
# Apostila Eletrônica



**Obrigado pela preferência,**

**Celta Informática.**
**http://www.celta.com.br**
**E-mail: comercial@celta.com.br**
**Fone: (0xx11) 7670-1586**

# Visual Basic 4

# SUMÁRIO

# Microsoft Visual Basic 4.0

## PRIMEIROS PASSOS

Vantagens: - Facilidade em alterações e implementações
- Melhor Estruturação do código

Linguagens: - Turbo Pascal, Quick Pascal, Turbo $C^{++}$ e $C/C^{++}$ (Baseadas em Objetos)

- Visual Basic e Delphi (Baseadas em Eventos)

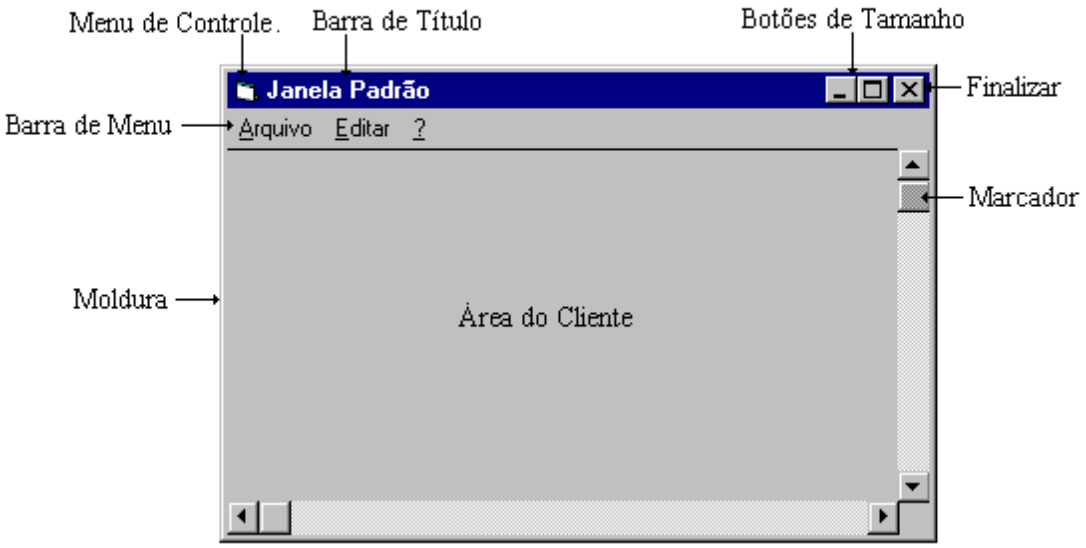
O Visual Basic Permite o uso de objetos, mas não a sua criação. Ele trabalha com eventos que dão início à alguma rotina de trabalho, ou seja, o programa fica parado até que um evento ocorra.

Um programa tradicional, feito para ser executado em DOS, é organizado em torno de estruturas de dados, com procedimentos e funções separadas para manipular os dados.

Um programa orientado para objetos e eventos é organizado em torno de um conjunto de objetos, que são estruturas que combinam dados e rotinas em uma mesma entidade. Um Objeto possui dados internos, que não podem ser acessados de fora dele e dados externos, também chamados de propriedades, que podem ser acessados de fora deste objeto. De maneira semelhante, ele possui rotinas internas que são usadas apenas internamente e rotinas externas, também chamadas de métodos, que podem ser acessadas externamente.

Um método é uma rotina própria do objeto que o dá funcionalidade, isto é, torna-o "vivo", e as propriedades fazem o intercâmbio entre o objeto e o programa.

## ANATOMIA DE UMA JANELA




Moldura - Os quatro lados da janela, que definem seu tamanho.

Barra de Título - Abaixo da moldura superior com nome da janela e documento corrente.

Menu de Controle - A esquerda da Barra de Título. Um botão com um ícone que representa o programa.

Botões de Tamanho - A direita da Barra de Título. São dois botões, um com um traço e o outro com duas janelinhas ou uma janela desenhada. Se forem duas janelinhas, mostra que a janela está maximizada e se for uma janela um pouco maior, mostra que a janela está em seu tamanho normal e pode ser maximizada. O botão com um traço serve para minimizar a janela.

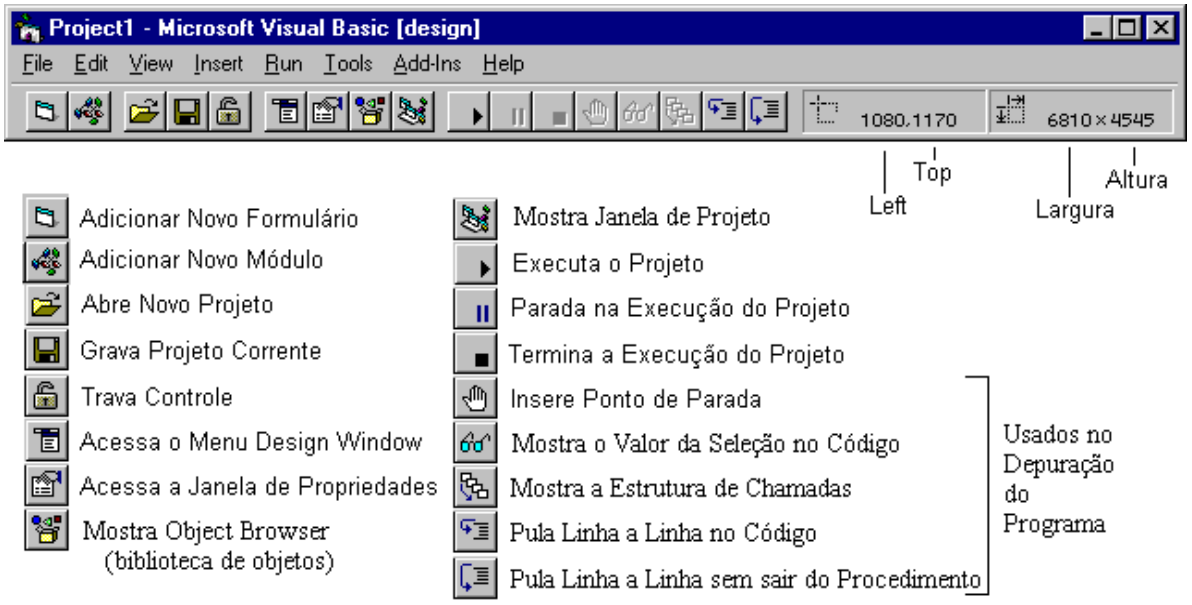
Barra de Menu - Está abaixo da barra de título e contém as opções de controle do aplicativo.

Área do Cliente - É a parte interna da janela, também chamada de área do documento. No VB, é o espaço que temos para inserir os controles da nossa aplicação.

Marcador - botão deslizante para rolar a tela.

Janela - Uma Janela é plena quando podemos dimensioná-la (mini, maxi e restaurá-la) e movê-la.

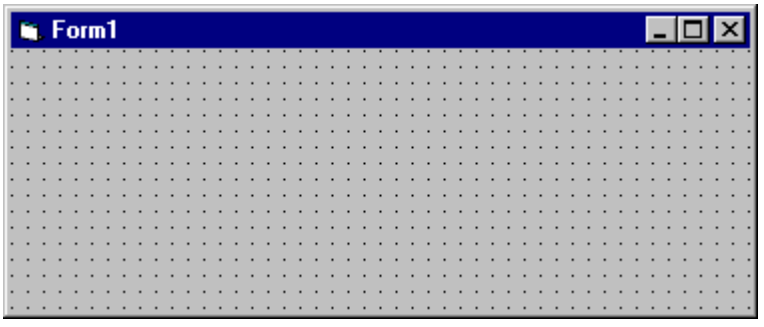
## BARRA DE MENU E FERRAMENTAS




## Janelas do Visual Basic

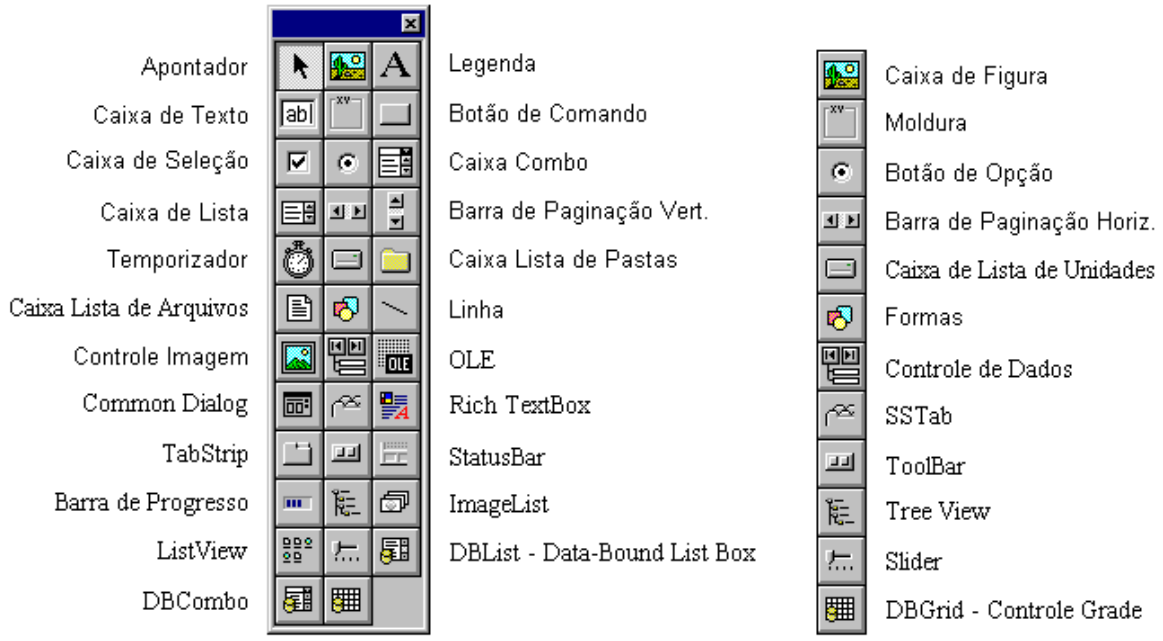
Formulário

Existem dois tipos de objetos no VB, formulários e controles. O Formulário é a janela que aparece no centro da tela do Visual Basic. Essa é a janela que estamos projetando. Quando rodarmos o aplicativo, será ela que irá aparecer com os objetos que nós incorporamos




Controles

Controles são todos os objetos que podemos trabalhar, inserindo-o em um formulário ou controlando os seus eventos e propriedades. Um controle é qualquer objeto que o usuário possa manipular, desde que não seja uma janela (formulário).




A Janela de Controles (Toolbox), possui todos os controles que iremos precisar para desenharmos nossa janela - formulário - como um programa de desenho livre. Para incluir um controle ao formulário, existem dois métodos:

1 - Click Duplo no ícone do controle, na caixa de ferramentas. Que fará com que o controle seja inserido no centro do formulário com um tamanho padrão.

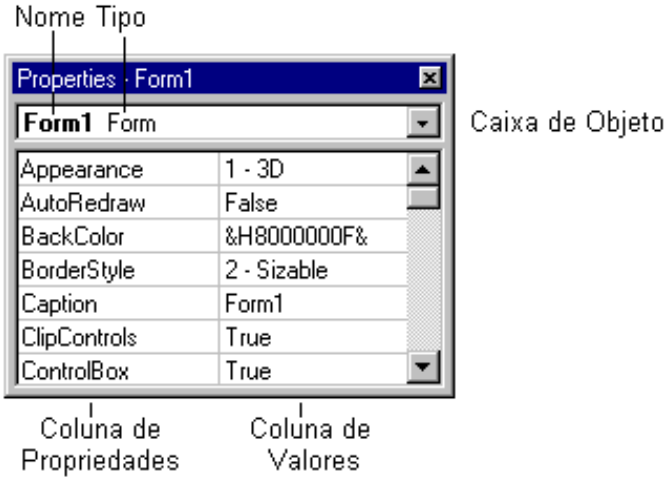
2 - Selecionar o ícone na caixa de ferramentas, e depois dimensioná-lo no formulário, arrastando e soltando o mouse na área do cliente, no formulário.

Podemos dimensionar estes controles, depois de inseridos, a qualquer momento durante o desenvolvimento. Primeiro, selecionamos o controle dando um clique em cima dele, em seguida, o dimensionamos arrastando um dos oito dimensionadores que circundam este objeto.

## Propriedades

Nesta janela definiremos como serão as características de cada objeto do formulário (botões de comando, quadros de texto, formulários, e outros), definindo como eles serão apresentados. Cada um desses objetos possui um conjunto específico de propriedades que podem ser associados a eles. Ao trabalharmos nos diferentes objetos, a janela de propriedades nos permitirá mudar as propriedades do objeto com que estamos trabalhando.

Existem propriedades que podemos mudar enquanto construímos nosso projeto, ou seja, em tempo de projeto, e outras propriedades que só podemos mudar durante a execução do projeto, neste caso, em tempo de execução.




Na janela de propriedades acima, temos algumas das propriedades do nosso formulário assim que iniciamos o VB.

Nome - Contém o nome do objeto atualmente selecionado, este nome está na propriedade Name.

Tipo - Nesta posição encontraremos qual é o tipo do objeto selecionado, se ele é um Form (formulário), Command Button (botão de comando), Label (legenda), ou então um Text Box (quadro de texto).

Caixa de Objeto - Está caixa mostra o objeto atualmente selecionado, através dela também podemos selecionar o objeto que queremos mudar as suas propriedades, basta dar um click na seta que um menu de cortina se abrirá, e podemos selecionar o objeto que queremos trabalhar.

Coluna de Propriedades - Exibe todas as propriedades que podemos modificar em tempo de projeto do referido objeto.

Coluna de Valores - Exibe o valor da propriedade correspondente.

## MEU PRIMEIRO PROGRAMA

Para iniciar, vamos construir um programa que, quando for dado um click no botão de comando, será mostrada uma mensagem. E posteriormente poderemos alterar a cor desta mensagem através de outros botões.

Existem três passos principais, para a criação de uma aplicação no Visual Basic, que iremos seguir:

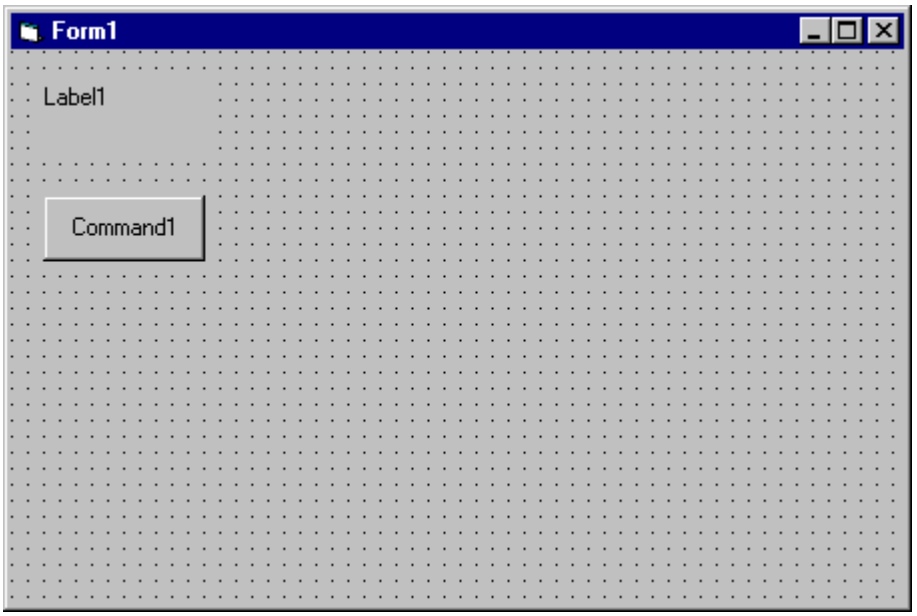
- **Desenhar as janelas que se deseja usar**
  Inserir no formulário os controles que serão necessários

- **Adaptar as propriedades dos objetos**
  Alterar as propriedades dos controles às necessidades da aplicação

- **Escrever o código para os eventos associados**

  Esta é a parte mais complexa do desenvolvimento, é ela que dá a funcionalidade ao programa, são as rotinas que começam a ser executadas a partir de um evento.

### *DESENHAR AS JANELAS QUE SE DESEJA USAR*




1 - Começamos inserindo um Label (Legenda) e um Botão de Comando no Formulário, de uma das duas maneiras indicadas anteriormente.

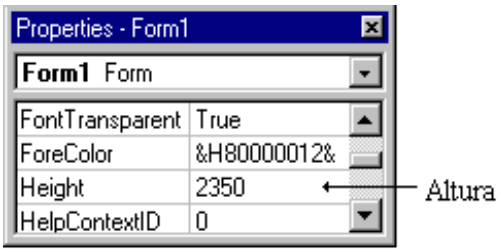
2 - Observe que, quando o controle estiver selecionado, podemos arrastá-lo para qualquer lugar no formulário.

### *ADAPTAR AS PROPRIEDADES DOS OBJETOS*

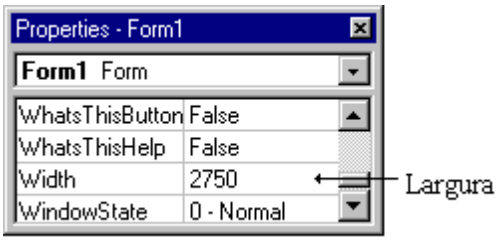
Para alterar a propriedade de um objeto, ele tem que estar selecionado (os oito pontos visíveis), em seguida, procurar o nome da propriedade a ser alterada e selecionar (no caso de valores padrão) o seu valor, ou então, escrever um valor.

1 - Dimensione o formulário da seguinte maneira:

Selecionar a propriedade Height, e entre com o valor 2350.

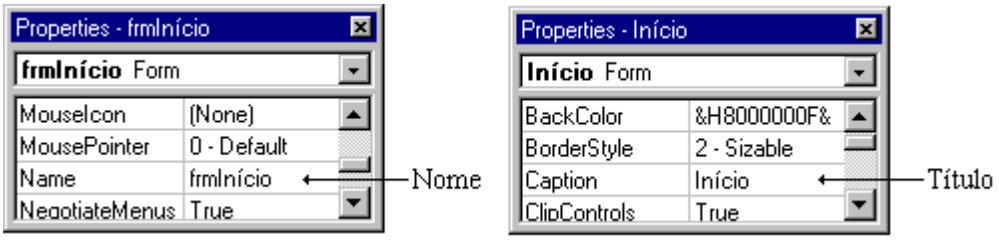



Selecionar a propriedade Width, e entre com o valor 2750.




Estes números correspondem a Twips, esta unidade foi criada para que houvesse uma independência do VB em relação aos dispositivos de entrada e saída (impressoras, monitores e scaner) e que fosse mais precisa que estes aparelhos. Um Twip corresponde a 1/1440 de polegada.

O mesmo deverá ser feito para as propriedades Name e Caption. A propriedade Name será a identificação do Objeto quando construirmos o código da aplicação. E a propriedade Caption, é a palavra que aparecerá como título da janela.

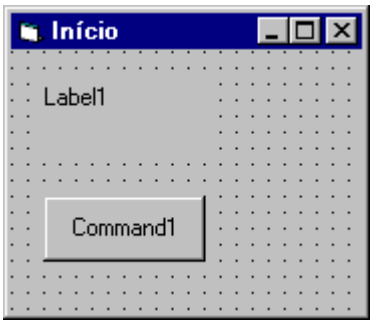



Para a propriedade Name há uma convenção indicada no manual do VB que iremos seguir neste curso; as três primeiras letras indicam o tipo do objeto, e as seguintes, um conjunto de caracteres qualquer que identifique o objeto. De preferência, com a primeira letra maiúscula para facilitar a leitura. A propriedade Name deve começar com uma letra e ter no máximo 40 caracteres e não pode conter espaços ou pontuação.

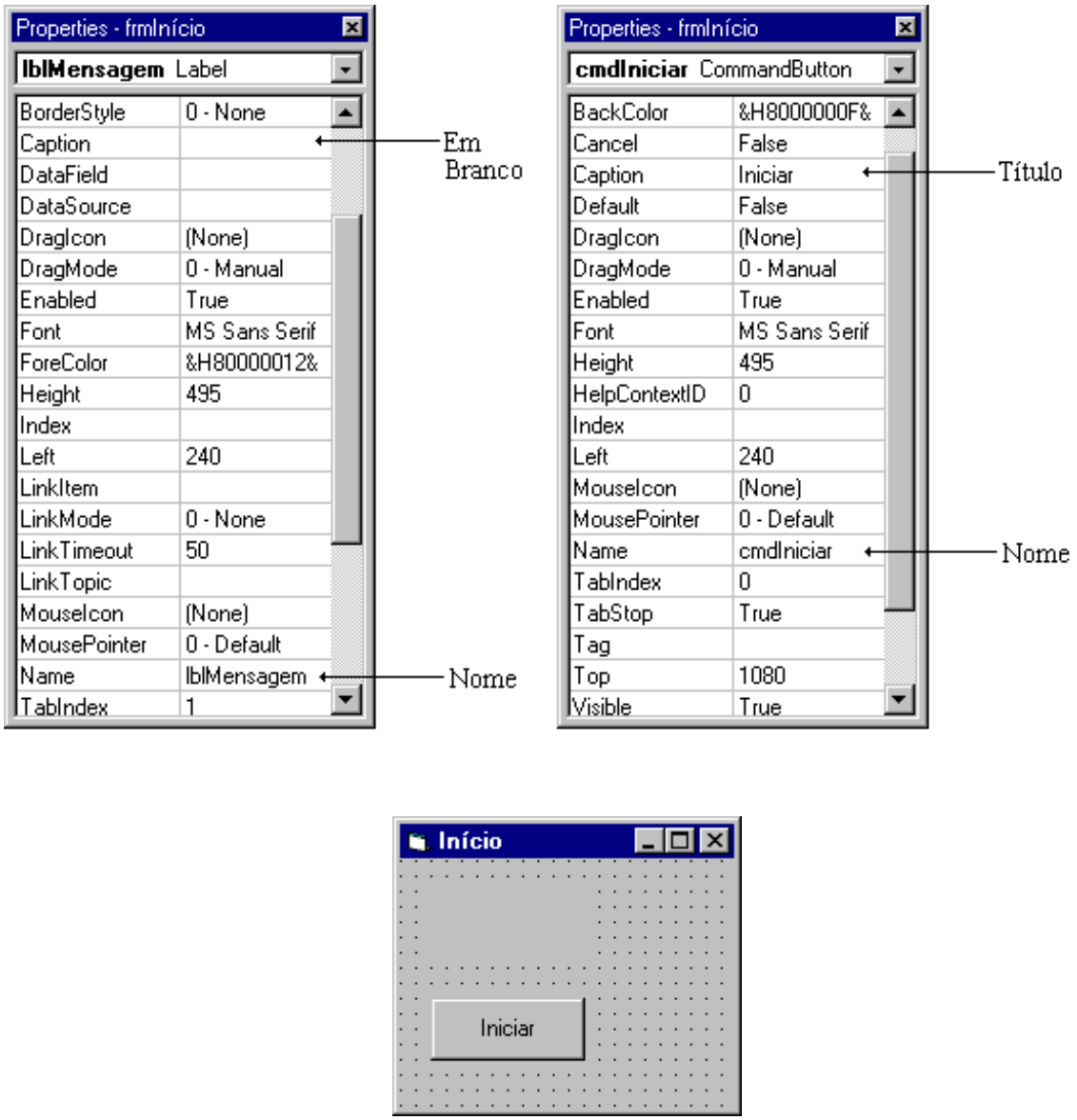
**Convenções de Nomes para o Visual Basic**

| Objeto | Prefixo | Exemplo |
|---|---|---|
| Form | frm | frmInício |
| Check box | chk | chkVerificar |
| Combo box | cbo | cboLivros |
| Command button | cmd | cmdCancelar |
| Data | dat | datBiblio |
| Directory list box | dir | dirDiretório |
| Drive list box | drv | drvDiscos |
| File list box | fil | filArquivos |
| Frame | fra | fraOpções |
| Grid | grd | grdPlanilha |
| Horizontal scroll bar | hsb | hsbVolume |
| Image | img | imgFigura |
| Label | lbl | lblNome |
| Line | lin | linSeparar |
| List box | lsl | lslCódigos |
| Menu | mnu | mnuEditar |
| OLE | ole | oleObjeto1 |
| Option button | opt | optGramas |
| Picture box | pic | picQuadro |
| Shape | shp | shpRetângulo |
| Text box | txt | txtCliente |
| Vertical scroll bar | vsb | vsbVolume |

Após você alterar estas quatro propriedades (Caption, Height, Name e Width) do formulário, ele estará assim:



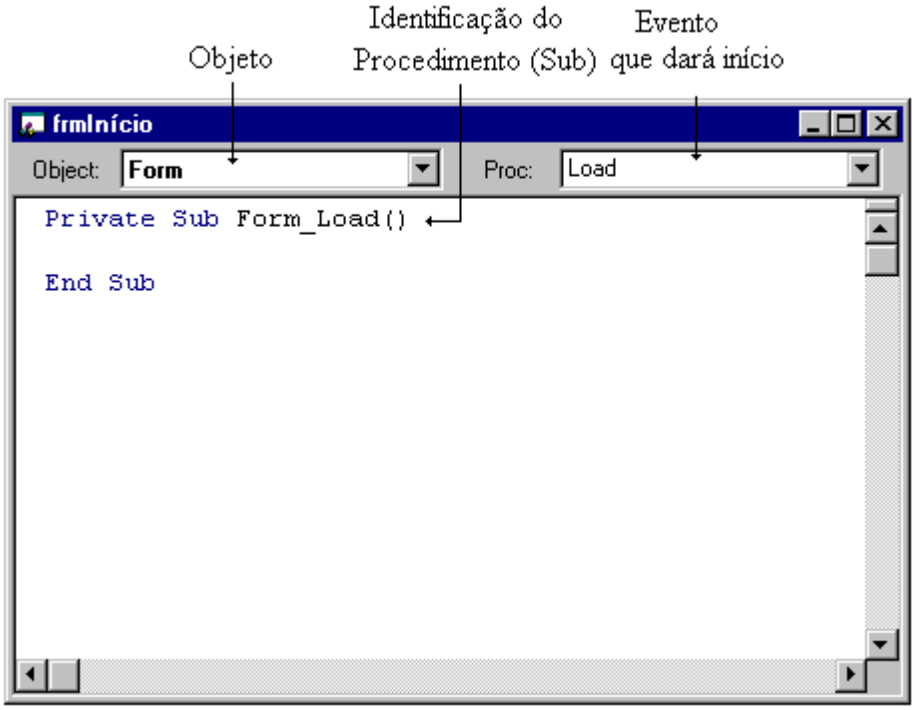

Agora, altere as propriedades do Label e do Botão de Comando.




### *ESCREVER O CÓDIGO PARA OS EVENTOS ASSOCIADOS*

O código é escrito na janela de código, para acessá-la, damos um duplo click em qualquer objeto do projeto.

## Janela de Código




Nesta janela podemos observar 3 elementos:

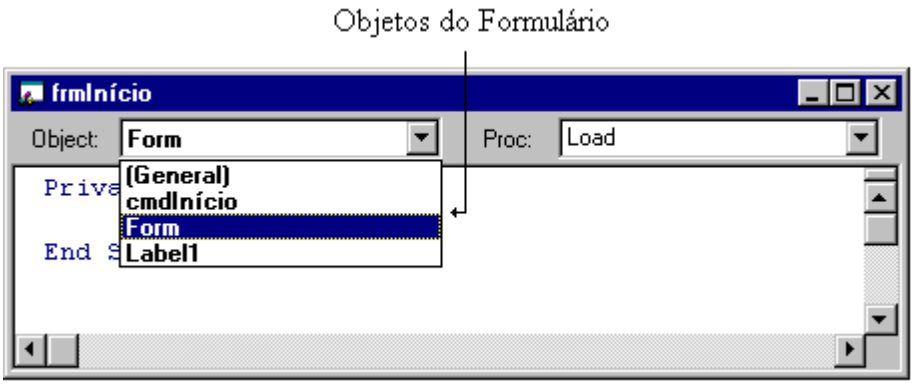
1 - Nome do Objeto associado ao procedimento.
2 - Nome do Evento que quando ocorrer, dará início ao procedimento.
3 - Procedimento que conterá os códigos

Todo procedimento inicia tendo na primeira linha o seu nome e, na última, a declaração End Sub (final).

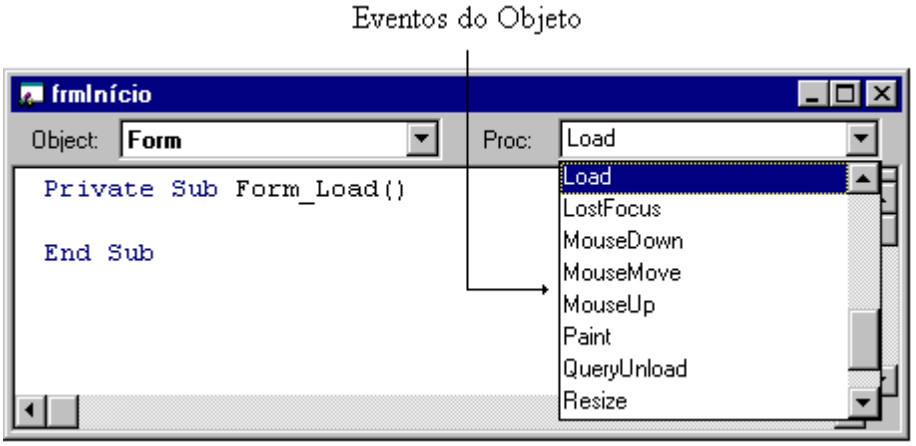
A primeira linha segue o padrão; Private Sub nome do objeto + _ + evento + ( ) .

Cada objeto tem um evento que é mais comumente utilizado, e é com este evento que o VB inicia a Janela de Código, não impedindo que utilizemos outro ou mais de um evento.

Se for dado um Click na seta do quadro de lista Object, serão mostrados todos os objetos neste formulário, e poderemos escolher em qual destes iremos trabalhar o código.




Da mesma forma, se dermos um Click na seta do Quadro de Lista Proc:, serão mostrados todos os eventos do Objeto escolhido, permitindo a seleção do evento para o qual queremos criar um procedimento.

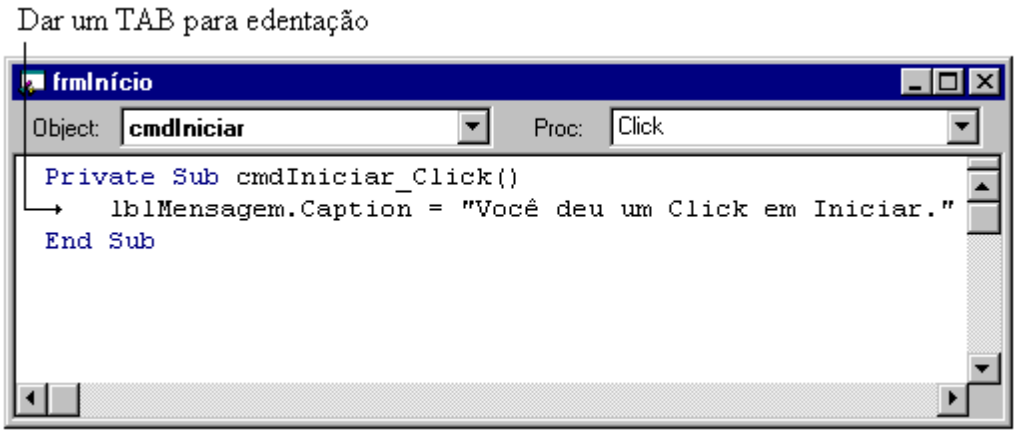



O nosso projeto Início, mostrará uma mensagem no Label (objeto) com um Click (evento) no Botão "Iniciar" (objeto). Ou seja, iremos alterar a propriedade Caption de lblMensagem, esta propriedade contém o que será mostrado ao usuário.

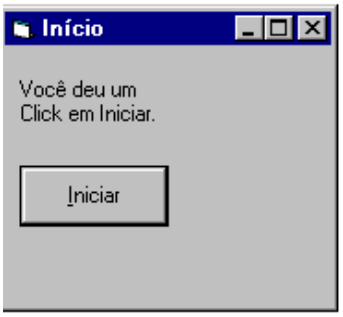
Atribuímos um valor à uma propriedade de um objeto seguindo o padrão:

objeto + . + propriedade + = + valor da propriedade

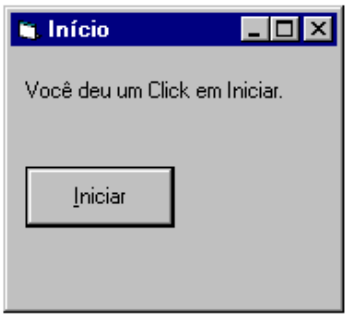
Abra a Janela de Código para o botão de comando, e digite o código conforme a figura a seguir. Quando for dado um clique em Iniciar será mostrada a mensagem "Você deu um Click em Iniciar."



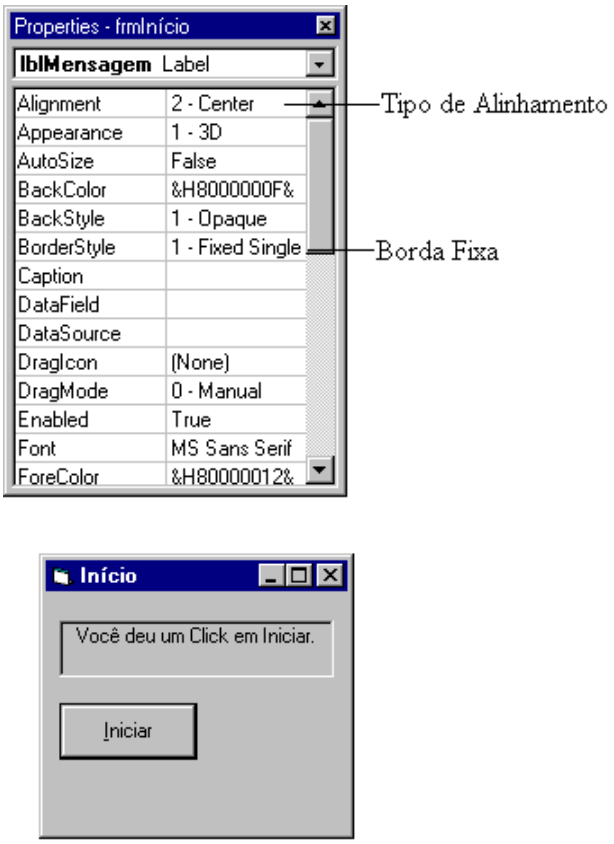

Em seguida, clique sobre o botão Executar da barra de ferramentas ( ), logo após, selecione o botão Iniciar para ver o resultado.



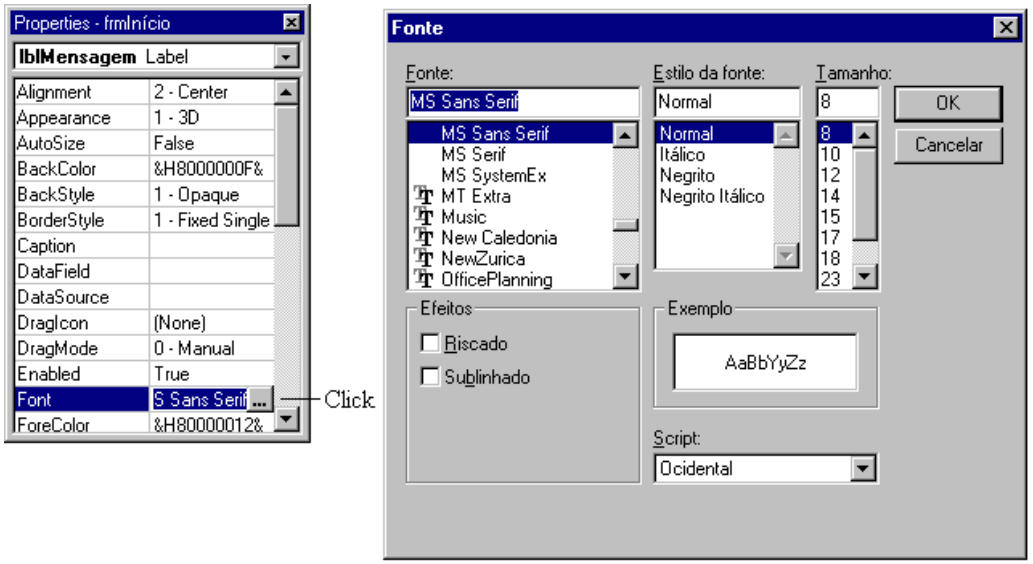

Finalize a execução através do botão Terminar ( ). Altere a dimensão do objeto lblMensagem para que toda a mensagem caiba na mesma linha. Execute e observe a mudança.




Pare a execução novamente, e altere as propriedades Aligment e BorderStyle de lblMensagem.




Existem propriedades que possuem vários valores, quando escolhemos Aligment e demos um clique na seta, apareceram os tipos de alinhamento para o texto, mas existem propriedades que possuem inúmeras escolhas, neste caso, ao invés de uma seta, encontraremos três pontos, é o caso da propriedade Font.



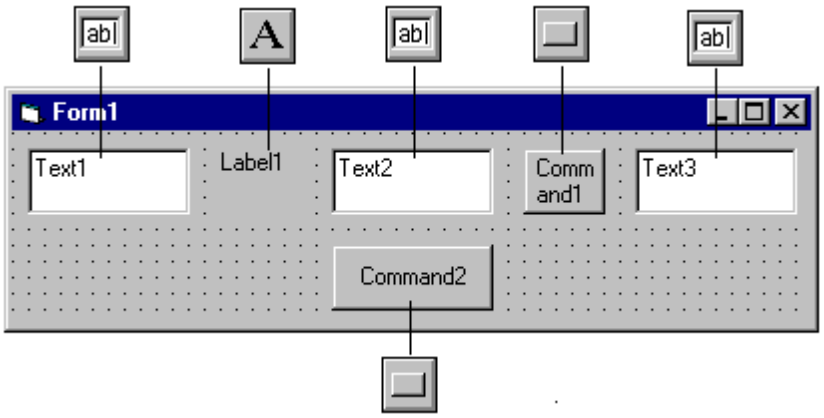

Quando selecionamos os três pontos, aparece um Quadro de Diálogo para escolher o formato da fonte para a exibição da Mensagem.

No seu projeto Iniciar, teste as alterações de fonte e observe as mudanças.

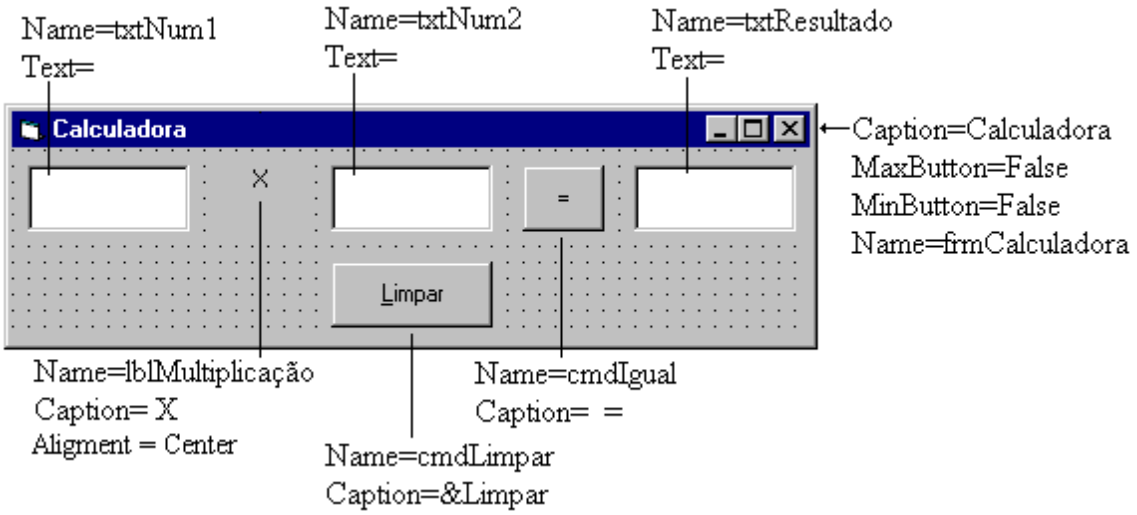
## Exemplo I - CALCULADORA

Para iniciar um novo projeto, escolha a opção New do menu File.

Dimensione e insira os controles, utilizando a Janela de Ferramentas (Toolbox) no formulário, como o exemplo abaixo. Caso a Toolbox não esteja visível, selecione a opção Toolbox do menu View. Dimensionamos o formulário no VB da mesma forma que no Windows dimensionamos as janelas.




Agora, altere as propriedades assinaladas dos Objetos conforme a figura a seguir:



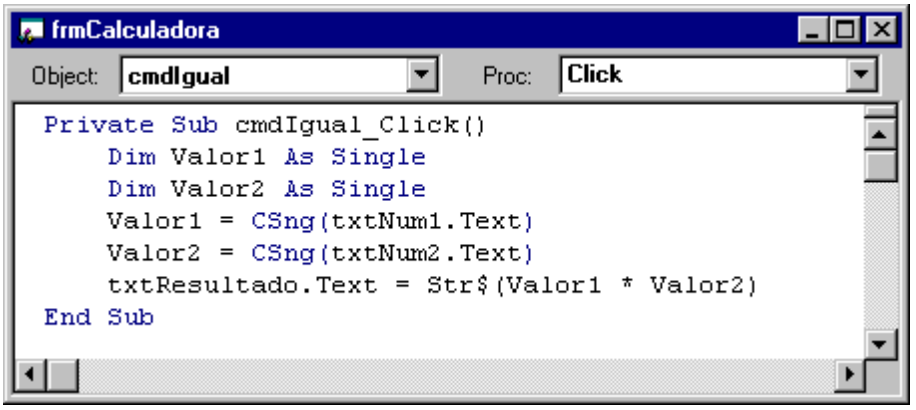

As propriedades MaxButton = False e MinButton = False, desabilitam os botões de maximizar e minimizar.

Neste exemplo de projeto, digitaremos um número em txtNum1, outro em txtNum2 e quando Click em cmdIgual, o resultado da multiplicação aparecerá em txtResultado. Para limpar os Quadros de texto, Click em cmdLimpar.

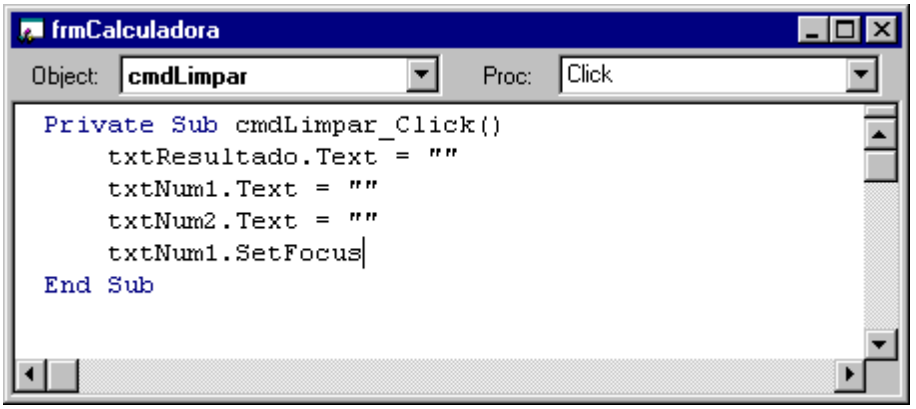
O projeto irá trabalhar basicamente com dois eventos :

Click em cmdIgual (=)
Click em cmdLimpar (Limpar)

Então, para escrever o código, damos um Duplo Click no Botão Igual, e a janela de código será mostrada. Entre com o código conforme a figura a seguir:




Alterne para o procedimento cmdLimpar_Click. E entre com os comandos a seguir:



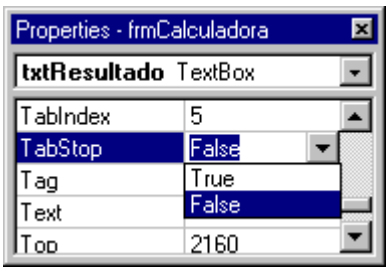

Execute o projeto. Para usar, entre com um número em txtNum1, outro em txtNum2 e dê um Click em "=", que o resultado aparecerá em txtResultado.

Note que alternamos os campos com a tecla Tab, a ordem de tabulação corresponde à ordem em que os controles foram colocados no formulário. Esta ordem é determinada pela propriedade TabIndex dos controles, caso o seu projeto não esteja, coloque-o na seguinte ordem:

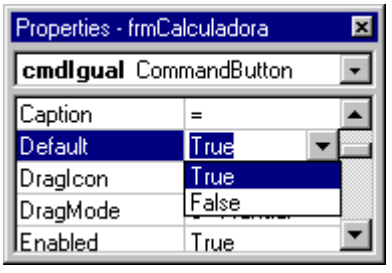
| Objeto | TabIndex |
|---|---|
| txtNum1 | 1 |
| txtNum2 | 2 |
| cmdIgual | 3 |
| cmdLimpar | 4 |
| txtResultado | 5 |
| lblMultiplicação | 6 |

Para alterar esta propriedade, basta selecionar o controle, Janela de Propriedades, procurar TabIndex e alterar o valor, o VB não aceita controles com TabIndex de mesmo valor.

Execute o projeto e observe a alteração. Note que podemos alterar o valor de txtResultado mesmo após a operação ter sido efetuada. Para evitar isso, defina a propriedade TabStop=False para txtResultado, e verá que o usuário não terá mais acesso com a tecla Tab ao txtResultado.




Existem, nas aplicações para Windows, Botões de Comando que são acionados com a tecla Enter ou com um Click neles. No nosso projeto, este Botão será o cmdIgual, para isso, defina a propriedade Default=True para esse objeto e aparecerá um contorno mais espesso, dando a indicação que se a tecla Enter for acionada, o comando será executado.



## PROPRIEDADES

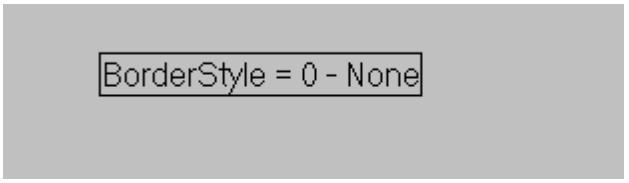
BorderStyle

Retorna ou dá o estilo de borda de um objeto;

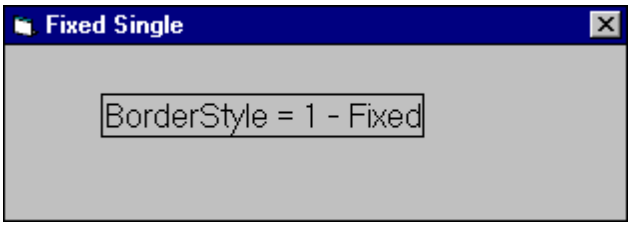
objeto.BorderStyle = [valor]

Existem 6 tipos de bordas:

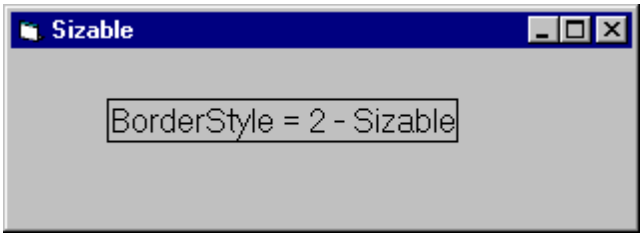
0 - None Nenhuma



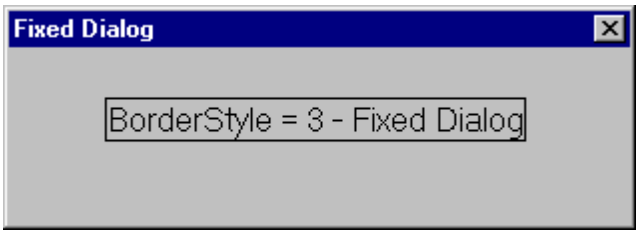

1 - Fixed Single Fixa Simples, o formulário não é dimensionável.



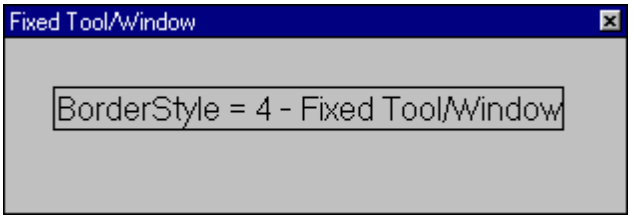

2 - Sizable Redimensionável



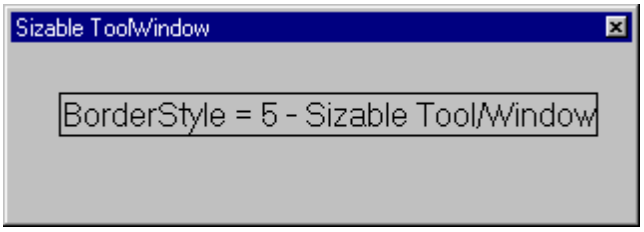

3 - Fixed Dialog O formulário não possui botões de maximizar e nem de minimizar. Não é redimensionável.



4 - Fixed ToolWindow Não mostra os botões de maximizar e de minimizar. Não é redimensionável, e mostra somente o botão de fechar e a barra de título com a fonte reduzida. E o formulário não aparece na barra de tarefa do Windows 95.




5 - Sizable ToolWindow Semelhante à Fixed ToolWindow, mas é dimensionável.




As Bordas Fixas não podem ser dimensionadas em tempo de execução. Ou seja, o usuário não poderá mudar o tamanho do formulário.

DEFAULT

Retorna ou dá o valor de um botão de comando em um formulário;

object.Default [= booleano]

Default= 0 - False
1 - True

Quando esta propriedade de um Command Button estiver como True, o VB chamará o evento click sempre que a tecla Enter for pressionada.
Ex: Desejo que o botão cmdMultiplicar seja o default:

cmdMultiplicar.Default = True

Desejo saber se o botão cmdMultiplicar é default ou não, e o resultado será armazenado na variável inteira Estado%:

Estado% = cmdMultiplicar.Default

TABSTOP

Retorna ou dá o valor ao objeto indicado;

objeto.TabStop [= booleano]

TabStop = 0 - False
1 - True

Ex: Para alterar a ordem de tabulação dos botões em tempo de execução basta incluir estas linhas em algum procedimento:

```
cmdMultiplicar.TabStop = 0
cmdDividir.TabStop = 1
cmdSomar.TabStop = 2
cmdSubtrair.TabStop = 3
```

Para saber qual a ordem de tabulação do objeto txtNum1 e armazená-la em uma variável que conterá a ordem de tabulação do objeto; faça:

```
Ordem_Tab%= txtNum1.TabStop
```

No ambiente Windows, é comum mudarmos o foco entre os controles com a tecla Tab. Quando não quisermos que o usuário acesse determinado controle usando Tab, definimos a propriedade TabStop desse controle como False.

NAME

Nome que é dado ao objeto como referência para o código e definição de propriedades. Em tempo de execução, retorna o nome usado por um controle:

objeto.Name

Ex: Desejo exibir o Name do Formulário em um quadro de texto:

```
txtNome_Form.Text = frmCalculadora.Name
```

CAPTION

objeto.Caption [= string]

Determina o texto mostrado na barra de título do formulário, o texto dentro de um controle ou um título na barra de menu.

Ex: Alterar o Caption do botão Limpar após o seu uso, basta inserir esta linha no procedimento cmdLimpar_Click:

```
cmdLimpar.Caption = “Iniciar”
```

TEXT

Retorna o texto que está escrito na área de edição de um quadro de texto (TextBox), ou escreve um String nesta área;

objeto.Text [= string]

## FUNÇÕES

CSNG

Converte uma expressão para um número de precisão simples (Single).

CSng(expressão)

Está expressão pode ser qualquer número.
Existem outras funções de conversão semelhantes;

Cint - Inteiro (Integer)
CLng - Longo (Long)
CDbl - Dupla (Double)
CCur - Corrente (Currency)
CStr - Cadeia de caracteres (String)

Ex: Label1.Caption = conversor (Text1.Text), o número que for digitado no quadro de texto será convertido e mostrado no label1.
Text1.Text = “13,2575312”

| Conversores | Label1.Caption |
|---|---|
| CInt | 13 |
| CLng | 13 |
| CSng | 13,25753 |
| CDbl | 13,2575312 |
| CCur | 13,2575 |
| CStr | 13,2575312 |

STR$

Retorna como cadeia de caracteres (string) o valor de uma expressão numérica.

Str$ (número)

**MÉTODO**

SETFOCUS

Dá o foco ao objeto indicado;

objeto.SetFocus

Fixa o foco a um formulário ou controle. Somente pode ser usado para um formulário ou controle visíveis.

**VARIÁVEIS**

Variável é um local onde podem ser guardados dados com possibilidade de alteração em tempo de execução. O nome de uma variável pode ter até 255 caracteres, tem que começar com uma letra e tem que ser única. O nome pode conter números e sublinhados e não pode ser uma palavra reservada.

Existem vários tipos de variáveis, dependendo do tipo de dados que queremos que ela armazene.

| Tipo | Número de Bytes | Caracter | Faixa |
|---|---|---|---|
| Byte | 1 | | 0 a 255 |
| Boolean | 2 | | True ( -1 ) ou False ( 0 ) |
| Date | 8 | | 1/Jan/100 a 31/Dez/9999 |
| Integer | 2 | % | -32.768 a 32.767 |
| Long | 4 | & | -2.147.483.647 a 2.147.483.647 |
| Single | 4 | ! | -3,402823E38 a -1,401298E-45<br>1,401298E-45 a 3,402823E38 |
| Double | 8 | # | -1,79769313486232E308 a -4,94065645841247E-324<br>4,94065645841247E-324 a 1,79769313486232E308 |
| Currency | 8 | @ | -922,337,203,685,477.5808 a 922,337,203,685,477.5807 |
| String | variável | $ | Não se aplica |

Podemos usar certos caracteres para indicar o tipo da variável desejada, quando usa-la inicialmente. Por exemplo: i%, troco@. Como tipo básico, o VB usa o tipo Single, portanto se tivermos uma variável com o nome de Valor!, será o mesmo que deixar como Valor.

Formas de Declarar uma Variável

1 - Usar a variável onde desejar. Na linha de código onde for necessária usando um dos caracteres que identificam o tipo.

2 - Usar as declarações Dim, Static ou Global, alocando um espaço na memória para a variável.

```
Dim NomeVariável  As tipo
Static NomeVariável As tipo
Global NomeVariável  As tipo
```

**Escopo e Tempo de Vida das Variáveis**

Escopo são os pontos da aplicação de onde podemos acessar a variável. Existem 4 locais diferentes para declarar variáveis.

1 - Local, a variável será usada apenas pelo procedimento onde ela foi declarada.

2 - Em nível de Formulário, a variável poderá ser acessada por todos os procedimentos do formulário onde for declarada na seção geral (general) .

3 - Em nível de Módulo, a variável poderá ser acessada por todos os procedimentos do módulo.

4 - Em nível Global, toda aplicação poderá usar esta variável.

Toda vez que executamos um procedimento, suas variáveis locais são reinicializadas. Para que a variável retenha o seu valor, usamos a declaração Static. Mais adiante, veremos um exemplo desta declaração e das outras.

*FORMATAÇÃO DE TEXTO*

A função Str$, transforma um número em texto, mas não padroniza a sua apresentação. Caso necessitemos formatar um dado a ser exibido usamos a função;

```
Format$(expressão [,formato] )
```

* expressão = expressão numérica ou string a ser formatado.
* formato = a maneira como deverá ser mostrada a expressão.

*Formatando números:*

| Formato | 5 positivo | 5 negativo | 5 decimal |
|---|---|---|---|
| 0 | 5 | -5 | 1 |
| 0,00 | 5,00 | -5,00 | 0,50 |
| #.##0 | 5 | -5 | 1 |
| #.##0,0 | 5,0 | -5,0 | 0,5 |
| $#.##0;($#.##0) | $5 | ($5) | $1 |
| $#.##0,00;($#.##0,00) | $5,00 | ($5,00) | $0,50 |
| 0% | 500% | -500% | 50% |
| 0,00E+00 | 5,00E+00 | -5,00E+00 | 5,00E-1 |

Em “formato” o número 0 será mostrado ou trocado pelo caractere em sua posição, já o nirus (#) não será mostrado. Podemos inserir símbolos na função Format, como no exemplo: $ , % e E .

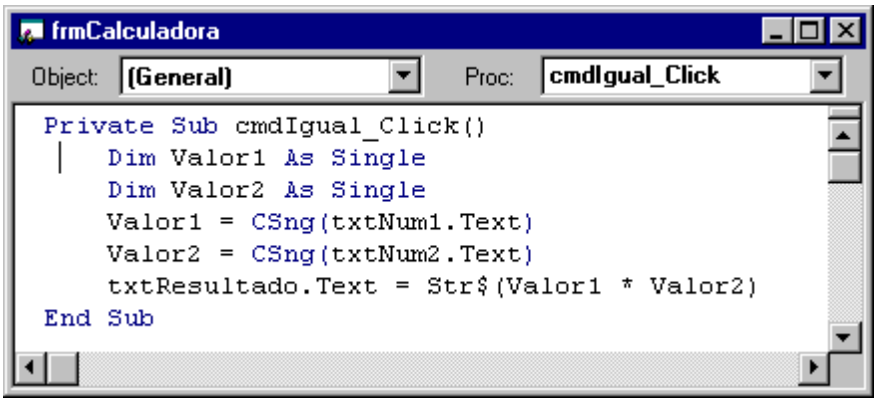
Formatando Data e Hora:

| Formato | Exibido |
|---|---|
| d/m/yy | 10/7/96 |
| dd-mm-yyyy | 01-Jun-1996 |
| dd-ddd | 02-dom |
| hh:mm AM/PM | 08:50 AM |
| h:mm:ss a/p | 8:50:20 a |
| d/m/yy h:mm | 03/12/95 9:30 |
| General Date | 06/09/96 9:40:18 |
| Long Date | Sexta, 9 de setembro de 1996 |
| Medium Date | 09-set-96 |
| Short Date | 09/09/96 |
| Long Time | 9:40:19 |
| Medium Time (12 horas) | 09:40 AM |
| Short Time (24 horas) | 09:40 |

MODIFICANDO A CALCULADORA

No formulário da calculadora, selecione o botão de comando cmdIgual e pressione a tecla Delete. Lembre-se que tínhamos um código associado a este objeto e agora que ele sumiu, para onde foi o código?

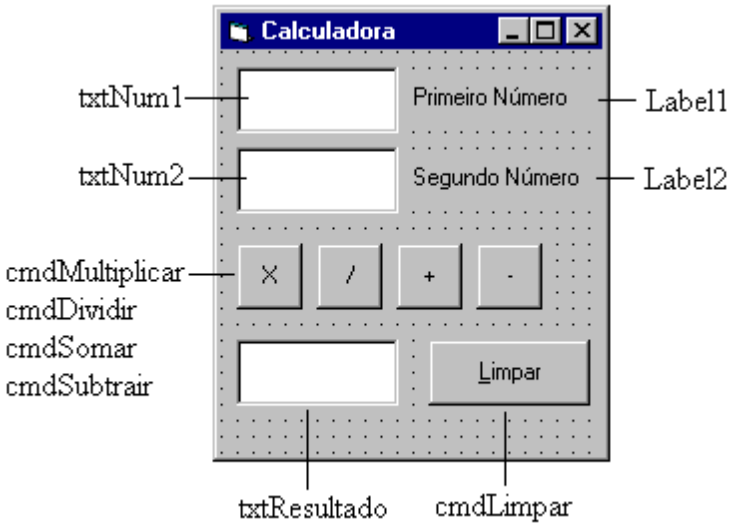
Chame a janela de código, indo até a janela de projeto, e clique em ViewCode. No quadro de objetos (object), procure a seção General, o código o cmdIgual estará lá como um procedimento geral.



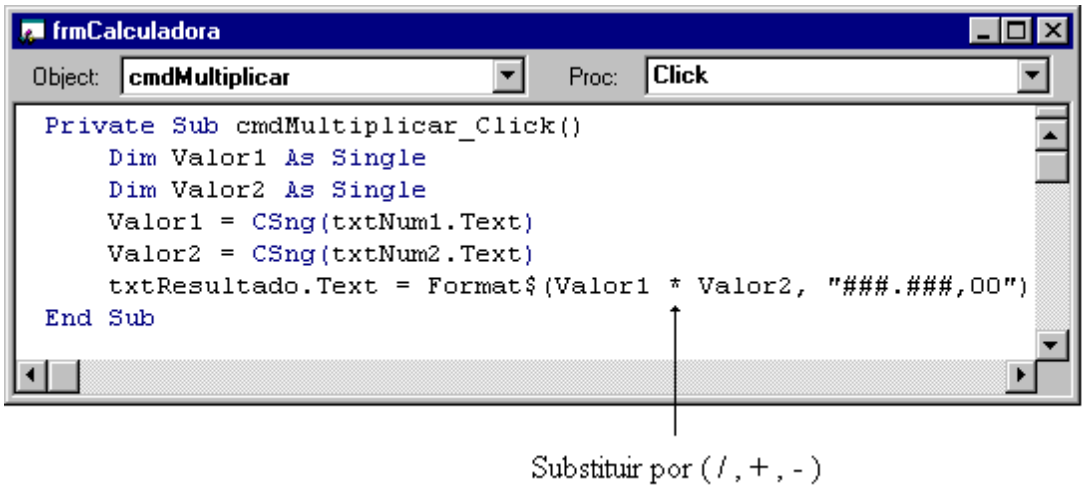

Procedimento geral é aquele que pode ser chamado por qualquer outro procedimento do formulário, funciona como uma sub-rotina. Ele não é como um procedimento associado a um objeto executado a partir de um evento, e sim, quando chamado.

Agora, deixe o formulário frmCalculadora como o exemplo a seguir:

Na figura aparecem as propriedades Name de cada objeto




Chame a janela de código dando um clique duplo no formulário ou na janela de projeto, View Code. Aparecerá então o procedimento Form_Load, vá até o quadro Object e selecione (General), no quadro Proc, procure o código do antigo cmdIgual (cmdIgual_Click) e altere o cabeçalho como no exemplo a seguir, observe que ao dar Enter, o procedimento que antes era geral, passou a ser associado a um novo objeto (cmdMultiplicar).




Na última linha já estamos usando a função Format$, para formatar o número a ser apresentado.

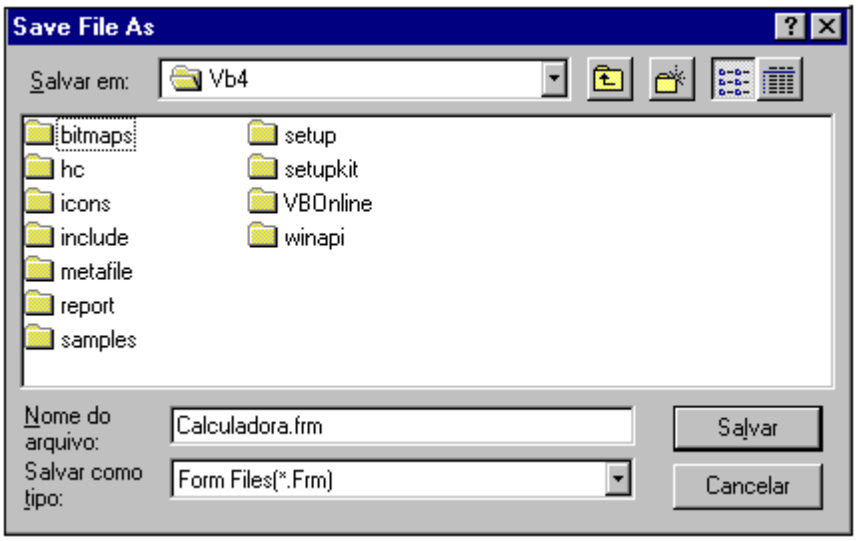
Selecione todo o texto, menos o cabeçalho e End Sub, e copie (Ctrl + C). Chame o procedimento para outro botão de operação, cole o texto e altere o operador correspondente.

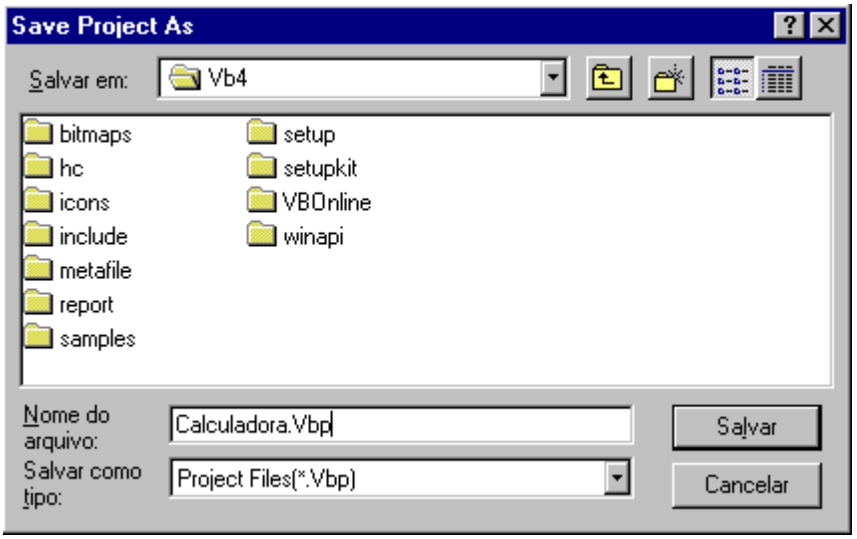
Teste os vários formatos de apresentação dos números, alterando a forma de apresentação da função Format$.

Um projeto em VB trabalha com vários arquivos. Um arquivo para cada formulário ou módulo, e um arquivo para o projeto.

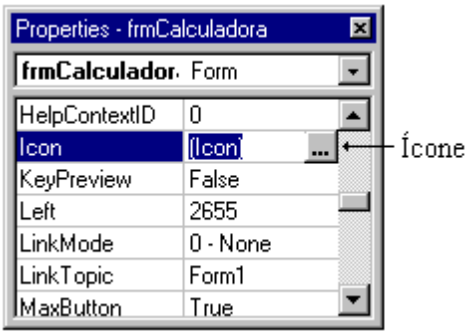
Vamos salvar o nosso projeto de calculadora.

No menu File, selecione Save Project, aparecerá o quadro de diálogo de Salvar do Windows pedindo para dar um nome ao arquivo de formulário, extensão *.frm,* dê o nome de calculadora.frm e clique em Salvar. A seguir, aparecerá o mesmo quadro pedindo para dar um nome ao arquivo de projeto, extensão *.vbp,* dê o nome de Calculadora.vbp e clique em Salvar. O projeto estará salvo.

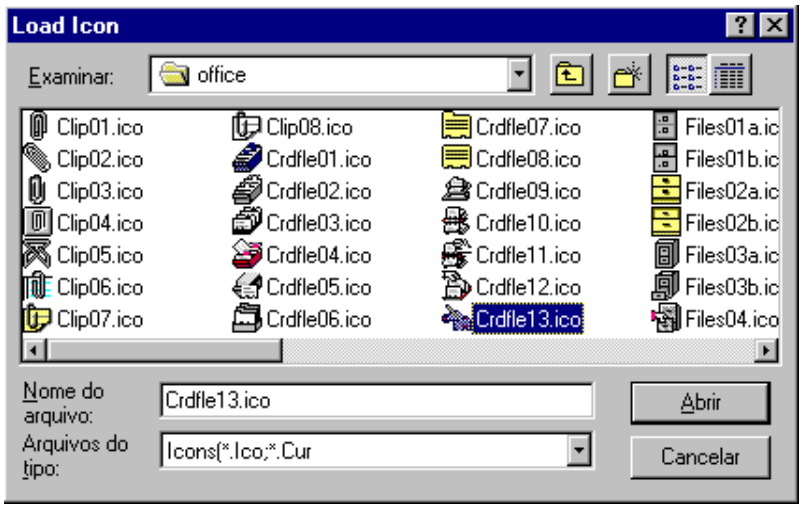


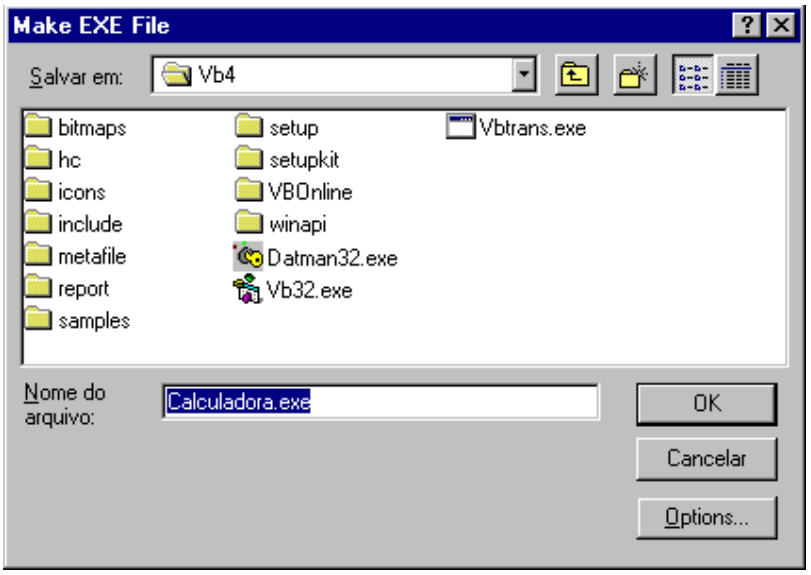

Antes de fazer do projeto um arquivo executável, vamos escolher um ícone para o nosso projeto ser representado no Windows. Selecione a propriedade Icon do formulário:




Clique no botão com reticências para termos acesso ao quadro para escolher um ícone, que será associado ao formulário;




Escolha um ícone e clique em Abrir. Quando o seu projeto aparecer no Windows, ele será representado por este ícone. Agora, vamos fazê-lo executável fora do VB, no Menu File, escolha Make EXE File..., aparecendo o quadro de diálogo para escolher o nome do arquivo executável:



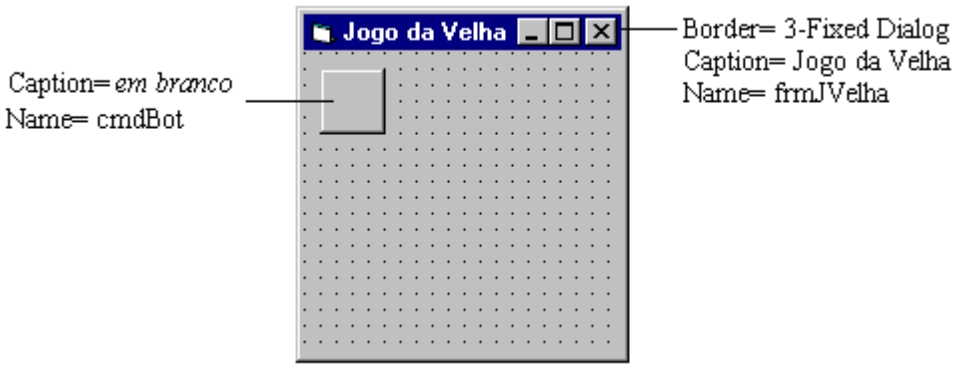

Escolhido o nome do arquivo executável, clique em Ok. Agora, você tem um programa executável em qualquer microcomputador que possua o sistema Windows 95, sem necessariamente ter o VISUAL BASIC instalado.

## Exemplo II - Jogo da Velha

Para iniciar um novo projeto, selecione New Project do Menu File. Caso você ainda não tenha salvo o seu projeto corrente, o VB abrirá as janelas para salvar o projeto. E só então iniciará o novo projeto.

Vamos iniciar um projeto de Jogo da Velha, onde o usuário irá jogar contra o computador que não "pensa" as suas jogadas, trabalhando com números aleatórios - e ao final da partida será mostrado um quadro de mensagem informando o ganhador. O objetivo é conhecermos as estruturas condicionais e de repetição, tão utilizadas nos programas.

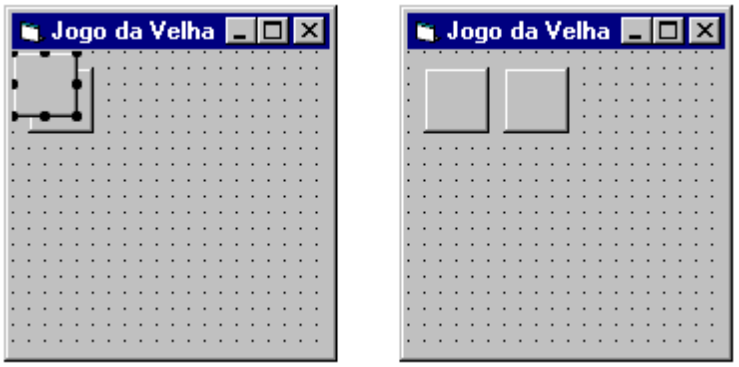
Insira um botão de comando no formulário dimensionando-o como um quadrado, e altere suas propriedades como mostra a figura:



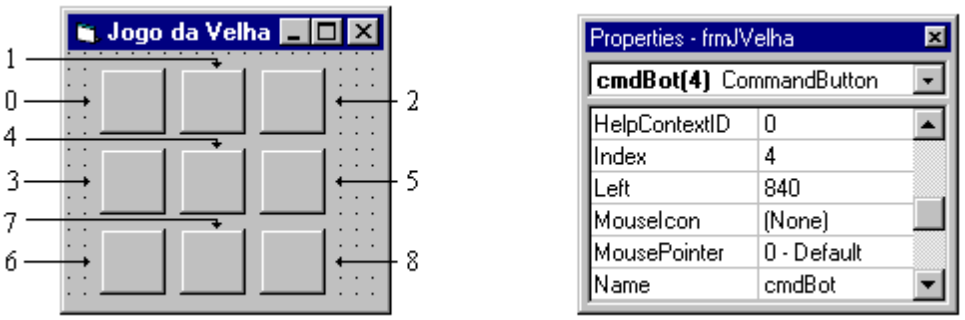

O nosso Jogo da Velha possui 9 botões iguais, todos irão executar a mesma rotina quando o usuário der um click em um deles. Para economizar trabalho, vamos criar um Array de controle (ordem de controles), onde todos esses controles possuirão o mesmo código, todos tendo o mesmo nome mas com índices diferentes.

Para criar um Array de controle, selecione o botão e copie-o (Ctrl + C) e depois cole-o (Ctrl + V). Quando for dada a ordem de colar, o VB abrirá um quadro de mensagem indicando que já existe um controle com o nome de cmdBot, e se você quer criar um Array de controle, responda Sim.

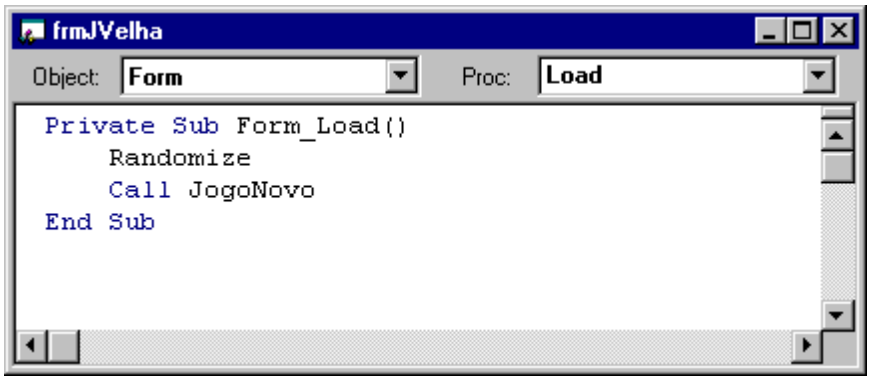

O botão de comando irá aparecer no canto superior esquerdo do formulário, depois é só arrastá-lo para a posição desejada.



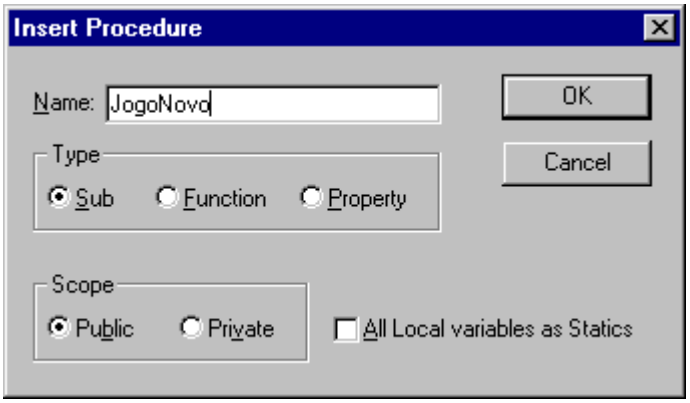

Para inserir os próximos botões, basta ir colando e o VB não perguntará mais sobre o Array de Controle. Posicione os botões da esquerda para a direita e de cima para baixo, pois assim o índice deles coincidirá com o do código, na hora da verificação. Se você observar a Janela de propriedades dos botões, notará que além do nome, eles possuem um índice, este índice está na propriedade Index de cada botão, caso você tenha errado as posições, basta corrigir alterando essa propriedade.



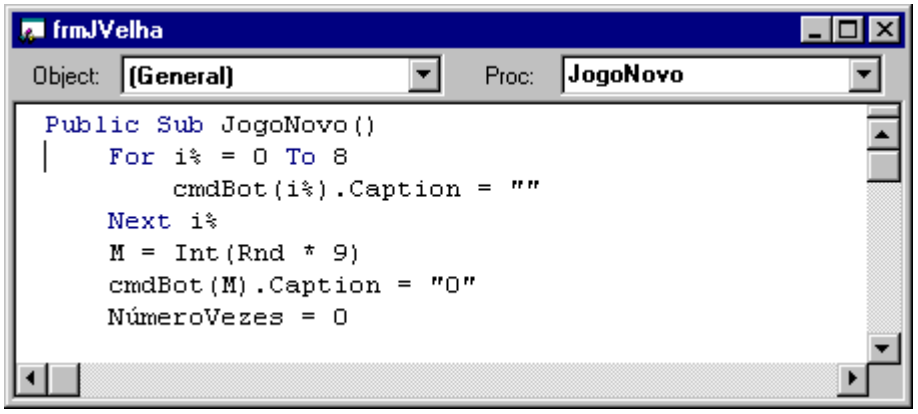

Quando o projeto iniciar, o formulário será carregado na memória , neste fato ocorre um evento Load. Neste evento colocamos um procedimento de inicio do jogo. Selecione o botão View Code para abrir a janela de Códigos e entre com as declarações Randomize e Call, responsáveis pela inicialização de números aleatórios e o carregamento de um procedimento - respectivamente.



No formulário temos a seção General, nesta seção colocamos os procedimentos e as variáveis que serão solicitados por todos os procedimentos do formulário. No nosso projeto, na seção General teremos o procedimento JogoNovo que dará início a um novo jogo, e a variável NúmeroVezes, que servirá para armazenar o número de vezes que jogamos - indicará se houve empate ou não. Para criar este procedimento, vá até o Menu Insert e escolha a opção Procedure..., aparecerá o seguinte quadro:




Selecione as opções Sub e Public e entre com o nome do procedimento, em seguida, clique em OK. Digite o código conforme figura a seguir:




A estrutura de repetição For... Next é utilizada aqui para apagar o conteúdo dos botões. A função Int e Rnd retornam, respectivamente, a porção inteira de um número, e um número aleatório entre 0 (inclusive) e 1.

A variável NúmeroVezes declaramos na seção General - Declarações, porque ela será utilizada em dois procedimentos distintos - JogoNovo e cmdBot_Click. O seu valor permanecerá até que o formulário seje retirado da memória ou o programa finalizado.

O próximo passo é fazer o código dos botões, dê um duplo click em qualquer um deles para a janela de código aparecer, e entre com o código conforme texto abaixo. A seguir, encontraremos o uso da estrutura condicional If... End If e a estrutura de repetição Do Until... Loop, conforme estudos anteriores. Procure entender as estruturas e tire todas as dúvidas.

```
Private Sub cmdBot_Click(Index As Integer)
    NúmeroVezes = NúmeroVezes + 1
   cmdBot(Index).Caption = "X"
   If cmdBot(0).Caption = "X" And cmdBot(1).Caption = "X" And cmdBot(2).Caption =
"X" Then GoTo MensX
   If cmdBot(3).Caption = "X" And cmdBot(4).Caption = "X" And cmdBot(5).Caption =
"X" Then GoTo MensX
   If cmdBot(6).Caption = "X" And cmdBot(7).Caption = "X" And cmdBot(8).Caption =
"X" Then GoTo MensX
   If cmdBot(0).Caption = "X" And cmdBot(3).Caption = "X" And cmdBot(6).Caption =
"X" Then GoTo MensX
   If cmdBot(1).Caption = "X" And cmdBot(4).Caption = "X" And cmdBot(7).Caption =
"X" Then GoTo MensX
   If cmdBot(2).Caption = "X" And cmdBot(5).Caption = "X" And cmdBot(8).Caption =
"X" Then GoTo MensX
   If cmdBot(0).Caption = "X" And cmdBot(4).Caption = "X" And cmdBot(8).Caption =
"X" Then GoTo MensX
   If cmdBot(2).Caption = "X" And cmdBot(4).Caption = "X" And cmdBot(6).Caption =
"X" Then GoTo MensX

   Do Until cmdBot(M).Caption = ""
      M = Int(Rnd * 9)
   Loop
   cmdBot(M).Caption = "0"
    If cmdBot(0).Caption = "0" And cmdBot(1).Caption = "0" And cmdBot(2).Caption =
"0" Then GoTo Mens0
    If cmdBot(3).Caption = "0" And cmdBot(4).Caption = "0" And cmdBot(5).Caption =
"0" Then GoTo Mens0
    If cmdBot(6).Caption = "0" And cmdBot(7).Caption = "0" And cmdBot(8).Caption =
"0" Then GoTo Mens0
    If cmdBot(0).Caption = "0" And cmdBot(3).Caption = "0" And cmdBot(6).Caption =
"0" Then GoTo Mens0
    If cmdBot(1).Caption = "0" And cmdBot(4).Caption = "0" And cmdBot(7).Caption =
"0" Then GoTo Mens0
    If cmdBot(2).Caption = "0" And cmdBot(5).Caption = "0" And cmdBot(8).Caption =
"0" Then GoTo Mens0
    If cmdBot(0).Caption = "0" And cmdBot(4).Caption = "0" And cmdBot(8).Caption =
"0" Then GoTo Mens0
    If cmdBot(2).Caption = "0" And cmdBot(4).Caption = "0" And cmdBot(6).Caption =
"0" Then GoTo Mens0
    If NúmeroVezes = 4 Then
         MsgBox "Partida Empatada", 64, "Empate"
```

```
            Call JogoNovo
            Exit Sub
        End If
    Exit Sub

MensX:
    MsgBox "Você Ganhou", 64, "Vencedor"
    Call JogoNovo
    Exit Sub

Mens0:
    MsgBox "Eu Ganhei", 64, "Vencedor"
    Call JogoNovo
    Exit Sub

End Sub
```

Terminando de digitar este procedimento, salve o formulário e o projeto. E execute pressionando F5, ou clique sobre o botão  . Teste o programa e observe o quadro de mensagem exibido no final.

O Windows possuí quadros padronizados de mensagem que servem para emitir aviso e recolher opções de tratamento dessas mensagens.

Estes quadros são fáceis de criar no VB com a declaração ou função MsgBox. MsgBox será uma declaração, quando não tratamos a resposta do usuário, e será uma função, quando esta resposta é tratada.

Para construir um Quadro de Mensagem, use o seguinte padrão:

Declaração - MsgBox mensagem, tipo, título
Função - MsgBox (mensagem, tipo, título)

Onde:

mensagem - expressão mostrada dentro do quadro de diálogo.
tipo - somatória de números, conforme o que queremos que seja exibido no Quadro de Mensagem, seguindo a tabela a seguir.
título - título do Quadro de Mensagem (barra de título).

Argumento tipo para a Declaração/Função MsgBox

| Valor | Significado |
|---|---|
| 0 | Somente o botão de OK |
| 1 | Botões de OK e Cancelar |
| 2 | Botões Anular, Repetir e Ignorar |
| 3 | Botões Sim, Não, Cancelar |
| 4 | Botões Sim, Não |
| 5 | Botões Repetir e Cancelar |
| 16 | Sinal de Stop |
| 32 | Sinal de Pesquisa |
| 48 | Sinal de Aviso |
| 64 | Ícone de Informação |
| 0 | Primeiro botão com foco |
| 256 | Segundo botão com foco |
| 512 | Terceiro botão com foco |

Teste o projeto alterando o valor de tipo para MsgBox, faça a sua soma escolhendo um item de cada seção.

Agora, vamos alterar o nosso projeto para que ele nos pergunte, ao final da partida, se queremos jogar novamente ou finalizar o programa. Para isso usaremos MsgBox como função, o que nos retornará o valor do botão acionado. Altere o procedimento cmdBot_Click como abaixo:

```
MensX:
    Resposta$ = MsgBox( "Você Ganhou, Deseja Jogar Novamente?", 36, "Vencedor")
    If Resposta$ = 6 Then
        Call JogoNovo
    Else
        End
    End If
    Exit Sub

Mens0:
    Resposta$ = MsgBox ("Eu Ganhei, Deseja Jogar Novamente?", 36, "Vencedor")
    If Resposta$ = 6 Then
        Call JogoNovo
    Else
        End
    End If
    Exit Sub

End Sub
```

A variável Resposta$ (declarada implicitamente como String), conterá a resposta do usuário que segue o padrão da tabela abaixo;

| Valor | Significado |
|---|---|
| 1 | Botão OK foi pressionado |
| 2 | Botão Cancelar foi pressionado |
| 3 | Botão Anular foi pressionado |
| 4 | Botão Repetir foi pressionado |
| 5 | Botão Ignorar foi pressionado |
| 6 | Botão Sim foi pressionado |
| 7 | Botão Não foi pressionado |

No nosso caso, o programa verificará se o botão Sim foi pressionado, em caso afirmativo, iniciará novo jogo, senão finalizará.

## DECLARAÇÕES

For ... Next

Repete um grupo de instruções num determinada número de vezes.

**For** contador = início **To** fim **[ Step** incremento **]**
    linhas de instruções
**[ Exit For ]**
**Next** contador

If ... Then ... Else

Permite a execução condicional, baseada na avaliação de uma expressão

**If** condição **Then**
    thenpart - se condição for verdadeira
**[Else**
    elsepart - se condição for falsa**]**
**End If**

Do Until ... Loop

Repete um bloco de declarações até a condição tornar-se verdadeira.

**Do Until** condição
    linhas de instruções
**[Exit Do]**
    linhas de instruções
**Loop**

Call

Transfere o controle do fluxo de programa para um procedimento **Sub**

**Call** nome do procedimento

Randomize

Inicializa o gerador de números aleatórios (randômicos).

**Randomize** [ número ]

número - pode ser qualquer expressão numérica válida que inicializará o gerador. Se for omitido, o gerador é inicializado com a função **Timer** que retorna a data e hora atual do sistema.

## FUNÇÕES

Int

Retorna a porção inteira de um número.

**Int** (número)

Rnd

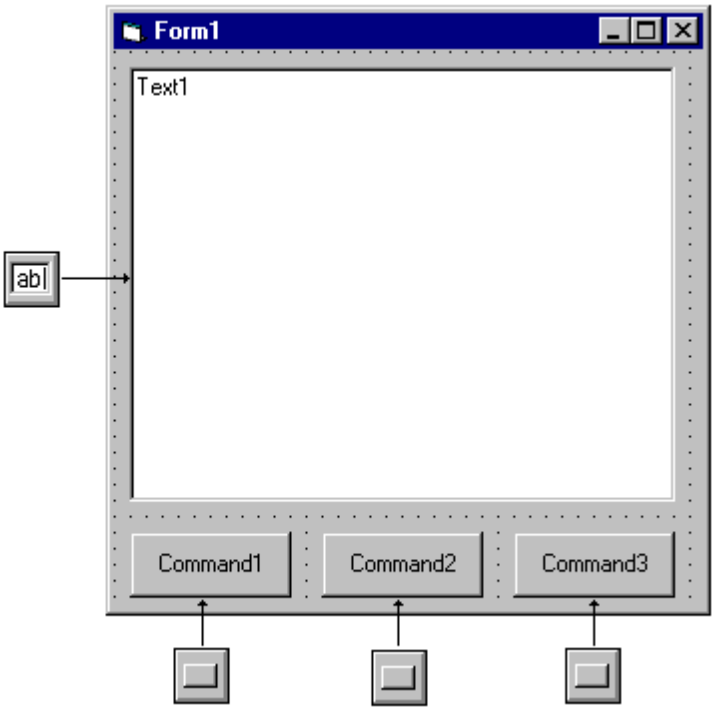
Retorna um número aleatório.
Maior ou igual a zero e menor que um ( 0<= Rnd < 1).

**Rnd**

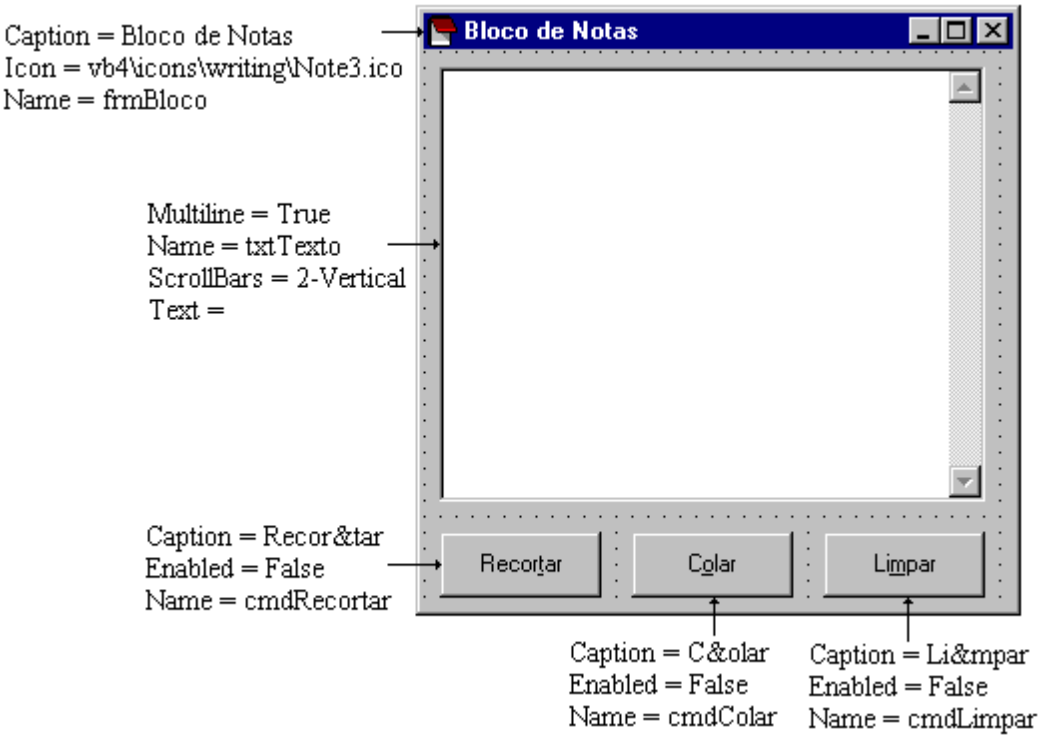
## Exemplo III - Bloco de Notas

O nosso próximo projeto será um editor de texto simples do tipo caractere, com ele poderemos alterar o tipo e tamanho da fonte utilizada em todo o texto, recortar, colar e copiar partes selecionadas, e salvar e abrir nosso texto em um arquivo de acesso seqüencial.

Monte o formulário conforme o exemplo:



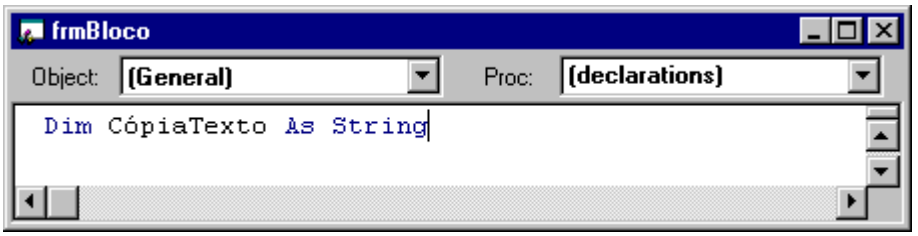

Agora, vamos alterar as propriedades dos 5 objetos:



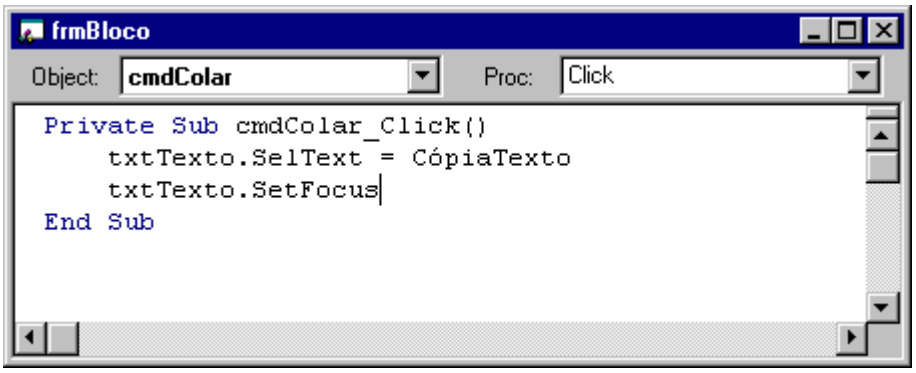

No nosso quadro de texto (txtTexto) temos a propriedade Multiline = True para permitir que este quadro tenha várias linhas, e a propriedade ScrollBars = Vertical, para possibilitar a paginação destas linhas quando ultrapassarem a área do quadro.

Os botões de comando tem a propriedade Enabled = False para tornar o botão desabilitado ( cinza claro ), e o usuário não tenha acesso à sua rotina. Esta propriedade será mudada em tempo de execução quando tivermos algum texto a ser Recortado, Colado ou Limpo.

Vamos ao Código:

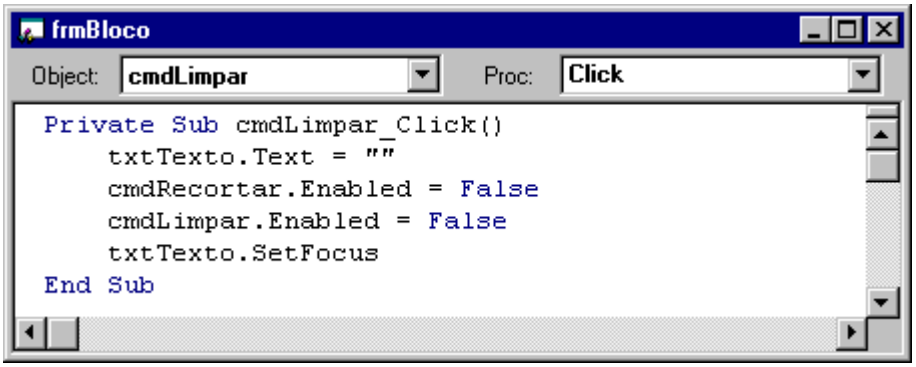



Declare a variável (CópiaTexto) que conterá o texto que foi Recortado ou Copiado.



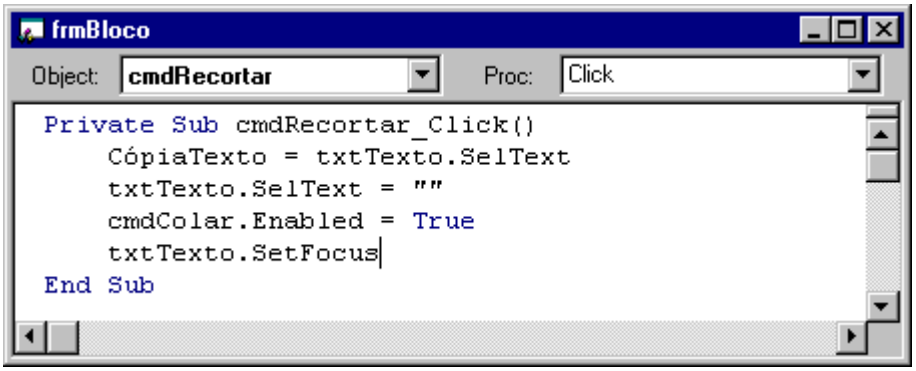

cmdColar

Copia a variável CópiaTexto para o quadro de texto no local do cursor ou área selecionada e devolve o foco para o quadro de texto, se o foco não fosse devolvido ele ficaria com o botão que foi acionado.



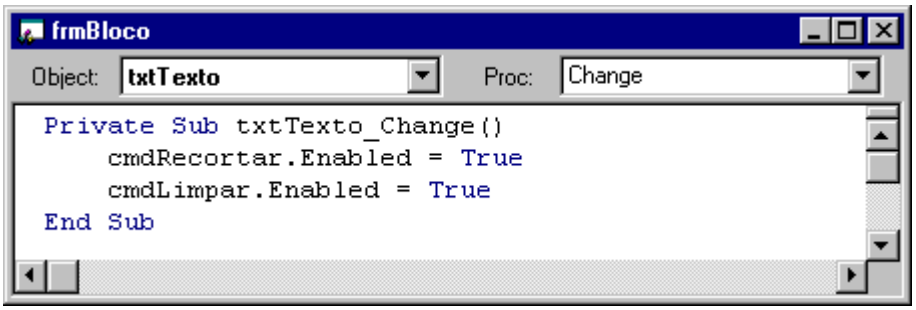

cmdLimpar

Limpa o quadro de texto, limpando a propriedade Text e desabilita os botões cmdRecortar e cmdLimpar. O botão cmdColar não é desabilitado porque ainda existe conteúdo na variável CópiaTexto, que poderá ser colado.




cmdRecortar

Atribui o texto selecionado, propriedade SelText, à variável CópiaTexto, limpa o texto selecionado e habilita o botão cmdColar.

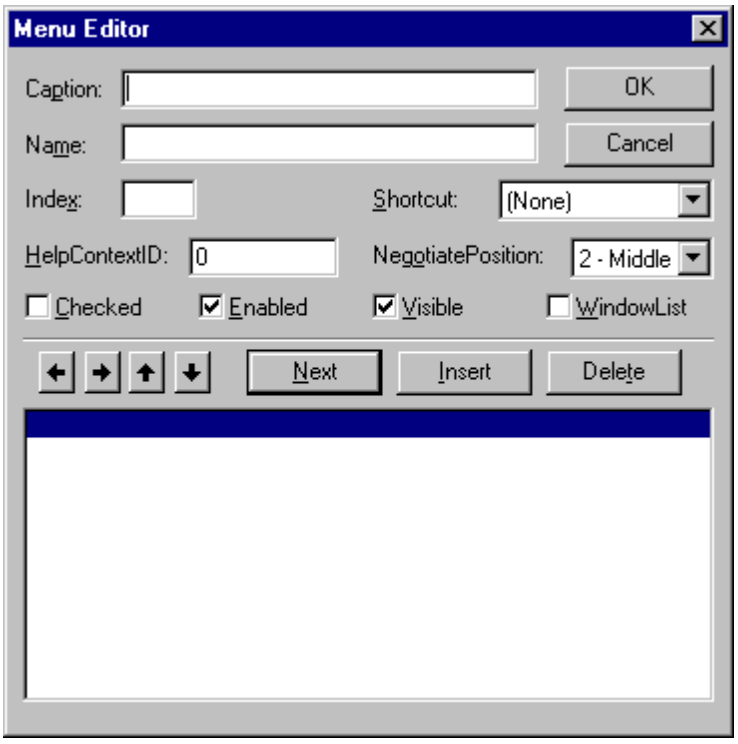



Salve o Formulário, o Projeto e depois execute e teste o funcionamento.

Em nosso Bloco de Notas, as opções de edição de texto estão na forma de botões, mas no Windows essas opções estão na forma de Menu. Neste caso, vamos agora trabalhar com menu e transferir o código dos botões para as opções do menu que iremos construir. Cada item de menu também é um objeto e portanto, também possui evento, código e propriedades.

## CRIANDO MENUS

Selecione o formulário ( frmBloco ) e escolha a opção Menu Editor... do menu Tools, ou clique no botão  (acessa o Menu Editor) da barra de ferramentas, aparecerá um quadro de diálogo para construirmos nosso menu.



O Quadro Menu Editor possui as seguintes partes:

**Caption**. O texto que aparecerá escrito no menu ou item de menu.

Para criar uma barra separadora em seu menu, basta digitar um hífen ( - ).
Para o acesso por teclado usamos o *e* comercial (&) antes da letra que queremos que seja o atalho, como nos botões.

**Name**. Contém o nome que será dado ao objeto que o identificará nas linhas de código.

**Index**. Número atribuído ao objeto para identificá-lo caso seje usado como um control array. Anteriormente usamos control array em botões e o Index foi determinado automaticamente, mas para itens de menu, teremos que determiná-los manualmente.

**Shortcut**. Uma drop-down lista de onde poderemos escolher a tecla de atalho para o item. Exemplo: Colar = Ctrl + V.

**HelpContextID**. Contém um valor numérico único que será usado para encontrar uma referência do objeto no arquivo de help.

**NegotiatePosition**. Determina a posição em que o menu irá aparecer quando objetos de outras aplicações estiverem ativos no formulário.

**Checked**. Seleciona se você quer que apareça uma marca de check antes do item de menu.

**Enabled**. Seleciona se você quer que o item de menu responda aos eventos ou não. Desabilita ou habilita o item de menu.

**Visible**. Seleciona se você quer que o item de menu esteja visível ou não.

 Use esses botões para mudar o nível do item no menu, criando submenus. Podem ser criados até quatro níveis de submenus.

 Use esses botões para mudar a posição do item de menu.

Botões:

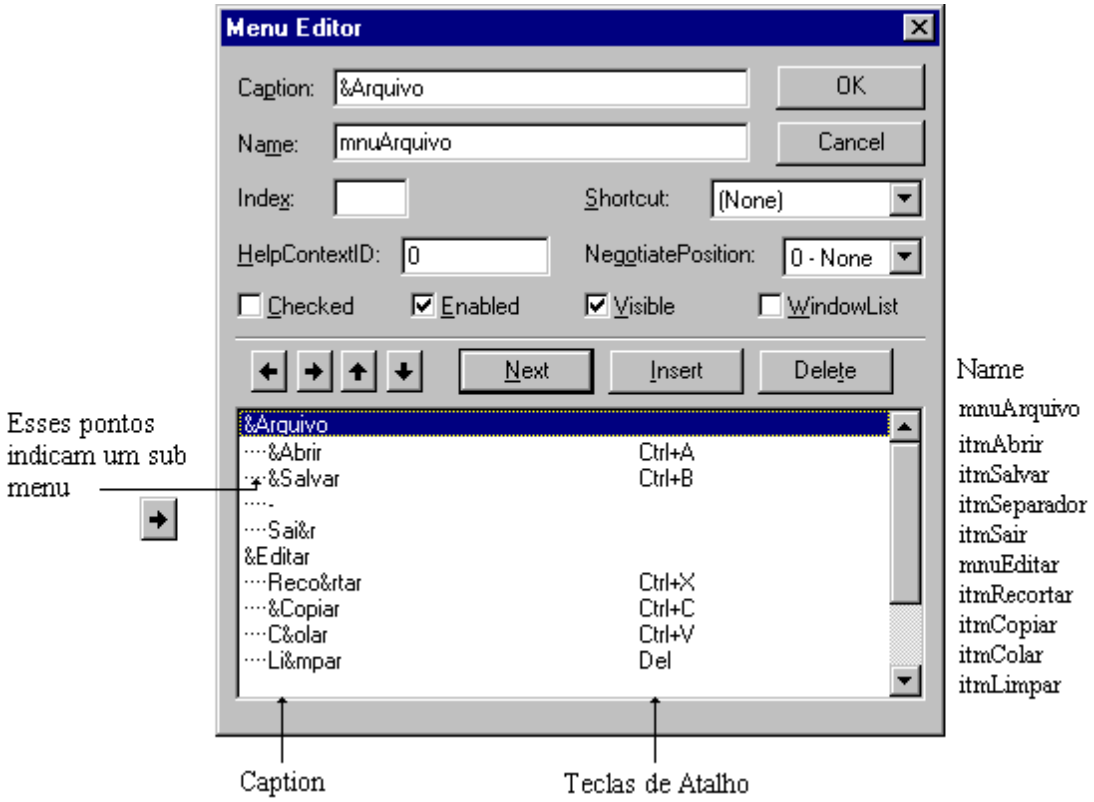
**Next** Move a seleção para a próxima linha.

**Insert** Insere uma linha acima da linha atualmente selecionada.

**OK** Fecha a Menu Editor e aplica todas as mudanças efetuadas.

**Cancel** Fecha a Menu Editor e cancela as mudanças efetuadas.

O nosso menu deverá ficar da forma mostrada abaixo.

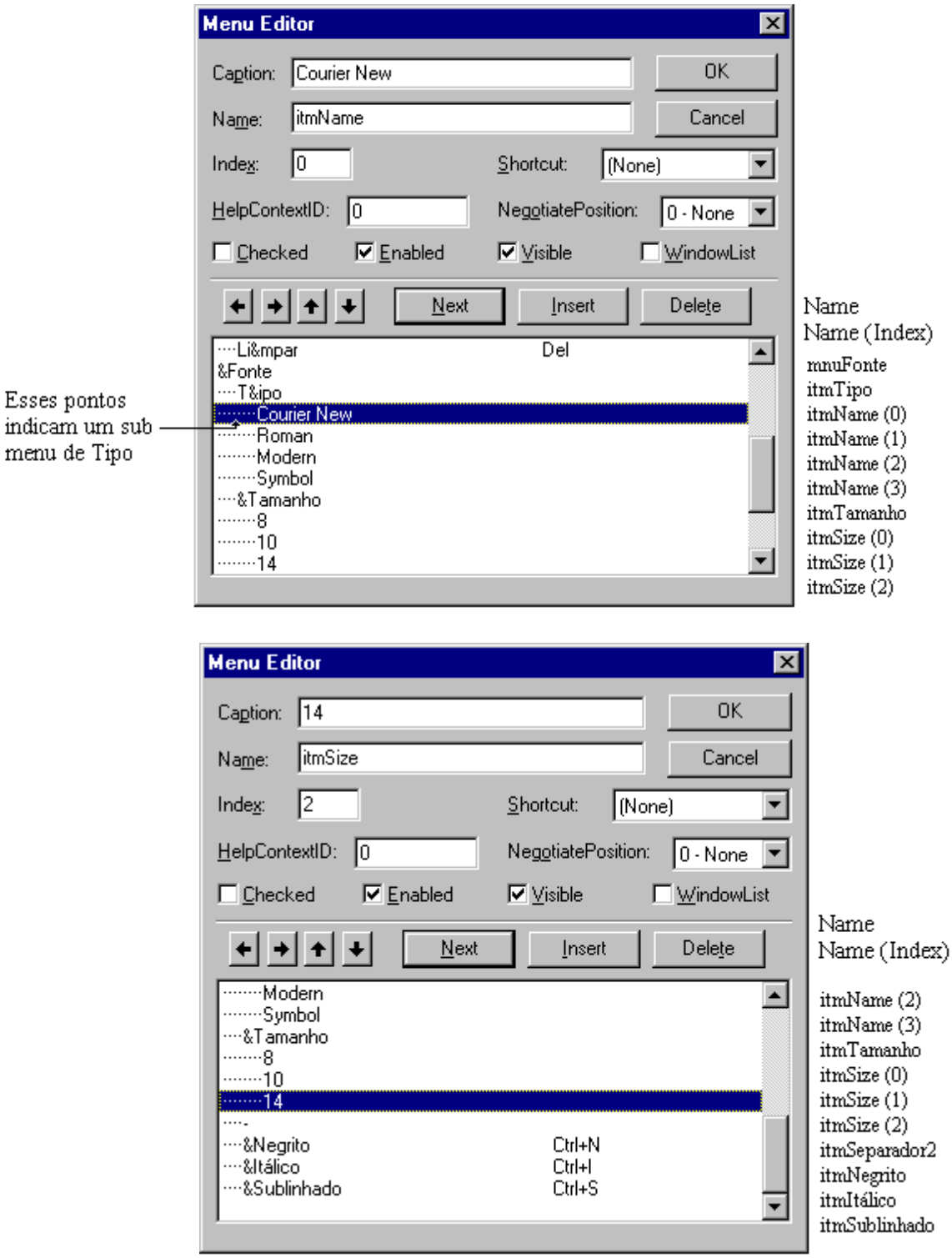



Para os itens: itmRecortar, itmCopiar, itmColar e itmLimpar, deixe a propriedade Enabled desabilitada. Clique em OK, e verifique o formulário, se você der um clique em alguma opção de menu, aparecerão os itens, e se der um clique em algum item de menu, aparecerá a janela de código deste item.

O nosso Bloco de Notas terá a opção de alterarmos o nome da fonte, o tamanho e a aparência das letras através de menu.

No menu, cada item de nome da fonte ou tamanho, terá a mesma função, ou seja, mudar as propriedades do txtTexto. Neste caso, criamos um mesmo procedimento para vários objetos utilizando para isso o recurso de Control Array. No projeto de jogo da velha usamos Control Array para os botões, e automaticamente o Index foi incrementado, para o menu, esse Index teremos que fazer manualmente.

Então vamos acessar novamente a Menu Editor... e complementá-lo.

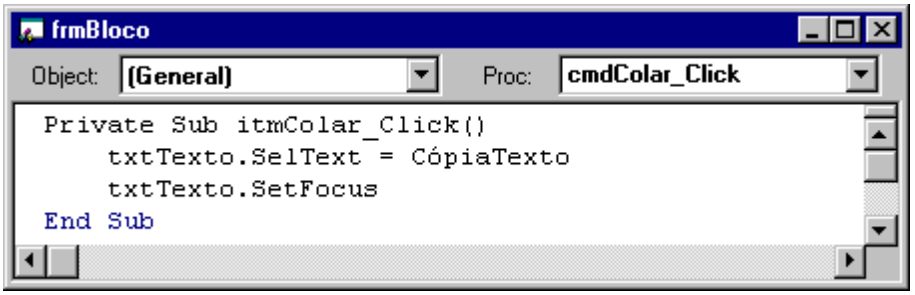



Agora, nosso menu está pronto, clique em OK e verifique no formulário a apresentação. Note que nos itens Tipo e Tamanho aparece uma seta para a direita que indica outro menu de opções.

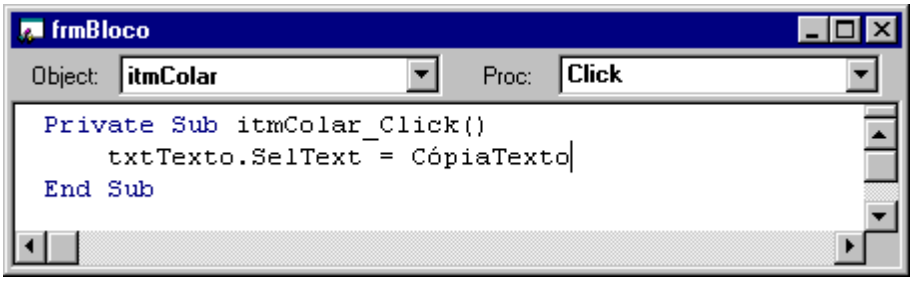
Nós tínhamos rotinas para Copiar, Recortar e Limpar mas na forma de botões, agora que temos na forma de menu, não precisamos mais dos botões. Vá até o formulário, selecione cada um desses botões e delete-o, ao fazer isso, o código associado irá para a seção general do formulário.

Selecione cada uma destas rotinas e altere o seu nome para que elas estejam associadas ao menu.

Nós não precisaremos mais devolver o foco para o txtTexto, porque ao fazer uma seleção no menu, ele desaparece, então o foco volta para o "único" objeto do formulário - txtTexto.

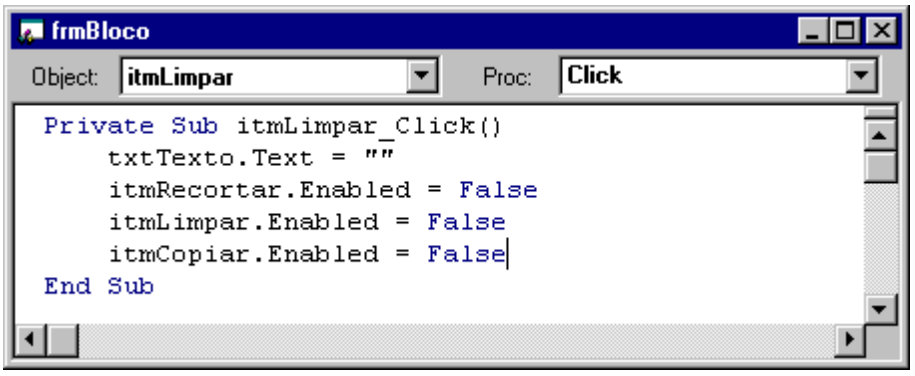


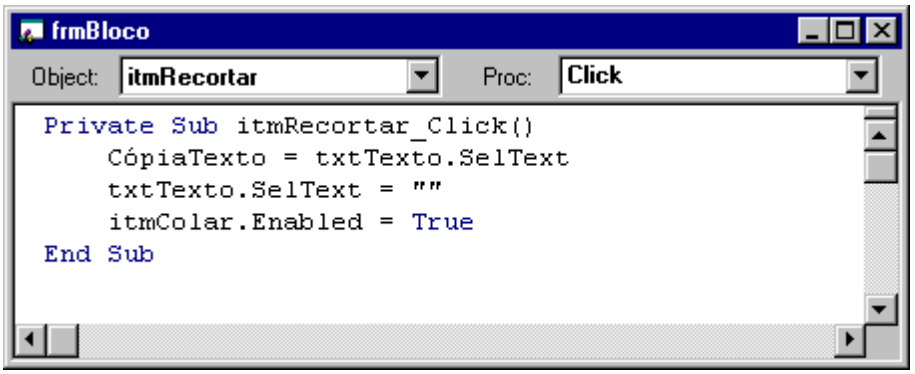

Após alterar o nome do procedimento, tecle a seta de direção para baixo e o VB irá alterar o nome do objeto (Object) e o Evento (Proc)

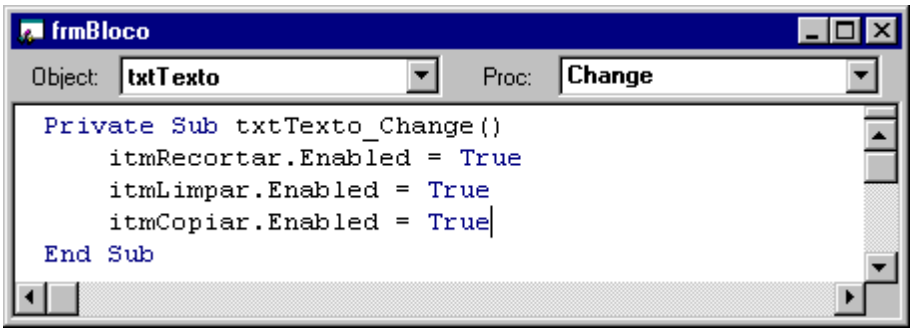



Faça o mesmo para os outros dois "ex botões", alterando o código aonde são feitas referências a estes botões.

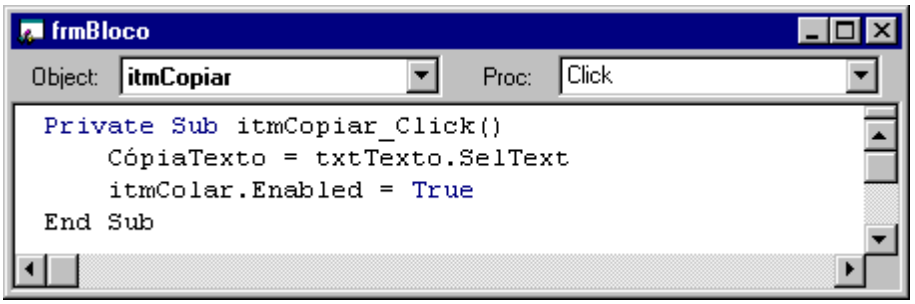
Note que teremos que acrescentar mais uma linha para itmCopiar.





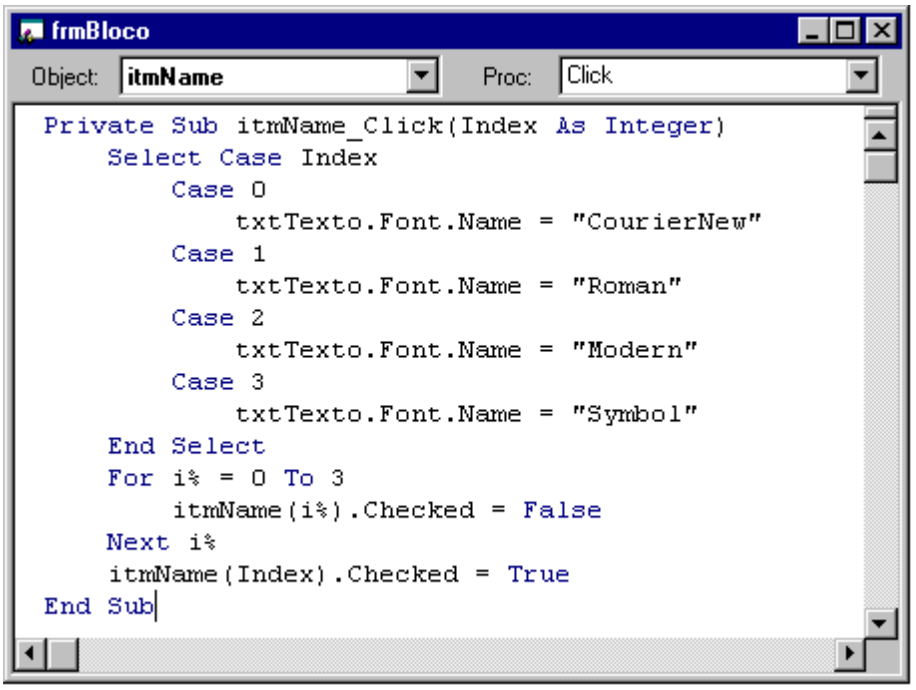




Para construir o procedimento itmCopiar_Click, utilize o recurso de copiar (Ctrl+C), o procedimento itmRecortar_Click, e colar (Ctrl+V). Logo após, deletar a linha que apaga o texto em txtTexto.




Salve e execute o projeto, verifique se os itens do menu Editar, os únicos que funcionam, ficam habilitados e desabilitados.

Faremos agora o código para os outros objetos.



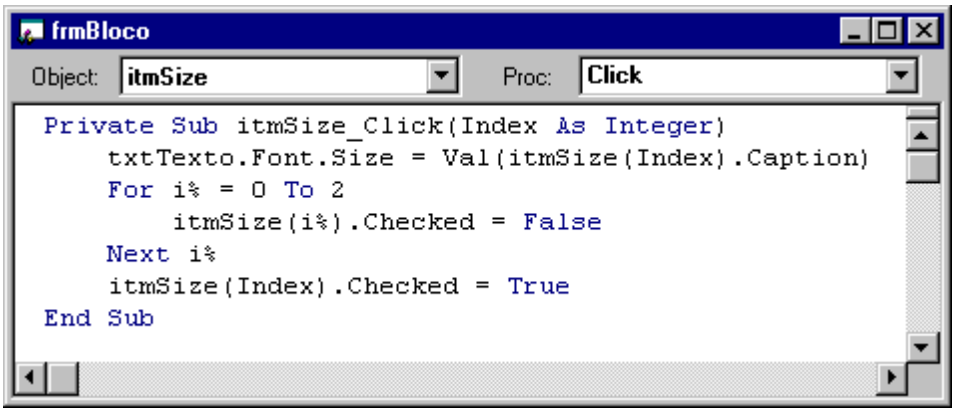

Quando damos um clique num item de Nome da Fonte, é iniciado o procedimento itmName_Click, e o procedimento recebe o Index do item acionado, esse Index é armazenado em uma variável de nome Index - (Index As Integer). Todo texto é exibido no novo formato e a marca de verificação (Checked) aparece ao lado do nome da fonte selecionada.

A declaração Select Case executa diferentes blocos de declarações dependendo do valor do Index.

Estrutura:

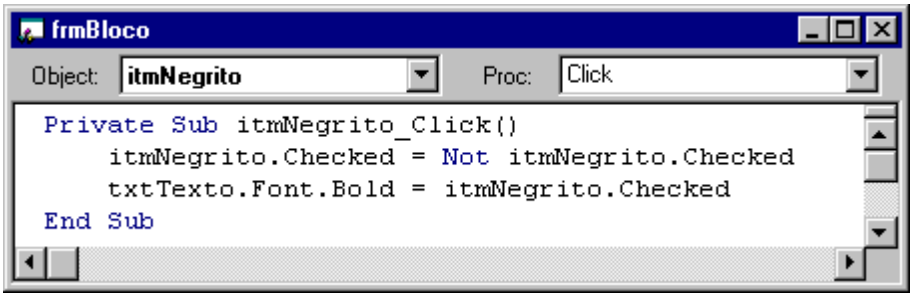
**Select Case** palavra teste
**Case** lista de palavras 1
declarações 1
**Case** lista de palavras 2
declarações 2
**End Select**

A declaração For...Next retira o Check do item anteriormente selecionado, é mais fácil retirar de todos, a ter que procurar qual o item que tem o Check e depois retirá-lo. E depois colocamos Check no item que foi clicado - itmName(Index).Checked = True.



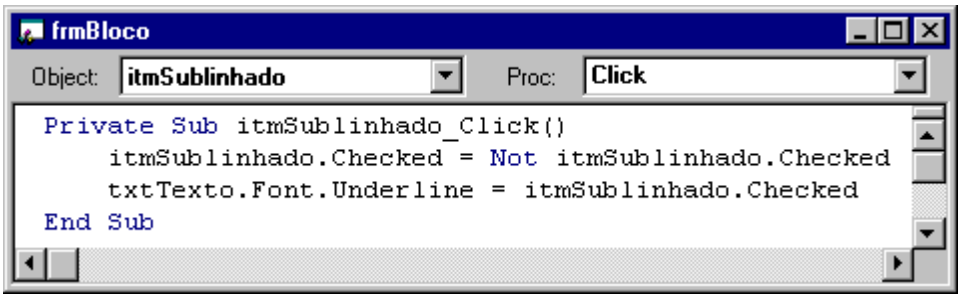

Para o procedimento itmSize_Click podíamos também usar a declaração Select Case, mas no exemplo é utilizada a propriedade Caption do item selecionado para alterar o tamanho da fonte.

Os itens Negrito, Sublinhado e Itálico quando estiverem selecionados, deverão apresentar um Check ao seu lado. Esta propriedade será utilizada para alterar ela mesma, e depois alterar a apresentação da fonte.

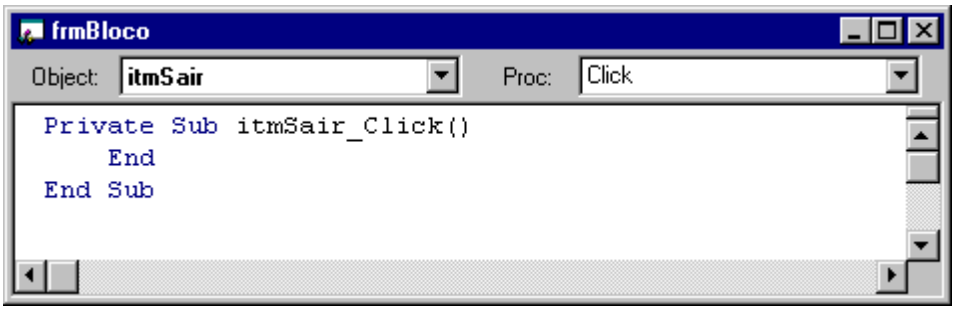



No início itmNegrito.Checked = False, quando o usuário der um clique em itmNegrito, o valor da propriedade será o inverso do que era - itmNegrito.Checked = True, e vice-versa, utilizando-se o operador lógico Not.

- mesmo é utilizado para Itálico e Sublinhado.




Finalize a execução com itmSair_Click.



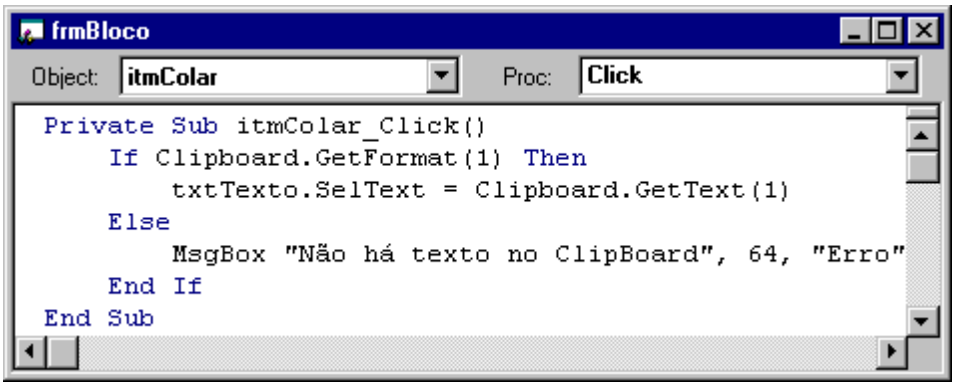

Salve e execute o projeto testando todos os itens, somente o menu Arquivo com as opções Abrir e Salvar, ainda não estarão ativos.

Nós utilizamos uma variável - CópiaTexto - para armazenar o texto que foi Recortado ou Copiado. Agora iremos utilizar a área de transferência do Windows, ou seja, o objeto **ClipBoard**.

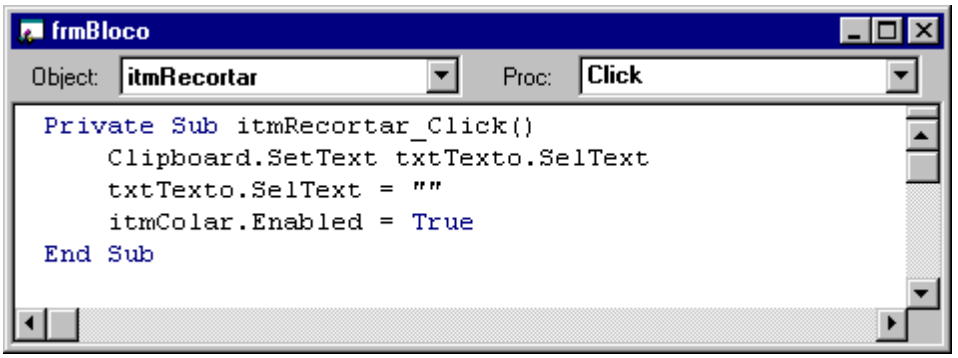
Existem objetos que não são incorporados ao formulário, mas comportam-se como tal, pois possuem propriedades e métodos associados, o ClipBoard é um deles. Ele possui os seguintes métodos:

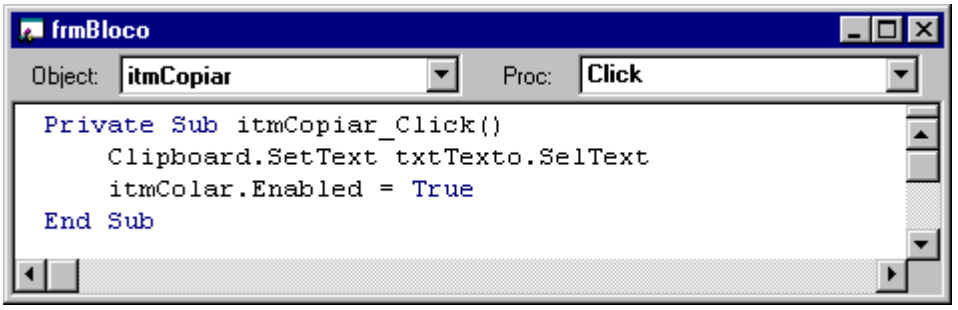
| | |
|---|---|
| Clear | - Limpa o conteúdo do ClipBoar |
| GetData | - Retorna um gráfico do ClipBoard |
| GetFormat | - Retorna um valor indicando qual o tipo de dado do ClipBoard |
| GetText | - Retorna um texto do ClipBoard |
| SetData | - Grava no ClipBoard um elemento gráfico |
| SetText | - Grava no ClipBoard um texto |

Para trabalhar com o ClipBoard em nosso projeto Bloco de Notas, altere os seguintes procedimentos:

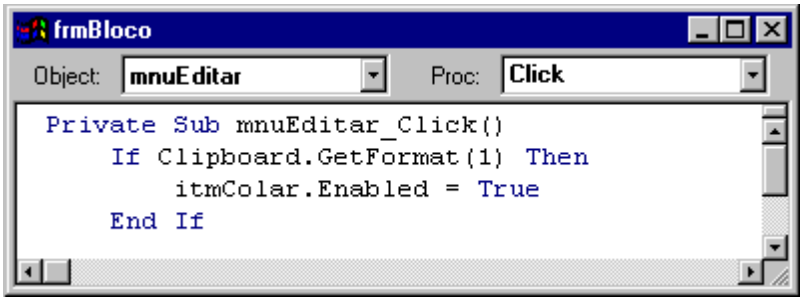



O procedimento itmColar_Click verifica se há realmente um texto no ClipBoard antes de Colar no Quadro de Texto txtTexto, caso não tenha um texto, é exibida uma mensagem de erro.




Para obter um melhor resultado, acrescente no procedimento mnuEditar_Click a verificação da existência ou não de texto no ClipBoard.

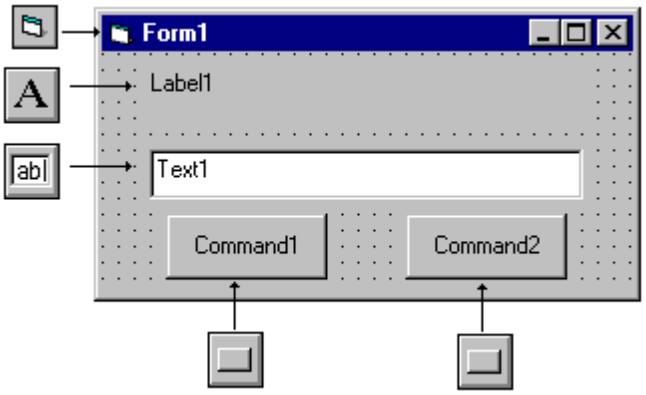


Salve e Execute o projeto. Para testar o funcionamento do ClipBoard, abra um editor de texto enquanto executa o projeto, copiando e colando textos entre eles. Faça um desenho no Paint do Windows, Copie e tente colar no Bloco de Notas, e verá que a mensagem "Não há texto no ClipBoard" aparecerá.

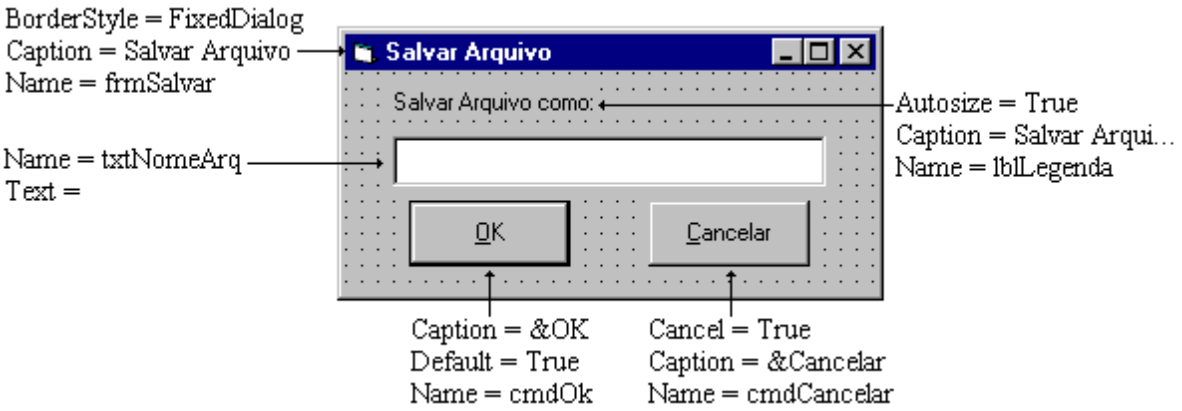
## SALVANDO E ABRINDO ARQUIVOS

Os itens Abrir e Salvar do menu Arquivo quando selecionados em aplicações para Windows, abrem outras janelas. Em nosso projeto de Bloco de Notas estas opções também irão abrir outras janelas ou formulários.

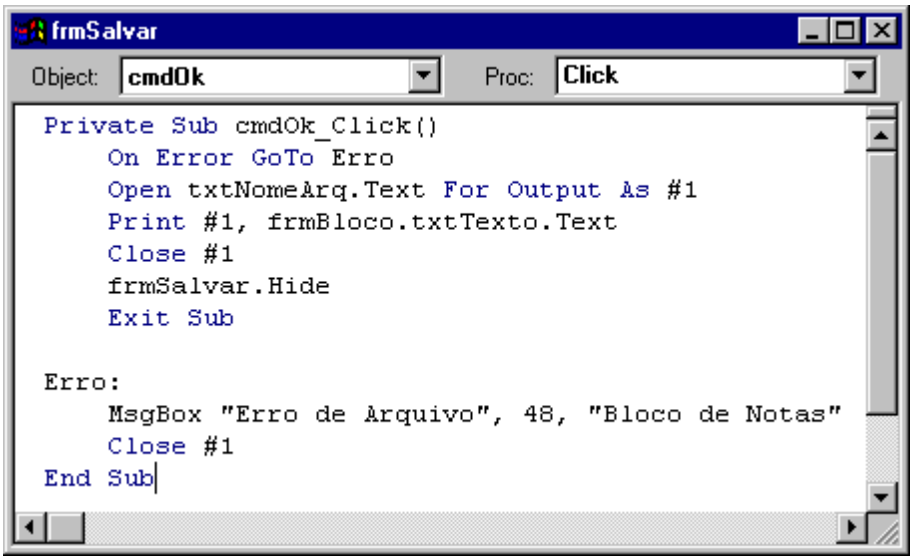
Primeiro vamos criar a janela de Salvar Arquivo. Por enquanto nosso projeto possui apenas um formulário chamado frmBloco, para criar mais um formulário - frmSalvar, escolha a opção Form do menu Insert ou dê um clique no botão  , na barra de ferramentas, para inserir um novo formulário ao projeto. Adicione neste novo formulário os objetos como mostrado abaixo.




Altere as propriedades dos objetos:



Digite o código para os botões OK e Cancelar:




A declaração **Open txtNomeArq.Text For Output As #1**, é utilizada para abrirmos um arquivo do tipo seqüencial. Sua sintaxe é a seguinte:

**Open** arquivo **For** modo **As** #número

Onde;

Arquivo - nome do arquivo a ser aberto

Modo - a maneira como o arquivo será aberto. Que pode ser:

Append (Adicionar): Adiciona mais conteúdo no final de um arquivo do tipo seqüencial.

Input (Entrada): Abre um arquivo do tipo seqüencial para leitura.

Output (Saída): Abre um arquivo do tipo seqüencial para escrita.

Random (Aleatório): Abre um arquivo do tipo de acesso aleatório, para leitura ou gravação

Número - Associa um número ao arquivo como referência para a aplicação. Pode variar de 1 até 511. Ou seja, podemos ter até 511 arquivos abertos ao mesmo tempo.

A declaração **Print #1, frmBloco.txtTexto.Text**, escreve o conteúdo do quadro de texto txtTexto do formulário frmBloco no arquivo que foi aberto como número 1. Sua sintaxe é a seguinte:

**Print** # número**,** expressão

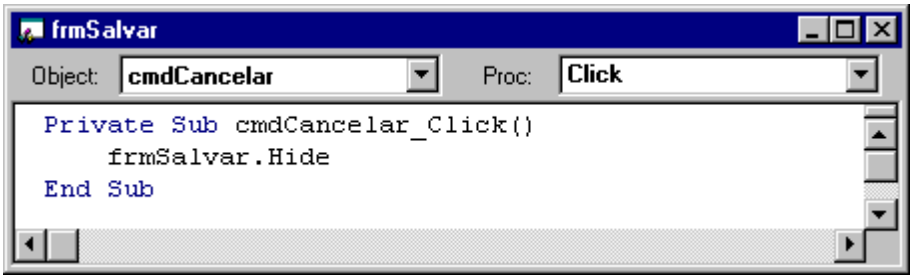
Número - número com o qual o arquivo que queremos escrever nele, foi aberto na declaração Open.

Expressão - cadeia de caracteres, números ou não, que serão escritos no arquivo.

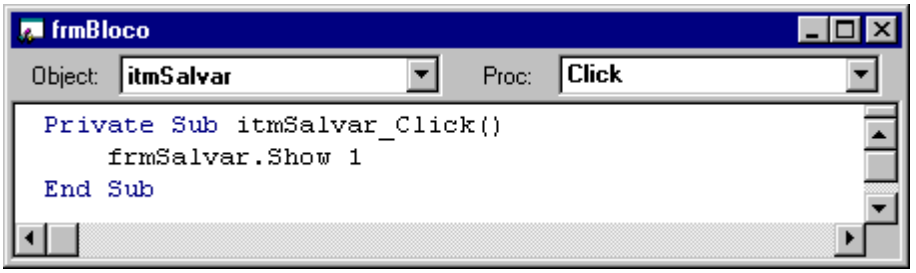
Com a declaração **Close #1**, fechamos nosso arquivo após a gravação do dados. Caso não fosse fornecido o número do arquivo a ser fechado, a declaração Close fecharia todos os arquivo abertos.

O método **Hide**, esconde um formulário mas não o descarrega da memória. Para que o formulário saia da memória e desapareça, usamos a declaração Unload.
Ex: Unload frmSalvar

A declaração **On Error** desvia a rotina do programa para um tratamento do erro. Caso esta declaração não exista, e ocorra um erro no momento de salvar o arquivo, o VB gera uma mensagem de erro e pára a execução do programa, e isto é muito desagradável para o programador frente ao usuário. No nosso projeto, caso ocorra um erro, será mostrada uma mensagem e encerrado o procedimento.



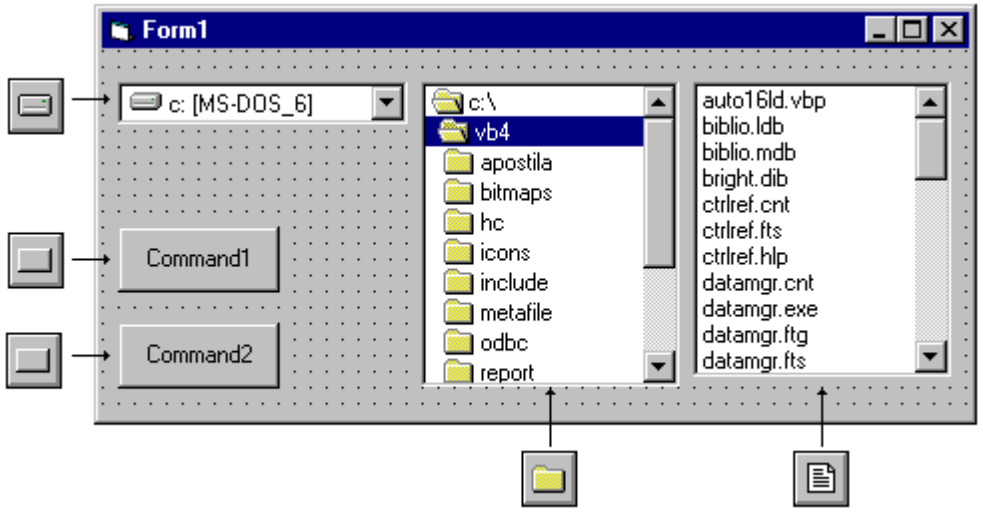

Para que o formulário frmSalvar apareça, devemos digitar o procedimento abaixo associado ao objeto itmSalvar do frmBloco.



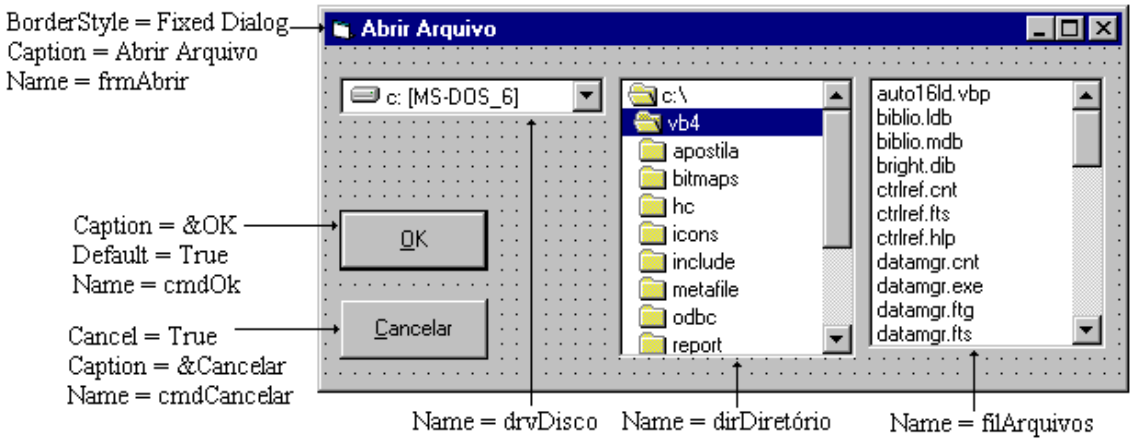

O método **Show** mostra um formulário. O número **1** após Show, indica que o formulário a ser mostrado será do tipo Modal, ou seja, não podemos alternar entre janelas antes de fechá-lo. Caso queiramos que seja alternado, basta informar o número 0 após o método Show.

O método Show antes de mostrar o formulário, carrega-o na memória. Para que o formulário seja carregado na memória e não apareça usamos a declaração Load.
Ex: Load frmSalvar.

Agora, vamos construir o formulário para abrir um arquivo - frmAbrir. Primeiro inserimos um novo formulário, em seguida, colocamos os objetos conforme figura a seguir. Agora, nosso projeto consta de três formulários.



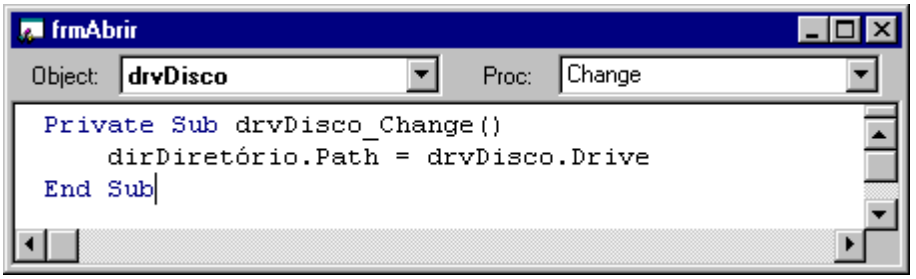

Altere as propriedades dos objetos:




Selecione o formulário frmBloco e dê um click na opção Abrir do menu Arquivo, veremos a janela de código para este item - itmAbrir_Click. Siga o exemplo a seguir, desta forma, quando este item for escolhido, aparecerá a janela para abrir o arquivo.

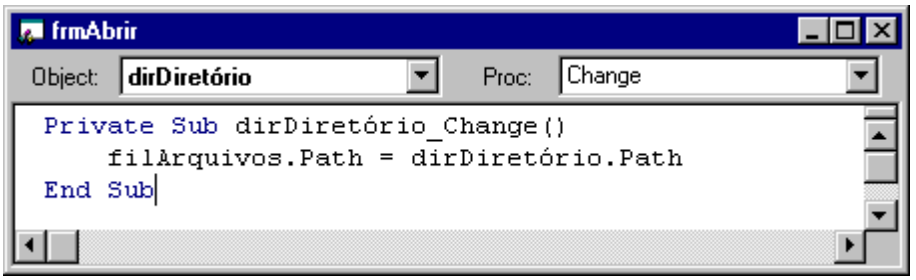

As caixas de Lista de Unidade, Diretório e Arquivos ainda não estão integradas, ou seja, caso alteremos o diretório, o conteúdo da caixa de arquivos não se altera. Experimente executar o projeto e escolha a opção Abrir no meu Arquivo para verificar.

Para que as três caixas fiquem em sincronismo, nós devemos alterar a propriedade Path das caixas de diretório e de arquivo, quando houver mudança de escolha em alguma delas. Esta propriedade determina o caminho absoluto atual incluindo o nome do drive. Ex: “C:\VB4\BITMAPS”, indica que o diretório atual é o subdiretório BITMAPS do diretório VB4 na unidade C: .




No código acima, toda vez que houver alteração (evento Change) na unidade atual, será chamado o procedimento que altera a propriedade Path de dirDiretório que será o diretório raiz da unidade selecionada na caixa de unidades, pois na propriedade Drive teremos a unidade atualmente selecionada.

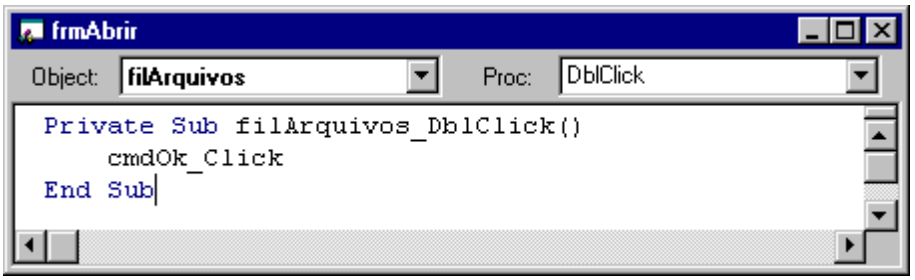
Feito isto, as caixas de lista de unidades e diretório estão sincronizadas, falta a caixa de lista de arquivo. Para sincronizá-la, é só transferir a propriedade Path da dirDiretório para a propriedade da filArquivos, como a seguir:



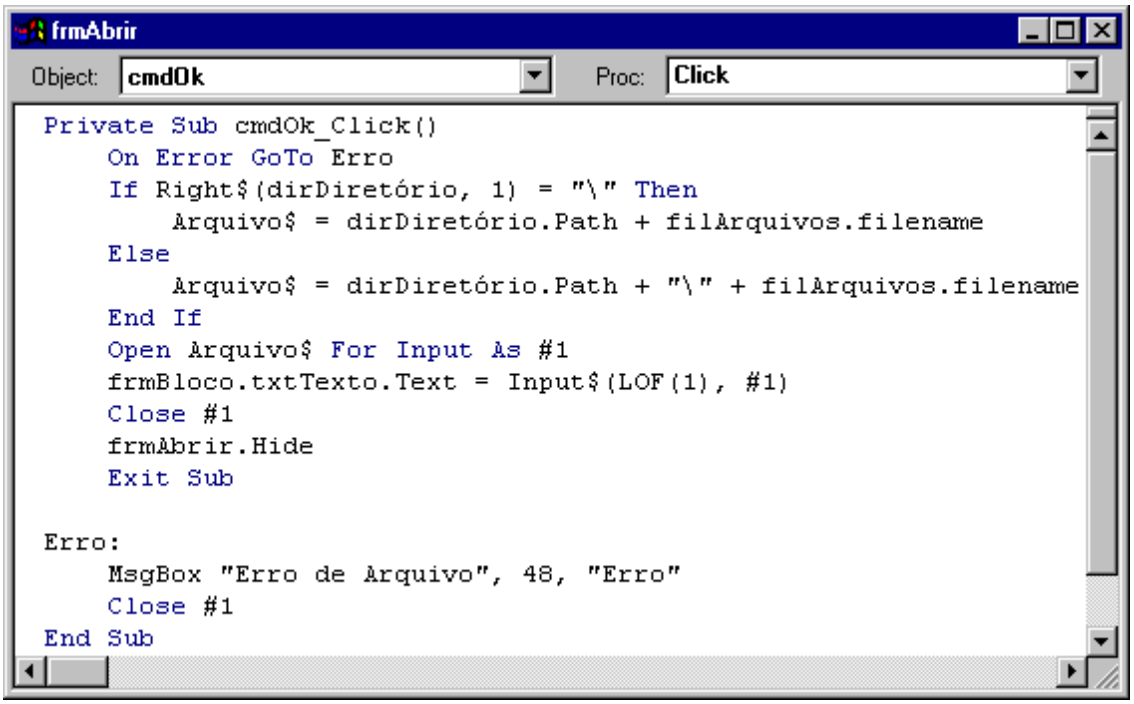

As três caixas estão agora sincronizadas, execute o projeto e alterne entre unidades e diretórios.

Nas aplicações para Windows, quando queremos abrir um arquivo de uma janela do tipo Abrir, podemos fazê-lo de duas formas: 1 - Dando um clique no arquivo e outro no botão Ok, ou 2 - Dando um duplo clique no arquivo selecionado. No VB não precisamos de dois procedimentos para estas duas funções. Um evento pode dar início a outro evento.

No projeto de Bloco de Notas, quando for dado um duplo clique no nome do arquivo, este evento dará partida ao seu procedimento (a seguir), que chamará o evento cmdOk_Click.




Entre com o código do botão cmdOk, conforme figura a seguir:




No código, a função **Right$ Right$(dirDiretório.Path,1)**, verifica qual é o ultimo caractere do caminho do diretório selecionado. Ela retorna n caracteres da direita para a esquerda de um string, e sua sintaxe é a seguinte:

**Right$(**expressão, n**)**

Onde; expressão é uma cadeia de caracteres numéricos ou não e,
n são quantos caracteres se quer retornar.

Ex: txtTexto.Text = Right$(impressão, 3),
Será exibido no quadro de texto as letras **são**.

O código verifica se o caminho atual possui "\" no seu final, caso não exista, ele adiciona à variável Arquivo$ que contém o caminho e nome do arquivo a ser aberto com a declaração Open.

O código **frmBloco.txtTexto.Text = Input$(LOF(1), #1)**, lê o conteúdo do arquivo que foi aberto com o número 1, e transfere este conteúdo a uma variável ou objeto, no nosso caso para o objeto - txtTexto.

**Input$ (**n, #número**)**

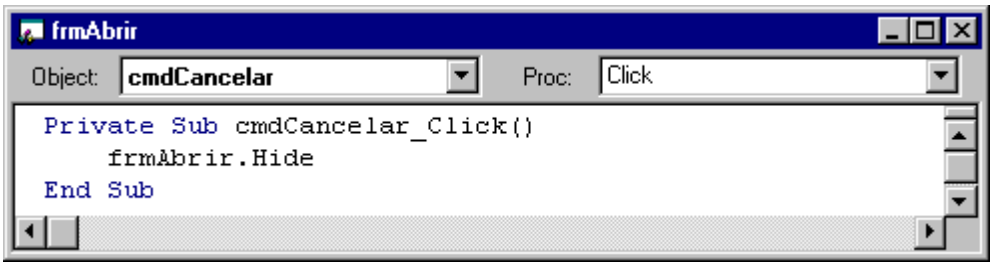
Onde; n indica o número de caracteres a serem lidos do arquivo e número é o número do arquivo a ser lido.

A função LOF, **Input$(LOF(1),#1 )**, retorna o tamanho em bytes de um arquivo aberto.

**LOF** (número)

Onde; número é o número do arquivo a ser lido.

Falta o botão cancelar, que esconderá o formulário - frmAbrir.




Salve e execute o projeto testando todas as opções apresentadas.

## Métodos Gráficos

Embora o uso dos métodos gráficos - que permitem desenhos de linhas, círculos e animações sejam complexos, poderá ser divertido e útil para quem deseja sofisticar seus programas. A seguir, conheceremos tais recursos através de exemplos simples.

O sistema de coordenadas do Visual Basic possui o seu ponto de início (0,0) no canto superior esquerdo, ao contrário do que nós estamos acostumados.

Existem no VB, diversos tipos de escala, são elas:

1 - (Default) Twip = 1440 twips por polegada; 567 twips por centímetro.
2 - Point =72 points por polegada.
3 - Pixel = Unidade do monitor ou resolução da impressora.
4 - Character = horizontal - 120 twips por unid.; vertical - 240 twips por unid.
5 - Polegada
6 - Milímetro
7- Centímetro

Estas escalas são definidas para cada objeto gráfico (formulário, picture box e impressora), na propriedade ScaleMode.

### Desenho de Ponto

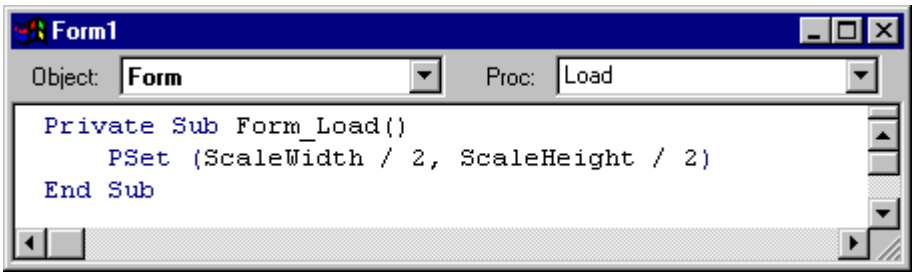
Para desenharmos um ponto utilizamos o método Pset;

objeto.**Pset** [Step] (x,y) [,cor]

Onde:

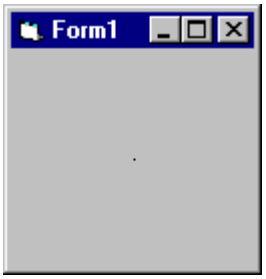
Objeto - objeto onde o ponto será desenhado, se for omitido o ponto será desenhado no formulário corrente.
Step - especifica que as coordenadas serão relativas à posição corrente, propriedades Currentx e CurrentY.
x,y - coordenadas horizontal e vertical
Cor - especifica uma cor para o ponto, se for omitida, será usada a cor da propriedade ForeColor do objeto.

Inicie um novo projeto, e digite o código a seguir para o evento Load do Formulário.

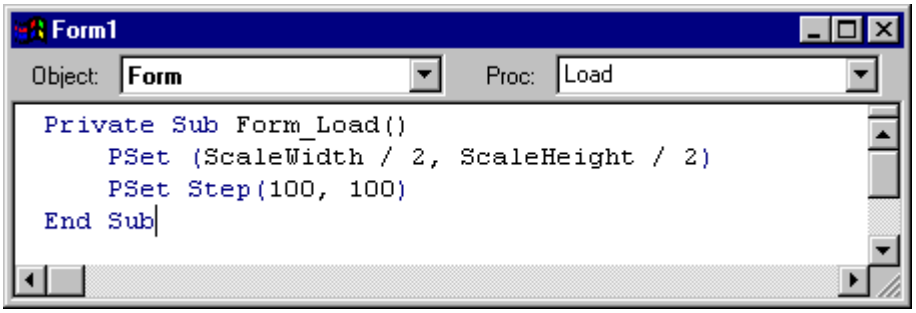


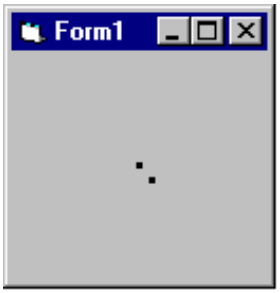

Aperte a tecla F5 para rodar o projeto e observe se algum ponto aparece no centro do formulário.

O ponto não aparece, porque teremos que alterar a propriedade do formulário AutoRedraw = True. Esta propriedade redesenha um gráfico ou formulário automaticamente quando houver alguma alteração nele ou, encoberto por outra janela. Quando usamos o evento Load para desenhar gráficos, temos que deixar AutoRedraw = True. Altere a propriedade e execute o projeto novamente.

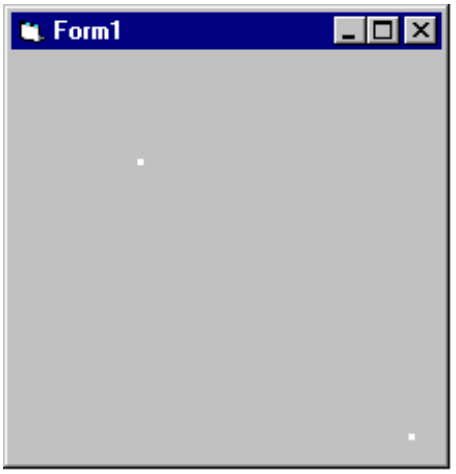



Para desenharmos outro ponto distante do primeiro em 100 Twips, usamos a palavra Step (figura abaixo), e para mudarmos a aparência e a dimensão do ponto, basta alterarmos os valores das propriedades DrawWidth e DrawMode.



Mude a propriedade do formulário ScaleMode = Point, e execute novamente o projeto. Caso o ponto não apareça no formulário, basta redimensionar o formulário durante a execução, e observe o aumento da distância entre os pontos com a mudança na escala.




Teste os outros tipos de escala, alterando a propriedade ScaleMode do formulário.

## CORES

Para determinarmos a cor de um objeto gráfico, temos 4 formas diferentes, são elas:

a) RGB (NRed, NGreen, NBlue), onde NRed, NGreen e NBlue podem variar de 0 a 255.

Ponto vermelho: Pset (100,100), RGB(255,0,0)

b) &H00bbggrr&, onde bb, gg e rr variam em hexadecimal de 00 a FF.

Ponto Amarelo: Pset (100,100), &H0000FFFF&

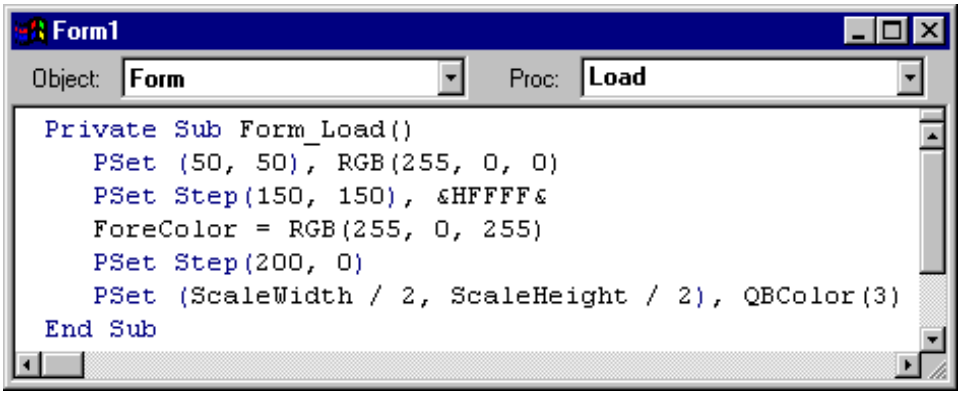
c) Definir a propriedade ForeColor do objeto antes de desenhar o gráfico, não alterando o que já estava desenhado.

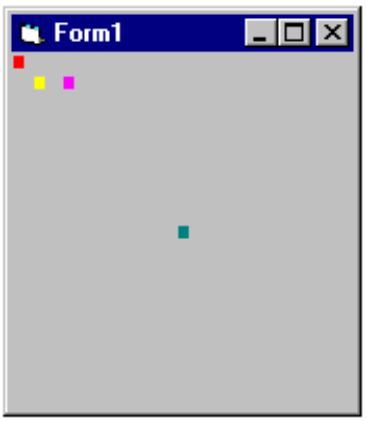
Ponto Magenta: ForeColor = RGB(255, 0, 255)
Pset (100,100)

d) Usando a função QBColor, que possui valores inteiros para as 16 cores mais utilizadas.

Ponto Ciano: PSet (ScaleWidth / 2, ScaleHeight / 2), QBColor(3)

Vamos fazer um exemplo utilizando os quatro métodos para cor, mas antes certifique que a propriedade DrawMode esteja com o valor 3-Copy Pen.




**LINHA**

Para desenharmos linhas, usaremos o método Line:

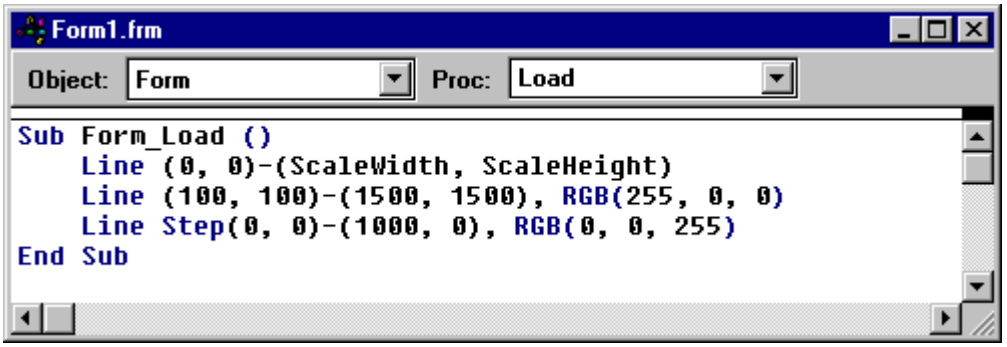
[objeto.] **Line** [ [Step$_a$] (x1,y1) ] - [Step$_b$] (x2,y2) [, [cor], [B], [F] ]

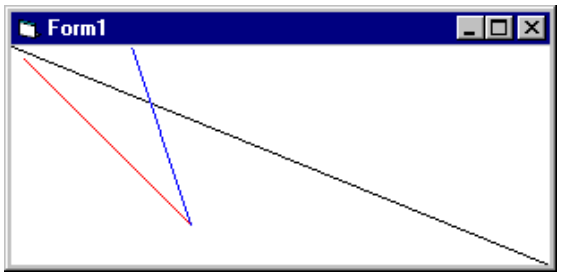
Onde:

Objeto - objeto onde será desenhada a linha (formulário, picture box ou impressora).
Stepa - especifica que as coordenadas são relativas à posição corrente, propriedades CurrentX e CurrentY.
(x1,y1) - coordenadas do ponto inicial.
Stepb - especifica que as coordenadas do ponto final são relativas às coordenadas do ponto inicial.
(x2,y2) - coordenadas do ponto final.
Cor - especifica a cor que a linha será desenhada.
B - opção que desenha um retângulo usando as coordenadas de cantos opostos.

F - especifica que o retângulo será preenchido com a mesma cor usada para desenhar a borda do retângulo. Se não for usada, o retângulo será preenchido com o valor da propriedade FillColor do objeto.

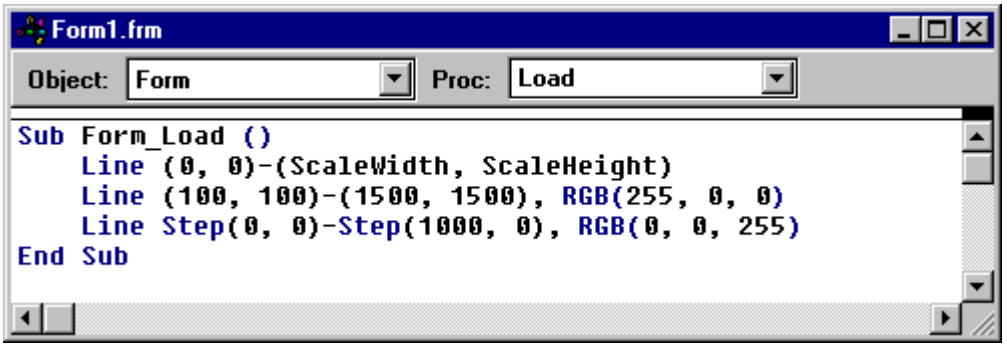
Digite o código abaixo, não esquecendo de alterar as propriedades AutoRedraw = True e BackColor = branco.

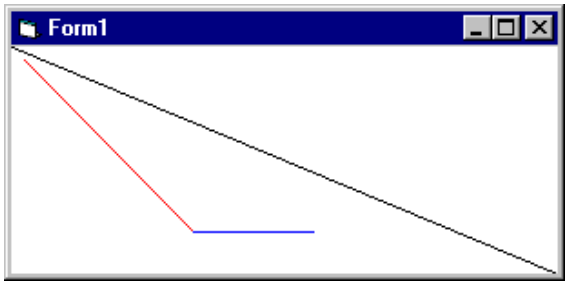


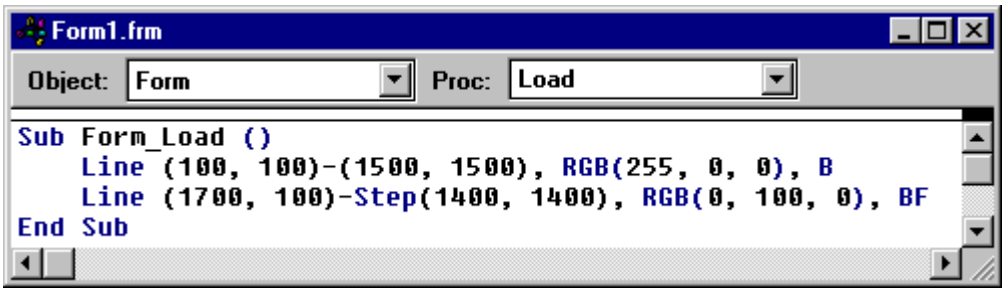

Note que a coordenada do ponto final da terceira linha (1000,0) é interpretada como um valor absoluto, enquanto a coordenada de ponto inicial (0,0) usa como referência o CurrentX e CurrentY - (1500,1500).

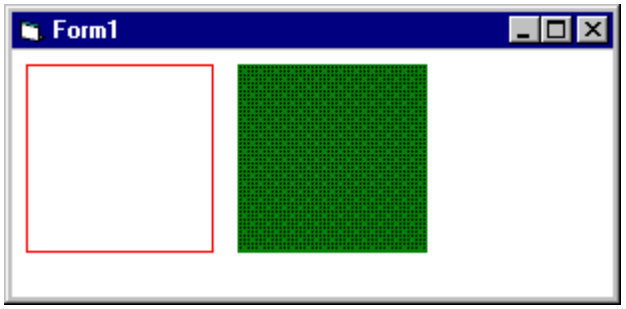
Altere o código e repare a mudança para coordenadas relativas.



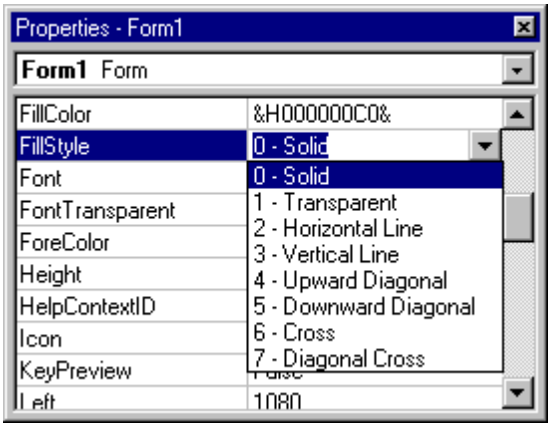



Para desenhar um retângulo, usaremos o método Line com a opção B, e para que este retângulo seje preenchido, usaremos também a opção F.

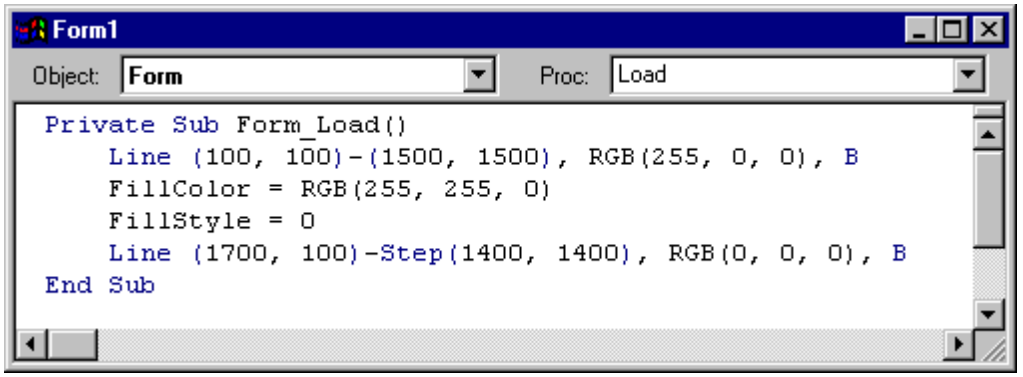


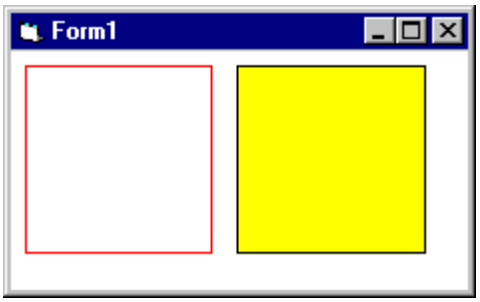




Vamos alterar agora as propriedades de preenchimento do formulário, FillColor=RGB(255,255,0) e FillStyle=0-Solid. Estas propriedades determinam a cor e o padrão de preenchimento e podem ser modificadas tanto em tempo de projeto, quanto em tempo de execução.

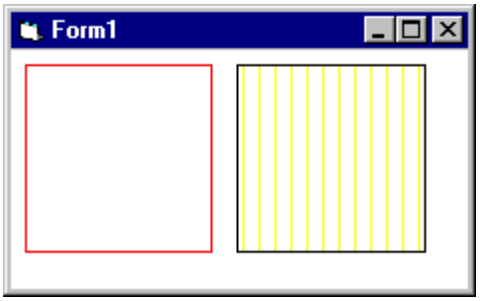



Entre com o código a seguir:

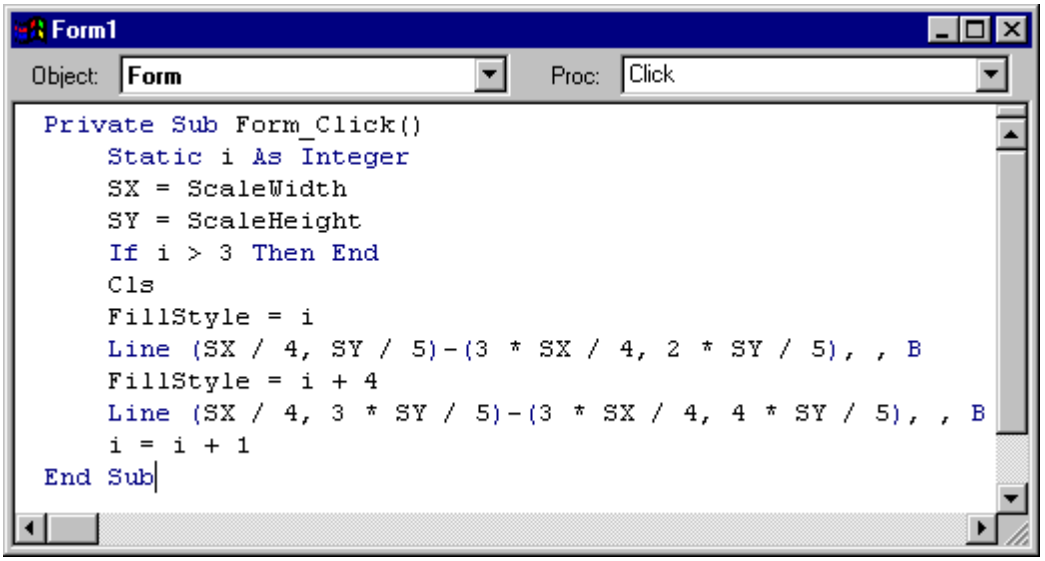





Altere a propriedade na janela de código, FillStyle= 3 - Vertical Line.

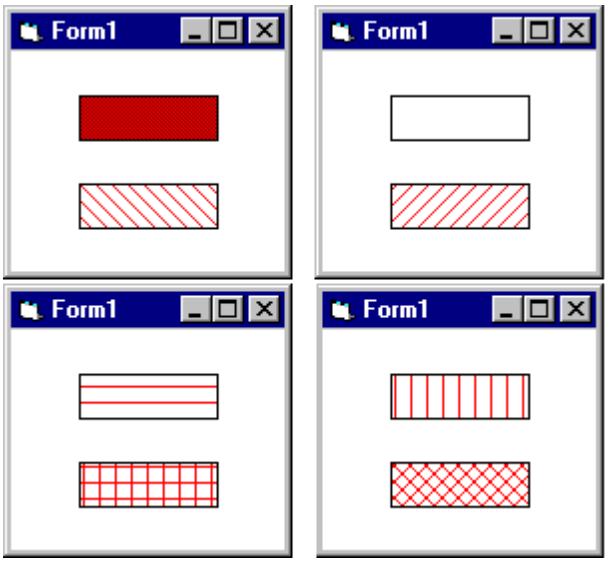



Faça o exemplo abaixo para visualizar todos os tipos de preenchimento;




Neste procedimento, quando damos um Click no formulário, aparecerão dois tipos de preenchimento, até o último, quando o programa será finalizado.

A declaração **Static i As Integer**, declara a variável i como um inteiro e o seu valor não é reinicializado junto com o procedimento, o VB armazena o último valor de i para ser utilizado na próximo vez que o evento Click ocorrer.




A figura acima ilustra todos os tipos de preenchimento (FillStyle).

Existem ainda as propriedades DrawStyle, DrawWidth e DrawMode, que veremos a seguir.

DrawStyle

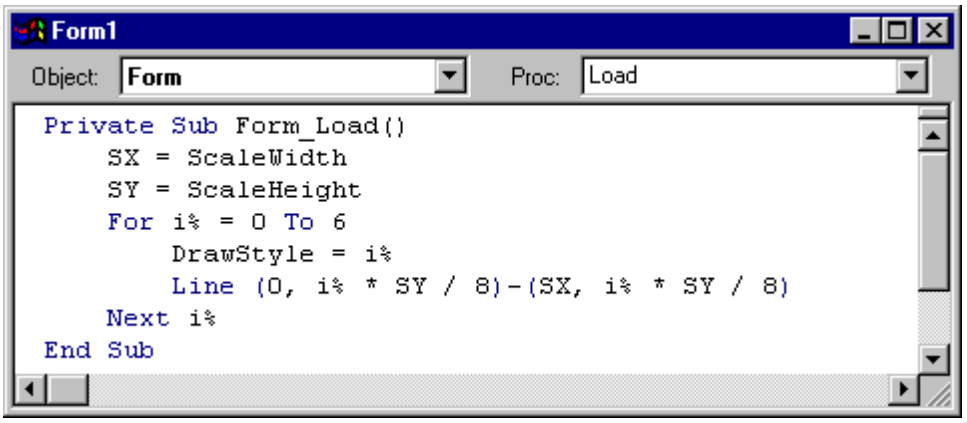
Define o estilo da linha a ser desenhada no objeto gráfico. Pode ser definida tanto em tempo de projeto, quanto em tempo de execução.

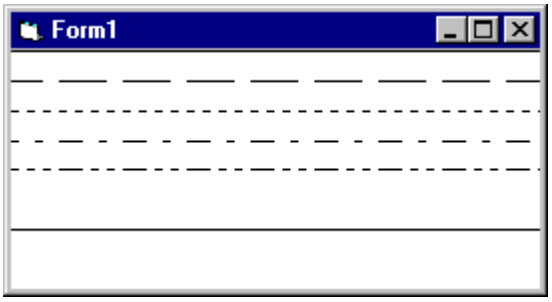
[objeto.] **DrawStyle** = estilo

Estilos: 0 - sólida
1 - tracejada
2 - pontilhada
3 - traço-ponto
4 - traço-ponto-ponto
5 - transparente
6 - interna

Normalmente o VB usa como referência o centro da linha, usando a propriedade DrawStyle=6, ele passa a usar a parte externa da linha como referência.

Entre com o código a seguir para visualizar as opções:



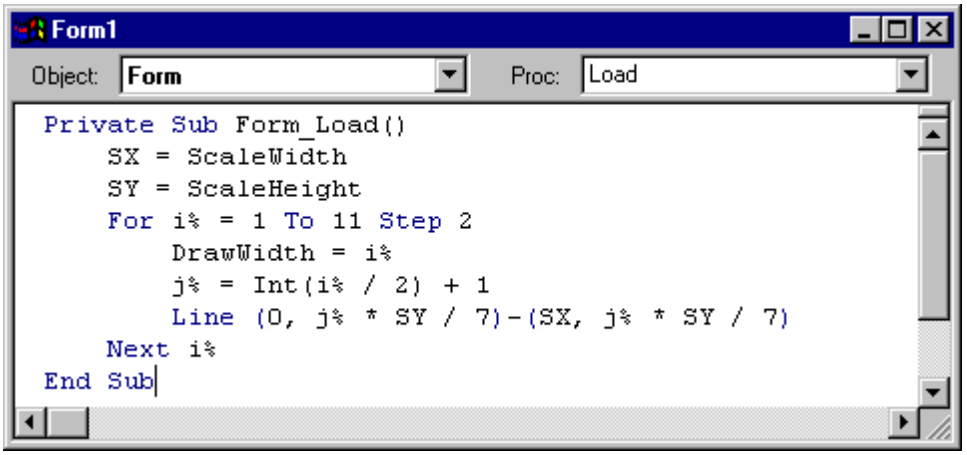



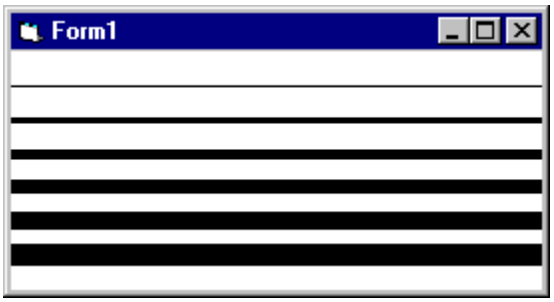

DrawWidth

Define a espessura da linha em pixel, a ser desenha no objeto gráfico e pode variar de 1 até 32.767. Se a propriedade DrawStyle estiver entre 1 e 4 e a DrawWidth for maior que 1, o estilo passará para sólido (DrawStyle=0). Ou seja, não conseguiremos desenhar linhas pontilhadas e tracejadas (modificando a propriedade DrawStyle) com espessura maior que 1 pixel.

[objeto.] **DrawWidth** = tamanho

Execute o exemplo abaixo para visualizar as opções:



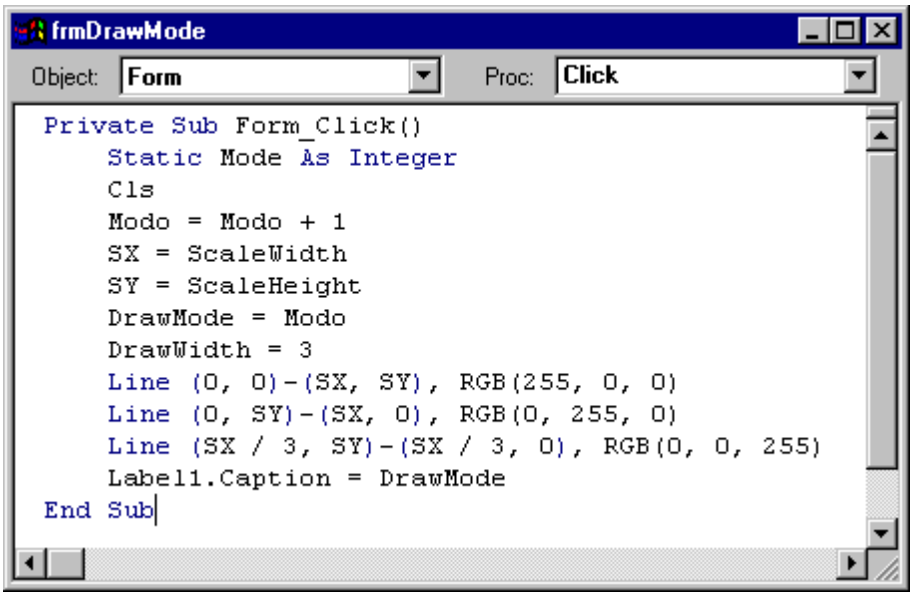


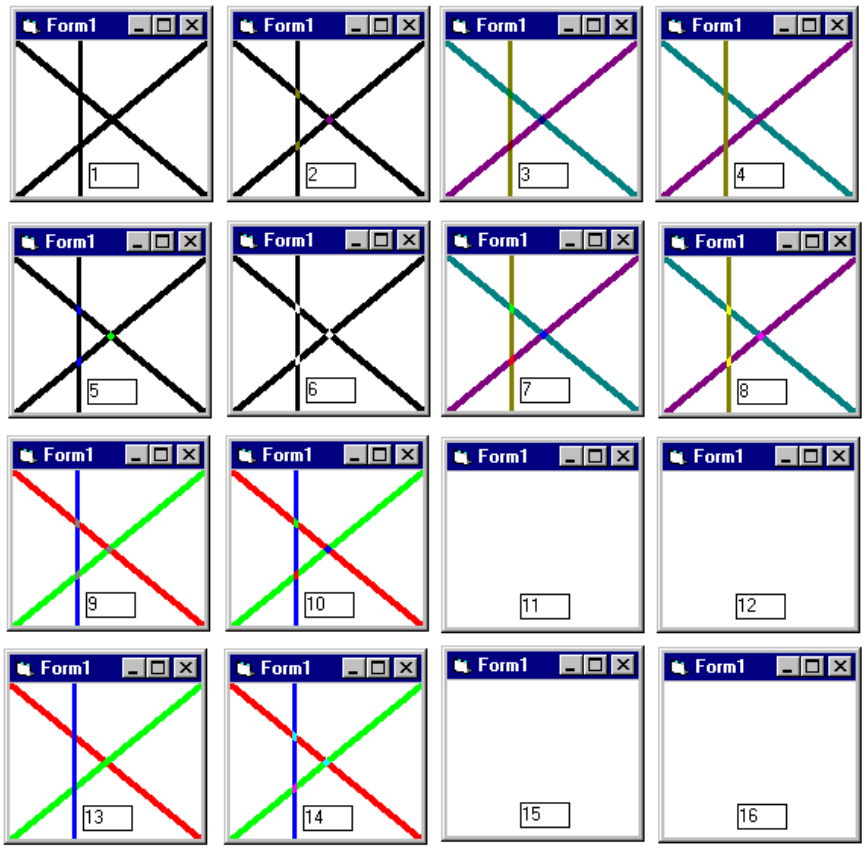

DrawMode

Especifica o modo como será desenhada uma figura no objeto gráfico.

[objeto.] **DrawMode** = modo

Existem 16 modos que podem ser utilizados criando efeitos variados. Execute o exemplo abaixo, para visualizar as alterações nesta propriedade:

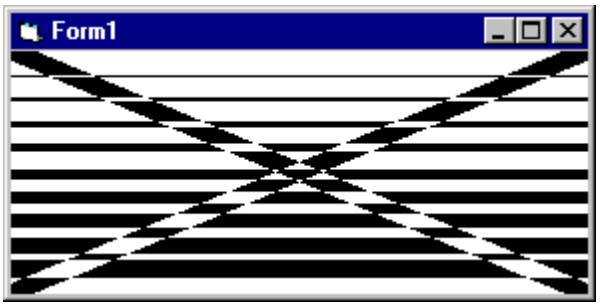





Um do modos mais utilizados é o InvertPen, DrawMode = 6. Vejamos um exemplo de seu uso.

Inicie um novo projeto e entre com o código abaixo. Logo após, execute-o para ver o resultado.

```
Private Sub Form_Load()
   For i = 1 To 9
      DrawWidth = i
      Line (0, i * ScaleHeight / 10)-(ScaleWidth, i * ScaleHeigth / 10)
   Next i
   DrawWidth = i
End Sub

Private Sub Form_Click()
   DrawMode = 6
   DrawWidth = 9
   Line (0, 0)-(ScaleWidth, ScaleHeight)
   Line (0, ScaleHeight)-(ScaleWidth, 0)
End Sub
```




## CÍRCULOS

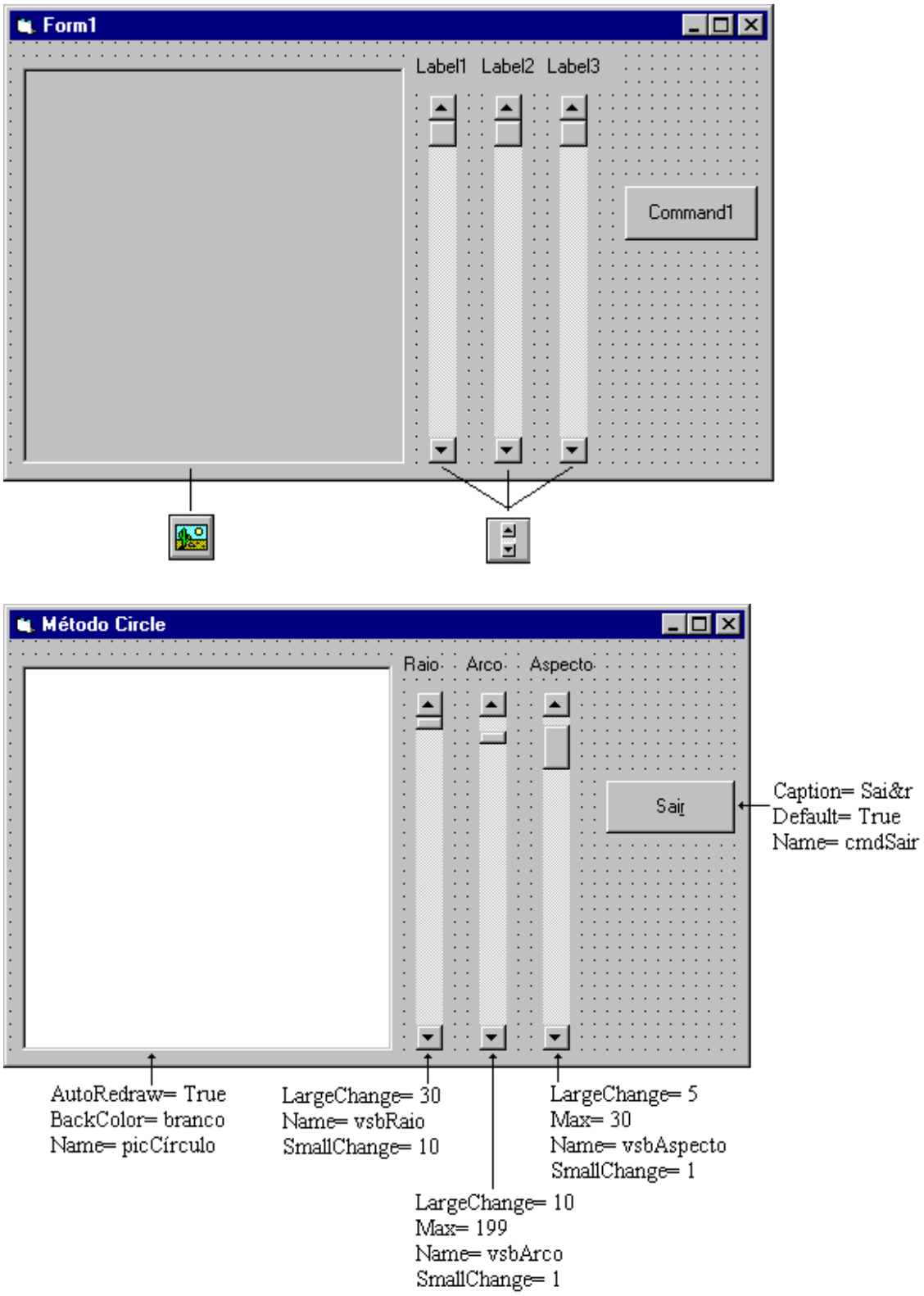
Para desenharmos círculos, usaremos o método Circle. Que desenha círculos, elipses ou arcos em um objeto gráfico.

[objeto,] **Circle** [Step] (x,y), radius , [cor], [start], [end], [aspect]

Onde:

Step - palavra que indica que a coordenada de centro será relativa à posição corrente.
(x,y) - coordenadas de centro do círculo, elipse ou arco.
Radius - raio do círculo, elipse ou arco.
Cor - especifica a cor para o círculo
Start/End - valores que especificam o início e fim do arco a ser desenhado. Valores expressos em radianos, o default é 0 radianos para Start e 2π radianos para End.
Aspecto - indica a proporção entre os raios, que para desenhar uma elipse é diferente de 1. No círculo perfeito o aspecto é igual a 1.

Faça o projeto Círculo conforme exemplo a seguir, para visualizar o método Circle.




As três Vertical Scroll Bar servirão para modificarmos os parâmetros do método Circle. Ou seja, poderemos desenhar um círculo, elipse ou arco dentro do Picture Box. Existem cinco propriedades primarias das barras de paginação em que estamos interessados: Min, Max, Value, LargeChange e SmallChange.

Onde:

Min - valor numérico atribuído ao lado superior ou esquerdo da barra de paginação.
Max - valor numérico atribuído ao lado inferior ou direito da barra de paginação.
Value - valor correspondente à posição do marcador na barra. Que está entre Min e Max.
LargeChange - indica a quantidade que a propriedade Value deverá variar toda vez que o usuário acionar a barra acima ou abaixo do marcador.
SmallChange - indica a quantidade que a propriedade Value deverá variar quando o usuário acionar as setas da barra de paginação.

O evento mais utilizado com as barras de paginação, é o evento Change, que ocorre toda vez que o usuário move o marcador.

Digite o código do procedimento geral de formulário RedesenhaCírculo.

```
Public Sub RedesenhaCírculo()
   SX = picCírculo.ScaleWidth
   SY = picCírculo.ScaleHeight
   picCírculo.Cls
   Raio = vsbRaio.Value
   Fim = vsbArco.Value / 100 * 3.1415
   Aspecto = vsbAspecto.Value / 10
   picCírculo.Circle (SX / 2, SY / 2), Raio, RGB(255, 0, 0), 0, Fim, Aspecto
End Sub
```

O procedimento acima irá redesenhar o círculo toda vez que o usuário mover o marcador de qualquer uma das barras de paginação. Para redesenhar o círculo, ele utilizará a propriedade Value das barras como parâmetros.

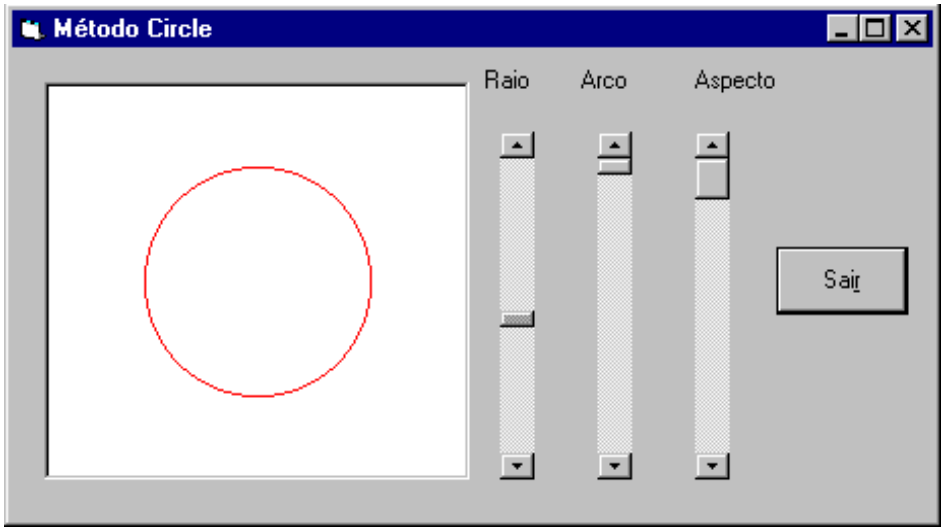
Digite os códigos aseguir, para os objetos do formulário.

```
Private Sub cmdSair_Click()
   End
End Sub

Private Sub Form_Load()
   SX = picCírculo.ScaleWidth
   SY = picCírculo.ScaleHeight
   picCírculo.Circle (SX / 2, SY / 2), SX / 10, RGB(255, 0, 0)
   vsbRaio.Max = SX / 2
   vsbRaio.Value = SX / 10
   vsbArco.Value = 0
   vsbAspecto.Value = 10
End Sub
```

O procedimento Form_Load desenhará um círculo no centro Picture Box e depois atribuirá valores às propriedades Value das barras de paginação.

```
Private Sub vsbArco_Change()
   RedesenhaCírculo
End Sub

Private Sub vsbAspecto_Change()
   RedesenhaCírculo
End Sub

Private Sub vsbRaio_Change()
   RedesenhaCírculo
End Sub
```

Toda vez que ocorre um evento Change em alguma barra de paginação, será chamado o procedimento geral do formulário RedesenhaCírculo.




## CARREGANDO FIGURAS

Podemos carregar três tipos de arquivos de figura:

- .ICO (ícones) RedesenharCírculo
- .BMP (mapa de bits)
- .WMF (windows metafiles)
- .RLE (run-length encoded)

Inserimos figuras em, formulários, quadros de figura e controle imagem. Existem diferenças entre o controle imagem e o quadro de figura:

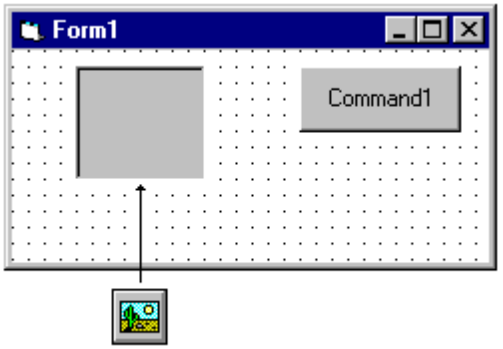
Controle Imagem - Não faz linhas nem círculos.
Muda automaticamente de tamanho de acordo com a figura.
Ajusta a figura de acordo com o seu tamanho se a propriedade Strech (esticar) = True.
Permite redimensionar a figura depois de carregada.

Para caregarmos uma figura em um objeto, usamos a função LoadPicture para atribuir o nome do desenho na propriedade Picture do objeto.

objeto.Picture = **LoadPicture** (“caminho e nome da figura")

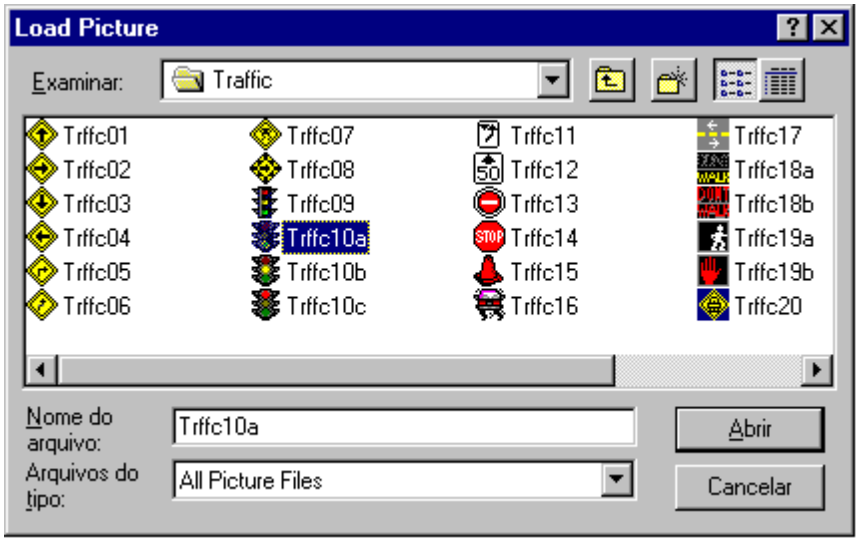
A propriedade Picture pode ser modificada tanto em tempo de projeto quanto em tempo de execução.

Como exemplo, vamos construir o projeto semáforo.

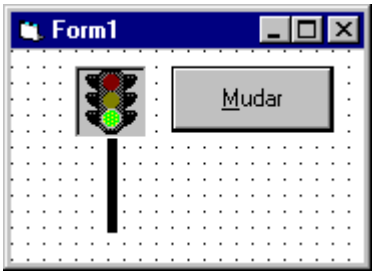



Altere a propriedade Autosize = True do Quadro de Figura, para que ele ajuste o seu tamanho ao da figura que será carregada.

Escolha a propriedade Picture do Quadro de Figura e selecione “\Icons\Traffic\Trffc10a .ico”, como mostra a figura abaixo.



Altere as propriedades restantes do objeto, coloque um objeto Line -  - e ajuste as suas coordenadas (propriedades X1, X2, Y1 eY2) para um acerto com o Quadro de Figura.




Quando o usuário selecionar o botão Mudar, o semáforo mudará de estado. Vamos agora construir o código para este evento.

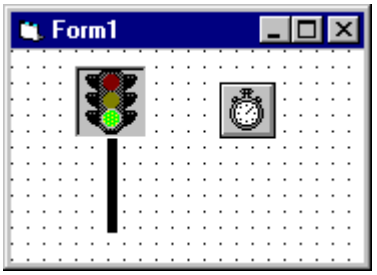
```
Private Sub cmdMudar_Click()
   Static Sinal As Integer
   If Sinal = 0 Then
      Picture1.Picture = LoadPicture("c:\vb4\icons\traffic\trffc10b.ico")
      Sinal = 1
   ElseIf Sinal = 1 Then
      Picture1.Picture = LoadPicture("c:\vb4\icons\traffic\trffc10c.ico")
      Sinal = 2
   ElseIf Sinal = 2 Then
      Picture1.Picture = LoadPicture("c:\vb4\icons\traffic\trffc10a.ico")
      Sinal = 0
   End If
End Sub
```

No exemplo acima, toda vez que o usuário selecionar o botão Mudar, uma nova figura será carregada no Picture1. Mas podemos também colocar no formulário três quadros de Figura, cada um com uma figura diferente e ir modificando a propriedade Visible desses quadros.

No nosso projeto de semáforo, insira mais dois quadros de figura e posicione-os no mesmo local do primeiro, um em cima do outro definindo as propriedades Picture e Visible, como segue:

```
Picture1.Picture = "c:\vb4\icons\traffic\trffc10a.ico"
Picture1.Visible = True
Picture2.Picture = "c:\vb4\icons\traffic\trffc10b.ico"
Picture2.Visible = False
Picture3.Picture = "c:\vb4\icons\traffic\trffc10c.ico"
Picture3.Visible = False
```

E também vamos eliminar o botão de comando, inserindo o objeto Timer -  . Este controle gera o evento Timer a intervalos de tempo determinados pela sua propriedade Interval. Esta propriedade é expressa em milisegundos, se quisermos um intervalo de 2 segundos - Interval = 2000.




```
Private Sub Timer1_Timer()
   Static Sinal As Integer
   If Sinal = 0 Then
      Picture1.Visible = True
      Picture3.Visible = False
      Sinal = 1
   ElseIf Sinal = 1 Then
      Picture2.Visible = True
      Picture1.Visible = False
      Sinal = 2
   ElseIf Sinal = 2 Then
      Picture3.Visible = True
      Picture2.Visible = False
      Sinal = 0
   End If
End Sub
```

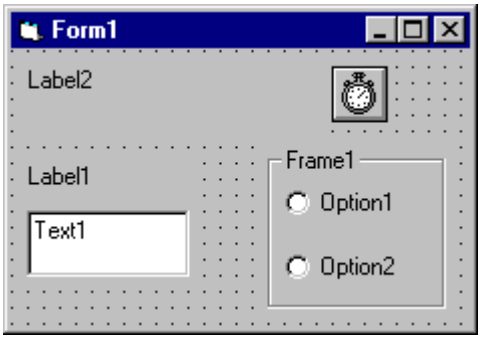
Execute o projeto alterando a propriedade Interval do Timer1, observando as mudanças.

## DESPERTADOR

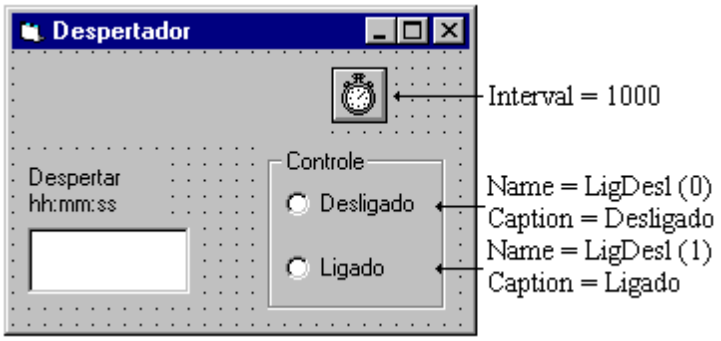
No Windows usamos botões de opção quando temos que escolher entre opções mutuamente excludentes, ou seja, apenas uma das opções pode estar selecionada. Vamos usar este recurso para construir um pequeno relógio despertador.

Em um formulário podemos ter somente um botão de opção selecionado de cada vez. Caso o projeto necessite de mais de um grupo de seleção, utilizamos o objeto Frame para separar os diferentes grupos.

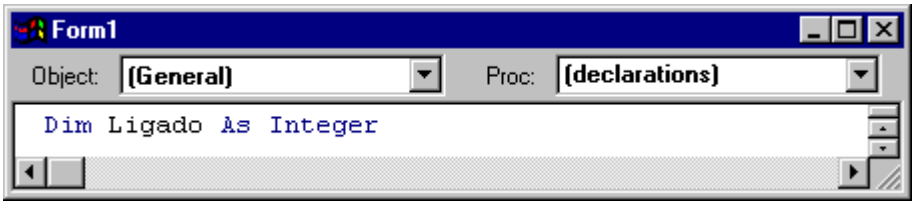
Coloque os controles no formulário, tendo o cuidado de inserir primeiro o Frame e depois os botões de opção dentro dele. Para verificar se eles estão realmente dentro do Frame, selecione-o e tente arrastar um botão de opção para fora, se ele sair, não estará dentro do Frame, apague-o e tente novamente pelo método de arraste do mouse.




Altere as propriedades dos objetos conforme figua abaixo:




Entre com os códigos abaixo.




**Private Sub LigDesl_Click(Index As Integer)**
  If Index = 1 Then
    Ligado = True
  Else
    Ligado = False
  End If
**End Sub**

Este procedimento atribui um valor para a variável geral Ligado, dependendo do botão de opção selecionado.

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
   Tecla$ = Chr$(KeyAscii)
   If ((Tecla$ < "0" Or Tecla$ > "9") And Tecla$ <> ":") Then
      Beep
      KeyAscii = 0
   End If
End Sub
```

O procedimento acima verifica se a tecla selecionada está entre os valores válidos.

Quando pressionamos uma tecla no quadro de texto, é gerado o evento KeyPress. Este evento retorna o código da tecla pressionada, para ser utilizado no código do procedimento. A função Chr$(charcode), retorna o caracter (String) de acordo com o código da tabela ANSI.

```
Private Sub Timer1_Timer()
   If (Time$ >= Text1.Text And Ligado) Then Beep
   Label2.Caption = Time$
End Sub
```

A função Time$ retorna a hora do sistema. O procedimento acima compara a hora do sistema (Time$) com a hora que está no quadro de texto (Text1), e se for maior e a variável Ligado =True, aciona o Beep. Este procedimento também atualiza o conteúdo do Label2 a cada segundo.

## ANIMAÇÃO

Para animar uma figura temos dois modos. O primeiro consiste em mover os pontos da figura, e o segundo mover toda a figura que está “dentro” de um objeto de imagem.

### *PRIMEIRO MODO*

Vamos construir primeiro um projeto que move os pontos da figura. Inicie um novo projeto e digite o código abaixo.

```
Private Sub Form_Click()
   Dim Passo As Integer
   Dim Cor As Integer
   Dim NaveArray(1 To 14) As String
   Dim Nave(15, 14) As Long
   NaveArray(1) = "0000000F0000000"
   NaveArray(2) = "000000FCF000000"
   NaveArray(3) = "000000FCF000000"
   NaveArray(4) = "00000FCCCF00000"
```

```
   NaveArray(5) = "00000FCCCF00000"
   NaveArray(6) = "0000FCCCCCF0000"
   NaveArray(7) = "0000FCCCCCF0000"
   NaveArray(8) = "000FCCCCCCCF000"
   NaveArray(9) = "00FCCCCCCCCCF00"
   NaveArray(10) = "00FCCCC9CCCCF00"
   NaveArray(11) = "0FCCC99A99CCCF0"
   NaveArray(12) = "0FC9900A0099C0F0
   NaveArray(13) = "FC90000A00009CF"
   NaveArray(14) = "990000AAA000099"
   DrawMode = 7
   BackColor = 0
   ScaleMode = 3
   GoSub FazerNave
   X = Int(ScaleWidth / 3)
   Y = ScaleHeight - 14
   GoSub DesenharNave
   Passo = Int(ScaleHeight / 10)
   For i% = (ScaleHeight - 14) To Passo Step -Passo
      Y = i%
      GoSub DesenharNave
      Y = i% - Passo
      GoSub DesenharNave
   Next i%
   Exit Sub

FazerNave:
   For i% = 1 To 14
      For j% = 1 To 15
         Cor = Asc(UCase$(Mid$(NaveArray(i%), j%, 1)))
         If Cor <= Asc("9") Then
            Cor = Cor - Asc("0")
         Else
            Cor = Cor - Asc("A") + 10
         End If
         Nave(j%, i%) = QBColor(Cor)
      Next j%
   Next i%
   Return

DesenharNave:
   For ii% = 1 To 15
      For jj% = 1 To 14
         PSet (X + ii% - 1, Y + jj% - 1), Nave(ii%, jj%)
      Next jj%
   Next ii%
   Return
End Sub
```

Na primeira parte do procedimento acima, declaramos as variáveis, duas como Inteiro e duas outras como matrizes.

A variável NaveArray( ), é do tipo String e terá 14 posições, onde serão armazenados os códigos das cores de cada linha do desenho. A variável Nave(15,14) é uma matriz 15x14 onde serão armazenados, em cada posição, as cores dos pontos do desenho, usando-se a função QBColor( ).

Depois atribuímos os valores à variável NaveArray( ) de acordo com o desenho que queremos montar, associando cada caractere a uma cor, como mostra a tabela abaixo:

| Caracter | Cor | Caracter | Cor |
|---|---|---|---|
| 0 | Preto | 8 | Cinza |
| 1 | Azul | 9 | Azul Claro |
| 2 | Verde | A | Verde Claro |
| 3 | Ciano | B | Ciano Claro |
| 4 | Vermelho | C | Vermelho Claro |
| 5 | Magenta | D | Magenta Claro |
| 6 | Amarelo | E | Amarelo Claro |
| 7 | Branco | F | Branco Brilhante |

Feito o desenho, definimos as propriedades do Formulário:

DrawMode=7 - XOR pen
BackColor = 0 - Preto
ScaleMode = 3 - Pixel

DrawMode=7- XOR pen. Esta propriedade realiza uma operação bit a bit entre a cor que esta na tela e a cor que esta se desenhando. Este modo de operação é útil em animação porque você pode desenhar qualquer coisa na tela usando o DrawMode=7 para fazê-la aparecer, e quando desenhar novamente a mesma figura ela desaparecerá. Nós usamos este modo no nosso projeto da seguinte forma: desenhamos a nave a primeira vez - ela aparece, desenhamos uma segunda vez - ela desaparece, movemos as coordenadas dando início a um novo ciclo.

A sub-rotina FazerNave, carrega os pontos da matriz Nave( ) com as cores fornecidas pela variável NaveArray( ). Ela lê caractere por caractere de cada linha retornando o código ASCII deste caractere colocando-o na variável Cor. A seguir, a variável Cor é convertida para valores da função QBColor, como mostra a tabela a seguir:

| Caracter | ASC(Caracter) | Cor | QBColor(Cor) |
|---|---|---|---|
| 0 | 48 | 0 | Preto |
| 1 | 49 | 1 | Azul |
| 2 | 50 | 2 | Verde |
| 3 | 51 | 3 | Ciano |
| 4 | 52 | 4 | Vermelho |
| 5 | 53 | 5 | Magenta |
| 6 | 54 | 6 | Amarelo |
| 7 | 55 | 7 | Branco |
| 8 | 56 | 8 | Cinza |
| 9 | 57 | 9 | Azul Claro |
| A | 65 | 10 | Verde Claro |
| B | 66 | 11 | Ciano Claro |
| C | 67 | 12 | Vermelho Claro |
| D | 68 | 13 | Magenta Claro |
| E | 69 | 14 | Amarelo Claro |
| F | 70 | 15 | Branco Brilhante |

A função Mid$, retorna um String que é parte de outro String.

**Mid$** (string, começo [,comprimento] )

Onde:

String - é a cadeia de caracteres original
Começo - especifica a partir de qual caractere começará a extração.
Comprimento - indica quantos caracteres se deseja extrair.

Ex: Palavra$ = Mid$ (“Constituição”,2,4)
equivale a: Palavra$ = “onst”

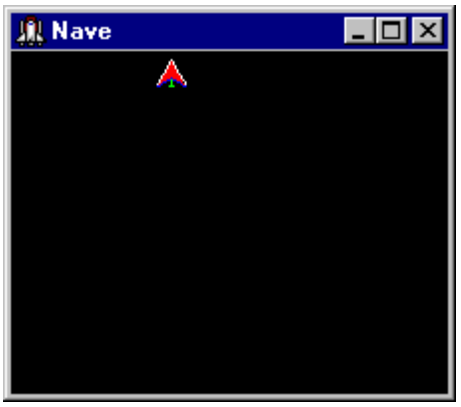
Como as letras maiúsculas e minúsculas possuem código ASCII diferentes, e nosso código só funciona com o código das maiúsculas, usamos a função Ucase$ para evitar erros caso por engano digitamos alguma letra minúscula na variável NaveArray( ). Esta função transforma todos caracteres da expressão em maiúsculos.

**UCase$** (expressão)

Ex: Palavra$ = UCase$ (“PalmTop”)
equivale a: Palavra$ = “PALMTOP”

A sub-rotina DesenharNave, usa o método Pset para desenhar todos os pontos, que estão na matriz Nave( ), no Formulário usando como referência as coordenadas variáveis X e Y.

Execute o projeto e observe a nave sendo desenhada e apagada sucessivas vezes.



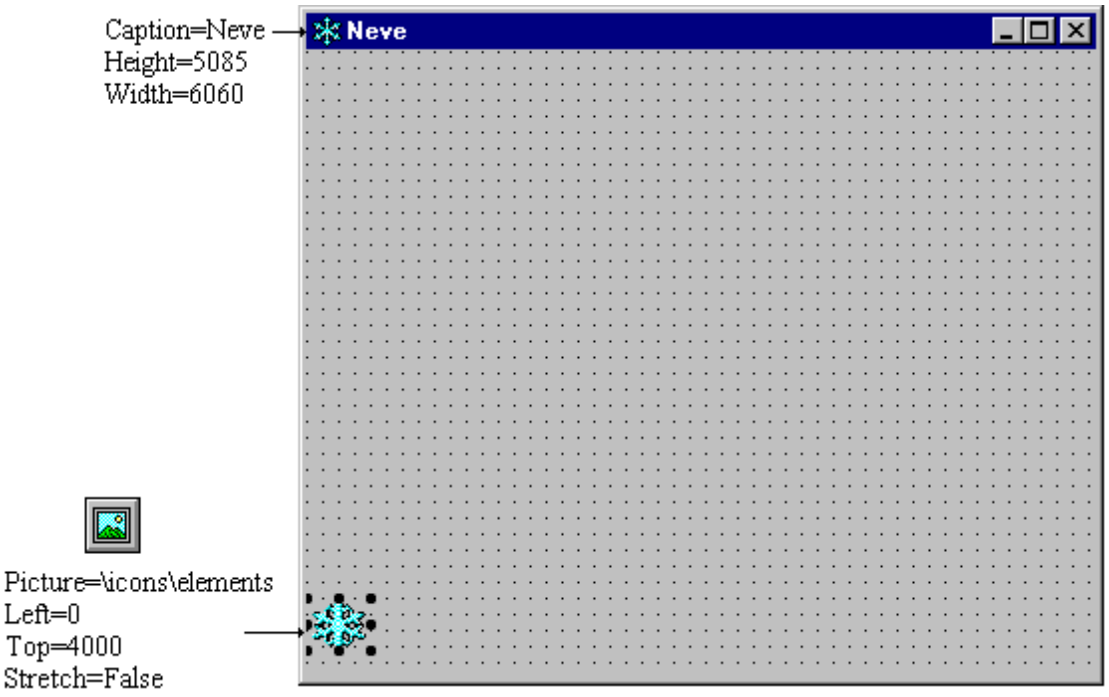

***SEGUNDO MODO***

No Visual Basic não costumamos mover os pontos do desenho para movimentá-lo, porque o VB é lento, ou melhor - não é uma linguagem apropriada para este tipo de programa - e com isso vemos o desenho sendo construído e apagado, durante o seu movimento. Como alternativa, poderíamos mover todo o desenho em um só bloco usando o método Move.

objeto.**Move** left[,top[,width[,height] ] ]

Onde:
left - coordenada horizontal
top - coordenado vertical
width - nova largura
height - nova altura

Inicie um novo projeto e insira no formulário um Controle Image.




Digite o código a seguir, e note que está sendo usada uma equação de segundo grau ( $Y=0,05X^2 - 24X + 4000$) para definir a trajetória do controle Image1.

```
Private Sub Form_Click()
   Dim X As Single
   Dim Y As Single
   Image1.Visible = True
   For X = 1 To 500 Step 2
      Y = (X ^ 2 * 0.05) - (24 * X) + 4000
      Image1.Move (100 + (X * 10)), Y
   Next X
   Image1.Visible = False
End Sub
```

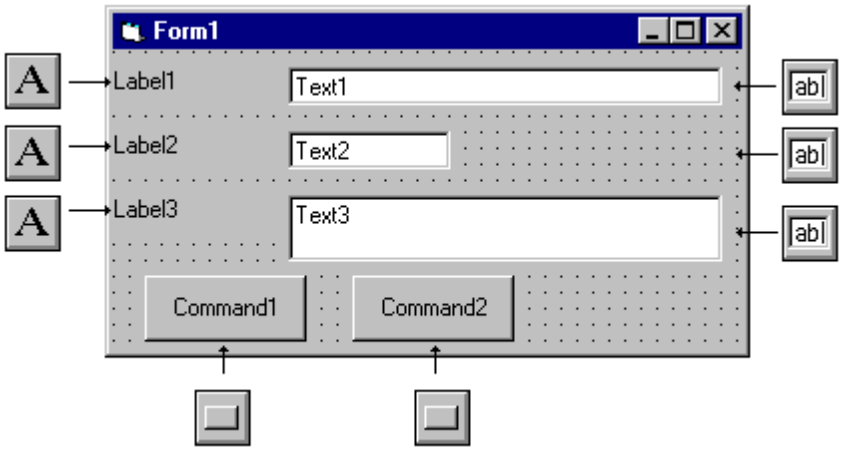
Toda vez que o usuário der um Clique sobre o formulário, o floco de neve se moverá, desaparecendo no final. Altere os parâmetros e observe as alterações de trajeto do controle Image.

A propriedade Stretch do Controle Image, permite que o tamanho da figura acompanhe o tamanho do controle. Deixe a propriedade do controle Image1 como Stretch=True, e depois no formulário, altere o tamanho do controle e observe que a figura acompanha o tamanho do controle, isto não é possível com o controle PictureBox.

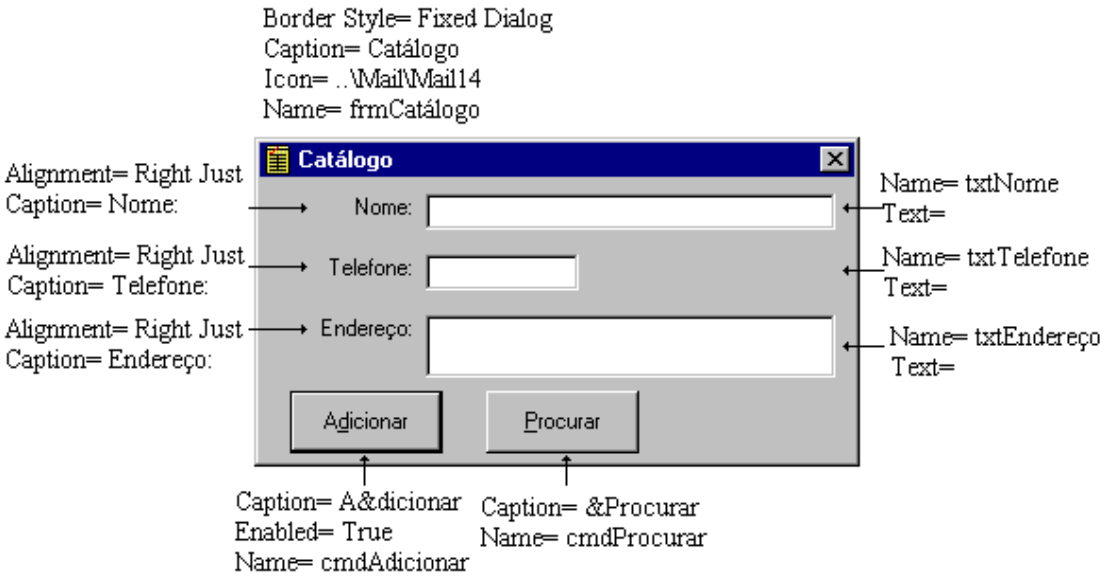
## Exemplo IV - Catálogo

Neste projeto, conheceremos a importância das variáveis composta e array, o uso do comando de impressão, e trabalharemos com acesso a arquivos do tipo aleatório, todas essas informações serão apresentadas através de um catálogo de endereços e telefones.

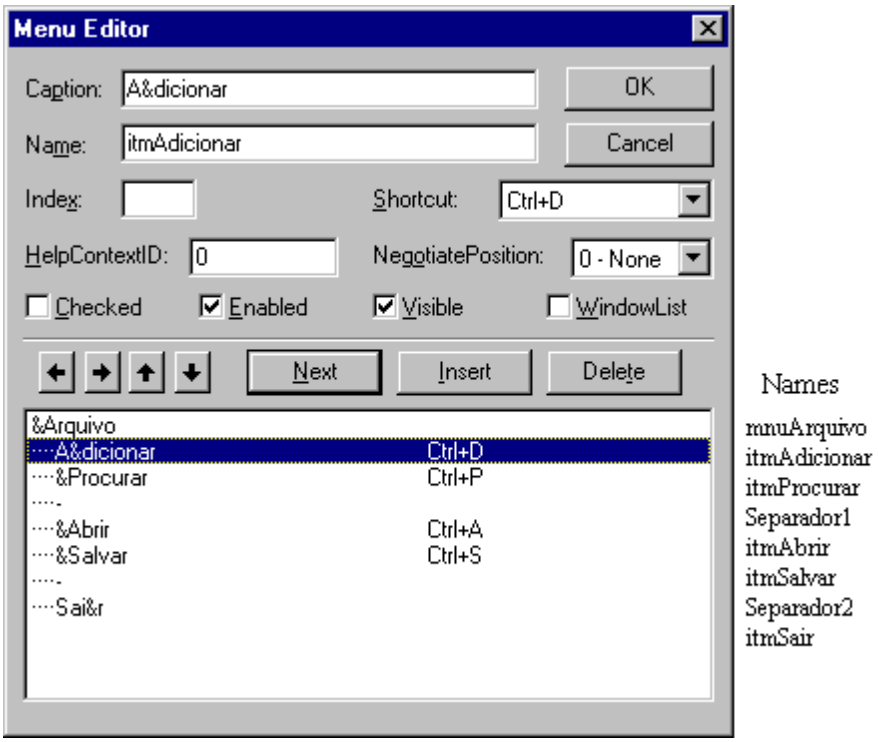
Inicie um novo projeto e crie o formulário como a figura abaixo.




Altere as propriedades dos objetos.



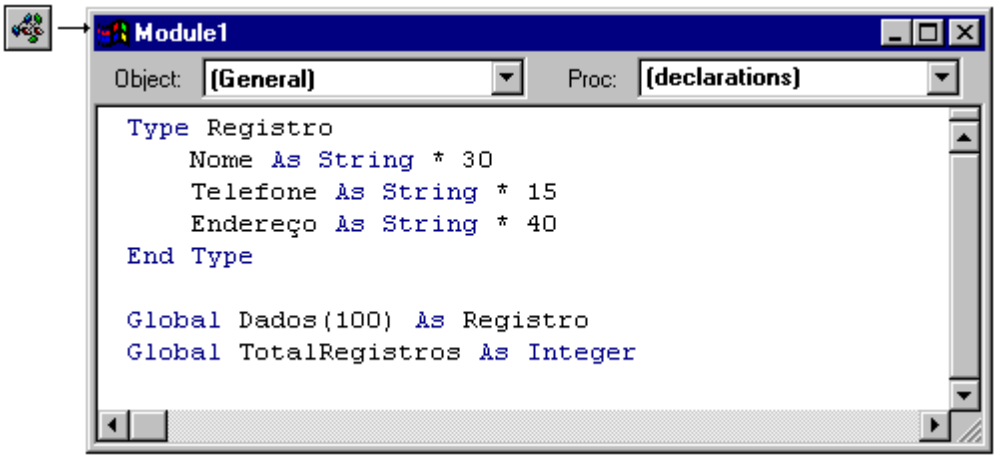

Vamos construir um menu para o nosso catálogo. Selecione o formulário Catálogo e escolha a opção Menu Editor... do menu Tool, e monte-o conforme a figura a seguir.




Um módulo contém as variáveis e procedimentos que poderão ser acessados por toda a aplicação, ou seja, qualquer outro procedimento de qualquer formulário poderá chamar um outro procedimento do módulo. Ou então, pode ser dado um valor a uma variável em um formulário e este valor ser lido por outro formulário, quando esta variável é declarada no módulo.

Para inserir um módulo ao projeto, basta escolher a opção Module do menu Insert ou, selecione o botão  na barra de ferramentas.

Aparecerá uma janela como mostrada a seguir, logo após, digite o código mostrado na figura.



## VARIÁVEIS COMPOSTAS E ARRAY

Neste módulo estamos declarando uma variável composta (Registro) que conterá os campos Nome, Telefone e Endereço, declarados como String de tamanho fixo. Para uma variável declarada como String, não precisa ser definido um tamanho para ela, a menos que estejamos preparando registros.

Declaramos também um Array de variável ( Dados(100) ). Quando declaramos um Array de variável, o VB reserva espaço na memória para conter todas as variáveis, que no nosso caso varia de Dados(0) até Dados(99). Esta variável é declarada como sendo do tipo Registro, declarado anteriormente, desta forma teremos dentro da variável Dados( ) os campos Nome, Telefone e Endereço, que poderão ser acessados usando o operador (.) ponto:

```
Dados(5).Nome = “Carlos”
Dados(5).Telefone = “981-0000”
Dados(5).Endereço = “Rua França, 70”
```

Neste Módulo também estamos declarando a variável TotalRegistros que conterá o número total de registros armazenados e poderá ser acessada por qualquer formulário.

Digite o código para cada objeto e evento do formulário catálogo (frmCatálogo), como os exemplos a seguir.

```
Private Sub cmdAdicionar_Click()
   TotalRegistros = TotalRegistros + 1
   Dados(TotalRegistros).Nome = txtNome.Text
   Dados(TotalRegistros).Telefone = txtTelefone.Text
   Dados(TotalRegistros).Endereço = txtEndereço.Text
   frmProcurar.lstListaNomes.AddItem txtNome.Text

   txtNome.Text = ""
   txtTelefone.Text = ""
   txtEndereço.Text = ""
   txtNome.SetFocus
End Sub
```

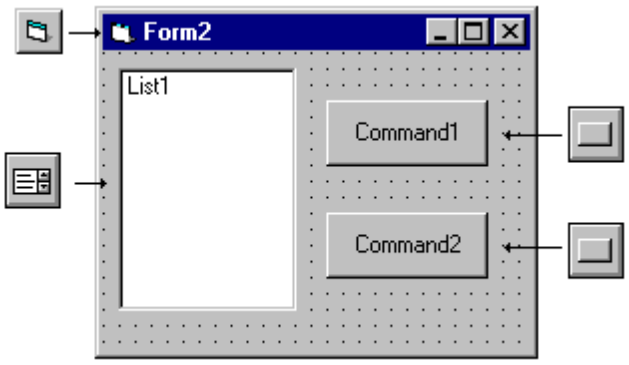
Neste procedimento, gravamos o conteúdo dos quadros de texto no Array de Variável Dados( ) com o índice do último registro existente mais 1. E utilizamos o método AddItem para adicionar um novo nome ao quadro de lista do formulário frmProcurar, que ainda será criado. O método AddItem serve para adicionar um novo item a um Quadro de Lista ou Quadro Combo:

```
controle.AddItem item [,index]
```

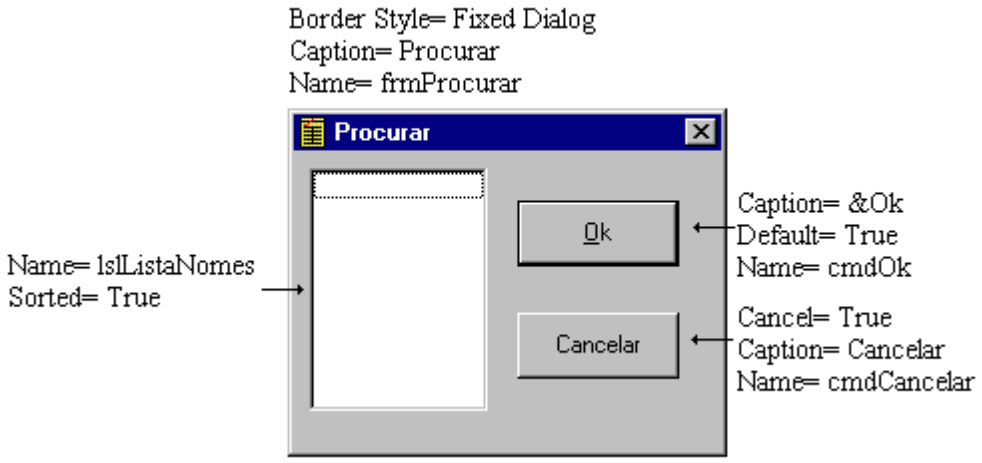
```
Private Sub cmdProcurar_Click()
   frmProcurar.Show 1
End Sub

Private Sub itmAbrir_Click()
   frmAbrir.Show 1
End Sub

Private Sub itmAdicionar_Click()
   cmdAdicionar_Click
End Sub

Private Sub itmProcurar_Click()
   cmdProcurar_Click
End Sub

Private Sub itmSair_Click()
   End
End Sub

Private Sub itmSalvar_Click()
   frmSalvar.Show 1
End Sub
```

## frmProcurar

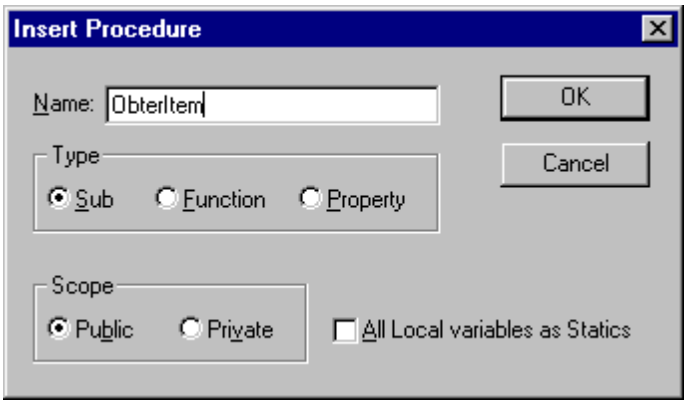
Insira um novo formulário, e deixe-o como a figura a seguir.




Altere as propriedades dos objetos.




Com a janela de código do formulário Procurar exibida, selecione a opção Procedure... do menu Insert, para inserir um procedimento geral de formulário. Aparecerá um quadro de diálogo, e no quadro Name digitaremos o nome do procedimento (ObterItem). Este procedimento poderá ser acessado por qualquer outro procedimento deste formulário, mas não de outro formulário.




Após entrar com o nome do procedimento encolha Ok, e será mostrada a seguinte janela de código:

Digite o código a seguir:

```
Public Sub ObterItem()
   For i% = 0 To 99
      If RTrim$(Dados(i%).Nome) = RTrim$(lstListaNomes.Text) Then Exit For
   Next i%
   frmCatálogo.txtNome.Text = Dados(i%).Nome
   frmCatálogo.txtTelefone.Text = Dados(i%).Telefone
   frmCatálogo.txtEndereço.Text = Dados(i%).Endereço
   frmProcurar.Hide
End Sub
```

A Estrutura de Repetição For...Next irá procurar em todos os campos Nome da variável Dados( ) um Nome igual ao nome selecionado no quadro de lista e, quando achar, sairá do loop (mantendo o valor de i%), e atribuirá os valores dos campos aos quadros de texto do frmCatálogo.

A função RTrim$ (expressão), elimina os espaços existentes após o String. Esta função tem outras semelhantes, LTrim$ (expressão) que elimina os espaços antes e, Trim$ (expressão) que elimina os espaços antes e depois do String. No procedimento acima usamos RTrim$, porque foi reservado um espaço de 30 caracteres para o campo Nome e na comparação entre Expressões, o número de espaços também será comparado.

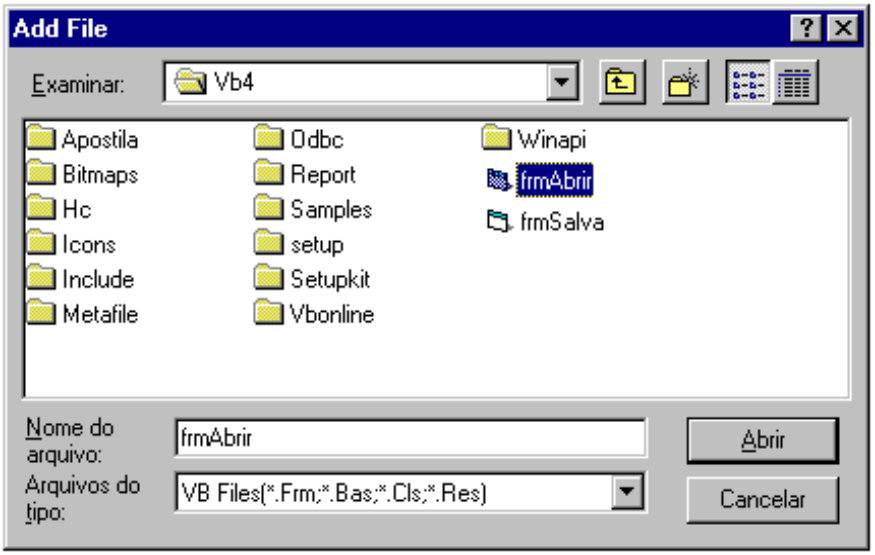
A seguir, encontraremos os códigos para os botões Cancelar e Ok, e o comando que chamará (Call) a rotina ObterItem após um duplo clique sobre um dos nomes da lista.

```
Private Sub cmdCancelar_Click()
   frmProcurar.Hide
End Sub

Private Sub cmdOk_Click()
   Call ObterItem
End Sub

Private Sub lstListaNomes_DblClick()
   Call ObterItem
End Sub
```

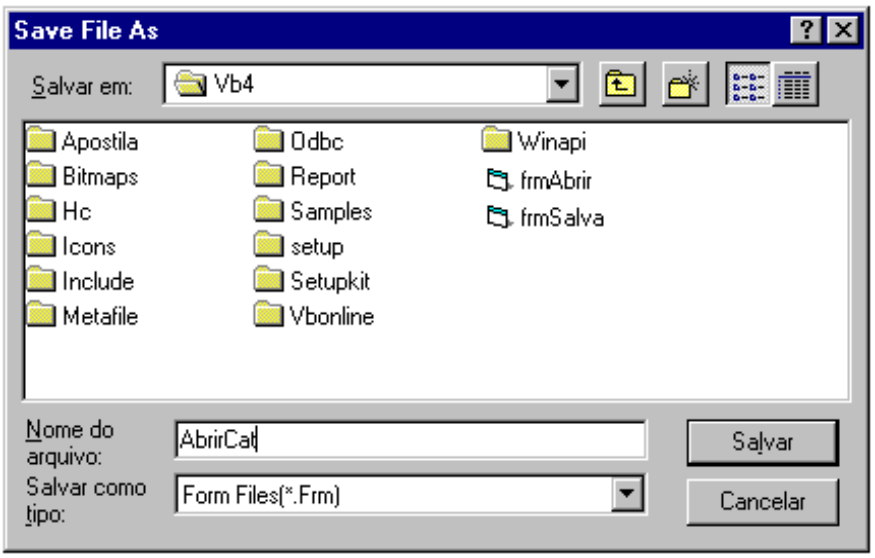
No exemplo do Bloco de Notas, nós construímos um quadro de diálogo para Salvar o arquivo e outro par Abrir um arquivo existente, neste projeto nós não precisaremos construir outros formulários, iremos apenas inseri-los neste projeto e alterar os seus códigos.

## frmAbrir

Para inserir um formulário existente construído em outro projeto, escolha a opção Add File... no menu File. E selecione o arquivo frmAbrir.




Observe que na Janela de Projeto apareceu mais um arquivo, o frmAbrir. Como o código deste arquivo será alterado, ele não irá mais funcionar com o Bloco de Notas, pois, o Bloco de Notas ao ser executado, procurará este arquivo alterado. Para evitarmos isto, deveremos salvar o formulário frmAbrir com outro nome. Selecione o arquivo do formulário Abrir - na janela de projeto - e escolha a opção Save File As... do menu File, dando um novo nome ao arquivo. Sendo assim, teremos no disco por enquanto, dois arquivos iguais com nomes diferentes.



Agora, os códigos (nem todos) já podem ser alterados.

```
Private Sub cmdCancelar_Click()
   frmAbrir.Hide
End Sub

Private Sub cmdOk_Click()
   On Error GoTo Erro
   If Right$(dirDiretório, 1) = "\" Then
      Arquivo$ = dirDiretório.Path + filArquivos.filename
   Else
      Arquivo$ = dirDiretório.Path + "\" + filArquivos.filename
   End If
   Open Arquivo$ For Random As #1 Len = Len(Dados(1))
   NúmeroRegistros = LOF(1) / Len(Dados(1))
   For i% = 1 To NúmeroRegistros
      Get #1, , Dados(i%)
   Next i%
   Close #1
   For i% = 1 To TotalRegistros
      frmProcurar.lstListaNomes.RemoveItem 0
   Next i%
   TotalRegistros = NúmeroRegistros
   For i% = 1 To TotalRegistros
      frmProcurar.lstListaNomes.AddItem Dados(i%).Nome
   Next i%
   frmCatálogo.txtNome.Text = Dados(1).Nome
   frmCatálogo.txtTelefone.Text = Dados(1).Telefone
   frmCatálogo.txtEndereço.Text = Dados(1).Endereço
   frmAbrir.Hide
Exit Sub

Erro:
   MsgBox "Erro de Arquivo", 48, "Erro"

End Sub
```

A estrutura deste procedimento é a seguinte:

a) Atribuir à variável Arquivo$, o nome com o caminho do arquivo que desejamos abrir.

b) Abrir este arquivo com a declaração Open. Abrindo-o como Random e número de identificação #1. Para arquivos deste tipo, temos que informar o tamanho de cada registro, e fazemos isso usando a função Len(variável) que retorna o tamanho em bytes da variável, no caso, Dados(1).

c) Determinar a quantidade de registros existentes no arquivo, dividindo-se o tamanho total do arquivo pelo tamanho de cada registro. A função LOF(número arquivo), retorna o tamanho em bytes do arquivo anteriormente aberto.

d) Ler os registros e guardá-los na variável Dados( ). Para ler um registro do arquivo, utilizamos a declaração Get.

**Get** #númeroarquivo, númeroregistro, variável

Variável indica onde será guardado o registro lido.
Númeroregistro é de uso opcional e indica o registro que queremos ler, no exemplo, a declaração Get incrementa automaticamente o númeroregistro.

e) Fechar o arquivo.

f) Remover os itens do quadro de lista de um outro arquivo anteriormente aberto. O método RemoveItem, remove itens de um quadro de lista ou combo de acordo com o index especificado, no exemplo ele elimina, a cada rodada do loop, o primeiro item.

Controle.**RemoveItem** index

g) Adicionar os Nomes ao quadro de lista.

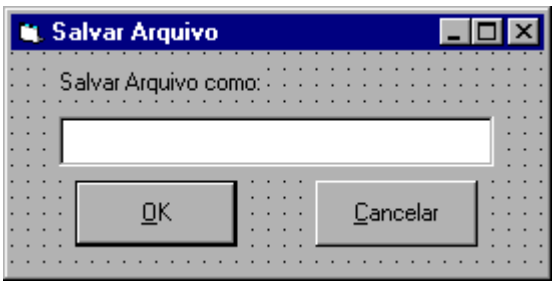
h) Atribuir valores aos quadros de texto com os campos do primeiro registro

i) Esconder o Formulário Abrir.

```
Private Sub dirDiretório_Change()
   filArquivos.Path = dirDiretório.Path
End Sub

Private Sub drvDisco_Change()
   dirDiretório.Path = drvDisco.Drive
End Sub

Private Sub filArquivos_DblClick()
   cmdOk_Click
End Sub
```

## frmSalvar

Vamos inserir agora, o formulário frmSalvar, seguindo os mesmos procedimentos do formulário Abrir, salve-o com outro nome antes de alterar os seus códigos.




```
Private Sub cmdCancelar_Click()
   frmSalvar.Hide
End Sub

Private Sub cmdOk_Click()
   On Error GoTo Erro
   Open txtNomeArq.Text For Random As #1 Len = Len(Dados(1))
   For i% = 1 To TotalRegistros
      Put #1, , Dados(i%)
   Next i%
   Close #1
   frmSalvar.Hide
   Exit Sub

Erro:
   MsgBox "Erro de Arquivo", 48, "Catálogo"
End Sub
```

Neste procedimento temos:

a) Abrir um arquivo de acesso aleatório com o nome que está no quadro de texto usando a declaração Open.

b) Gravar todos os valores do Array de Variável Dados( ) no arquivo aberto. Utilizamos para isso a declaração Put. A declaração Put serve para gravar dados em um arquivo composto por registros com campos diferenciados.

**Put** #númeroarquivo, numeroregistro, variável

Onde;

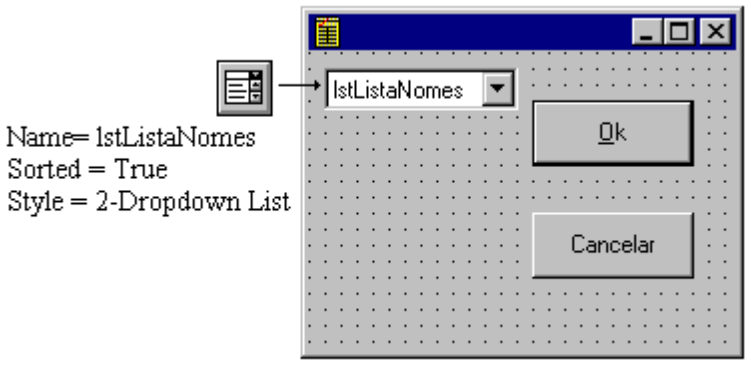
Númeroregistro é o número do registro que queremos gravar, caso não seja informado, será gravado o próximo registro do anteriormente gravado.
Variável é o local onde encontram-se os dados a serem gravados no arquivo.

c) Fechar o Arquivo

d) Esconder o formulário frmSalvar.

Teste o projeto experimentando gravar dois nomes de arquivos com a extensão DAT.

No formulário frmProcurar, podemos substituir o quadro de lista por um quadro combo, porque ambos trabalham de forma semelhante. Para isto, selecione o quadro de lista e escolha a opção Delete no menu Edit para retirar este controle do formulário, insira no mesmo local um quadro Combo, como segue:




O quadro Combo possui o mesmo nome do quadro de lista para não termos que alterar o código. A propriedade Style, determina qual o tipo de quadro combo que iremos trabalhar, que pode ser:

Tipo 0-Dropdown Combo; inclui uma lista de queda e um quadro de texto. O usuário pode selecionar um item da lista ou digitar no quadro de texto.

Tipo 1-Simple Combo; inclui uma lista sempre visível e um quadro de texto.

Tipo 2-Dropdown List; Mostra somente uma lista de queda onde o usuário escolherá o seu item.

Salve a execute o projeto com esta alteração.

## COMANDO DE IMPRESSÃO

Agora, vamos incluir um comando para impressão. Acesse novamente o Menu Editor e inclua um menu para impressão com o Name = mnuImprimir.

Inicie declarando a variável página como geral de formulário. E depois escolha a opção Procedure..., do menu Insert, para criar um procedimento de formulário com o nome de Cabeçalho, e digite o código a seguir:


```
Public Sub Cabeçalho()
   Página = Página + 1
   Printer.Print Format$(Now, "dd/mm/yyyy"); Tab(70); "Pág.:"; Página
   Printer.Print
   Printer.FontSize = 12
   Printer.Print Tab(13); "Nome"; Tab(35); "Telefone"; Tab(60); "Endereço"
   Printer.Line (0, Printer.CurrentY)-(Printer.ScaleWidth, Printer.CurrentY)
   Printer.CurrentX = 0
   Printer.CurrentY = 800
   Printer.FontSize = 10
End Sub
```

O procedimento acima começa imprimindo a data atual e o número da página, usando o método Print para o objeto Printer.

O método Printer também se aplica para Formulários e PictureBox da mesma forma que para a impressora. A sintaxe deste método é a seguinte:

[objeto,]**Print** [Tab(n)] [expressão] [ ; | , ]

Onde:

Objeto - objeto onde desejamos imprimir uma expressão, que pode ser form, picture box ou printer.
Tan(n) - número opcional para indicar o número de colunas antes de começar a impressão.
Expressão - número ou cadeia de caracteres que se deseja imprimir.
(; | ,) - caracteres que determinam o local do cursor para a próxima impressão. O (;) faz a impressão ser imediata, sem espaços, e a (,) faz o cursor passar para a próxima zona de tabulação antes de imprimir.

Digite o código para o menu imprimir:

```
Private Sub mnuImprimir_Click()
  Dim TamanhoPapel As Integer
   Página = 0
   TamanhoPapel = Printer.ScaleHeight - 2880
   Cabeçalho
   For i% = 1 To TotalRegistros
      If Printer.CurrentY >= TamanhoPapel Then
         Printer.NewPage
         Cabeçalho
      End If
      Printer.Print RTrim$(Dados(i%).Nome);
      Printer.Print Tab(36); RTrim$(Dados(i%).Telefone);
      Printer.Print Tab(60); RTrim$(Dados(i%).Endereço);
      Printer.Print
   Next i%
   Printer.EndDoc
End Sub
```

A variável TamanhoPapel contém a área de impressão, ou seja, o comprimento do papel (ScaleHeight) menos duas polegadas (2880Twips).

O método NewPage faz a impressora avançar para a próxima página quando o limite de impressão for atingido. E o método EndDoc finaliza o documento e libera a impressão.

Pronto, nosso catálogo já pode ser impresso, abra o arquivo que havia sido salvo anteriormente e imprima a lista de nomes.

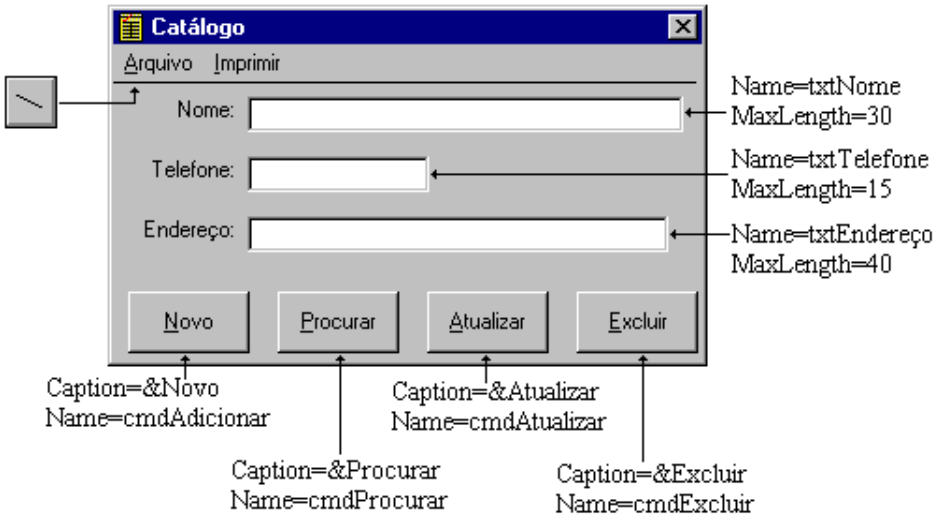
## ACESSO ALEATÓRIO

Até o momento, nós utilizamos um Array de Variável para trabalhar com os nossos registros, agora vamos modificar o projeto Catálogo para trabalhar com acesso a arquivo aleatório.

Para começar, retire os itens Abrir e Salvar do menu Arquivo. Para fazer isto, selecione o Menu Editor... e na janela de edição, selecione os dois itens, um de cada vez, e dê um clique em Delete.

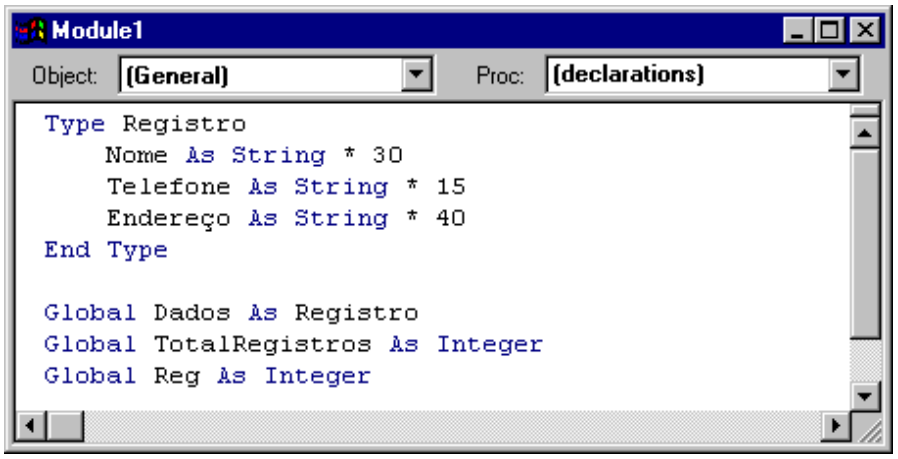
Como não temos mais os itens Abrir e Salvar, não precisaremos mais dos formulários frmAbrir e frmSalvar e eles podem ser excluídos do nosso projeto. Selecione um dos formulários na janela de projeto e escolha a opção Remove File no menu File para excluir o formulário selecionado, do projeto.

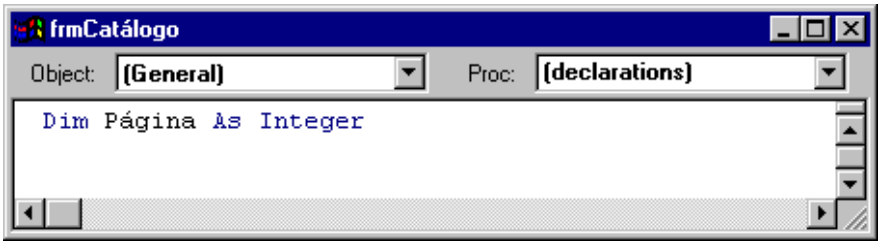
Excluídos os dois formulários (frmAbrir e frmSalvar), modifique o formulário frmCatálogo mostrado a seguir:




A propriedade MaxLength do Quadros de Texto, limita a quantidade de caracteres dentro de cada quadro. Usamos este limite porque a nossa variável Dados está com limites para cada campo. Caso não fosse dado limite aos quadros de texto, e o usuário escrevesse mais que o suportado pela variável, este excesso não seria gravado no arquivo de dados, dando uma falsa impressão ao usuário.

A seguir, modifique , se necessário, os procedimentos mostrados abaixo.




```
Public Sub Cabeçalho()
   Página = Página + 1
   Printer.Print Format$(Now, "dd/mm/yyyy"); Tab(70); "Pág.:"; Página
   Printer.Print
   Printer.FontSize = 12
   Printer.Print Tab(13); "Nome"; Tab(35); "Telefone"; Tab(60); "Endereço"
   Printer.Line (0, Printer.CurrentY)-(Printer.ScaleWidth, Printer.CurrentY)
   Printer.CurrentX = 0
   Printer.CurrentY = 800
   Printer.FontSize = 10
End Sub


Private Sub cmdAdicionar_Click()
   If cmdAdicionar.Caption = "&Novo" Then
      txtNome.Text = ""
      txtTelefone.Text = ""
      txtEndereço.Text = ""
      cmdAdicionar.Caption = "A&dicionar"
   Else
      TotalRegistros = TotalRegistros + 1
      Dados.Nome = txtNome.Text
      Dados.Telefone = txtTelefone.Text
      Dados.Endereço = txtEndereço.Text
      frmProcurar.lstListaNomes.AddItem txtNome.Text
      Put #1, TotalRegistros, Dados
      cmdAdicionar.Caption = "&Novo"
```

```
    Reg = TotalRegistros
  End If

  txtNome.SetFocus
End Sub

Private Sub cmdAtualizar_Click()
    Dados.Nome = txtNome.Text
    Dados.Telefone = txtTelefone.Text
    Dados.Endereço = txtEndereço.Text
    frmProcurar.lstListaNomes.RemoveItem Reg - 1
    frmProcurar.lstListaNomes.AddItem txtNome.Text, Reg - 1
    Put #1, Reg, Dados
    txtNome.SetFocus
End Sub
```

O procedimento cmdAdicionar_Click, executará instruções de acordo com a situação: quando o nome do botão for Novo, apenas limpa as caixas de texto e muda o nome para Adicionar; quando Adicionar, é gravado o conteúdo dos quadros de texto em arquivo, em seguida, muda o nome do botão para Novo.

Quando o usuário atualizar o registro exibido, ele deverá dar um clique no botão Atualizar. Este procedimento irá atualizar a variável Dados e o registro correspondente no arquivo de dados. Ele também atualizará o quadro de lista, removendo o item correspondente, e atualizando-o com o novo Nome.

```
Private Sub cmdExcluir_Click()
  Open "C:\VB4\Catálogo.tmp" For Random As #2 Len = Len(Dados)
  For i% = 1 To TotalRegistros
    If i% <> Reg Then
      Get #1, i%, Dados
      Put #2, , Dados
    End If
  Next i%
  Close #1
  Close #2
  Kill "C:\VB4\Catálogo.dat"
  Name "C:\VB4\Catálogo.tmp" As "C:\VB4\Catálogo.dat"
  Open "C:\VB4\Catálogo.dat" For Random As #1 Len = Len(Dados)
  TotalRegistros = TotalRegistros - 1
  txtNome.Text = ""
  txtTelefone.Text = ""
  txtEndereço.Text = ""
  Reg = 1
  Get #1, Reg, Dados
  frmCatálogo.txtNome.Text = Dados.Nome
  frmCatálogo.txtTelefone.Text = Dados.Telefone
  frmCatálogo.txtEndereço.Text = Dados.Endereço
```

```
End Sub
```

O procedimento cmdExcluir_Click, exclui um registro do arquivo de dados. Ele abre um novo arquivo com o nome Catálogo.tmp, que armazenará temporariamente os dados, após a exclusão do registro atual.

A estrutura de verificação If i% <> Reg Then, grava os registros do arquivo Catálogo.dat no arquivo temporário, quando eles forem diferentes do registro atual (que está vazio). Após isso, o procedimento fecha os dois arquivos, deleta (Kill) o arquivo Catálogo.dat e renomeia o Catalogo.tmp para Catálogo.dat.

```
Private Sub cmdProcurar_Click()
   frmProcurar.Show 1
End Sub

Private Sub Form_Load()
   Open "C:\VB4\Catálogo.dat" For Random As #1 Len = Len(Dados)
   TotalRegistros = LOF(1) / Len(Dados)
   For i% = 1 To TotalRegistros
      Get #1, , Dados
      frmProcurar.lstListaNomes.AddItem Dados.Nome
   Next i%
   Reg = 1
   Get #1, Reg, Dados
   frmCatálogo.txtNome.Text = RTrim$(Dados.Nome)
   frmCatálogo.txtTelefone.Text = RTrim$(Dados.Telefone)
   frmCatálogo.txtEndereço.Text = RTrim$(Dados.Endereço)
   Load frmProcurar
End Sub
```

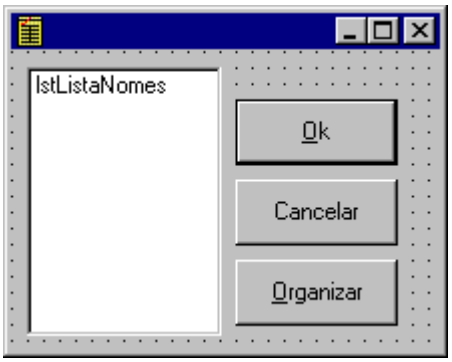
A função RTrim$(expressão), retorna a expressão eliminando todos os espaços do lado direito. Temos também LTrim$(expressão), para os espaços do lado esquerdo e Trim$(expressão), para eliminar os espaços de ambos os lados.

A declaração Load carrega o formulário frmProcurar na memória sem exibi-lo. Desta forma poderemos trabalhar de uma maneira mais ágil com este formulário, pois ele sempre estará disponível na memória.

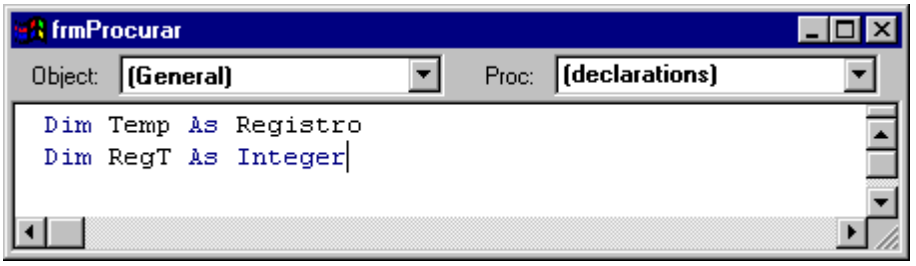
```
Private Sub Form_Unload(Cancel As Integer)
   Close #1
End Sub

Private Sub itmAdicionar_Click()
   cmdAdicionar_Click
End Sub

Private Sub itmProcurar_Click()
   cmdProcurar_Click
End Sub
```

```
Private Sub itmSair_Click()
   End
End Sub
```

Insira mais um Botão de Comando no formulário frmProcurar para ordenar os registros em ordem alfabética. Com as propriedades Name=cmdOrganizar e Caption=&Organizar.




Digite, ou altere os procedimentos mostrados a seguir:




```
Public Sub ObterItem()
   If lstListaNomes.ListIndex = -1 Then
      MsgBox "Selecione algum Nome", 48, "Erro"
      Exit Sub
   End If
   Reg = lstListaNomes.ListIndex + 1
   Get #1, Reg, Dados
   frmCatálogo.txtNome.Text = RTrim$(Dados.Nome)
   frmCatálogo.txtTelefone.Text = RTrim$(Dados.Telefone)
   frmCatálogo.txtEndereço.Text = RTrim$(Dados.Endereço)
   frmProcurar.Hide
End Sub
```

A propriedade ListIndex do quadro de lista, retorna o índice do item selecionado na lista, como esta propriedade inicia em 0 e os registros iniciam em 1, devemos somar 1 à ListIndex. O nosso projeto está usando o índice do nome na lista para poder

procurá-lo no arquivo, portanto a propriedade Sorted, do quadro de lista, deve ser igual a: Sorted=False.

Com a propriedade Sorted=False, os nomes na lista estarão na ordem em que foram inseridos no arquivo. A procura de nomes é mais fácil se os nomes estiverem em ordem alfabética, por isso foi criado o procedimento mostrado abaixo, que organiza o arquivo, colocando os nomes em ordem alfabética.

```
Private Sub cmdCancelar_Click()
   frmProcurar.Hide
End Sub

Private Sub cmdOk_Click()
   Call ObterItem
End Sub

Private Sub cmdOrganizar_Click()
   For i% = 1 To TotalRegistros
      lstListaNomes.RemoveItem 0
   Next i%
   For Reg = 1 To TotalRegistros
      For RegT = Reg To TotalRegistros
         Get #1, Reg, Dados
         Get #1, RegT, Temp
         If (Dados.Nome > Temp.Nome) Then
            Put #1, Reg, Temp
            Put #1, RegT, Dados
         End If
      Next RegT
   Next Reg
   For Reg = 1 To TotalRegistros
      Get #1, Reg, Dados
      lstListaNomes.AddItem Dados.Nome
   Next Reg
End Sub
```

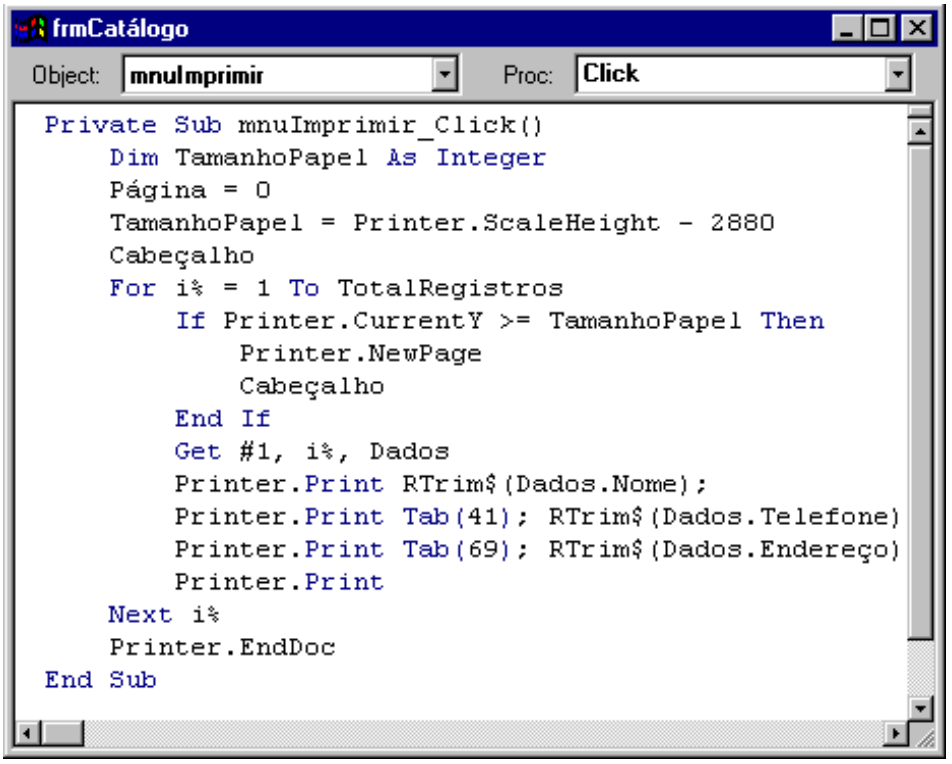
Este procedimento compara os registros do arquivo, e se um for maior que outro, ele troca ambos de posição, colocando o nosso arquivo em ordem alfabética.

```
Private Sub Form_Activate()
   lstListaNomes.Clear
   For i% = 1 To TotalRegistros
      Get #1, i%, Dados
      lstListaNomes.AddItem Dados.Nome
   Next i%
End Sub
```

O evento Activate ocorre toda vez que o formulário se torna o formulário ativo, ou seja, toda vez que o usuário chamar pelo formulário Procurar. Este procedimento irá limpar o conteúdo do quadro de lista usando o método Clear, e depois, adicionar todos os nomes existentes no arquivo, no momento da reativação do formulário.

**Private Sub lstListaNomes_DblClick()**
Call ObterItem
**End Sub**

Quanto a opção Imprimir, faça as alterações conforme a figura a seguir:




Pronto, salve o projeto e execute-o para ver o resultado.

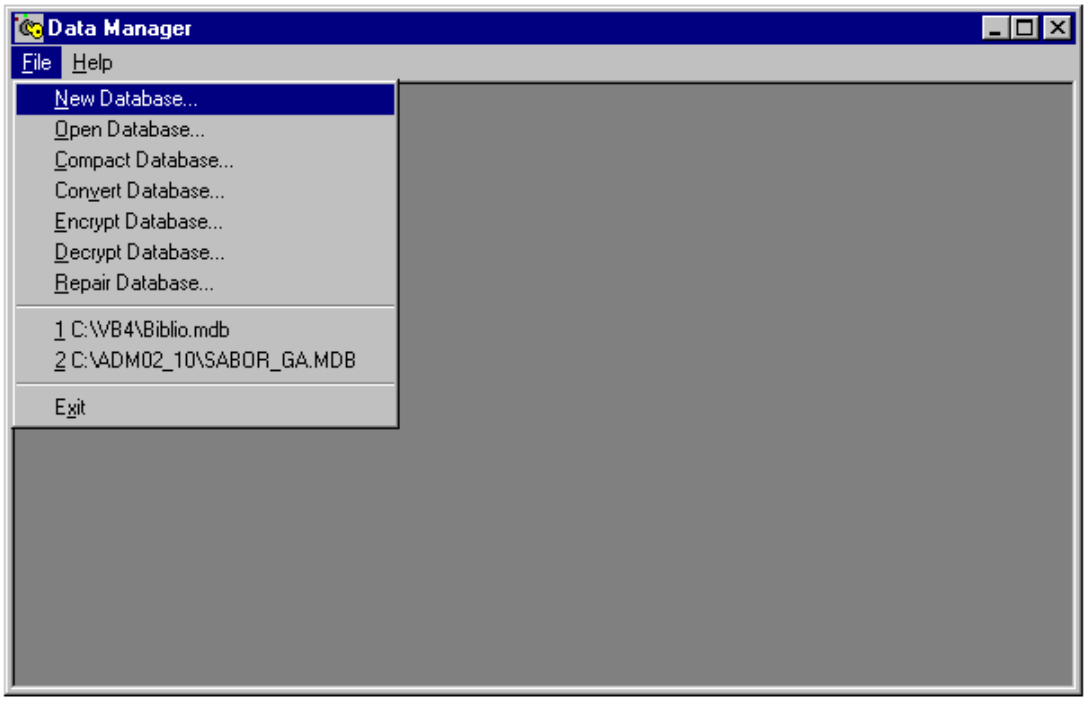
# Controle de Dados

## CONTROLE DATA

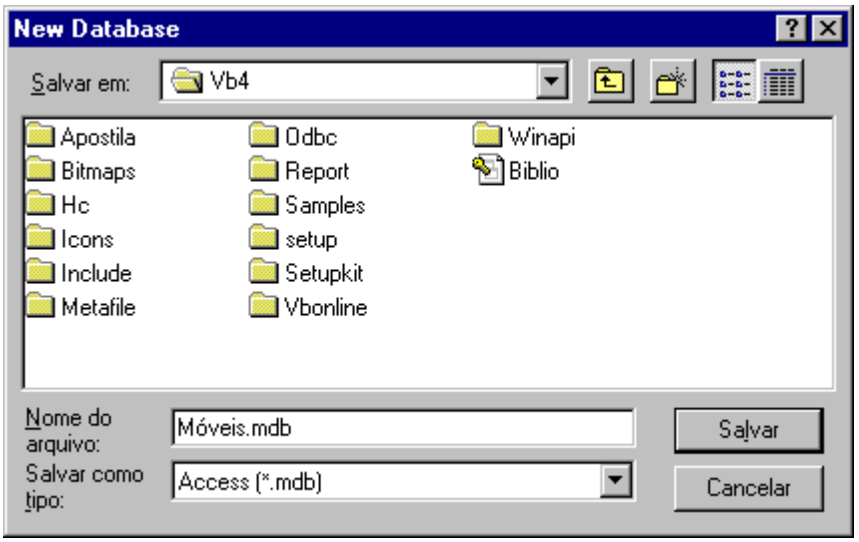
O Controle Data permite que você crie programas que acessem banco de dados como a Access, FoxPro, dBaseIII, dBaseIV, Paradox. Com o uso deste controle podemos economizar muitas linhas de código. Ele permite ao usuário visualizar registro por registro, ir para o último ou o primeiro registro de uma tabela, mostrando o conteúdo dos campos através de outros controles.

Os controles que podem exibir informações do Controle de Dados são: CheckBox, Label, TextBox, Image Control e PictureBox. Estes controles são ligados ao Controle Data através da sua propriedade DataSource, como veremos adiante.

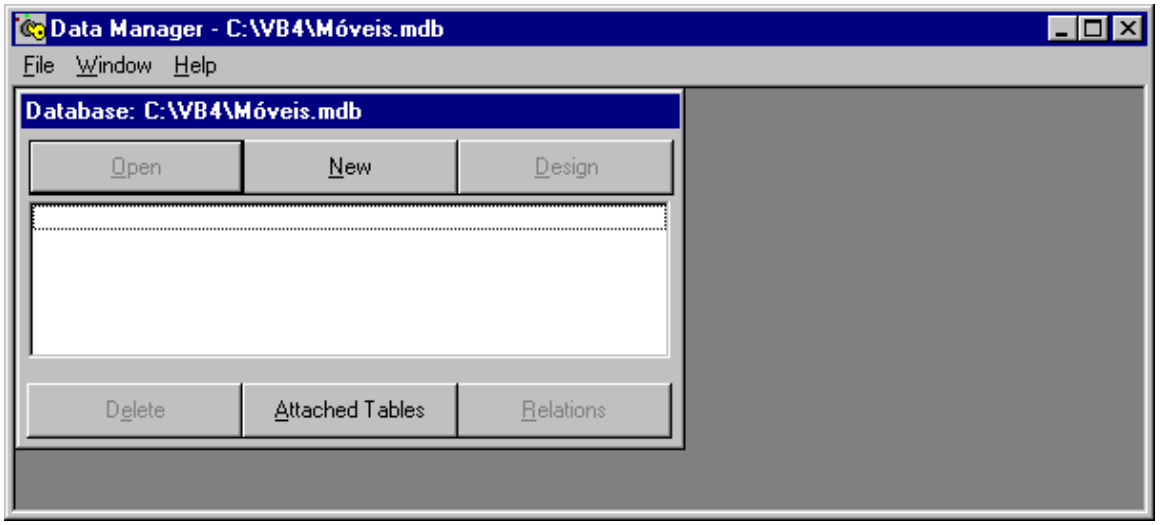
O Visual Basic possui o utilitário Data Manager, por onde poderemos criar um banco de dados para ser usado em nosso projeto exemplo. Para iniciar o Data Manager, escolha a opção Data Manager... do menu Add-Ins, aparecendo a janela mostrada abaixo.




Escolha a opção New Database... do menu File, para abrir a janela New Database.

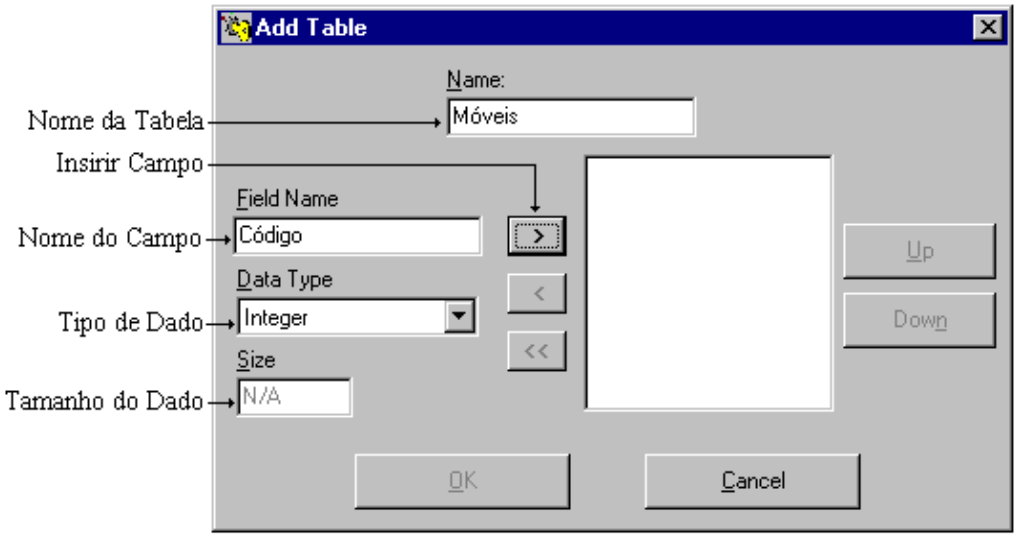



Dê o nome de Móveis.mdb e escolha Salvar, aparecendo a janela de construção do banco de dados - Database: C:\VB4\Móveis.mdb.



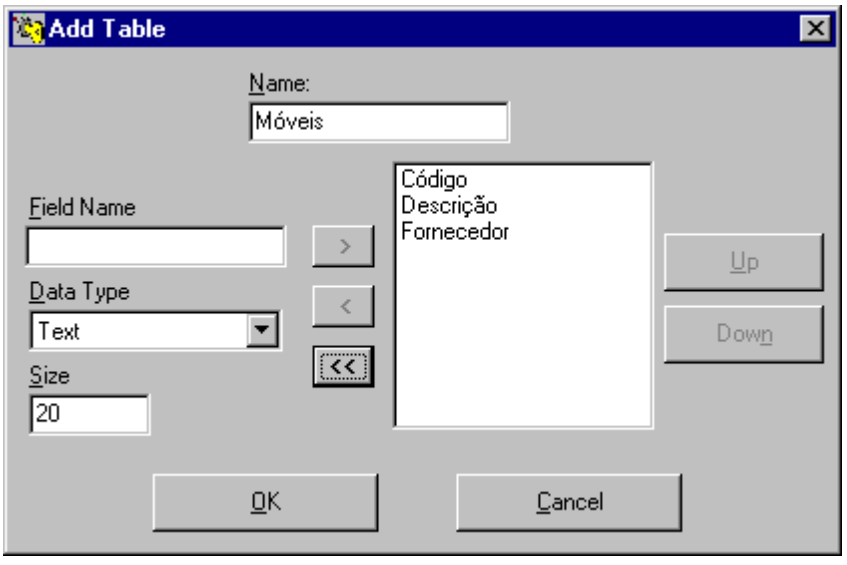

A partir desta janela adicionaremos nossa tabela. Uma tabela é composta de linhas (registros) e colunas (campos). No momento que estivermos criando uma tabela nova, definiremos quais os campos que farão parte dela e quais os tipos de dados que podem ser armazenados em cada campo.

Escolha o botão New, para construirmos nossa tabela.

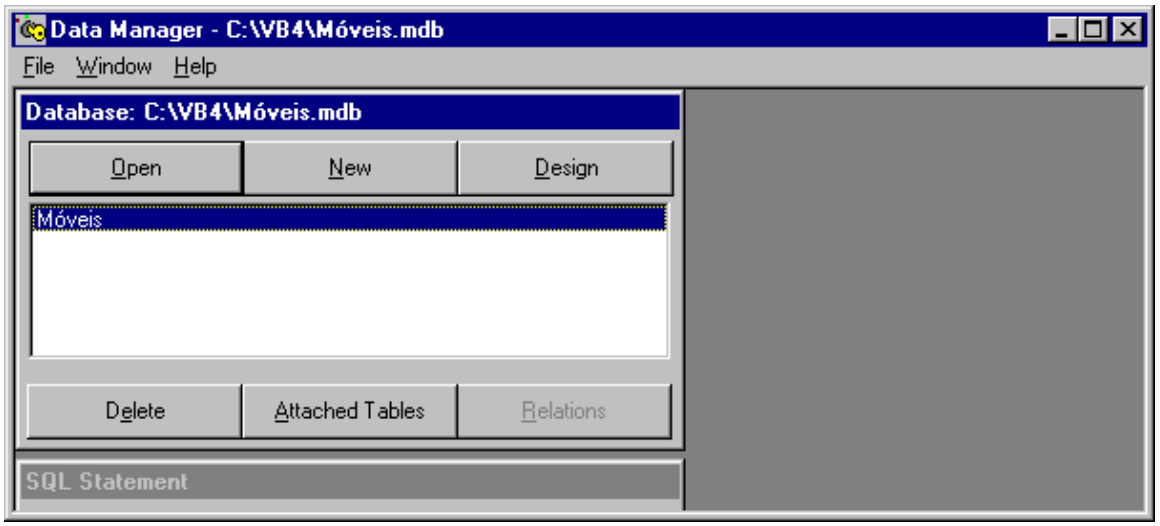


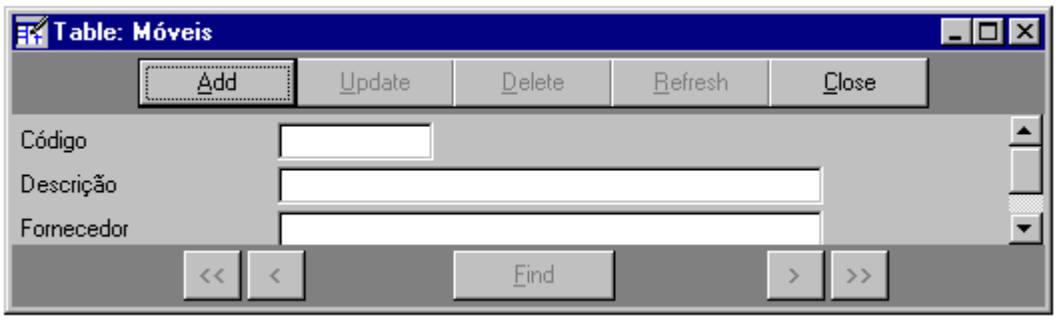

Entre com o nome da tabela - Móveis. E com os campos, acionando o botão Inserir campo, como mostra a tabela abaixo:

| Nome do Campo | Tipo de Dado | Tamanho |
|---|---|---|
| Código | Integer | N/A |
| Descrição | Text | 20 |
| Fornecedor | Text | 20 |




Após entrar com todos os campos, dê um clique em OK, retornando à janela Database. Ao retornar, selecione a tabela Móveis e escolha o botão Open (abrir), para inserirmos dados em nossa tabela.



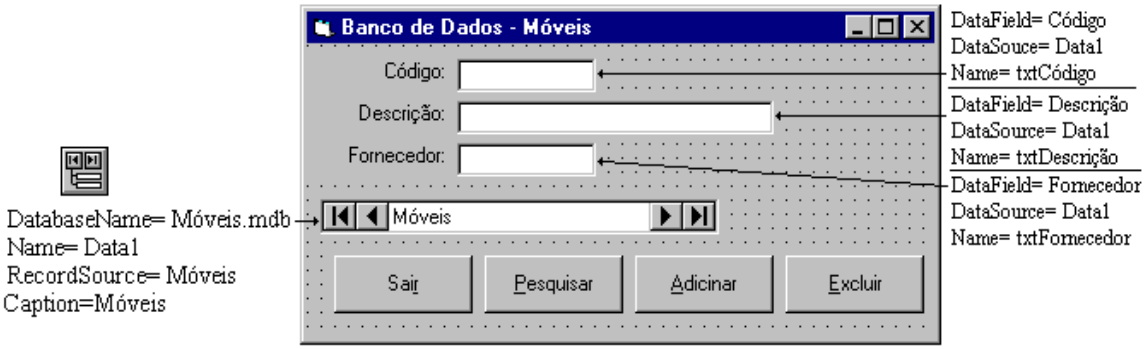




Selecione o botão Add, para digitar um registro. Logo após, dê um clique no botão Update e novamente Add, e assim sucessivamente até completar todos os registros mostrados na tabela abaixo:

| Código | Descrição | Fornecedor |
|---|---|---|
| 1001 | cama casal | rampazzo |
| 1002 | cama beliche | rampazzo |
| 1003 | cama solteiro | rampazzo |
| 1004 | cama beliche c/ gaveta | rampazzo |
| 2001 | mesa quadrada | santos andirá |
| 2002 | mesa redonda | santos andirá |
| 2004 | mesa oval | santos andirá |
| 3001 | cadeira giratória | santos andirá |
| 3006 | cadeira gir. c/ roda | sutran |
| 4005 | sofá | sutran |

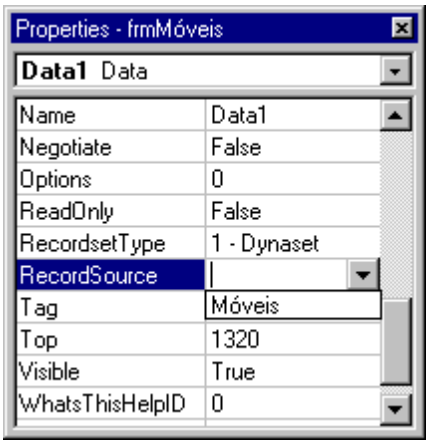
Após digitar todos os registro acima feche a tabela (Close), e também o Data Manager escolhendo a opção Exit no menu File.

Agora, construa o formulário mostrado abaixo:

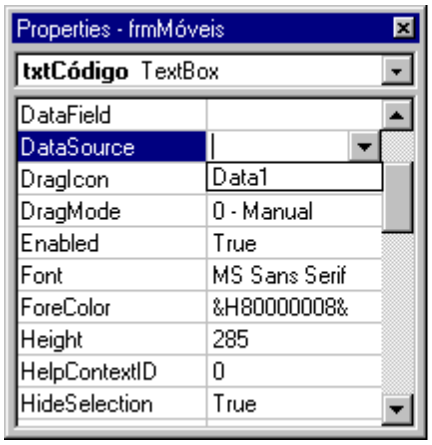


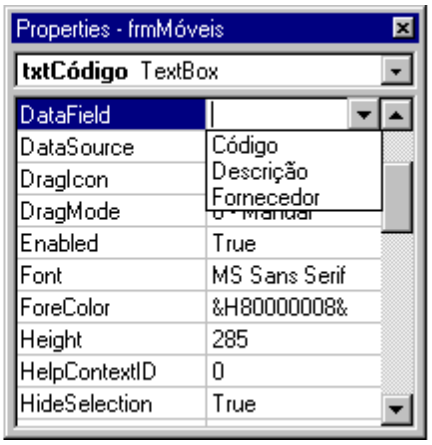

A propriedade DatabaseName do controle Data faz a ligação entre o controle e o banco de dados existente. Quando selecionamos a propriedade DatabaseName surge um quadro de diálogo para escolhermos qual o banco de dados será vinculado a este controle, no nosso projeto escolheremos o banco que acabamos de criar, ou seja, Móveis.mdb.

A propriedade RecordSouce define qual tabela do banco de dados selecionado será utilizada pelo controle Data, no nosso casso só existe a tabela móveis, caso existissem mais tabelas, elas seriam mostras na lista da propriedade.




Os quadros de texto possuem as propriedades DataSource e DataField para fazer o vínculo com o controle Data. A propriedade DataSource define qual o controle data que o quadro de texto estará relacionado, neste projeto existe apenas um controle Data, e a propriedade DataField determina qual o campo da tabela o quadro de texto estará vinculado.




```
Private Sub cmdSair_Click()
   End
End Sub

Private Sub cmdPesquisar_Click()
   Data1.RecordSource = "Select * from Móveis where Código > 1000 And Código
    < 2000"
   Data1.Refresh
End Sub
```

O programa pode utilizar instruções SQL (Structured Query Language) para selecionar alguns registros a serem exibidos. Esta linguagem foi criada para ser uma linguagem padrão de consulta, atualização e manipulação de banco de dados, no nosso projeto usaremos a linguagem SQL para extrair registros selecionados. Lembre-se que a propriedade RecordSouce original do controle Data é toda a tabela Móveis, através do comando Select (SQL) recuperamos todos os registros que satisfaçam a seguinte condição: 1000 < Código < 2000.

O método Refresh atualiza o conteúdo do controle Data após a pesquisa.

```
Private Sub cmdAdicionar_Click()
   Data1.Recordset.AddNew
   txtCódigo.SetFocus
End Sub
```

A propriedade RecordSet contém os registros atuais que o programa pode acessar. Como esta propriedade representa os registros do nosso banco de dados, e nós queremos inserir um novo registro, devemos adicionar este registro a esta propriedade e para isso usamos o método AddNew.

Quando o usuário der um clique no botão Adicionar, todos os quadros de texto ficarão em branco aguardando uma entrada de dados, para que estes dados “entrem” no banco de dados, basta alternar algum registro.

```
Private Sub cmdExcluir_Click()
   Data1.Recordset.Delete
   Data1.Recordset.MoveNext
End Sub
```

O procedimento acima apaga - Delete - o registro atualmente selecionado e move - MoveNext - para o próximo registro da tabela.

```
Private Sub txtCódigo_KeyPress(KeyAscii As Integer)
   If KeyAscii = 13 Then txtDescrição.SetFocus
End Sub
```

Grande parte dos usuários estão acostumados a pressionar a tecla Enter quando finaliza um campo, mas no Windows para alternarmos entre objetos utilizamos a tecla Tab. Para contornar este problema, deveremos criar o procedimento acima que parte do evento KeyPress (ato de pressionar e soltar uma tecla no teclado) que retorna o código da tecla pressionada (KeyAscii), o procedimento verifica se foi pressionada a tecla Enter (KeyAscii=13) e, se foi, muda o foco para o próximo quadro de texto.

```
Private Sub txtDescrição_KeyPress(KeyAscii As Integer)
   If KeyAscii = 13 Then txtFornecedor.SetFocus
End Sub
```

```
Private Sub txtFornecedor_KeyPress(KeyAscii As Integer)
   If KeyAscii = 13 Then cmdAdicionar.SetFocus
End Sub
```

Salve e execute o projeto verificando o seu funcionamento.