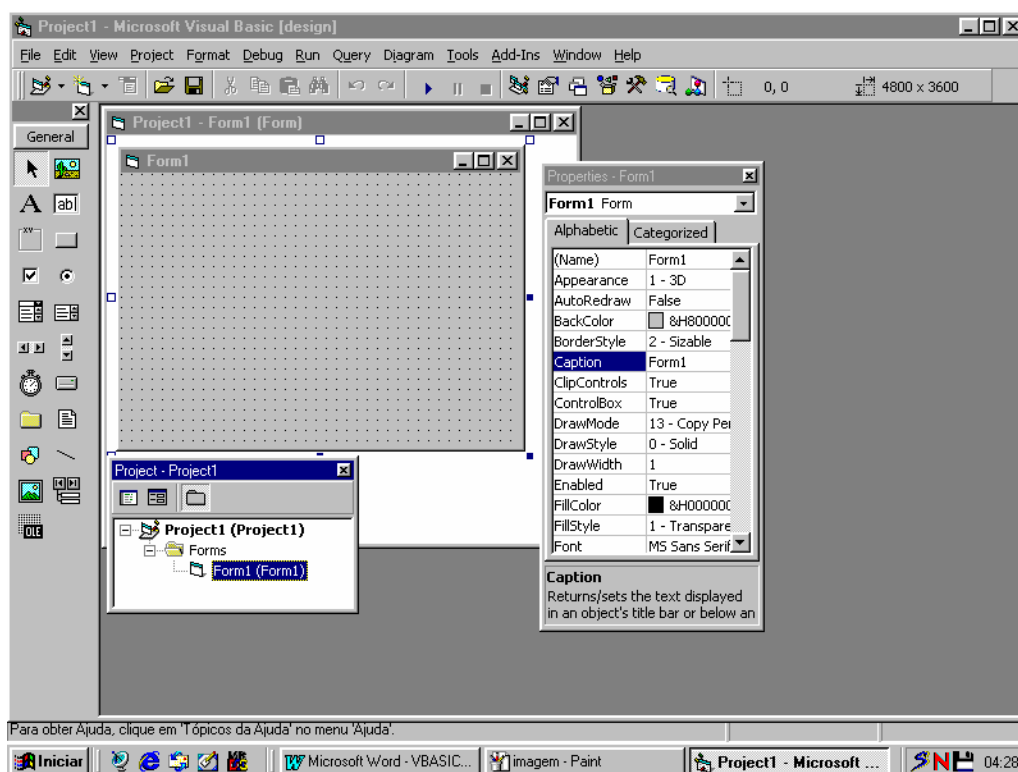


## 1.0 Apresentando o Visual Basic

### Objetivo do módulo

Apresentar os principais elementos do Visual Basic sem, no entanto, se aprofundar demais em nenhum tópico. Nos próximos capítulos, todos estes tópicos serão discutidos mais detalhadamente.

### 1.1 O Formulário e a Janela de Projeto do Visual Basic



O **formulário** é o centro de uma aplicação gráfica. É com ele que o usuário interage de modo a executar suas tarefas. Nele, você define e posiciona os controles que apresentarão ao usuário as opções disponíveis

A **Janela de Projeto** ( Project Window ) é uma lista usada pelo Visual Basic para controlar que arquivos fazem parte da sua aplicação. Esta lista poderá ser composta por arquivos do tipo :

- ✓ .FRM ( equivalente aos formulários da aplicação )
- ✓ .VBX ( que representam controles adicionais )
- ✓ .BAS ( que são blocos de código).

**Exemplo - Uma aplicação em VB com somente um formulário****Para Inicializar o VB :**

1. Dar duplo-click no ícone do Visual Basic.  
Visual Basic será inicializado com um formulário em branco na tela.

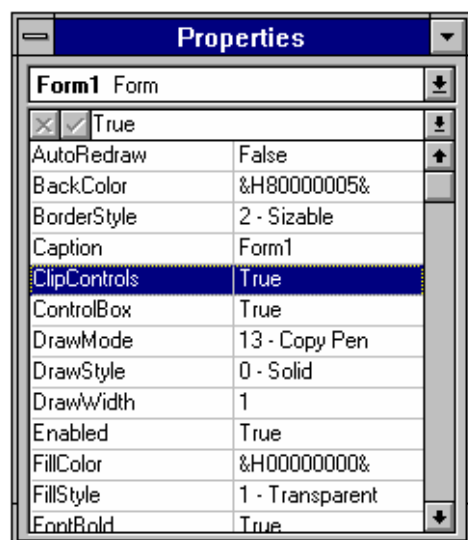
**Para executar a aplicação:**

1. Selecionar no menu, Run / Start.  
O programa será executado e mostrará o formulário.
2. Selecionar no menu, Run / End.  
O programa será finalizado.

**1.2 A Caixa de Ferramentas ( Toolbox )**

A **Caixa de Ferramentas**, como o próprio nome sugere, é aonde você pega os elementos básicos que compoem qualquer aplicação desenvolvida em Visual Basic. Existem duas formas de colocar um controle em um formulário : Clicando duas vezes sobre o controle ou arrastando-o (drag) para dentro do formulário.

### 1.3 A Caixa de Propriedades

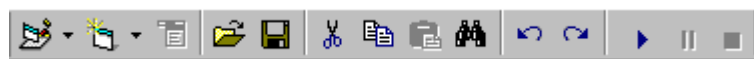


As propriedades definem as características de cada objeto/controle da aplicação. O conjunto de propriedades depende do controle selecionado. Por exemplo, um formulário tem propriedades diferentes de uma figura. As propriedades podem ser alteradas em tempo de construção ou de execução, sendo que algumas delas somente em tempo de execução.

Para alterar o valor de uma propriedade em tempo de construção, você deve:

1. Selecionar o controle cuja propriedade desejar alterar;
2. Rolar pela lista de propriedades até encontrar a propriedade desejada ( apertando CTRL+SHIFT+<primeira letra do nome da propriedade> o VB se posiciona automaticamente na propriedade desejada .);
3. Digitar o novo valor;
4. Clicar o checkmark (✓) ou pressionar ENTER para confirmar a alteração efetuada.

### 1.4 A Barra de Ferramentas



A **Barra de Ferramentas** coloca à sua disposição os comandos e funções normalmente mais utilizados. Todos estes comandos e funções também se encontram nos menus do VB.

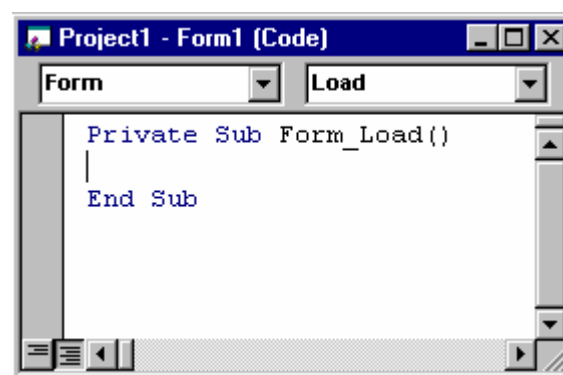
Os botões apresentados acima, da esquerda para a direita, executam as seguintes tarefas :

- Adiciona um Projeto
- Adiciona um Formulário
- Editor de Menu

- Abre um Projeto gravado
- Grava um Projeto
- Recorta
- Copia
- Cola
- Procura por uma sequência de caracteres
- Desfaz última operação
- Refaz operação desfeita
- Inicia a execução do Projeto
- Paraliza a execução do Projeto
- Finaliza a execução do Projeto





### 1.5 A Janela de Código

A janela de código é o lugar onde você escreve o código que a máquina deve executar para responder às ações do usuário. Para abrir uma janela de código, basta dar um click-duplo em cima do objeto do qual um evento deva ser tratado.








### 1.6 O Menu de Comandos do Visual Basic




**Menu File - Gerenciando Formulários e Projetos**

New Project	Ctrl+N	Abre um novo Projeto
 Open Project...	Ctrl+O	Abre um Projeto já existente
Add Project...		Incorpora um arquivo ao Projeto já existente
Remove Project		Exclui o Projeto atual
 Save Project Group		Salva um Projeto
Save Project Group As...		Salva o Projeto com nome diferente do atual
Save Form1	Ctrl+S	Salva o Formulário
Save Form1 As...		Salva o Formulário com nome diferente do atual
 Print...	Ctrl+P	Imprime o Projeto
 Print Setup...		Configuração da Impressora
Make Project1.exe...		Cria o executável do Projeto
Make Project Group...		Cria o executável de um grupo de Projetos

**Menu Edit - Editando o código Visual Basic**

 Undo Insert Text	Ctrl+Z	Desfaz a última ação
 Cut	Ctrl+X	Recorta
 Copy	Ctrl+C	Copia
 Paste	Ctrl+V	Cola
Delete	Del	Deleta parte selecionada
Select All	Ctrl+A	Seleciona Tudo
 Find...	Ctrl+F	Procura um conjunto de caracteres
Find Next	F3	Procura o próximo
Replace...	Ctrl+H	Procura e substitui um conjunto de caracteres

**Menu Run - Testando Aplicações**

 Start	F5	Inicia a execução do Projeto
 Break	Ctrl+Break	Paraliza a execução do Projeto
 End		Finaliza a execução do Projeto
Restart	Shift+F5	Renicializa a execução do Projeto

2.0 Construindo Aplicações Visual Basic

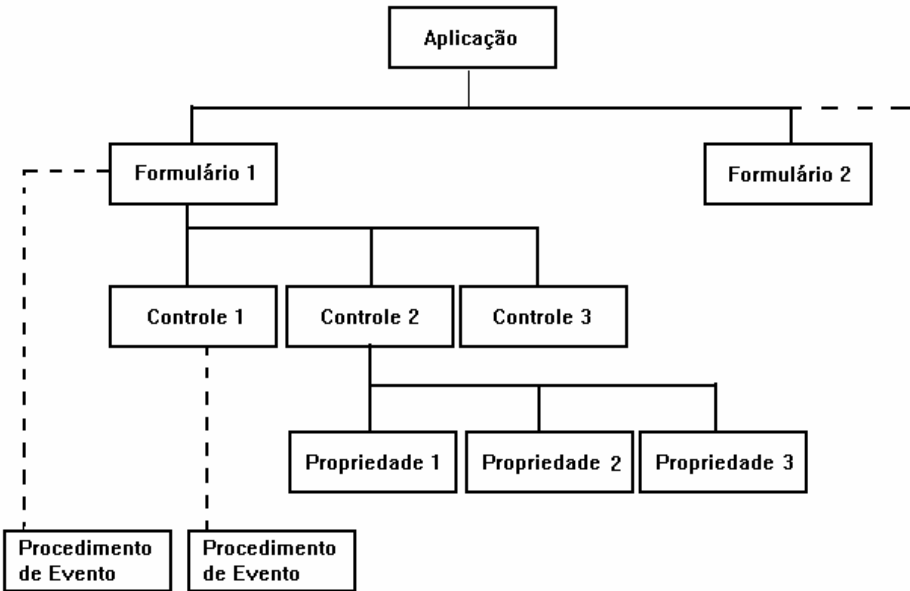
Objetivo do módulo

Apresentar vários conceitos interrelacionados que o ajudarão na transição do mundo procedural para o mundo orientado a eventos. Este módulo será a base lógica utilizada no resto do curso. Aqui, cria-se o contraste entre a programação voltada para ambiente Windows e aquela para DOS.

2.1 Aplicações Procedurais versus Aplicações Orientadas a Eventos

Procedural	Orientada a Eventos
Programação Linear□	Eventos podem ser acionados pelo usuário ou pelo sistema.□
Baseada em caracteres□	Baseada em objetos gráficos
Mono-tarefa□	Multi-tarefa□
Programador tem controle do ambiente□	Windows tem controle do ambiente□

2.2 A Terminologia Visual Basic



**Objetos** - São ferramentas que o Visual Basic fornece com as quais você construirá aplicações. Um formulário é um tipo de objeto; controles dentro do formulário, como: botões, caixas de texto e figuras também são objetos. Cada objeto possui uma lista de **propriedades**; alterando-as você estará caracterizando, criando a identidade do seu objeto. A objetos você pode aplicar **métodos**: mostrar um formulário é o método mostrar aplicado a um formulário, adicionar um item a uma lista é o método adicionar aplicado a uma lista, etc.. A objetos

também acontecem **eventos**. Eventos são percebidos pelo sistema, e você pode programar a sua aplicação para que ela reaja a estes eventos.

Vimos portanto vários conceitos importantes :

- ♦ **Objetos** - estrutura básica do VisualBasic. Podem ser : controles, formulários, impressora, etc.
- ♦ **Propriedades** - são as características que personalizam seu objeto. Cada objeto tem uma lista de propriedades própria.
- ♦ **Métodos** - Procedimentos fornecidos pelo Visual Basic que podem ser aplicados aos objetos. Cada objeto possui uma lista de métodos própria.  
Características de um método :
  - Você não pode criar um método, o VB já os cria para você. Métodos somente podem ser chamados.
  - Não é possível ver ou alterar o código de um método.
  - Os nomes de métodos são palavras reservadas do Visual Basic.

Sintaxe :

nome\_controle.método

Exemplos :

Form1.Hide  
List1.AddItem  
GRDAgenda.RemoveItem  
Picture1.Drag

- ♦ **Eventos** - Acontecem aos objetos e são detectados pelo Windows. Podemos programá-los para que os objetos reajam aos eventos.

### 2.3 Passos para a Criação de Uma Aplicação em Visual Basic

1. Abra um novo projeto.
2. Crie um formulário para cada janela que você conseguir visualizar dentro da aplicação.
3. Desenhe os controles nos formulários.
4. Crie uma barra de menus.
5. Defina as propriedades do formulário e dos controles.
6. Codifique os *event procedures* e *general procedures*.
7. Crie um arquivo executável.

### 2.4 Como Distribuir o Executável da Aplicação

Para instalar o executável para os usuários ou clientes, você precisa levar também uma cópia da biblioteca do Visual Basic VBRUN300.DLL. Este arquivo faz parte dos arquivos de instalação do Visual Basic e pode ser distribuído livremente.

### 2.5 Projeto de Interface com Usuário

A interface com o usuário deve seguir um padrão. A interface atualmente adotada mundialmente para desenvolvimento de aplicações Windows, é a própria interface criada pela Microsoft.

#### Determinações Básicas :

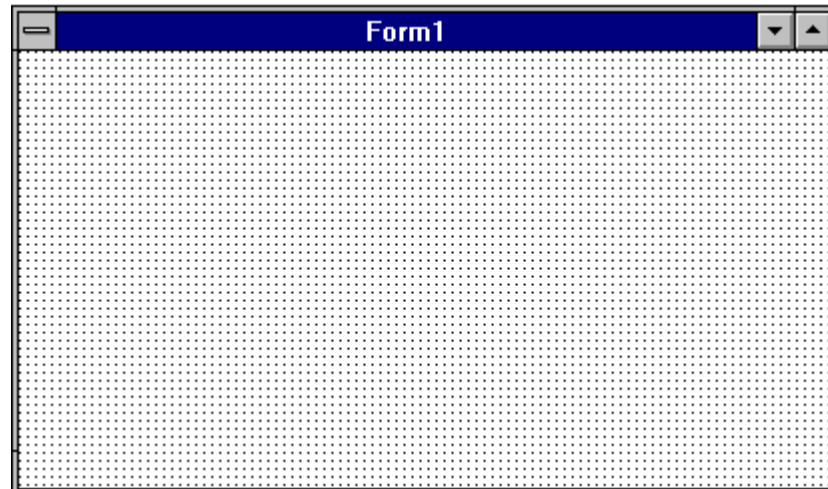
- Projetar a interface para o usuário, não para o sistema. - Típico caso de criar telas diferentes para Inclusão, Alteração e Exclusão ( quando, na maioria dos casos, estas três funções poderiam estar agrupadas em uma única tela).
- Ter em mente que agora é o usuário que mantém o controle da aplicação, e não mais o programa.
- Clareza - Ter certeza de que o propósito de cada tela está bem claro para o usuário.
- Estética
- *Feedback* - Dar sempre um retorno ao usuário do que está acontecendo no momento.
- Usar cores como um chamariz de atenção, porém cuidado para não exagerar !
- Preferir cores complementares, ao invés de cores inversas.
- A letra, normalmente utilizada em aplicações Windows, é qualquer uma *Sans Serif*. Ao usar tipos de letras diferentes, cuidado para não abusar da variedade.



### **3.0 Trabalhando com Formulários**

#### **Objetivo do módulo**

Apresentar a ferramenta principal para o programador : o Formulário. Apesar de ser apresentado de uma maneira simples, os conceitos e termos mostrados aqui serão necessários para um entendimento completo deste ambiente de programação.



3.1 Principais Propriedades

Propriedade	Default	Definição / Comentários
BorderStyle	2 - Sizeable	No padrão de interface Windows, o tipo da borda segue o seguinte padrão : <ul style="list-style-type: none"><li>Fixed Single - para caixas de diálogo não modais.</li><li>Sizeable - borda padrão para qualquer tela que não seja uma caixa de diálogo.</li><li>Fixed Double - para caixas de diálogo modais.</li></ul>
Caption	Form1	Título da tela.
ControlBox	True	False inibe a aparição do controlbox na barra de título do programa.
FontName	Helv	Tipo da letra utilizada.
Name	Form1	Nome pelo qual o formulário será conhecido dentro da aplicação.
Height	4425 twips	Altura do formulário.
Icon		Ícone default para o formulário. Se este for o <i>Startup Form</i> , será o ícone default da aplicação.
Left		Distância da margem esquerda da tela.
MaxButton	True	False inibe a aparição do botão de maximizar na barra de título do programa.
MinButton	True	False inibe a aparição do botão de minimizar na barra de título do programa.
MousePointer	0 Default	Altera o formato do mouse. Quando algo estiver sendo executado, o cursor deve assumir o formato de ampulheta (11).
Top		Distância da margem superior da tela.
Visible	True	False torna o formulário invisível.
Width	7485 twips	Largura do formulário.

3.2 Principais Comandos

3.2.1 Load

O comando LOAD é utilizado para carregar um formulário ou controle em memória. Normalmente aparece dentro de *event procedures* de outros formulários ou controles.

Sintaxe :

**Load** *objeto*

**Importante** O comando Load não mostra automaticamente o formulário; ele apenas carrega em memória. Para tornar o formulário visível, é necessário chamar o método Show.

### 3.2.2 Unload

O comando Unload é utilizado para retirar um formulário de memória. Note que ao retirar um formulário de memória apenas a parte gráfica da tela é descarregada, o código continua em memória.

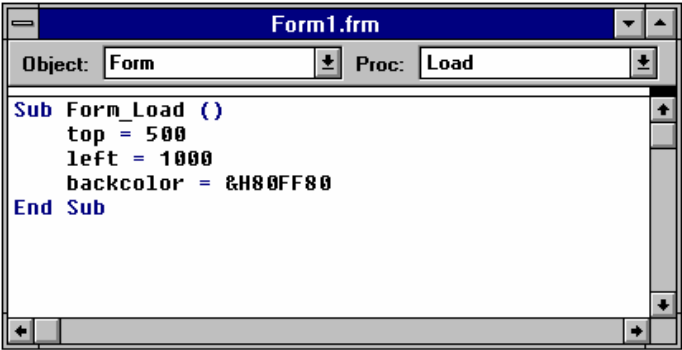
Sintaxe :

**Unload** *objeto*

3.3 Principais Eventos

3.3.1 Load

Acontece sempre antes que um formulário seja carregado em memória. É utilizado normalmente para inicializar os controles do formulário.



3.3.2 Unload

Este evento ocorre sempre momentos antes de um formulário ser descarregado. Nele devem ser tratados procedimentos de finalização de banco de dados, cálculos e etc.

3.3.3 Query Unload

O Evento QueryUnload de um formulário permite que você consiga detectar como o Unload do formulário foi iniciado. Este evento possui dois parâmetros : o *UnloadMode* e o *Cancel*. O UnloadMode indica como o Unload foi iniciado. O Cancel, que inicialmente possui valor *False*, se receber o valor *True* impede que o Unload do Formulário continue.

UnloadMode	Significado
0	O usuário iniciou o Unload a partir do Control Menu do formulário.
1	Pelo comando Unload, utilizado no código.
2	O Usuário está saindo do Windows.
3	O usuário selecionou Finalizar Tarefa a partir do Gerenciador de Tarefas.
4	Um formulário MDI-filho está sendo fechado porque o MDI-pai está sendo fechado.

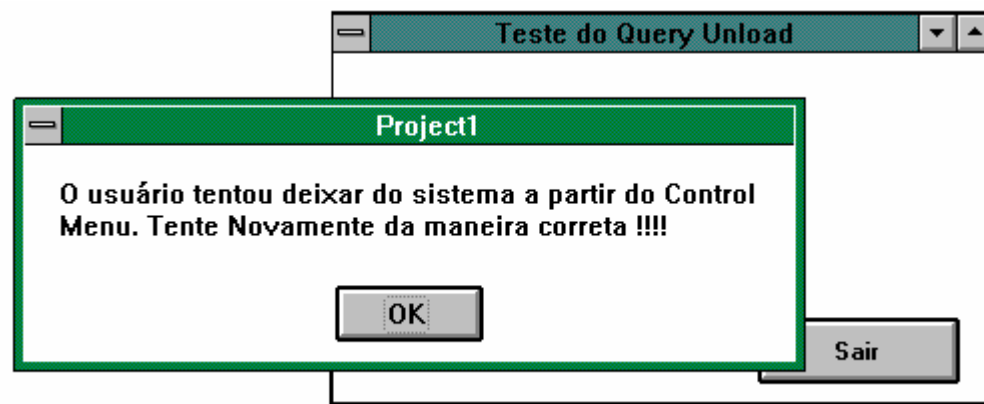
O evento QueryUnload é de grande importância para o programador de sistemas, pois impede o usuário de sair do sistema de uma forma incorreta, deixando cálculos incompletos, conexões com banco de dados pendentes, etc..

Exemplo :

```
Sub Form1_QueryUnload ( Cancel as Integer, UnloadMode as Integer )  
  If UnloadMode <> 1 Then  
    MsgBox "Utilize o botão 'Sair' para finalizar a tarefa"  
    Cancel = true  
  End If  
End Sub
```

### Exemplo - Testando o evento Query Unload

Para exemplificar a utilização do evento Query Unload, criaremos uma aplicação onde somente será possível finalizar a aplicação através do comando *End* a partir do código.



1. Inicializar o Visual Basic.
2. Criar um Botão em um formulário. Nomeá-lo *CMDsair* e o Caption = *Sair*.
3. Associar ao evento CLICK do objeto CMDsair o comando *End*
4. Associar ao evento Query Unload do form o seguinte código :

```
if unloadmode <> 1 then
    MsgBox "O usuário tentou deixar o sistema a partir do
    Control Menu. Tente novamente da maneira correta !!!!"
    cancel = True
End if
```

### Executando a Aplicação

1. Tecle F5 ou clique na opção RUN / START do Menu

### 3.3.4 Activate

Funciona como o GotFocus, mas para formulários. O Activate ocorre a um formulário sempre que este passa a ser o formulário ativo ( com foco ).

### 3.3.5 Deactivate

Funciona como o LostFocus, mas para formulários. O Deactivate ocorre a um formulário sempre que este deixa de ser o formulário ativo.

## 3.4 Principais Métodos

### 3.4.1 Método Hide

O método Hide esconde um formulário. O formulário continua carregado em memória porém não mais visível. O método Hide atribui o valor **False** à propriedade Visible do formulário. Se você tentar esconder um formulário que não esteja carregado em memória, este é carregado porém fica escondido.

Sintaxe :

[formulário].**Hide**

### 3.4.2 Método Show

O método Show é utilizado para mostrar um formulário. Assim como o método Hide, o Show trabalha com a propriedade Visible do formulário, atribuindo-lhe o valor **True**. Caso o formulário não esteja carregado em memória, o método show executa automaticamente o comando Load.

Sintaxe :

[formulário].**Show** [estilo%]

Observação : O parâmetro *estilo%* é um valor inteiro que determina se um formulário será mostrado de forma modal ou não. Estes conceitos serão explicados mais adiante.

## Exemplo - Abrindo e Fechando formulários

Para exemplificar a abertura e fechamento de formulários, construiremos uma aplicação composta por dois formulários. Ao clicar em qualquer lugar do formulário, o outro formulário será mostrado.

1. Inicializar o Visual Basic.

### Criando o segundo formulário

2. Clicar com o mouse sobre o menu *File*. Selecionar *New Form*.

### Programando o click do primeiro formulário

3. Clicar duas vezes sobre o primeiro formulário. Uma janela de código aparecerá.
4. Na lista de eventos, selecionar o evento *Click*.
5. Escrever as seguintes linhas de código :  
`Form2.show`

### Programando o click do segundo formulário

6. Clicar duas vezes sobre o segundo formulário. Uma janela de código aparecerá.
7. Na lista de eventos, selecionar o evento *Click*.
8. Escrever as seguintes linhas de código :  
`Form2.hide`

**Exemplo - Eventos Activate e Deactivate**

Para exemplificar os eventos Activate e Deactivate de formulários, construiremos uma aplicação composta por dois formulários. Ao passar de um formulário para outro, a aplicação indicará quando um formulário passou a ser o formulário ativo da aplicação.

1. Inicializar o Visual Basic.

**Programando o Load do primeiro formulário**

2. Clicar duas vezes sobre o primeiro formulário. Uma janela de código aparecerá.
3. Na lista de eventos, selecionar o evento *Load*.
4. Escrever a seguinte linha de código :  
`MsgBox "Load do form1"`

**Programando o Activate do primeiro formulário**

5. Na lista de eventos, selecionar o evento *Activate*.
6. Escrever a seguinte linha de código :  
`MsgBox "Activate do form1"`

**Programando o Deactivate do primeiro formulário**

7. Na lista de eventos, selecionar o evento *Deactivate*.
8. Escrever a seguinte linha de código :  
`MsgBox "Deactivate do form1"`

**Programando o Unload do primeiro formulário**

9. Na lista de eventos, selecionar o evento *Unload*.
10. Escrever a seguinte linha de código :  
`MsgBox "Unload do form1"`

**Criando o segundo formulário**

11. Clicar com o mouse sobre o menu *File*. Selecionar *New Form*.

**Programando o Activate do segundo formulário**

12. Clicar duas vezes sobre o segundo formulário. Uma janela de código aparecerá.
13. Na lista de eventos, selecionar o evento *Activate*.
14. Escrever a seguinte linha de código :  
`MsgBox "Activate do form2"`

**Programando o Deactivate do segundo formulário**

15. Na lista de eventos, selecionar o evento *Deactivate*.
16. Escrever a seguinte linhas de código :  
`MsgBox "Deactivate do form2"`
17. Na lista de eventos, selecionar o evento *CLICK*.
18. Escrever as seguintes linhas de código:  
`Form2.Hide`  
`Form1.Show`

**Criando um botão no primeiro formulário**

19. Clicar duas vezes sobre a ferramenta Command Button da Caixa de Ferramentas.



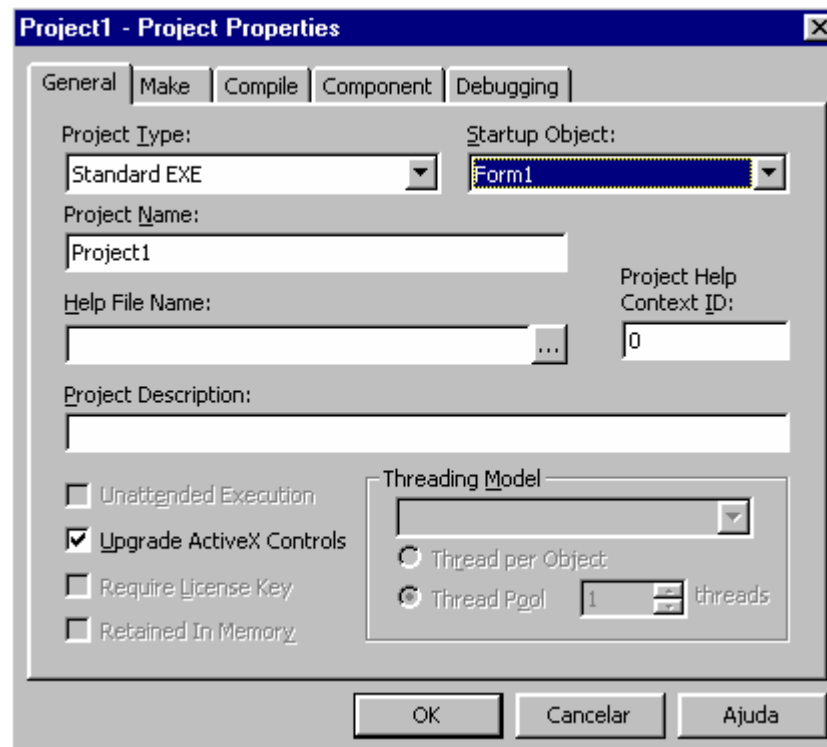
20. Alterar propriedade Caption = *Outro Formulário*.
21. Clicar duas vezes sobre o botão que aparecerá no formulário.
22. Na lista de eventos, selecionar o evento *Click*.
23. Escrever as seguintes linhas de código :

```
Unload Form1  
Form2.show
```

### Executando a Aplicação

24. Tecle F5 ou clique na opção *Run / Start* do menu.

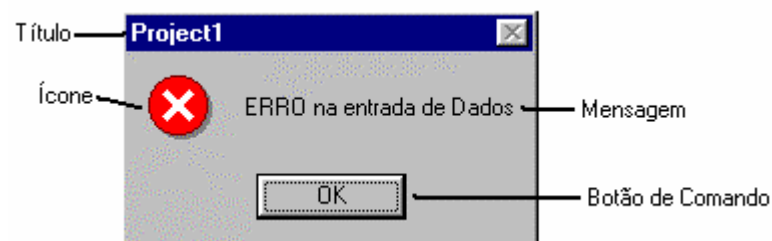
## 3.5 Determinando o Formulário Inicial de uma Aplicação



Normalmente uma aplicação contém mais de um formulário. Se este for seu caso, você precisa definir o formulário inicial da aplicação (que por default é aquele que foi criado primeiro). Para fazer isto selecionar a opção *Project Properties* do menu *Project*. O text box *Startup Object* deve conter o nome (propriedade *Name*) do formulário inicial.

## 3.6 Message Box

O *MsgBox()* é uma função que produz uma caixa de mensagem pop-up. A figura abaixo mostra uma caixa de mensagem. Como podemos ver, uma caixa de mensagem exibe um ícone e uma mensagem com pelo menos um botão de comando. O botão de comando dá ao usuário uma chance de ler a mensagem dentro da caixa e dar um clique no botão depois de concluir.

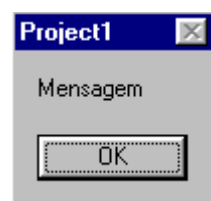


Os argumentos que você fornece para a função *MsgBox()* determinam qual ícone a função exibirá, a mensagem é o número de botões de comando. Portanto, o programador controla exatamente como a caixa de mensagem aparecerá para o usuário. Quando a *MsgBox()* for concluída, seu valor de retorno especificará em qual botão de comando o usuário clicou. Portanto o seu programa fará sempre um teste no valor de retorno da função *MsgBox()*, caso a função tenha dois botões de comando. O programa pode então utilizar uma instrução *If* a fim de determinar o melhor curso de ação com base na seleção do botão de comando do usuário.

O formato básico da função *MsgBox()* é a seguinte:

*MsgBox("mensagem")*

Essa linha de comando exibirá na tela:



Note que a mensagem ficou simples, ou seja, sem ícone e o título ficou sendo o nome do projeto. Para melhorar a aparência de uma *MsgBox()* podemos utilizar argumentos opcionais. Esses argumentos podem ser:

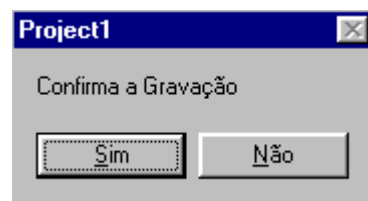
*Resposta=MsgBox("Mensagem", TipodoBotão + ícone + BotãoDefault , "Titulo")*

### **Tipo do Botão**

Valor	Constante Identificada	Descrição
0	VbOKOnly	Botão OK
1	VbOKCancel	Botões OK e Cancel
2	VbAbortRetryIgnore	Botões Abort, Retry e Cancel
3	VbYesNoCancel	Botões Yes, No e Cancel
4	VbYesNo	Botões Yes e No
5	vbRetryCancel	Botões Retry e Cancel

Exemplo de uma função MsgBox() utilizando botões:

*Resposta = MsgBox(“Confirma Gravação”, 4)*



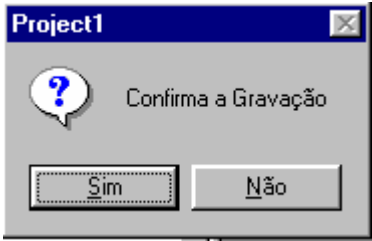
Você pode utilizar tanto o valor numérico como a cosntante identificada para expecificar o tipo do botão.

### Ícone

Valor	Constante Identificada	Descrição	Ícone
16	VbCritical	Ícone de mensagem crítica	
32	VbQuestion	Cone de ponto de interrogação	
48	VbExclamation	Mensagem de aviso	
64	VbInformation	Mensagem de informação	

Exemplo de uma função MsgBox() utilizando botões e ícone:

```
Resposta = MsgBox("Confirma Gravação", 4 + 32)
```



**Botão Default**

O primeiro botão de comando em uma caixa de mensagem é sempre o botão de comando padrão. O Visual Basic seleciona o primeiro botão de comando (à esquerda) e, se o usuário pressionar Enter sem dar um clique em um botão de comando, o botão de comando selecionado será acionado.

Você pode alterar o botão que aparece como botão padrão quando a caixa de mensagem aparecer pela primeira vez ao adicionar um dos seguintes valores ao argumento Botão Default:

Valor	Cosntante Identificada	Descrição
0	VbDefaultButton1	O primeiro botão é o padrão
256	VbDefaultButton2	O segundo botão é o padrão
512	VbDefaultButton3	O terceiro botão é o padrão

Exemplo de uma função MsgBox() utilizando botões , ícone e definido o segundo botão como padrão:

```
Resposta = MsgBox("Confirma Gravação", 4 + 32 + 256)
```



Observe que o botão *Não* é o botão ativo quando a mensagem é exibida.

**Título**

O argumento Título é utilizado quando você deseja especificar um título para sua caixa de mensagem, como por exemplo:

```
Resposta = MsgBox(“Deseja Realmente Sair” , 4 + 32 + 256 , “Sair”)
```



**Controle sobre os Botões**

Quando você utiliza uma caixa de mensagem com mais de um botão você pode utilizar uma variável (*Resposta*) para armazenar o comando que recebeu o clique, e utilizar essa informação em uma instrução *If* para executar um determinado comando para cada botão. A tabela a seguir mostra os valores de cada botão:

Valor	Cosntante Identificada	Descrição
1	VbOk	O usuário deu um clique em OK
2	VbCancel	O usuário deu um clique em Cancel
3	VbAbort	O usuário deu um clique em Abort
4	VbRetry	O usuário deu um clique em Retry
5	VbIgnore	O usuário deu um clique em Ignore
6	VbYes	O usuário deu um clique em Yes
7	vbNo	O usuário deu um clique em No

Independente de quantos botões de comando você visualizar em uma caixa de mensagem, o usuário poderá clicar somente em um botão. Assim que o usuário clicar em qualquer um dos botões da caixa de mensagem, ela desaparecerá e o valor de retorno será preenchido com o valor do botão que recebeu o clique.

O código abaixo exemplifica a utilização dos valores dos botões:

```
resposta = MsgBox("Deseja Realmente Sair", 4 + 32 + 256, "Sair")
If resposta = "6" Then
    'O código colocado aqui tratará a resposta YES
Else
    'O código colocado aqui tratará a resposta NO
End If
End Sub
```

## 4.0 Criando Menus

### Objetivo do módulo

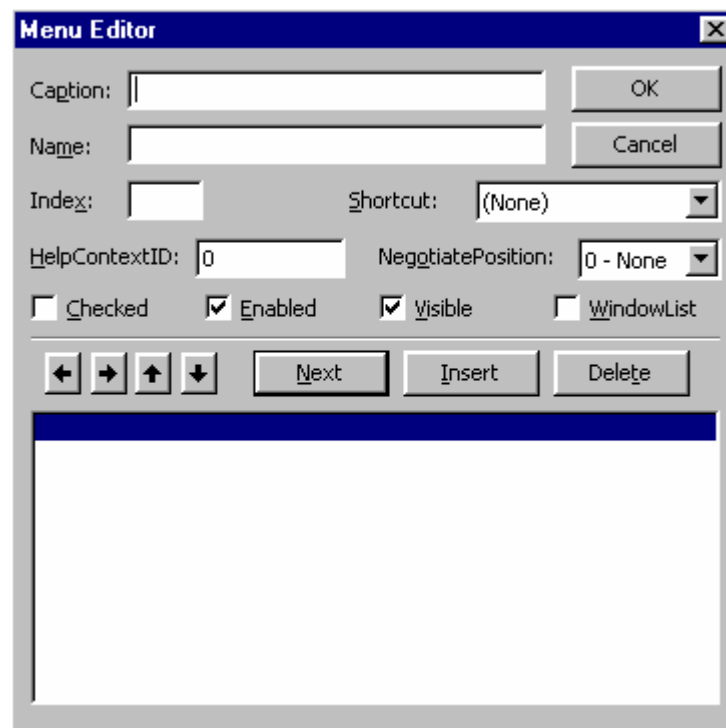
Mostrar como trabalhar com aplicação composta por mais de um formulário, montando menus. Nos módulos anteriores, trabalhamos com um único formulário, basicamente, para mostrar o funcionamento de uma aplicação Visual Basic.

### 4.1 Padrão de Menus Windows

A seguir destacamos alguns itens que devem ser observados para que um menu esteja dentro do padrão Windows.

- A opção *Sair* deve ser a última opção do primeiro menu;
- Três pontos devem ser colocados ao final do comando para indicar a existência de uma caixa de diálogo, quando esta existir;
- Para indicar que uma opção está ativa, colocar um check-mark (✓) ao lado da opção.
- Barras de separação devem ser utilizadas para separar, visualmente, itens relacionados ou opções perigosas para o usuário.
- Teclas de acesso ou atalho devem ser definidas para todos os itens do menu. ( Pelo padrão Windows, as aplicações devem possibilitar seu uso, mesmo quando o usuário não tem um mouse).

## 4.2 A Implementação do Visual Basic para a Criação de Menus



 Par













A janela de projeto de menu

A janela de projetos de menu está dividida em duas partes:

Ítems de Menu

**Item**

**Descrição**

Caption

Nome que aparece no menu.

Name

Nome usado em código.

Index

Usado na adição de menus dinamicamente.

Negotiate Position

ShortCut

Cria combinações de tecla para acesso rápido.

Checked

Indica a existência de uma check-mark ao lado do item.

Enabled

Indica se uma opção de menu está habilitada.

HelpContextId

Especifica um identificador para um item de menu em um sistema de Help.

Visible

Indica se uma opção de menu está visível ou não.

WindowList

Especifica se o menu conterà uma lista de formulários MDI abertos.

Na parte de manipulação de ítems, você define a hierarquia dos ítems de menu, inclui novos ítems e apaga outros. Para qualquer uma das operações, o ítem deve estar selecionado.

Manipulação de Lay-Out

**Item**

**Função**

Seta para esquerda

Aumenta o nível hierárquico de um menu.



Seta para a direita

Diminui o nível hierárquico de um menu.

Seta para Cima	Move a posição do cursor para um item de menu acima.
Seta para Baixo	Move a posição do cursor para um item de menu abaixo.
Next	Move a seleção para o próximo item da lista.
Insert	Insere uma linha em branco acima do item
Delete	Apaga um item de menu.

Exemplo - Construindo Menus

Para exemplificar a construção de menus, contruiremos uma tela simples, onde apenas as funções de construção serão enfocadas.



- 1. Abrir o Visual Basic.
- 2. Do menu Window, seleccionar a opção Menu Design.
- 3. Criar os seguintes ítems de menu :

Item de Menu	Caption	Name
Arquivo☐	&Arquivo☐	MNUArquivo☐
Abrir☐	&Abrir	MNUAbrir
-☐ -☐		
MNUSeparador		

Sair

&Sair

MNUSair

Editar

&Editar

MNUEditar

Copiar

&Copiar

MNUCopiar

Colar

Co&lar

MNUColar

4. Pressione F5 para executar o programa e verifique o encadeamento dos menus. Note que, para que os menus executem alguma ação é necessário que o evento Click de cada objeto menu seja codificado. No caso acima, deveríamos codificar o evento click dos seguintes objetos : MNUAbrir, MNUSair, MNUCopiar e MNUColar.

## 5.0 Utilizando Controles

### Objetivo do módulo

Apresentar os controles, suas principais propriedades e eventos. Não é intenção do curso, que ao final deste módulo, você tenha domínio absoluto de todos os controles e propriedades existentes no Visual Basic.

Ao longo do módulo, uma série de pequenos exercícios e apresentações serão feitas de modo que você se familiarize com o uso de cada controle, e perceba como as propriedades os afetam.

Os controles apresentados serão :

- Labels
- Text boxes
- Frames
- Command Buttons
- Check Boxes
- Option Buttons
- Combo Boxes
- List Boxes
- Horizontal/Vertical Scroll Bars
- Timers
- Picture Boxes
- Grid
- Panel 3d

Além de controles, neste capítulo também introduziremos dois conceitos muito importantes, que são diretamente relacionados aos controles. São eles :

- / Foco
- / Control Arrays

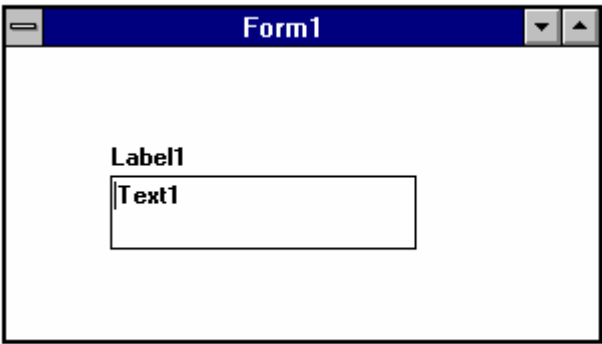
## 5.1 Tipos de Controles

Caixa de Ferramentas



	Pointer		PictureBox
	Label		TextBox
	Frame		CommandButton
	CheckBox		OptionButton
	ComboBox		ListBox
	HScrollBar		VScrollBar
	Timer		DriveListBox
	DirListBox		FileListBox
	Shape		Line
	Image		Date
	FlexGrid		Masked Edit

5.2 Label e Text Box



Labels

Labels são normalmente utilizados para exibir textos que o usuário não deve modificar. Um exemplo clássico seria o título de campos em formulários.

Propriedade	Descrição
Alignment	Alinhamento do texto dentro do label.
BackColor	Cor de fundo do label.
BorderStyle	Tipo de borda do label. Recomendável não utilizar borda.
Caption	Texto que aparece no label.
Font	Tipo de letra utilizada.
FontSize	Tamanho da letra utilizada.
ForeColor	Cor da letra.
Height	Altura do label.
Left	Posição do extremo esquerdo do label em relação ao formulário.
Name	Nome do controle a ser utilizado no programa.
Top	Posição do topo do label em relação ao formulário.

**Text Boxes**

O text box é um controle utilizado, basicamente, para a exibição de informação pelo sistema ou para a entrada de dados do usuário.

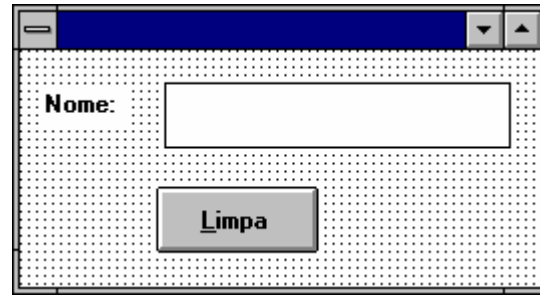
Propriedade	Descrição
Alignment	Alinhamento do texto dentro do text box.
Font	Tipo de letra utilizada.
Height	Altura do text box.
Left	Indica margem esquerda do text box em relação ao formulário.
MaxLegth	Tamanho máximo do texto.
Multiline	Indica se um text box tem mais de uma linha.
Name	Nome do controle a ser utilizado pelo programa.
PasswordChar	Caracter utilizado para esconder palavras secretas. Funciona apenas quando a propriedades Multiline tem valor <i>False</i> . Ex.: senhas.
ScrollBars	Indica a existência de barras de rolagem.
Text	Texto do text box.
Top	Indica do topo do text box em relação ao formulário.
Width	Largura da text box.

EVENTOS

**Change** Este evento acontece toda vez que o usuário altera o conteúdo de uma Text Box. Por exemplo, quando o usuário escreve a palavra "papel", o evento Change ocorre cinco (5) vezes.



## Exemplo de Utilização de Label e Text Box



1. Inicializar o Visual Basic.

### Criando um Label

2. Clicar duas vezes sobre a ferramenta Label na Caixa de Ferramentas.
3. Selecionar o label1 e chamar a Caixa de Propriedades.
4. Atualizar a propriedade *Caption* com *Nome:* e a propriedade *Name* com *LBLNome*.

### Criando um Text Box

5. Clicar duas vezes sobre a ferramenta Text Box na Caixa de Ferramentas.
6. Selecionar o text1 e chamar a Caixa de Propriedades.
7. Atualizar a propriedade *Name* com *TXBnome*.
8. Na propriedade *Text*, apagar o que estiver escrito.

### Criando um Command Button

9. Clicar duas vezes sobre a ferramenta Command Button na Caixa de Ferramentas.
10. Selecionar o Command1 e chamar a Caixa de Propriedades.
11. Atualizar a propriedade *Caption* com *Limpa* e a propriedade *Name* com *CMDLimpa*.

### Programando o evento Click do botão

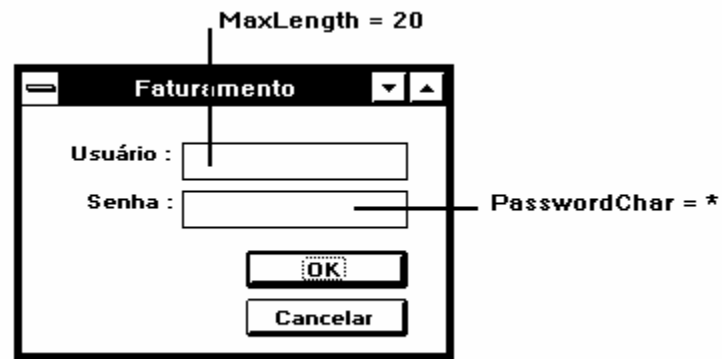
12. Clicar duas vezes sobre o controle Command1.
13. Na lista de eventos, selecionar o evento *Click*.
14. Escrever a seguinte linha de código :

```
TXBnome.text = ""
```

### Executando a Aplicação

15. Tecle F5 ou clique na opção *Run / Start* do menu.

### Propriedades que Restringem a Entrada de Dados



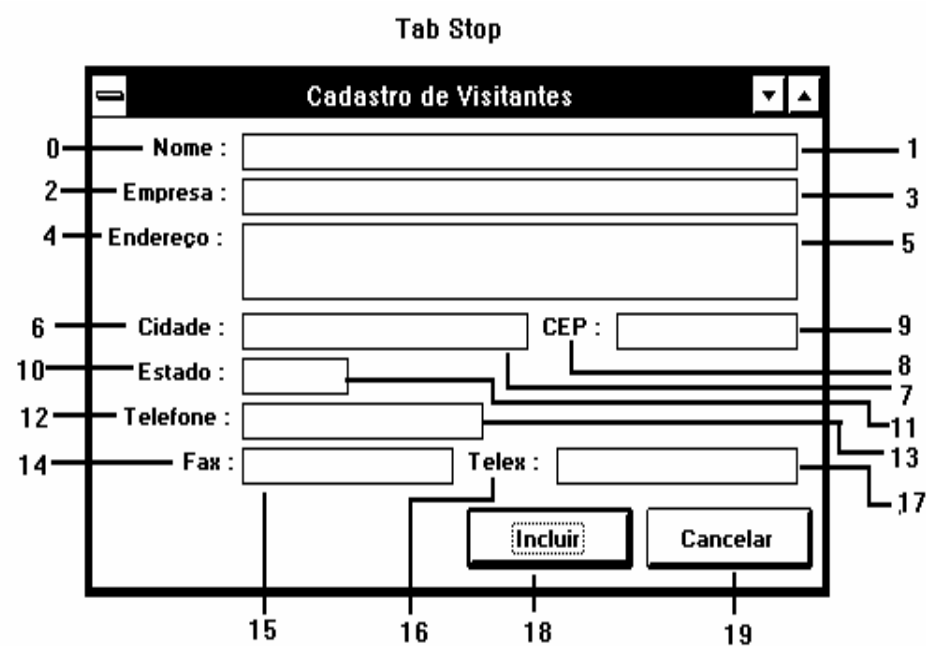
#### Maxlength de TextBoxes

Determina o número máximo de caracteres que uma caixa de texto pode conter. Se o usuário digitar mais que o permitido, o sistema emitirá um *beep*, e ignorará tudo que o usuário digitar que ultrapasse o limite.

#### PasswordChar de TextBoxes

Determina o caracter a ser mostrado na tela, independente do que o usuário estiver digitando. Normalmente é utilizado para que senhas e códigos secretos não apareçam na tela. Normalmente o caracter utilizado em aplicações Windows é o asterísco (\*).

### 5.2.1 Controlando a Tabulação



A ordem de tabulação entre os campos de um formulário é designada pela propriedade *TabIndex* dos controles do formulário. Esta propriedade vai sendo atribuída à medida que os controles vão sendo criados dentro do formulário. Portanto, a ordem de tabulação default é a ordem de criação dos controles. No entanto, para alterar esta ordem, basta alterar a propriedade *TabIndex* dos controles.

#### Tab Stop

Quando esta propriedade estiver com o valor *false*, o cursor não parará aqui quando o usuário estiver usando o Tab para se movimentar pelo formulário.

### 5.2.2 Designando Teclas de Acesso

O diagrama mostra uma janela intitulada "Cadastro de Visitantes". Ela possui duas caixas de texto. A primeira caixa, rotulada "Nome :", está na linha 0 da grade e tem o índice de tabulação 1. A segunda caixa, rotulada "Empresa :", está na linha 2 da grade e tem o índice de tabulação 3. As linhas da grade são indicadas por números 0, 2 e 3.

Assim como podemos usar o & ( E comercial) para designar teclas de acesso a menus e botões, podemos também utilizá-lo para acessar caixas de texto. Para isso, basta inserir um label na frente ( na ordem de tabulação ) de um TextBox, e utilizar o & para designar uma tecla de acesso para o label.

Como labels não recebem foco, o foco irá para o controle que tenha o TabIndex imediatamente maior, portanto o Text Box receberá o foco.

#### Exemplo - Utilizando Labels e Text Boxes

Suponha que você tivesse que montar a seguinte tela para o envio de mensagens :

O diagrama mostra uma janela intitulada "Inclusão de Mensagens". Ela possui duas caixas de texto. A primeira caixa, rotulada "Nome :", contém o texto "José da Silva". A segunda caixa, rotulada "Mensagem :", contém o texto "O resultado da análise laboratorial solicitada já está terminada. Segundo os testes realizados".

1. Inicialize o Visual Basic.
2. Com o formulário selecionado, pressione F4 para ter acesso à lista de propriedades.
3. Na propriedade *Caption*, escrever o título do formulário *Inclusão de Mensagens* e na propriedade *Name* escrever *FRMInclusão*.

#### Criando os Label Nome e Mensagem

4. Clique duas vezes sobre a ferramenta Label da Caixa de Ferramentas.
5. Com o label selecionado, pressione F4 para ter acesso à lista de propriedades.
6. Na propriedade *Caption*, escreva o título do label *Nome* e na propriedade *Name* escrever *LBLNome*.
7. Posicione o label de acordo com o desenho da tela acima.
8. Repita os passos de 4 até 7 para criar o label "Mensagem" ( *Name=LBLMensagem*).

#### Criando o Text Box Nome

9. Clique duas vezes sobre a ferramenta Text Box da Caixa de Ferramentas.
10. Com o Text Box selecionado, pressione F4 para ter acesso à lista de propriedades.

11. Na propriedade *Text*, apagar o que estiver escrito.
12. Na propriedade *Name*, escrever *TXBnome*.
13. Posicione o textbox de acordo com o desenho da tela acima.

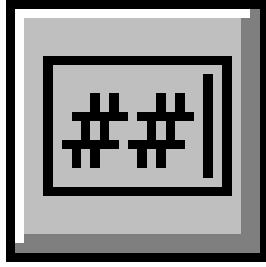
#### **Criando o Text Box Mensagem**

14. Clique duas vezes sobre a ferramenta Text Box da Caixa de Ferramentas.
15. Com o Text Box selecionado, pressione F4 para ter acesso à lista de propriedades.
16. Na propriedade *Text*, apagar o que estiver escrito.
17. Na propriedade *Name*, escrever *TXBmensagem*.
18. Na propriedade *ScrollBars*, selecionar *2-Vertical*.
19. Na propriedade *Multiline*, escrever *True*. Sem esta propriedade, o scroll bar vertical não funcionará.
20. Posicione o Text Box de acordo com o desenho da tela acima.

#### **Executando a Aplicação**

21. Pressione F5 ou selecione Start do menu Run.
22. Experimente dar Tab entre controles; selecionar, cortar e colar parte do texto, etc.

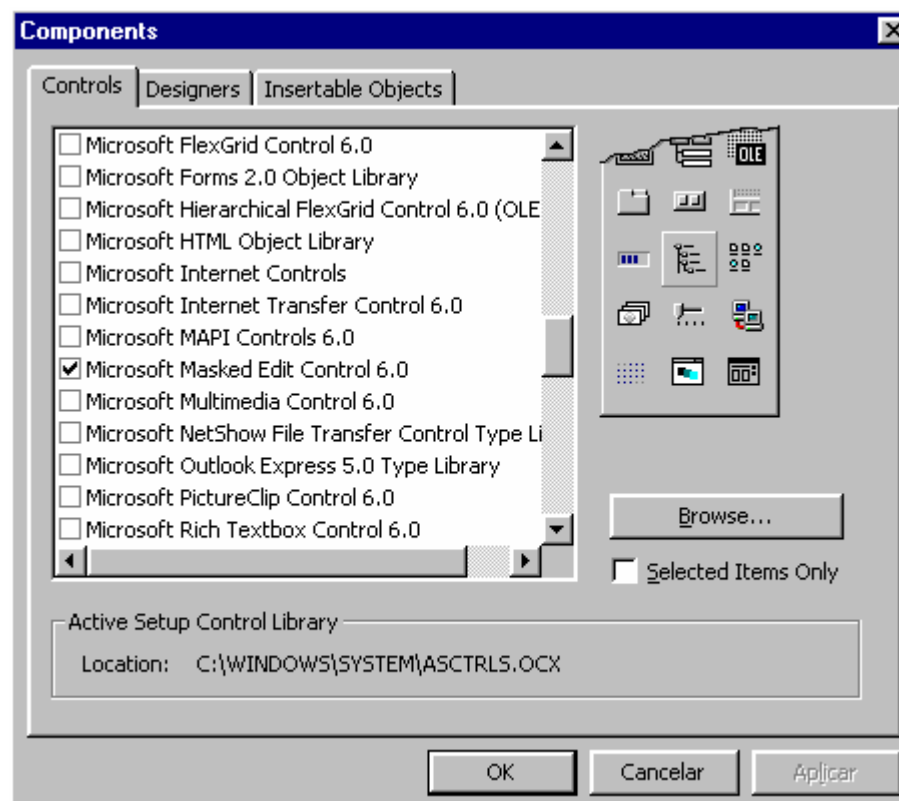
### 5.3 Masked Edit



É um controle que se assemelha a um Text box. No entanto, ele permite restringir a entrada de dados do usuário e também formatar a saída dos dados.

Essa ferramenta *Masked Edit* deverá ser inserida na caixa de ferramentas da seguinte forma:

Na opção *Components* do menu *Project* selecionar o item *Microsoft Masked Edit Control*.



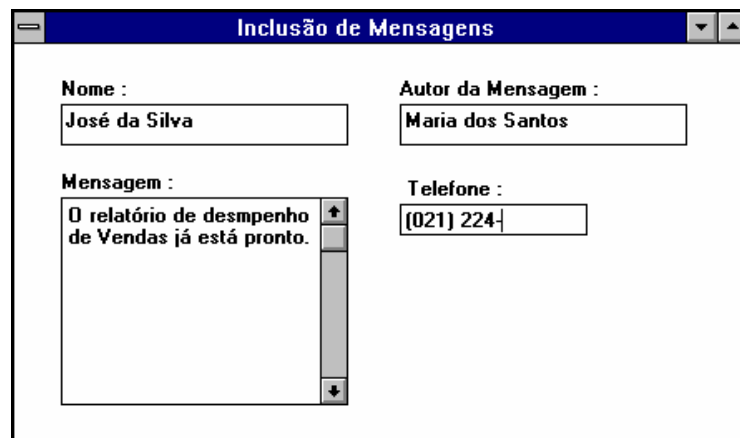
Propriedade	Descrição
ClipMode	Determina se os caracteres de máscara são incluídos ou não quando operações de Cut/Copy são executadas. Valores possíveis: 1 = inclui; 2 = não inclui.
Format	Formato com que números, data, hora e strings serão mostrados na Masked Edit.
Mask	Determina a máscara de entrada de dados.
MaxLength	Tamanho máximo do campo.
PromptChar	Caracter utilizado para pedir input ao usuário.
PromptInclude	Indica se os caracteres <i>PromptChar</i> serão incluídos na propriedade <i>Text</i> .

Caracteres de Máscara	Descrição
,	Separador de casas decimais.
.	Separador de milhar.
:	Separador de hora.
/	Separador de data.
&	Reserva lugar para caracter.
?	Reserva lugar para letras.
A	Reserva lugar para alfanumérico.
#	Reserva lugar para dígito.

Obs.: O separador de casas decimais e o separador de milhar se regulam pelo definido na configuração do Windows. Desta forma, se a configuração do Windows for formato brasileiro ( "," para casas decimais e "." para milhares), apesar de você colocar "." para reservar o lugar do separador decimal, o que aparecerá na tela em tempo de execução será ",".

## Exemplo - Utilizando Masked Edit

Continuando o exemplo mostrado anteriormente, queremos agora colocar o nome do autor da mensagem e seu telefone de contato. O campo de autor da mensagem só deverá conter letras e o telefone deverá ser colocado no formato (999) 999-9999. Para fazer isto usa-se *Masked Edits*.



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie os labels de "Autor da Mensagem" e "Telefone".

### Criando a Masked Edit Nome do Autor

3. Clique duas vezes sobre a ferramenta Masked Edit da Caixa de Ferramentas.
4. Com o Masked Edit selecionado pressione F4 para ter acesso à lista de propriedades.
5. Na propriedade *Name* escrever *MEBNomeAutor*.
6. Na propriedade *Mask* colocar ??????????????????????????????????????. Repare o que acontece com a propriedade *MaxLength*.

### Criando a Masked Edit Telefone

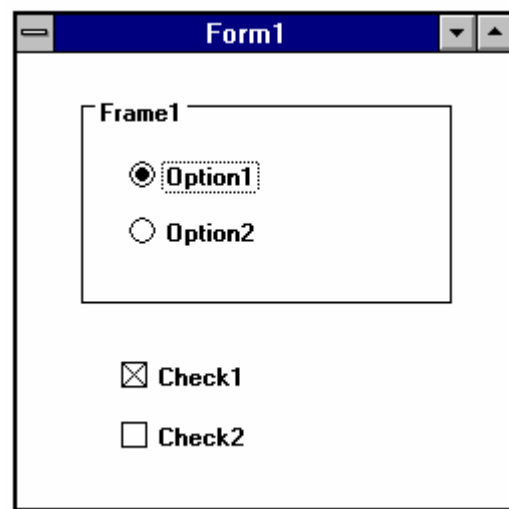
7. Clique duas vezes sobre a ferramenta Masked Edit da Caixa de Ferramentas.
8. Com o Masked Edit selecionado pressione F4 para ter acesso à lista de propriedades.
9. Na propriedade *Name* escrever *MEBTelefone*.
10. Na propriedade *Mask* colocar (###) ###-####..

### Executando a Aplicação

11. Pressione F5 ou selecione Start do menu Run.
12. Experimente digitar números no nome, letras no telefone e etc.



#### 5.4 Frames, Option Buttons e Check Boxes



Frames, check boxes e option buttons possibilitam mostrar ao usuário conjuntos de opções entre as quais ele pode escolher. O arranjo na tela dos frames e option buttons definem grupos independentes de opções. A escolha em um grupo não afeta a escolha em outro.

### 5.4.1 Frames

Frames permitem agrupar, gráfica e funcionalmente, um grupo de controles. Um exemplo clássico do uso de Frames é o agrupamento de Option Buttons mutuamente exclusivos.

Propriedades	Descrição
Caption	O título do frame.
Name	Nome do controle usado no código.
Visible	Indica se um frame e seus controles estão visíveis ou não.

Observação : Para colocar controles dentro de um frame, você tem que criar primeiro o frame; selecioná-lo e, então, criar os controles dentro dele. Ou então, caso os controles tenham sido criados anteriormente, deve-se recortá-los, selecionar o frame e, dentro deste, colá-los (Cut e Paste).

### 5.4.2 Check Boxes

Check Boxes são usados, normalmente, para permitir uma ou mais escolhas independentes, não exclusivas, em relação a outras já efetuadas.

Pro priedades	Descrição
Caption	Título da opção na tela.
Enabled	Habilita e desabilita o acesso do usuário.
Name	Nome do controle usado internamente.
Value	Indica se um check box está marcado ou não.
Visible	Mostra ou esconde o controle.

## EVENTOS

**Click** O evento click indica que o usuário fez uma seleção. Ao ocorrer este evento , o Check Box é marcado/desmarcado conforme o seu valor anterior.

### 5.4.3 Option Buttons

Option Buttons são normalmente utilizados para criar conjuntos de escolhas mutuamente exclusivas dentro de um contexto, que pode ser a tela, um frame ou um frame dentro de outro.

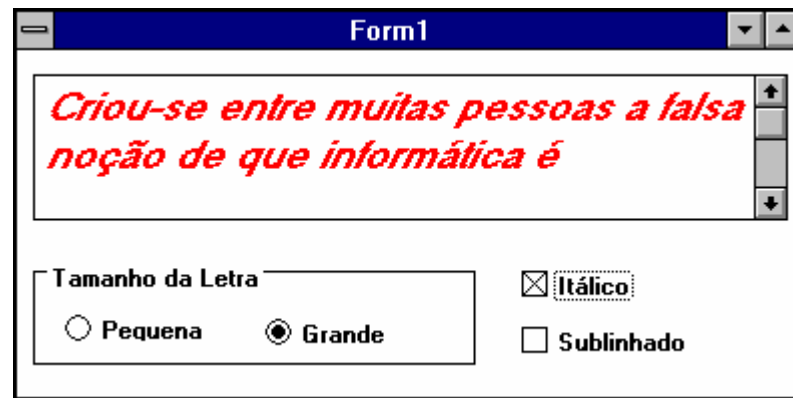
Propriedades	Descrição
Caption	Valor da opção na tela.
Name	Nome interno do controle.
Enabled	Define se o controle está ativo ou não.
Value	Indica se um option button está marcado ou não.
Visible	Define se o controle está aparente ou não.

#### EVENTOS

**Click** Quando o usuário clica sobre o option button, uma série de acontecimentos ocorrem : o option button selecionado é marcado, os outros Option Buttons do contexto são desmarcados e a propriedade *value*, para cada option button do contexto é atualizada com a nova seleção.

## Exemplo - Utilização de Frames, Check Boxes e Option Buttons

Suponha que você queira criar uma aplicação onde um texto escrito em um Text Box possa ser formatado por você :



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie o Text Box (Name = *TXBExemplo*).

### Criando os Option Buttons

3. Clique duas vezes sobre a ferramenta Frame da Caixa de Ferramentas.
4. Com o Frame selecionado pressione F4 para ter acesso à lista de propriedades.
5. Nomeie este Frame "FRATamanho".
6. Com o frame selecionado, clique duas vezes sobre a ferramenta Option Button da Caixa de Ferramentas.
7. Com o Option Button selecionado pressione F4 para ter acesso à lista de propriedades.
8. Nomeie este Option Button *OPTPequena*.
9. Atribua à propriedade *Caption* o valor *Pequeno*.
10. Repita os passos 6 e 7 e nomeie o segundo botão *OPTGrande*.
11. Atribua à propriedade *Caption* o valor *Grande*.

### Codificando os Option Buttons

12. Clique duas vezes sobre OPTPequena, uma janela de código deve aparecer.
13. Escreva o seguinte comando :  

```
TXBExemplo.FontSize = 12
```
14. Clique duas vezes sobre OPTGrande, uma janela de código deve aparecer.
15. Escreva o seguinte comando :  

```
TXBExemplo.FontSize = 18
```

### Criando os Check Boxes

16. Clique duas vezes sobre a ferramenta Check Box da Caixa de Ferramentas.
17. Com o Check Box selecionado pressione F4 para ter acesso à lista de propriedades.
18. Nomeie este Check Box *CKBItálico*.
19. Atribua à propriedade *Caption* o valor *Itálico*.
20. Repita os passos 16, 17 e 18 e nomeie o segundo Check Box *CKBSublinhado*.
21. Atribua à propriedade *Caption* o valor *Sublinhado*.

### Codificando os Check Boxes

22. Clique duas vezes sobre CKBItálico, uma janela de código deve aparecer.
23. Escreva os seguintes comandos :  

```
If CKBItálico.Value = 1 Then
    Text1.FontItalic = True
Else
```

```
Text1.FontItalic = False  
End If
```

24. Clique duas vezes sobre CKBSublinhado, uma janela de código deve aparecer.

25. Escreva os seguintes comandos :

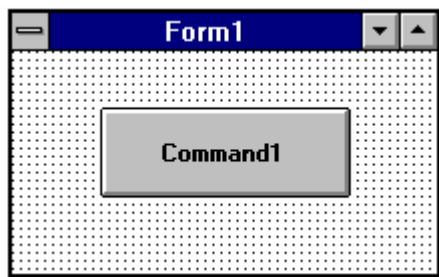
```
If CKBSublinhado.Value = 1 Then  
    Text1.FontUnderline = True  
Else  
    Text1.FontUnderline = False  
End If
```

### **Executando a Aplicação**

26. Pressione F5 ou selecione Start do menu Run.

27. Experimente apertar os botões e os Check boxes para ver o que acontece.

5.5 Command Buttons Normais e Tridimensionais

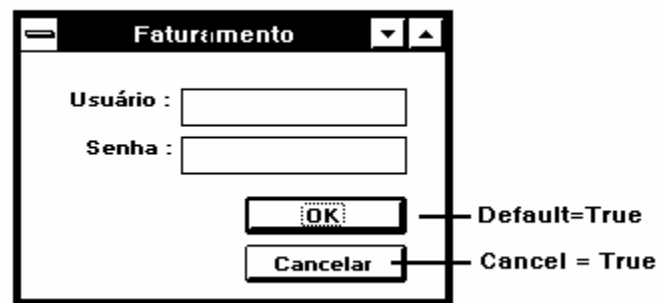


Propriedade	Descrição
Cancel	Ativa o botão quando o <i>Esc</i> for pressinado.
Caption	Valor do botão que aparece na tela.
Default	Ativa botão quando o <i>Enter</i> for pressionado.
Enabled	Permite, ou não, o acesso ao botão.
Height	Altura do botão.
Left	Posição da borda esquerda em relação ao formulário.
Name	Nome do controle usado internamente.
Top	Posição da borda superior em relação ao formulário.
Width	Largura do botão.

EVENTOS

**Click** Normalmente, significa que o usuário deseja que alguma ação seja tomada. Neste caso, você deverá escrever o código correspondente.

## Propriedades Default e Cancel



### Default de Command Buttons

Determina que, se o usuário pressionar a tecla de Enter, de qualquer ponto da tela, o evento Click do botão Default ocorrerá. Existe no máximo um botão default por tela.

### Cancel de Command Buttons

Determina que se o usuário pressionar a tecla de Esc, de qualquer ponto da tela, o evento Click do botão, cuja propriedade Cancel estiver com valor True, ocorrerá. Existe no máximo um botão cancel por tela.

Obs.: No evento *Click* de botões cujas propriedades *Default* ou *Cancel* tenham valor *True*, deve existir um comando transferindo o foco para o próprio botão (`nome_botão.setfocus`), porque se o click do botão for ativado porque o usuário pressionou ENTER ou CANCEL, o *LostFocus* do controle onde o foco estava anteriormente não ocorre.

## Exemplo - Utilização de Command Buttons

Suponha que você queira criar uma aplicação onde a hora será informada sempre que você desejar:



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie o Text Box.
3. Nomeie o Text Box de *TXBHora*.
4. Com TXBHora selecionado, altere as propriedades *FontSize*, *FontItalic* e *FontName*.

### Criando os Command Buttons

5. Clique duas vezes sobre a ferramenta Command Button da Caixa de Ferramentas.
6. Com o Command Button selecionado pressione F4 para ter acesso à lista de propriedades.
7. Nomeie este Command Button *CMDInforma*.
8. Atribua o valor *Informa Hora* à propriedade *Caption*.
9. Repetir os passos 5 e 6 e nomeie este Command Button *CMDLimpa*.
10. Atribua o valor *Limpa* à propriedade *Caption*.

### Codificando os Command Buttons

11. Clicar duas vezes sobre CMDInforma, uma janela de código irá aparecer.
12. Digitar a seguinte linha de código:  

```
TXBHora.Text = Format(Now, "hh:mm:ss")
```
13. Clicar duas vezes sobre CMDLimpa. Uma janela de código irá aparecer.
14. Digitar a seguinte linha de código:  

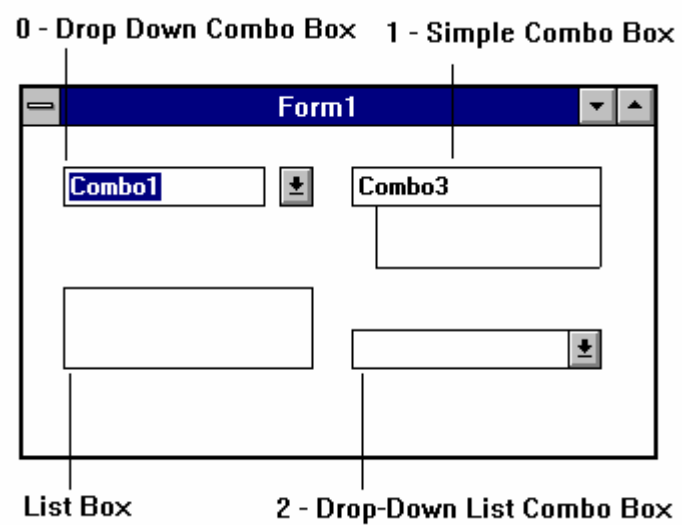
```
TXBHora.Text = ""
```

### Executando a Aplicação

13. Pressione F5 ou selecione Start do menu Run.
14. Experimente apertar os botões.



## 5.6 Combo Box e List Box



Existem quatro tipos de controles de listas, cada um com uma função ligeiramente diferente. Ambas as listas que caem ( Drop Down Combo Box e Drop Down List Combo Box ) foram projetadas para economizar o espaço de tela utilizado pelo controle. Usuários somente podem adicionar itens a uma lista do tipo *Drop-Down Combo Box* ou *Simple Combo Box*, pois os outros tipos permitem apenas escolha entre os itens existentes.

Todas os List Boxes e combo boxes apresentarão automaticamente uma barra de rolagem, caso o tamanho da lista ou combo ultrapasse a sua própria altura ( definida pela propriedade *Height*).

Nas páginas seguintes, serão apresentados Combo Boxes e List Boxes mais detalhadamente.

5.6.1 List Box

List Boxes mostram um conjunto de ítems entre os quais o usuário pode escolher um ou mais. Através de código, você pode adicionar ou remover ítems do list box.



Propriedade	Descrição
Columns	Permite a exibição de múltiplas colunas em uma única lista.
List	Vetor que contém a lista dos ítems da ListBox.
Index	Índice do item ,da lista, selecionado.
MultiSelect	0 - Permite selecionar apenas um item por vez. 1 - Permite selecionar mais de um item da lista. 2 - Permite seleção do tipo File Manager do Windows, usando teclas CTRL e SHIFT.
Name	Nome interno do controle.
Sorted	Indica se a lista estará em ordem alfabética ou não.
Text	Retorna o item selecionado da lista.

O vetor booleano SELECTED() indica quais os ítems selecionados da lista, caso você tenha escolhido uma lista de seleção múltipla.

EVENTOS

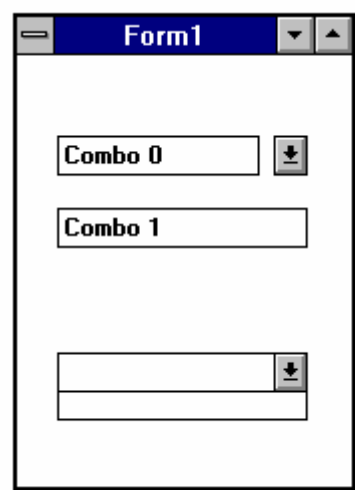
**Click**                      Seleciona um item da lista.

**Double-Click**            Combina dois eventos : a seleção do item da lista e a inicialização de um processo associado àquele item.

MÉTODOS

Método	Descrição	Sintaxe
AddItem	Adiciona um item à lista. Normalmente executado no Form Load.	list1.additem "João da Silva"
RemoveItem	Remove um item específico da lista.	list1.removeitem (ListCount)
Clear	Remove todos os itens da lista.	List1.Clear

5.6.2 Combo Box



Propriedade	Descrição
Name	Nome do controle internamente.
Style	Tipo da combo : 0 - Drop-Down Combo 1 - Simple Combo 2 - Drop-Down List
Text	Texto selecionado.
Height	Altura do Combo

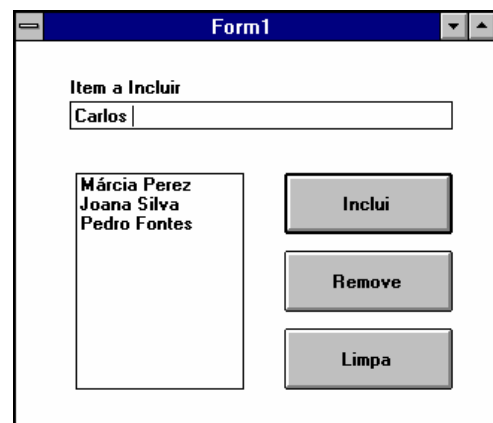
EVENTOS

**Change** O evento Change indica que o conteúdo do controle foi alterado. No caso de combo boxes, o evento change ocorre toda vez que, a parte editavel do combo á alterada.

Obs.: Os métodos *AddItem*, *RemoveItem* e *Clear*, demonstrados no item anterior sobre listas, também são válidos para Combo Boxes.

## Exemplo - Utilização de List Box

Suponha que você queira criar uma aplicação onde você possa incluir nomes em uma lista, removê-los e limpar a lista por completo :



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie um Text Box.
3. Nomeie o Text Box de *TXBNome*.

### Criando a Lista

4. Clique duas vezes sobre a ferramenta List Box da Caixa de Ferramentas.
5. Com o List Box selecionado pressione F4 para ter acesso à lista de propriedades.
6. Nomeie esta lista *LSBNomes*.

### Criando os Command Buttons

7. Crie três botões e nomeie-os: *CMDInclui*, *CMDRemove* e *CMDLimpa*.
8. Com *CMDInclui* selecionado pressione F4 para ter acesso à lista de propriedades.
9. Colocar o valor TRUE na propriedade *Default*.

### Codificando os Command Buttons

10. Clicar duas vezes sobre *CMDInclui*, uma janela de código irá aparecer. Certifique-se que o evento é *Click*.
11. Digitar as seguintes linhas de código:

```
If Trim(TXBNome.Text) <> "" Then
    LSBNomes.AddItem TXBNome.Text
End If
TXBNome.Text = ""
TXBNome.SetFocus
```

12. Clicar duas vezes sobre *CMDRemove*. Uma janela de código irá aparecer. Certifique-se que o evento é *Click*.
13. Digitar as seguintes linhas de código:

```
If LSBNomes.ListIndex <> -1 Then
    LSBNomes.RemoveItem LSBNomes.ListIndex
End If
TXBNome.SetFocus
```

14. Clicar duas vezes sobre *CMDLimpa*. Uma janela de código irá aparecer. Certifique-se que o evento é *Click*.

15. Digitar as seguintes linhas de código:

```
LSBNomes.Clear
TXBNome.SetFocus
```

### **Executando a Aplicação**

16. Pressione F5 ou selecione Start do menu Run.
17. Experimente adicionar e retirar nomes da lista.

### **Exemplo - Utilização de Combo Box**

Suponha que você queira criar uma aplicação que associe nomes de funcionários à cargos e à departamentos de uma empresa:

OBS: Os cargos e departamentos existentes já estão definidos e não deverão ser alterados por sua aplicação.

ATENÇÃO: Em uma aplicação real, os Combo Box de Cargo e Departamento deveriam ser preenchidos a partir de uma arquivo que contivessem estas informações.

1. Inicialize o Visual Basic.

#### **Criando o Label e o Combo Box para o Nome**

2. Como demonstrado anteriormente, crie um Label com *Caption = Nome* e *Name=LBLNome*.
3. Crie um Combo Box com *Style = 0 - Dropdownm Combo* e *Name=CMBNome*.
4. Inicialize sua propriedade *Text* com branco.

#### **Criando o Label e o Combo Box para o Cargo**

5. Crie um Label com *Caption = Cargo* e *Name=LBLCargo*.
6. Crie um Combo Box com *Style = 2 - Dropdownm List* e *Name=CMBCargo*.

#### **Criando o Label e o Combo Box para o Departamento**

7. Crie um Label com *Caption= Departamento* e *Name=LBLDepartamento*.
8. Crie um Combo Box com *Style = 2 - Dropdownm List* e *Name=CMBDepartamento*.

**Programando o Load do form - (Carregando os dados de Cargos e Departamentos)**

9. Clicar duas vezes sobre o form, uma janela de código irá aparecer. Certifique-se que o evento é *Load*.

10. Escreva as seguintes linhas de código:

```
CMBCargo.AddItem "Estagiário"  
CMBCargo.AddItem "Analista de Sistemas"  
CMBCargo.AddItem "Gerente de Desenvolvimento"  
CMBCargo.AddItem "Analista de Suprimentos"  
CMBCargo.AddItem "Gerente de Marketing"
```

```
CMBDepartamento.AddItem "Marketing"  
CMBDepartamento.AddItem "Sistemas"  
CMBDepartamento.AddItem "Suprimentos"  
CMBDepartamento.AddItem "Recursos Humanos"
```

**Programando o LostFocus do CMBNome (Inclusão de novos nomes no Combo)**

9. Clicar duas vezes sobre o combo, uma janela de código irá aparecer.

10. Selecione o evento LostFocus.

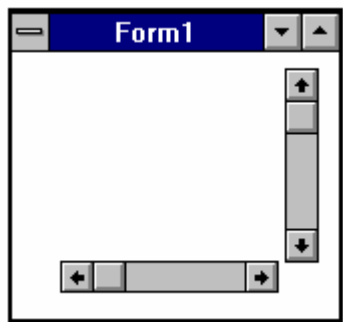
11. Escreva a seguinte linha de código:

```
CMBNome.AddItem CMBNome.text
```

**Executando a Aplicação**

12. Pressione F5 ou selecione Start do menu Run.

5.7 Scroll Bars



Scroll Bars são utilizados quando se deseja demonstrar a posição corrente em uma escala.

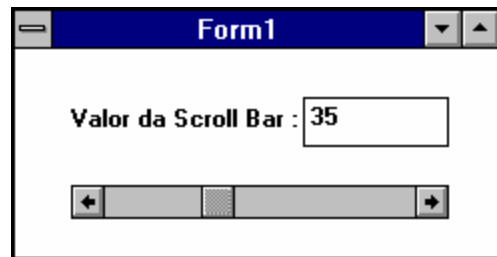
Propriedade	Descrição
LargeChange	Mudança de valor que ocorrerá quando o usuário clicar sobre o marcador de posição da barra.
Max	Valor máximo da barra.
Min	Valor mínimo da barra.
Name	Nome interno do controle.
SmallChange	Mudança de valor que ocorrerá quando o usuário clicar sobre as setas da barra.
Value	Valor correspondente à posição da barra.

EVENTOS

**Change** O evento Change indica que o conteúdo do controle foi alterado. No caso de scroll bars , ocorre quando o usuário rola a barra.

### Exemplo - Utilização de Scroll Bar

Para exemplificar o uso de um scrollbar, usaremos um text box. À medida que você mexer no Scrollbar, o valor do Text Box também alterará.



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie o Text Box.
3. Nomeie o Text Box de *TXBValor*.

#### Criando o Scroll Bar

4. Clique duas vezes sobre a ferramenta Horizontal Scroll Bar da Caixa de Ferramentas.
5. Com o Scroll Bar selecionado pressione F4 para ter acesso à lista de propriedades.
6. Nomeie este Scroll Bar *HSBValor*.
7. Coloque o valor 100 na propriedade *Max*.
8. Coloque o valor 0 na propriedade *Min*.
9. Coloque o valor 1 na propriedade *SmallChange*.
10. Coloque o valor 5 na propriedade *LargeChange*.

#### Codificando a Scroll Bar

11. Clique duas vezes sobre o scroll bar. A janela de código do evento Change aparecerá.
12. Digite a linha de código :

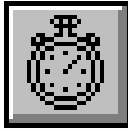
```
TXBValor.Text = HSBValor.Value
```

#### Executando a Aplicação

13. Pressione F5 ou selecione Start do menu Run.
14. Movimente-se pelo scrollbar.
15. Experimente alterar o valor das propriedades *Large Change* e *SmallChange*.

## 5.8 Timer





Timers são utilizados para ativar eventos, periodicamente, em um prazo definido por você. Um exemplo seria sair de uma tela qualquer, quando o usuário deixasse o computador parado por um intervalo de tempo.

O controle timer somente é visível em *design time*.

Propriedade	Descrição
Enabled	Indica se o timer está ativo.
Interval	Intervalo de tempo que o controle será ativado.
Name	Nome interno do controle.

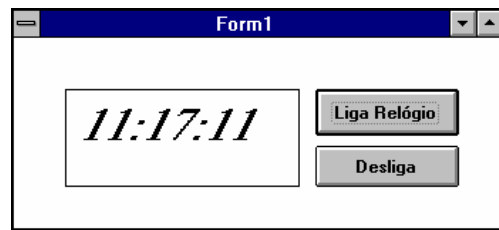
Obs.: Para um timer funcionar, a propriedade *Enabled* deve ter o valor *True* e a propriedade *Interval* deve ser diferente de 0.

## EVENTOS

**Timer** Este evento ocorre a cada intervalo de tempo especificado na propriedade *Interval* do controle.

## Exemplo - Utilizando um Timer

Relembrando a aplicação do relógio, agora vamos utilizar um timer para que a hora seja atualizada a cada segundo.



1. Inicialize o Visual Basic.
2. Como demonstrado anteriormente, crie o Text Box.
3. Nomeie o Text Box de *TXBHora*.

### Criando os Command Buttons

4. Crie dois botões, nomeie-os *CMDLiga* e *CMDDesliga* e atribua, às suas propriedades *Caption*, os valores *Liga Relógio* e *Desliga*, respectivamente.

### Criando o Timer

5. Clique duas vezes sobre a ferramenta Timer da Caixa de Ferramentas.
6. Com o Timer selecionado pressione F4 para ter acesso à lista de propriedades.
7. Nomeie este timer *TIMsegundo*.
8. Coloque o valor 1000 na propriedade *Interval*, e False em *Enabled*.

### Codificando os Command Buttons

9. Clicar duas vezes sobre *CMDLiga*, uma janela de código irá aparecer. Certifique-se que o evento é *Click*.
10. Digitar a seguinte linha de código:

```
TIMsegundo.enabled = true
```

11. Clicar duas vezes sobre *CMDDesliga*. Uma janela de código irá aparecer. Certifique-se que o evento é *Click*.
12. Digitar a seguinte linha de código:

```
TIMsegundo.enabled = False
```

### Codificando o Timer

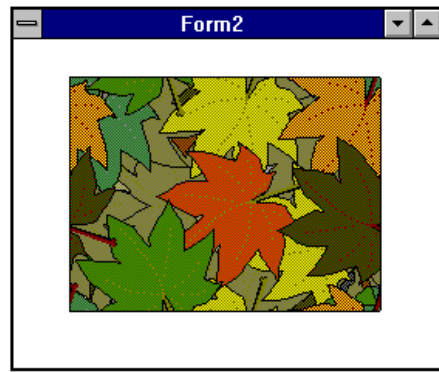
13. Clicar duas vezes sobre *TIMSegundo*, uma janela de código irá aparecer. Certifique-se que o evento é *Timer*.
14. Digitar a seguinte linha de código:

```
TXBHora.text = Format (Now, "hh:mm:ss")
```

### Executando a Aplicação

15. Pressione F5 ou selecione Start do menu Run.
16. Ligue e desligue o relógio.
17. Pare a execução do programa e experimente alterar o valor do *Interval* do timer.

## 5.9 Picture Box



Picture Boxes são basicamente utilizados para mostrar gráficos ou figuras. No entanto, picture boxes também podem conter controles. Desta forma, além de agrupar controles afins, você pode ter várias versões de uma parte de uma tela, em pictures diferentes, fazendo que apenas a versão (picture) que lhe convier no momento esteja aparente.

### EVENTOS

**Paint** Acontece toda vez que um formulário ou controle é movimentado, mostrando parte de um picture box que inicialmente estava escondido. Este evento entretanto, só acontece quando a propriedade *AutoRedraw* do Picture Box está com valor *False*; caso contrário, a repintura do Picture Box é feita automaticamente, portanto o evento Paint não acontece.

---

**Cuidado !!** Em alguns casos não é aconselhável deixar a propriedade *AutoRedraw* de um picture Box com valor *True*; isto pode consumir muita memória do seu micro-computador.

---

Propriedade	Descrição
AutoRedraw	Indica se a pintura da Picture Box será feita automaticamente ou não
AutoSize	Se True, ajusta o tamanho da Picture Box de acordo o tamanho da figura.
Name	Nome interno do controle.
Picture	Exibe um Dialog Box para definir uma figura, no formato BMP, WMF ou ICO, com o qual o Picture Box deve ser preenchido.

CARREGANDO E DESCARREGANDO FIGURAS EM RUNTIME

Para carregar ou descarregar figuras, em tempo de execução, utiliza-se a função *LoadPicture()*.

Exemplo - Utilizando um Picture Box

- 1. Inicialize o Visual Basic.
- Criando dois Command Button**
- 2. Clique duas vezes sobre a ferramenta Command Button da Caixa de Ferramentas.
- 3. Com o objeto selecionado pressione F4 para ter acesso à lista de propriedades.
- 4. Fazer *Name=CMDCarregar* e *Caption=Carregar*.
- 5. Repetir os passos 2 e 3 para criar outro Command Button, com as seguintes propriedades: *Name=CMDLimpar* e *Caption=Limpar*.

Criando o Picture Box

- 6. Clique duas vezes sobre a ferramenta Picture Box da Caixa de Ferramentas.
- 7. Com o objeto selecionado pressione F4 para ter acesso à lista de propriedades.
- 8. Nomeie esta picture box de PICFigura.

Codificando o Command Button CMDCarregar

- 9. Escrever o seguinte código no evento CLICK  
`Picture1.picture = LoadPicture ("c:\vb\icons\arrows\point12.ico")`

Codificando o Command Button CMDLimpar

- 10. Escrever o seguinte código no evento CLICK  
`Picture1.picture = LoadPicture ()`

Executando a Aplicação

- 15. Pressione F5 ou selecione Start do menu Run.

5.10 Grid

Nome	Telefone
João	5674532
Maria	2665478

Para inserir a feramenta *GRID* na caixa de ferramentas devemos seleccionar o item *Microsoft FlexGrid Control* da opção *Components* do menu *Project*

Este controle trabalha com um conjunto de propriedades através das quais é possível seleccionar linhas e colunas ( como células em uma planilha ), escrever dentro delas e copiar dados para elas. Um grid possui um número inicial de linhas e colunas que não pode ser menor que o seu número de linhas e colunas fixas. O dados que preenchem um grid são representados por um texto no qual separa-se colunas por *Tabs* e linhas por *CarriageReturn*.

Propriedade	Descrição
Rows, Cols	Indica o numero de linhas e colunas do grid.
FixedRows, FixedCols	Número de linhas e colunas fixas do grid.

Principais Métodos

**AddItem** - Adiciona uma linha ao grid na posição especificada e a preenche.

Sintaxe : nome-grid.additem conteúdo, índice

**RemoveItem** - Adiciona uma linha e seu conteúdo do grid. A linha a ser retirada é a especificada pelo índice.

Sintaxe : nome-grid.removeitem índice

Obs.: Índice em ambos os casos significa o número da linha a ser incluída ou retirada.

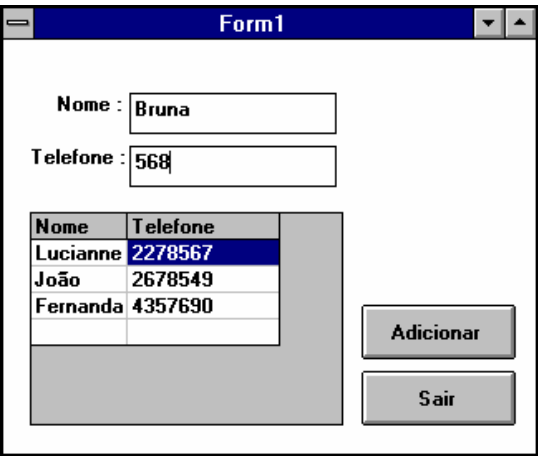
Exemplos :

```
Sub Command1_Click ()
    'Determina a célula inicial e a preenche
    Grid1.col =0
    Grid1.Row =0
    Grid1.Text = "Nome"
End Sub

Sub Command2_Click ()
    'Seleciona um grupo de células e as preenche
    Grid1.SelStartCol = 1
    Grid1.SelEndCol = 3
    Grid1.SelStartRow = 1
    Grid1.SelEndRow = 3
    tab = chr(9)
    carriage_return = chr(13)
    texto = "L1C1" + tab + "L1C2" + carriage_return
    texto = texto + "L2C1"+ tab + "L2C2"
    Grid1.clip = texto
End Sub
```

Exemplo - Utilização de Grid

Para exemplificar a utilização de um grid, construiremos uma pequena aplicação onde se possa cadastral nomes e telefones.



- 1. Inicialize o Visual Basic.
- 2. Como demonstrado anteriormente, crie dois Text Boxes.
- 3. Nomeie os Text Boxes de *TXBNome* e *TXBTelefone*.
- 4. Esvazie o conteúdo de suas propriedades *Text*.

Criando os Command Buttons

- 5. Crie dois botões de comando e nomeie-os *CMDAdicionar*, *CMDsair*, e atribua à sua propriedade *Caption*, os valores *Adicionar* e *Sair*, respectivamente.

Criando o Grid

- 6. Clique duas vezes sobre a ferramenta *Grid* da Caixa de Ferramentas.
- 7. Com o *Grid* selecionado pressione F4 para ter acesso à lista de propriedades.
- 8. Nomeie este grid *GRDAgenda*.
- 9. Coloque o valor 0 na propriedade *FixedCols*, e 1 em *FixedRows*.
- 10. Coloque o valor 2 na propriedade *Cols*, e 2 em *Rows*.

Codificando o Load do Formulário

- 11. Clique duas vezes sobre o formulário, a janela de código deve aparecer. Verificar se o evento selecionado é o *Load*.
- 12. Escrever as seguintes linhas de código :

```
GRDAgenda.Row = 0
GRDAgenda.Col = 0
GRDAgenda.Text = "Nome"
GRDAgenda.Col = 1
GRDAgenda.Text = "Telefone"
GRDAgenda.ColWidth(0) = 2500
GRDAgenda.ColWidth(1) = 1500
GRDAgenda.Width = 4300
```

Codificando os Command Buttons

- 13. Clicar duas vezes sobre *CMDAdiciona*, uma janela de código irá aparecer. Certifique-se que o evento é *Click*.
- 14. Digitar as seguintes linhas de código:

```
If Trim(TXBNome.Text) <> "" And Trim(TXBTelefone.Text) <> "" Then
```

```
GRDAgenda.AddItem TXBNome.Text + Chr(9) + TXBTelefone.Text,  
GRDAgenda.Rows - 1  
End If
```

15. Clicar duas vezes sobre CMDsair. Uma janela de código irá aparecer. Certifique-se que o evento é *Click*.

16. Digitar a seguinte linha de código:  
End

### Executando a Aplicação

17. Pressione F5 ou selecione Start do menu Run.

## 5.11 Foco

Em Windows, apenas um controle, formulário ou janela pode ter o foco de cada vez. Ter o foco significa que qualquer ação do usuário recai sobre aquele elemento, seja ele um controle ou uma janela. O foco pode ser transferido pelo usuário ou pela aplicação.

O Visual Basic possui alguns eventos e métodos para o tratamento de foco. A seguir detalharemos cada um deles.

### EVENTOS

**GotFocus** Acontece sempre que um controle recebe o foco, ou porque o usuário pressionou Tab, ou porque ele clicou sobre o controle, ou através de aplicação. Note que o Formulário somente recebe o evento GotFocus quando não há nenhum controle visível dentro dele.

**Lost Focus** Acontece em um controle sempre que um outro controle recebe o foco, ou porque o usuário pressionou Tab, ou porque ele clicou sobre o outro controle, ou através de aplicação.

### MÉTODOS

**SetFocus** Este método, aplicado a qualquer controle, transforma aquele controle no controle ativo. O *SetFocus* em um controle, provoca o *LostFocus* do controle que possuía o foco anteriormente e o *GotFocus* do controle ao qual foi aplicado o método.

Sintaxe : controle.*Setfocus*

---

Cuidado !!!! Não utilize nunca o método *SetFocus* dentro de um evento *LostFocus*.  
Dependendo da lógica de programação, isto pode causar loop infinito !!

---

6.0 Tipos de Dados

Objetivo do módulo

Introduzir os tipos de dados do Visual Basic e suas regras de escopo e durabilidade. Este é o primeiro, de um conjunto de capítulos, que ensina a programação em Visual Basic.

6.1 Tipos de Dados de Variáveis

Tipo		Descrição
Integer	%	Inteiro de 2 bytes
Long	&	Inteiro de 4 bytes
Single	!	Num. Ponto Flutuante de 4 bytes
Double	#	Num. Ponto Flutuante de 8 bytes
Currency	@	Num. Ponto Decimal Fixo de 8 bytes
String	\$	String de caracteres
Variant		Data/Hora, string, num de ponto Flutuante

Obs.: Operações que utilizam o tipo de dado *Currency* são mais rápidas e exatas do que as que utilizam o tipo *Single* e *Double*.

6.2 Declaração de Variáveis

Você pode declarar um variável de duas maneiras : usando o comando *Dim* ou então, uma das duas palavras reservadas - *Global* ou *Static*. A declaração de variáveis no Visual Basic é extremamente importante, pois qualquer variável não declarada é considerada como sendo do tipo *Variant*.

Existem duas sintaxes de declaração explícita de variáveis :

Usando a palavra reservada AS	Usando caracter de tipo
Dim I as Integer	Dim I%
Dim Total as Double	Dim Total#
Dim Nome as Sring	Dim Nome\$
Dim Valor as Currency	Dim Valor@

Para fazer declaração de variáveis compostas, usar :

```
Dim I as Integer, Total as Double
```

**Cuidado !!!** Em declarações de variáveis compostas, especificar o tipo para cada uma delas. No exemplo abaixo, apenas a última variável será do tipo Integer, as outras serão do tipo Variant.

```
Dim I, J, K as Integer
```

Declaração Obrigatória de Variáveis

Para que a declaração de variáveis no seu projeto seja obrigatória, colocar *Yes* no item *Require Variable Declaration* nas opções de configuração do ambiente.



### Strings de Tamanho Fixo e de Tamanho Variável

Quando você não souber o tamanho de uma string, você poderá declará-la com tamanho variável. Caso contrário, você deverá declará-lo. Uma string de tamanho fixo jamais terá valor nulo (NULL).

Ex :

```
Dim palavra as string
Dim palavra2 as string * 50
```

### Inicialização de Variáveis

O Visual Basic, automaticamente, inicializa todas as variáveis numéricas com zero (0) e todas as strings, de tamanho fixo, com brancos.

### Nomenclatura de Variáveis

1. Nomes de variáveis podem ter, no máximo, 40 caracteres.
2. Nomes podem conter letras, números e *underscores* (\_).
3. O primeiro caracter do nome deve ser uma letra.
4. Palavras reservadas do Visual Basic não poderão ser utilizadas.

## 6.3 Tipo de Dado Variant

*Variant* é o tipo de dado *default* do Visual Basic. Uma variável do tipo *variant* pode conter qualquer tipo de dado : números, letras ou datas. Não é necessário fazer qualquer tipo de conversão para atribuir valores destes tipos a um *variant*; o Visual Basic se encarrega de fazer a conversão automaticamente.

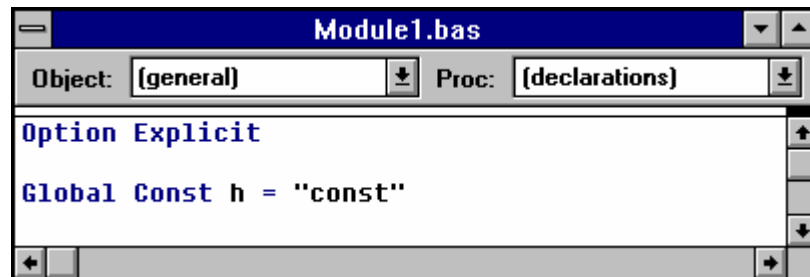
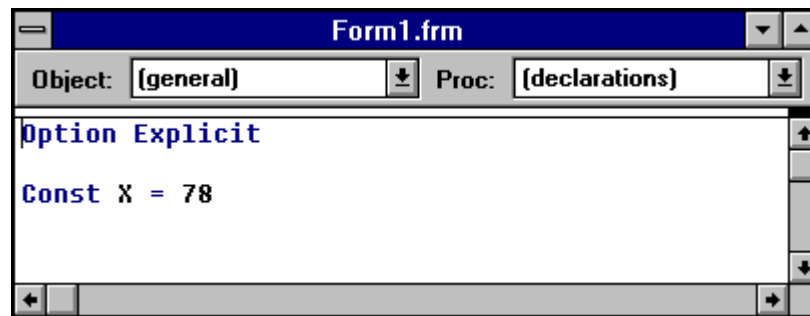
Para saber o tipo de dado que está dentro de um *variant*, você pode utilizar as seguintes funções booleanas do VB : `IsNumeric` e `IsDate`.

Para somar variáveis do tipo *variant*, as mesmas devem conter valores numéricos. Para concatenar variáveis do tipo *variant* utilizar o símbolo **&** para evitar ambigüidade.

```
Sub Command1_Click ()
Dim a
Dim b
  a = 3
  b = 5
  Print (a & b)    'resulta em 35
  Print (a + b)    'resulta em 8
End Sub
```

*Variant* tem o valor *Empty* até que algum valor seja atribuído a ela. *Empty* é um valor especial diferente de Null, brancos ou zero. Para verificar se uma variável está Empty, utilizara função `IsEmpty` do Visual Basic.

## 6.4 Constantes

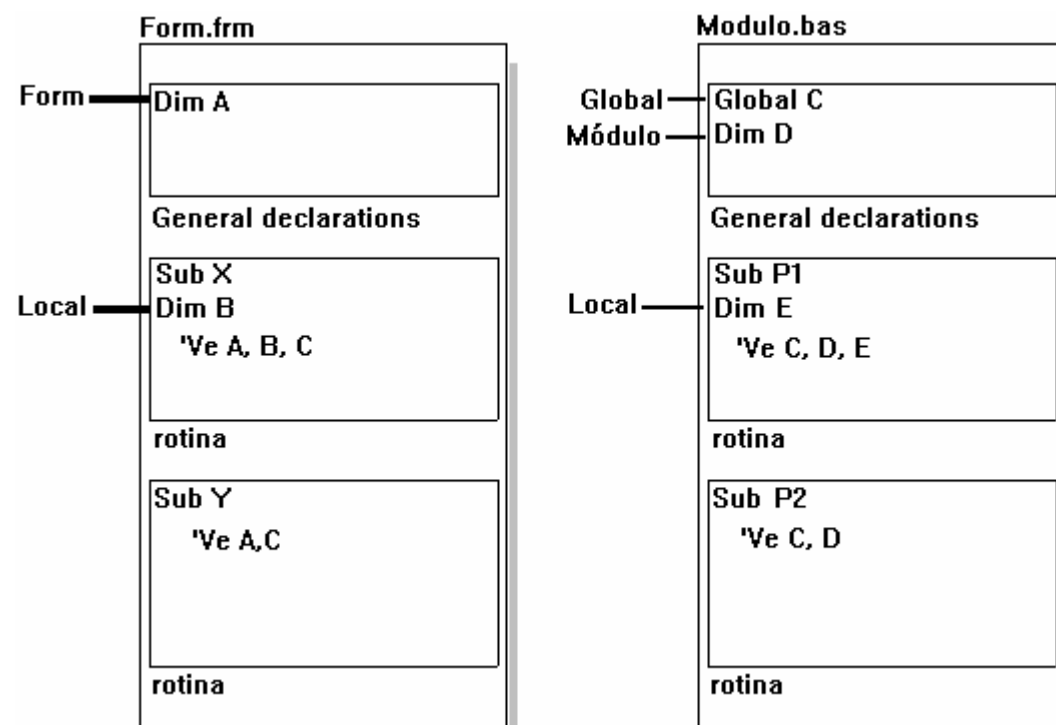


**Constantes** são entidades do programa cujo valor você necessita apenas saber, sem ter que atualizar. O Visual Basic mantém um arquivo, o CONSTANT.TXT que contém uma série de constantes predefinidas.

Este arquivo poderá ser adicionado ao seu projeto.

Para declarar uma constante, utilize a palavra reservada **Const** dentro do *general declarations* de qualquer formulário ou de um módulo de código (\*.BAS).

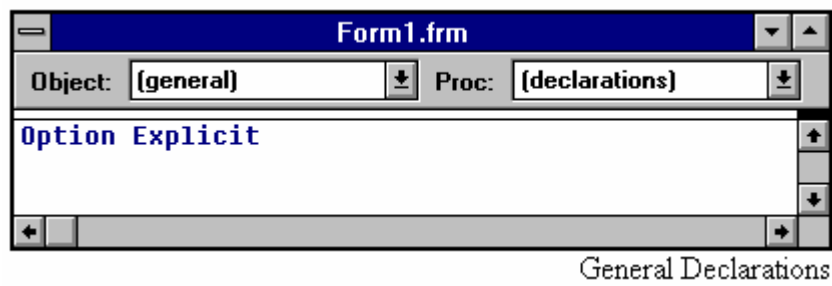
## 6.5 Escopo de Dados



Pode-se definir o escopo como sendo o nível de visibilidade de uma variável dentro de uma aplicação. Já vimos anteriormente que, uma aplicação Visual Basic é composta de formulários e módulos de procedimentos. Assim sendo, podemos ter variáveis visíveis, dentro de :

- ✓ Apenas uma rotina dentro de um formulário
- ✓ Dentro de todas as rotinas de um formulário
- ✓ Dentro de uma única rotina de um módulo de procedimentos
- ✓ Dentro de todas as rotinas de um módulo de procedimentos
- ✓ Dentro de todos os formulários e módulos de procedimentos da aplicação.

6.5.1 General Declarations



O "objeto" *general* de um formulário/módulo é a área onde você declara todas as variáveis e rotinas, que são comuns a todos os procedimentos daquele formulário/módulo. Na parte de *declarations*, você define suas variáveis (globais ou de módulo/formulário). Para definir uma rotina basta escrever *Sub nome\_rotina* ou *Function nome\_funcao (parametros)* que o Visual Basic abre uma janela para que o código seja escrito.

Escopo	Declaração	Visibilidade
Local	Dim, Static (dentro de um procedimento)	dentro do procedimento
Módulo ou Formulário	Dim (dentro da General Declarations de um módulo de procedimentos ou formulário)	dentro de todas as rotinas de um formulário ou módulo de procedimentos
Global	Global (dentro do General Declarations de um módulo de procedimentos)	em todos os pontos da aplicação

6.5.2 Static versus Dim

Ao usar a palavra reservada *Static* ao invés de *Dim* dentro de uma rotina de um formulário (.Frm) ou de um módulo de procedimento (.Bas), você estará trabalhando não com o escopo de uma variável, mas com sua durabilidade. Declarar uma variável como *static* dentro de uma rotina significa dizer que aquela variável não será reinicializada cada vez que a rotina for chamada (apenas no Load do formulário), no entanto ela só estará visível dentro daquela rotina. Ou seja, ela terá escopo Local e durabilidade enquanto o formulário estiver ativo.

## 6.6 Tipos de Dados Adicionais do VB

### Tipos de Dados Definidos pelo Usuário

Além dos tipos de dados oferecidos pelo Visual Basic, você também pode criar estruturas de dados suas. Um exemplo típico é criar estruturas semelhantes a um registro do seu arquivo, ou criar variáveis que servirão de padrão para a aplicação ( Ex.: Tipo nome é um string de 50 posições; qualquer nome do seu sistema será declarado como sendo do tipo nome, para que todas tenham o mesmo tamanho).

Sintaxe :

```
Type tipo-do-usuário  
    elemento as string  
    elemento as string  
End Type
```

Exemplo :

```
Type Reg_cliente  
    nome as string *50  
    telefone as string *11  
End Type
```

Em uma rotina utilizar :

```
Sub Le_Cliente  
Dim Cliente as Reg_cliente  
    Cliente.nome = TXBNome.text  
    Cliente.telefone = TXBTelefone.text  
End Sub
```

## Exemplo - Utilização de tipos de dados definidos pelo usuário

1. Inicializar o Visual Basic.

### Criação de Controles

2. Criar 2 controles do tipo Label com os seguintes valores nas propriedades: *Caption=Nome* e *Name=LBLNome* para um controle e *Caption=Telefone* e *Name=LBLTelefone* para o outro controle.
3. Criar 2 controles do tipo Text Box com os seguintes valores nas propriedades: *Name=TXBNome* para um controle e *Name=TXBTelefone* para o outro controle. Inicializar com branco a propriedade *Text* de ambos controles.
4. Criar 2 controles do tipo Command Button com os seguintes valores nas propriedades: *Name=CMDIncluir* e *Caption=Incluir* para um controle e *Name=CMDMostrar* e *Caption=Mostrar* para o outro controle.

### Criação do módulo BAS

5. Selecionar no menu as opções *File / New Module*

### Declaração do tipo definido pelo usuário

6. Digitar as seguintes linhas de código no *General / Declarations* do módulo BAS:

```
Type Reg_cliente
    nome As String * 50
    telefone As String * 11
End Type
```

### Declaração da variável do tipo Reg\_cliente

7. Digitar a seguinte linha de código no *General / Declarations* do form:  
`Dim Cliente as Reg_cliente`

### Codificando o evento *Click* do CMDIncluir

8. Digitar as seguintes linhas de código:  
`Cliente.nome = TXBNome.Text`  
`Cliente.telefone = TXBTelefone.Text`

### Codificando o evento *Click* do CMDMostrar

9. Digitar as seguintes linhas de código:  
`Dim Msg As String`  
`Msg = "Nome do cliente: " + Cliente.nome`  
`Msg = Msg + " Telefone do cliente: " + Cliente.telefone`  
`MsgBox Msg`

## 6.7 Vetores

Assim como várias outras linguagens de programação, o Visual Basic também permite a criação de vetores. Vetores são grupos de variáveis de um mesmo tipo que compartilham um nome. Cada elemento do vetor é identificado por um índice único.

Sintaxe :

```
Dim nome_vetor(limite_superior) as tipo_de_dado
```

```
Dim nome_vetor(limite_inferior To limite_superior) as tipo_de_dado
```

---

**Observação** Por default o primeiro elemento de um vetor tem índice 0, portanto se você declarar um vetor com o comando *Dim vetor(10) as integer*, você terá um vetor de 11 elementos e não de 10. Para evitar problemas prefira utilizar a segunda sintaxe, declarando *Dim vetor(1 to 10) as integer*.

---

### Vetores Multidimensionais ou Matrizes

O Visual Basic permite a criação de vetores com mais de uma dimensão. Desta forma, você pode criar vetores de até 60 dimensões (matrizes).

Exemplo :

```
Dim matriz(9, 9) as single  
Dim Matriz (1 to 10, 1 to 10) as single
```

### Vetores Dinâmicos

Em alguns casos, você sentirá a necessidade de utilizar um vetor, mas o tamanho do vetor somente será definido em runtime. O VB permite que você crie vetores dinâmicos, ou de tamanho variável. Para isso, o vetor deve ser declarado sem tamanho dentro do *general declarations* de um formulário ou módulo.

Exemplo :

```
'Colocar dentro do general declarations de um formulário ou módulo  
Dim Vetor() as string * 25
```

Feito isto, dentro da rotina onde vai ser definido o tamanho, redimensionar o vetor usando o comando *ReDim*.

Exemplo :

```
Sub Command1_click ()  
    ReDim Vetor (List1.listcount)  
End Sub
```

---

**Importante:** Sempre que você usa o comando *ReDim*, todos os valores contidos no vetor são perdidos e o vetor é todo preenchido com o valor NULL. Para aumentar o tamanho de um vetor sem que seu conteúdo seja perdido, utilizar a palavra chave *Preserve*.

---

Exemplo :

```
Sub Command1_click ()  
    ReDim Preserve Vetor (List1.listcount)  
End Sub
```

Para limpar uma matriz e reduzir o seu tamanho fazer :

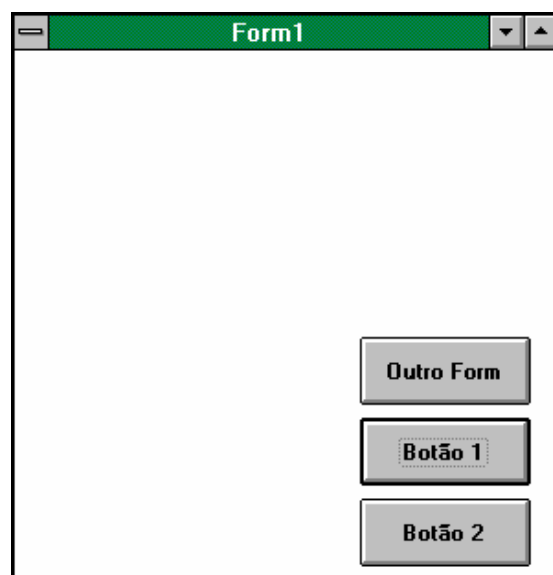
```
Redim vetor(0)
```

### Exemplo - Escopo de Dados

Para demonstrar a declaração de variáveis, criaremos um exemplo onde quatro variáveis serão declaradas diferentemente :

- VAR\_GLB - variável global.
- VAR\_FRM - variável a nível de formulário.
- VAR\_TIM - variável estática local.
- VAR\_LOC - variável estática local a dois procedimentos diferentes.

Teremos também dois formulários, onde um terá todas as funções de display e mudança de valores das variáveis e outro terá apenas a função de alterar o valor da variável de nível global.



1. Inicializar o Visual Basic.

#### Criação de Formulários

2. Nomear o formulário aberto de *FRMPrimeiro*.
3. Criar outro formulário. Nomeá-lo de *FRMSegundo*.
4. Criar um módulo de código, ou seja, um .BAS.



Criação de Controles

5. Em *FRMPrimeiro*, criar três botões. Atribuir os seguintes valores às suas propriedades :

Caption	Name
Outro Form	CMDOtroForm
Botão 1	CMDBotao1
Botão 2	CMDBotao2

6. Em *FRMPrimeiro*, criar um timer. Habilitá-lo e colocar como intervalo de funcionamento o valor de 1 segundo ( lembre-se que a unidade de VB é milésimo de segundo ).

Criação de Variáveis

- 7. Declarar uma variável global do tipo inteiro com o nome de VAR\_GL.
- 8. Declarar uma variável a nível de formulário no *FRMPrimeiro* do tipo inteiro com o nome de VAR\_FRM.
- 9. Declarar uma variável estática local ao evento *Timer* do *Timer* no *FRMPrimeiro* do tipo single com o nome de VAR\_TIM.
- 10. Declarar uma variável estática local ao evento *click* do *CMDBotao1* no *FRMPrimeiro* do tipo inteiro com o nome de VAR\_LOC.
- 11. Declarar uma variável estática local ao evento *click* do *CMDBotao2* no *FRMPrimeiro* do tipo inteiro com o nome de VAR\_LOC.

Escrevendo o código

12. Associar ao evento *Timer* do *Timer* no *FRMPrimeiro*, as seguintes linhas de código :

```
Var_Tim = Var_Tim + 1
If Var_Tim MOD 2 = 0 Then
    Var_Frm = Var_Frm + 1
Endif
```

13. Associar ao evento *Click* do *CMDBotao1* no *FRMPrimeiro*, as seguintes linhas de código :

```
Var_Loc = Var_Loc + 1
Print Var_Loc, Var_Frm, Var_gl
Print
```

14. Associar ao evento *Click* do *CMDBotao2* no *FRMPrimeiro*, as seguintes linhas de código :

```
Var_Loc = Var_Loc - 1
Print Var_Loc, Var_Frm, Var_gl
Print
```

15. Associar ao evento *Click* do *CMDOtroForm* no *FRMPrimeiro*, a seguinte linha de código:

```
Form2.Show
```

16. Associar ao evento *Load* do *FRMSegundo*, a seguinte linha de código :

```
Var_gl = Var_gl + 1
```

Executando a Aplicação

- 17. Pressionar F5 ou selecione Start do menu Run.
- 18. Pressione várias vezes os botões e observe o comportamento das variáveis.

## **7.0 Estruturas Lógicas e Condicionais**

### **Objetivo**

Sendo este um curso direcionado a profissionais de informática acostumados à utilização de estruturas de controle como condições e repetições, este módulo apresentará apenas as sintaxes das estruturas já conhecidas e também quaisquer novidades que estas estruturas no Visual Basic possuam.

### **7.1 If / Then / Else**

#### **Formato:**

**If** condição **Then** comando

**If** condição **Then**

bloco de comandos

**End If**

Existem dois tipos de estrutura condicional do tipo *If...Then...*. A primeira sintaxe, escrita em apenas uma linha, deve ser utilizada quando apenas um comando for ser executado como resultado de uma condição verdadeira. Quando, como resultado de uma condição, um bloco de comandos tiver que ser executado, torna-se necessária a utilização do *End If* como demarcador do final do bloco de comandos.

Os seguintes operadores podem ser usados na composição de condições : =, <>, <, >, <=, >=.

**If** condição1 **Then**

Bloco de comandos

**ElseIf** condição2 **Then**

Bloco de comandos

**Else**

Bloco de Comandos

**End If**

Nesta estrutura permite-se testar várias condições em um único bloco de *ifs*, e reagir diferentemente a cada uma das situações. Esta estrutura de comandos possui as seguintes características :

- ✓ Pode comportar um número ilimitado de *ElseIfs*
- ✓ Se nenhuma condição for verdadeira, o bloco de comandos seguindo o *Else* será executado.

## 7.2 Select Case

### Formato:

**Select Case** variável a ser testada

**Case** expressão

Bloco de comandos

**Case** expressão

Bloco de comandos

**Case Else**

bloco de comandos

**End Select**

A estrutura de comandos *Select Case* funciona como uma estrutura *If...Then...ElseIf...Else...* , porém é mais eficiente. Nos exemplos abaixo, demonstra-se o uso das palavras reservadas *TO* e *IS*, que diversificam ainda mais a utilização desta estrutura.

Exemplos :

```
Select case variável
  case 1,3,5,7 to 11
    comandos      'Entra se variável for igual a 1,3,5,7,8,9,10,11
  case 2, 4, 6, IS >=12
    comandos      'Entra se variável for igual a 2,4,6,12,13,14,15,...
End Select
```

```
Select case palavra
  Case "A" to "a"
    'Entra se pertencer a este intervalo. Ex.: "BRASIL" entra
    comandos
  Case "H" to "c"
    comandos
End Select
```

### 7.3 Do While

**Formato:**

**Do While** condição  
bloco de comandos

**Loop**

**Do**  
bloco de comandos

**Loop While** condição

Ambas sintaxes acima tem a mesma funcionalidade : executar um bloco de comandos enquanto uma condição for verdadeira. A diferença entre elas é : a composição *do while/loop* testa a condição antes de executar o bloco de comandos. Já a composição *do/loop while* executa o bloco de comandos uma vez e depois testa a condição.

### 7.4 Do Until

**Formato:**

**Do Until** condição  
bloco de comandos

**Loop**

**Do**  
bloco de comandos

**Loop Until** condição

Ambas sintaxes acima tem a mesma funcionalidade : executar um bloco de comandos até que uma condição se torne verdadeira. A diferença entre elas é : a composição *do until/loop* testa a condição antes de executar o bloco de comandos. Já a composição *do/loop until* executa o bloco de comandos uma vez e depois testa a condição.

### 7.5 For Next

**Formato:**

```
For contador = valor_inicial To valor_final {Step incremento}  
  
    bloco de comandos  
  
    { Exit For }  
  
    bloco de comandos  
  
Next contador
```

Esta estrutura de controle permite a execução de um bloco de comandos por um número fixo de vezes. O incremento pode ser positivo, negativo e em valores não inteiros. Cuidado para não criar loops infinitos!!!

O *Exit For* permite a finalização da execução do *For Next* a partir daquele ponto. Deve-se ter cuidado para não desestruturar o programa.

## 7.6 Go to

**Formato:**

```
GoTo label  
  
GoTo linha
```

Este comando faz com que a execução do programa pule para o label ou linha especificada. Deve ser usado basicamente em rotinas de tratamento de erro, pois sua utilização generalizada pode causar desestruturação do programa.

## **8.0 Codificando em VB**

### **Objetivo do módulo**

Fornecer as ferramentas necessárias para que você possa começar a escrever o código por trás das interfaces em Visual Basic.

### **8.1 Procedimentos (Sub e Functions)**

Procedimentos são conjuntos de comandos Visual Basic que podem ser chamados de unidades lógicas. Existem dois tipos de procedimentos : subrotinas e funções.

**Subrotinas** ➔ Conjunto de comandos que não obrigatoriamente retorna um valor ao procedimento que o chamou. Ao chegar ao final de sua execução retorna ao módulo que fez a chamada.

Sintaxe :

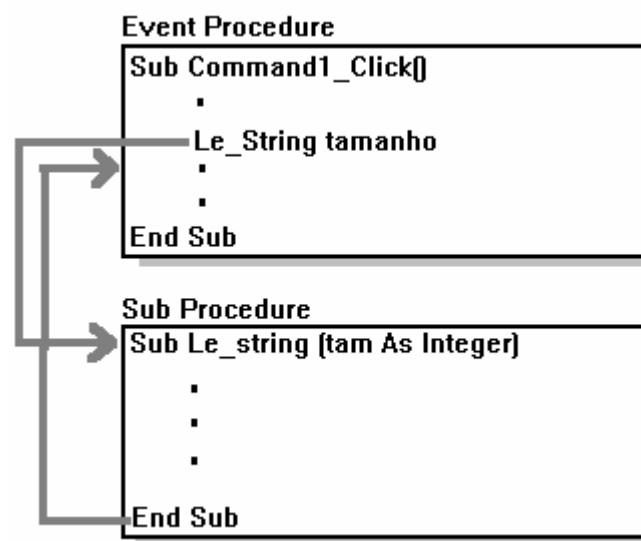
```
Sub Nome_da_subrotina()  
    Bloco de comandos  
End Sub
```

**Funções** ➔ Similar a uma rotina. No entanto, é de um tipo de dados, assim como variáveis. Ao final de sua execução, retorna um valor ao módulo que fez a chamada.

Sintaxe :

```
Function nome_função() As tipo_de_dados  
    Blocos de comandos  
    nome_função = Valor  
End Function
```

## 8.2 Passando Parâmetros para uma Subrotina



### Argumentos

Qualquer procedimento pode receber dados, se tiver sido declarado para tal. Cada argumento passado deve ter seu equivalente ( do mesmo tipo de dado ) na lista de parâmetros da subrotina ou função.

Sintaxe - declaração da subrotina :

```

Sub nome_subrotina ( parâmetro1 as Integer, parâmetro2 as Integer)
    Bloco de comandos
End Sub
  
```

Sintaxe - chamada da subrotina :

```

nome_subrotina argumento1, argumento2
  
```

Sintaxe - declaração de função :

```

Function nome_função ( parâmetro1 as Integer, parâmetro2 as Integer) as Integer
    Bloco de comandos
    nome_função = valor
End Function
  
```

Sintaxe - chamada de função :

```

Dim resultado as Integer
Resultado = nome_função(argumento1, argumento2)
  
```

**Passagem de Argumentos por Valor e por Referência**

A passagem de argumentos para uma função ou subrotina pode ser feita por Valor ou por Referência. Na passagem por valor, a subrotina ou função recebe apenas uma cópia do argumento, sendo assim qualquer alteração no argumento dentro da subrotina/função não terá efeito no dado real.

Já na passagem de argumentos por referência, a subrotina/função recebe o endereço que realmente contém o dado. Assim sendo, qualquer alteração no argumento alterará o dado de verdade.

O Visual Basic, por default, passa argumentos por referência. Para passar argumentos por valor, utiliza-se a palavra chave *ByVal* na lista de parâmetros, ou então coloca-se o argumento entre parênteses na chamada da subrotina/função.

Observações :

1. Propriedades de objetos somente podem ser passadas por valor.
2. Se o seu argumento é do tipo variant, e a sua subrotina/função espera um parâmetro de um tipo de dado definido, a passagem do argumento deve ser feita por valor.
3. Você precisa apenas utilizar a palavra chave ByVal ou então os parênteses extra.

Exemplos :

-- declaração da subrotina --

```
Sub nome_subrotina ( ByVal parâmetro1 as Integer )  
    Bloco de comandos  
End Sub
```

-- chamada da subrotina --

```
nome_subrotina (argumento1)
```

-- declaração de função --

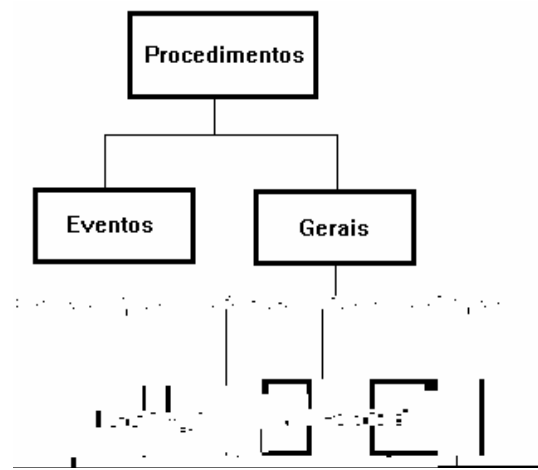
```
Function nome_função ( ByVal parâmetro1 as Integer) as Integer  
    Bloco de comandos  
    nome_função = valor  
End Function
```

-- chamada de função --

```
Dim resultado as Integer  
Resultado = nome_função((argumento1))
```



### 8.3 Tipos de Procedimentos



O Visual Basic possui duas categorias de procedimentos : Eventos e Gerais. Procedimentos gerais, como vimos anteriormente podem ser funções ou subrotinas, e são ativados pelo próprio programador via código.

#### Event Procedures

Procedimentos de Eventos, ou Event Procedures, são sempre ativados a partir de um evento acionado pelo usuário ou então pelo sistema (Windows).

Event Procedures estão sempre ligados a um formulário ou controle. O nome do procedimento indica o evento e a qual formulário/controle que o evento está associado.

Sintaxe : `Sub nome-objeto_nome-evento()`

Exemplo : `Command1_Click`, `Form1_Load`, `Text1.Change`, etc.

## 8.4 Criando uma Event Procedure

O Visual Basic já fornece as declarações de todos os procedimentos de eventos existentes para cada controle. Para inserir código de tratamento para qualquer evento, você deve abrir a janela de código, selecionar o objeto que você deseja tratar, selecionar o evento, e então digitar o código.

### Ativando um Procedimento de Evento

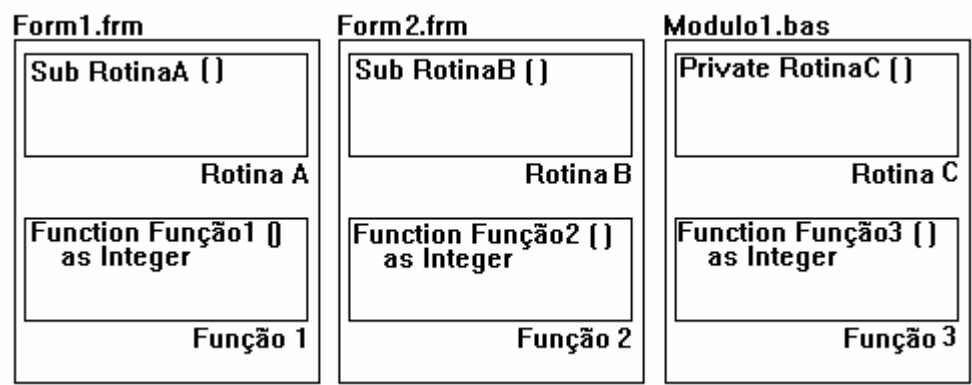
O Visual Basic reconhece automaticamente o acontecimento de um evento. Ao reconhecer o evento, ele executa imediatamente o código de tratamento para aquele evento.

### Escopo de Procedimentos de Eventos

Procedimentos de Eventos somente estão disponíveis dentro do formulário onde eles foram definidos.

8.5 Escopo de Procedimentos Gerais

Assim como variáveis, procedimentos gerais também têm escopo. Veja no exemplo abaixo de onde as funções e subrotinas declaradas poderão ser vistas :



Formulário/Módulo	Pode ver
Form1.frm	Rotina A, Função1, Função 3
Form2.frm	Rotina B, Função2, Função 3
Modulo.bas	Rotina C, Função 3

Como você pôde observar no exemplo acima, existem três escopos de rotinas: a nível de formulário, a nível global e a nível privado em módulos.

Escopo a nível de formulário

Declarada com as palavras reservadas Sub ou Function dentro de um formulário. Somente podem ser chamadas de dentro dele.

Escopo a nível global

Declarada com as palavras reservadas Sub ou Function dentro de um módulo. Podem ser chamadas de qualquer ponto da aplicação.

Escopo privado em módulos

Utilizar a palavra reservada Private antes de Sub ou Function dentro de um módulo. Somente podem ser chamadas de dentro dele.

8.6 Funções de Conversão de Expressões Numéricas e de Caracteres

Função	Descrição
Chr	Retorna um caracter para o código ANSI informado. Ex.: Chr(13) + Chr(10) , CarriageReturn e LineFeed.
Format	Rotina que retorna um número no formato que você quiser. Ex.: Format(Now, "dd-mm-yy"), traz a data de hoje no formato especificado.
Lcase	Retorna o caracter minúsculo do caracter informado. Ex.: LCase("H"), retorna "h".
Left	Traz os <i>n</i> caracteres mais a esquerda. Ex.: Left("gravador",5), retorna "grava".
Len	Traz o tamanho de um string. Ex.: Len("Mar"), retorna 3.
LTrim	Retira espaços à esquerda em um string.
Mid	Retorna parte de uma string. Ex.: Mid( "reflorestamento", 3, 8), retorna "floresta".
Right	Traz os <i>n</i> caracteres mais a direita. Ex.: Right("armário",3), retorna "rio".
Rtrim	Retira espaços à direita em um string.
Trim	Retira espaços à direita e à esquerda em um string.
Ucase	Retorna o caracter maiúsculo do caracter informado. Ex.: UCase("g"), retorna "G".
Val	Converte um string de dígitos em um número. Ex.: Val("100"), retorna 100; Val("1345,98") retorna 1345.

Função	Descrição
CCur	Converte uma expressão character para tipo currency.
Cdbl	Converte uma expressão character para tipo double.
Cint	Converte uma expressão character para tipo integer.
CLng	Converte uma expressão character para tipo long.
CSng	Converte uma expressão character para tipo single.
CStr	Converte uma expressão character para tipo string.
CVar	Converte uma expressão character para tipo variant.

OBS.: A função *CVAR* reconhece o indicador de decimal do Brasil (.). Para capturar um número de casas decimais, usá-la em conjunção com *CDBL*, *CINT*, *CLNG*, *CSNG*, dependendo do tipo de número que desejar armazenar.

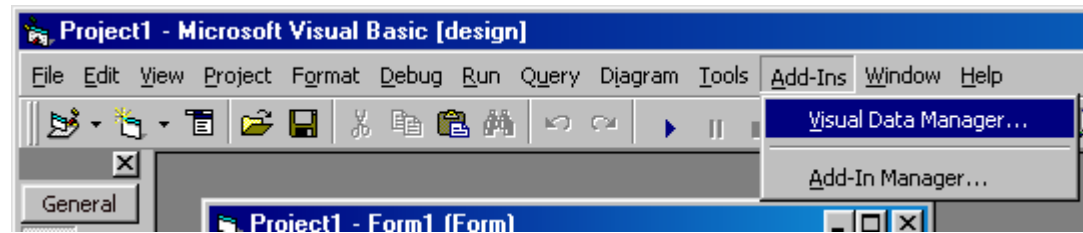
## 9. DATA MANAGER

O DataManager é um gerenciador de banco de dados, permite criar, modificar e apagar um banco de dados, inserir, modificar e excluir registros e criar índices.

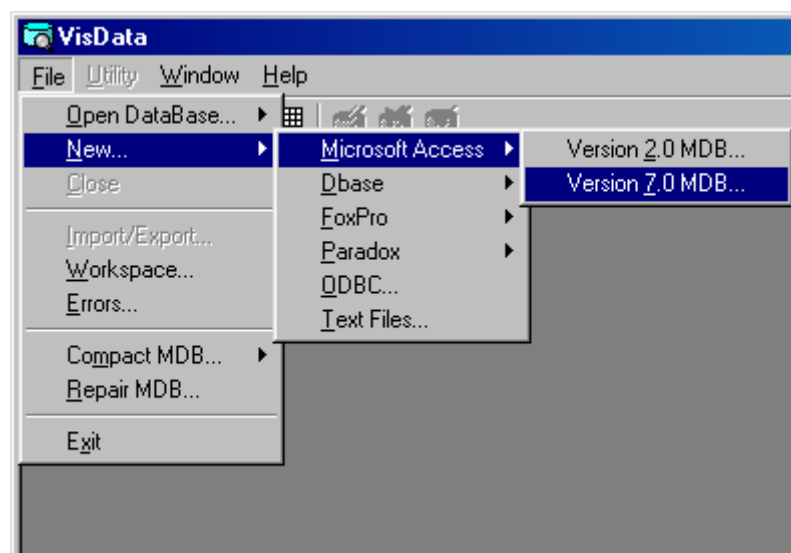
Neste capítulo será visto passo-a-passo como construir um banco de dados ACCESS.

### 9.1 Criando um Banco de Dados

Para criar um banco de dados selecione a opção Visual Data Manager do menu Add-Ins

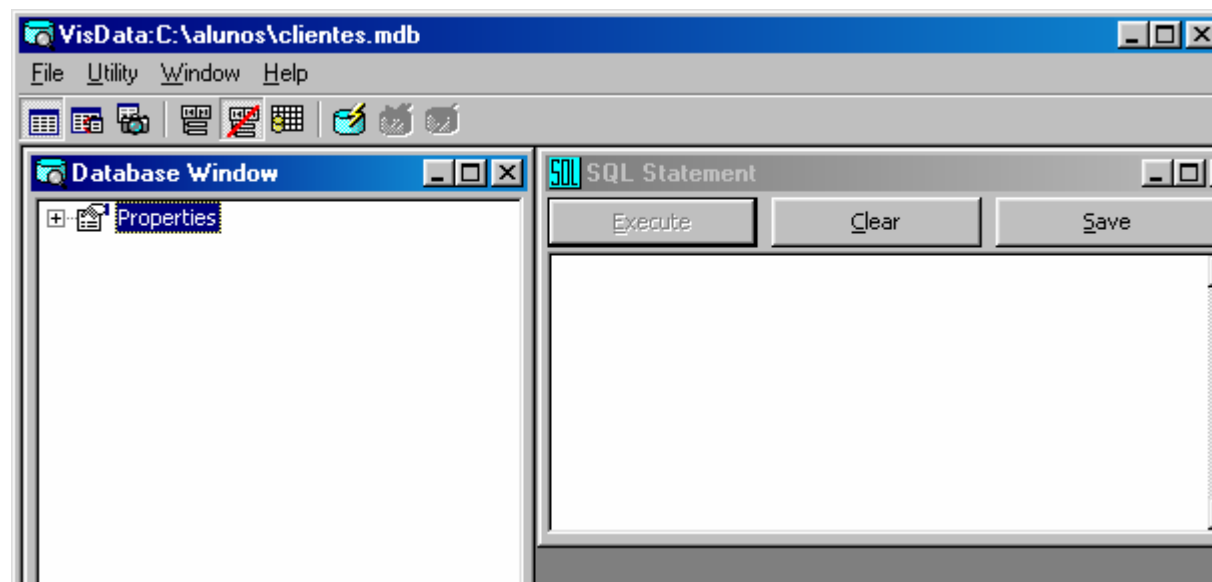


No Data Manager, escolha a opção New do menu File, no exemplo estamos criando um Banco de Dados ACCESS da versão 7.0



Nomeie e salve o Banco de Dados no diretório desejado

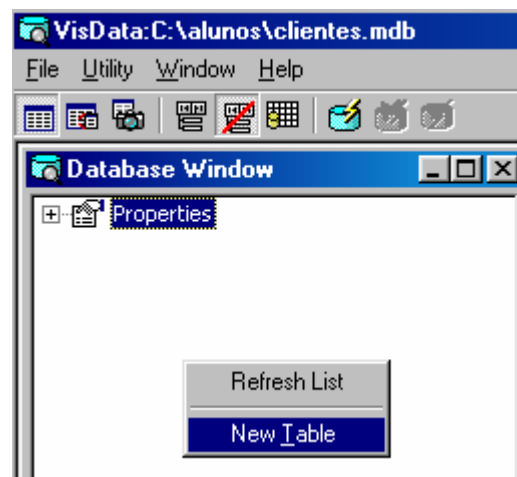
A seguinte janela será exibida:



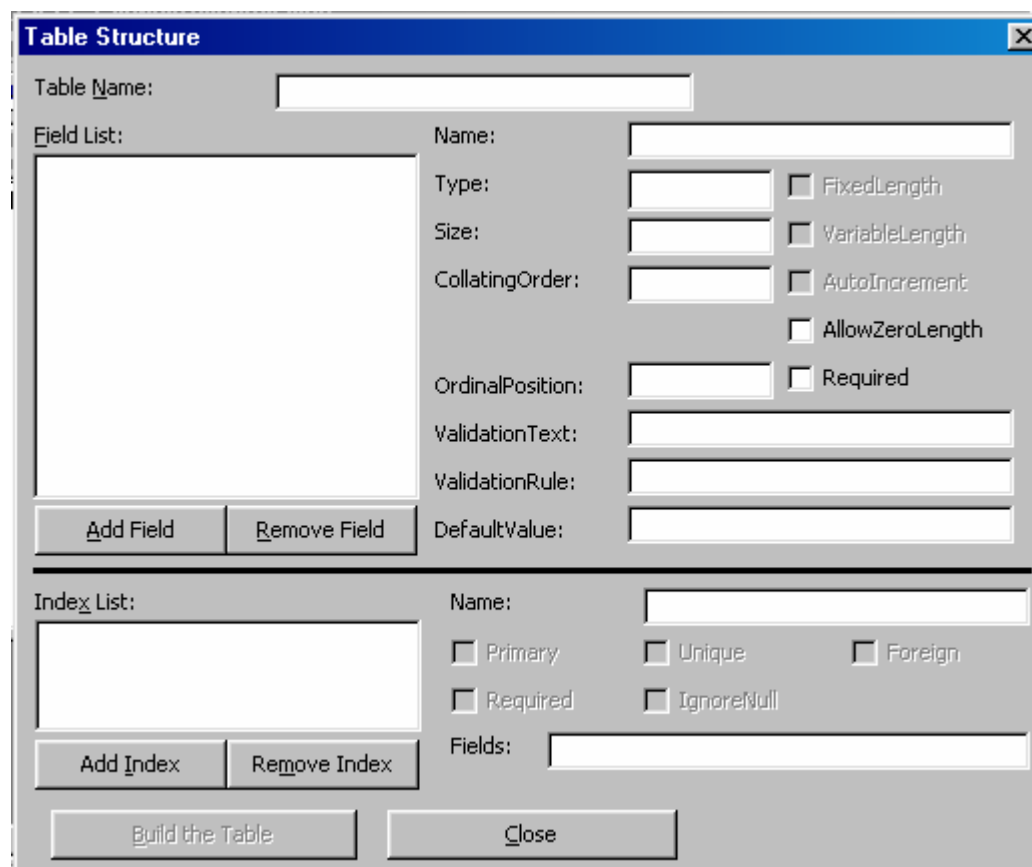
## 9.2 Criação de tabelas

Vamos montar a estrutura do banco de dados, iniciaremos pelas tabelas.

Dentro da janela Database Window dê um clique com o botão direito do mouse e selecione a opção New Table do menu de contexto.

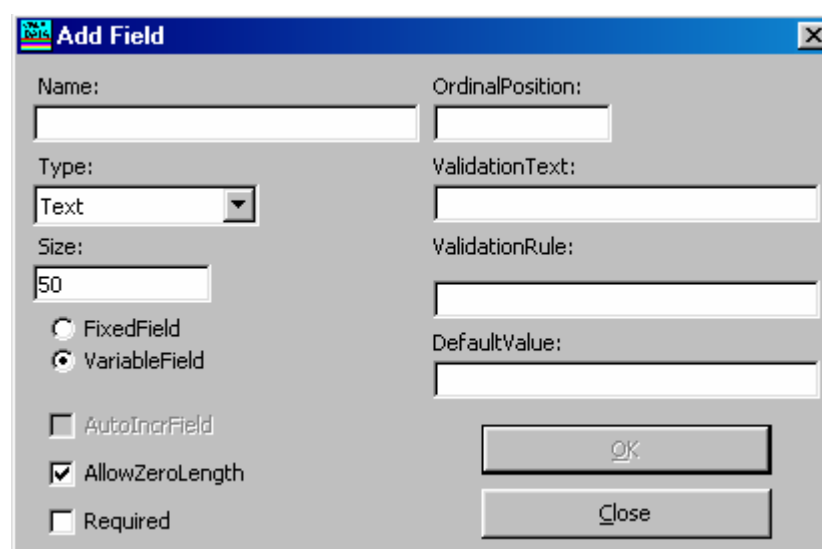


A seguinte janela aparece:



The **Table Structure** dialog box is used to define the structure of a new table. It features a **Table Name** field at the top. Below it, the **Field List** is shown as an empty list box. To the right of the field list, there are input fields for **Name**, **Type**, **Size**, **CollatingOrder**, **OrdinalPosition**, **ValidationText**, **ValidationRule**, and **DefaultValue**. Checkboxes for **FixedLength**, **VariableLength**, **AutoIncrement**, **AllowZeroLength**, and **Required** are also present. Below the field list are **Add Field** and **Remove Field** buttons. The **Index List** section at the bottom has an empty list box, checkboxes for **Primary**, **Unique**, **Foreign**, **Required**, and **IgnoreNull**, and a **Fields** input field. **Add Index** and **Remove Index** buttons are located below the index list. At the very bottom are **Build the Table** and **Close** buttons.

Dê um nome para sua tabela e clique no botão Add Field para adicionar um campo à sua tabela. A seguinte janela aparece:



The **Add Field** dialog box is used to define the properties of a new field. It includes input fields for **Name** and **OrdinalPosition**. The **Type** is set to **Text** in a dropdown menu. The **Size** is set to **50**. There are radio buttons for **FixedField** and **VariableField**, with **VariableField** selected. Checkboxes for **AutoIncField**, **AllowZeroLength** (checked), and **Required** are also present. On the right, there are input fields for **ValidationText**, **ValidationRule**, and **DefaultValue**. At the bottom right are **OK** and **Close** buttons.

Defina as propriedades do novo campo:

Propiedades	Descrição
Name	Nome do campo
Type	Tipo do campo. Pode ser: Boolean, Byte, Integer, Long, Currency, Single, Double, Date/Time, Text, Binary ou Memo
Size	Tamanho do campo. É definido automaticamente, após a seleção do tipo de dados, podendo ser alterado apenas para campos do tipo Text
AllowZeroLenght	Define se o campo permite dados de comprimento zero
Required	Define se o preenchimento do campo é obrigatório
Ordinal Position	Define a ordem do campo na tabela
Validation Text	Mensagem a ser exibida ao se violar a regra de validação
Validadtion Rule	Regra de validação para os dados a serem inseridos no campo
Default Value	Valor padrão para o campo

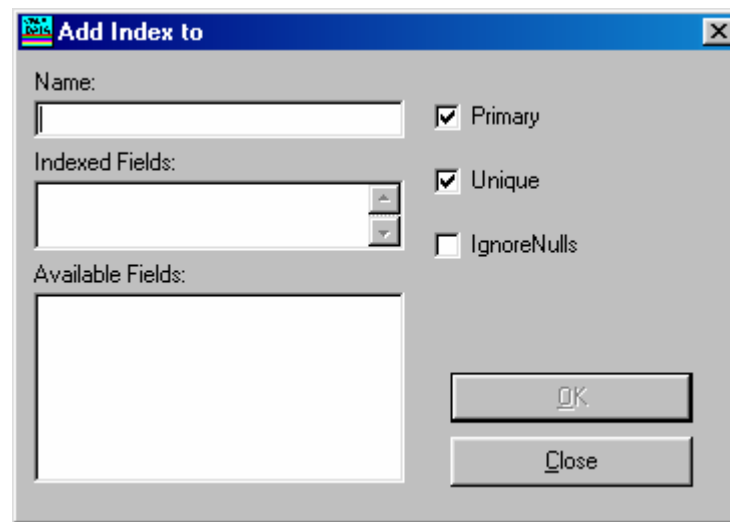
Definida as propriedades do novo campo, pressione o botão OK. Repita o mesmo processo para todos os campos que desejar incluir e pressione o botão Close para finalizar a criação de novos campos.

Agora vamos criar índices para a tabela. Clique no botão AddIndex. A seguinte janela é mostrada:

Escolha um nome para o índice e os campos a serem indexados dando em duplo clique sobre eles (na caixa Avaliable Fields). Especifique se o índice é primário (Primary) e se o campo aceita repetições (Unique).

Para criar outro índice clique no botão OK e para finalizar a criação de índices clique no botão Close.

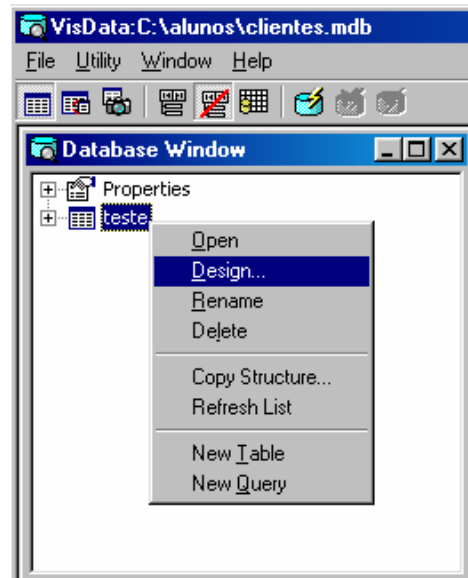




Após criar os campos e índices desejados, você deve confirmar a estrutura dando um clique no botão Build the Table.

### 9.3 Modificando a estrutura de uma tabela existente

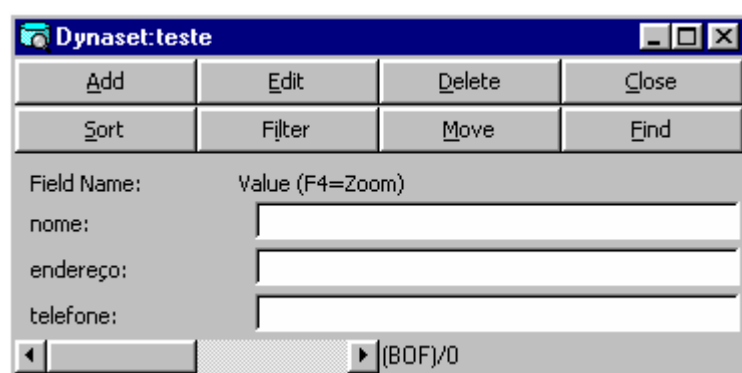
Você pode adicionar novos campos ou alterar algumas das propriedades de uma tabela existente. Isso é feito clicando com o botão direito do mouse sobre a tabela que você deseja alterar, em seguida escolha a opção Design do menu de contexto.



## 9.4 Inserindo dados

Após criado o Banco de Dados você pode inserir, apagar, editar e pesquisar registros em cada tabela.

Na janela Database Window, clique com o botão direito do mouse sobre a tabela que deseja trabalhar e selecione a opção Open. A seguinte janela é mostrada:



The screenshot shows a window titled "Dynaset: teste". It contains a grid of buttons: Add, Edit, Delete, Close in the first row, and Sort, Filter, Move, Find in the second row. Below the buttons, there is a section for field names and values. The "Field Name:" label is followed by "Value (F4=Zoom)". Below this, there are three input fields labeled "nome:", "endereço:", and "telefone:". At the bottom, there is a navigation bar with a left arrow, a right arrow, and the text "(BOF)/0".

Para inserir um registro, clique no botão Add e entre com os dados nas caixas referentes aos campos da tabela. Em seguida clique no botão Update.

Para excluir, primeiro selecione o registro a ser apagado e clique no botão Delete.

Para alterar o valor de algum dado, clique no botão Edit. Nas caixas correspondentes entre com o novo valor e clique no botão Update.

Com os outros botões você pode impor um critério de ordenação (SORT), fazer uma seleção dos registros (FILTER), mover diretamente a um certo registro (MOVE) e realizar uma busca (FIND).

## 10. Utilizando um Banco de Dados

A abertura de um banco de dados é feita com o uso do método *OpenDatabase*, como é mostrado no exemplo abaixo:

*Dim banco as Database*

*Set banco = OpenDatabase("C:\diretório\banco.mdb")*

No código acima foi definido na primeira linha uma variável objeto de banco de dados (Banco), e a seguir usamos o método *OpenDatabase* para designar o caminho de nosso banco de dados.

Após abrir o banco de dados você deve definir um *Recordset* para ter acesso aos dados contidos no banco de dados.

### 10.1 Recordset

É o recurso mais importante relacionado com os objetos de acesso a dados. Há três tipos de Recordset disponíveis:

**Tabelas** → são estruturas físicas que contém os dados em um banco de dados

**Dynaset** → Um conjunto de ponteiros para acesso de campos e registros localizados em uma ou mais tabelas.

**Snapshot** → Uma cópia somente de leitura dos dados de uma ou mais tabelas.

Nos nossos estudos utilizaremos a Tabela, cujo as vantagens são :

- É a única forma de Recordset que aceita o uso de índices, tornando a pesquisa em uma tabela muito mais rápida
- As alterações realizadas pelos usuários ficam imediatamente disponíveis

10.2 Abertura de uma tabela

Para abrir uma tabela, defina um objeto Rcordset e use o método *OpenRecordset* para acessar a tabela.

Informe a constante *DbOpenTable* para identificar o tipo de recordset criado. Veja o código abaixo:

```
Dim Banco as Databse

Dim Tabela as Recordset

Set Banco=OpenDatabase(C:\diretorio\banco.mdb")

Set Tabela=Banco.OpenRecordset("Clientes",DbOpenTable)
```

No código acima é definida a variável objeto Recordset Tabela e atribuido a esta variável a tabela Clientes do banco de dados Banco.mdb.

10.3 Métodos de movimentação e localização

Os métodos de movimentação permitem a passagem de um registro para outro no interior do *Recordset*, e alteram a posição do ponteiro do registro ao passar de um registro ativo para outro registro. Os métodos de movimentação são:

Método	Ação
MoveFirst	Movimenta o ponteiro para o primeiro registro do recordset
MoveNext	Movimenta o ponteiro do registro para o próximo registro. Se não houver registro seguinte o flag final de arquivo (EOF) será ativado
MovePrevious	Movimenta o ponteiro do registro para o registro anterior. Sé não houver registro anterior o flag de início de arquivo (BOF) será ativado
MoveLast	Move o ponteiro para o último registro do recorset
Move n	Movimenta o ponteiro de registro para n posições para frente (n

	positivo) ou para trás (n negativo) a partir do registro ativo. Se a movimentação levar o ponteiro além dos limites do recordset ocorrerá um erro
--	---

O método *Seek* somente pode ser utilizado com o recordset do tipo tabela (*Table*).

Um comando *Seek* só localiza o primeiro registro que satisfaz aos valores de índice especificados. Para invocar o método *Seek* é necessário fazer a chamada ao método usando operadores de comparação (<, <=, >, >=, =, <>), lembrando que os valores de chaves que estão sendo comparados devem ter o mesmo tipo de dados dos campos no índice de controle.

Após executar o método de localização você deve verificar o status da propriedade *NOMATCH*. Se *Nomatch* é verdadeira o método não encontrou o registro desejado, se *Nomatch* é falsa o ponteiro do registro se posiciona no registro encontrado. Veja o exemplo a seguir :

```
Dim Banco as Database
Dim Tabela as Recordset
Set Banco=OpenDatabase(C:\diretorio\banco.mdb")
Set Tabela=Banco.OpenRecordset("Clientes",DbOpenTable)
Tabela.index = "nome"
Tabela.seek "=", "Bill Gates"
If tabela.nomatch then
    MsgBox "Cliente não localizado"
Else
    MsgBox "Cliente localizado com sucesso"
Endif
```

No código acima definimos a recordset *Tabela* do tipo *Table* e a seguir definimos o índice ativo *Tabela.index = "nome"* e invocamos o método *Seek* usando o operador = com nome a ser procurado. Note que o índice já deve ter sido criado, o operador deve estar entre aspas e que deve ser separado por vírgula do valor a ser localizado.

10.4 Trabalhando com a tabela

Após a consulta realizada na tabela, você pode incluir, excluir, editar o seu conteúdo, para isso utilizamos as propriedades:

Propriedades	Ação
AddNew	Prepara a tabela para a inserção de um novo registro
Delete	Apaga o registro atual da tabela
Edit	Prepara a tabela para a edição do registro atual
Update	Atualiza os dados após uma operação de gravação ou alteração de um registro

O Código abaixo demonstra a utilização das propriedades descritas acima:

```
Dim banco As Database
Dim tabela As Recordset
Private Sub cmdapagar_Click()
    tabela.Delete
End Sub
Private Sub cmdeditar_Click()
    tabela.Edit
    tabela.endereco = txtendereco
    tabela.telefone = txttelefone
    tabela.Update
End Sub
Private Sub cmdgravar_Click()
    tabela.AddNew
    tabela.nome = txtnome
    tabela.endereco = txtendereco
    tabela.telefone = txttelefone
    tabela.Update
End Sub
Private Sub Cmdlocalizar_Click()
    tabela.Index = "nome"
    tabela.Seek "=", txtnome
    If tabela.NoMatch Then
        MsgBox "Cliente não localizado"
    Else
        MsgBox "Cliente localizado com sucesso"
        txtnome = tabela.nome
    End If
End Sub
```

```
        txtendereco = tabela.endereco  
        txttelefone = tabela.telefone  
    End If  
End Sub  
Private Sub Form_Load()  
    Set banco = OpenDatabase("c:\alunos\teste.mdb")  
    Set tabela = banco.OpenRecordset("teste", dbOpenTable)  
End Sub
```