adiciona o nome da Caixa de Texto
            listBox1.Items.Add(textBox1.Text);
            textBox1.Text = "";
            radioButton1.Checked = false;
            radioButton2.Checked = false;
            comboBox1.Text = "";
        }

        private void button2_Click(object sender, EventArgs e)
        {
            /*-----
               Remove Usuários
            -----*/
            int num;
            //Verifica o valor corrente da Caixa de Lista

```

```

        num = listBox1.SelectedIndex;
        //Remove o Usuário se existir
        if (num != -1)
            listBox1.Items.RemoveAt(num);
        textBox1.Text = "";
        radioButton1.Checked = false;
        radioButton2.Checked = false;
        comboBox1.Text = "";
    }

    private void button3_Click(object sender, EventArgs e)
    {
        string texto = " Deseja sair do Programa Lista de usuários?";
        string titulo = "+++++++ FINALIZANDO ++++++";
        if (MessageBox.Show(texto, titulo, MessageBoxButtons.YesNo,
            MessageBoxIcon.Question,
            MessageBoxDefaultButton.Button1) == DialogResult.Yes)
        {
            Application.Exit();
        }
    }
}

```


Imagem

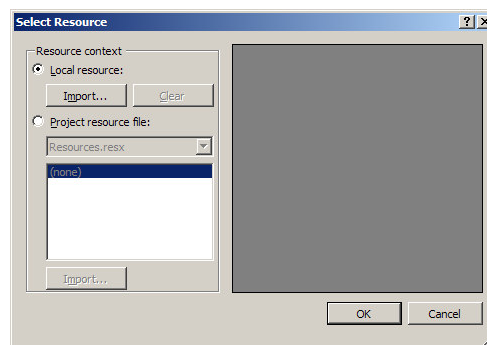
Imagem (PictureBox) – são componentes que permitem ao usuário inserir no programa uma imagem gráfica.

A grande parte das propriedades para o objeto Imagem é comum as propriedades explicadas anteriormente.

Segue uma lista das principais propriedades de um Imagem.

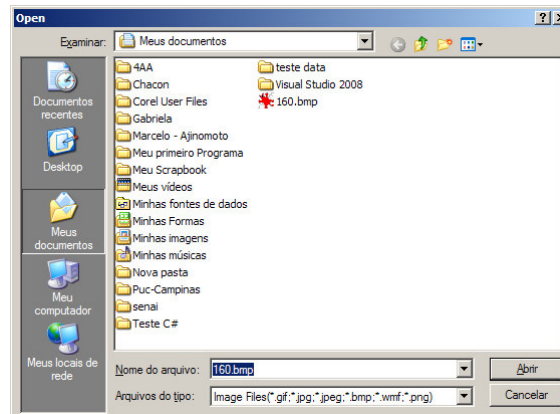
Propriedades – Características de um componente

- **Image** – Clicar sobre a reticências para abrir o Editor de Imagens



Clicar no Local resource e no botão Import e escolher a figura desejada e clicar no botão abrir.

Será carregada a figura escolhida no Editor de Imagens



Clicar no botão Abrir, e OK

SizeMode – Ajusta o tamanho da figura não tamanho estabelecido para a imagem. O tamanho da imagem escolhida é forçada a se adequar ao tamanho do componente Picture Box.

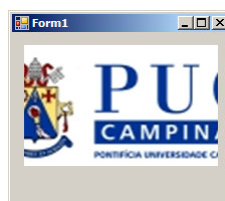
Normal – A imagem fica com o tamanho original da figura.



StretchImage – A imagem se adequa ao tamanho do componente PictureBox. O objeto poderá expandir ou diminuir de tamanho.



CenterImage – A imagem irá centralizar no objeto, podendo recortar parte da figura.



Zoom – Ajusta a imagem no tamanho do objeto.



Caixa de grupo

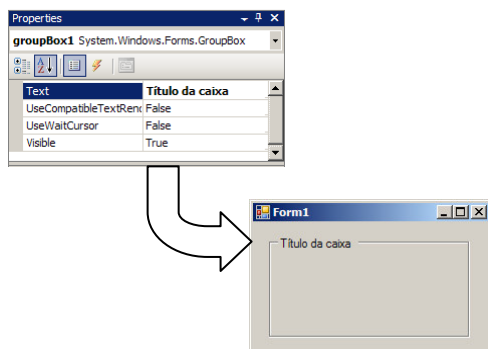
Caixa de grupo (GroupBox) são objetos no qual você pode agrupar outros objetos.

A grande parte das propriedades para o objeto caixa de grupo é comum as propriedades explicadas em outros objetos.

Segue uma lista das principais propriedades de um rótulo.

Propriedades – Características de um componente

Text – Define o texto que será escrito como título da caixa.



Itens Fundamentais

Tipos de Dados

O ambiente de programação C# trabalha com vários tipos de dados que podem ser manipulados pelo programa.

Os tipos de dados que podemos trabalhar estarão listados a seguir:

Tipos de Dados Inteiros:

Os tipos de dados inteiros são números, positivos ou negativos, excluindo qualquer número fracionário.

TIPO	FAIXA
byte	0 a 255
sbyte	-128 a 127
short	-32768 .. 32767
ushort	0 a 65535
int	-2147483648 a 2147483647
uint	0 a 4294967295
ulong	0 a 18446744073709551615
long	-9223372036854775808 a 9223372036854775807

Tipos de Dados Reais:

Os tipos de dados reais são números, positivos ou negativos, incluindo números fracionários.

TIPO	FAIXA
float	-5.1×10^{-45} a 3.4×10^{38}
decimal	-1×10^{-28} a 7.9×10^{28}
double	-3.4×10^{324} a 3.4×10^{308}

Tipos de Dados Caracteres:

Os tipos de dados caracteres são seqüência contendo letras, números e símbolos especiais. São

TIPO
<i>String</i>

Tipos de Dados Lógicos:

Os tipos de dados lógicos são representados pelas palavras *true* para verdadeiro e *false* para falso.

Variáveis

Variáveis são informações que serão armazenadas na memória do computador e poderão sofrer alterações durante a execução do programa.

Para criar uma variável é necessário fazer a declaração de variável.

Declaração de variáveis

A declaração de variável é feita em qualquer parte da função seguindo a seguinte forma:

```
<tipo> <nome das variáveis>;
```

Exemplo;

```
int A;
```

```
double B;
```

```
string nome;
```

O nome das variáveis em C# é composto de letras e números, sendo que as letras maiúsculas e minúsculas são diferentes.

Comentários

Os comentários são utilizados para documentar o programa fonte, mas não fazem parte do mesmo na compilação, pois serão substituídos por espaços em branco.

Os comentários poderão ser feitos em qualquer parte do programa.

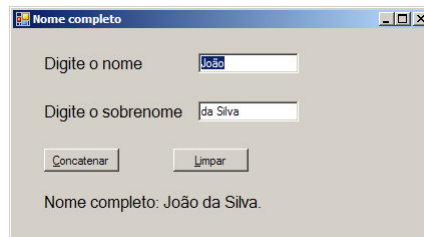
Os comentários são feitos através dos símbolos `//` se o comentário for de uma linha e através dos símbolos `/* */` se for para um trecho do programa.

Ex:

```
/* Programa elaborado por Professional Computer */  
// linha de comentário
```

Aplicação com Tipo de Dado Caractere

O exemplo a seguir concatena o nome e sobrenome de uma pessoa

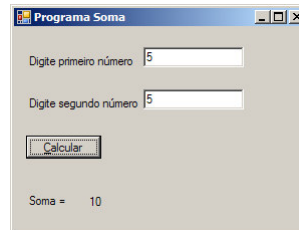


```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
  
namespace Nome_Completo  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
            label4.Text = "Nome completo: " +  
                textBox1.Text + " " +  
                textBox2.Text + ".";  
  
            textBox1.Focus();  
            textBox1.SelectAll();  
        }  
  
        private void button2_Click(object sender, EventArgs e)  
        {  
            label4.Text = "Nome completo: ";  
            textBox1.Text = "";  
            textBox2.Text = "";  
        }  
    }  
}
```

```
        textBox1.Focus();  
    }  
}
```


Aplicação com Tipo de Dado Inteiro

O exemplo a seguir soma os valores de dois números inteiros.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Soma_Numeros
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int N1, N2, S;
            try
            {
                N1 = int.Parse(textBox1.Text);
                N2 = int.Parse(textBox2.Text);
                S = N1 + N2;
                label4.Text = S.ToString();
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO ***** ",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                textBox1.Text = "";
                textBox2.Text = "";
                label4.Text = "";
                textBox1.Focus();
            }
        }
    }
}
```

Aplicação com Tipo de Dado Real

O exemplo a seguir faz calculo de salário líquido utilizando números reais.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Salario_Liquido
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double VH, HT, PD, SL;
            try
            {
                VH = double.Parse(textBox1.Text);
                HT = double.Parse(textBox2.Text);
                PD = double.Parse(textBox3.Text);
                SL = (VH * HT) - (VH * HT * PD / 100);
                SL = Math.Round(SL, 2);
                label4.Text = "R$ " + SL.ToString();
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO *****",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                textBox1.Text = "";
                textBox2.Text = "";
                textBox3.Text = "";
                label4.Text = "";
                textBox1.Focus();
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
            label4.Text = "";
            textBox1.Focus();
        }

        private void button3_Click(object sender, EventArgs e)
        {

```

```
{
    string texto = " Deseja sair do Programa Salário Líquido?";
    string titulo = "+++++++ FINALIZANDO ++++++";
    if (MessageBox.Show(texto, titulo, MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button1) == DialogResult.Yes)
    {
        Application.Exit();
    }
}
```

Operadores

Operador de atribuição =

O operador de atribuição representado pelo sinal de igual é utilizado para atribuir um valor a uma variável.

Ex:

```
int x;
```

```
x = 3;
```

ou

```
int x = 3;
```

Operadores aritméticos (+,-,*,/,%)

As expressões aritméticas são escritas linearmente, seguindo as seguintes restrições:

- a) o sinal de multiplicação é o asterisco (*)
- b) a divisão é indicada com a barra (/)
- c) o módulo de uma divisão é representado por (%)
- d) Os parênteses são usados para quebrar as prioridades entre os operadores

Ex:

```
A + B * 4
```

```
17 % 3 (Retorna 2, pois o resto da divisão de 17 por 3 é 2)
```

```
A * B * (C + (A + 5))
```

Operadores de incremento (++)

O operador de incremento soma 1 a variável.

O operador pode ser pré-fixado ou pós fixado.

Operadores de decremento (--)

O operador de decremento subtrai 1 da variável.

O operador pode ser pré-fixado ou pós fixado.

Operadores Aritméticos de Atribuição

Os operadores aritméticos são:

`+=` , `-=` , `*=` , `/=` , `%=`

Estes operadores eqüivalem a:

<code>i += 2;</code>	equivale	<code>i = i + 2;</code>
<code>i -= 2;</code>	equivale	<code>i = i - 2;</code>
<code>i *= 2;</code>	equivale	<code>i = i * 2;</code>
<code>i /= 2;</code>	equivale	<code>i = i / 2;</code>
<code>i %= 2;</code>	equivale	<code>i = i % 2;</code>

Operadores Relacionais

Uma relação é a comparação realizada entre dois valores do mesmo tipo.

A natureza da operação é indicada por um operador relacional que pode ser:

OPERADOR	RESULTADO
=	igual a
!=	diferente de
<=	menor ou igual a
<	menor que
>	maior que
>=	maior ou igual a

O resultado da relação será um valor lógico (0 ou 1).

Ex.: $A \neq B$

$C > B$

$A \leq C$

Operadores Lógicos

Os operadores lógicos são descritos em tabela abaixo:

OPERADOR	RESULTADO
&&	conjunção
	disjunção
!	negação

Tabelas:

&&		
0	0	0
0	1	0
1	0	0
1	1	1

0	0	0
0	1	1
1	0	1
1	1	1

!	
0	1
1	0

Ex.: P && Q

A || B !C

Precedência entre operadores

A tabela representa a precedência entre os vários tipos de operadores. Para quebrar estas precedências são utilizadas os parênteses.

Operador	Tipos
-	Sinal de menos unário
++	Incremento
--	Decremento
!	Não Lógico
*	Multiplicação
/	Divisão
%	Módulo
+	Adição
-	Subtração
<	Menor que
<=	Menor ou igual a
>	Maior
>=	Maior ou igual a
= =	Igual a
!=	Diferente de
&&	E lógico
	Ou lógico
?:	Condicional
=	Atribuição
*=	Aritmético de Atribuição (vezes)
/=	Aritmético de Atribuição (divide)
%=	Aritmético de Atribuição (módulo)
+=	Aritmético de Atribuição (soma)
-=	Aritmético de Atribuição (menos)

Estruturas Condicionais

Estruturas Condicionais

A estrutura condicional faz com que uns grupos de ações sejam realizados quando uma determinada condição é satisfeita.

As estruturas condicionais podem ser divididas em três partes:

- Estrutura condicional simples
- Estrutura condicional composta
- Comando seletivo

Estrutura Condicional Simples

Executa uma seqüência de comandos se a condição for verdadeira.

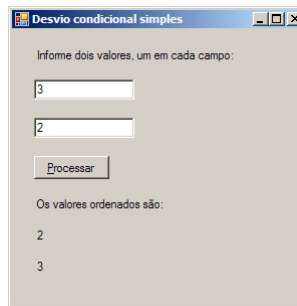
A estrutura condicional simples é representada por:

```
if (Expressão teste)
    Instrução;

ou

if (Expressão teste)
{
    Instrução;
    :
    Instrução N;
}
```

Exemplo:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace cond_simples
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int X, A, B;
            A = int.Parse(textBox1.Text);
            B = int.Parse(textBox2.Text);
            if (A > B)
            {
                X = A;
                A = B;
                B = X;
            }
            label3.Text = A.ToString();
            label4.Text = B.ToString();
            textBox1.Focus();
        }
    }
}
```

Estrutura Condicional Composta

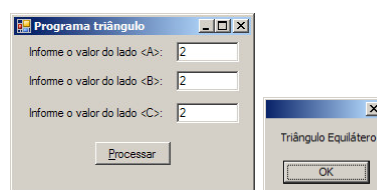
Executa uma seqüência de comandos se a condição for verdadeira, e outra seqüência de comandos se for falsa, e é representado por:

```
if (Expressão teste)
    Instrução;
else
    Instrução;
```

ou

```
if (Expressão teste)
{
    Instrução;
    :
    Instrução N;
}
else
{
    Instrução;
    :
    Instrução N;
}
```

Exemplo:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Triangulos
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

```

    }

    private void button1_Click(object sender, EventArgs e)
    {
        double A, B, C;
        A = double.Parse(textBox1.Text);
        B = double.Parse(textBox2.Text);
        C = double.Parse(textBox3.Text);
        if (A < B + C && B < A + C && C < A + B)
            if (A == B && B == C)
                MessageBox.Show("Triângulo Equilátero");
            else
                if (A == B || A == C || C == B)
                    MessageBox.Show("Triângulo Isósceles");
                else
                    MessageBox.Show("Triângulo Escaleno");
            else
                MessageBox.Show("Os valores fornecidos não formam um triângulo");
        textBox1.Focus();
    }
}

```

Comandos Seletivos

O comando `switch` é um seletor de opções, executando apenas a opção que for igual à expressão.

A forma geral do comando `switch` é:

```

switch (variável ou constante)
{
    case constante1:
        Instrução;
        :
        Instrução N;
        break;
    case constante2:
        Instrução;
        :
        Instrução N;
        break;
    case constante3:
        Instrução;
        :
        Instrução N;
}

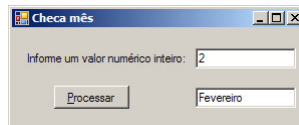
```

```

        break;
    default:
        Instrução;
        :
        Instrução N;
    }

```

Exemplo:



```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Checa_mês
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int MES;
            MES = int.Parse(textBox1.Text);
            switch (MES)
            {
                case 1: textBox2.Text = "Janeiro"; break;
                case 2: textBox2.Text = "Fevereiro"; break;
                case 3: textBox2.Text = "Março"; break;
                case 4: textBox2.Text = "Abril"; break;
                case 5: textBox2.Text = "Maio"; break;
                case 6: textBox2.Text = "Junho"; break;
                case 7: textBox2.Text = "Julho"; break;
                case 8: textBox2.Text = "Agosto"; break;
                case 9: textBox2.Text = "Setembro"; break;
                case 10: textBox2.Text = "Outubro"; break;
                case 11: textBox2.Text = "Novembro"; break;
                case 12: textBox2.Text = "Dezembro"; break;
                default: textBox2.Text = "Mês inválido"; break;
            }
            textBox1.Focus();
        }
    }
}

```

Estruturas de Repetição

Estrutura de Repetição

A estrutura de repetição permite que uma seqüência de comandos seja executada repetidamente, até que, uma determinada condição de interrupção seja satisfeita.

Existem três formas de interrupção que são:

1. Interrupção automática
2. Interrupção no início
3. Interrupção no final

Estrutura de Repetição com Interrupção Automática

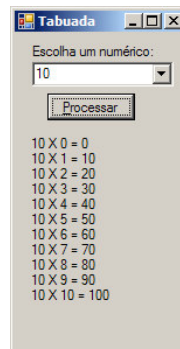
Executa um número determinado de vezes a seqüência de comandos estabelecida na repetição. É representado por:

```
for( i = 0      ; i < 20      ; i++)  
    \  Inicialização  \  Enquanto  \  Incremento  
  
for ( i = 0 ; i < 20 ; i++)  
{  
    Instrução;  
    :  
    Instrução N;  
}
```

Para a variável ser decrementada utilizamos o operador de decremento. É representado por:

```
for ( i = 30 ; i > 0 ; i--)  
{  
    Instrução;  
    :  
    Instrução N;  
}
```

Exemplo:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Tabuada
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Activated(object sender, EventArgs e)
        {
            for (int i = 1; i <= 100; i++)
                comboBox1.Items.Add(i);
        }

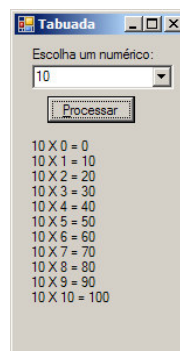
        private void button1_Click(object sender, EventArgs e)
        {
            int multiplicador, multiplicando, Resultado;
            try
            {
                listBox1.Items.Clear();
                multiplicador = int.Parse(comboBox1.Text);
                for (multiplicando = 0; multiplicando <= 10; multiplicando++)
                {
                    Resultado = multiplicador * multiplicando;
                    listBox1.Items.Add(multiplicador.ToString() +
                        " X " +
                        multiplicando.ToString() +
                        " = " +
                        Resultado.ToString());
                }
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO *****",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                comboBox1.Text = "";
                listBox1.Items.Clear();
                comboBox1.Focus();
            }
        }
    }
}
```

Estrutura de Repetição com Interrupção no Início

A condição de interrupção é testada no início, antes da seqüência de comandos, e se a condição não for satisfeita não realizará a seqüência de comandos. É representada por:

```
while (Expressão de teste)
{
    Instrução 1;
    Instrução 2;
    :
    Instrução n;
}
```

Exemplo:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Tabuada
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Activated(object sender, EventArgs e)
        {
            int I = 1;
            while(I<=100)
            {
                comboBox1.Items.Add(I);
                I++;
            }
        }

        private void button1_Click(object sender, EventArgs e)
```



```

    {
        int multiplicador, multiplicando, Resultado;
        try
        {
            listBox1.Items.Clear();
            multiplicador = int.Parse(comboBox1.Text);
            multiplicando = 1;
            while (multiplicando <= 10)
            {
                Resultado = multiplicador * multiplicando;
                listBox1.Items.Add(multiplicador.ToString() +
                                   " X " +
                                   multiplicando.ToString() +
                                   " = " +
                                   Resultado.ToString());
                multiplicando++;
            }
        }
        catch (Exception erro)
        {
            MessageBox.Show(erro.Message, "***** ERRO ***** ",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error);
            comboBox1.Text = "";
            listBox1.Items.Clear();
            comboBox1.Focus();
        }
    }
}

```

Estrutura de Repetição com Interrupção no Final

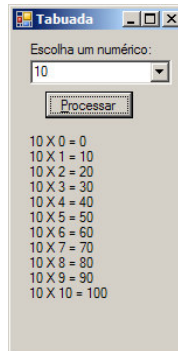
A condição de interrupção é testada após a execução da seqüência de comandos, e executará pelo menos uma vez esta seqüência de comandos. É representada por:

```

do
{
    Instrução;
    :
    Instrução N;
} while (teste);

```

Exemplo:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Tabuada
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Activated(object sender, EventArgs e)
        {
            int I = 1;
            do
            {
                comboBox1.Items.Add(I);
                I++;
            } while (I <= 100);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int multiplicador, multiplicando, Resultado;
            try
            {
                listBox1.Items.Clear();
                multiplicador = int.Parse(comboBox1.Text);
                multiplicando = 1;
                do
                {
                    Resultado = multiplicador * multiplicando;
                    listBox1.Items.Add(multiplicador.ToString() +
                                     " X " +
                                     multiplicando.ToString() +
                                     " = " +
                                     Resultado.ToString());
                    multiplicando++;
                } while (multiplicando <= 10);
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO ***** ",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
                comboBox1.Text = "";
                listBox1.Items.Clear();
                comboBox1.Focus();
            }
        }
    }
}
```

Funções

A função é um conjunto de tarefas que desejamos realizar.

Devemos escrever funções para quebrarmos o programa em subprogramas que executem tarefas simples e com baixa complexidade.

Devem-se escrever funções para tarefas que repitam várias vezes no programa.

A estrutura básica de um programa em C# são as funções. O programa em C# é constituído de uma ou mais funções.

Forma geral das funções

```
tipo    nome ( tipo )  
{  
    instrução_1;  
    instrução_2;  
    :      :  
    instrução_N  
}
```

onde:

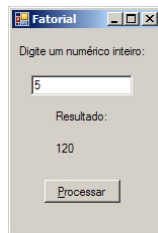
tipo ⇒ define que tipo de variável é recebida ou enviada pela função.

instrução ⇒ instruções utilizadas para compor o programa

{ ⇒ marca o início da função

} ⇒ marca o final da função

Exemplo de Funções



A estrutura básica das funções em C# é apresentada abaixo:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Fatorial
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        //Função declarada pelo usuário
        int Fatorial(int N)
        {
            int I, FAT = 1;
            for (I = 1; I <= N; I++)
            {
                FAT = FAT * I;
            }
            return (FAT);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int N, FAT;
            try
            {
                N = int.Parse(textBox1.Text);
                if (N < 1 || N > 10)
                {
                    MessageBox.Show("Entrada de Valores Inválidos\nDigite os valores
novamente!!!",
                    "***** NÚMERO INVÁLIDO *****",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Warning);
                }
                else
                {
                    FAT = Fatorial(N);
                    label3.Text = FAT.ToString(); // Mostra resultado
                    textBox1.Focus(); // Posiciona cursor em Edit1
                }
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.Message, "***** ERRO ***** ",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
                textBox1.Text = "";
                label3.Text = "";
                textBox1.Focus();
            }
        }
    }
}
```

Matrizes

Matrizes correspondem a conjuntos de elementos de um mesmo tipo, onde todos são igualmente acessíveis, isto quer dizer que o tempo e tipo de procedimento para acessar qualquer um dos elementos é identificado pelo nome da matriz seguido de um ou mais índices que define sua posição na matriz. Podem ser dos tipos: unidimensional ou multidimensional.

Podemos montar uma matriz de caracteres para armazenarmos uma frase.

Matriz Unidimensional

A criação de variáveis compostas unidimensionais é feita através da seguinte declaração:

```
int notas [5];
```

```
string letras [6];
```

Para entendermos melhor a definição de matrizes, podemos imaginar a seguinte situação: temos que armazenar dados de uma empresa por meses, e para isto poderíamos ter 12 variáveis diferentes que poderiam por sua vez, estar em locais diferentes de memória.

```
int    mes01, mes02, mes03,  
        mes04, mes05, mes06,  
        mes07, mes08, mes09,  
        mes10, mes11, mes12;
```

ou ainda, estar em uma área contínua, permitindo um acesso um pouco mais facilitado, aliás a grande vantagem de se ter uma matriz está justamente no fato de se poder criar algoritmos, que facilitam a lógica de alguns programas:

```
int    meses [12];
```

a referência a um dos elementos de uma matriz é feita, de acordo com o deslocamento de endereço, partindo do primeiro elemento, mais o tamanho em bytes de cada elemento, multiplicado pela posição menos 1. Se na matriz meses acima descrita, quisermos nos referir ao mês de maio, basta que coloquemos o valor 5 como índice:

```
meses[5] = expressão;
```

Ex: Dado um conjunto de 8 notas, armazená-las em uma variável nota, determinando sua média.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Vetor_Notas
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double[] Nota = new double[8];
            double SOMA=0, MEDIA;
            try
            {
                Nota[0]=double.Parse(textBox1.Text);
                Nota[1]=double.Parse(textBox2.Text);
                Nota[2]=double.Parse(textBox3.Text);
                Nota[3]=double.Parse(textBox4.Text);
                Nota[4]=double.Parse(textBox5.Text);
                Nota[5]=double.Parse(textBox6.Text);
                Nota[6]=double.Parse(textBox7.Text);
                Nota[7]=double.Parse(textBox8.Text);
                for (int i=0;i<=7;i++)
                {
                    SOMA+=Nota[i];
                }
                MEDIA=SOMA/8;
                MEDIA = Math.Round(MEDIA, 2);
                label3.Text = MEDIA.ToString();
            }
            catch (Exception erro)
            {
            }
        }
    }
}
```

```

    {
        MessageBox.Show(erro.Message, "***** ERRO ***** ",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        textBox4.Text = "";
        textBox5.Text = "";
        textBox6.Text = "";
        textBox7.Text = "";
        textBox8.Text = "";
        textBox1.Focus();
    }
}
}

```

Matriz Multidimensional

São as variáveis que utilizam mais que um índice para a definição da posição de seus elementos.

A dimensão da matriz em computação é conceituada diferente da matemática.

Um vetor em matemática pode possuir n dimensões, enquanto em computação ele possui apenas uma dimensão.

A declaração da matriz multidimensional é feita:

```
int notas [5][4]; // [5] => número do aluno, [4] => número da
classe.
```

Ex: Programa de matriz multidimensional.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

```

```

namespace Matriz1
{
    public partial class Form1 : Form
    {
        int[,] A = new int[4, 5];
        int[,] B = new int[4, 5];

        public Form1()
        {
            InitializeComponent();
        }

        void Limpa()
        {
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
            textBox4.Text = "";
            textBox5.Text = "";
            textBox6.Text = "";
            textBox7.Text = "";
            textBox8.Text = "";
            textBox9.Text = "";
            textBox10.Text = "";
            textBox11.Text = "";
            textBox12.Text = "";
            textBox13.Text = "";
            textBox14.Text = "";
            textBox15.Text = "";
            textBox16.Text = "";
            textBox17.Text = "";
            textBox18.Text = "";
            textBox19.Text = "";
            textBox20.Text = "";
            textBox21.Text = "";
            textBox22.Text = "";
            textBox23.Text = "";
            textBox24.Text = "";
            textBox25.Text = "";
            textBox26.Text = "";
            textBox27.Text = "";
            textBox28.Text = "";
            textBox29.Text = "";
            textBox30.Text = "";
            textBox31.Text = "";
            textBox32.Text = "";
            textBox33.Text = "";
            textBox34.Text = "";
            textBox35.Text = "";
            textBox36.Text = "";
            textBox37.Text = "";
            textBox38.Text = "";
            textBox39.Text = "";
            textBox40.Text = "";
            textBox1.Focus();
        }

        void Entrada()
        {
            A[0, 0] = int.Parse(textBox1.Text);
            A[0, 1] = int.Parse(textBox2.Text);
            A[0, 2] = int.Parse(textBox3.Text);
            A[0, 3] = int.Parse(textBox4.Text);
            A[0, 4] = int.Parse(textBox5.Text);
            A[1, 0] = int.Parse(textBox6.Text);
            A[1, 1] = int.Parse(textBox7.Text);
            A[1, 2] = int.Parse(textBox8.Text);
            A[1, 3] = int.Parse(textBox9.Text);
            A[1, 4] = int.Parse(textBox10.Text);
            A[2, 0] = int.Parse(textBox11.Text);
            A[2, 1] = int.Parse(textBox12.Text);
            A[2, 2] = int.Parse(textBox13.Text);
            A[2, 3] = int.Parse(textBox14.Text);
            A[2, 4] = int.Parse(textBox15.Text);
            A[3, 0] = int.Parse(textBox16.Text);
            A[3, 1] = int.Parse(textBox17.Text);
            A[3, 2] = int.Parse(textBox18.Text);
            A[3, 3] = int.Parse(textBox19.Text);
            A[3, 4] = int.Parse(textBox20.Text);
        }
    }
}

```



```

    }

    void Saida()
    {
        textBox21.Text = B[0, 0].ToString();
        textBox22.Text = B[0, 1].ToString();
        textBox23.Text = B[0, 2].ToString();
        textBox24.Text = B[0, 3].ToString();
        textBox25.Text = B[0, 4].ToString();
        textBox26.Text = B[1, 0].ToString();
        textBox27.Text = B[1, 1].ToString();
        textBox28.Text = B[1, 2].ToString();
        textBox29.Text = B[1, 3].ToString();
        textBox30.Text = B[1, 4].ToString();
        textBox31.Text = B[2, 0].ToString();
        textBox32.Text = B[2, 1].ToString();
        textBox33.Text = B[2, 2].ToString();
        textBox34.Text = B[2, 3].ToString();
        textBox35.Text = B[2, 4].ToString();
        textBox36.Text = B[3, 0].ToString();
        textBox37.Text = B[3, 1].ToString();
        textBox38.Text = B[3, 2].ToString();
        textBox39.Text = B[3, 3].ToString();
        textBox40.Text = B[3, 4].ToString();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        Limpa();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        int NUMERO, RESPOSTA, I, J;
        Entrada();
        for (I = 0; I <= 3; I++)
            for (J = 0; J <= 4; J++)
            {
                NUMERO = A[I, J];
                RESPOSTA = NUMERO * NUMERO;
                B[I, J] = RESPOSTA;
            }
        Saida();
    }

    private void button3_Click(object sender, EventArgs e)
    {
        int NUMERO, RESPOSTA, I, J;
        Entrada();
        for (I = 0; I <= 3; I++)
            for (J = 0; J <= 4; J++)
            {
                NUMERO = A[I, J];
                RESPOSTA = NUMERO * NUMERO * NUMERO;
                B[I, J] = RESPOSTA;
            }
        Saida();
    }

    private void button4_Click(object sender, EventArgs e)
    {
        int RESPOSTA, I, J;
        Entrada();
        for (I = 0; I <= 3; I++)
            for (J = 0; J <= 4; J++)
            {
                RESPOSTA = A[I, J];
                if (RESPOSTA % 2 == 0)
                    B[I, J] = RESPOSTA;
                else
                    B[I, J] = 0;
            }
        Saida();
    }

    private void button5_Click(object sender, EventArgs e)
    {
        int RESPOSTA, I, J;

```

```

Entrada();
for (I = 0; I <= 3; I++)
    for (J = 0; J <= 4; J++)
    {
        RESPOSTA = A[I,J];
        if (RESPOSTA % 2 != 0)
            B[I,J] = RESPOSTA;
        else
            B[I,J] = 0;
    }
Saida();
}

private void button6_Click(object sender, EventArgs e)
{
    int SOMAT, LIMITE, I, J, K;
    Entrada();
    for (I = 0; I <= 3; I++)
        for (J = 0; J <= 4; J++)
        {
            SOMAT = 0;
            LIMITE = A[I,J];
            for (K = 1; K <= LIMITE; K++)
                SOMAT += K;
            B[I,J] = SOMAT;
        }
    Saida();
}

private void button7_Click(object sender, EventArgs e)
{
    int FAT, LIMITE, I, J, K;
    Entrada();
    for (I = 0; I <= 3; I++)
        for (J = 0; J <= 4; J++)
        {
            FAT = 1;
            LIMITE = A[I,J];
            for (K = 1; K <= LIMITE; K++)
                FAT *= K;
            B[I,J] = FAT;
        }
    Saida();
}
}
}
}

```

Estruturas

Estrutura é um grupo de informações relativas a uma mesma entidade. Estas informações podem ter características diferentes, como em cadastro onde temos o nome, endereço, telefone, etc.

struct

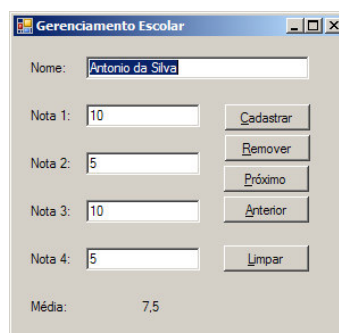
A declaração de uma `struct` permite-nos a definir um estrutura.

```
Public struct registro
{
    public int num;
    public string nome;
    public string end;
    public string fone;
};
```

Após a declaração da estrutura deve-se declarar variáveis que conterão esta estrutura.

```
CadAluno[] registro = new CadAluno[8];
```

Ex:



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```

namespace Registro
{
    public struct CadAluno
    {
        public string NOME;
        public double NOTA1;
        public double NOTA2;
        public double NOTA3;
        public double NOTA4;
        public double MEDIA;
    };

    public partial class Form1 : Form
    {
        CadAluno[] aluno = new CadAluno[8];
        int I;
        public Form1()
        {
            InitializeComponent();
        }

        void LimpaTela()
        {
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
            textBox4.Text = "";
            textBox5.Text = "";
            label7.Text = "";
            textBox1.Focus();
        }

        void Limpar()
        {
            for (int J = 0; J <= 7; J++)
            {
                aluno[J].NOME = "";
                aluno[J].NOTA1 = 0;
                aluno[J].NOTA2 = 0;
                aluno[J].NOTA3 = 0;
                aluno[J].NOTA4 = 0;
                aluno[J].MEDIA = 0;
            }
            LimpaTela();
            /* Inicializa contador de registro */
            I = 0;
        }

        void ExibeRegistro(int POSICAOATUAL)
        {
            textBox1.Text = aluno[POSICAOATUAL].NOME;
            aluno[POSICAOATUAL].NOTA1 = Math.Round(aluno[POSICAOATUAL].NOTA1, 2);
            textBox2.Text = aluno[POSICAOATUAL].NOTA1.ToString();
            aluno[POSICAOATUAL].NOTA2 = Math.Round(aluno[POSICAOATUAL].NOTA2, 2);
            textBox3.Text = aluno[POSICAOATUAL].NOTA2.ToString();
            aluno[POSICAOATUAL].NOTA3 = Math.Round(aluno[POSICAOATUAL].NOTA3, 2);
            textBox4.Text = aluno[POSICAOATUAL].NOTA3.ToString();
            aluno[POSICAOATUAL].NOTA4 = Math.Round(aluno[POSICAOATUAL].NOTA4, 2);
            textBox5.Text = aluno[POSICAOATUAL].NOTA4.ToString();
            aluno[POSICAOATUAL].MEDIA = Math.Round(aluno[POSICAOATUAL].MEDIA, 2);
            label7.Text = aluno[POSICAOATUAL].MEDIA.ToString();
            textBox1.Focus();
        }

        private void Form1_Activated(object sender, EventArgs e)
        {
            Limpar();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double SOMA;
            if (I <= 7 && String.Equals(aluno[I].NOME, ""))
            {
                aluno[I].NOME = textBox1.Text;
                aluno[I].NOTA1 = double.Parse(textBox2.Text);
                aluno[I].NOTA2 = double.Parse(textBox3.Text);
            }
        }
    }
}

```

```

        aluno[I].NOTA3 = double.Parse(textBox4.Text);
        aluno[I].NOTA4 = double.Parse(textBox5.Text);
        SOMA = 0;
        SOMA += aluno[I].NOTA1 + aluno[I].NOTA2 +
                aluno[I].NOTA3 + aluno[I].NOTA4;

        aluno[I].MEDIA = SOMA / 4;
        aluno[I].MEDIA = Math.Round(aluno[I].MEDIA, 2);
        label7.Text = aluno[I].MEDIA.ToString();
        I++; /* Conta registro */
        LimpaTela();
    }
    else
    {
        LimpaTela();
        if (I > 7)
        {
            MessageBox.Show("Memória cheia", "***** Erro de memória *****",
                            MessageBoxButtons.OK);
        }
        else
        {
            MessageBox.Show("Posição ocupada", "***** Erro de memória *****",
                            MessageBoxButtons.OK);
        }
    }
}

private void button2_Click(object sender, EventArgs e)
{
    aluno[I].NOME = "";
    aluno[I].NOTA1 = 0;
    aluno[I].NOTA2 = 0;
    aluno[I].NOTA3 = 0;
    aluno[I].NOTA4 = 0;
    aluno[I].MEDIA = 0;
    LimpaTela();
}

private void button3_Click(object sender, EventArgs e)
{
    if (I < 7)
    {
        I++;
        ExibeRegistro(I);
    }
}

private void button4_Click(object sender, EventArgs e)
{
    if (I > 0)
    {
        I--;
        ExibeRegistro(I);
    }
}

private void button5_Click(object sender, EventArgs e)
{
    Limpar();
}
}
}

```

Bibliografia

- 1 SANTOS, Luis Carlos dos, **Microsoft Visual C# 2008 Express Edition, Aprenda na prática**, São Paulo, SP, Editora Érica, 2009.
- 2 MATEUS, César Augusto, **C++ Builder 5 - Guia Prático**, São Paulo, SP, Editora Érica, 2000.
- 3 MANZANO, José Augusto Navarro Garcia, **Estudo Dirigido de C++ Builder**, São Paulo, SP, Editora Érica, 2003.
- 4 MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C++**. São Paulo. Makron Books do Brasil Editora Ltda.
- 5 BERRY, John. **Programando em C++**. São Paulo. Makron Books do Brasil Editora Ltda.
- 6 SCHILDT, Herbert. **Linguagem C – Guia Prático e Interativo**. São Paulo. Makron Books do Brasil Editora Ltda.
- 7 SCHILDT, Herbert. **Linguagem C – Completo e total**. São Paulo. Makron Books do Brasil Editora Ltda.
- 8 ELLIS, Margaret A., Stroustrup Bjarne. **C++ Manual de Referência Comentado**. Rio de Janeiro. Editora Campus Ltda.
- 9 DAVIS, Stephen R. **C++ Para Leigos**. São Paulo. Berkeley.
- 10 JAMSA, Kris. **Salvo pelo... C++**. Rio de Janeiro. Livros Técnicos e Científicos Ltda.
- 11 BERGSMAN, Paul. **Controlling the world with your PC**. Eagle Rock, Virginia. LLH Technology Publishing.