

# Delphi 4

# SUMÁRIO

<b>PRIMEIROS PASSOS</b>	<b>2</b>
ANATOMIA DE UMA JANELA	3
BARRA DE MENU, BARRA DE FERRAMENTAS E PALETA DE COMPONENTES	4
<i>Barra de Ferramentas</i>	4
PALETA DE COMPONENTES	4
JANELAS DO DELPHI	6
<b>MEU PRIMEIRO PROGRAMA</b>	<b>8</b>
ADAPTAR AS PROPRIEDADES DOS OBJETOS	9
ESCREVER O CÓDIGO PARA OS EVENTOS ASSOCIADOS.	12
<b>EXEMPLO I - CALCULADORA</b>	<b>17</b>
PROPRIEDADES	23
<i>BorderStyle</i>	23
<i>Default</i>	24
<i>Tabstop</i>	25
<i>Name</i>	25
<i>Caption</i>	26
<i>Text</i>	26
MÉTODO	26
<i>Setfocus</i>	26
VARIÁVEIS NO DELPHI	26
<i>Formas de Declarar uma Variável</i>	27
FORMATAÇÃO DE NÚMEROS	28
<i>Formatando Data e Hora:</i>	28
MODIFICANDO A CALCULADORA	29
<b>DEPURAÇÃO</b>	<b>34</b>
<b>EXEMPLO II - JOGO DA VELHA</b>	<b>38</b>
UNIDADE (UNIT)	39
<b>EXEMPLO III - BLOCO DE NOTAS</b>	<b>51</b>
<b>EXEMPLO IV - RELÓGIO DESPERTADOR</b>	<b>61</b>
<b>MÉTODOS GRÁFICOS</b>	<b>65</b>
DESENHO DE PONTO	65
CORES	66
DESENHO DE LINHAS	66
DESENHO DE RETÂNGULOS	70
DESENHO DE ELIPSES	71
FIGURAS	72
<b>EXEMPLO V - CATÁLOGO</b>	<b>77</b>
INSERINDO UMA NOVA UNIDADE	79
TIPOS DE DADOS	80
REGISTROS	80
PONTEIRO DE REGISTROS	82
<b>BANCO DE DADOS SIMPLES</b>	<b>88</b>
DATABASE DESKTOP	89
<b>LISTA DE EXERCÍCIOS</b>	<b>108</b>



# Delphi 4

## PRIMEIROS PASSOS

---

Vantagens :

- Facilidade em alterações e implementações
- Melhor Estruturação do código
- Velocidade
- Verdadeira orientação a objetos

Delphi possui um ambiente de desenvolvimento fácil de usar, com uma grande Biblioteca de Componentes Visuais (VCL - Visual Component Library). A VCL contém código de botões, campos, rótulos, gráficos, caixas de diálogo e acesso a tabelas de bancos de dados, e foi desenvolvida levando em conta as velocidades no desenvolvimento de aplicativos e na execução destes aplicativos.

O rápido desenvolvimento de aplicativos é possível graças aos vários controles disponíveis na paleta de componentes, onde o programador escolhe um destes componentes, e coloca-o diretamente no local desejado, dentro de um **formulário**. O formulário será a janela do aplicativo apresentada ao usuário.

O Delphi trabalha com eventos que dão início à alguma rotina de trabalho, ou seja, o programa fica parado até que um evento ocorra. Ele utiliza o modelo orientado objetos, permitindo além do uso de objetos existentes, também a criação de outros tantos.

Um programa tradicional, feito para ser executado em DOS, é organizado em torno de estruturas de dados com um loop principal e uma série de sub-rotinas constituindo o aplicativo, com procedimentos e funções separados para manipular os dados.

Um programa orientado a objetos e eventos é organizado em torno de um conjunto de objetos. Onde objeto é uma variável com propriedades que o definem, e vários códigos dando funcionalidade a este objeto. Ou seja, objetos são estruturas que combinam dados e rotinas em uma mesma entidade.

Um Objeto possui dados internos, que não podem ser acessados por outros objetos e dados externos, também chamados de propriedades, estas podendo ser acessadas de fora deste objeto. De maneira semelhante, um objeto possui rotinas internas que são usadas apenas internamente e rotinas externas, também chamadas de métodos, que podem ser acessadas externamente.

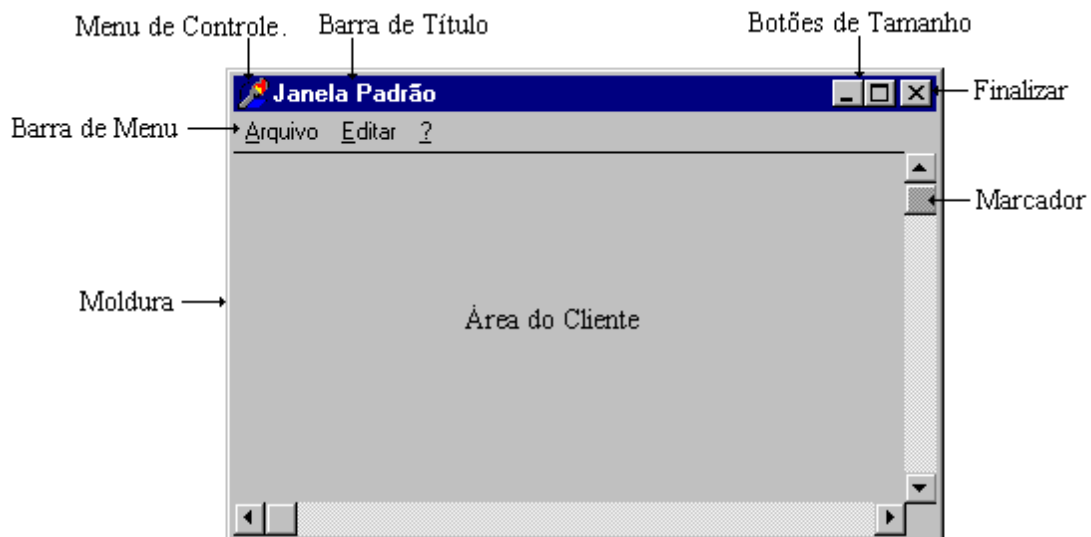
Um carro é um objeto que possui propriedades e métodos. A tabela abaixo lista algumas propriedades e comportamentos do objeto real **carro**.

Propriedades	Métodos
cor	dar partida
comprimento	mudar marcha
potência do motor	acelerar
tipo de pintura	frear

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

### ***ANATOMIA DE UMA JANELA***

A figura abaixo, mostra uma janela típica do Windows e a denominação de seus componentes básicos.



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

*Barra de Título* - Acima da moldura superior como nome da janela e documento corrente.

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

*Botões de Tamanho* - À direita da Barra de Título. São dois botões, no primeiro temos um traço, o outro com duas janelinhas ou uma janela desenhadas. Se forem duas janelinhas, mostra que a janela está maximizada, e se for uma janela um pouco maior, mostra que a janela está em seu tamanho normal e pode ser maximizada. O botão com um traço serve para minimizar a janela.

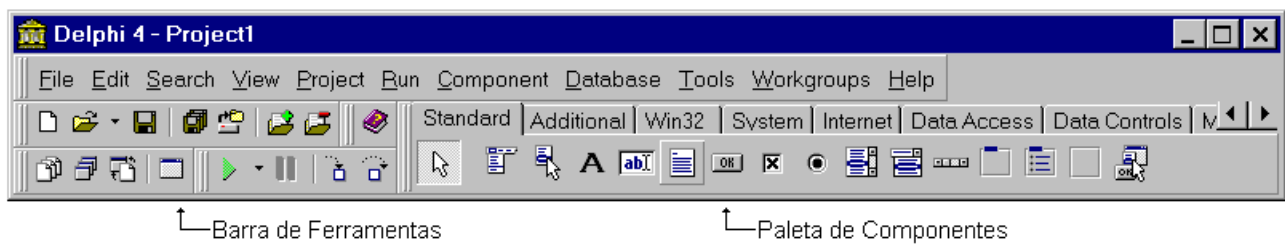
*Barra de Menu* - Está abaixo da barra de título e contém as opções de controle do aplicativo.

**Área do Cliente** - É a parte interna da janela, também chamada de área do documento, utilizada para inserir os controles da nossa aplicação.

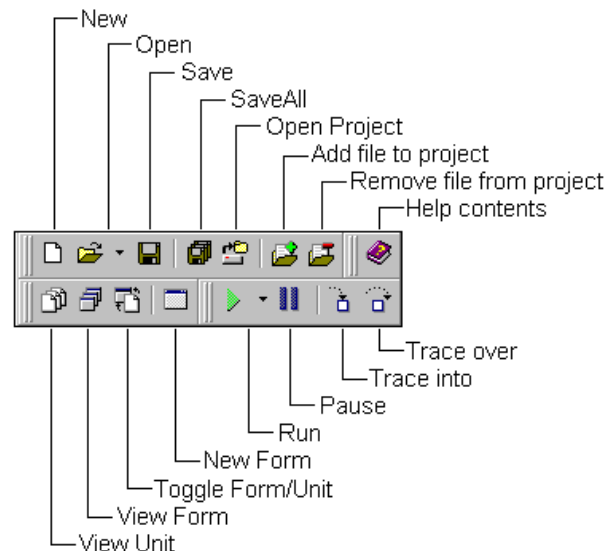
**Área Externa à Janela** - É sua Mesa de Trabalho onde você pode ter tantos aplicativos abertos quantos necessitar e a memória do seu computador permitir.

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

### ***BARRA DE MENU, BARRA DE FERRAMENTAS E PALETA DE COMPONENTES***



### **Barra de Ferramentas**



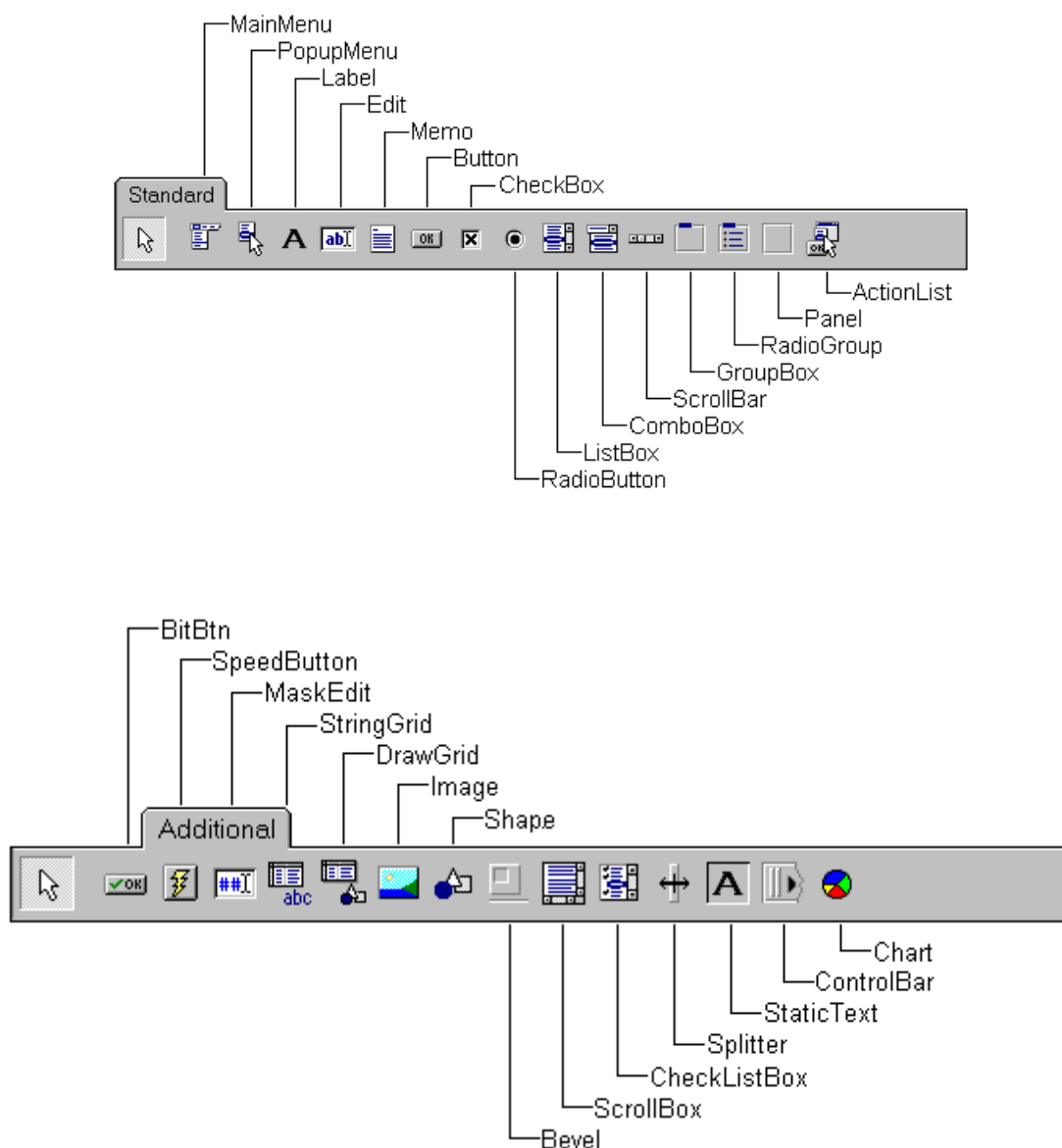
### ***PALETA DE COMPONENTES***

A Paleta de Componentes, possui todos os controles que iremos precisar para desenharmos nossa janela - formulário - como um programa de desenho livre. Para incluir um controle no formulário, existem dois métodos:

1 - Click Duplo no ícone da paleta de componentes. Insere o controle no centro do formulário com um tamanho padrão.

2 - Selecionar o ícone na caixa de ferramentas e depois dar um clique no formulário, na posição desejada para o objeto (canto superior esquerdo deste).

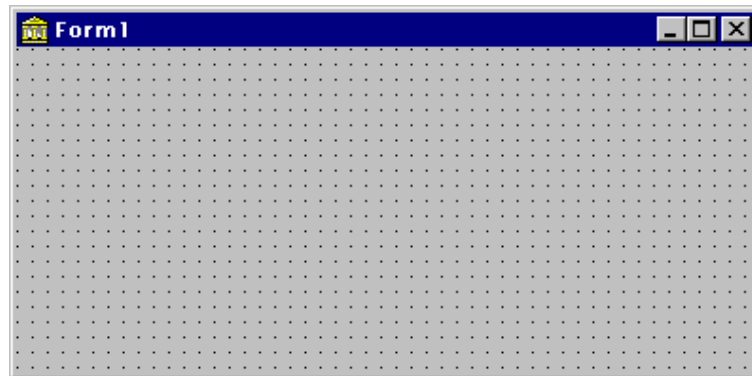
Podemos dimensionar estes controles, depois de inseridos, a qualquer momento durante o desenvolvimento. Primeiro, selecionamos o controle dando um clique em cima dele e depois o dimensionamos arrastando um dos oito botões dimensionadores que circundam este objeto.



## **JANELAS DO DELPHI**

### Formulário

É a janela que aparece no centro da tela do Delphi. Essa é a janela que estamos projetando. Quando rodarmos o aplicativo, será ela que irá aparecer com os objetos que nós incorporamos.



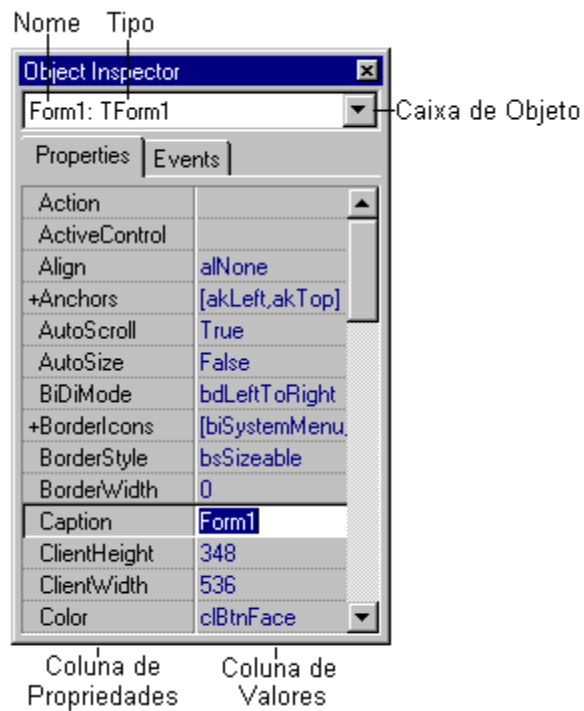
### Controles

Existem vários tipos de objetos no Delphi, entre eles, formulários e controles. Controles são todos os objetos que compõem um formulário. Um controle é qualquer objeto que o usuário possa manipular desde que não seja uma janela (formulário).

### Object Inspector

Nesta janela escolhemos como serão as características de cada objeto do formulário (botões de comando, quadros de texto, formulários, e outros), e a que eventos eles responderão, definindo como eles serão apresentados e operados pelo código. Cada objeto possui um conjunto específico de propriedades, com seus valores, associadas a eles. Ao trabalharmos nos diferentes objetos, a janela **Object Inspector** nos permitirá mudar as propriedades do objeto com o qual estamos trabalhando. Esta janela possui além da divisória **Properties** a divisória **Events** onde estão listados todos os eventos possíveis de ocorrerem com o objeto selecionado.

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



Na janela Object Inspector mostrada anteriormente, temos algumas das propriedades do formulário assim que iniciamos o Delphi.

**Caixa de Objeto** - Esta caixa mostra o objeto atualmente selecionado, através dela também poderemos selecionar todos os objetos do formulário ativo, alterando as suas propriedades, basta dar um clique na seta, que um menu de cortina se abre, onde poderemos selecionar um objeto.

**Nome** - Contém o nome do objeto atualmente selecionado, que será utilizado pelo código. Este nome está na propriedade Name.

**Tipo** - Nesta posição encontraremos qual é o tipo (classe) do objeto selecionado, se ele é um TForm (formulário), TButton (botão de comando), TLabel (legenda), ou então um TEdit (quadro de texto).

**Coluna de Propriedades** - Exibe todas as propriedades que podemos modificar em tempo de projeto.

**Coluna de Valores** - Exibe o valor da propriedade correspondente.



## MEU PRIMEIRO PROGRAMA

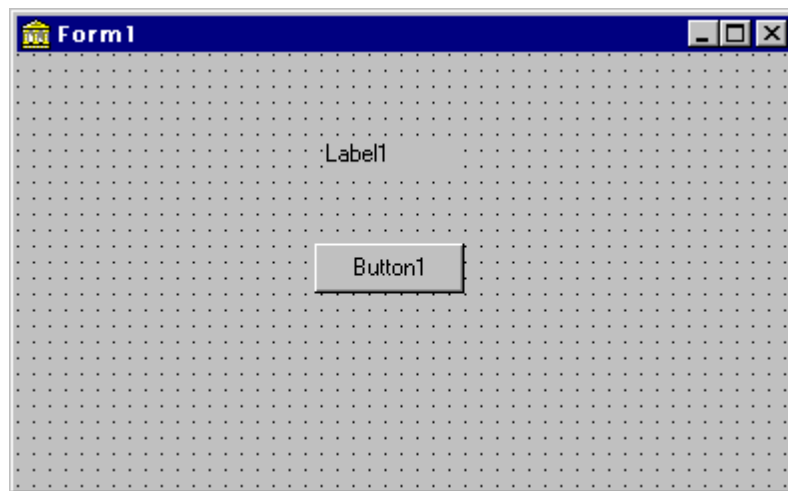
---

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

Existem três passos principais para a escrita de uma aplicação no Delphi que iremos seguir:

- **Desenhar as janelas que se deseja usar.**  
Inserir no formulário os controles que serão necessários
- **Adaptar as propriedades dos objetos.**  
Alterar as propriedades dos controles às necessidades da aplicação
- **Escrever o código para os eventos associados.**  
Esta é a parte mais complexa do desenvolvimento, é ela que dá a funcionalidade ao programa, são as rotinas que começam a ser executadas a partir de um evento.

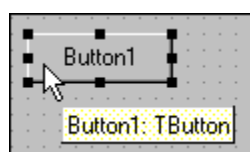
### Desenhar as janelas que se deseja usar.



1 - Começamos inserindo um **Label** (Legenda) e um **Botão de Comando** no formulário, de uma das duas maneiras:

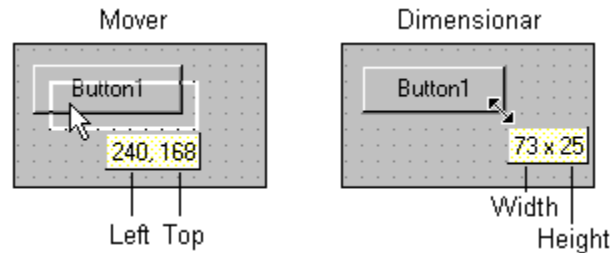
- a) Dando um duplo clique na barra de ferramentas no controle desejado;
- b) Selecionar o controle e dar um clique no formulário.

2 - Ao passar o ponteiro do mouse sobre o controle, nos será mostrado o seu nome e tipo.



3 - Arraste o Botão para as coordenadas (40,80) e o Label para (8,16).

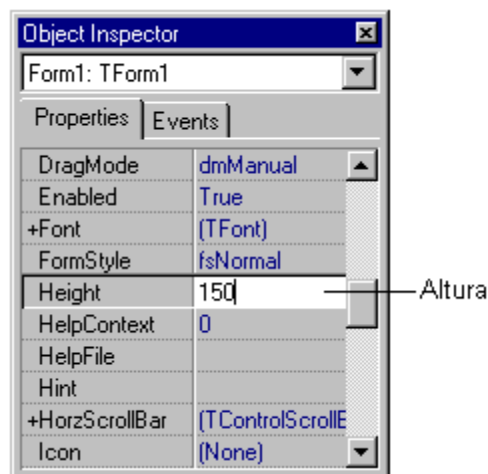
4 - Observe que, quando o controle estiver selecionado, poderemos arrastá-lo e dimensioná-lo dentro do formulário utilizando os botões dimensionadores. Enquanto movemos o controle, nos é mostrada a posição, e quando dimensionamos aparecem as dimensões.



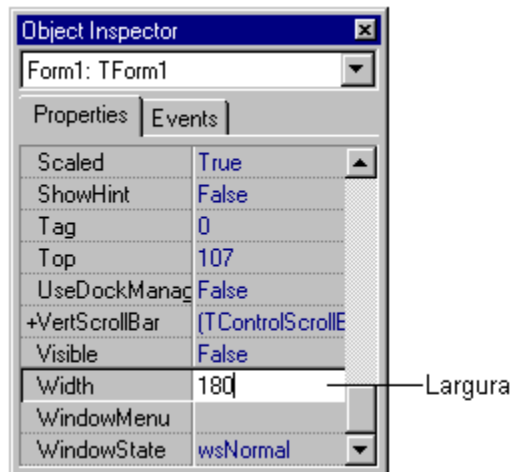
### ADAPTAR AS PROPRIEDADES DOS OBJETOS

Para alterar as propriedades de um objeto, ele tem que estar selecionado (com os oito pontos dimensionadores visíveis), depois procurar o nome da propriedade a ser alterada, na janela Object Inspector, e selecionar (no caso de valores padrão) o seu valor, ou então escrever um novo valor.

1 - Dimensione o formulário da seguinte maneira:  
Selecionar a propriedade **Height**, e atribuir a ela o valor de 150.



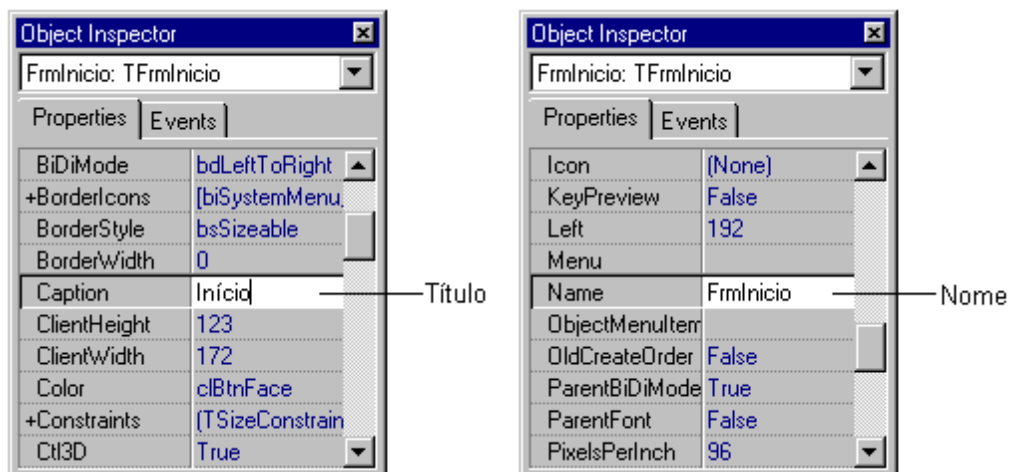
Selecionar a propriedade **Width** e dar o valor de 180.



Estes números correspondem a Pixels, que são os pontos do monitor. Sendo o tamanho do objeto no monitor dependente da resolução de vídeo utilizada pelo computador.

Altere também as propriedades **Name** e **Caption**. A propriedade Name é a identificação do Objeto no código da aplicação, pode ser de qualquer tamanho mas só são considerados os primeiros 255 caracteres, deve começar com uma letra ou uma sublinha (\_) não pode conter espaço, e após o primeiro caractere são permitidos números, letras (não portuguesas) ou sublinha.

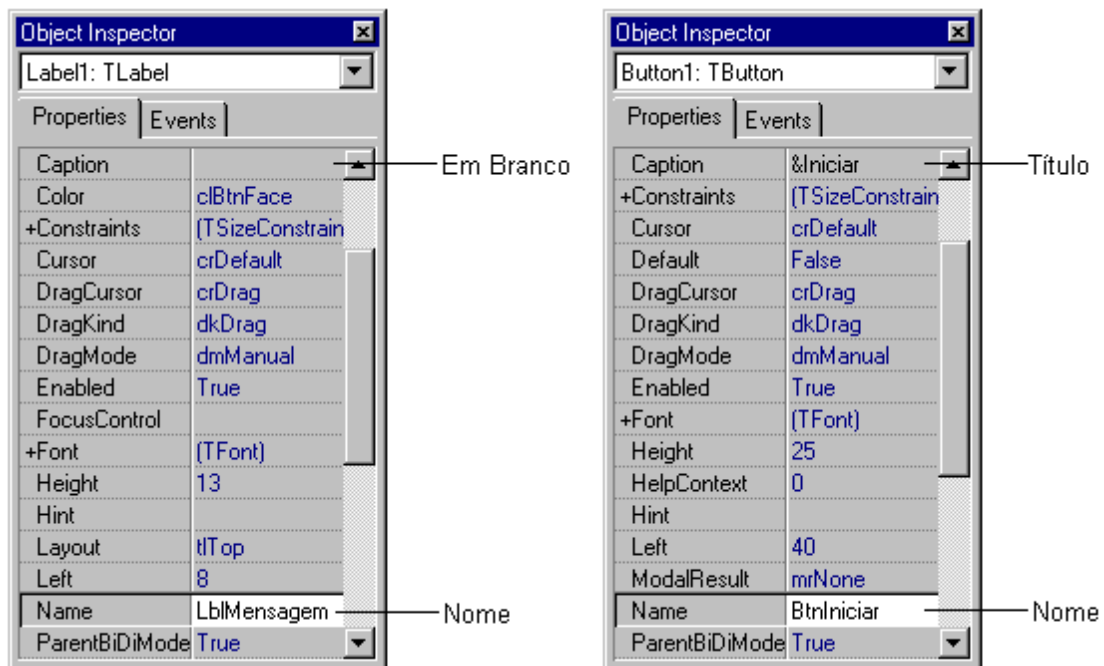
A propriedade Caption é a palavra que aparecerá como título da janela, com até 255 caracteres.



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



Agora, altere as propriedades do **TLabel** e do **TButton**. O caractere **&** quando utilizado na propriedade Caption de Botões irá sublinhar o caracter seguinte a ele, identificando-o como atalho para o Windows

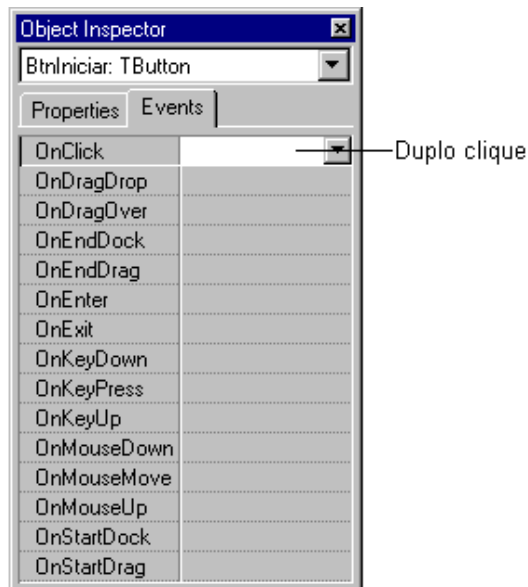
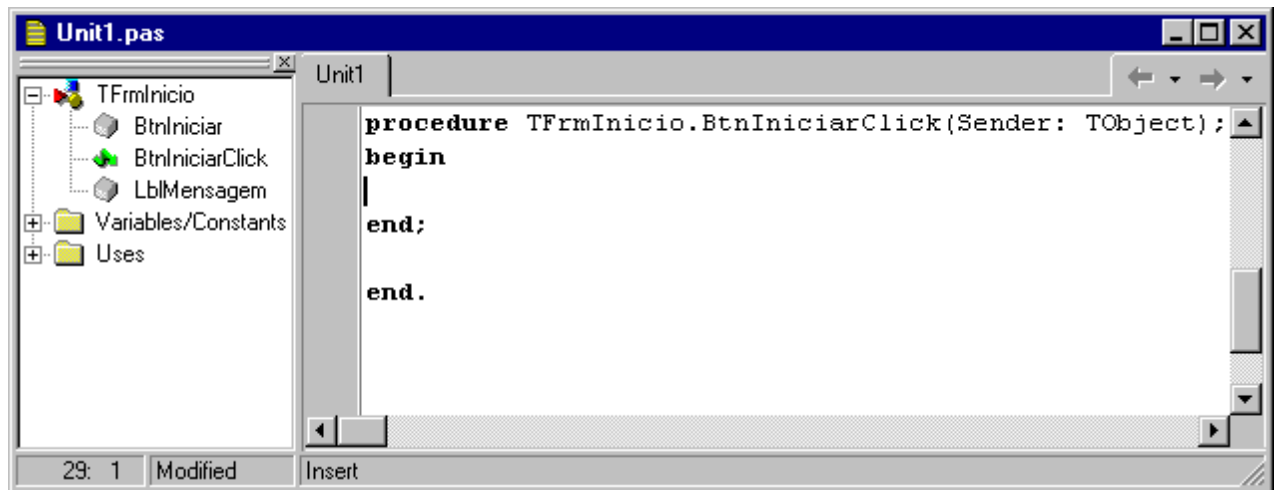


Após alterarmos a propriedade Caption do controle Label este ficará pequeno, pois a propriedade **AutoSize** é True, ou seja, o tamanho do Label estará sempre de acordo com o conteúdo de Caption.

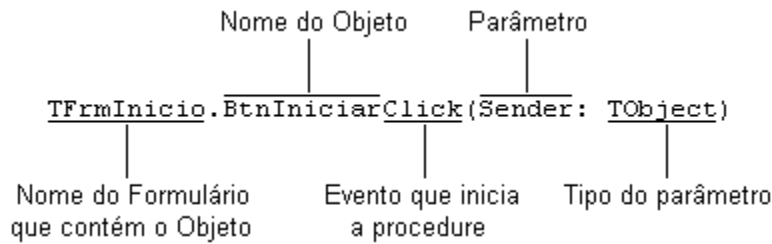


**ESCREVER O CÓDIGO PARA OS EVENTOS ASSOCIADOS.**

O código é escrito na janela **Code editor**, para acessá-la, selecione o botão **Iniciar** e na janela *Object Inspector*, chame a divisória *Events* dando um duplo clique na parte direita da linha que contém o evento **OnClick** - a rotina escrita para este evento, será executada quando o botão **Iniciar** for clicado.

Janela Code Editor

Nesta janela observamos o nome da procedure, identificando qual o objeto e o evento que dará início à execução do código, e onde está localizado este objeto. O parâmetro **Sender** informa ao Delphi qual componente recebeu o evento que chamou a execução deste procedimento.



Todas as instruções a serem executadas por um procedimento devem estar entre as palavras reservadas **Begin** e **End**.

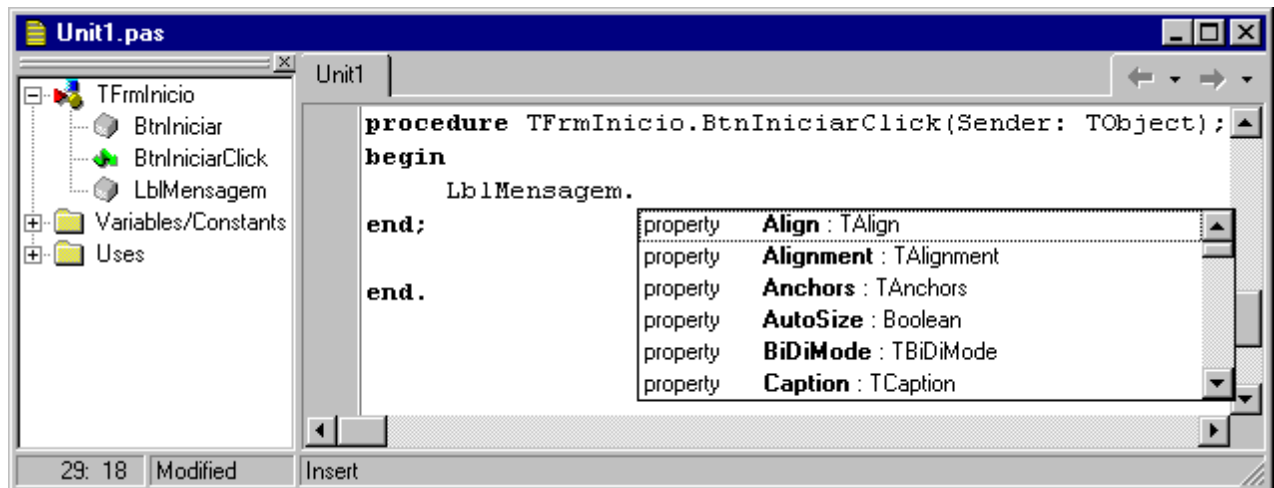
O **Code Editor** também pode ser acessado dando-se um duplo clique no objeto que se quer criar um código. Cada objeto tem um evento que é mais comumente utilizado, e é com este evento que o Delphi iniciará o Code Editor quando acessado desta forma, isto não impede que criemos outros códigos utilizando mais de um evento para o mesmo componente.

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

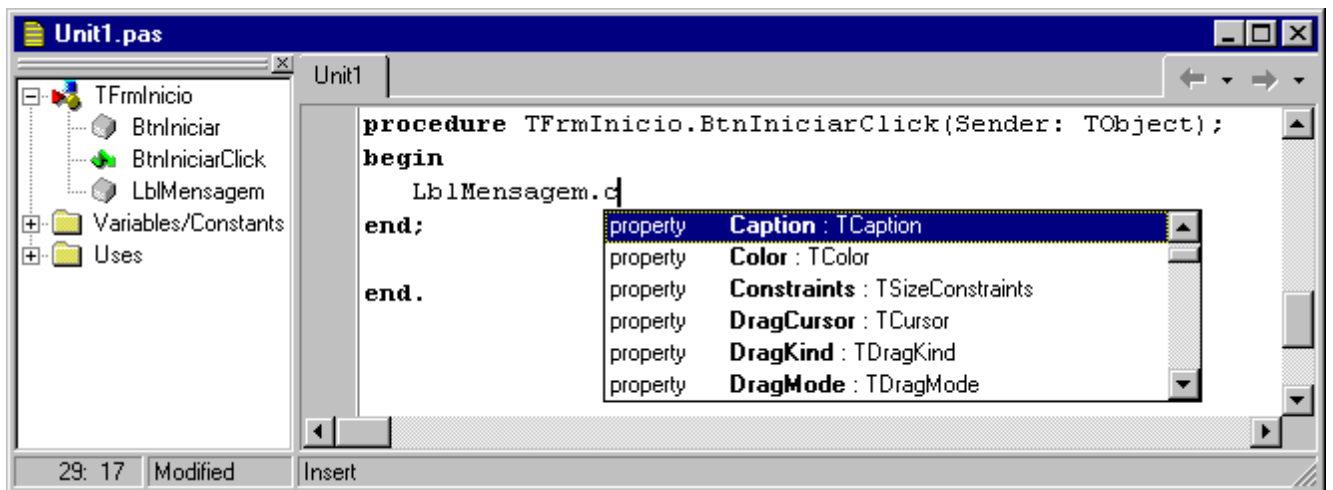
Atribuímos valores a uma propriedade de objeto seguindo o padrão:

objeto + . + propriedade + := + valor da propriedade ;

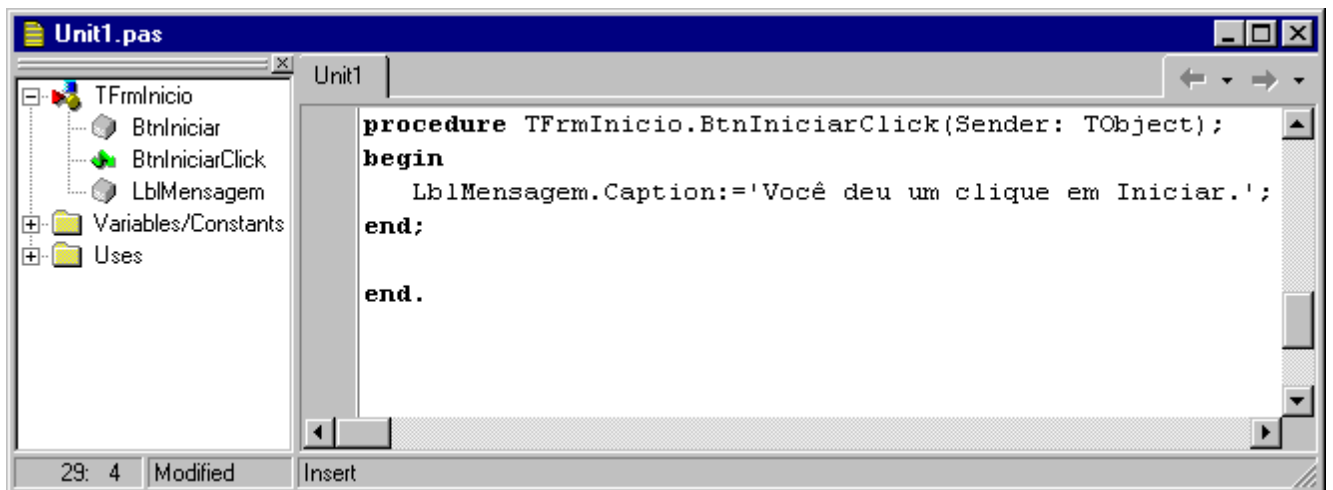
Abra o Code Editor para o botão de comando e digite o código conforme a figura a seguir. Repare que ao digitar o ponto após LblMensagem, e aguardar alguns instantes, o Delphi exibirá uma lista de propriedades e métodos do controle Label.




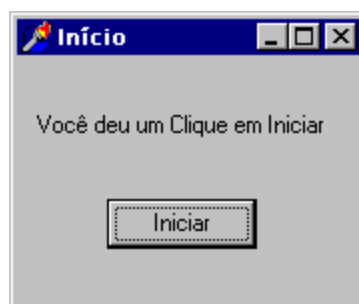
Para escolher uma propriedade do LblMensagem, selecione-a com as setas de direção e então pressione Enter, inserindo-a na linha de comando. Ou então, digite a primeira letra da propriedade, selecionando-a.





Continue com o código, seguindo a figura mostrada abaixo. Quando for dado um clique em Iniciar, será mostrada a mensagem “Você deu um clique em Iniciar”.



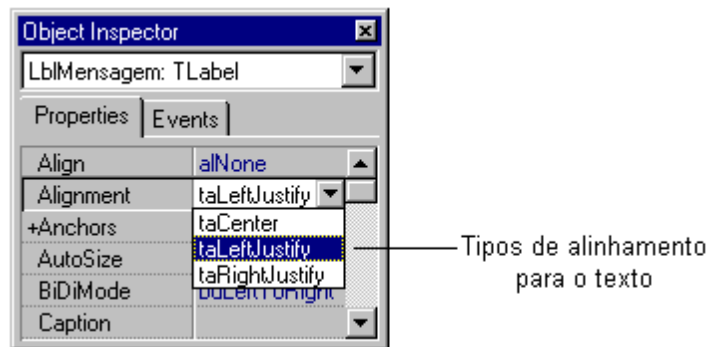
Clique sobre o botão **Run** da barra de ferramentas (  ) para que o Delphi compile e execute o projeto. Em seguida, selecione o botão Iniciar para ver o resultado.



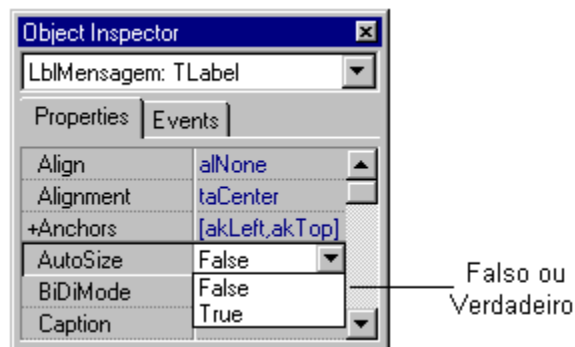
Finalize a execução do projeto teclando **Alt+F4** ou no botão Finalizar (  ) da barra de título da janela.

Para alternar a visualização entre o Formulário e o Code Editor, utilize o botão **Toggle Form/Unit** (  ) na barra de ferramentas ou a tecla **F12**.

Existem propriedades que possuem valores predefinidos, quando escolhemos a propriedade **Alignment** e damos um clique na seta da caixa de valor, aparecem os tipos de alinhamento para o texto dentro do componente.

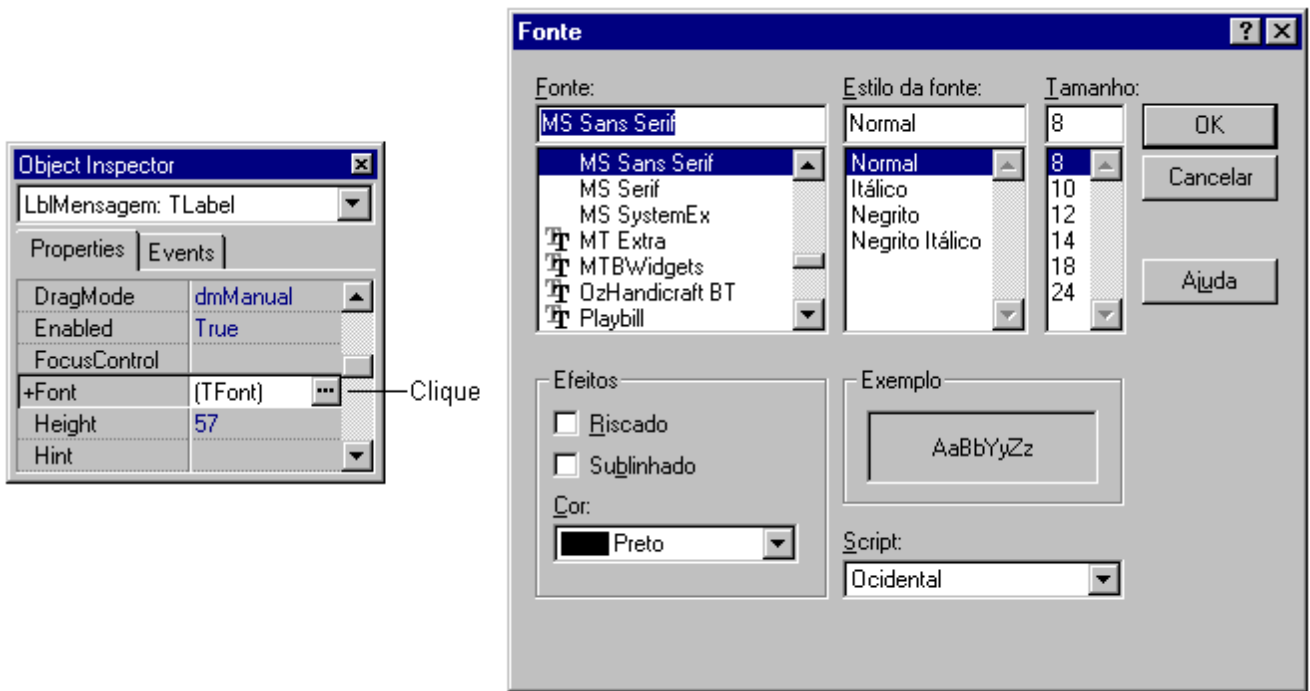


Selecione o objeto **TLabel** através da Caixa de Objeto da janela Object Inspector, e altere a propriedade **Alignment** para **taCenter**, para que o texto no TLabel fique centralizado. Altere também a propriedade **AutoSize** para **False**, e no Formulário aumente o tamanho do TLabel.



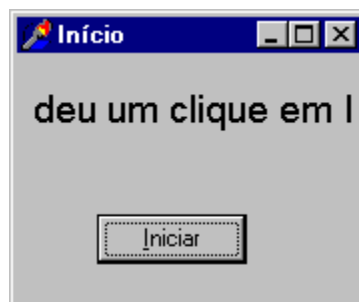
Além das propriedades descritas acima, com padrões pré-definidos, existem outras que possuem inúmeras escolhas, neste caso, ao invés de uma seta, observaremos três pontos, este é o caso da propriedade Font.



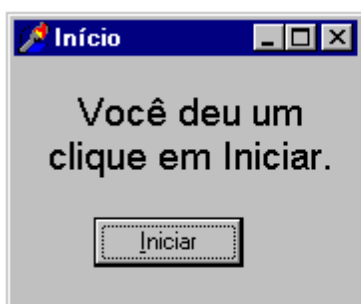


Quando selecionamos os três pontos, aparece um Quadro de Diálogo onde escolheremos o formato da fonte da mensagem apresentada.

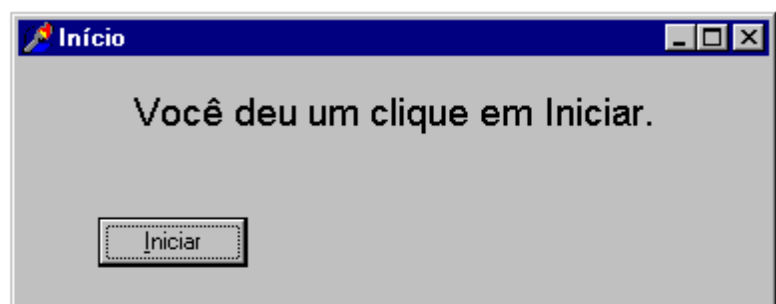
No seu projeto **Iniciar**, teste as alterações de fonte e observe as mudanças. Na figura a seguir, foi utilizada a fonte Arial com tamanho de 14 pontos.



Observe na figura acima que o texto não coube na área de exibição do Label e nem no Formulário, nós temos duas opções para que este texto apareça integralmente. A primeira, é alterar para True, a propriedade **WordWrap** do **Label**, esta propriedade insere uma mudança de linha quando o texto atinge a margem direita do objeto. A segunda, é redimensionar o Label e o Formulário. Como mostram as figuras a seguir.



WordWrap

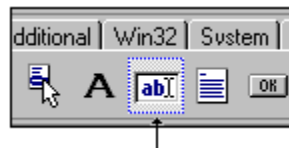


Redimensionar

## EXEMPLO I - CALCULADORA

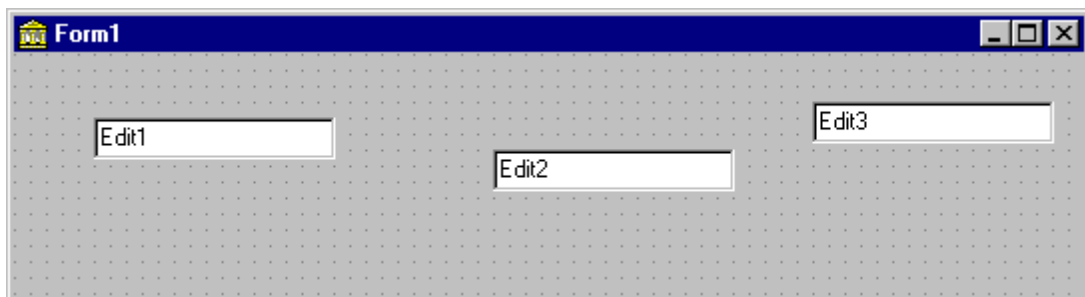
Para iniciar um novo projeto, escolha a opção **New Application** do menu **File**. Não salvando o projeto Iniciar anterior.

Para inserir vários objetos repetidos no Formulário, damos um clique no ícone do objeto escolhido enquanto pressionamos a tecla **Shift**, ele ficará conforme a figura abaixo.



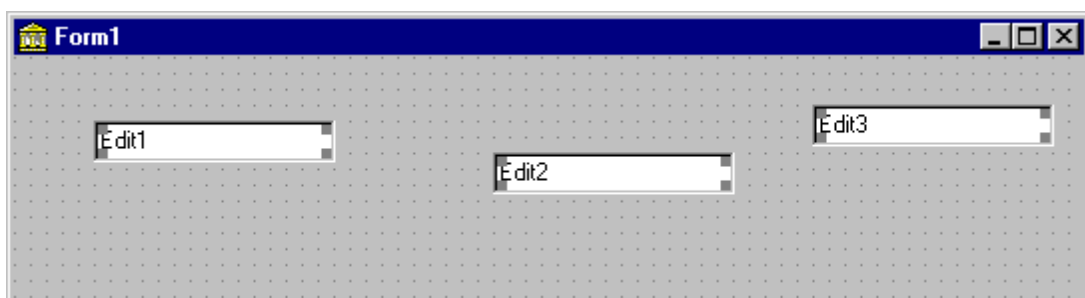
Dimensione e insira os controles **Edit**, utilizando a Paleta de Componentes, no formulário como o exemplo abaixo. Dimensionamos o formulário no Delphi da mesma forma que no Windows dimensionamos as janelas.

Logo após, basta ir inserindo os controles (Edit) dentro do Formulário, sem se preocupar com a estética. Após inserir todos os componentes repetidos, clique no cursor bitmap em forma de seta, isto desativará o botão selecionado anteriormente na Paleta de Componentes.

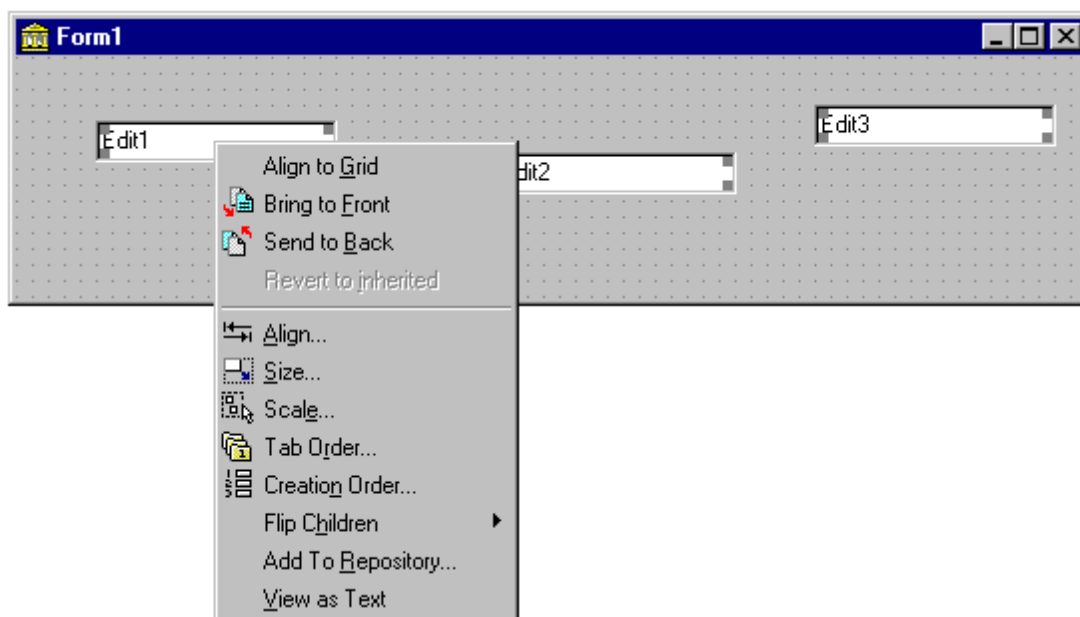


Observe que os objetos Edit estão desalinhados, o Delphi nos oferece um recurso para realizar rapidamente um alinhamento entre objetos.

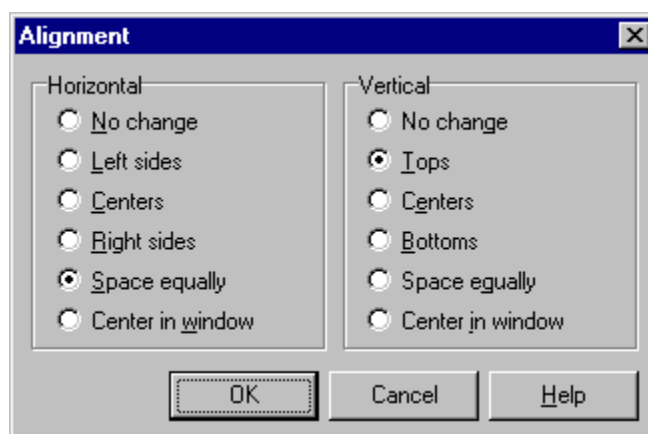
Primeiro deveremos selecionar os objetos que queremos alinhar. Pressione a tecla **Shift** enquanto você dá um clique em cada um dos Edit, selecionando todos ao mesmo tempo como mostra a figura a seguir.



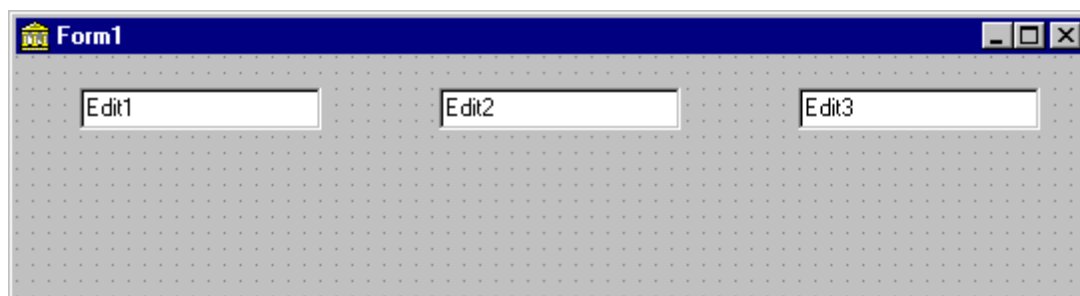
Depois disto, pressione o botão direito do mouse em cima de um Edit, para aparecer o pop-menu. Escolha **Align...**, aparecendo então a janela **Alignment**.



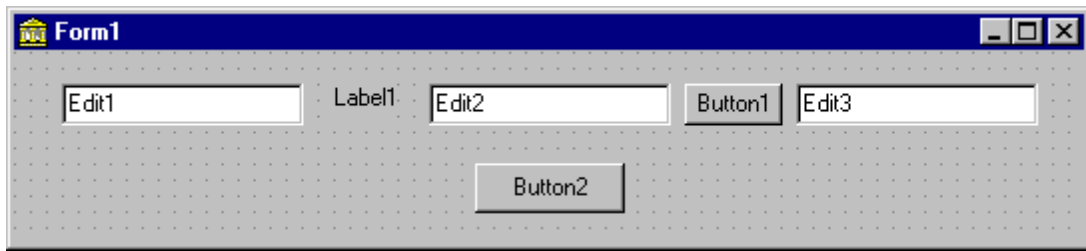
Escolha **Space equally** (igualmente espaçado) para alinhamento horizontal, e **Tops** (topo) para o alinhamento vertical.



Após o alinhamento, nosso Formulário estará conforme a figura abaixo:

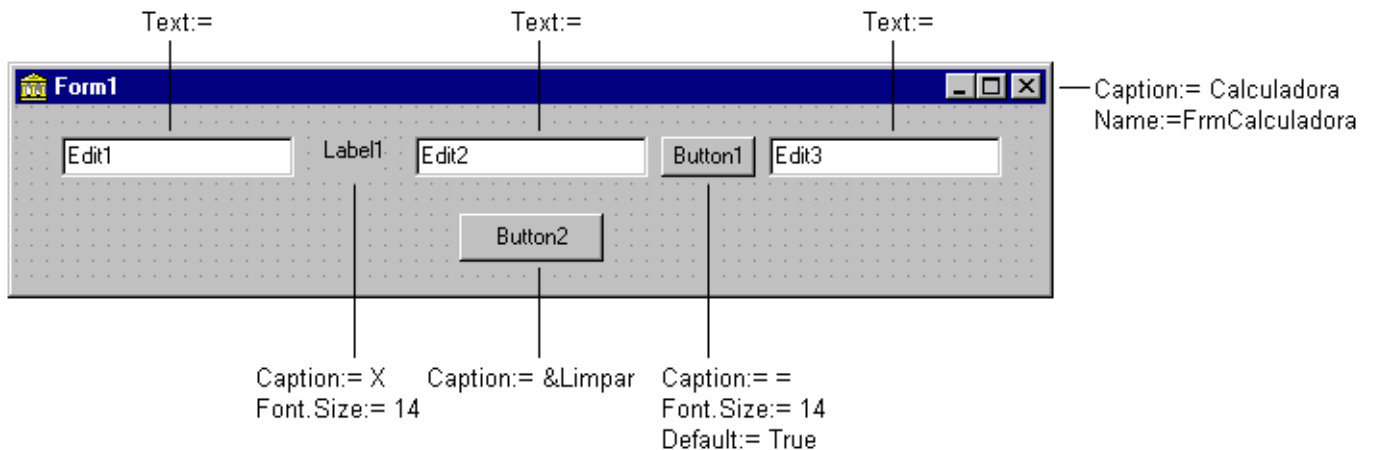


Insira os objetos restantes da maneira que preferir, posicionando-os de acordo com a figura a seguir:



Utilize o alinhamento **Horizontal - Center in Window** para o Button2.

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



Neste exemplo de projeto, digitaremos um número em Edit1, outro em Edit2, e quando for dado um Clique em Button1, o resultado da multiplicação aparecerá em Edit3. Para limpar os Quadros de texto (Edit), usaremos o Button2.

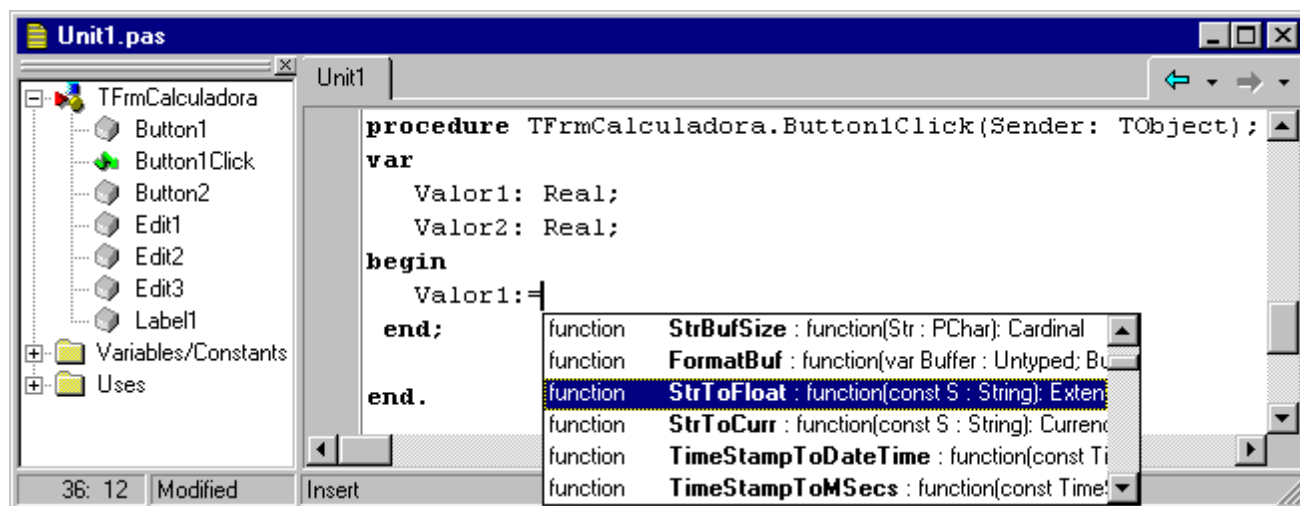
O projeto irá trabalhar basicamente com dois eventos :

- Clique em Button1 (=)
- Clique em Button2 (Limpar)

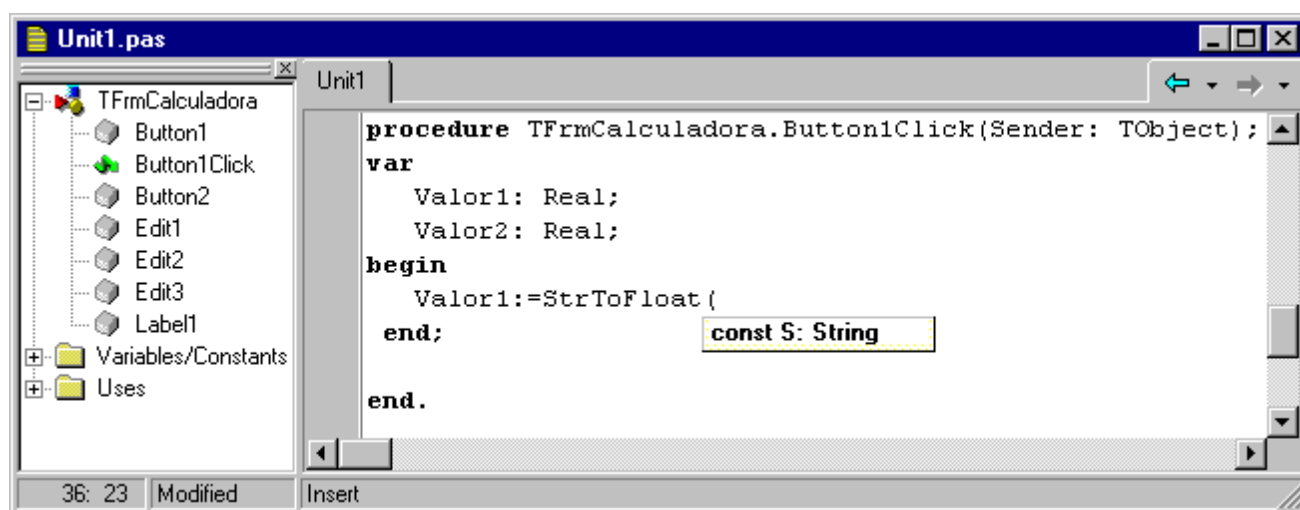
Então, para escrevermos o código, daremos dar um Duplo Clique no Button1 - o Code Editor será exibido.

O editor de código possui várias ferramentas que auxiliam o programador durante a digitação do programa. Uma delas é a **Code Completion** (conclusão de código), ela lista as propriedades, métodos e eventos apropriados à classe exibida.

Após digitar *Valor1:=*, pressione **Ctrl+Espace** para exibir a lista, procure a função **StrToFloat** (transforma uma string em número com ponto flutuante) em seguida pressione Enter.

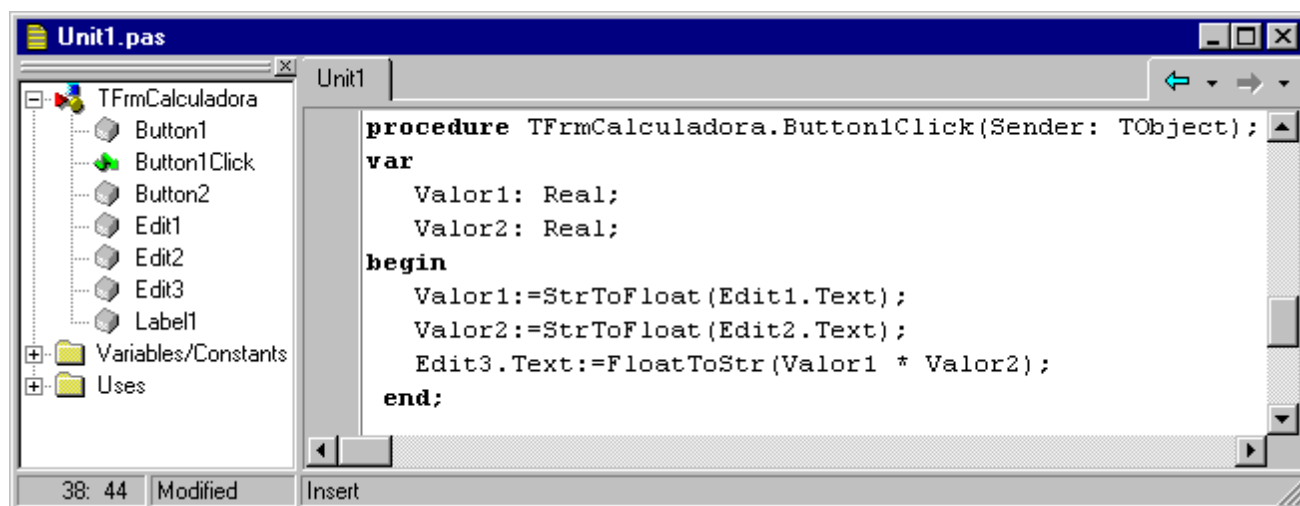


Outra ferramenta é o **Code Parameters** (parâmetros de código), que mostra uma lista dos parâmetros exigidos pela rotina ou função. Esta lista aparece logo após a abertura de parêntese.

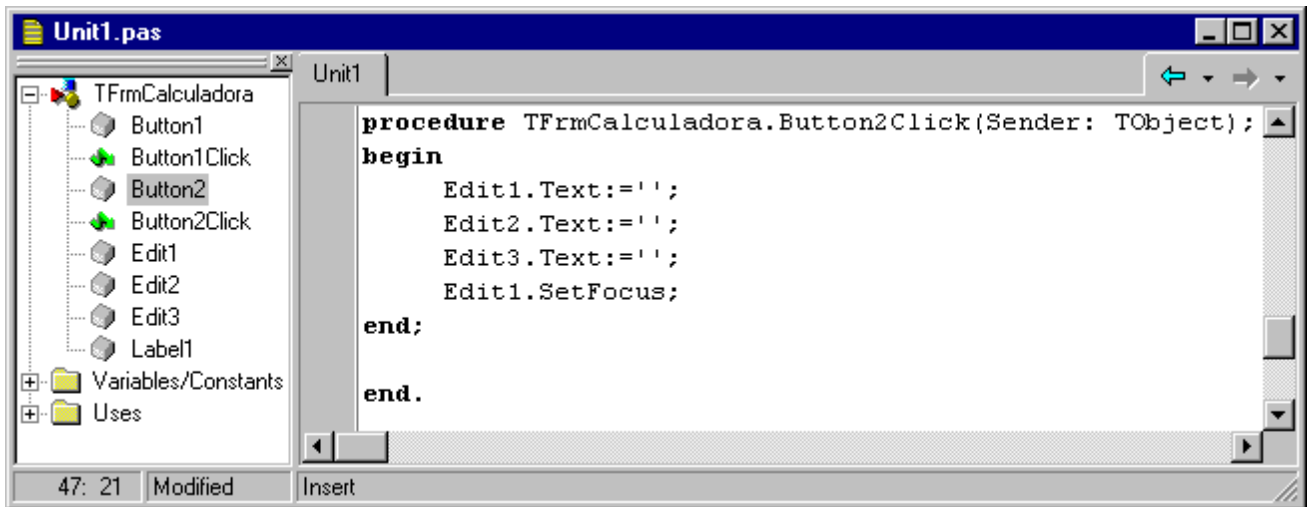


Na figura anterior, a função **StrToFloat** necessita de um parâmetro do tipo String (cadeia de caracteres), que no nosso exemplo está na propriedade Text do Edit1.

Digite o restante do código de acordo com a figura a seguir.



Altere para o evento `OnClick` do `Button2`. E entre com os comandos a seguir:




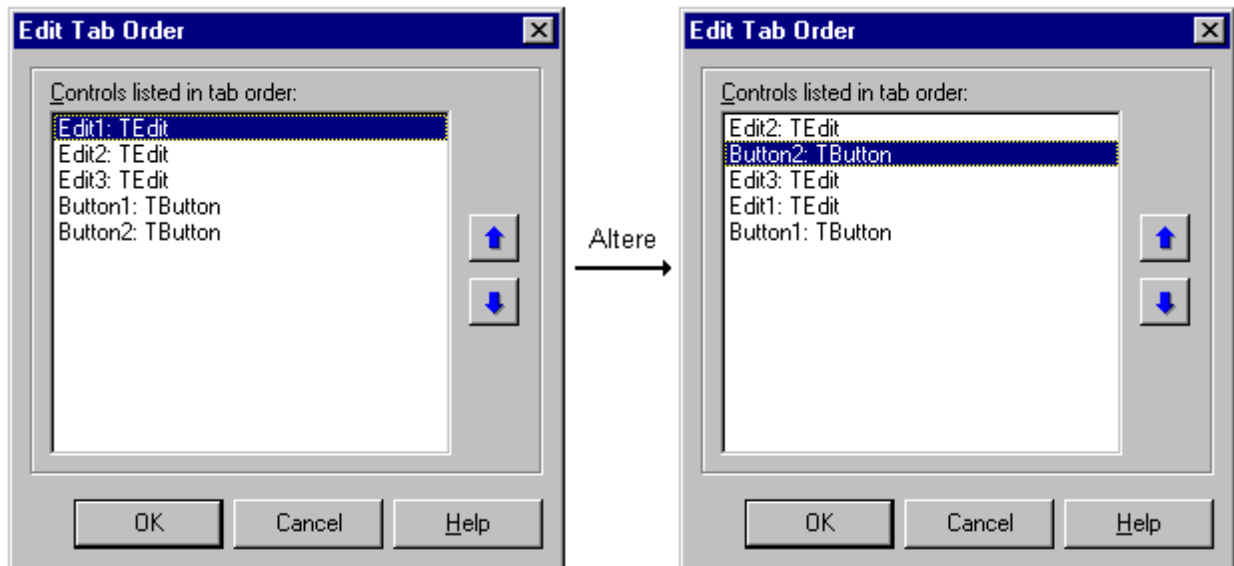
Execute o projeto. Para utilizá-lo, entre com um número em `Edit1`, outro em `Edit2` e dê um Clique em “=”, e o resultado da multiplicação aparecerá em `Edit3`.

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

Objeto	TabOrder
Edit1	0
Edit2	1
Edit3	2
Button1	3
Button2	4

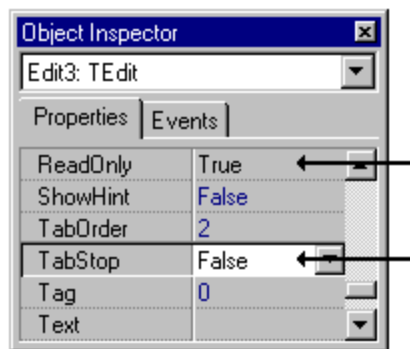
Para alterar esta propriedade basta selecionar o controle, e na janela `Object Inspector`, procure `TabOrder` e altere o seu valor, o Delphi não aceita controles com `TabOrder` de mesmo valor.

Poderemos também alterar a ordem de `Tab` acessando o popup-menu. Dê um clique com o botão direito em cima de qualquer componente escolhendo a opção `TabOrder` (  `Tab Order...`), abrindo o quadro de diálogo **Edit Tab Order**, onde poderemos alterar a ordem de todos os controles inseridos no formulário. Modifique a ordem de acordo com a figura a seguir:



Execute o projeto e observe as alterações.

O usuário pode alterar o valor de Edit3 mesmo após a multiplicação ter sido efetuada. Para evitarmos isso, defina as propriedades **TabStop:=False** e **ReadOnly:=True** para o Edit3 e verá que o usuário não terá mais acesso ao texto em Edit3.



Existem, nas aplicações para Windows, botões de comando que são acionados com a tecla Enter ou com um Clique. No nosso projeto este botão será o Button1, por isso, a propriedade **Default** foi selecionada para True. Fazendo aparecer um contorno mais espesso no botão, dando a indicação que se a tecla Enter for acionada, ocorrerá o evento OnClick no botão.

## PROPRIEDADES

### BorderStyle

Retorna ou dá um valor para o estilo de borda do objeto;

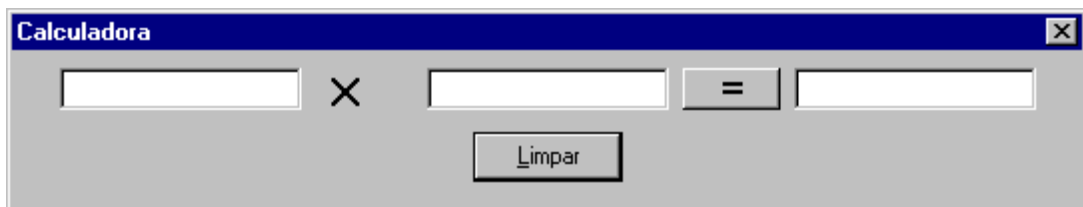
`objeto.BorderStyle := [valor]`



Existem 6 tipos de bordas:

1 - bsDialog

O formulário não possui os botões de maximizar e nem de minimizar. Não é redimensionável.



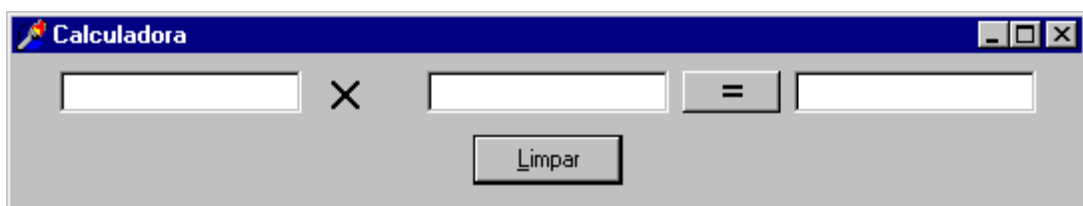
2 - bsNone

Nenhuma borda



3 - bsSingle

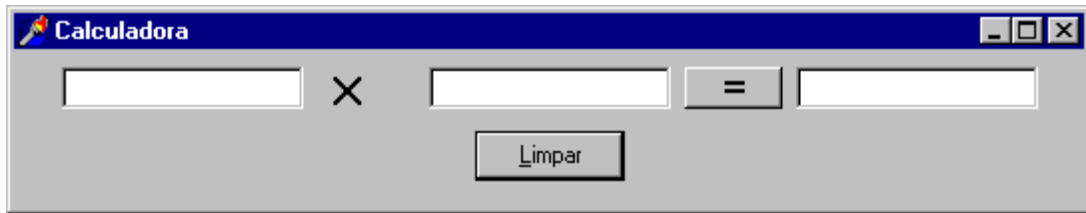
Fixa Simples, o formulário só é dimensionável através dos botões de minimizar e maximizar.





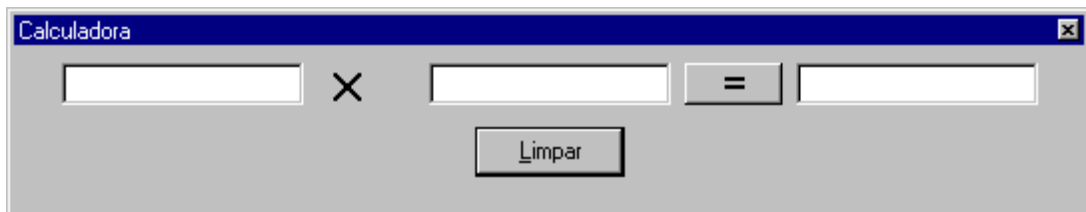
4 - bsSizeable

Borda Redimensionável



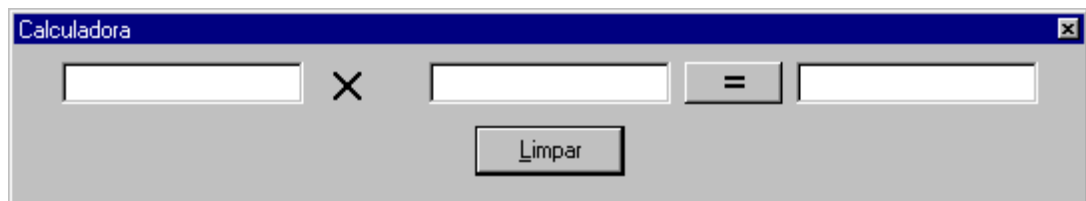
5 - bsSizeToolWin

Semelhante à bsToolWindow, mas é dimensionável.



6 - bsToolWindow

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



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

## Default

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

object.Default :[= booleano]

Default=     False  
              True

Quando esta propriedade de um TButton estiver como True o Delphi chamará o evento Click sempre que a tecla Enter for pressionada.

Ex:     Desejo que o botão BtnMultiplicar seja o default:

```
btnMultiplicar.Default := True;
```

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

```
Estado := btnMultiplicar.Default;
```

## **Tabstop**

Retorna ou dá o valor ao objeto indicado;

```
objeto.TabStop: [= booleano]
```

```
TabStop =   False  
           True
```

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

```
btnMultiplicar.TabStop := 0;  
btnDividir.TabStop := 1;  
btnSomar.TabStop := 2;  
btnSubtrair.TabStop := 3;
```

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

```
Ordem_Tab:= editNum1.TabStop;
```

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

## **Name**

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

```
objeto.Name
```

Ex: Desejo exibir o Name do Formulário em um Edit:

```
editNome_Form.Text := frmCalculadora.Name;
```

## **Caption**

objeto.Caption :[= string]

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

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

btnLimpar.Caption := 'Iniciar'

## **Text**

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

objeto.Text := [string];

## **MÉTODO**

### **Setfocus**

Dá o foco ao objeto indicado;

objeto.SetFocus

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

## **VARIÁVEIS NO DELPHI**

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

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

<b>Tipos Inteiros</b>	<b>Número de Bytes</b>	<b>Faixa</b>
ShortInt	1	-128 a 127
Integer	2	-32768 a 32767
LongInt	4	-2147483648 a 2147483647
Byte	1	0 a 255
Word	2	0 a 65535
<b>Tipos Booleanos</b>		
Boolean	1	1 byte booleano
ByteBool	1	Byte - sized Booleano
WordBool	2	Word - sized Booleano
LongBool	4	Double - word - sized Booleano
<b>Tipos Reais</b>		
Real	6	$2,9 \cdot 10^{-39}$ a $1,7 \cdot 10^{38}$
Single	4	$1,5 \cdot 10^{-45}$ a $3,4 \cdot 10^{38}$
Double	8	$5 \cdot 10^{-324}$ a $1,7 \cdot 10^{308}$
Extended	10	$3,4 \cdot 10^{-4932}$ a $1,1 \cdot 10^{4932}$
Comp	8	$-2^{63+1}$ a $2^{63-1}$

## Formas de Declarar uma Variável

As variáveis são declaradas usando-se a palavra reservada **Var**, o nome da variável, dois pontos e o tipo de dado a ser armazenado nesta variável.

**var**

Valor1: Real;

Elas podem ser declaradas em três locais diferentes, conforme a sua abrangência:

1 - Variável Local; ela será utilizada somente pelo procedimento onde está declarada, terminado o procedimento ela desaparecerá da memória. É declarada logo após o cabeçalho do procedimento.

2 - Variável a nível de Unidade (Unit); a variável será utilizada por todos os procedimentos e funções da unidade. É declarada logo após a palavra reservada **implementation**.

3 - Variável a nível de projeto; é a variável que poderá ser utilizada por toda a aplicação, ou seja, poderá ser utilizada por outras unidades. É declarada na seção **interface** da unidade.

**FORMATAÇÃO DE NÚMEROS**

A função **FloatToStr**, transforma um número em texto, mas não padroniza a sua apresentação. Caso necessitemos formatar um número a ser exibido, usaremos a função;

**FormatFloat**(formato , expressão), onde:

- formato = a maneira como deverá ser mostrada a expressão.
- expressão = expressão numérica a ser formatada.

Formatando números:

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

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

**Formatando Data e Hora:**

Para formatar data e hora usamos a função **FormatDateTime**:

**FormatDateTime** (formato , data);

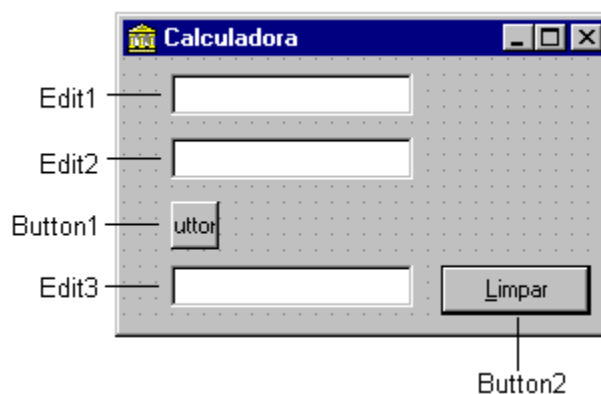
Formato	Exibido
d/m/yy	10/5/98
dd-mm-yyyy	18-Ago-1998
dd-ddd	02-dom
hh:mm AM/PM	08:50 AM
h:mm:ss a/p	8:50:20 a
d/m/yy h:mm	03/07/98 9:30

## MODIFICANDO A CALCULADORA

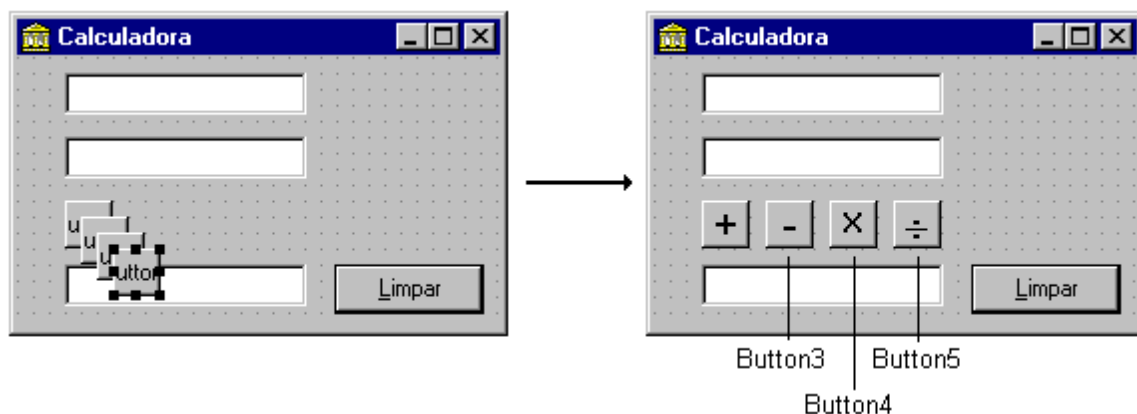
No Formulário da calculadora, selecione o botão de comando **Button1** e pressione a tecla **Delete**. O botão de igual desaparecerá do formulário, mas o seu código continuará no editor de código. Selecione o editor de código, e observe que a procedure `TfrmCalculadora.Button1Click` continua no mesmo lugar.

Modifique o formulário `TfrmCalculadora` como o exemplo a seguir, inserindo um novo botão:

Na figura aparecem as propriedades Name de cada objeto



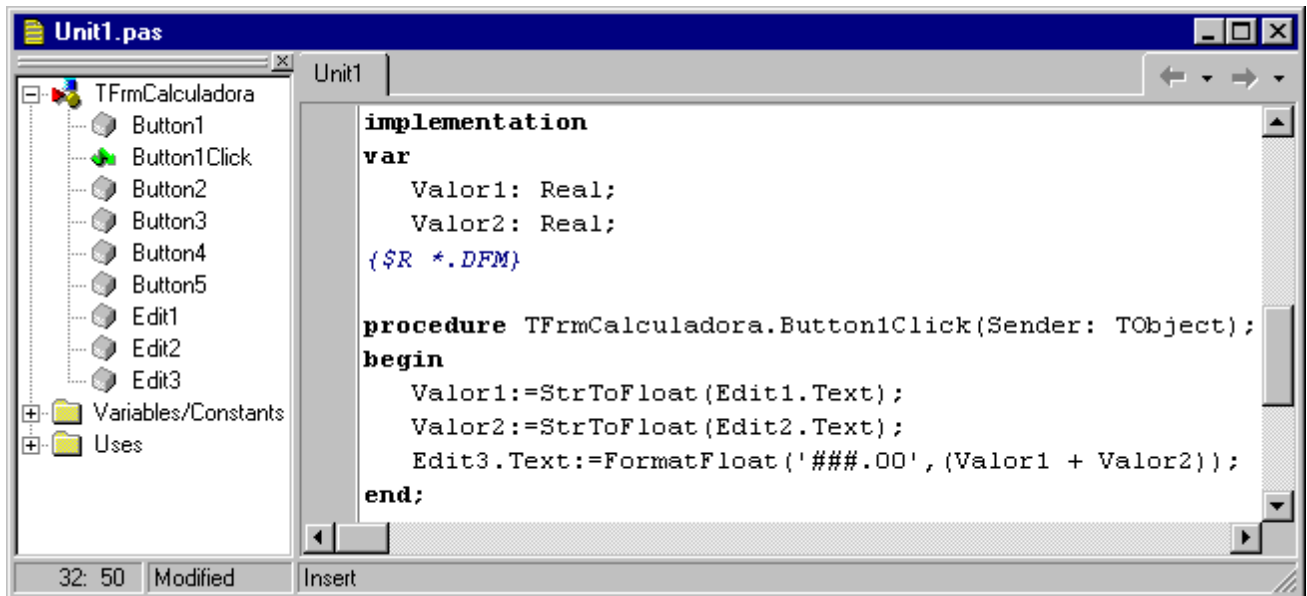
Selecione o **Button1**, copie (Ctrl+C) e cole mais três botões (Ctrl+V). Depois posicione-os de acordo com a figura:



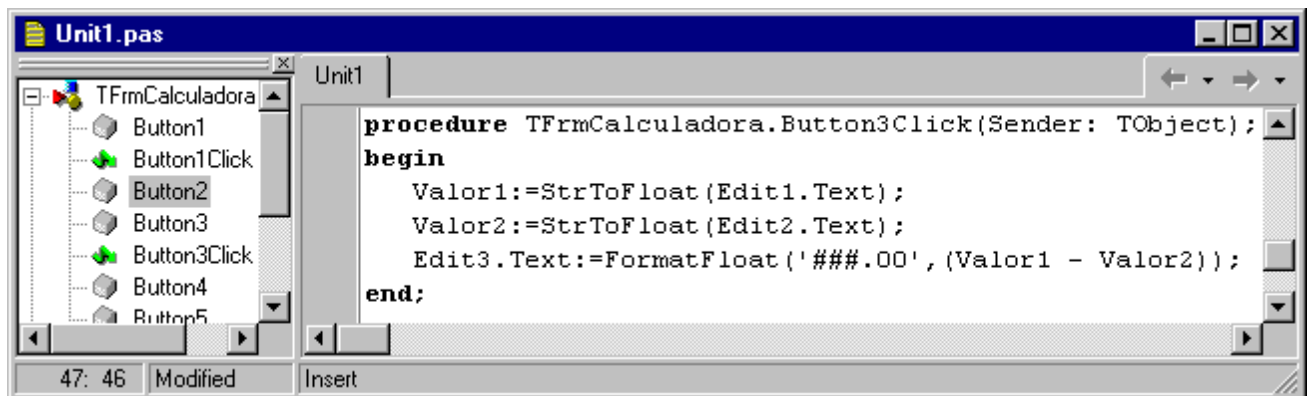
Para que o **Button5** exiba o símbolo correto da divisão, primeiro altere a sua fonte para Symbol. Depois, abra o Mapa de caracteres do Windows e procure pelo símbolo da divisão na fonte Symbol, e então copie e cole para a propriedade Caption deste botão. Provavelmente o caractere que aparecerá na caixa de propriedade não será o mesmo do botão, pois a fonte da Object Inspector não é Symbol, mas não se preocupe com este problema.

Chame o procedimento para o **Button1** dando um duplo clique no botão de comando. Note que antes este procedimento executava uma multiplicação, agora deverá executar uma soma. Usaremos também a função `FormatFloat` para formatar a apresentação do resultado.

As duas variáveis **Valor1** e **Valor2** que antes pertenciam a apenas um procedimento, agora deverão ser utilizadas pelos procedimentos das outras operações. Para que isso ocorra, retire-as do procedimento `TfrmCalculadora.Button1Click` e declare-as na seção `Implementation` da Unidade.



Substitua o tipo de operação em **Button1Click**, e nos demais botões, utilizando as ferramentas de Copiar e Colar.



Teste os vários formatos de apresentação dos números, alterando os parâmetros de apresentação da função **FormatFloat**.


Um projeto em Delphi trabalha com vários arquivos. Um arquivo para cada Formulário, outro para Unidade e um arquivo para o Projeto. Os arquivos de Unidades possuem a extensão **.PAS**, o arquivo do Projeto **.DPR** e o do Formulário, **.DFM**. E outros arquivos de apoio que são:

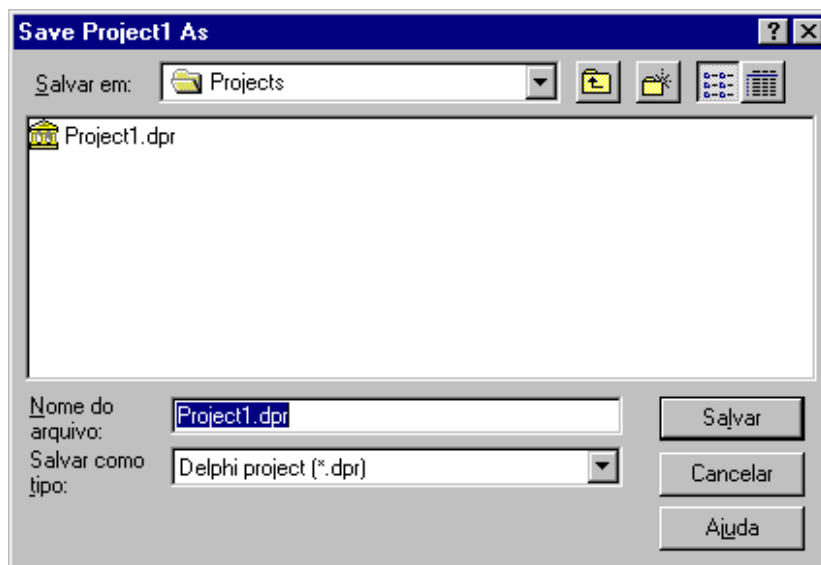
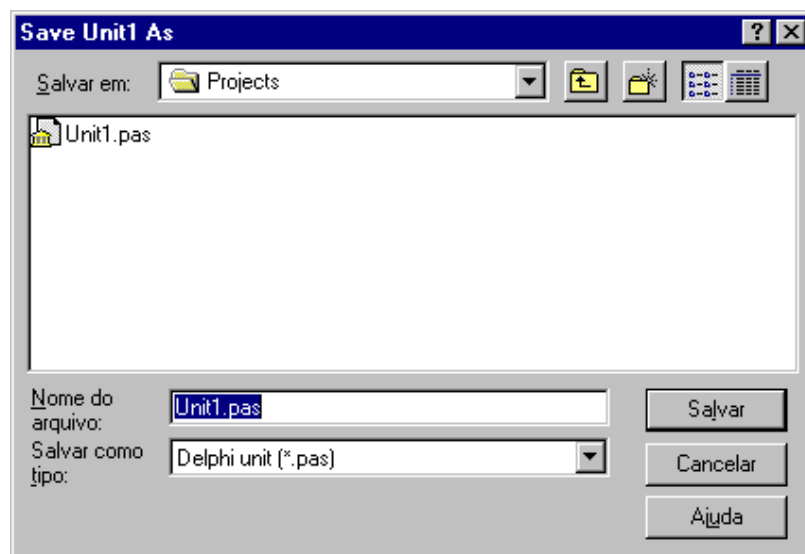
- .CFG** - armazena as definições de configuração do projeto.
- .DOF** - contém o compilador e as definições de linker, diretórios de procura, informação de versão, e assim sucessivamente.

**.RES** - contém os recursos do projeto como o ícone de aplicação, bitmaps e cursores.

Quando salvamos nosso projeto, o Delphi solicita apenas os nomes dos arquivos de Unidade e Projeto, os outros ele cria automaticamente.

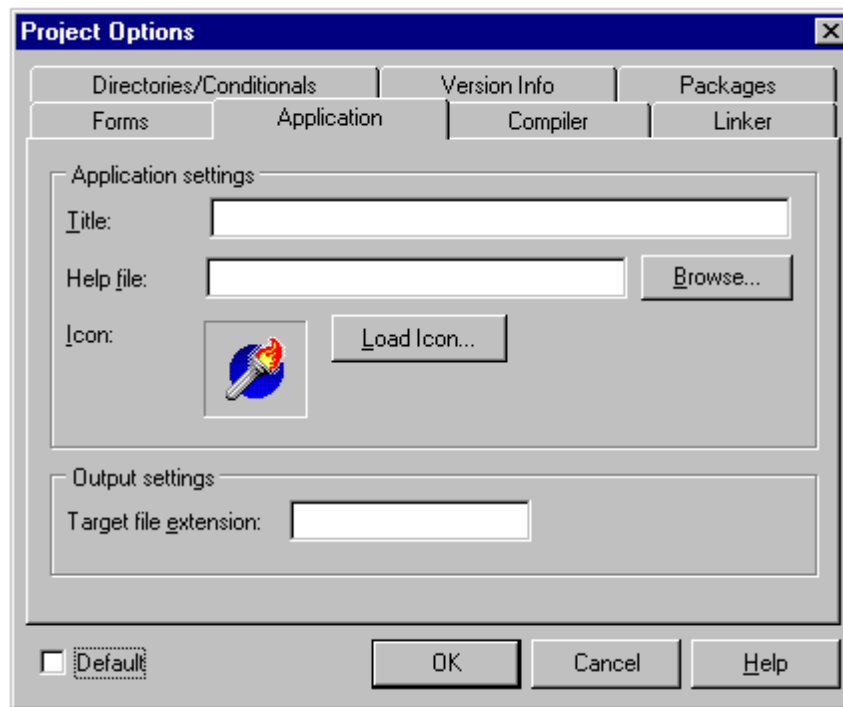
*Vamos salvar o nosso projeto Calculadora.*

No menu, selecione **File / Saveall** ou  e aparecerá o quadro de diálogo de salvar do Windows, pedindo para dar um nome ao arquivo do Formulário, extensão **.pas**, dê o nome de **CalculadoraUnt.pas** e clique em **Salvar**. A seguir, aparecerá o mesmo quadro pedindo para dar um nome ao arquivo de projeto, extensão **.dpr**, dê o nome de **Calculadora.dpr** e clique em **Salvar**. Os nomes dos arquivos da Unidade e do Projeto deverão ser diferentes, apesar da extensão já o ser.

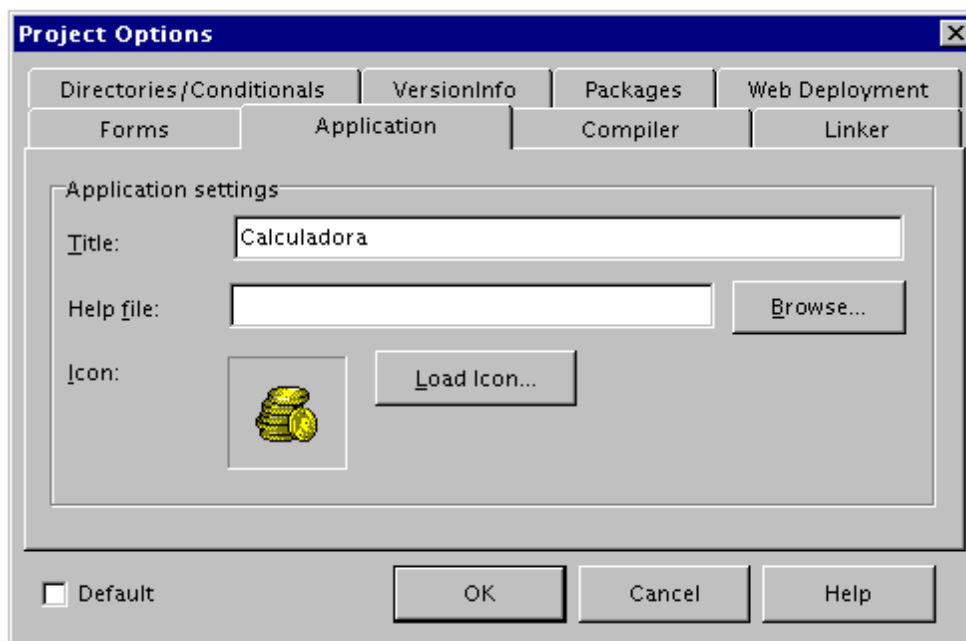



O nosso projeto está salvo. Agora, precisamos escolher um ícone que o representará na tela do Windows. No menu principal do Delphi, selecione **Project / Options...**, e escolha a página **Application**:






Clique no botão **L**oad **I**con... , e procure o ícone **Finance.ico**, no diretório **\\Borland Shared\\Images\\Icons**, escolha **A**brir para voltar à janela *Project Options*. Dê um título ao programa e pronto, o programa já possui um ícone e um título associados.



Compile o projeto utilizando a opção Compile Calculadora do menu Project (ícone  Compile Calculadora-Ctrl+F9), gerando um novo arquivo executável contendo as últimas alterações realizadas sem executar o projeto. Agora, você tem um programa executável em qualquer microcomputador que possua o sistema Windows, sem necessariamente ter o DELPHI ou de suas bibliotecas instalados.


Verifique no Windows Explorer todos os arquivos que compõem este projeto.

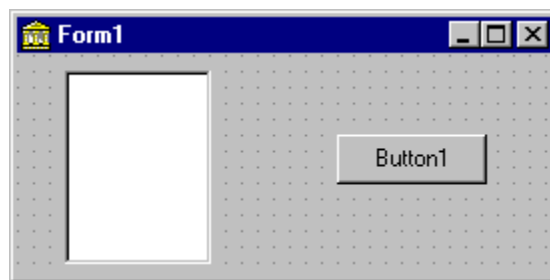
Nome	Tamanho	Tipo	Modificado
 Calculadora.cfg	1KB	CFG Arquivo	01/08/98 11:01
 Calculadora.dof	1KB	DOF Arquivo	01/08/98 11:01
 Calculadora.dpr	1KB	Delphi Project	01/08/98 11:01
 Calculadora.exe	287KB	Aplicativo	01/08/98 11:22
 Calculadora.res	1KB	RES Arquivo	01/08/98 10:24
 CalculadoraUnt.dcu	5KB	Delphi Compiled ...	01/08/98 11:22
 CalculadoraUnt.dfm	2KB	Delphi Form	01/08/98 10:54
 CalculadoraUnt.pas	2KB	Delphi Unit	01/08/98 10:58

## DEPURAÇÃO

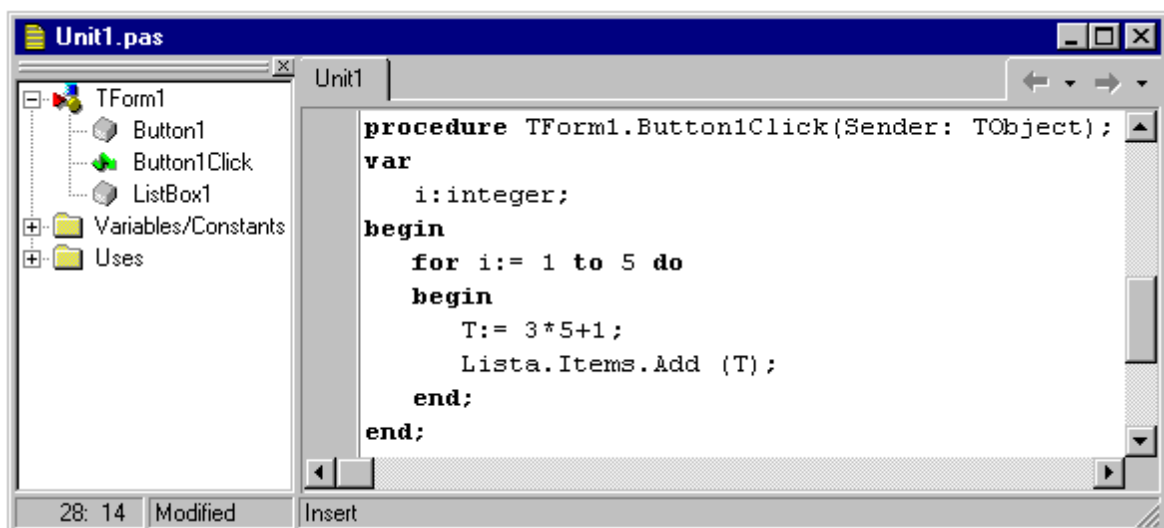
Quando construímos uma aplicação, é possível errarmos na sintaxe durante a digitação, fornecendo parâmetros errados ou trocando nomes de variáveis, ou então, criarmos erros lógicos que são detectados como saídas incorretas do programa. Para resolver estes problemas o Delphi possui várias ferramentas que controlam a execução do programa ajudando-nos a resolver tais erros.

Quando o Delphi compila o programa (Ctrl+F9) e encontra algum erro de sintaxe ele mostra a linha onde está o erro, dando uma dica do problema, na parte inferior da janela do **Code Editor**.

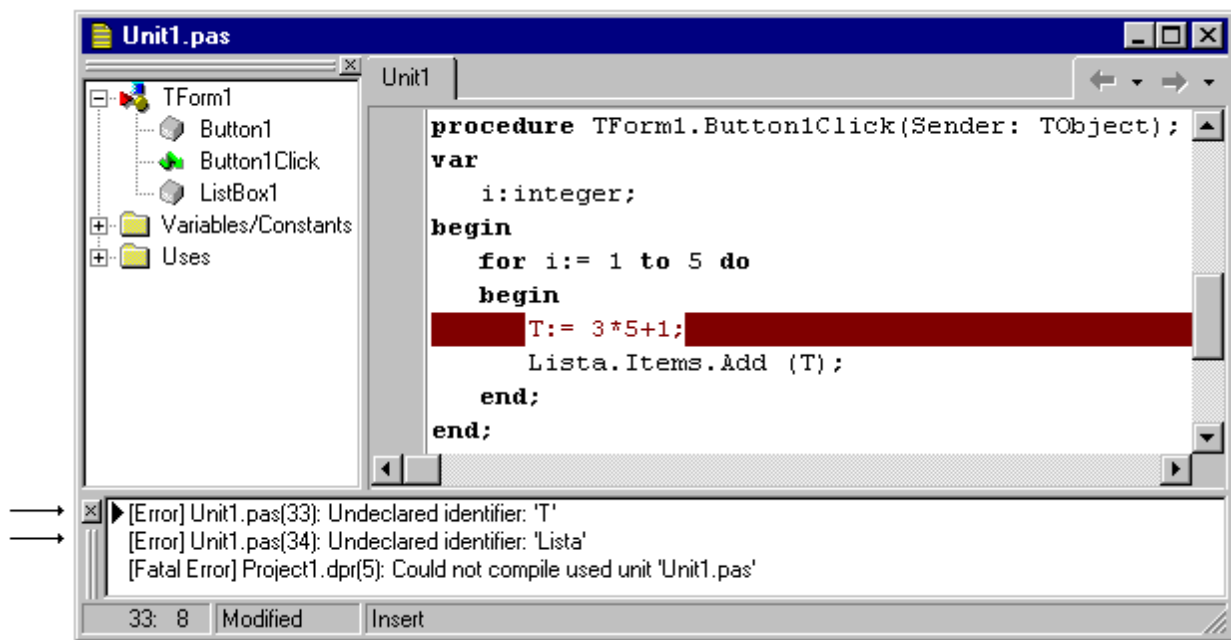
Para praticarmos a depuração de programas, vamos construir um aplicativo que nos dê como saída uma lista com os seguintes números: 18, 21, 24, 27 e 30. Construa um formulário com um botão e um quadro de lista - ListBox (  ), sem alterar nenhuma propriedade dos componentes:



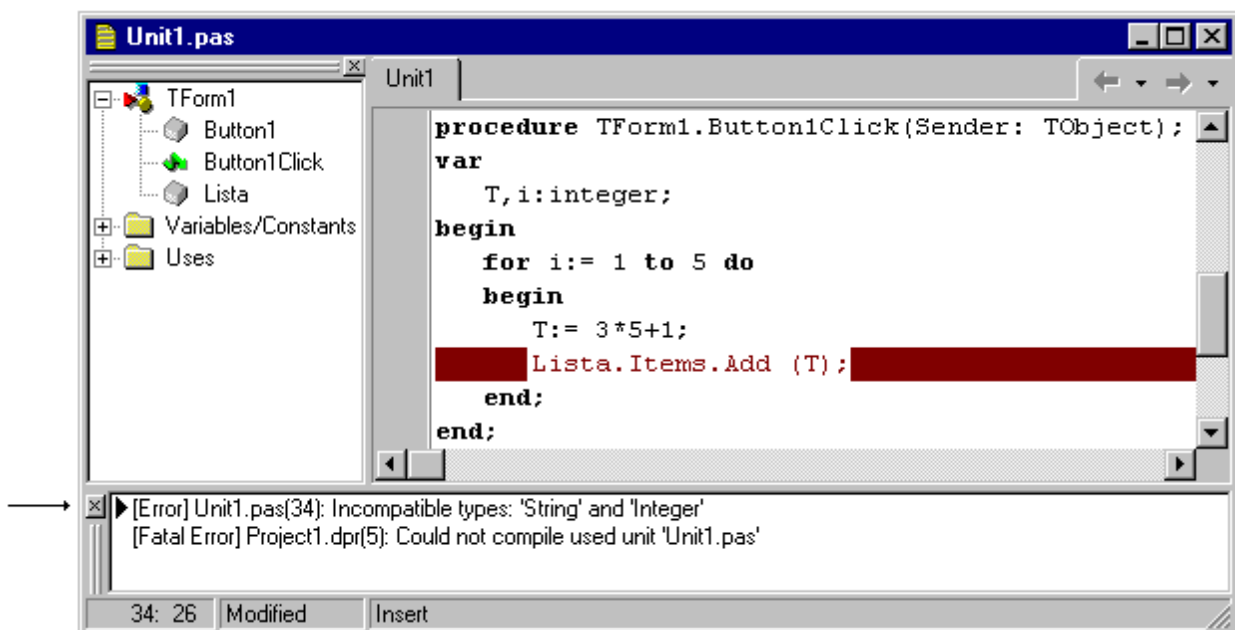
Entre com o seguinte código, exatamente como está, para o botão:



Compile o programa pressionando Ctrl+F9, e observe os erros de sintaxe encontrados pelo Delphi. São erros que indicam a não declaração ou existência da variável T e do objeto Lista. A linha destacada em vermelho indica aonde o compilador parou.



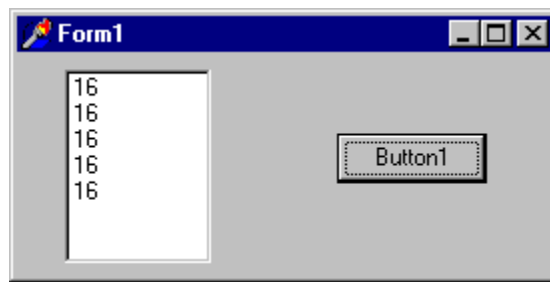
Declare a variável T como do tipo inteiro semelhante à i e altere a propriedade **Name** da **ListBox** para **Lista**, e compile novamente.



Agora nos foi apresentado um erro onde o tipo de dado requerido é um (**String**) mas o fornecido é outro (**Integer**). Para corrigir isto devemos transformar do conteúdo da variável T em **String** utilizando a função **IntToStr()** na seguinte linha.

- Lista.Items.Add (IntToStr(T));

Após esta correção, o compilador não detectará mais nenhum erro de sintaxe. Portanto, execute este exemplo pressionando **F9**.

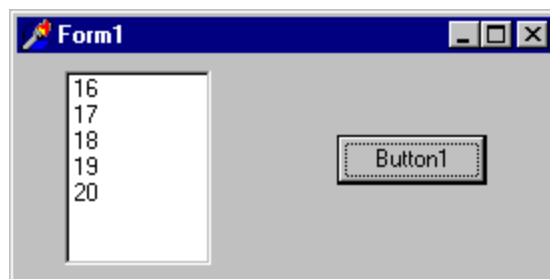


Observamos que a saída do programa não corresponde à desejada, temos então erros lógicos neste projeto.

A função digitada está incorreta pois ela não varia em função do valor de  $i$ , altere então a seguinte linha:

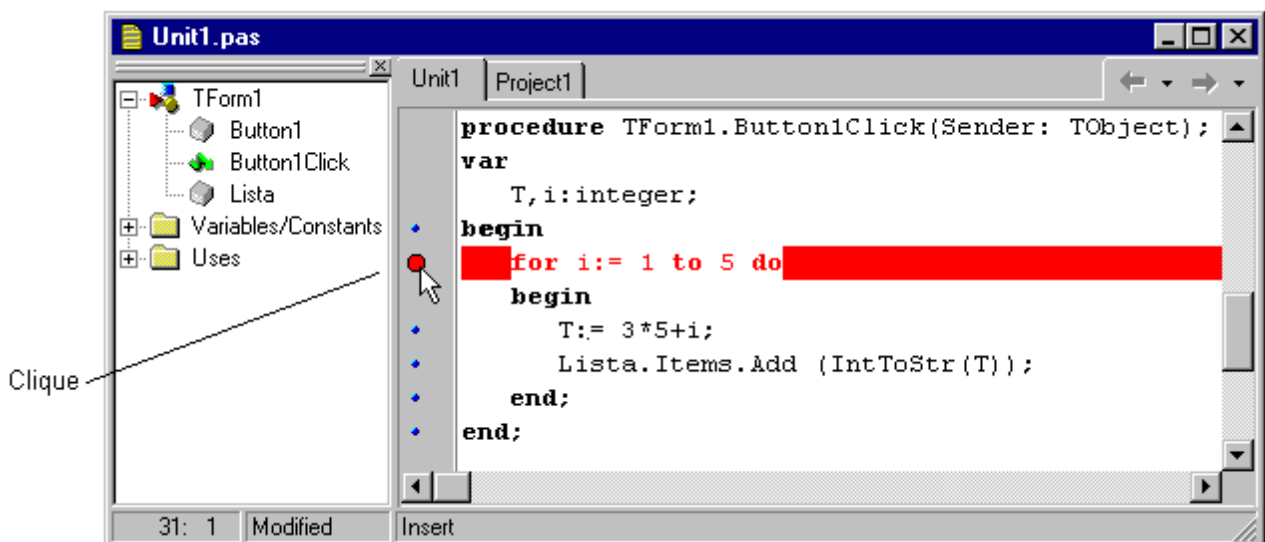
-  $T := 3*5+i$ ;

Mas a saída do programa ainda não está de acordo.

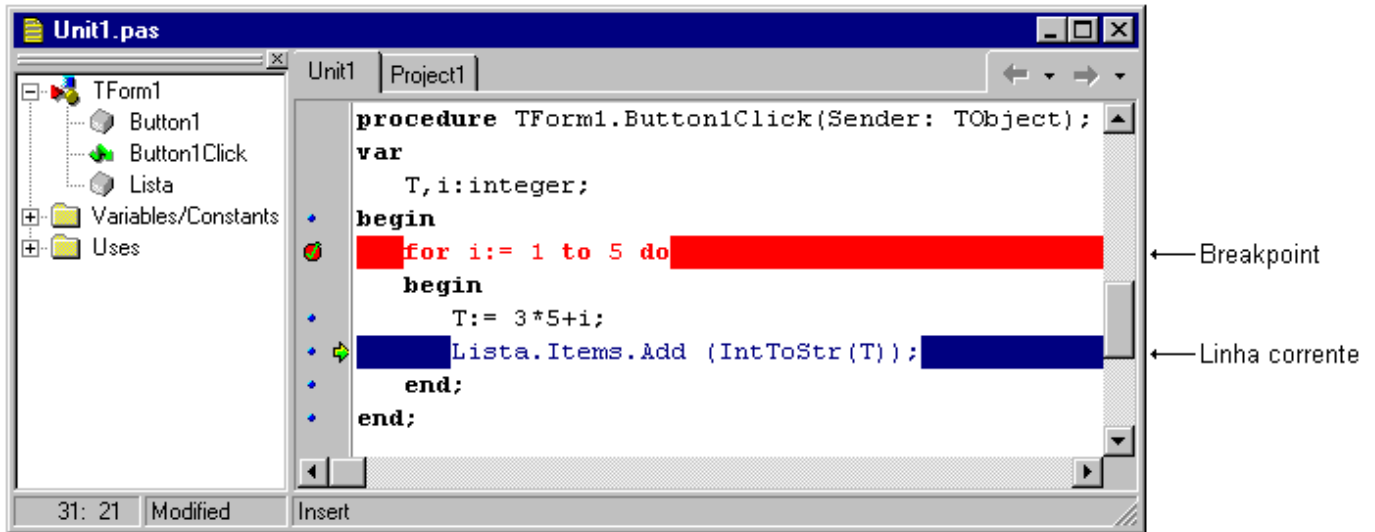


Vamos agora acompanhar o programa passo-a-passo verificando o valor das variáveis. Primeiro inserimos um ponto de parada (**Breakpoint**), o programa será executado até este ponto e depois pára.

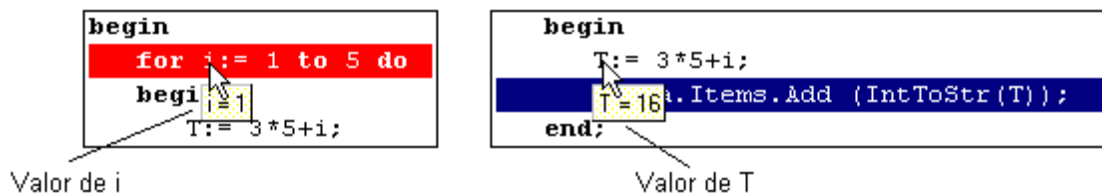
Para inserir um Breakpoint, dê um clique na barra do lado esquerdo da linha que deseja ser a parada. Quando desejar remover o Breakpoint basta dar um novo clique na mesma posição.



Quando for dado um clique no botão o programa irá parar e será mostrada a janela do Code Explorer. Para executar o programa passo-a-passo pressione **F8**, até a linha indicada na figura abaixo como linha corrente.

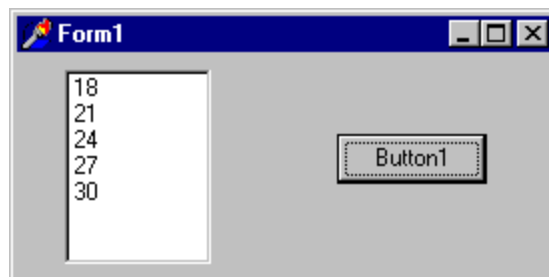


Para verificarmos o valor atual de uma variável basta posicionar o ponteiro em cima dela e aguardar uns instantes para o Delphi nos mostrar o seu valor.



Avance o programa aos passos até a finalização, acompanhando a variação de valores. Para a saída estar correta estão faltando os parênteses entre 5+i, que deve ser (5+i), para executar primeiro a soma e depois a multiplicação.

Digite os parênteses e execute o programa, agora sem nenhum erro.



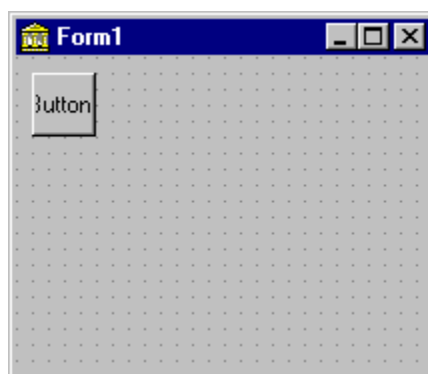
## EXEMPLO II - JOGO DA VELHA

---

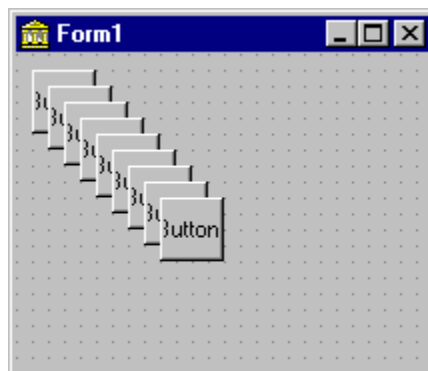
Para iniciar um novo projeto, selecione **New Application** do menu **File**. Caso você ainda não tenha salvo o seu projeto corrente, o Delphi abrirá as janelas necessárias para salvar o projeto. E só então iniciará o novo projeto.

Vamos iniciar um projeto de Jogo da Velha, onde o usuário irá jogar contra o computador que não “pensa” as suas jogadas, pois ele trabalha com jogadas aleatórias, e ao final da partida será mostrado um quadro de mensagem informando quem ganhou a partida.

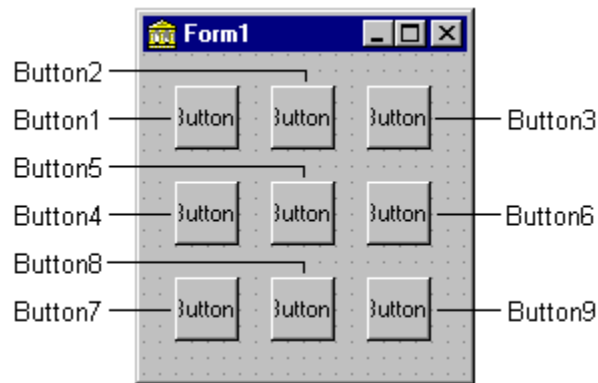
Insira um botão de comando no Formulário dimensionando-o como um quadrado, como mostra a figura.



Selecione o Botão, Copie-o (Ctrl+C) e Cole (Ctrl+V) mais oito botões iguais. Formando um total de nove botões.

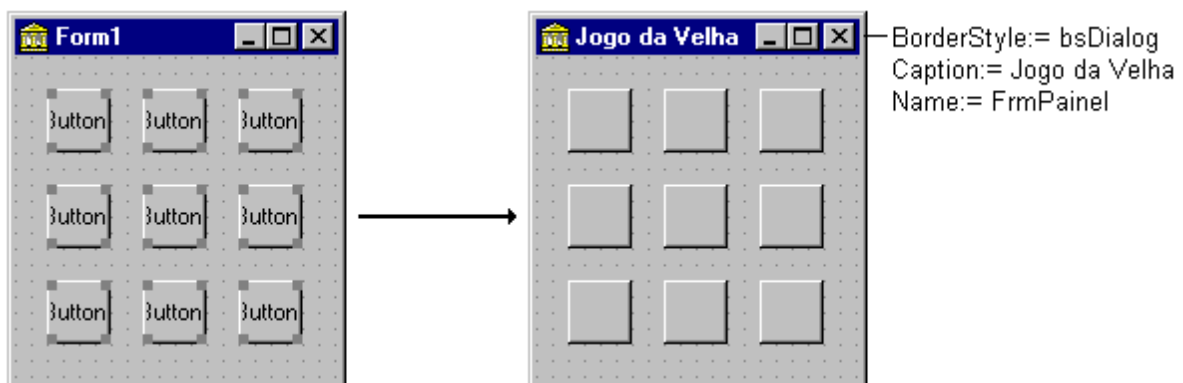


Estes botões de comando estão nomeados de Button1 a Button9, de cima para baixo. Arraste-os de forma que fiquem como a figura a seguir.



Nosso jogo irá modificar a propriedade **Caption** desses botões, mas para iniciar o jogo, todos devem estar em branco. Primeiro devemos selecionar todos ao mesmo tempo, para isso, posicione o ponteiro do mouse em um canto do Formulário e depois arraste-o até que o retângulo pontilhado envolva todos os botões.

Vá até a janela **Object Inspector** e deixe a propriedade **Caption** em branco. Qualquer propriedade que for alterada enquanto a seleção estiver ativa, servirá para todos os botões. Altere também as propriedades **Name**, **Caption** e **BorderStyle** do Formulário.



## UNIDADE (UNIT)

Uma Unit possui as seguintes seções:

- Cabeçalho
- Interface
- Implementação

O cabeçalho identifica o nome da Unit. Ele é bem compacto, contendo a palavra reservada **Unit** e o nome da unidade, que não precisa ser o mesmo nome do arquivo **.PAS**, que a contém.

Na seção **Interface** declaramos as variáveis e códigos que podem ser usados por códigos de outras unidades e lista todas as outras Units que ela precisa “ver”. Se esta Unit precisar utilizar algum recurso de outra Unit, deveremos referenciar esta outra Unit na cláusula **uses** da seção de interface. Na seção **Implementation** são declarados os códigos e variáveis que pertençam exclusivamente à unidade.



<p>UNIDADE &lt;nome da unidade&gt;;</p> <p>INTERFACE</p> <p>Seção de interface</p> <p>IMPLEMENTATION</p> <p>Seção de implementação</p> <p>FIM.</p>
--

Na seção de **Interface** da nossa unidade está sendo definido o tipo TFrmPainel (o nosso Formulário) pertencente à classe TForm, a seguir, são listados todos os controles, seus tipos, e procedimentos pertencentes a este formulário.

A figura abaixo mostra a listagem da Unit do nosso projeto:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs;

type
  TFrmPainel = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

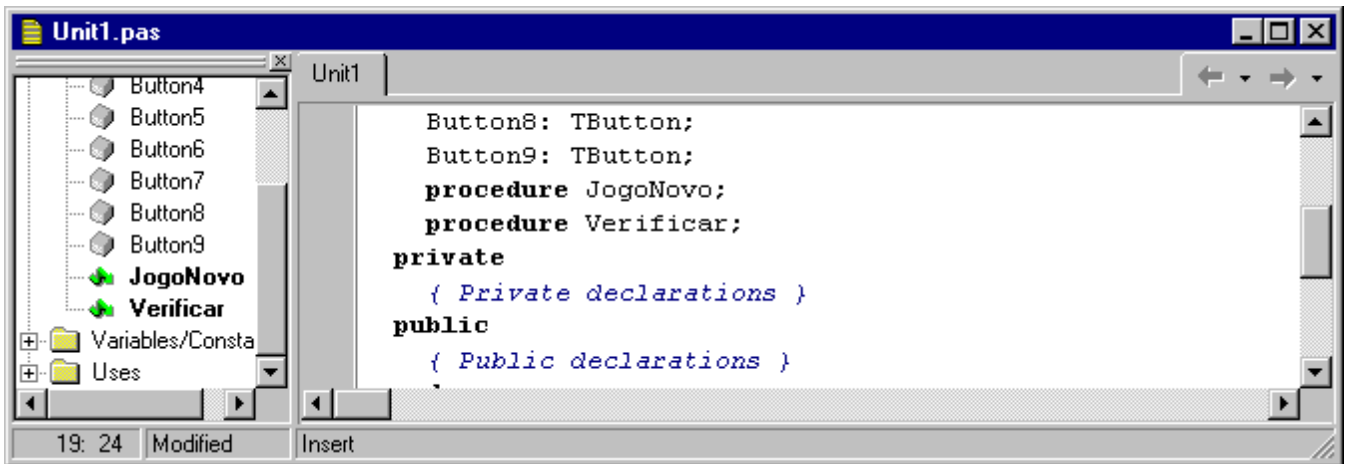
var
  FrmPainel: TFrmPainel;

implementation
  {$R *.DFM}

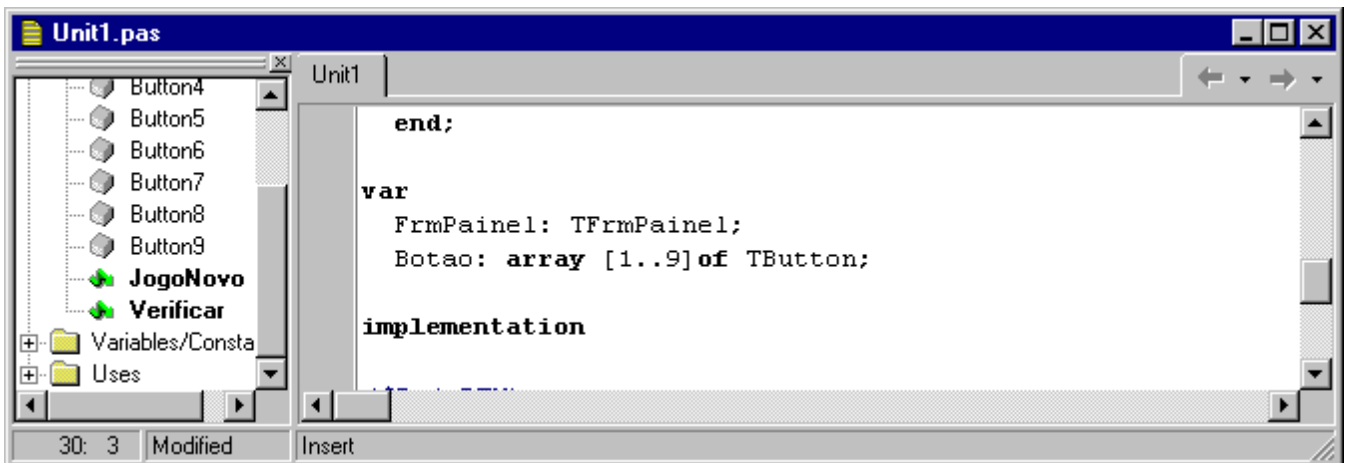
end.
```

O projeto Jogo da Velha, utiliza dois procedimentos que não estão associados a nenhum controle, são eles *JogoNovo* e *Verificar*, Estes procedimentos devem primeiro ser primeiro

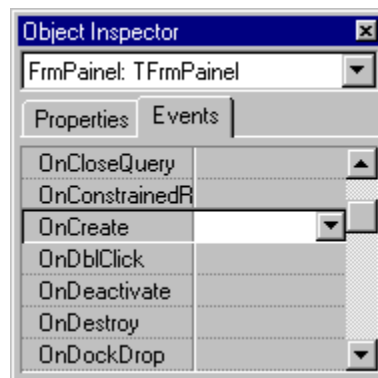
declarados para depois serem implementados, declare os procedimentos *JogoNovo* e *Verificar* na definição de tipo do formulário como mostra a listagem abaixo. Logo mais, faremos a implementação destes procedimentos que nada mais é do que a construção do seu código.



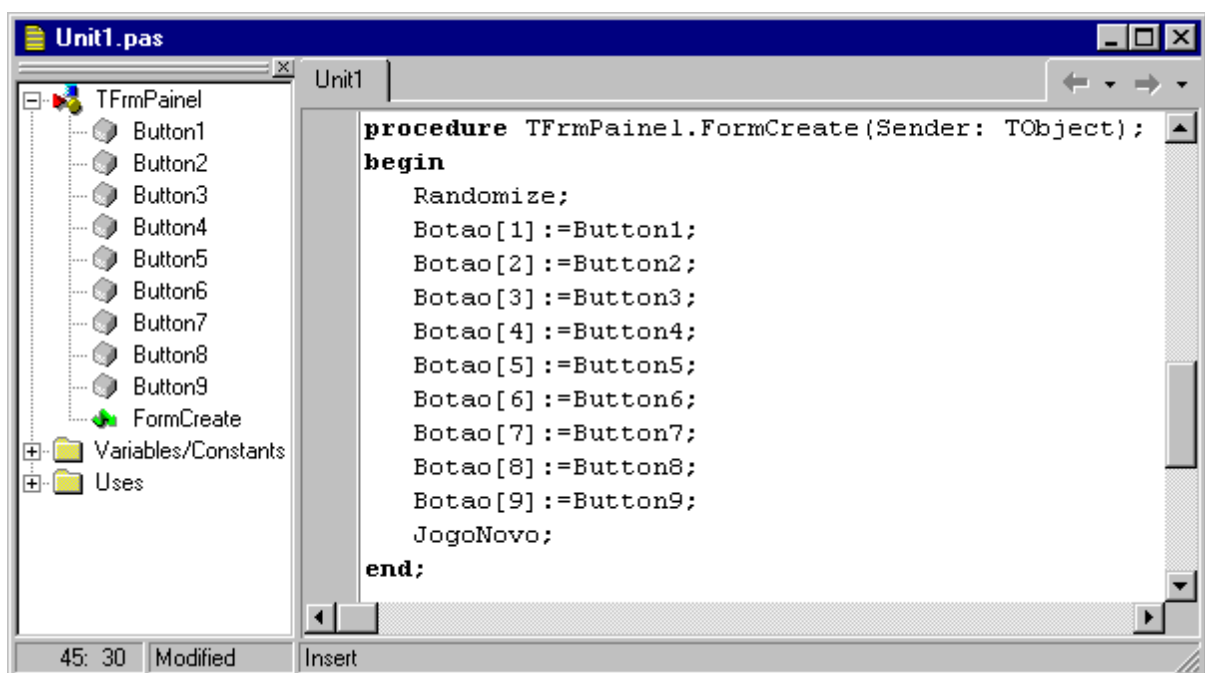
Devemos declarar também a variável **Botao** na seção **Implementation**. Esta variável é uma matriz unidimensional com nove elementos do tipo TButton, ou seja, cada variável dessas possui as mesmas propriedades de um botão de comando. Esta matriz poderia conter números inteiros (**array [1..9] of Integer;**) ou strings.



Quando o projeto iniciar, o formulário será carregado na memória, neste momento ocorre um evento **OnCreate**. Neste evento colocaremos um procedimento de início do jogo. Dê um duplo clique no formulário para ter acesso ao editor de código, ou selecione o formulário, vá até a janela **Object Inspector** escolhendo o indicador **Events**, e dê um duplo clique em **OnCreate**. Aparecendo o editor de código com o cabeçalho do procedimento.



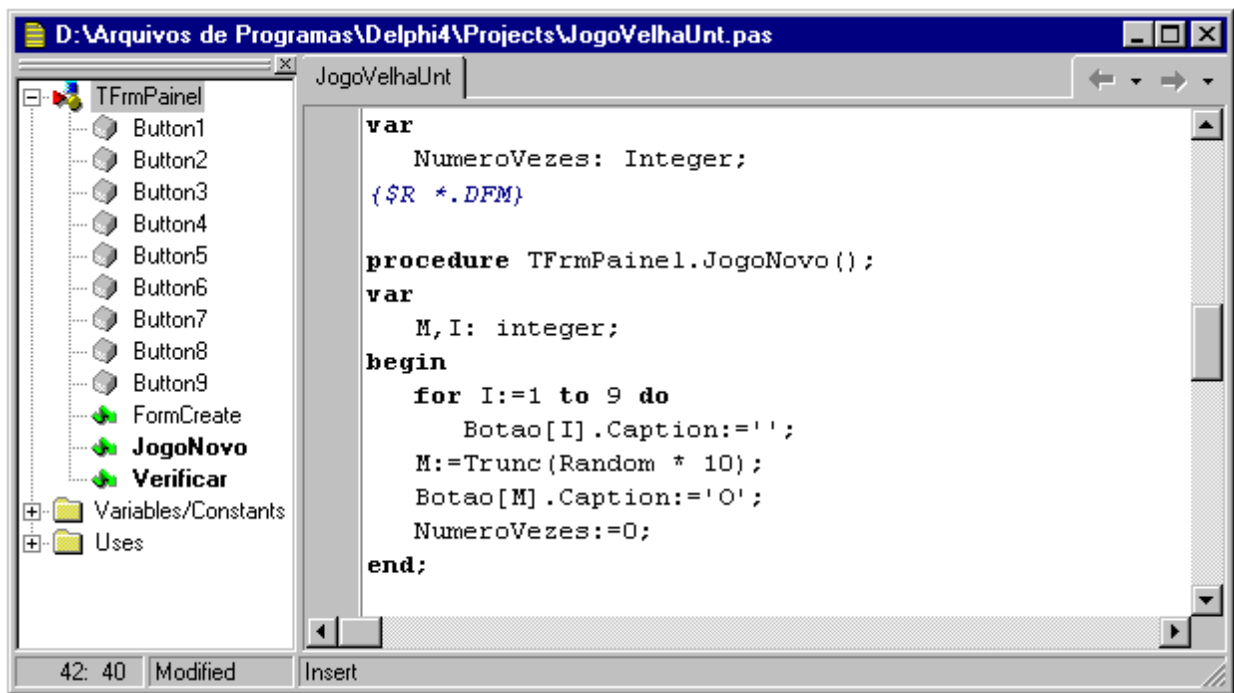
Entre com o código mostrado na figura abaixo:



No procedimento acima, temos a procedure **Randomize** que inicializa o gerador de números aleatórios utilizando como base o relógio do sistema. Nas próximas linhas atribuímos valores à variável `Botao[ ]`, associando-a aos botões do Formulário. E logo a seguir é chamada a procedure `JogoNovo`.

Salve a Unit como `JogoVelhaUnt.pas` e o projeto como `JogoVelha.dpr`.

Agora vamos implementar a procedure `JogoNovo` na seção **Implementation**, como segue. Declare também a variável `NumeroVezes` que poderá ser utilizada por todos os procedimentos desta Unidade.



Uma declaração de procedure tem a seguinte forma:

```
procedure Nome (Lista de Parâmetros); Diretivas;
Declarações Locais
begin
  Instruções
end;
```

A Lista de Parâmetros, Diretivas e Declarações Locais são opcionais. Uma Lista de Parâmetros é uma sequência de parâmetros que deverão ser enviados pela linha de instrução que chamou a procedure. Cada parâmetro possui um nome seguido pela identificação de tipo, eles podem ser precedidos pelas palavras reservadas **var** ou **const** indicando variável ou constante.

Exemplos:

(X, Y: Real)

(var S: string; X: Integer)

(HWND: Integer; Text, Caption: PChar; Flags: Integer)

(const P; I: Integer)

As Diretivas informam à procedure como ela tratará os parâmetros e como será a construção deste procedimento em relação a outros anteriores.

No cabeçalho, observamos que o nome da procedure é composto do nome do objeto que contém a procedure(Form) e o nome da procedure.

<b>procedure</b> TPainel.JogoNovo ();	
↑	↑
objeto	procedure

O procedimento **JogoNovo**, inicia um novo jogo apagando todas as legendas dos botões e fazendo uma nova jogada aleatoriamente.

A variável **I** do tipo inteiro (Integer) é de uso exclusivo deste procedimento. Findo o procedimento, ela deixará de existir.

Para apagar as legendas usamos a instrução **for...to**, que irá em loop apagando as legendas. Aqui usamos a variável **Botao[I]** para identificar os botões do formulário, com o index **I** do loop **for..to**. A instrução **for...to**, pode ser também **for...downto**, para decrementar o valor da variável de controle (**I**).

No exemplo, usamos somente uma declaração após o **do**, se fossem necessárias mais declarações, teríamos que usar **begin..end**; como uma maneira de agrupar estas declarações, indicando que todas fazem parte do loop **for...to**, como mostra o exemplo abaixo :

```
for i:=1 to 3 do
  begin
    Botao[i].Caption:='';
    Botao[i+1].Caption:='';
    Botao[i+2].Caption:='';
  end;
```

Na linha **M:= Int (Random \*10);**, atribuímos um valor inteiro à variável **M** entre 1 e 10.

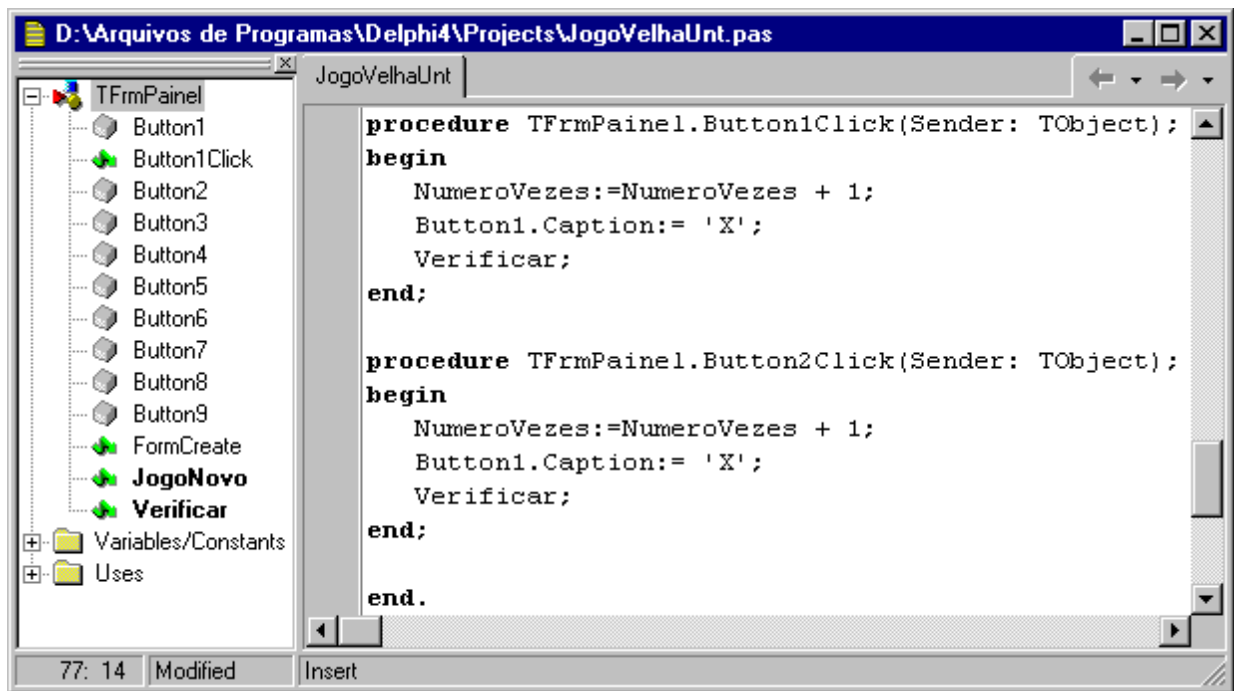
A procedure **Random** gera um número do tipo Real **X** sendo,  $0 \leq X < 1$ , multiplicamos este número por 10 e pegamos a sua parte inteira, que apesar de ser um número inteiro é do tipo Real, não podendo usá-lo para indicar um botão.

Para converter um número real em inteiro, podemos utilizar as funções **Trunc** ou **Round**. **Trunc**, retorna a parte inteira de um número, enquanto **Round** arredonda um número real.

Real	Trunc	Round
1,4	1	1
1,5	1	2
-1,4	-1	-1
2,5	2	2
-2,7	-2	-3

No nosso código utilizamos a função **Trunc** para retornar o índice do botão que terá sua legenda alterada.

O próximo passo é fazer o código dos botões, dê um duplo clique no **Button1** para a janela de código aparecer, e copie o exemplo. Faça o mesmo para os demais botões.



O evento **OnClick** nos botões é que dará partida ao programa. O procedimento associado a este evento, incrementará o número de jogadas (**NumeroVezez**), mudará a legenda do botão e chamará o procedimento para verificar se o jogo terminou.

Construa o procedimento **Verificar** na seção de implementação da unidade, como segue:

```

procedure TFrmPainel.Verificar ();

label 1,2;
var I: integer;

begin
    if (Button1.Caption='X') and (Button2.Caption='X')
        and (Button3.Caption='X') then goto 1;
    if (Button4.Caption='X') and (Button5.Caption='X')
        and (Button6.Caption='X') then goto 1;
    if (Button7.Caption='X') and (Button8.Caption='X')
        and (Button9.Caption='X') then goto 1;
    if (Button1.Caption='X') and (Button4.Caption='X')
        and (Button7.Caption='X') then goto 1;
    if (Button2.Caption='X') and (Button5.Caption='X')
        and (Button8.Caption='X') then goto 1;
    if (Button3.Caption='X') and (Button6.Caption='X')
        and (Button9.Caption='X') then goto 1;
    if (Button1.Caption='X') and (Button5.Caption='X')
        and (Button9.Caption='X') then goto 1;
    if (Button3.Caption='X') and (Button5.Caption='X')
        and (Button7.Caption='X') then goto 1;

    repeat I:=Trunc (Random * 10) until Botao[I].Caption='' ;
    Botao[I].Caption:='O';

```

```

if (Button1.Caption='O') and (Button2.Caption='O')
  and (Button3.Caption='O') then goto 2;
if (Button4.Caption='O') and (Button5.Caption='O')
  and (Button6.Caption='O') then goto 2;
if (Button7.Caption='O') and (Button8.Caption='O')
  and (Button9.Caption='O') then goto 2;
if (Button1.Caption='O') and (Button4.Caption='O')
  and (Button7.Caption='O') then goto 2;
if (Button2.Caption='O') and (Button5.Caption='O')
  and (Button8.Caption='O') then goto 2;
if (Button3.Caption='O') and (Button6.Caption='O')
  and (Button9.Caption='O') then goto 2;
if (Button1.Caption='O') and (Button5.Caption='O')
  and (Button9.Caption='O') then goto 2;
if (Button3.Caption='O') and (Button5.Caption='O')
  and (Button7.Caption='O') then goto 2;
if NumeroVezes= 4 then
  begin
    ShowMessage ('Partida Empatada');
    JogoNovo;
    exit;
  end;
exit;

1:
  begin
    ShowMessage ('Você Ganhou');
    JogoNovo;
    exit;
  end;

2:
  begin
    ShowMessage ('Eu Ganhei');
    JogoNovo;
    Exit;
  end;
end;

```

Logo após o cabeçalho, temos as declarações locais. Usamos a declaração **label** para definir os locais de desvio da instrução **goto**. E também declaramos a variável **I** como do tipo integer.

A instrução **repeat declaração until expressão**; repete a declaração enquanto a expressão for verdadeira. No nosso exemplo, o programa selecionará um botão aleatoriamente, até que seja encontrado um botão “vazio” - Caption:= ' ', preenchendo-o com “O”, indicando a jogada do computador.

A procedure **ShowMessage**, mostra um quadro de mensagem com o botão de **Ok**. O programa fica parado até que o usuário dê um clique em Ok, retornando à linha seguinte no procedimento.

O Windows possui quadros padronizados de mensagem que servem para emitir avisos e recolher opções de tratamento dessas mensagens. Podemos incrementar o nosso programa do Jogo da Velha, criando um quadro de mensagem com vários botões e que retorne uma resposta do usuário, indicando qual botão foi escolhido. Para isso utilizamos a função **MessageBox**.

Esta função pertence à biblioteca do Windows (API). O Delphi a relaciona ao objeto do tipo **TApplication**.

MessageBox (Mensagem ,Título, Tipo);

Onde:

mensagem - expressão mostrada dentro do quadro de diálogo.

tipo - somatória de números, conforme o que queremos que seja exibido no Quadro de Mensagem, seguindo a tabela Tipo.

título - título do Quadro de Mensagem (barra de título).

Mude o código do procedimento **Verificar**, como mostrado abaixo.

```
1:
  begin
    Resposta:=Application.MessageBox
      ('Você ganhou, quer Jogar Novamente?','Vencedor',36);
    if Resposta = 7 then Close;
    JogoNovo;
    exit;
  end;

2:
  begin
    Resposta:=Application.MessageBox
      ('Eu ganhei, quer Jogar Novamente?','Vencedor',36);
    if Resposta = 7 then Close;
    JogoNovo;
    exit;
  end;
```



**Argumento Tipo para a função MessageBox**

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

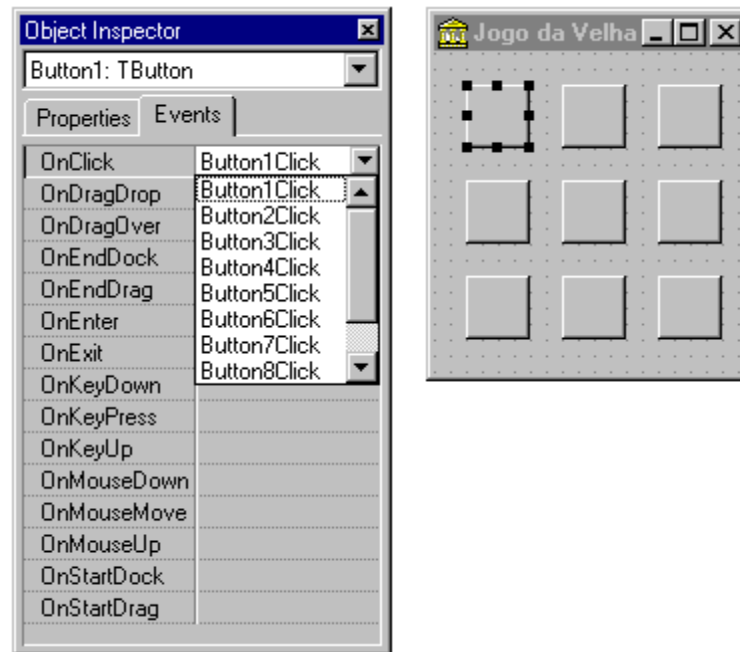
Teste o projeto alterando o valor de tipo para MessageBox, faça a sua soma escolhendo um item de cada seção.

A variável **Resposta**, conterà a resposta do usuário que segue o padrão da tabela abaixo.

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

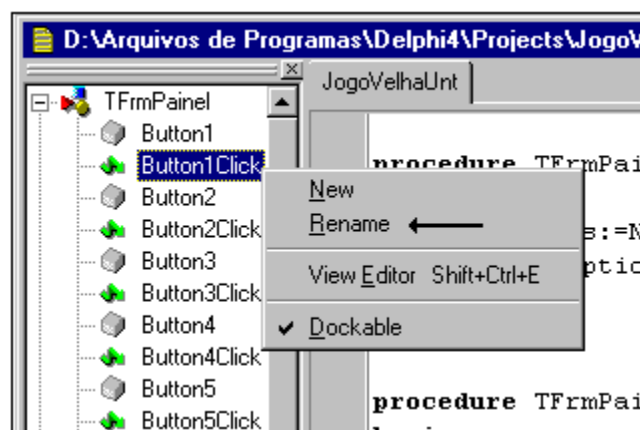
No nosso caso, o programa verificará se o botão **Não** foi pressionado, e se foi, fechará o formulário principal, encerrando o programa.

Em nosso exercício, todos os botões realizam basicamente a mesma operação quando respondem ao evento OnClick. Vamos alterar o projeto para que exista somente uma procedure que trate o evento OnClick para todos os botões.

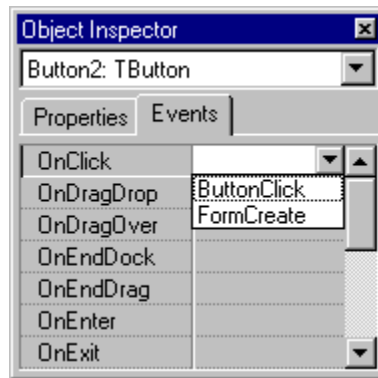


Primeiro apague as linhas entre **begin...end** dos procedimentos relativos aos botões exceto o **Button1Click**. Compile o projeto (Ctrl+F9) para que o Delphi exclua estes procedimentos do editor de código - qualquer procedimento "vazio" será excluído.

No Code Explorer, de um clique com o botão direito e no menu, escolha Rename e altere o nome do procedimento **Button1Click** para **ButtonClick**.



Após a compilação, os procedimentos associados aos demais botões foram excluídos do editor de códigos e como consequência, estes botões não responderão mais ao evento **OnClick**. Acesse a página **Events** do **Object Inspector**, associando a procedure **ButtonClick** como resposta ao evento **OnClick** para todos os botões. Desta forma todos os botões executarão o mesmo procedimento a partir do evento **OnClick**.



Por último vamos alterar a procedure ButtonClick para que ela reconheça o botão que recebeu o evento OnClick, alterando sua Caption.

```
procedure TFrmPainel.ButtonClick(Sender: TObject);  
begin  
    NumeroVezes:=NumeroVezes + 1;  
    (Sender as TButton).Caption:= 'X';  
    Verificar;  
end;
```

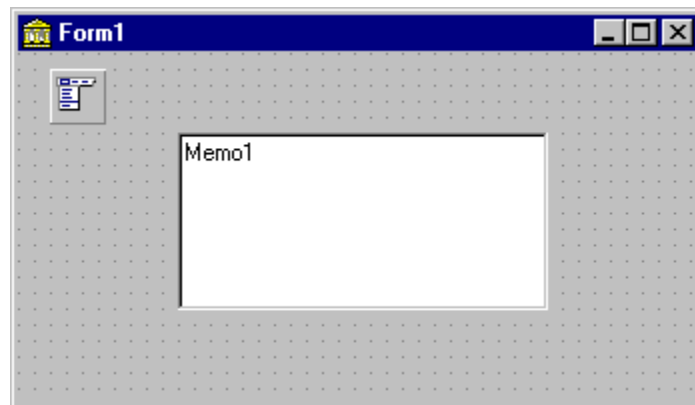
O parâmetro Sender indica qual objeto recebeu o evento acionador da procedure.

## EXEMPLO III - BLOCO DE NOTAS

---

O nosso próximo projeto será um editor de texto simples do tipo caractere, com ele poderemos alterar o tipo e tamanho da fonte utilizada em todo o texto, recortar, colar e copiar partes selecionadas, salvar e abrir nosso texto utilizando as caixas de diálogo padrão fornecidas pelo Delphi.

Monte o formulário conforme o exemplo:



Defina a propriedade **Align** do componente Memo como **alCliente**. Esta propriedade faz com que o Memo ocupe toda a área do cliente do formulário, mesmo que ele seja redimensionado.

O Delphi possui componentes visíveis e não visíveis. Os componentes visíveis são aqueles que durante a execução do programa são vistos pelo usuário. Até aqui todos os componentes que trabalhamos são visíveis, como TEdit, TButton, TForm e outros.

Os componentes não visíveis, não aparecem na janela do usuário em tempo de execução. São eles, Timer, Menus, Caixas de diálogo comuns e controle de acesso a dados.

O nosso exemplo de Bloco de Notas, usará um menu e quatro caixas de diálogo.

Para começar a editar o menu, dê um duplo clique no controle **MainMenu** que está dentro do formulário, para que a janela **Menu Designer** apareça.



É nesta janela que iremos construir o menu do nosso exemplo. Observe que o primeiro título já está selecionado.

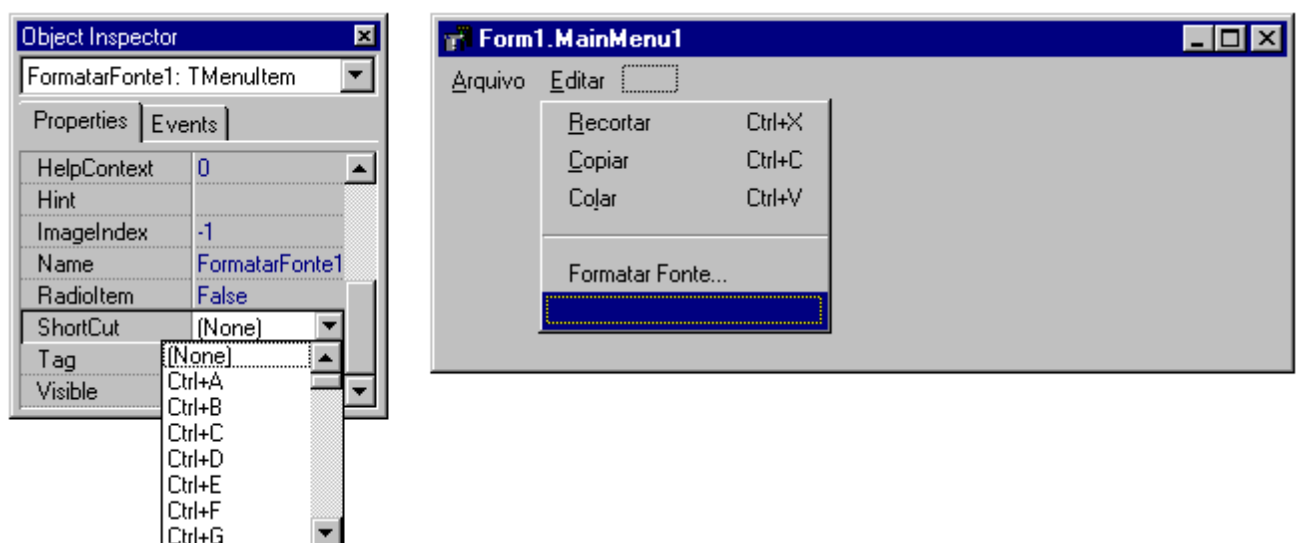
Vá até a janela **Object Inspector** e mude a propriedade **Caption** para **&Arquivo** e pressione Enter - para acesso via teclado, usamos o “&” comercial antes da letra que queremos que seja o atalho. Este procedimento, cria o menu Arquivo e move a barra de destaque para o lado, dê um clique logo abaixo da opção Arquivo inserindo um item deste menu. Repare que o Delphi coloca um nome para este menu baseado na propriedade Caption, neste caso Name:=Arquivo.



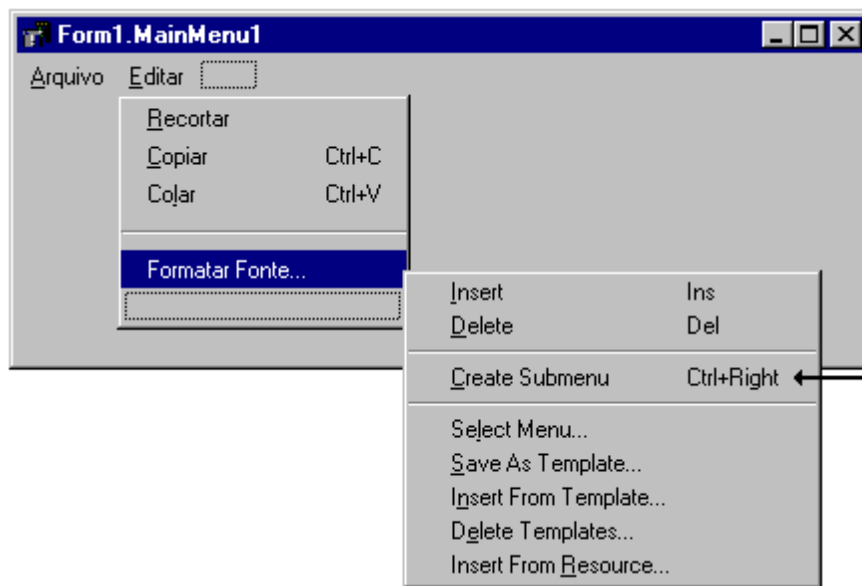
Monte as demais opções do nosso menu seguindo a janela Menu Designer mostrada abaixo. Para criar um separador no menu, digite apenas um sinal de menos ( - ) na propriedade Caption do item abaixo de Salvar.



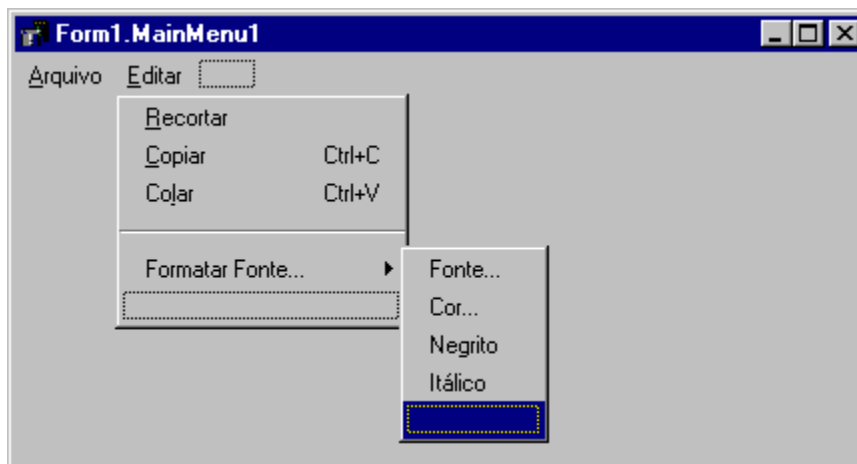
Terminado o menu Arquivo, inicie o menu Editar, como segue abaixo. Defina as teclas de atalho na propriedade **ShortCut** dos itens de menu.



Quando chegarmos ao item **Formatar Fonte...**, escolha a opção **Create Submenu** do PopupMenu, para criar um submenu deste item.



O sub-menu de **Formatar Fonte...**, deverá estar igual a figura mostrada abaixo. Como o Delphi não reconhece caracteres brasileiros, o nome que ele dará para o item Itálico será Itlico, suprimindo do Name o caractere **á** ( Name:= Itlico).



Feche a Menu Designer, voltando ao formulário principal. Insira nele as caixas de diálogo que irão formatar a fonte exibida no componente Memo, e as caixas que irão operar com o disco (HD). Os componentes destas caixas estão na paleta **Dialogs** do Delphi.

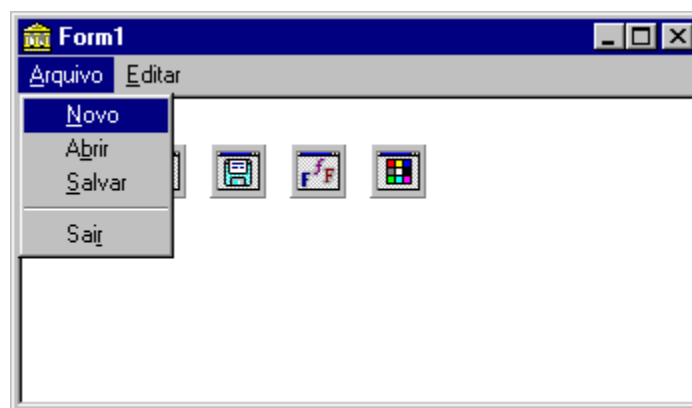


Acabamos de desenhar o nosso formulário, colocando todos os componentes a ele pertencentes. Os visíveis e os não visíveis. Mesmo os componentes não visíveis estando em cima do Memo, não atrapalharão a apresentação do texto.

Salve seu trabalho para darmos início à construção do código.

As caixas de diálogo são mostradas através do método **Execute**. Este método responde True se o usuário selecionou Ok, indicando que o programa deverá responder às alterações da caixa de diálogo exibida. Se o usuário não quiser efetuar as mudanças, será retornado False.

Dê um clique no item **Novo** do menu, para chamar o procedimento associado. Este procedimento irá limpar a caixa Memo1 e desabilitar as opções de edição do texto. Estas opções estarão desabilitadas até que se tenha um texto para Recortar, Copiar ou Colar. Siga o código mostrado a seguir:



```
procedure TPrincipal.Novo1Click(Sender: TObject);
begin
    Memo1.Clear;
    Recortar1.Enabled:=False;
    Copiar1.Enabled:=False;
    Colar1.Enabled:=False;
end;
```

A opção **Abrir** trabalhará com a caixa de diálogo OpenFileDialog, verificando o valor retornado por **Execute** e carregando o conteúdo do arquivo selecionado, na propriedade Lines do objeto Memo.

```
procedure TPrincipal.Abrir1Click(Sender: TObject);
begin
    if OpenFileDialog1.Execute then
        Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

Digite o código para as outras caixas de diálogo. Elas trabalham alterando as propriedades do Memo após as mudanças realizadas pelo usuário.

```
procedure TPrincipal.Salvar1Click(Sender: TObject);
begin
    if SaveDialog1.Execute then
        Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

```
procedure TPrincipal.Fonte1Click(Sender: TObject);
begin
    FontDialog1.Font:=Memo1.Font;
    {inicializa a FontDialog com a font de Memo}
    if FontDialog1.Execute then
        Memo1.Font:=FontDialog1.Font;
end;
```

A linha entre chaves indica um comentário, não sendo tratada pelo compilador do Delphi.

```
procedure TPrincipal.Cor1Click(Sender: TObject);
begin
    ColorDialog1.Color:=Memo1.Color;
    if ColorDialog1.Execute then
        Memo1.Font.Color:=ColorDialog1.Color;
end;
```

Quando o programa começa a ser executado, o evento OnCreat ocorre com o Formulário, no procedimento deste evento, iremos apagar o conteúdo do Memo e desabilitar as opções do menu Editar.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Memo1.Clear;
    Recortar1.Enabled:=False;
    Copiar1.Enabled:=False;
    Colar1.Enabled:=False;
end;
```



As opções Recortar e Copiar do menu Editar, estarão ativas assim que o Memo contiver algum texto. Cada vez que ocorre uma mudança no Memo, o evento OnChange é gerado.

```
procedure TPrincipal.Memo1Change(Sender: TObject);
begin
    Recortar1.Enabled:=True;
    Copiar1.Enabled:=True;
end;
```

Os recursos de Recortar, Colar e Copiar utilizam o objeto **TClipboard**. Com ele nós usamos a área de transferência do Windows e podemos trocar informação entre programas. O objeto TMemo possui métodos próprios de trabalhar com o Clipboard, eles estão nos procedimentos para os itens do menu Editar mostrados abaixo.

```
procedure TPrincipal.Recortar1Click(Sender: TObject);
begin
    Memo1.CutToClipboard;
    Colar1.Enabled:=True; {habilita o item Colar}
end;
```

```
procedure TPrincipal.Copiar1Click(Sender: TObject);
begin
    Memo1.CopyToClipboard;
    Colar1.Enabled:=True;
end;
```

```
procedure TPrincipal.Colar1Click(Sender: TObject);
begin
    Memo1.PasteFromClipboard;
end;
```

As opções **Negrito** e **Itálico**, formatarão o texto e mudarão também a propriedade **Checked** do item no menu, indicando que elas estão selecionadas. Os procedimentos associados à negrito e itálico, trabalham juntos com o procedimento **Fonte** que verificará o estado das opções alterando as propriedades da fonte do Memo.

Inicie declarando o procedimento **Fonte** na seção de definição do tipo TPrincipal - nosso formulário. E depois, implemente-o na seção **Implementation**.

```
procedure Memo1Change(Sender: TObject);
procedure Recortar1Click(Sender: TObject);
procedure Copiar1Click(Sender: TObject);
procedure Colar1Click(Sender: TObject);
procedure Fonte;
private
    { Private declarations }
public
    { Public declarations }
end;
```

**implementation**

```
{ $R *.DFM }
procedure TPrincipal.Fonte ();
begin
    if (Negritol.Checked=False) and (Itlicol.Checked=False) then
        Memol.Font.Style:= [];
    if (Negritol.Checked=True) and (Itlicol.Checked=False) then
        Memol.Font.Style:= [fsBold];
    if (Negritol.Checked=False) and (Itlicol.Checked=True) then
        Memol.Font.Style:= [fsItalic];
    if (Negritol.Checked=True) and (Itlicol.Checked=True) then
        Memol.Font.Style:= [fsBold, fsItalic];
end;
```

```
procedure TPrincipal.NegritolClick(Sender: TObject);
begin
    Negritol.Checked:= Not Negritol.Checked;
    Fonte;
end;
```

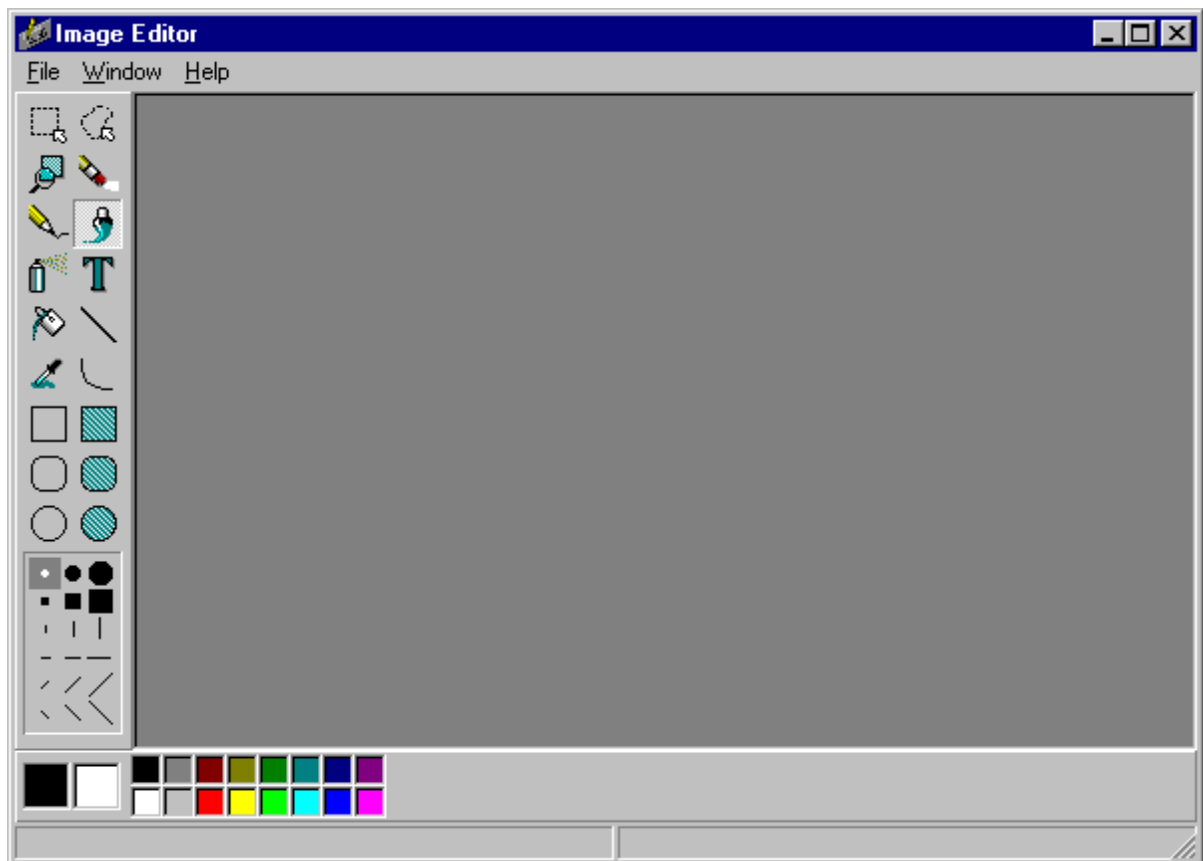
```
procedure TPrincipal.ItlicolClick(Sender: TObject);
begin
    Itlicol.Checked:= Not Itlicol.Checked;
    Fonte;
end;
```

Quando o usuário clicar no menu Sair, fechará o formulário, finalizando a execução do programa pois este é o único formulário do nosso aplicativo. Isto é feito com o uso do método Close.

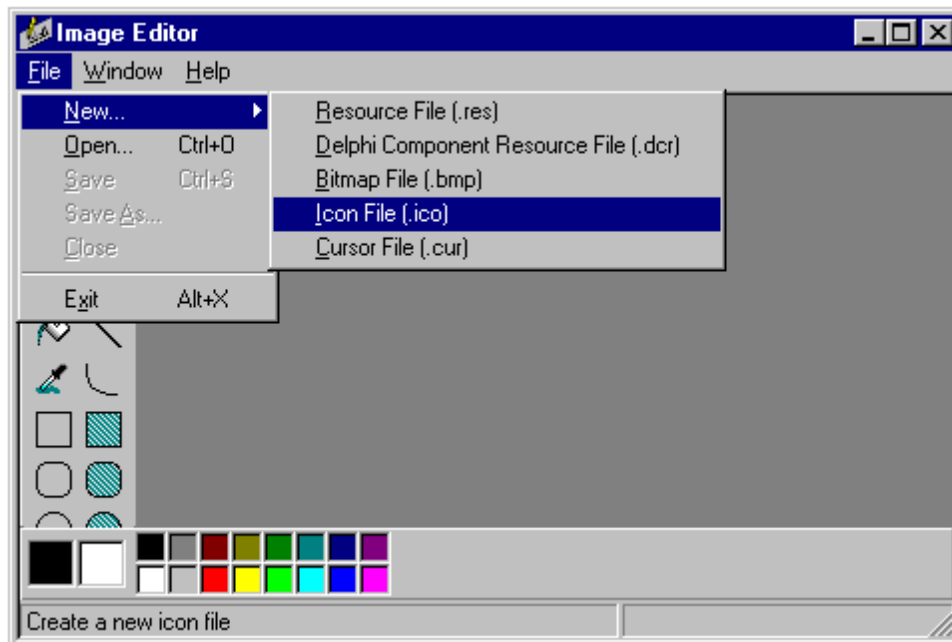
```
procedure TForm1.Sair1Click(Sender: TObject);
begin
    Close;
end;
```

Salve seu trabalho, e teste o programa pressionando F9.

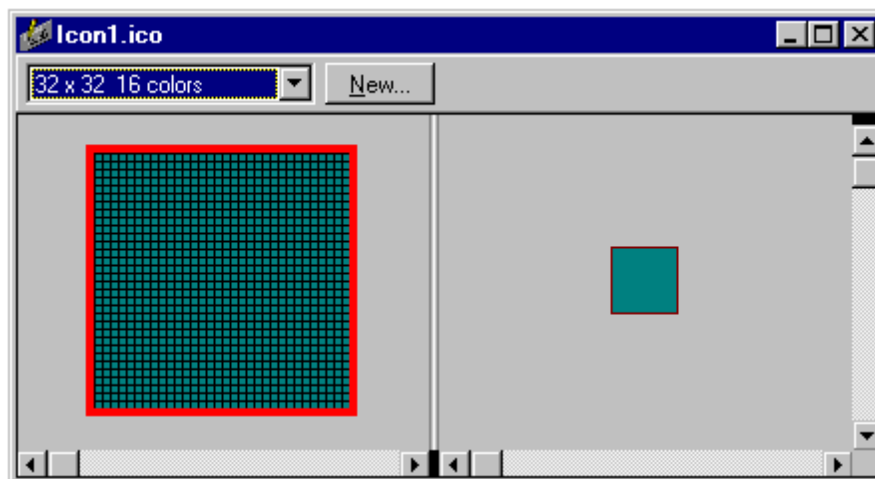
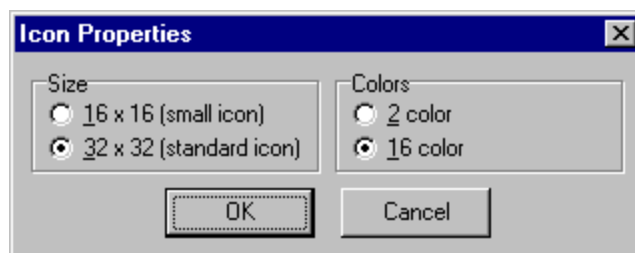
Nós já aprendemos a associar um ícone ao nosso projeto, agora iremos aprender a desenhar um, utilizando o aplicativo **Image Editor** do Delphi. Para inicializá-lo, vá até a opção Tools do menu principal do Delphi, e escolha Image Editor.



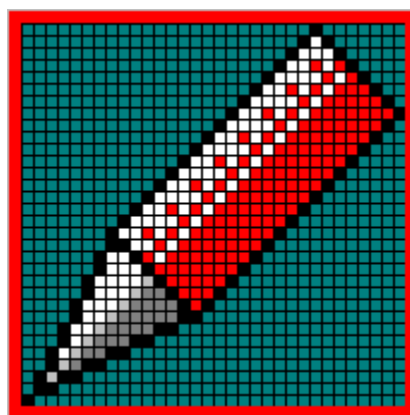
Escolha a opção **New** do menu **File**, e **IconFile**.



Na janela de propriedades, escolha uma matriz de 32 x 32 pixels e 16 cores.

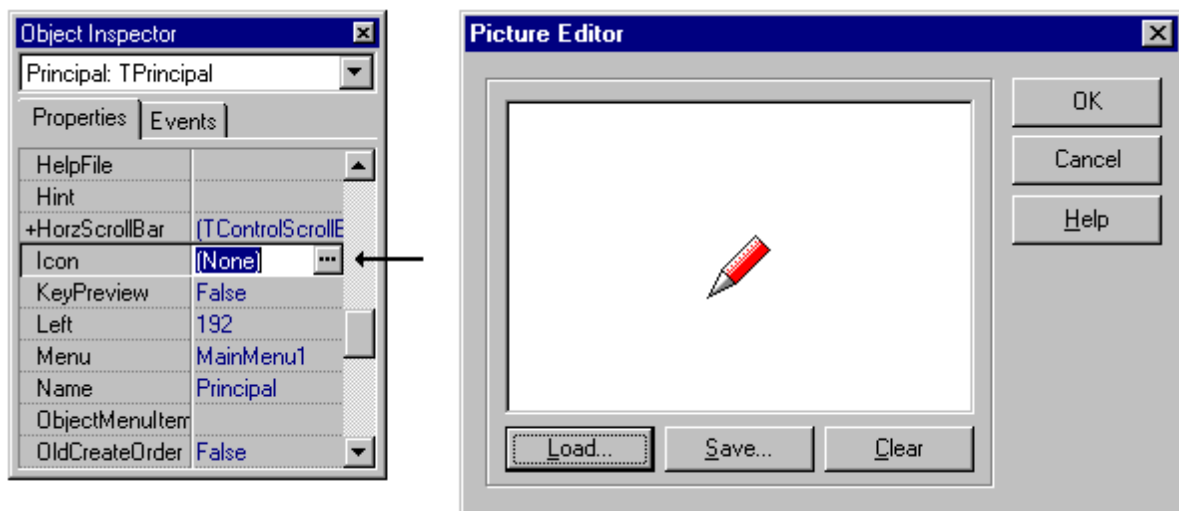


Desenhe um ícone semelhante ao da figura abaixo, utilizando recursos semelhantes ao programa Paint do Windows. A cor de fundo (verde) funciona como uma cor Cromaqui - transparente ao fundo.

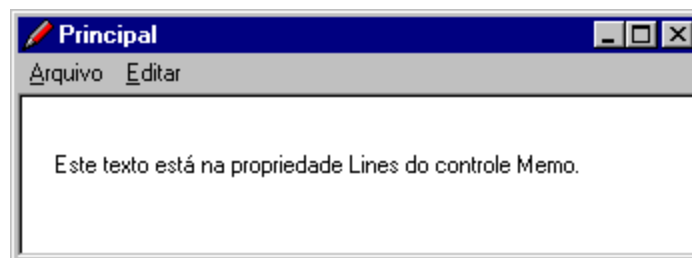


Salve o seu ícone memorizando o nome completo incluindo o caminho. E feche o aplicativo Image Editor.

O formulário possui no canto superior esquerdo o ícone padrão do Delphi. Para trocar este ícone pelo nosso ícone da caneta, deveremos alterar a propriedade Icon do formulário e carregar o ícone que acabamos de desenhar.



Use este ícone para representar o Bloco de Notas no Windows, da mesma forma descrita para a Calculadora.

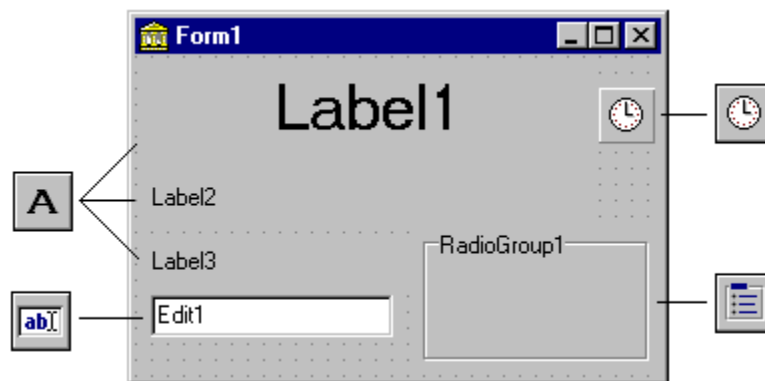


## EXEMPLO IV - RELÓGIO DESPERTADOR

Este projeto contém um outro componente não visível - o **Timer**. Este componente gera o evento **OnTimer** a intervalos regulares determinados em sua propriedade **Interval**. Esta propriedade está expressa em milissegundos, ou seja, para que o evento OnTimer ocorra a cada segundo: Interval:=1000. Se Interval:=0 o Timer estará desativado.

Devemos ter cuidado na programação do Timer, porque mesmo ele sendo capaz de gerar um evento a cada milissegundo, o procedimento que trata o evento pode demorar mais do que o valor de Interval, ocorrendo perda de algumas execuções do procedimento ligado a ele, ou se Interval for muito pequeno pode ocorrer travamento do programa.

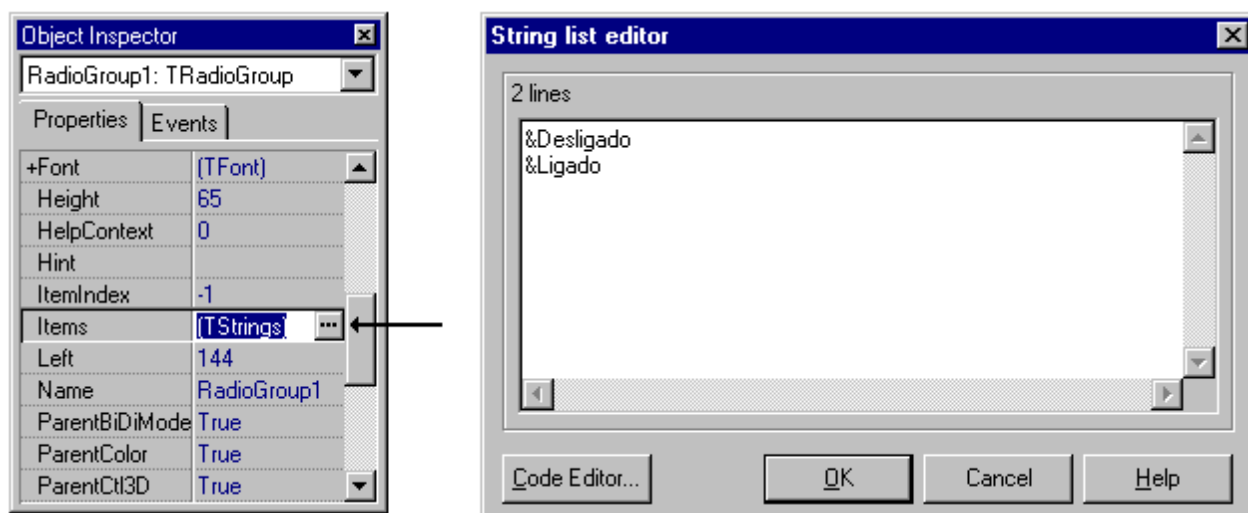
Construa o formulário como mostrado abaixo.



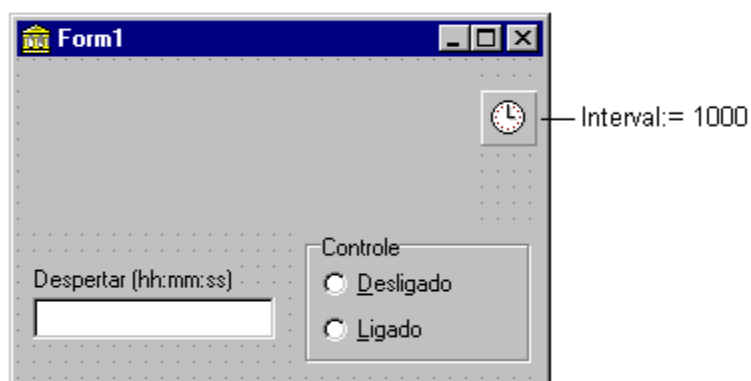
A propriedade Caption dos Label1 e Label2 deverá estar em branco. No Label1, será apresentada a hora do sistema, e em Label2 a data atual.

O componente RadioGroup permite construir um grupo de botões de opção, utilizamos estes botões quando precisarmos selecionar opções mutuamente excludentes. Só poderá haver um botão de opção selecionado por vez, em cada grupo ou no formulário caso o botão esteja fora de algum grupo.

As opções de escolhas do **RadioGroup** estão na propriedade **Items**. Quando selecionamos esta propriedade, a janela **String list editor** é mostrada, nela editamos os botões de opção do grupo. Entre com as duas opções, como mostrado a seguir.



Formulário pronto



Feito o formulário, vamos digitar o código para os eventos associados.

Declare primeiro as variáveis Ligado e Tecla. Ligado indicará qual o botão de opção está selecionado, e Tecla armazenará um caractere digitado no teclado.

```
implementation
var
  Ligado: Boolean;
  Tecla: Char;
{$R *.DFM}
```

Quando o Formulário for criado, o botão Desligado será selecionado através da linha "RadioGroup1.ItemIndex:= 0". A propriedade **ItemIndex**, indica qual o botão do grupo está selecionado.

Colocaremos a data do sistema em Label2 usando a função `FormatDateTime`, de acordo com a tabela mostrada no projeto da Calculadora.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    RadioGroup1.ItemIndex := 0;
    Label2.Caption:=FormatDateTime('dddd,dd "de" mmmm "de" yy'
,Date); {a função Date retorna a data do sistema}
end;
```

A propriedade `ItemIndex` do `RadioGroup1` será igual a 1 se o botão Ligado for selecionado, e igual a 0, caso Desligado esteja selecionado.

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
    if RadioGroup1.ItemIndex = 1 then
        Ligado:=True
    else
        Ligado:=False;
end;
```

Quando o usuário digita algo em um quadro `Edit`, o ocorre evento **KeyPress**. No procedimento a seguir, é realizada uma validação dos caracteres para definir a hora de acionamento do despertador. Caso não seja um caractere válido, é soado um alarme e este caractere é excluído do `Edit`.

O evento **KeyPress** envia para a procedure, o parâmetro **Key** do tipo **Char**, sendo esta variável verificada pelo programa.

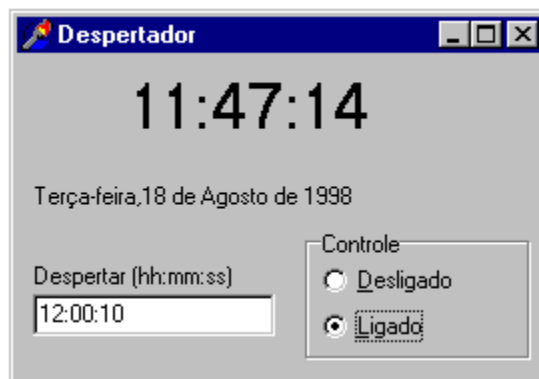
```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    Tecla:=(Key);
    if ((Tecla < '0') or (Tecla > '9')) and (Tecla <> ':') then
        begin
            Beep;
            Key:=chr(0); {a função chr(X:byte), retorna o caractere
correspondente ao número X na tabela ASCII}
        end;
end;
```



Quando o evento Timer ocorrer (no caso a cada segundo - Interval:=1000), será feita uma verificação do conteúdo do Edit1 e se o alarme está Ligado ou não. O Timer também atualiza a hora mostrada no Label1. Se a hora no Edit1 for menor ou igual a hora do sistema e a variável Ligado verdadeira, o Beep soará.

```
procedure TForm1.Timer1Timer(Sender: TObject);  
begin  
    if (Edit1.Text <= TimeToStr(Time))and (Ligado) then  
        begin  
            Beep;  
            Beep;  
        end;  
    Label1.Caption:=TimeToStr(Time);  
end;
```

Despertador sendo executado



## MÉTODOS GRÁFICOS

---

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

O Sistema de coordenadas do Delphi, possui o seu ponto de origem (0,0) no canto superior esquerdo da tela, ao contrário do que nós estamos acostumados desde Descartes.

A escala utilizada no Delphi é o Pixel, que representa um ponto visível na tela, de acordo com a resolução do monitor.

As funções e procedimentos para desenho, estão agrupadas em um objeto da classe TCanvas, o Delphi cria para nós um objeto deste tipo de nome Canvas. É com este objeto que desenhamos na tela pois, a tela é realmente um objeto da classe TCanvas.

### **DESENHO DE PONTO**

Para desenhar um ponto usamos a propriedade **Pixels**, do objeto Canvas;

```
Canvas.Pixels [x,y]:= cor;
```

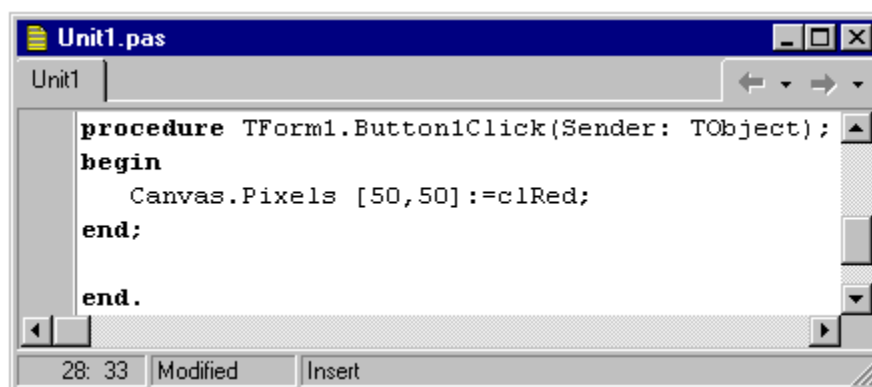
Onde:

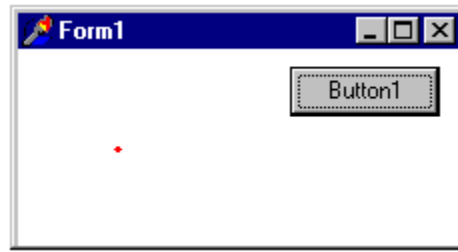
Canvas - é o objeto do tipo TCanvas, declarado desta forma, o ponto será desenhado na área do cliente - Formulário.

x,y - coordenadas horizontal e vertical, respectivamente. Elas são absolutas, ou seja, a partir da origem.

cor - especifica uma cor para o ponto.

O exemplo abaixo desenha um ponto vermelho nas coordenadas (50,50) do formulário, quando o botão for selecionado.





## CORES

O Delphi possui três formas diferentes de atribuir valores para o parâmetro *cor* de um objeto gráfico, são elas:

a) **RGB** - (NRed, NGreen, NBlue), onde NRed, NGreen e NBlue pode variar de 0 a 255.

Ponto vermelho: `Canvas.Pixels[x,y]:= RGB(255,0,0);`

b) **\$00bbggrr** - onde bb, gg e rr variam em hexadecimal de 00 a FF.

Ponto Amarelo: `Canvas.Pixels[x,y]:= $0000FFFF;`

c) Usando uma das constantes válidas para o objeto TColor.

- Para cores fixas :clAqua, clBlack, clBlue, clDkGray, clFuchsia, clGray, clGreen, clLime, clLtGray, clMaroon, clNavy, clOlive, clPurple, clRed, clSilver, clTeal, clWhite, e clYellow.
- Para cores definidas pelo sistema: clActiveBorder, clActiveCaption, clAppWorkSpace, clBackground, clBtnFace, clBtnHighlight, clBtnShadow, clBtnText, clCaptionText, clGrayText, clHighlight, clHighlightText, clInactiveBorder, clInactiveCaption, clInactiveCaptionText, clMenu, clMenuText, clScrollBar, clWindow, clWindowFrame, e clWindowText.

Exemplo: Ponto Azul - `Canvas.Pixels[x,y]:= clBlue;`

## DESENHO DE LINHAS

Para desenharmos no Delphi, usamos a propriedade **Pen** do objeto TCanvas, que nada mais é que uma caneta invisível. Podemos alterar o modo como esta caneta desenha em nossa tela.

O Delphi indica a posição desta caneta através da propriedade **PenPos**.

Uma linha é desenhada utilizando-se os métodos **MoveTo** e **LineTo** para mover a **Pen** (caneta). **MoveTo** posiciona a caneta em um determinado lugar no Formulário. **LineTo** traça uma linha, que vai da posição corrente (**PenPos**) até as coordenadas especificadas.

Copie o procedimento abaixo, inserindo-o no mesmo formulário do exemplo do ponto.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  With Canvas do
    begin
      MoveTo(0,0);
      LineTo(100,ClientHeight); {ClienteHeight:=Altura da
                                área de cliente no formulário}
      LineTo(200,150);
    end;
  end;
```

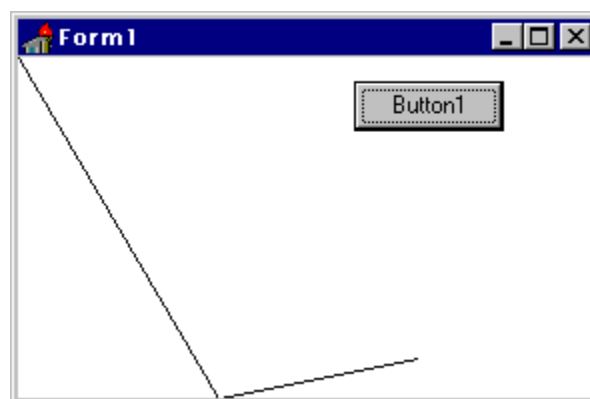
Quando queremos alterar várias propriedades ou métodos de um mesmo objeto, nós não precisamos indicar o seu nome em todas as linhas, basta usarmos a instrução **With...do**. Esta instrução significa que entre as palavras reservadas **Begin ... End**, o uso do objeto fica implícito para qualquer acesso a uma de suas propriedades ou métodos.

```
Canvas.Pen.Style:= psDot;
Canvas.Pen.Color:=clBlue;
```

ou

```
With Canvas.Pen do
  begin
    Style:= psDot;
    Color:=clBlue;
  end;
```

No exemplo acima, utilizamos este recurso com o objeto Canvas e seus métodos MoveTo e LineTo.



O tipo TCanvas usa como propriedades outros dois tipos de objetos: o TPen e TBrush. O TPen é a nossa caneta, este tipo possui propriedades que modificam a forma das linhas a serem desenhadas, são elas:

- Color - define a cor da linha
- Width - define a espessura da linha - em pixels
- Style - determina o tipo de linha.

Modifique o exemplo anterior, como mostrado a seguir:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  With Canvas do
    begin
      MoveTo(0,0);
      LineTo(100,ClientHeight);
      Pen.Color:=clBlue;
      LineTo (200,150);
    end;
end;
```

Note que a segunda linha desenhada é da cor azul.

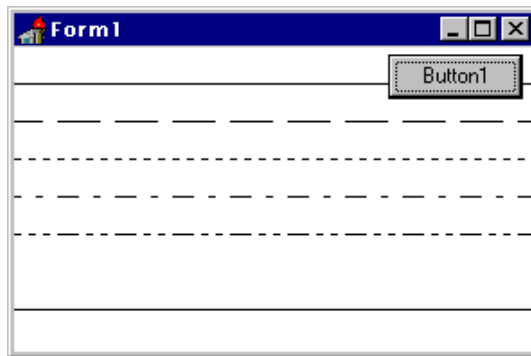
Vamos montar exemplos que usem as outras duas propriedades - Width e Style.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Estilo: array [0..6] of TPenStyle;
  i: Integer;
  SX: Integer;
  SY: Integer;
begin
  Estilo[0]:= psSolid;
  Estilo[1]:= psDash;
  Estilo[2]:= psDot;
  Estilo[3]:= psDashDot;
  Estilo[4]:= psDashDotDot;
  Estilo[5]:= psClear;
  Estilo[6]:= psInsideFrame;
  SX := ClientWidth;
  SY := ClientHeight;
  With Canvas do
    begin
      SY:=Trunc(SY/8); {a procedure Trunc, transforma um
        valor do tipo Real em tipo Inteiro}
      for i:= 0 to 6 do
        begin
          Pen.Style:= Estilo[i];
          MoveTo(0,(i*SY)+20);
          LineTo(SX,(i*SY)+20);
        end;
      end;
    end;
end;
```

Começamos declarando a matriz **Estilo** como do tipo **TPenStyle**. Nesta matriz armazenaremos todos os estilos da caneta.

**ClientWidth** e **ClientHeight**, retornam um valor do tipo inteiro indicando o tamanho da área do cliente no formulário, quando este número é dividido ele passa a ser um tipo Real, então nós precisamos convertê-lo novamente em Inteiro para que os métodos **MoveTo** e **LineTo** possam aproveitá-lo.

Antes de executar o exemplo acima, altere a propriedade **Color** de Form1 para **clWhite** (Color:=clWhite) para uma visualização melhor das linhas, teremos então todos os estilos de linha desenhados no formulário mostrado abaixo:

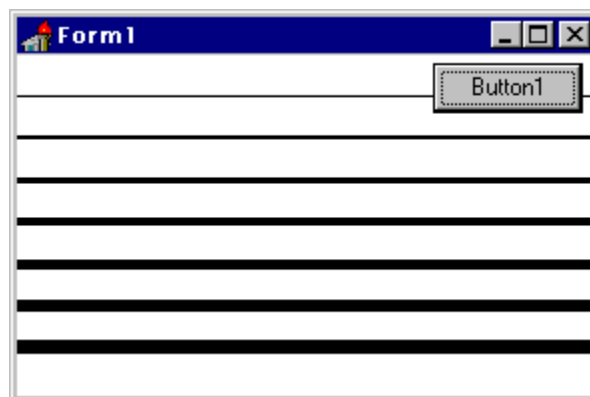


Altere o procedimento anterior, para serem mostradas as várias espessuras de linha.

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i: Integer;
    SX: Integer;
    SY: Integer;
begin
    SX := ClientWidth;
    SY := ClientHeight;
    With Canvas do
        begin
            SY:=Trunc(SY/8); {a procedure Trunc, transforma um
                           valor do tipo Real em tipo Inteiro}
            for i:= 0 to 6 do
                begin
                    Pen.Width:= i+1;
                    MoveTo(0,(i*SY)+20);
                    LineTo(SX,(i*SY)+20);
                end;
            end;
        end;
    end;

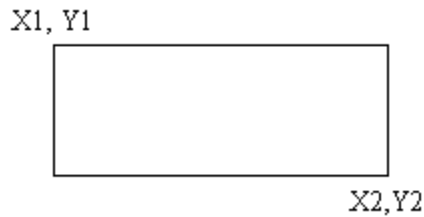
```



**DESENHO DE RETÂNGULOS**

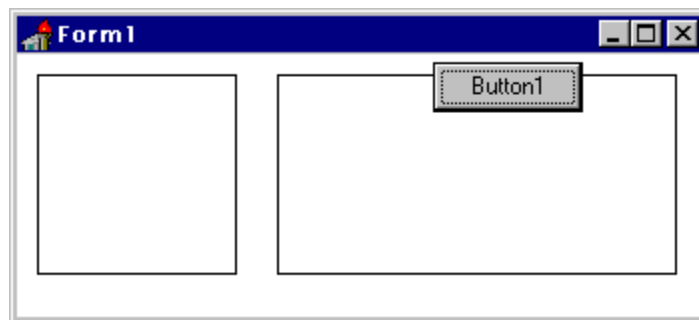
Para desenharmos retângulos usamos o método Rectangle do TCanvas. Como parâmetros são utilizadas as coordenadas do canto superior esquerdo e inferior direito do retângulo.

Canvas.Rectangle (X<sub>1</sub>, Y<sub>1</sub> , X<sub>2</sub>, Y<sub>2</sub>);



Digite o exemplo abaixo.

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  With Canvas do  
  begin  
    Rectangle (10,10,110,110);  
    Rectangle (130,10,330,110);  
  end;  
end;
```



Observe que o desenho ficou por trás do botão, isto ocorre por que o botão está acima do formulário e o desenho faz parte da tela do formulário, seria como uma borracha em cima de um papel desenhado.

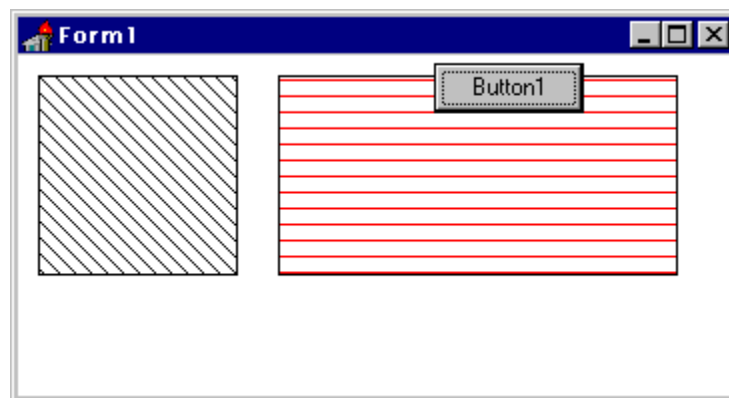
Para preenchermos estes retângulos, usamos o tipo **TBrush** que como TPen, é uma propriedade do TCanvas. As propriedades mais comuns do TBrush são:

- Color - define a cor do interior da figura
- Style - indica o padrão de preenchimento

As cores são as mesmas da Pen e os estilos são: bsSolid, bsClear, bsBDiagonal, bsFDiagonal, bsCross, bsDiagCross, bsHorizontal e bsVertical.

Faça o exemplo a seguir.

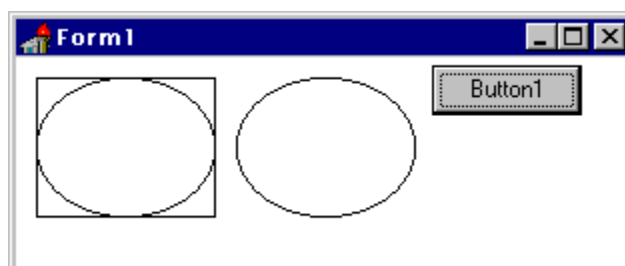
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  With Canvas do
  begin
    Brush.Style:=bsFDiagonal;
    Brush.Color:=clBlack;
    Rectangle (10,10,110,110);
    Brush.Color:=clRed;
    Brush.Style:=bsHorizontal;
    Rectangle (130,10,330,110);
  end;
end;
```



### DESENHO DE ELIPSES


O desenho de elipses é feito com o método **Ellipse**. A elipse ou o círculo são desenhados usando-se as duas coordenadas de um retângulo delimitador - imaginário - onde a figura estará inscrita. Siga o exemplo a seguir.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  With Canvas do
  begin
    Rectangle (10,10,100,80);
    Ellipse (10,10,100,80);
    Ellipse (110,10,200,80);
  end;
end;
```



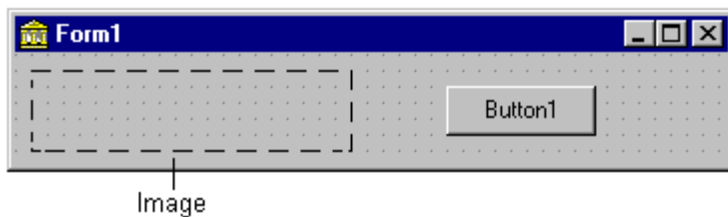


## FIGURAS

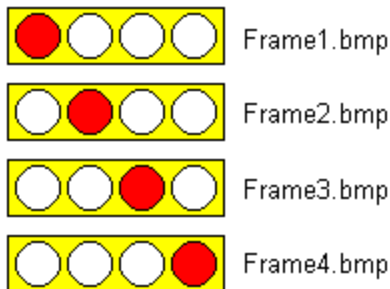
O Delphi permite a inclusão de figuras no formulário através do objeto **Image** () , onde poderemos carregar arquivos dos tipos EMF, WMF, BMP ou ICO.

As figuras são vistas como valores da propriedade **Picture** do controle **Image**. Podemos atribuir um valor para esta propriedade em tempo de projeto, ou utilizar a procedure **LoadFromFile** para carregar um arquivo de imagem durante a execução do programa.

Construiremos agora um projeto que irá carregar sucessivas figuras em um controle Image a cada clique em um botão.



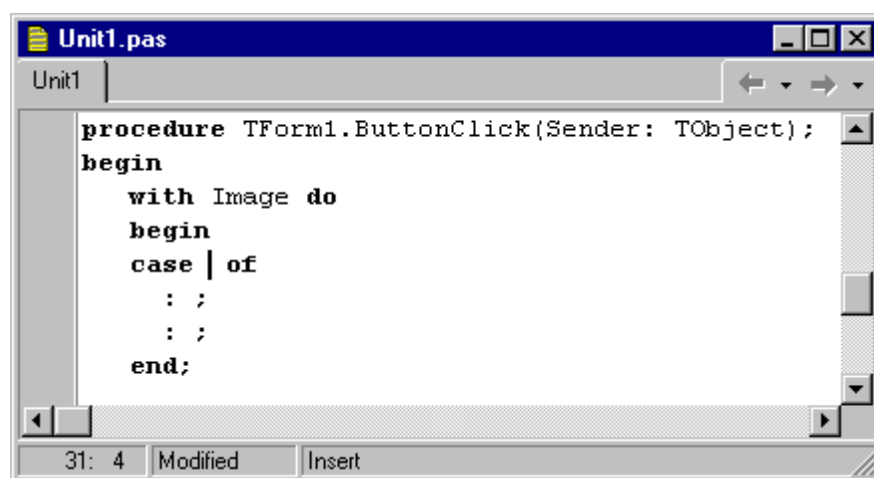
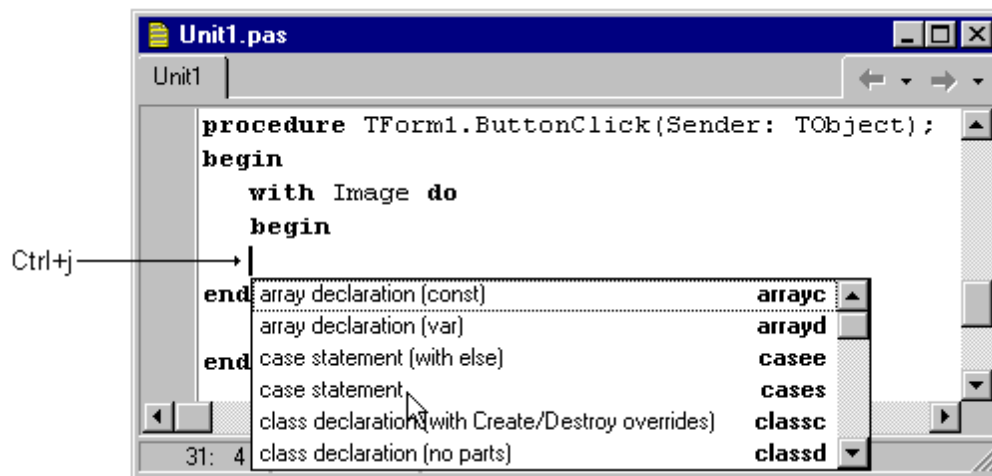
Abra o programa Paint do Windows e desenhe quatro figuras semelhantes às mostradas abaixo, salvando-as com os nomes indicados.



Estas figuras darão a impressão de movimento da "luz".

Para o código do botão utilizaremos a estrutura **case...of**. O Delphi possui modelos de código (**code templates**) caso o programador não se lembre da sintaxe exata de determinada estrutura, estes modelos podem ser editados ou adicionados outros. Para ativar o code template basta acionar **Ctrl+j**, e logo será mostra um popup com os códigos permitidos.

Entre com o código a seguir para o botão até o ponto indicado, pressionando **Ctrl+j** e escolha **case statement**.



Complete o código seguindo a listagem a seguir.

```
implementation
var
    sinal:integer;
{$R *.DFM}

procedure TForm1.ButtonClick(Sender: TObject);
begin
    with Image do
    begin
        case sinal of
            0:begin
                picture.loadfromfile ('c:\...\frame1.bmp');
                sinal:=1;
            end;
            1:begin
                picture.loadfromfile ('c:\...\frame2.bmp');
                sinal:=2;
            end;
            2:begin
                picture.loadfromfile ('c:\...\frame3.bmp');
                sinal:=3;
            end;
        end;
    end;
end;
```

```

        3:begin
            picture.loadfromfile ('c:\...\frame4.bmp');
            sinal:=0;
        end;
    end;
end;
end;

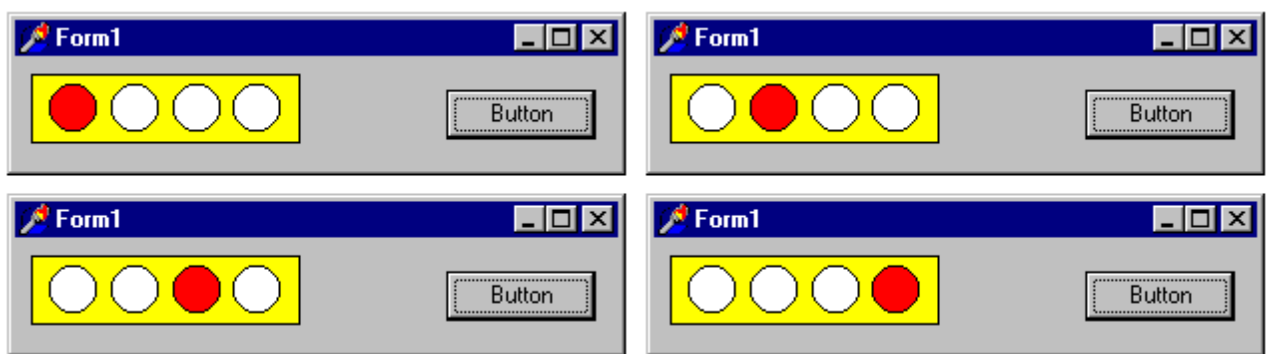
```

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    sinal:=0;
end;

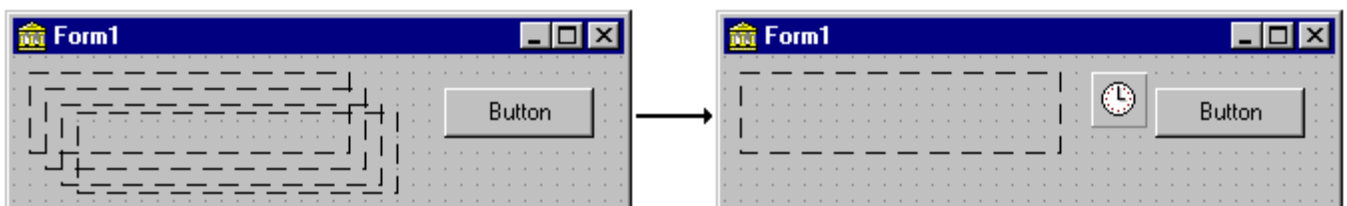
```

Executando o programa:



Ao invés de cada figura ser carregada do disco quando necessitarmos, poderemos incluir todas de uma só vez no aplicativo.

Vamos então alterar o programa anterior copiando o controle **Image** e colando mais outros três Image, todos na mesma posição. Incluindo também um controle **Timer** para que a mudança ocorra automaticamente.



Altere a propriedade **Enabled:=False** do Timer. E para os controle Image, **Visible:=False** e na propriedade Picture carregue as figuras utilizadas anteriormente - Frame1 a 4.

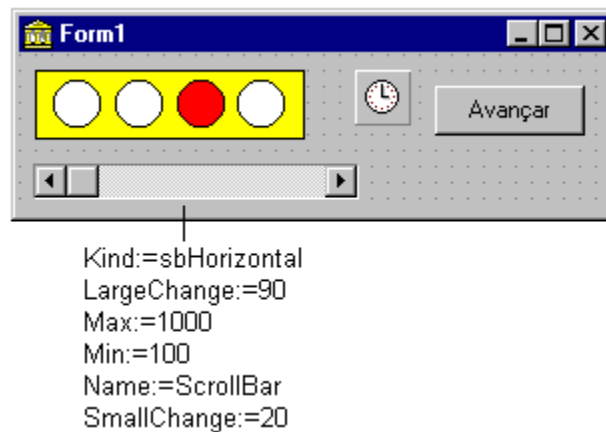
Digite o código a seguir:

```
procedure TForm1.ButtonClick(Sender: TObject);
begin
  if Button.caption='Parar' then
  begin
    Button.caption:='Automático';
    Timer.enabled:=False;
    exit;      {sai da procedure, sem executar as linhas seguintes}
  end;
  Timer.enabled:=true;
  Button.caption:='Parar';
end;
```

```
procedure TForm1.TimerTimer(Sender: TObject);
begin
  case sinal of
    0:begin
      Image1.Visible:=True;
      Image4.Visible:=False;
      sinal:=1;
    end;
    1:begin
      Image2.Visible:=True;
      Image1.Visible:=False;
      sinal:=2;
    end;
    2:begin
      Image3.Visible:=True;
      Image2.Visible:=False;
      sinal:=3;
    end;
    3:begin
      Image4.Visible:=True;
      Image3.Visible:=False;
      sinal:=0;
    end;
  end;
end;
```

Como próximo passo vamos incluir um controle **ScrollBar** () que irá controlar a variação da propriedade Interval do Timer.

Altere as propriedades da ScrollBar de acordo com a figura a seguir:



Onde:

**Kind** - especifica se a scrollbar será horizontal ou vertical.

**LargeChange** - determina o quanto muda a posição quando o usuário clica a barra de rolagem em um de seus lados.

**Max** - especifica o valor máximo da posição.

**Min** - especifica o valor mínimo da posição.

**SmallChange** - determina o quanto muda a posição quando o usuário clica nas setas da barra de rolagem.

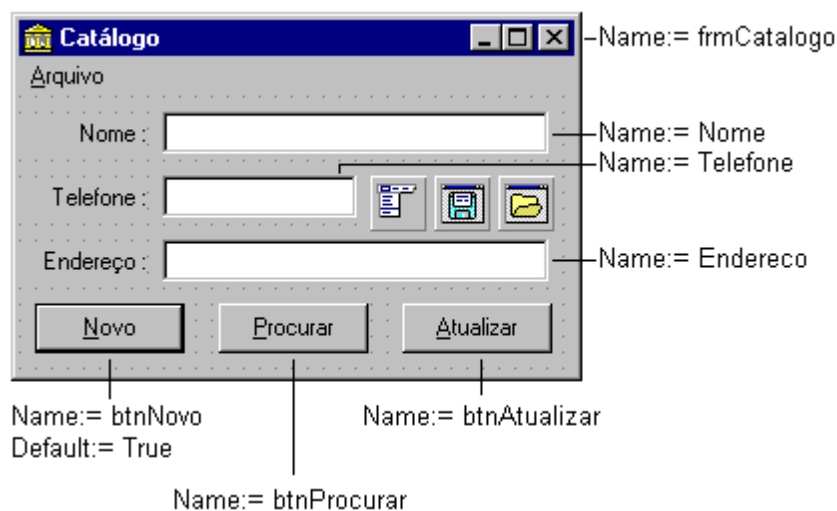
Digite a procedure a seguir para o evento **OnChange** da ScrollBar, que ocorre quando mudamos a posição da barra de rolagem.

```
procedure TForm1.ScrollBarChange(Sender: TObject);  
begin  
    Timer.interval:=ScrollBar.position;  
    //A propriedade position contém o valor da posição da barra de  
    rolagem, e varia entre 100 e 1000 neste exemplo.  
end;
```

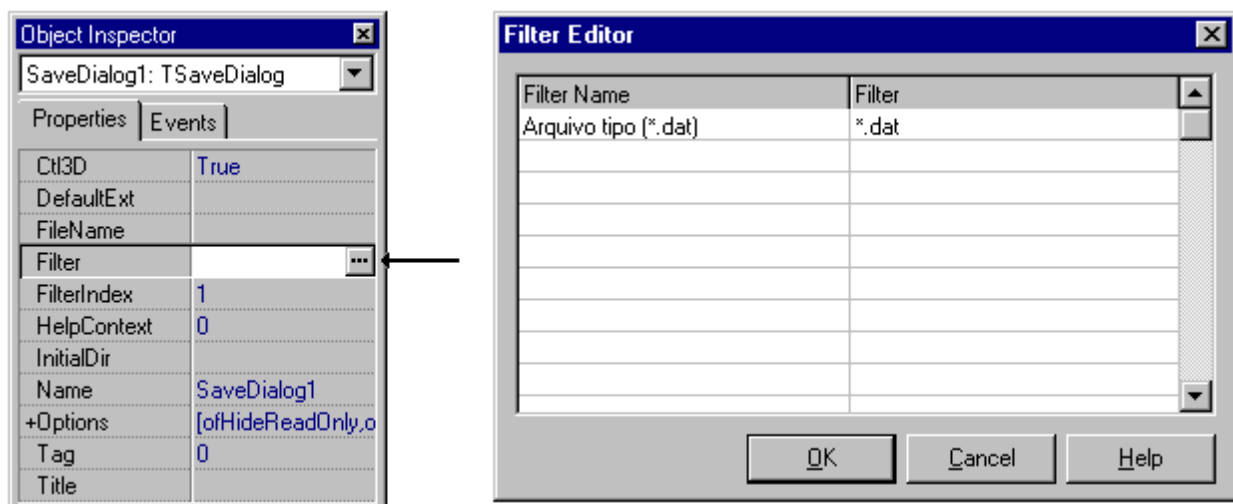
## EXEMPLO V - CATÁLOGO

Neste projeto de catálogo telefônico, trabalharemos com acesso a arquivos do tipo aleatório. Ele será constituído por três Units e dois Formulários. Duas Units serão associadas aos formulários e a outra Unit conterá apenas códigos de acesso ao disco (HD) sem estar associada a nenhum formulário.

Inicie um novo projeto e crie o formulário, inserindo e modificando as propriedades dos objetos, como a figura abaixo.




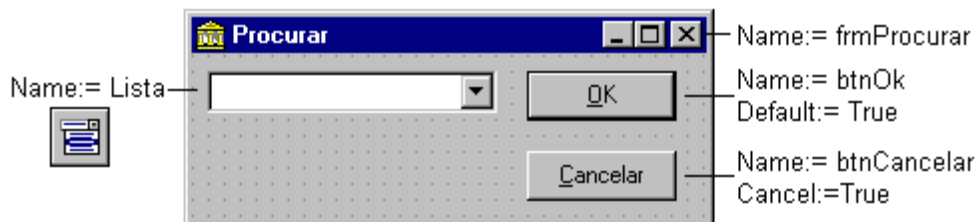
Altere a propriedade **Filter** do controle **SaveDialog**. Será então mostrada a janela **Filter Editor**, edite-a como mostra o modelo abaixo.



Dê um duplo clique no controle MainMenu e edite o menu.



Deveremos inserir um novo formulário ao nosso projeto que será utilizado para selecionar um determinado registro do nosso arquivo de dados. Para inserir um novo formulário ao projeto, escolha a opção **New Form** do menu **File** ou selecione o botão (  ) na barra de ferramentas. Monte este novo Formulário como mostrado abaixo.



Este novo formulário irá apresentar ao usuário os nomes de pessoas - registros no arquivo - contidos no catálogo através do componente **ComboBox** (caixa combinada).

O ComboBox, é um controle de edição associado à uma lista contendo itens disponíveis para a escolha do usuário. O usuário poderá tanto digitar uma opção no quadro de edição, quanto escolher uma opção fornecida pela lista associada.

A propriedade Style, determina o tipo de ComboBox que iremos trabalhar, que pode ser:

**csDropDown** - Cria uma lista drop-down com uma caixa de texto, para entrada de texto manualmente. Todos os itens são Strings de qualquer comprimento.

**CsSimple** - Cria uma caixa de texto associada a uma caixa de lista suspensa. Esta lista não aparecerá em tempo de execução a menos que o ComboBox seja dimensionado para acomodá-la.


**CsDropDownList** - Cria uma lista drop-down com uma caixa de texto, mas o usuário não poderá entrar com texto manualmente.

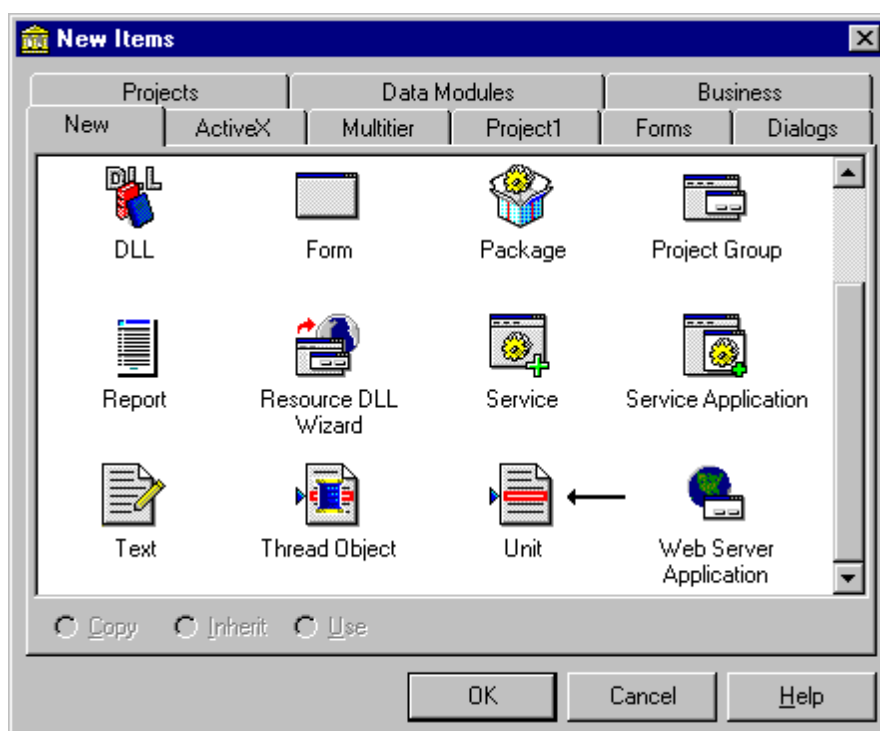
**csOwnerDrawFixed** - Cria uma lista drop-down com uma caixa de texto, para entrada de texto manualmente. Mas cada item do ComboBox terá seu comprimento em caracteres determinado pela propriedade ItemHeight.

**CsOwnerDrawVariable** - Cria uma lista drop-down com uma caixa de texto, para entrada de texto manualmente. Os itens neste ComboBox podem ter comprimento variável.

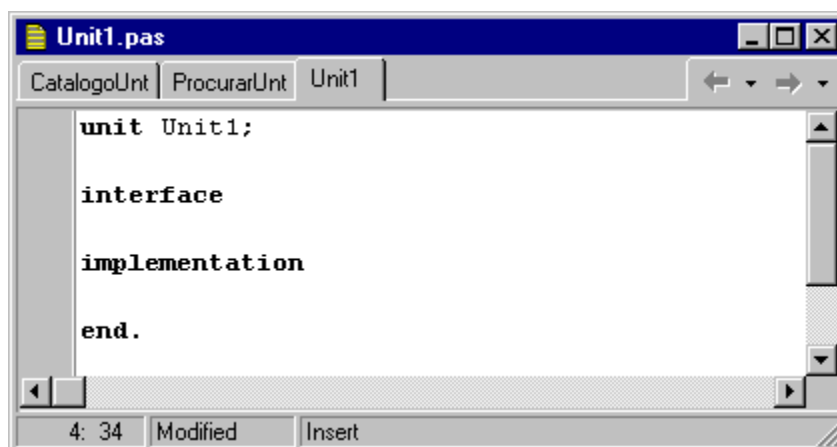
Vamos alterar a propriedade **Style** do nosso quadro combo, para **csDropDownList**, pois não queremos que o usuário altere o conteúdo do combo. Evitando que o usuário entre com um nome de pessoa inexistente.

### **INSERINDO UMA NOVA UNIDADE**

Uma Unit pode estar vinculada a um formulário ou não, nós iremos criar uma Unit não vinculada a nenhum formulário. Escolha a opção **New...** do menu **File** ou o botão (  ) na barra de ferramentas e aparecerá a janela **New Items**, escolha **Unit**.



O Delphi criará uma nova Unit não vinculada a nenhum formulário, já com o cabeçalho pronto. Onde iremos digitar as rotinas de acesso ao disco.





## TIPOS DE DADOS

No Delphi nós podemos criar nossos próprios tipos de dados, para fazer isto, declare o seu tipo de dado dentro de um bloco **Type**, antes do bloco de definição de variáveis **Var**:

```
type
    TMeusValores = ShortInt;

var
    Venda, Custo : TMeusValores;
```

É interessante definir um tipo de variável, porque quando tivermos uma família de variáveis relacionadas e for necessário mudar o tipo de todas elas, basta alterar a declaração de definição de tipo destas variáveis:

```
type
    TMeusValores = LongInt;
```

Venda e Custo eram variáveis ShortInt, após a nova definição passam a ser LongInt.

## REGISTROS

Para armazenar os dados de uma pessoa (Nome, Endereço e Telefone), usamos um registro. Registro é um tipo de dado que reúne alguns itens de tipos diferentes, nosso tipo registro terá o nome de TPessoasReg. Em um projeto podem existir várias variáveis do tipo TPessoasReg.

Declare o novo tipo na seção Interface da nova Unit (DiscoUnt - nome salvo em disco).

```
unit DiscoUnt;

interface

type
    {registro com dados pessoais}
    TPessoasReg=record
        CNome: string[30];
        CTelefone: string[15];
        CEndereco: string[40];
    end;

    TPessoas = file of TPessoasReg;

var
    PesArq: TPessoas;
```

O Delphi possui um tipo denominado **arquivo**, que permite escrever e ler arquivos em disco. Para declarar um tipo arquivo, associamos o nome da variável a um tipo de dado previamente existente. No nosso caso: TPessoasReg.

Na listagem acima definimos além do tipo TPessoasReg, o tipo TPessoas, definindo-o como um tipo de arquivo formado por registros do tipo TPessoasReg.

Os campos são declarados como String de tamanhos fixos pois, caso não fossem declarados os tamanhos, eles iriam ocupar o espaço de 255 Bytes.

Logo após, informamos ao Delphi que existe um tipo de arquivo composto por esse registro, e também declaramos uma variável do tipo desse arquivo de registros, essa variável conterá o nome do arquivo aberto em disco.

Algumas procedures e functions, precisam de parâmetros para trabalharem, nós declaramos esses parâmetros no cabeçalho da procedure, indicando um nome de referência, o seu tipo de dado e o prefixo **var**. Primeiro declaramos as procedures na seção Interface e depois implementamos na seção Implementation.

```
procedure AbrirArq(var A:TPessoas; var NomeArq:string;
                  var N:boolean);
procedure FecharArq(var A:TPessoas);
procedure GravarArq(var A:TPessoas;var RegN:integer;
                  var Reg:TPessoasReg);
procedure LerArq(var A:TPessoas;var RegN:integer;
                 var Reg:TPessoasReg);

implementation

procedure AbrirArq(var A:TPessoas; var NomeArq:string;
                  var N:boolean);
begin
  Assign (A, NomeArq);
  if N then
    Rewrite (A)
  else
    Reset (A);
end;

procedure FecharArq (var A:TPessoas);
begin
  Close (A);
end;

procedure GravarArq (var A:TPessoas; var RegN:integer;
                    var Reg:TPessoasReg);
begin
  Seek (A,RegN);
  Write (A,Reg);
end;

procedure LerArq (var A:TPessoas; var RegN:integer;
                 var Reg:TPessoasReg);
begin
  Seek (A,RegN);
```

```
Read (A, Reg);  
end;  
  
end.
```

Antes de abrir um arquivo para leitura e gravação, nós temos que associá-lo a uma variável do mesmo tipo dos registros contidos no arquivo em disco. Para isso usamos a procedure **Assign**. **Assign** diz ao Delphi que todas as operações com a variável terão ligação com um arquivo em particular.

**Assign**(variável, nome do arquivo);

Após a associação, abrimos o arquivo, que pode ser de duas formas:

- 1) **Reset** - abre um arquivo existente para leitura ou gravação.
- 2) **Rewrite** - abre um arquivo novo para leitura ou gravação. Caso já exista o arquivo, todos os dados anteriores serão perdidos.

**Reset**(nome do arquivo);  
**Rewrite**(nome do arquivo);

Para gravar ou ler um registro, usamos as funções **Write** e **Read**, informando a variável do arquivo e em qual variável estará ou conterá o registro.

**Write**(PessArq, Reg);  
**Read**(PessArq, Reg);

### **PONTEIRO DE REGISTROS**

O **ponteiro** do arquivo indica qual registro será afetado com a próxima operação de leitura ou gravação. Sempre que um arquivo é aberto, o ponteiro é posicionado no início do arquivo. Cada operação de leitura ou gravação, move o ponteiro um registro à frente. Ou seja, se o registro 3 for lido, o ponteiro estará apontando para o registro 4, e a próxima operação será realizada com o registro 4.

Para acessar um registro aleatoriamente, usamos a procedure **Seek**. Esta procedure posiciona o ponteiro no registro que quisermos, antes de uma leitura ou gravação.

**Seek**(PessArq, RegN);

Os registros em um arquivo são numerados seqüencialmente a partir do registro 0. Existem mais duas funções úteis para manipulação de arquivos, **FileSize** e **FilePos**. A primeira indica quantos registros o arquivo tem, e a segunda indica a posição do ponteiro.

Para ligarmos as Units, devemos colocar o nome das Units acessadas na seção **uses**. Uma vez declarada uma unidade na seção **uses** de outra unidade em que esteja trabalhando, você poderá referenciar qualquer coisa na seção de interface da unidade nomeada. Feita esta ligação, poderemos acessar as variáveis e procedures da outra unidade.

O nosso formulário frmCatalogo (CatalogoUnt), terá acesso aos procedimentos das unidades ProcurarUnt e DiscoUnt. Então, declaramos estas duas unidades logo após as unidades já colocadas pelo Delphi.

```
unit Catalogo;  
  
interface  
  
uses  
Windows, Messages, SysUtils, Classes, Graphics, Controls,  
Forms, Dialogs, Menus, StdCtrls,  
ProcurarUnt, DiscoUnt;
```

Declare as variáveis que iremos utilizar na unidade CatalogoUnt, como mostrado abaixo.

```
private  
    { Private declarations }  
    ArqNovo, ArqAtual: string;  
    Novo, ArquivoAberto: boolean;  
public  
    { Public declarations }  
end;
```

Comece a implementação dos procedimentos associados aos objetos e seus eventos, seguindo as listagens abaixo.

```
procedure TfrmCatalogo.Sair1Click(Sender: TObject);  
begin  
    Close;  
end;
```

```
procedure TfrmCatalogo.Novo1Click(Sender: TObject);  
begin  
    Nome.Text := '';  
    Telefone.Text := '';  
    Endereco.Text := '';  
    Novo := True;  
    ArqNovo := 'Temp.dat';  
    ArqAtual := ArqNovo;  
    DiscoUnt.AbrirArq (PesArq, ArqNovo, Novo);  
    ArquivoAberto := True;  
end;
```

```
procedure TfrmCatalogo.Abrir1Click(Sender: TObject);  
begin  
  Nome.Text:='';  
  Telefone.Text:='';  
  Endereco.Text:='';  
  if OpenDialog1.Execute then  
    begin  
      ArqAtual:=OpenDialog1.FileName;  
      Novo:=False;  
      DiscoUnt.AbrirArq (PesArq, ArqAtual, Novo)  
    end  
  else  
    Exit;  
  ArquivoAberto:=True;  
end;
```

Para se referir a um procedimento em outra unidade, nós informamos primeiro o nome da unidade e depois o nome do procedimento: DiscoUnt.AbrirArq(...);.

```
procedure TfrmCatalogo.btnNovoClick(Sender: TObject);  
var  
  Reg: DiscoUnt.TPessoasReg;  
  RegN: integer;  
begin  
  Nome.SetFocus;  
  if ArquivoAberto=false then  
    begin  
      ShowMessage ('Abra primeiro um arquivo');  
      exit;  
    end;  
  if btnNovo.Caption='&Novo' then  
    begin  
      Nome.Text:='';  
      Telefone.Text:='';  
      Endereco.Text:='';  
      btnNovo.Caption:='A&dicionar';  
    end  
  else  
    begin  
      with Reg do  
        begin  
          CNome:=Nome.Text;  
          CTelefone:=Telefone.Text;  
          CEndereco:=Endereco.Text;  
        end;  
      frmProcurar.Lista.Items.Add (Nome.Text);  
      RegN:=  FileSize(PesArq);  
      DiscoUnt.GravarArq (PesArq, RegN, Reg);  
      btnNovo.Caption:='&Novo';  
    end;  
  end;
```

```
procedure TfrmCatalogo.btnAtualizarClick(Sender: TObject);  
var  
    Reg : TPessoasReg;  
    RegN :integer;  
begin  
    with Reg do  
        begin  
            CNome:=Nome.Text;  
            CTelefone:=Telefone.Text;  
            CEndereco:=Endereco.Text;  
        end;  
    frmProcurar.Lista.Items.Add (Nome.Text);  
    RegN:= FilePos(PesArq)-1;  
    DiscoUnt.GravarArq (PesArq, RegN, Reg);  
end;
```

```
procedure TfrmCatalogo.FormCreate(Sender: TObject);  
begin  
    ArquivoAberto:=False;  
end;
```

```
procedure TfrmCatalogo.Salvar1Click(Sender: TObject);  
begin  
    if SaveDialog1.Execute then  
        begin  
            ArqNovo:=SaveDialog1.FileName;  
            DiscoUnt.FecharArq (PesArq);  
            RenameFile(ArqAtual,ArqNovo);{a função RenameFile,}  
            Novo:=False;                {renomeia um arquivo}  
            DiscoUnt.AbrirArq (PesArq, ArqNovo, Novo);  
        end;  
end;
```

```
procedure TfrmCatalogo.btnProcurarClick(Sender: TObject);  
begin  
    frmProcurar.ShowModal;  
end;
```

Para mostrar outro Formulário, usamos o método **Show** ou **ShowModal**. Show abre uma janela que pode ser alternada entre outras janelas do aplicativo, enquanto ShowModal não aceita uma troca da janela ativa. Com ShowModal o usuário só poderá ir para outra janela, após ter fechado a janela Modal.

Comece a construção do código para o formulário frmProcurar, informando ao Delphi que ela irá acessar a unidade **DiscoUnt**.

```
unit ProcurarUnt;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls,  
  Forms, Dialogs, StdCtrls,DiscoUnt;
```

Declare as variáveis **ReN** e **Reg**.

```
private  
  { Private declarations }  
  RegN: integer;  
  Reg: TPessoasReg;
```

Siga a listagem abaixo para os eventos associados.

```
procedure TfrmProcurar.FormActivate(Sender: TObject);  
begin  
  Lista.Clear;           {limpa o conteúdo do ComboBox}  
  Seek (PesArq,0);  
  while Not Eof (PesArq) do {enquanto não chegar ao final  
                           do arquivo, faça}  
  begin  
    Read(PesArq,Reg);  
    Lista.Items.Add (Reg.CNome);  
  end;  
end;
```

```
procedure TfrmProcurar.btnCancelarClick(Sender: TObject);  
begin  
  Close;  
end;
```

```

procedure TfrmProcurar.btnOkClick(Sender: TObject);
begin
    RegN:=Lista.ItemIndex;
    DiscoUnt.LerArq (PesArq,RegN,Reg);
    With frmCatalogo do
        begin
            Nome.Text:=Reg.CNome;
            Endereco.Text:=Reg.CEndereco;
            Telefone.Text:=Reg.CTelefone;
        end;
    Close;
end;

```

Para preencher um quadro combo com dados, temos duas possibilidades:

- 1) Usar o método **Add** para a sua propriedade **Items**. Em tempo de execução.
- 2) Editar a propriedade **Items** em tempo de projeto.

Por fim, declare a unidade **CatalogoUnt** na seção **Implementation**. Porque a unidade **ProcurarUnt** também se referencia à **CatalogoUnt**, mas se **CatalogoUnt** for declarada na seção **uses** da interface, o Delphi gerará um erro de referência circular.

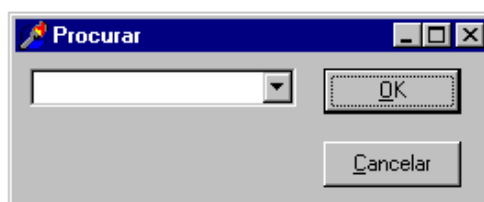
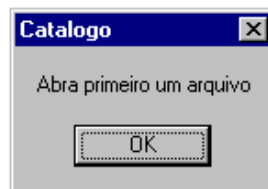
```

implementation

uses
    CatalogoUnt;

{$R *.DFM}

```





## BANCO DE DADOS SIMPLES

---

Um banco de dados pode ser definido como um conjunto unificado de informações compartilhadas por pessoas autorizadas de uma organização. A função de um Banco de Dados é permitir o armazenamento e a recuperação da informações necessárias para as pessoas da organização tomarem decisões.

Um Banco de Dados é formado por um conjunto de arquivos de dados, também chamados de tabelas. Em uma tabela temos, linhas e colunas. Onde uma linha contém informações a respeito de um item em particular, e a coluna possui uma parte das informações de uma linha. A figura a seguir ilustra uma tabela de Clientes.

Código	Nome	Telefone
001	Aldair	554-8788
002	Marina	879-9687
005	Aline	572-2258
012	José	7070-8580
150	Márcio	446-3987

A tabela mostrada acima possui os campos ou colunas: Código, Nome e Telefone. E em cada linha temos a identificação de um cliente em particular.

Cada tabela em um banco de dados possui um campo que identifica unicamente cada registro. O conjunto de atributos ou campos utilizados para gerar este índice é conhecido como **chave primária**. Nós podemos escolher qualquer campo de uma tabela para ser uma chave primária, sempre levando em consideração que ela deve ser curta. Por exemplo, usar um campo numérico é melhor do que um campo formado por Strings. Como a chave primária determina a singularidade de um registro, quando se usa um string não poderemos ter dois registros com o mesmo nome de cliente.

Na tabela do exemplo acima, o mais sensato é determinar o campo Código como nossa chave primária.

Além de determinarmos os campos e a chave primária da nossa tabela, também teremos que definir o **tipo de dado**, **comprimento**, **formato** e o **domínio** para cada campo desta tabela.

**Tipo de Dado:** refere-se ao conteúdo do dado, podendo ser numérico, alfabético, data, hora ou cadeia de caracteres longa ou curta.

**Comprimento:** refere-se ao tamanho ou número máximo de posições que poderá assumir o valor ou conteúdo de cada dado.

**Formato:** refere-se à forma com que os dados deverão ser editados ou apresentados, definindo-se as posições de símbolos.

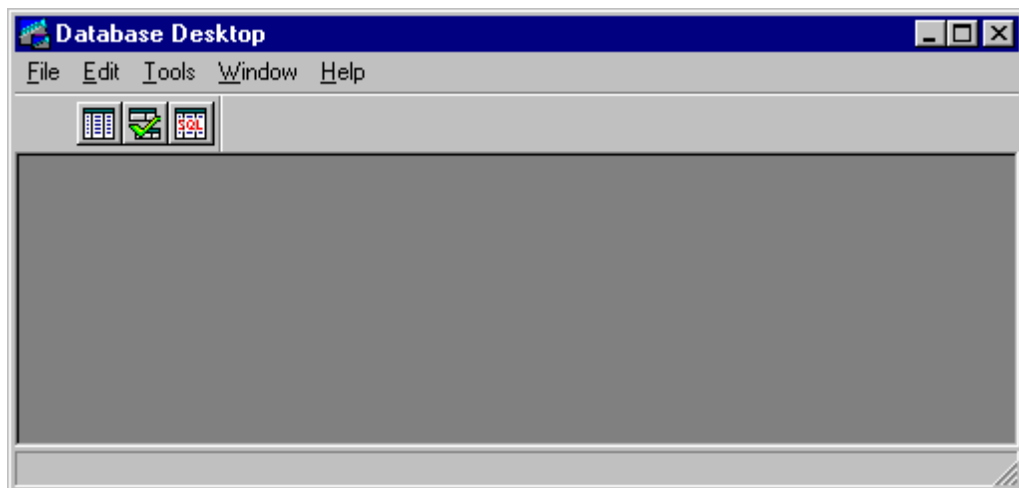
**Domínio:** especifica se os valores dos dados pertencem a uma lista de valores pré definidos, podendo estar em uma lista, ou satisfazer uma regra.

**DATABASE DESKTOP**

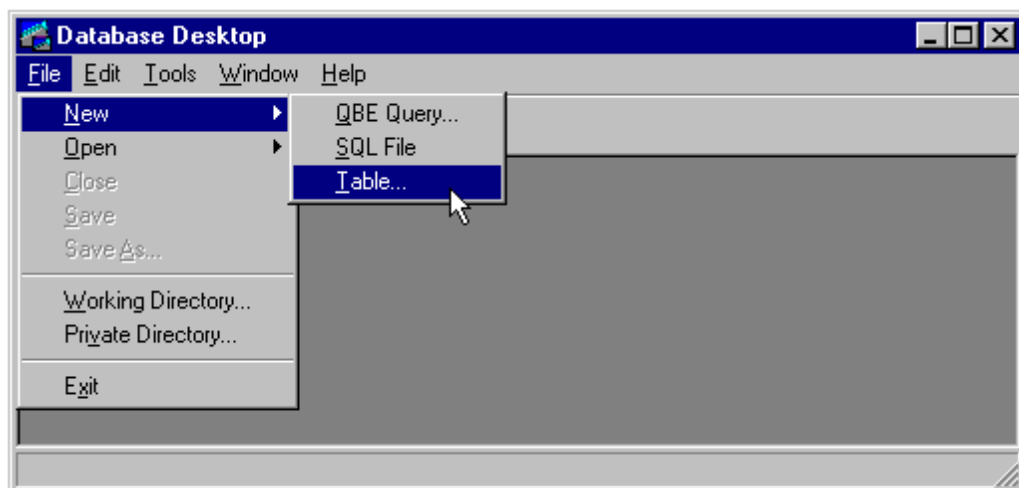
No grupo de programas do Delphi, encontramos o utilitário Database Desktop, que permite a criação e manipulação de banco de dados de diversos formatos.

O nosso próximo exemplo, gerenciará duas tabelas. Uma com o cadastro dos clientes e outra com as vendas realizadas para estes clientes.

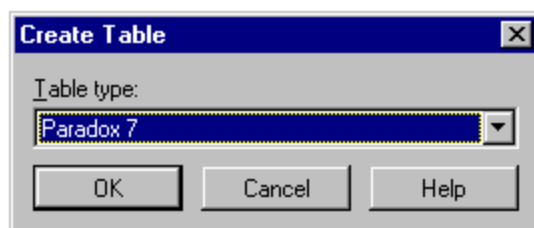
Execute o Database Desktop, selecionando a opção **Database Desktop** no menu **Tools**.



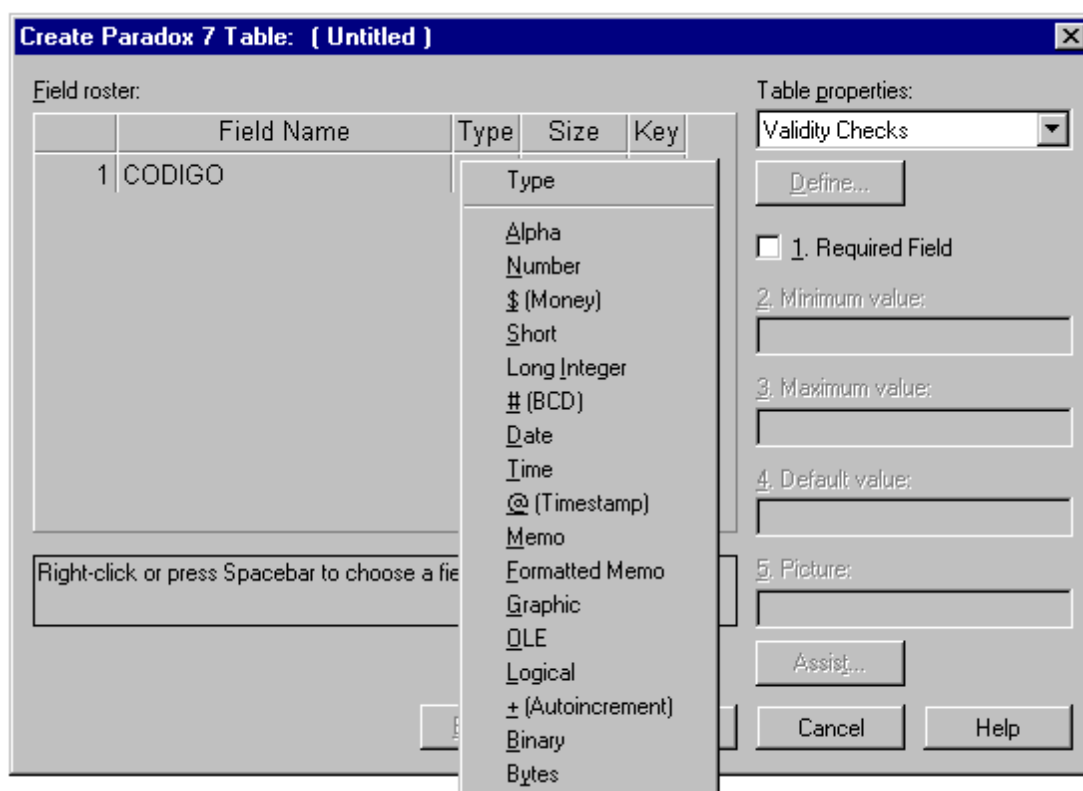
Abra uma nova tabela como mostra a figura a seguir.



Escolha o tipo Paradox 7. Aparecendo então a janela Create Paradox 7 Table.



Digite o nome para o primeiro campo (CODIGO) e mude para tipo de campo (Type) usando a tecla Tab. Com o campo Type selecionado, de um clique com o botão direito do mouse nele, para ser exibido o menu pop-up com todos os valores para o tipo de campo. As letras sublinhadas correspondem à letra de atalho para cada tipo de dado. O nosso campo CODIGO será definido como tipo Alpha. Apesar de ser um campo com números nós não iremos realizar nenhum cálculo matemático com ele, por isso o definimos como Alpha.



O campo CODIGO será a nossa chave primária para a identificação dos clientes na tabela Clientes. Para definir um campo como chave primária, pressione qualquer tecla quando o campo Key estiver selecionado, marcando este campo com um \* - asterisco.

**Create Paradox 7 Table: (Untitled)**

Field roster:

	Field Name	Type	Size	Key
1	CODIGO	A	3	*

Dica de operação → Double-click or press any character to set or remove key. Keyed fields must be the top fields in the Field Roster.

Table properties:

Validity Checks

Define...

☐ 1. Required Field

2. Minimum value:

3. Maximum value:

4. Default value:

5. Picture:

Assist...

Borrow... Save As... Cancel Help

Dê continuidade entrando com outros campos, seguindo o modelo apresentado abaixo.

**Create Paradox 7 Table: (Untitled)**

Field roster:

	Field Name	Type	Size	Key
1	CODIGO	A	3	*
2	NOME	A	30	
3	ENDERECO	A	30	
4	CIDADE	A	20	
5	ESTADO	A	2	
6	CEP	A	9	
7	TELEFONE	A	15	
8				

Enter a field name up to 25 characters long.

Table properties:

Validity Checks

Define...

☐ 1. Required Field

2. Minimum value:

3. Maximum value:

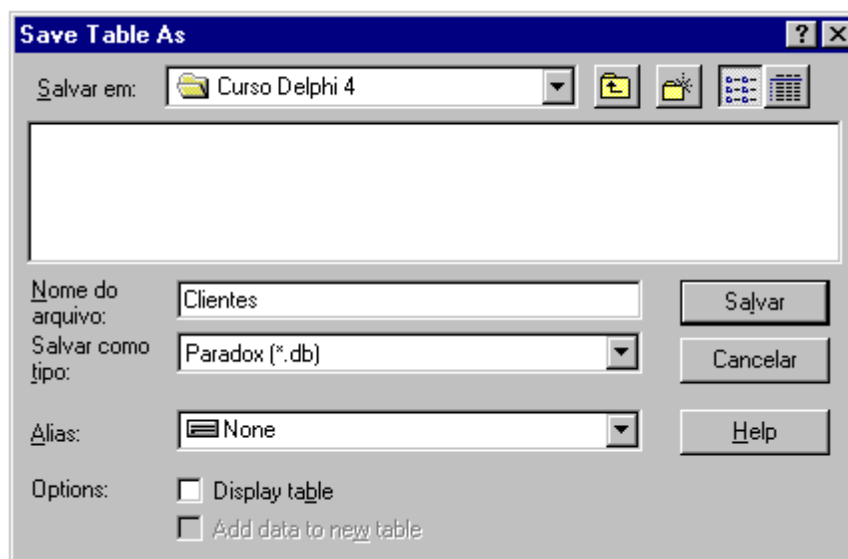
4. Default value:

5. Picture:

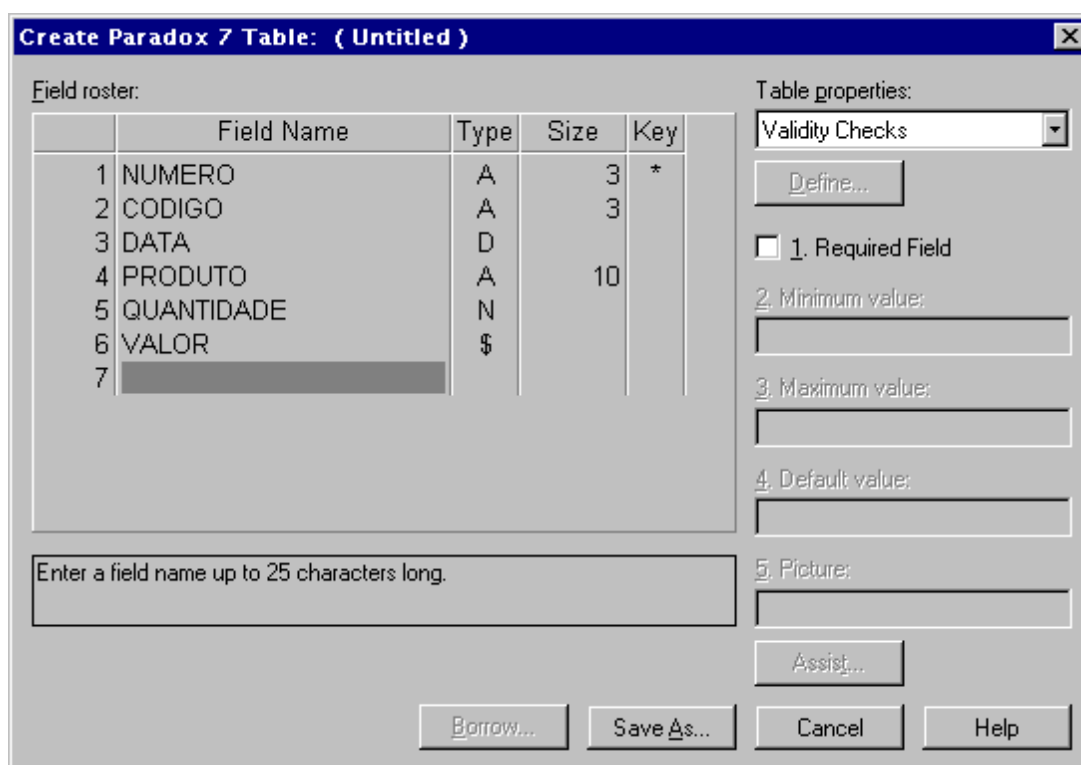
Assist...

Borrow... Save As... Cancel Help

Salve-a como Clientes.db.



Logo após, abra uma nova tabela no mesmo formato, e construa-a como mostrado abaixo.

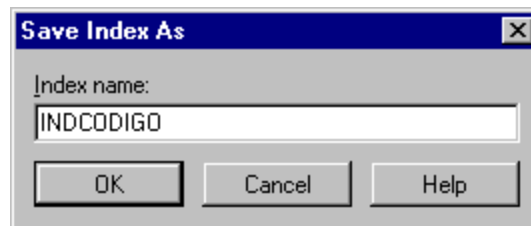
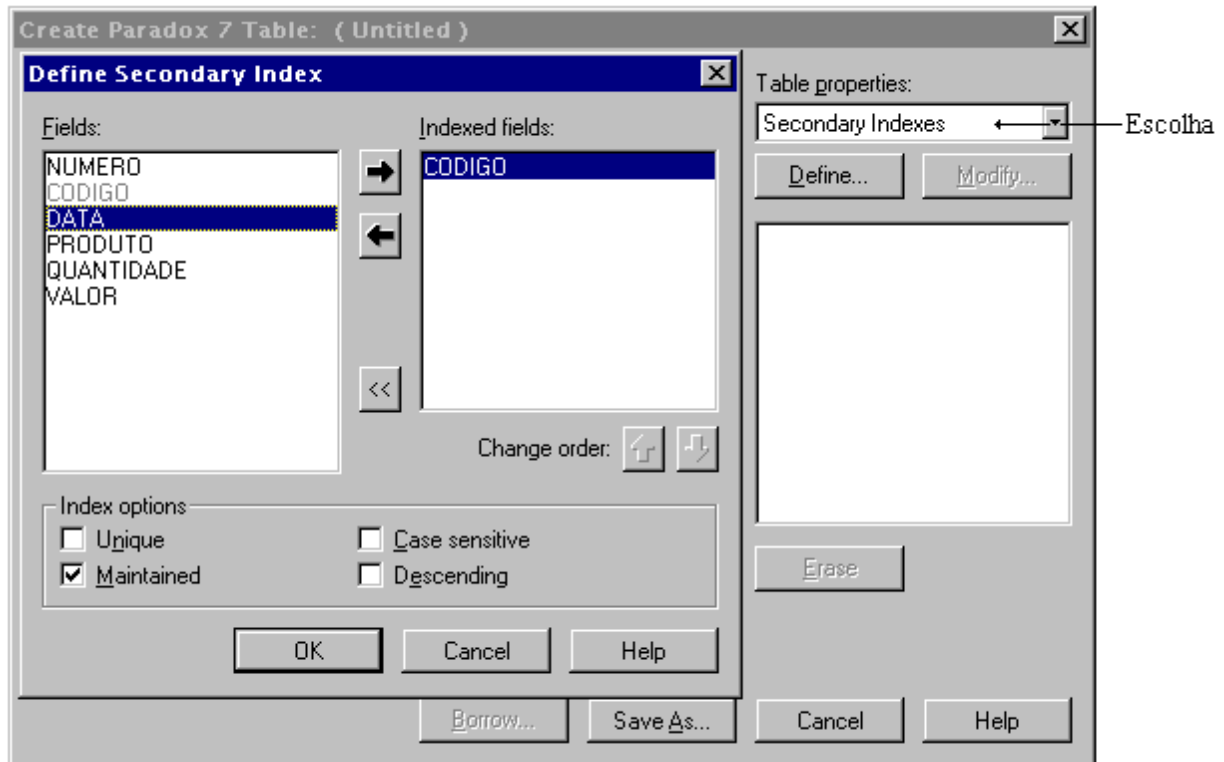


Para que a tabela Vendas se relacione com a tabela Clientes, deveremos definir o campo CODIGO como índice secundário na tabela Vendas.

No ComboBox **Table properties**, selecione a opção **Secondary Indexes** e clique no botão **Define...**, aparecendo o quadro de diálogo **Define Secondary Index**.


Na janela **Define Secondary Index**, selecione o campo CODIGO e dê um clique na seta para introduzi-lo como campo índice, e finalmente dê um clique no botão OK para aceitar a opção.

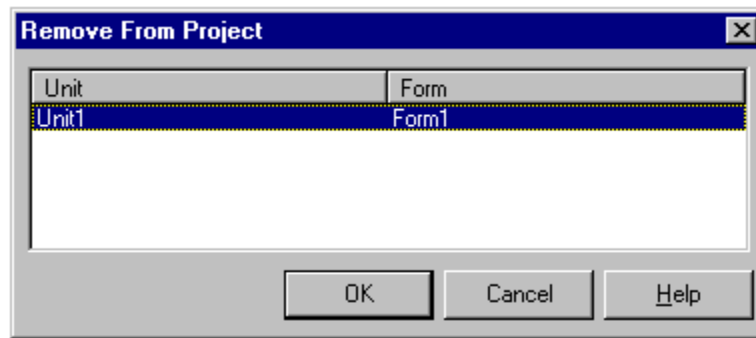
Será então solicitado um nome para este índice, digite INDCODIGO.



Salve a tabela como Vendas.db. Saia do Database Desktop, através do menu **File**, opção **Exit**.

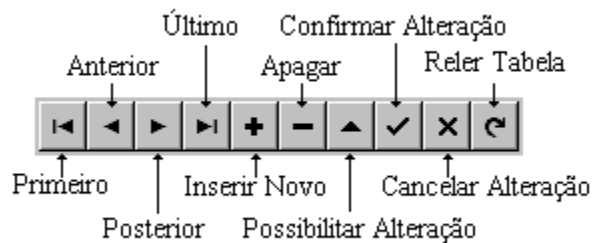
O Delphi possui um assistente para a construção de formulários de acesso a Banco de Dados (**Form Wizard**), nós iremos utilizá-lo para construirmos o nosso projeto, que utiliza as tabelas de Clientes e Vendas construídas anteriormente.

Volte ao Delphi, e exclua o formulário Form1 que o Delphi cria junto com um novo projeto. No menu **Project**, escolha a opção **Remove from Project...**, ou o botão  na barra de ferramentas, e exclua Form1 não salvando o arquivo. Pois o **Form Wizard** criará um novo formulário automaticamente.

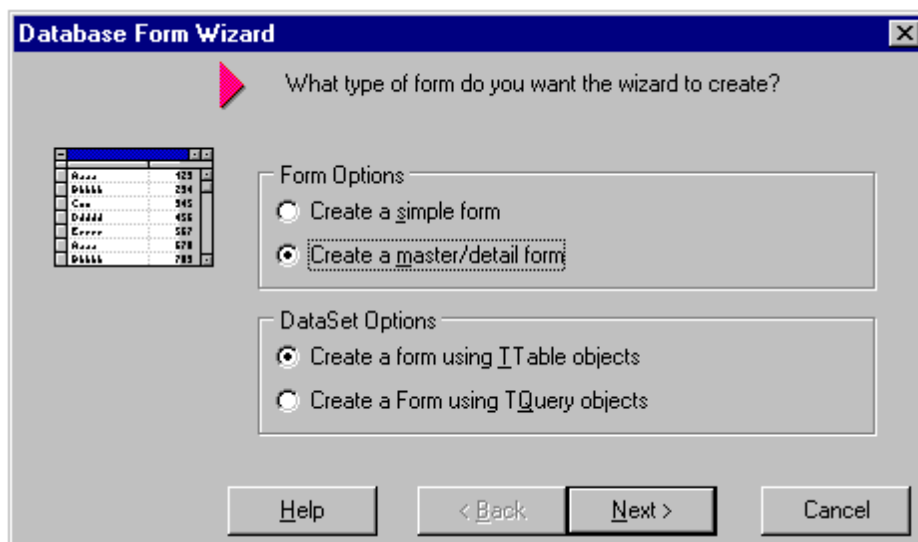


Excluído o formulário, inicie o wizard escolhendo a opção **Form Wizard...** do menu **Database**. O Database Form Wizard é um assistente para a criação de Formulários envolvendo banco de dados. Este assistente ajudará na definição dos campos a serem mostrados e o layout do formulário, além de incluir um componente DBNavigator, que será usado para a navegação entre os registros das tabelas.

#### DBNavigator



O nosso primeiro formulário terá duas divisões, uma para cada tabela. A primeira conterá os dados relativos a um cliente, e a segunda as compras realizadas por este cliente. Inicie o Form Wizard, seguindo as opções mostradas nas figuras a seguir; pressionando o botão **Next** para continuar com o assistente.



Estas opções criam um formulário com duas divisões, usando componentes **TTable**. Um componente TTable realiza a conexão entre o formulário e uma tabela do banco de dados. Ele é um componente visível somente em tempo de projeto - semelhante ao componente TTimer.

Inclua a tabela Clientes.db, como sendo a **master query**. Ou seja, a tabela que irá comandar a apresentação do dados.

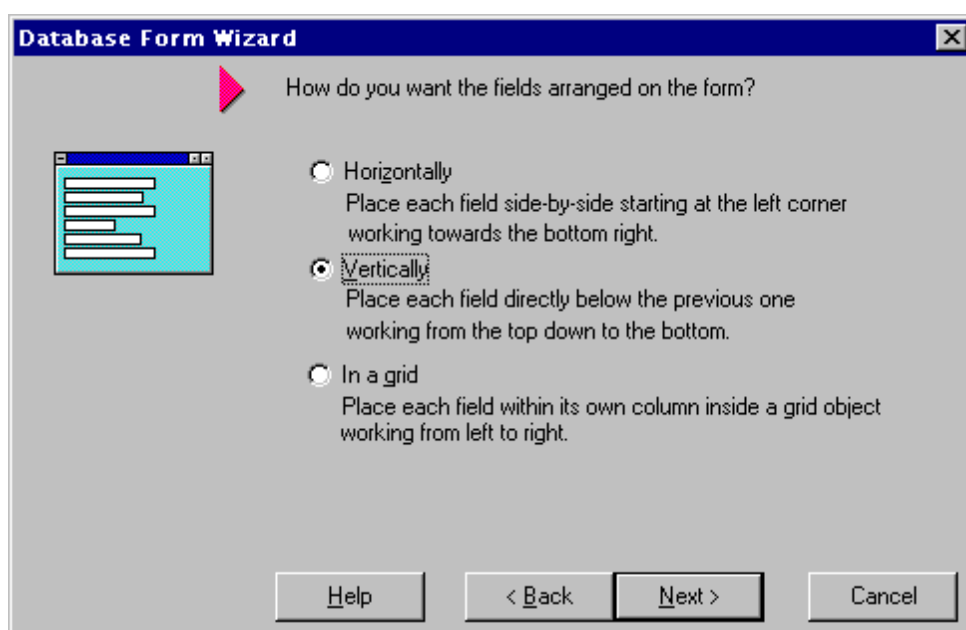


Selecione o botão >> para incluir todos os campos da tabela Clientes.db no formulário.





A opção **Vertical** posiciona os campos dentro do formulário, um acima do outro.



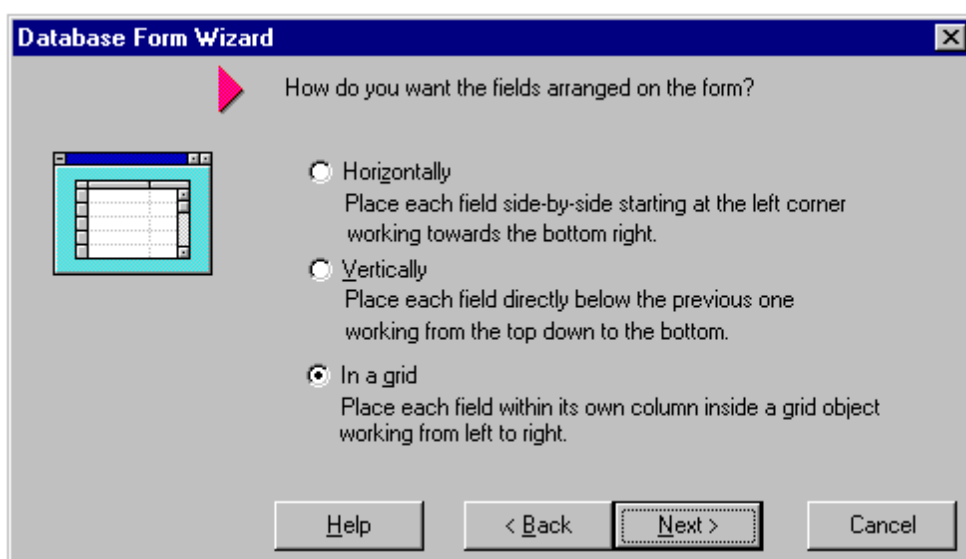
Alinhamento **Left**, posiciona as legendas ao lado esquerdo dos quadros de Edit.



Defina a tabela Vendas.db como a tabela de detalhes.



Inclua todos os campos da tabela Vendas, e a seguir, escolha o formato **Grid** para a apresentação da tabela Vendas.



Selecione os campos que farão a ligação entre as duas tabelas, escolhendo **INDCODIGO** como índice. Em **Detail Fields** e **Master Fields**, selecione o campo **CODIGO** como campo de junção em ambas as tabelas e pressione **Add**. Como mostrado a seguir.

**Database Form Wizard**

Select a pair of fields from the field lists that will join the two queries. Use the add button to add the selected pair to the list.

Available Indexes: **INDCODIGO**

**Detail Fields:** CODIGO

**Master Fields:** CODIGO, NOME, ENDERECO, CIDADE

**Joined Fields:**

Buttons: Help, < Back, Next >, Cancel

**Database Form Wizard**

Select a pair of fields from the field lists that will join the two queries. Use the add button to add the selected pair to the list.

Available Indexes: **INDCODIGO**

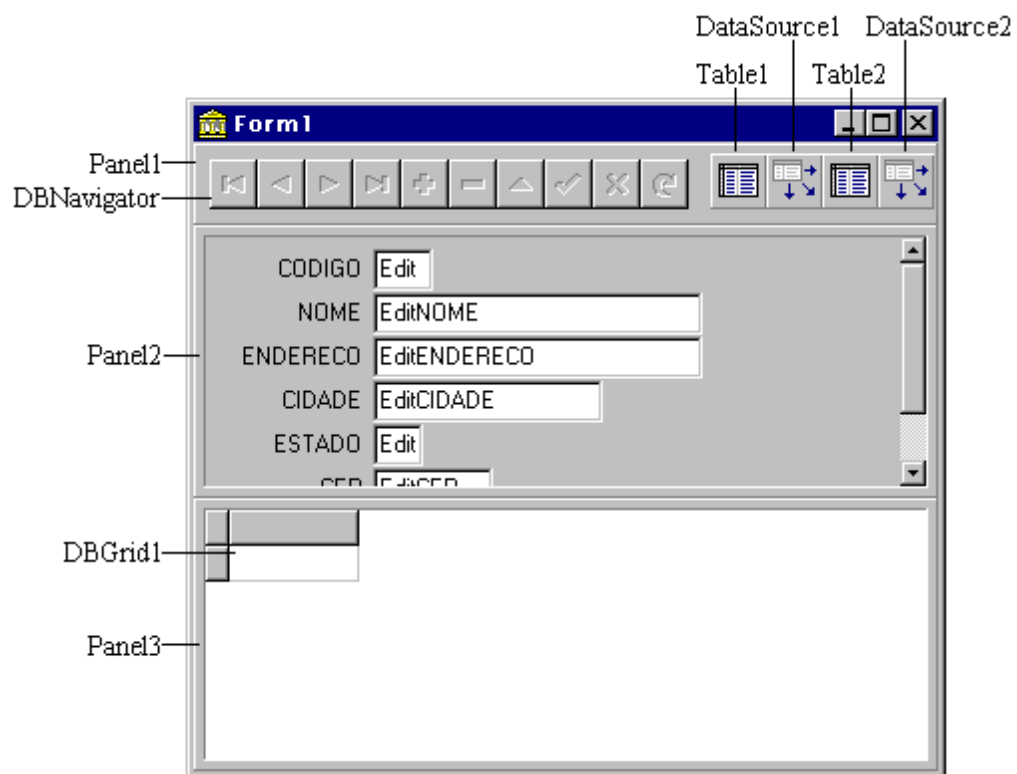
**Detail Fields:**

**Master Fields:** NOME, ENDERECO, CIDADE, ESTADO

**Joined Fields:** CODIGO -> CODIGO

Buttons: Help, < Back, Next >, Cancel

Pressione **Next** e finalize o Form Expert. O seu formulário estará como a figura a seguir.

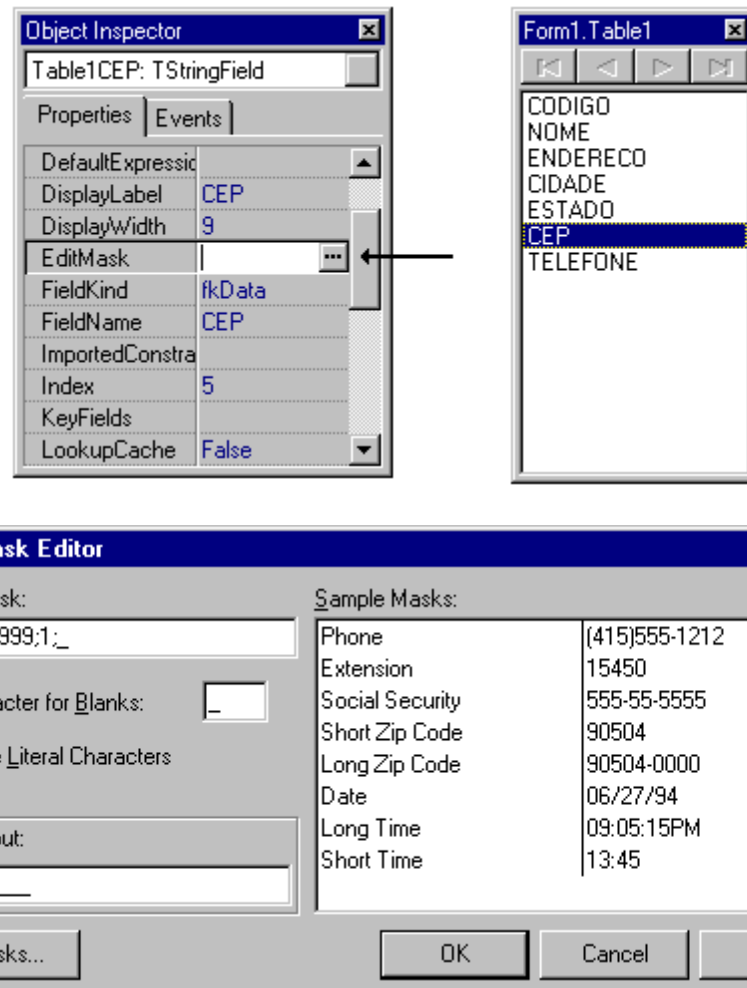


Vamos agora, alterar as propriedades de alguns componentes do formulário.

Comece alterando o tamanho do Panel3 referente aos dados dos clientes. E redesenhe o formulário como a figura abaixo.



Para formatar a entrada de dados, ou seja, definir o formato do dado, para a tabela Clientes.db, dê um clique duplo em **Table1** para acessar o editor de campos (**fields edit**). Selecione o campo CEP e no *Object Inspector*, procure a propriedade **Edit Mask**. Esta propriedade abre o quadro de diálogo *Input Mask Editor* onde poderemos escolher uma máscara para o campo CEP, escolha **Long Zip Code** e na caixa Input Mask elimine um dígito 9, como mostra as figuras a seguir.



Faça o mesmo para o campo Telefone, escolhendo a máscara Phone. Feche a *Input Mask Edit*.

Selecione o campo **CODIGO** em **Table2**, e mude a propriedade **Visible:=False**, para não aparecer o código do cliente no componente DBGrid, pois ele já será exibido por um componente Edit.

Selecione o **EditEstado** e altere a propriedade **Charcase:=ecUpperCase**, para que os caracteres neste TEdit sejam apresentados em maiúscula.

Insira um botão BitBtn no Formulário com as seguintes definições de propriedades:

- 1 - Caption:=Sai&r; Kind:=bkCLose

A propriedade **Kind** do **BitBtn** determina qual desenho será mostrado na face do botão. Esta propriedade também implementa algumas funções ao botão eliminando linhas de código, neste caso não será necessário nenhuma linha de código para fechar o formulário.

Finalmente, mude a propriedade **Active** dos dois componentes **TTable** (Table1 Table2) de **False** para **True**. Isto tornará o Table ativo tanto em tempo de projeto quanto em tempo de execução. Ao ativarmos o Table, os controles **TEdit** mudarão seu conteúdo, passando a exibir o primeiro registro da tabela a eles ligada.


Dê o nome de **frmClientes** ao Formulário e salve como **frmClientesUnt.pas**.



NUMERO	DATA	PRODUTO	QUANTIDADE	VALOR

O próximo formulário será construído manualmente, sem o uso do Form Wizard. Não utilizaremos o Form Wizard apenas por questões didáticas, mas este novo Formulário também poderia ser construído utilizando este assistente.

Este segundo formulário servirá para cadastrar vendas. O usuário selecionará um nome de cliente já cadastrado, a partir de um quadro combo, digitará a data da venda e escolherá um produto, através de botões de opção, definindo também a quantidade de itens de um mesmo produto e o valor total da venda.

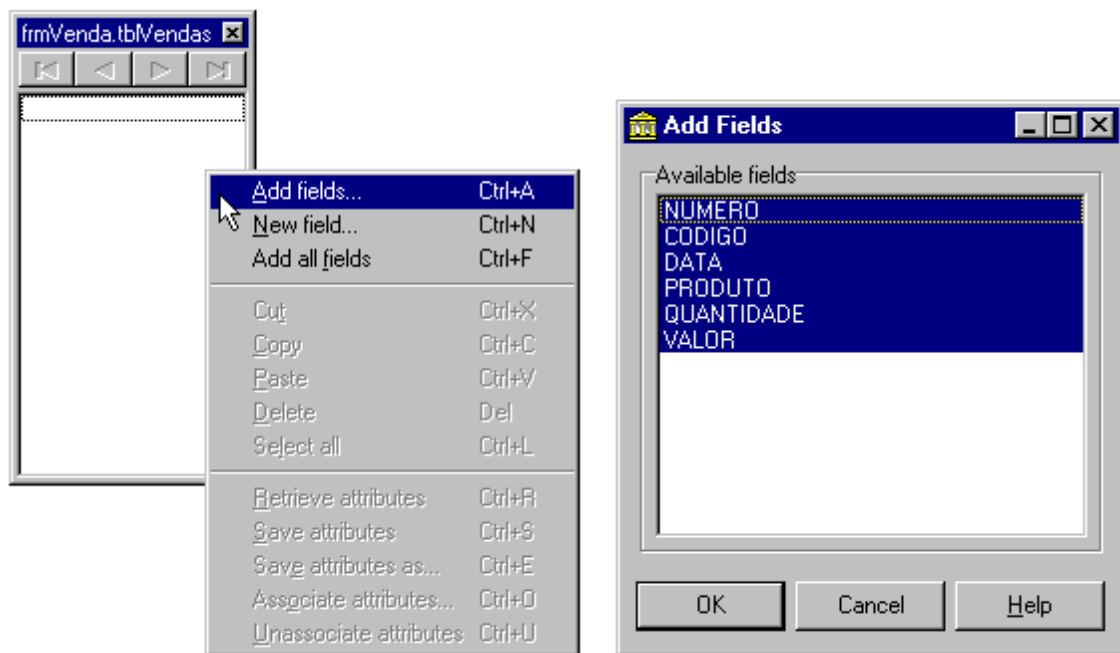
Insira um novo formulário ao projeto escolhendo a opção **New Form** do menu **File** ou o botão . Monte este novo formulário utilizando os componentes descritos na figura abaixo.

Altere as propriedades, de alguns componentes, indicadas na tabela a seguir:

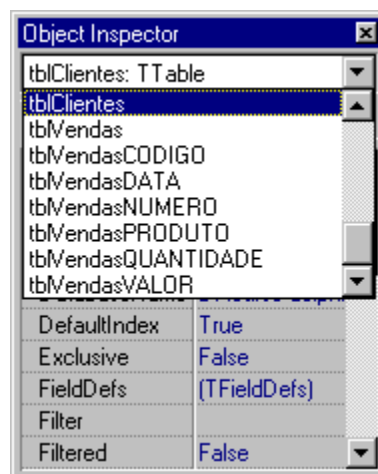
Componente	Propriedade	Valor
Table1	DatabaseName	C:\Curso Delphi 4
	TableName	Vendas.db
	Name	tblVendas
	Active	True
DataSource1	DataSet	tblVendas
	Name	dsrVendas
Table2	DatabaseName	C:\Curso Delphi 4
	TableName	Clientes.db
	Name	tblClientes
	Active	True
DataSource2	DataSet	tblClientes
	Name	dsrClientes
DbNavigator	DataSource	dsrVendas
	Name	navVendas
	ShowHint	True

Selecione todos os TEdit, dando um clique em cada um enquanto pressiona a tecla Shift, e altere a propriedade DataSource de todos ao mesmo tempo para dsrVendas. Depois selecione um por vez e altere a propriedade DataField correspondente a cada TEdit.

Dê um clique duplo em tblVendas para editar o campo Data. Repare que a janela Fields Editor aparece vazia sem nenhum campo listado. Clique com o botão direito do mouse no Fields Editor e escolha a opção **Add Fields...**, selecione todos os campos e pressione **Ok**, para inserir todos os campos da tabela Vendas.db no Fields Editor.



Voltando ao Fields Edit, selecione o campo **DATA** e altere o formato da propriedade **EditMask** para **Date**. Após a inserção de todos os campos da tabela Vendas no componente `tbVendas`, as propriedades destes campos podem ser acessadas diretamente através do **Object Inspector**. Como ilustra a figura a seguir.



O componente `DbLookupComboBox`, exibe os valores de um campo em uma lista, onde o usuário poderá escolher um valor para este campo. Os valores exibidos pertencem a uma tabela e quando o usuário fizer uma escolha, o valor escolhido será vinculado a uma outra tabela.

No formulário que estamos projetando - Vendas, o nome dos clientes virá da tabela Clientes e quando inserirmos a venda, o nome do cliente escolhido será vinculado a esta venda na tabela Vendas através da chave primária CODIGO.

Portanto o `DbLookupComboBox` trabalha com duas tabela simultaneamente, uma que fornece os dados para a lista e outra que está associada ao `DbLookupComboBox`. A tabela abaixo explica as propriedades relativas a esta vinculação.



Propriedade	Uso
DataSource	Indica qual tabela associada.
DataField	Indica o campo associado.
ListSource	Determina a tabela que fornecerá os dados para a lista.
KeyField	Indica qual o campo da tabela acima será usado como vínculo.
ListField	Indica qual campo será mostrado na lista do quadro combo.

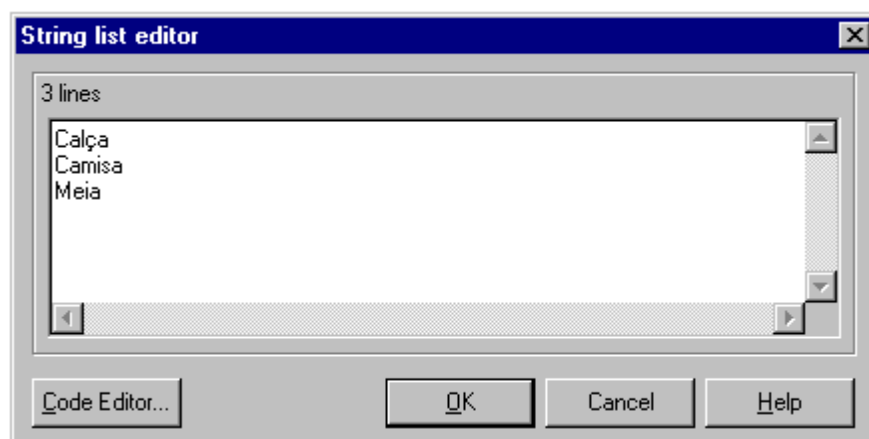
Siga a tabela abaixo para determinar as propriedades do nosso DbLookupComboBox.

Propriedade	Valor
Data Source	dsrVendas
Data Field	CODIGO
ListSource	dsrClientes
KeyField	CODIGO
ListField	NOME

A lista de clientes será vinculada através do campo CODIGO mas estará exibindo ao usuário, o campo NOME.

O componente **DBRadioGroup** apresenta uma série de opções pré-definidas a serem selecionadas pelo usuário, contendo um botão para cada opção. Estas opções são determinadas pela propriedade **Items**. O valor do campo na tabela associada ao DBRadioGroup, será o mesmo valor do botão selecionado.

Selecione o DBRadioGroup e edite a propriedade Items, configurando três opções para produtos vendidos, seguindo o modelo a seguir.



Altere outras propriedades conforme a tabela mostrada abaixo:

Propriedade	Valor
DataSource	dsrVendas
DataField	PRODUTO
Columns	3
Caption	vazia

Salve este Formulário com o nome de frmVendasUnt.Pas. O Formulário final é mostrada abaixo.

O próximo formulário será a abertura do programa. A partir dele serão chamados os dois formulários construídos anteriormente.

Acrescente um formulário vazio e construa-o da seguinte forma:

A propriedade **Hint** dos **SpeedButtons**, contém o texto que será mostrado quando o mouse ficar parado por alguns instantes sobre o botão. Defina ShowHint:=True, para ativar esta função.

Com isto concluímos a construção de nossa interface gráfica com o usuário.

Temos então em nosso projeto três formulários - frmControle, frmClientes e frmVendas. Todos foram salvos no disco com o mesmo nome da sua propriedade Name mais "Unt".

Começaremos o código pelo **frmControle**.

```
implementation
```

```
uses frmVendaUnt, frmClientesUnt;
```

```
{ $R *.DFM }
```

```
procedure TfrmControle.SpeedButton3Click(Sender: TObject);  
begin  
    Close;  
end;
```

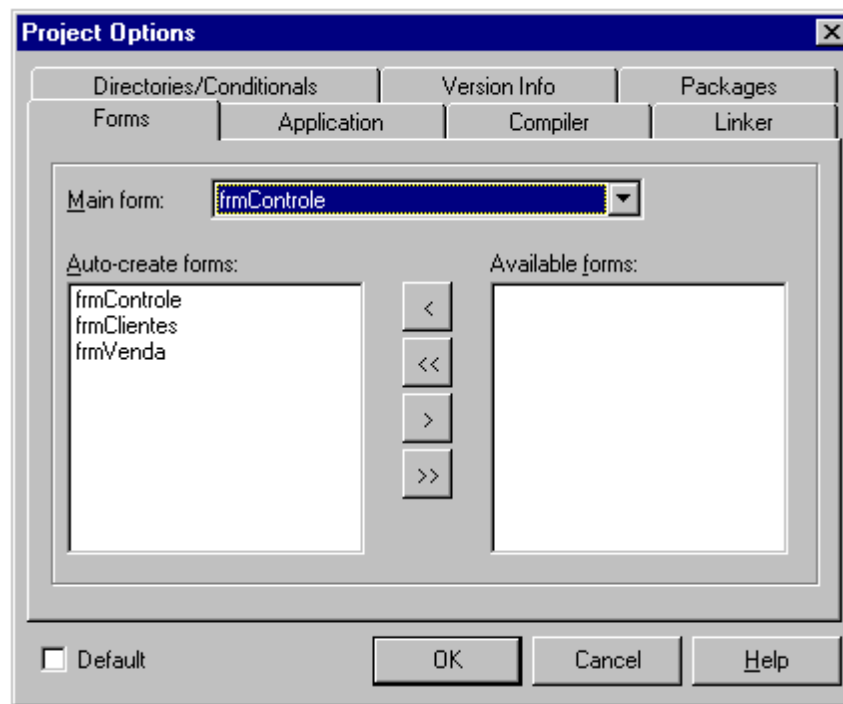
```
procedure TfrmControle.SpeedButton2Click(Sender: TObject);  
begin  
    frmVenda.Show;  
end;
```

```
procedure TfrmControle.SpeedButton1Click(Sender: TObject);  
begin  
    frmClientes.Show;  
end;
```

Código para **frmVendas**.

```
procedure TfrmVenda.BitBtn1Click(Sender: TObject);  
begin  
    frmVenda.Hide; {o método Hide esconde o Formulário, mas  
    não a elimine da memória, enquanto que Close retira  
    o Formulário da memória }  
end;
```



Antes de compilar o projeto, verifique se o formulário inicial do programa é mesmo o frmControle. No menu **Project** do Delphi, escolha a opção **Option...**, e defina como o Main Form o formulário frmControle.



Execute o programa pressionando F9.

## LISTA DE EXERCÍCIOS

---

1. Qual a diferença entre um programa feito para trabalhar sob o DOS e outro construído através do Delphi?
2. O que encontramos na Paleta de Componentes?
3. Qual a função da janela Object Inspector?
4. Quais os passos para desenvolvermos um programa em Delphi após a análise do projeto?
5. No primeiro programa que desenvolvemos, coloque um botão de comando que apague o conteúdo do label, e outro que finalize o programa.
6. Na calculadora, mesmo redefinindo a propriedade TabStop do Edit de resultado, o usuário poderá alterar o resultado da operação. Que outro controle poderíamos utilizar no lugar de um TEdit para exibir o resultado sem que o usuário possa alterar seu valor? Altere o projeto.
7. O ícone escolhido para representar o projeto, não estará presente no formulário. Qual propriedade deveremos alterar para que o ícone representante do projeto também esteja no formulário? Responda e execute.
8. Altere os botões do Jogo da Velha para o tipo BitBtn, e reconstrua o projeto para alterar as propriedades Kind e Caption, exibindo nos botões os ícones -  e .
9. No Jogo da Velha, sempre o computador inicia uma nova partida. Altere o código para que o programa pergunte ao usuário quem iniciará uma nova partida.
10. No projeto Bloco de Notas, quando escolhemos a opção **Sair**, o programa não pergunta se queremos salvar as alterações realizadas no texto. Inclua uma variável booleana na seção Implementation, alterando o seu valor no procedimento Memo1Change e verificando-a no momento de encerrar o programa.
11. Inclua mais um botão no Relógio Despertador com a opção de Soneca, soando o Beep dez minutos após o primeiro despertar.
12. Construa um projeto que mostre todos os tipos de preenchimentos (hachuras) dentro de círculos em um único formulário.
13. Construa um exemplo semelhante ao da luz em movimento, mas que a figura ao invés de ser desenhada em outro aplicativo, seja desenhada por instruções (ellipse e rectangle) contidas dentro do programa em Delphi.
14. Substitua o controle ComboBox por um ListBox no exemplo de Catálogo, com pesquisa pelo número do telefone.
15. No último exemplo, crie mais uma tabela de Estoque, e a cada venda realizada os valores de estoque deverão ser atualizados.
16. Crie mais duas tabelas, uma de Fornecedores e outra de Compras, semelhantes a Clientes e Vendas, e acrescente os campos ao seu critério.