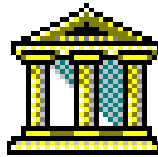


SUMÁRIO

PRIMEIROS PASSOS	1
ANATOMIA DE UMA JANELA	2
BARRA DE MENU, BARRA DE FERRAMENTAS E PALETA DE COMPONENTES	3
<i>Barra de Ferramentas</i>	3
PALETA DE COMPONENTES	4
JANELAS DO DELPHI	8
MEU PRIMEIRO PROGRAMA	11
ADAPTAR AS PROPRIEDADES DOS OBJETOS	12
ESCREVER O CÓDIGO PARA OS EVENTOS ASSOCIADOS.	14
EXEMPLO I - CALCULADORA	21
PROPRIEDADES	26
<i>BorderStyle</i>	26
<i>Default</i>	28
<i>Tabstop</i>	28
<i>Name</i>	29
<i>Caption</i>	29
<i>Text</i>	29
MÉTODO	29
<i>Setfocus</i>	29
VARIÁVEIS NO DELPHI	30
<i>Formas de Declarar uma Variável</i>	31
FORMATAÇÃO DE NÚMEROS	31
<i>Formatando Data e Hora:</i>	32
MODIFICANDO A CALCULADORA	32
EXEMPLO II - JOGO DA VELHA	37
EXEMPLO III - BLOCO DE NOTAS	48
EXEMPLO IV - RELÓGIO DESPERTADOR	60
MÉTODOS GRÁFICOS	64
DESENHO DE PONTO	64
CORES	65
DESENHO DE LINHA	66
EXEMPLO V - CATÁLOGO	73
INSERINDO UMA NOVA UNIDADE	75
TIPOS	76
REGISTROS	76
PONTEIRO DE REGISTROS	79
LISTA DE EXERCÍCIOS	85



Delphi 3.0

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 e 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. Formulário este que será a janela do aplicativo apresentada ao usuário.

O Delphi permite o uso de objetos, e sua criação. Ele trabalha com eventos que dão início à alguma rotina de trabalho, ou seja, o programa fica parado até que um evento ocorra.

Um programa tradicional, feito para ser executado em DOS, é organizado em torno de estruturas de dados com 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.

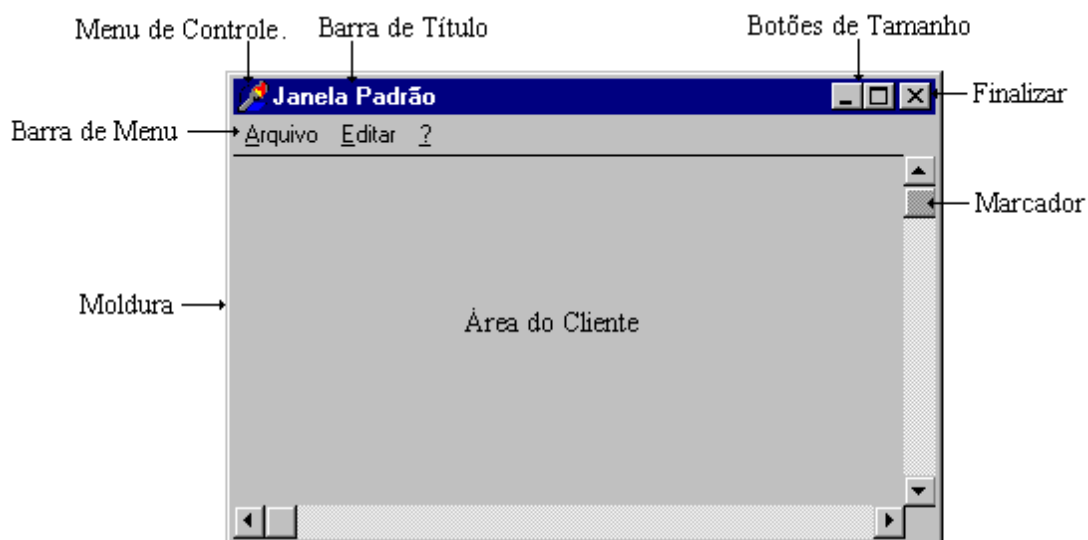
Um carro é um objeto que possui propriedades e métodos. A tabela abaixo lista algumas propriedades e comportamentos do objeto real **carro**.

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.

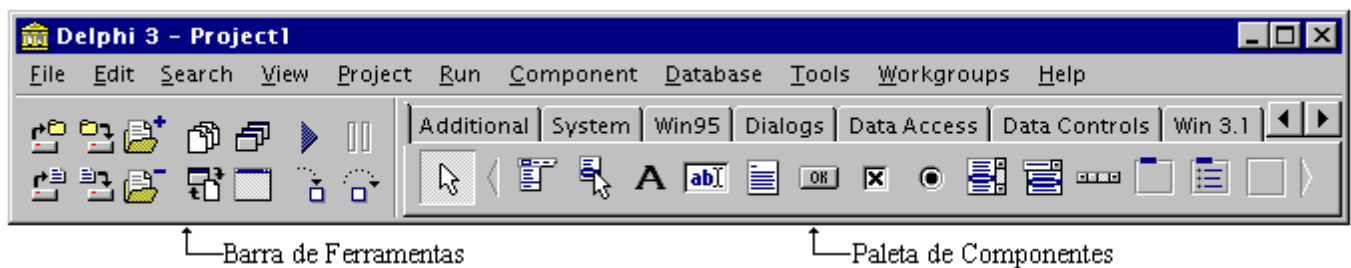
Barra de Menu - Está abaixo da barra de título e contém as opções de controle do aplicativo.

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

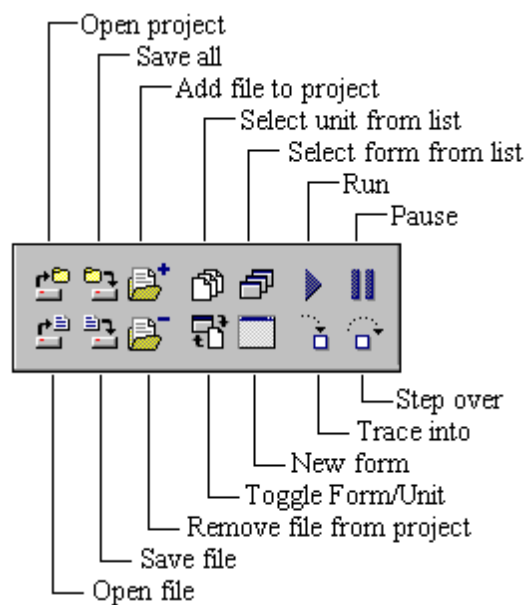
Área Externa à Janela - É sua Mesa de Trabalho onde você pode ter tantos aplicativos quantos necessitar e a memória do seu computador permitir.

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



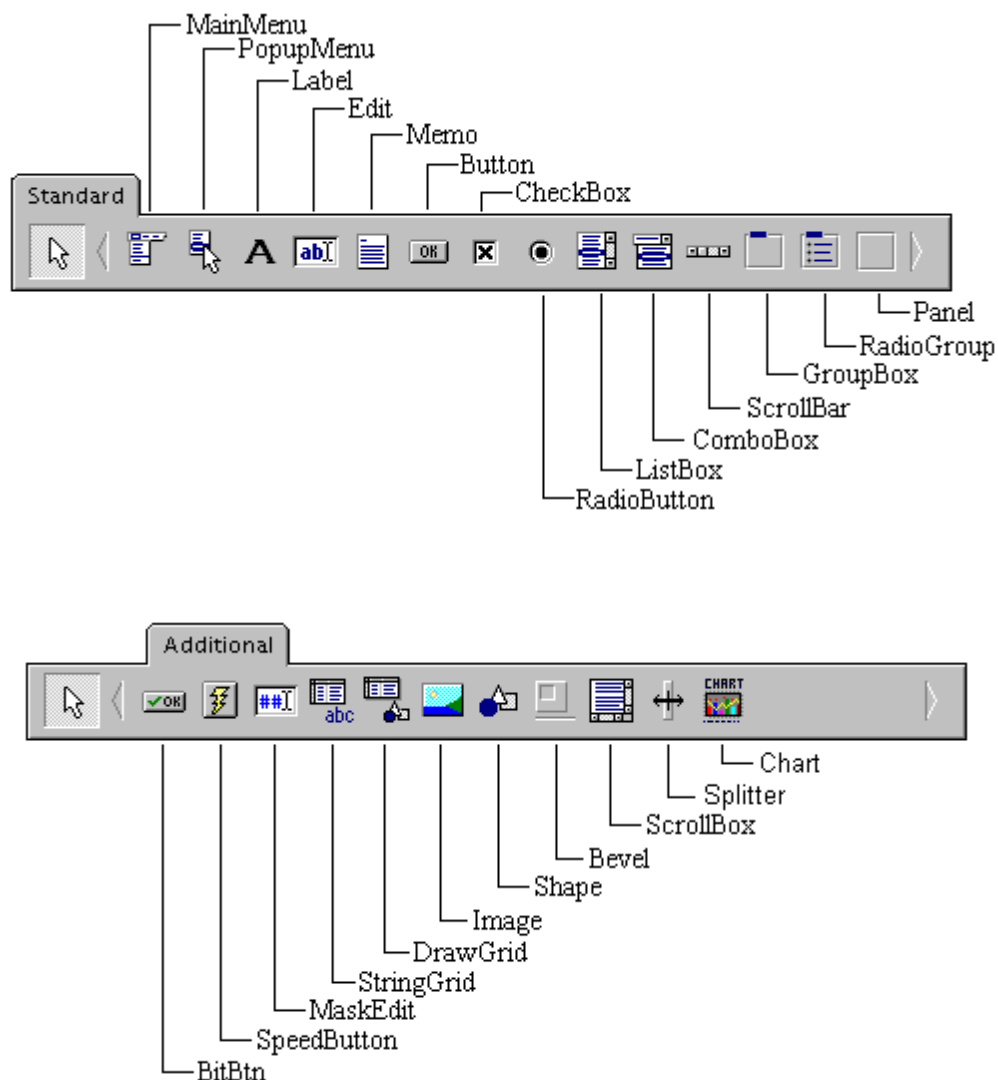
PALETA DE COMPONENTES

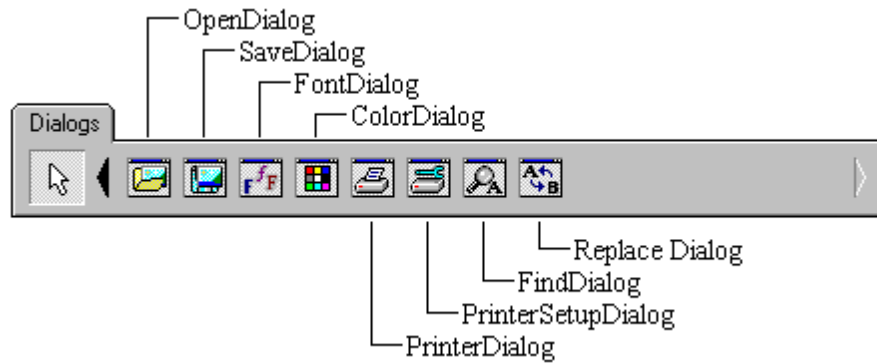
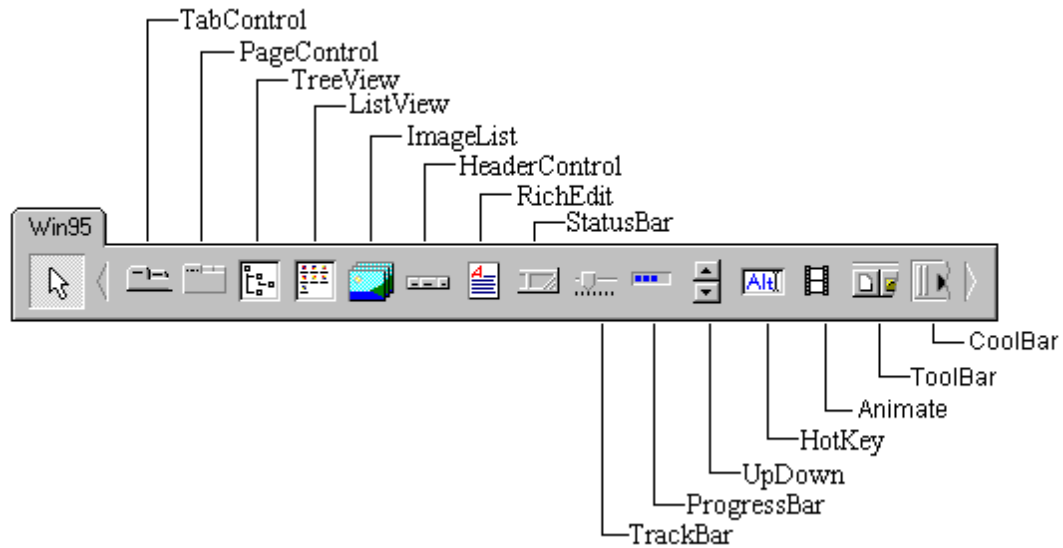
A Paleta de Componentes, possui todos os controles que iremos precisar para desenharmos nossa janela - formulário - como um programa de desenho livre. Para incluir um controle no formulário, existem dois métodos:

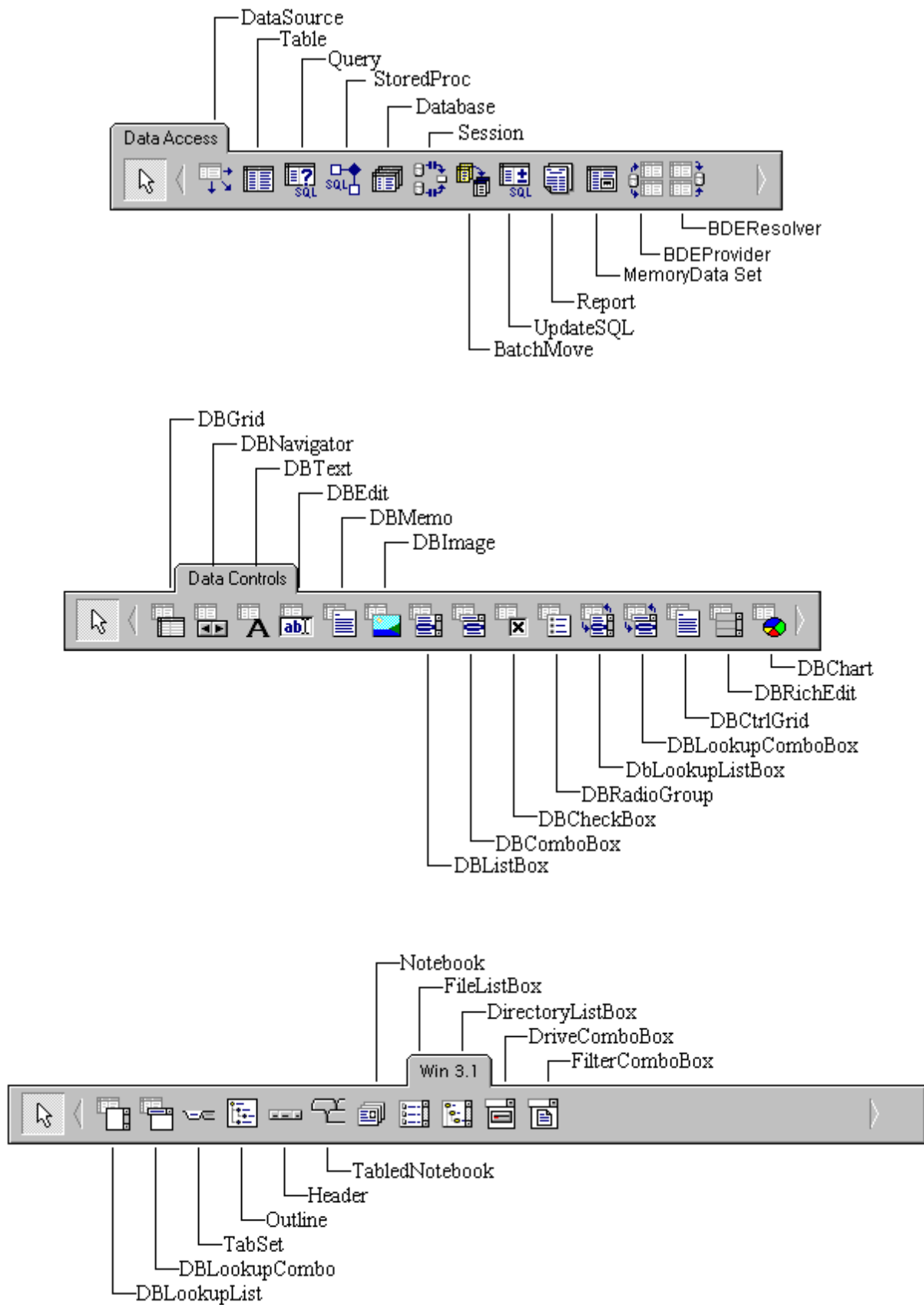
1 - Click Duplo no ícone da paleta de componentes. Fará com que o controle seja inserido no centro do formulário com um tamanho padrão.

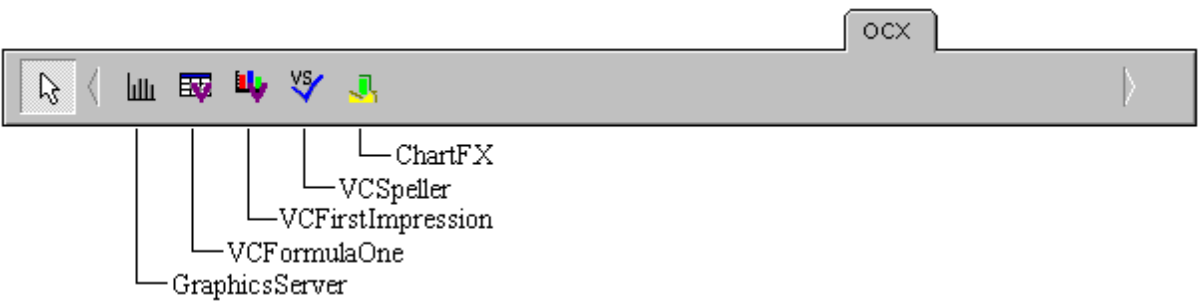
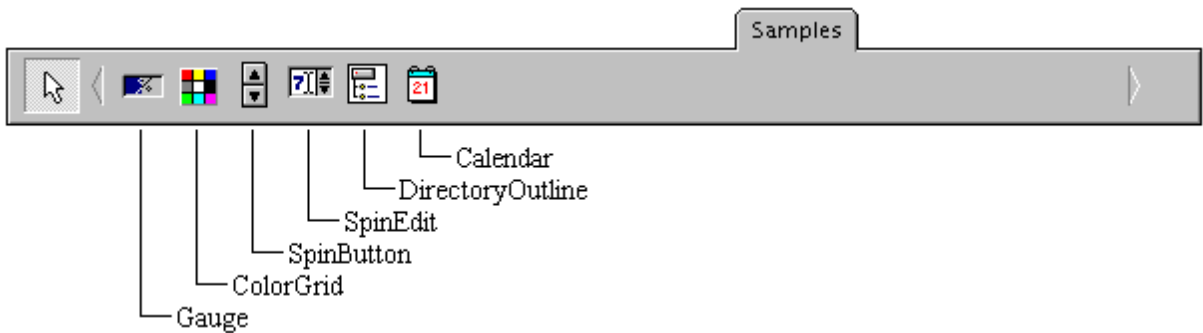
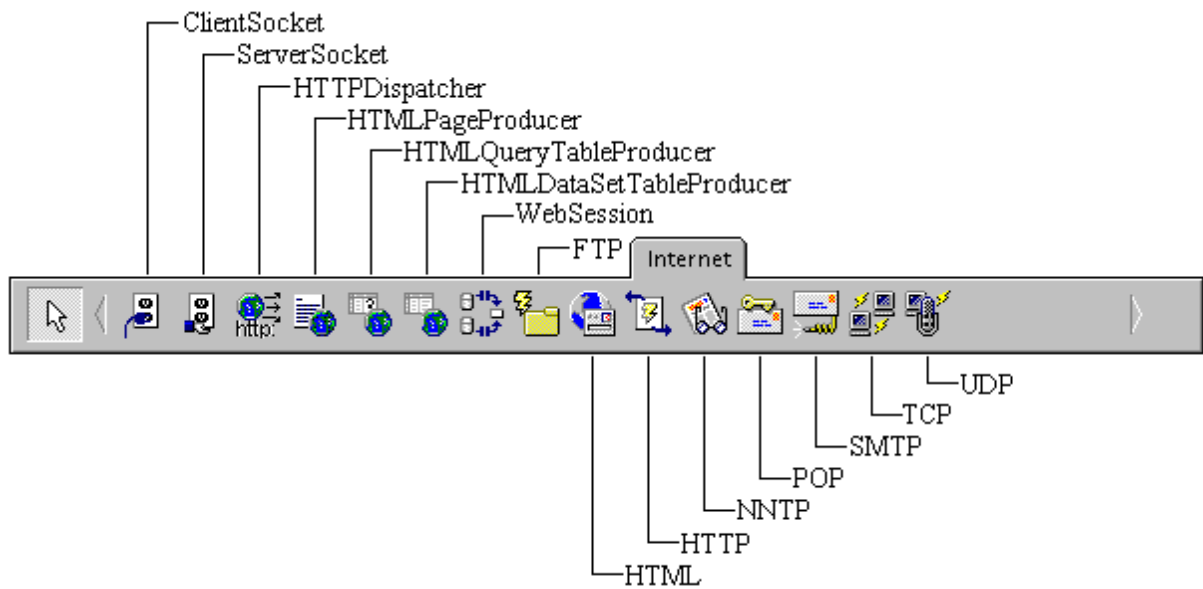
2 - Selecionar o ícone na caixa de ferramentas e depois dar um clique no formulário, na posição desejada para o objeto (canto superior esquerdo deste).

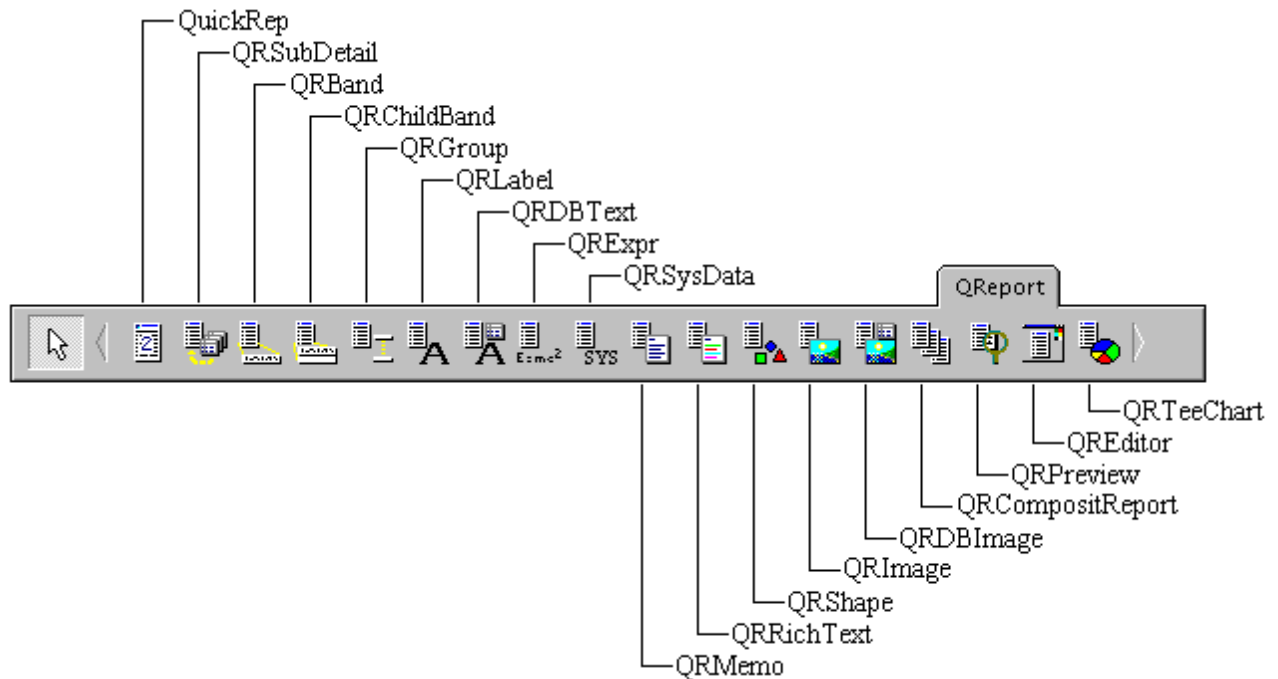
Podemos dimensionar estes controles, depois de inseridos, a qualquer momento durante o desenvolvimento. Primeiro, selecionamos o controle dando um clique em cima dele e depois o dimensionamos arrastando um dos oito botões dimensionadores que circundam este objeto.







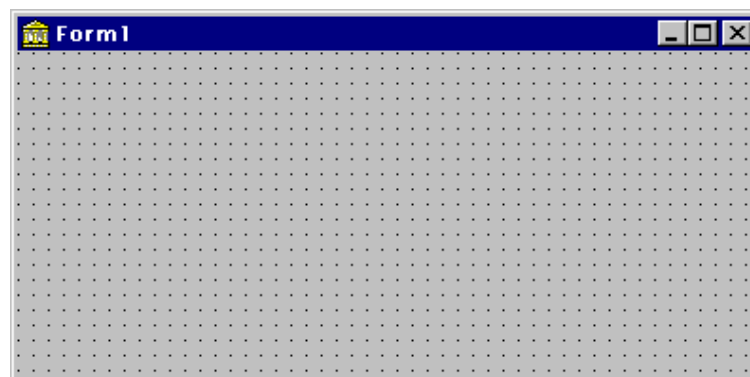




JANELAS DO DELPHI

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.



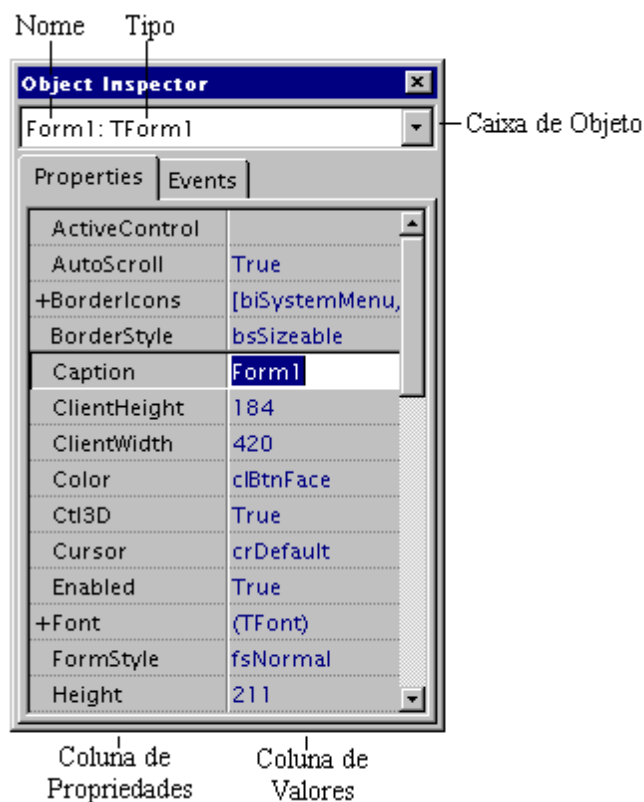
Controles

Existem vários tipos de objetos no Delphi, entre eles, formas e controles. Controles são todos os objetos que compõem um formulário, inserindo-o em um formulário ou controlando os seus eventos e propriedades. Um controle é qualquer objeto que o usuário possa manipular desde que não seja uma janela (formulário).

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 um desses objetos possui um conjunto específico de propriedades que podem ser associadas a eles. Ao trabalharmos nos diferentes objetos, a janela **Object Inspector** nos permitirá mudar as propriedades do objeto com que estamos trabalhando. Esta janela possui a divisória *Events* onde estão listados todos os eventos possíveis de ocorrerem com o objeto especificado.

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



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

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

Coluna de Valores - Exibe o valor da propriedade correspondente.

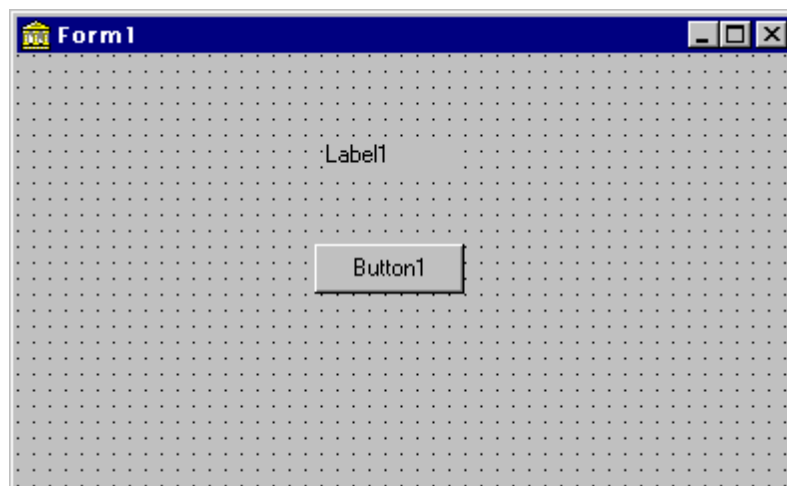
MEU PRIMEIRO PROGRAMA

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

Existem três passos principais para a 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:

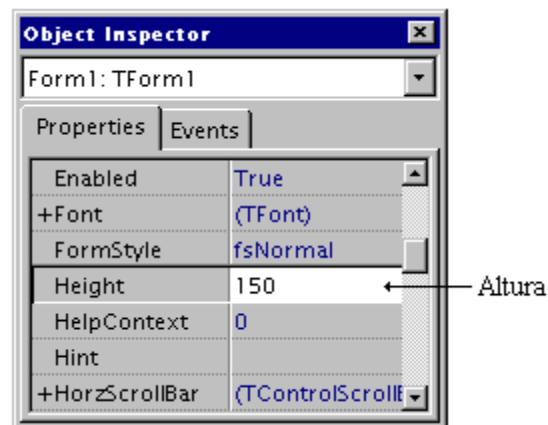
- a) Dando um duplo clique na barra de ferramentas no controle desejado;
- b) Selecionando o controle e dar um clique no formulário.

2 - Observe que, quando o controle estiver selecionado, poderemos arrastá-lo e dimensioná-lo dentro do formulário.

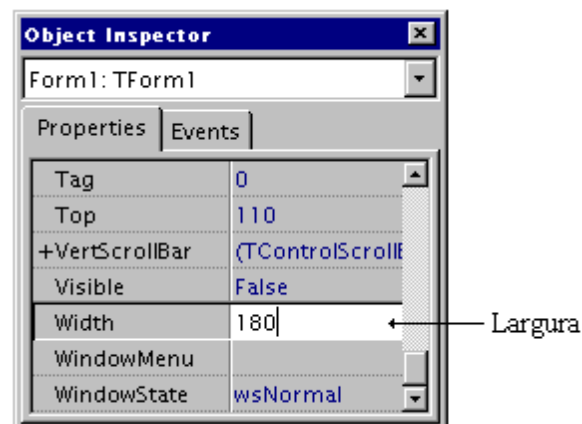
ADAPTAR AS PROPRIEDADES DOS OBJETOS

Para se alterar a propriedade 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 valor.

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

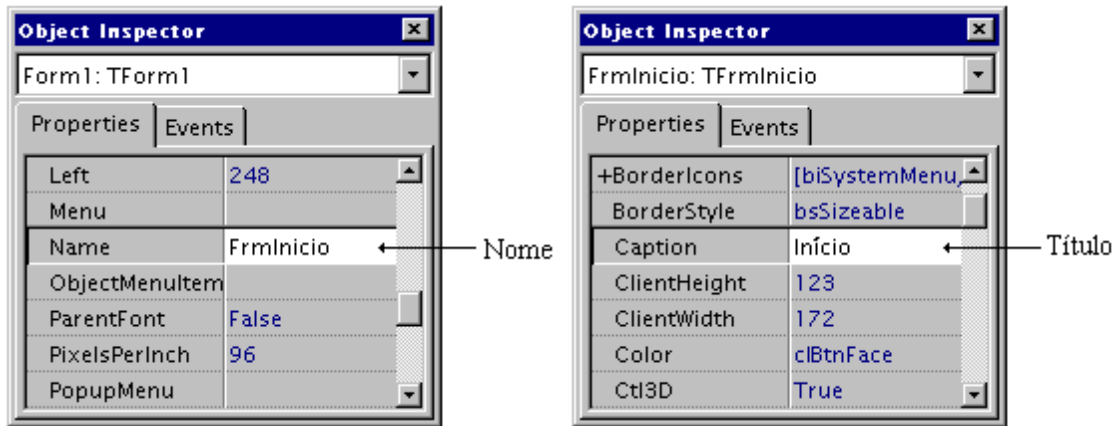


Selecionar a propriedade **Width** e dar o valor de 180.

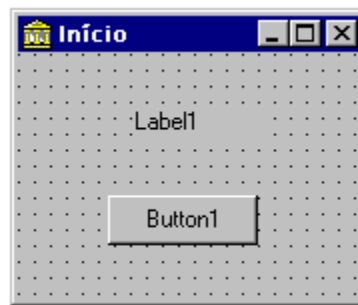


Estes números correspondem a Pixels, que é a quantidade de pontos do monitor .

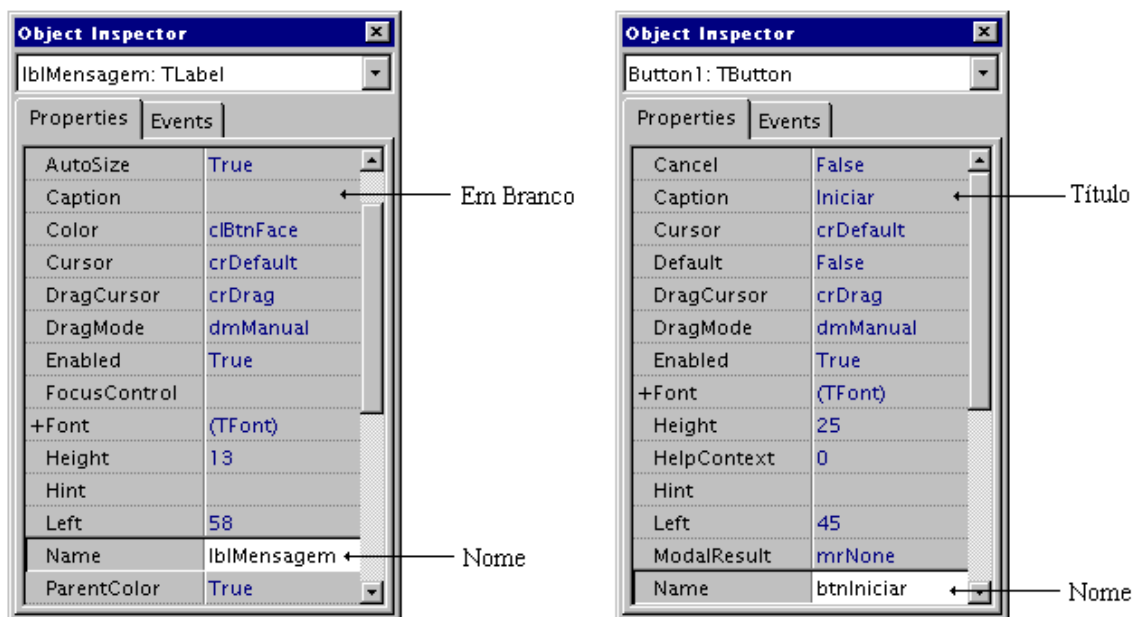
O mesmo deverá ser feito para as propriedades **Name** e **Caption**. A propriedade Name será a identificação do Objeto quando construirmos o código da aplicação. E a propriedade Caption é a palavra que aparecerá como título da janela.



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



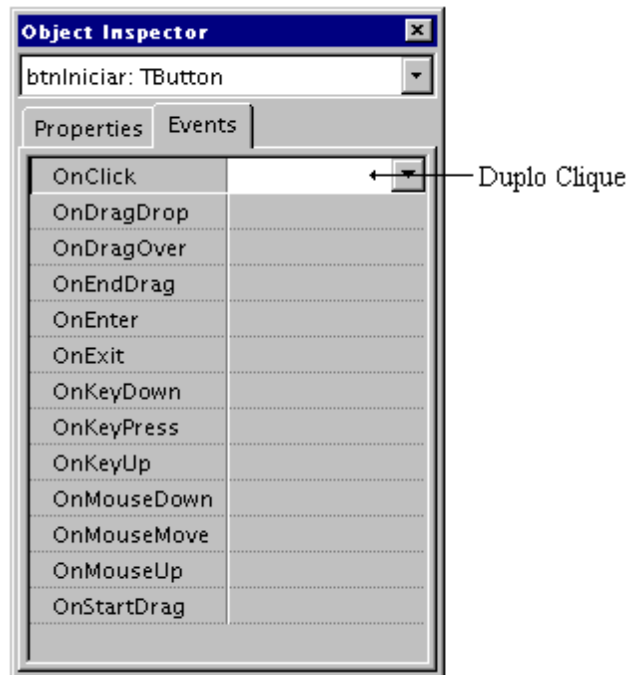
Agora, altere as propriedades do **TLabel** e do **TButton**.



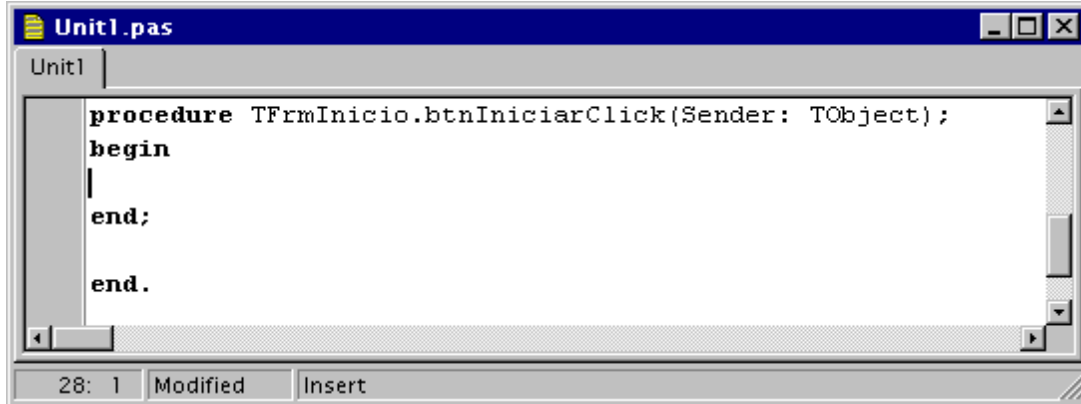


ESCREVER O CÓDIGO PARA OS EVENTOS ASSOCIADOS.

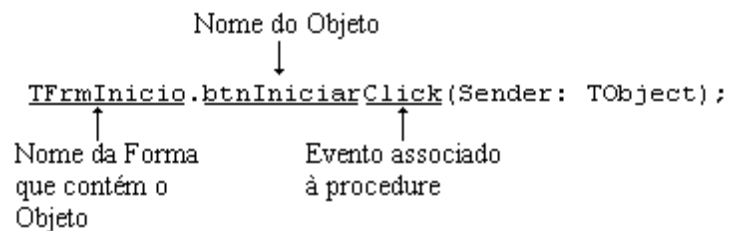
O código é escrito na janela **Unit**, para acessá-la, selecione o botão **Iniciar** e na janela *Object Inspector*, chame a divisória *Events* e dê um duplo clique na parte direita da linha que contém o evento `OnClick` - a rotina escrita para este evento, será executada quando o botão **Iniciar** for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



Janela Unit



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.



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

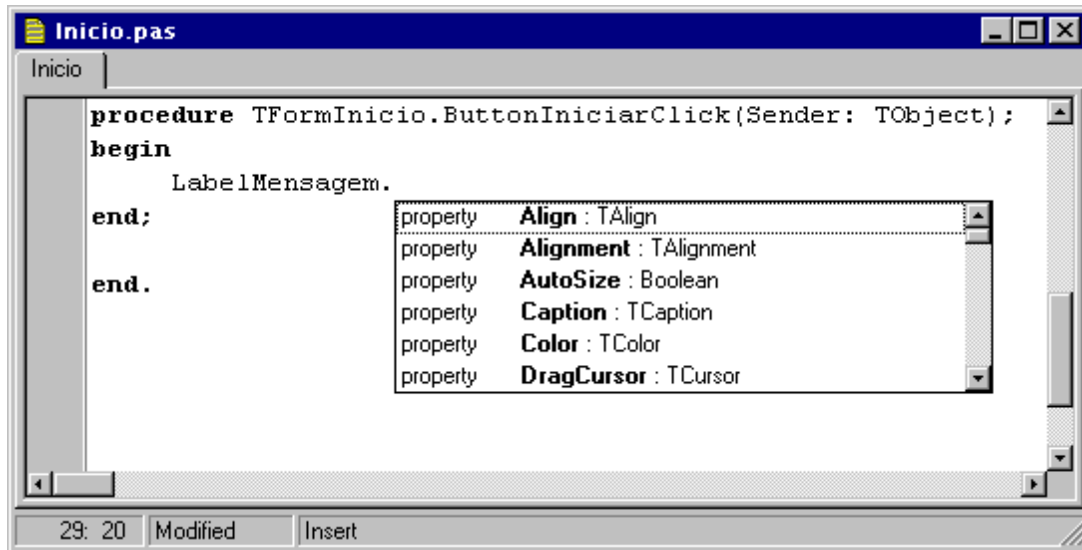
A Janela Unit também pode ser acessada dando-se um duplo clique no objeto que se quer criar um código. Cada objeto tem um evento que é mais comumente utilizado, e é com este evento que o Delphi iniciará a Janela Unit quando acessada desta forma, isto não impede que criemos outros códigos utilizando mais de um evento ao mesmo tempo.

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.

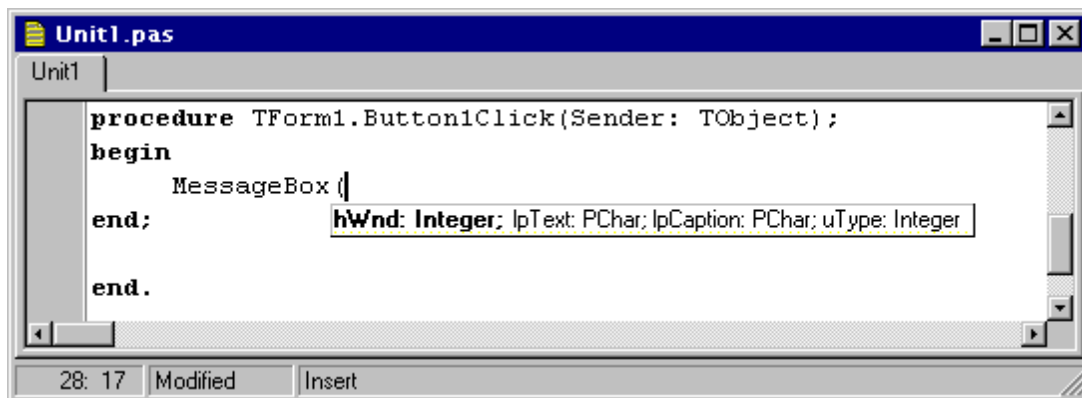
Atribuimos valores a uma propriedade de objeto seguindo o padrão:

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

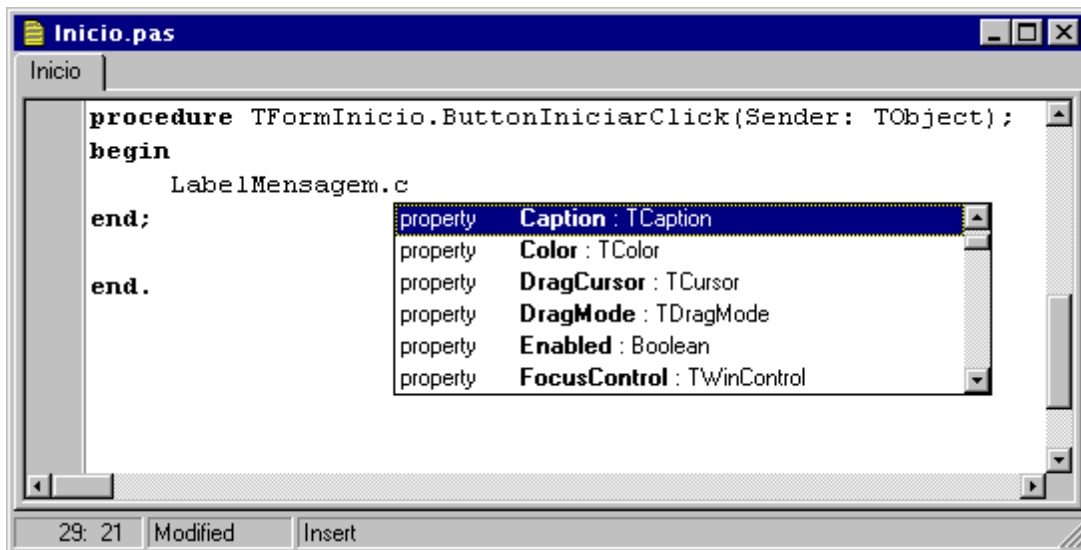
Abra a Janela **Unit** para o botão de comando e digite o código conforme a figura a seguir. Repare que ao digitar o ponto após LabelMensagem, e aguardando alguns instantes, o Delphi exibirá uma lista de propriedades e métodos do controle Label.



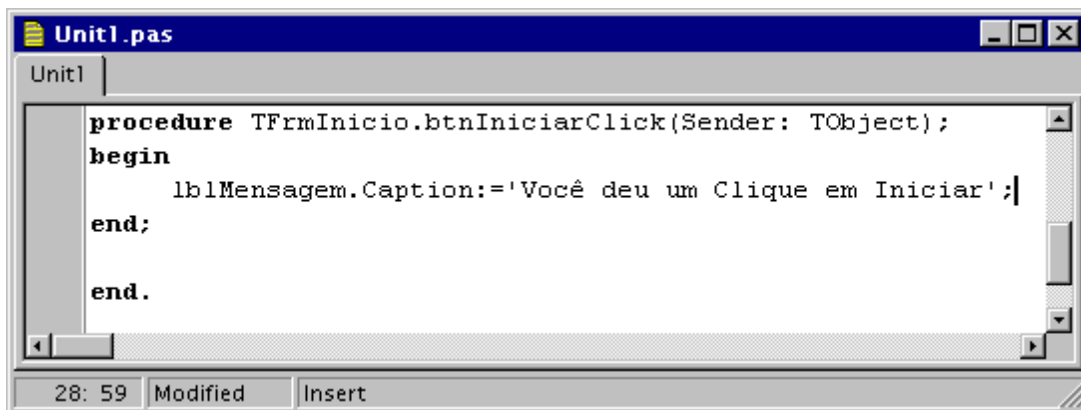
Esta ajuda do Delphi pode ser acionada para qualquer controle ou função, quando digitamos o nome de uma função, ele exibe os parâmetros necessários para a execução desta função.




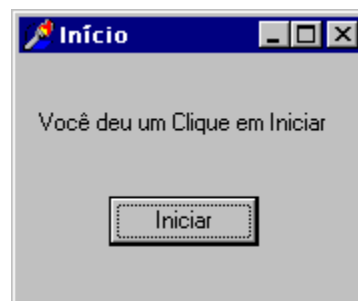
Para escolher uma propriedade do LabelMensagem, selecione-a com as setas de direção e então pressione Enter, inserindo-a na linha de comando. Ou então, digite a primeira letra da propriedade, selecionando-a.



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”.



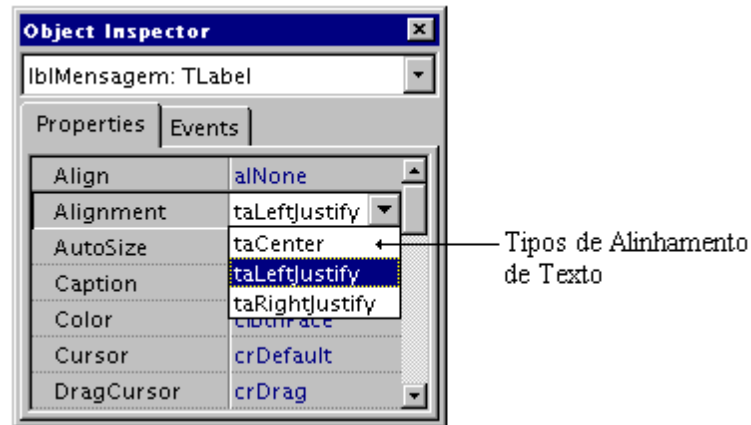
Clique sobre o botão **Run** da barra de ferramentas () para que o Delphi inicie a compilação do projeto. Em seguida, selecione o botão Iniciar para ver o resultado.



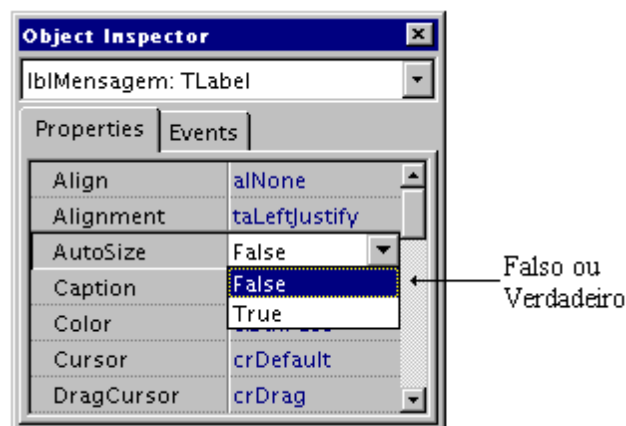
Finalize a execução do projeto teclando **Alt+F4** ou no botão Finalizar (☒) da barra de título da janela.

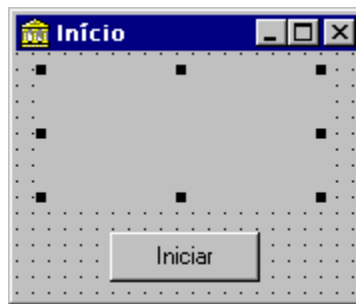
Para alternar a visualização entre o Formulário e a janela de código Unit, utilize o botão **Toggle Form/Unit** (☐) na barra de ferramentas.

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.

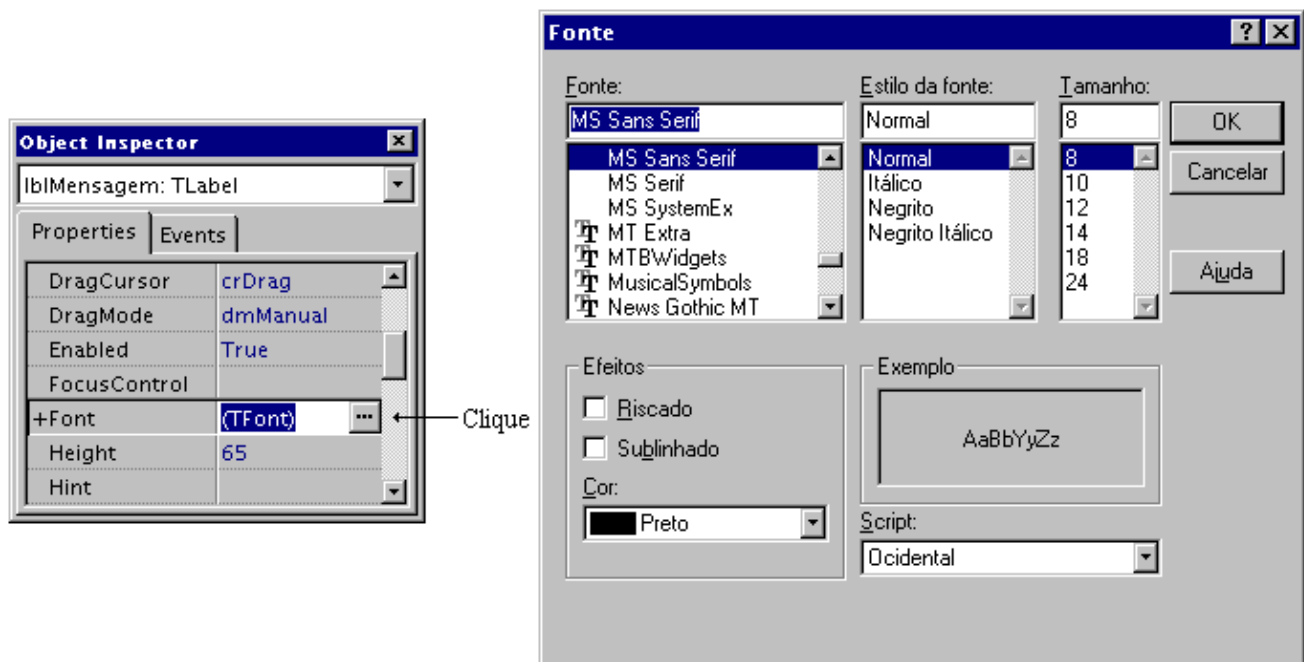


Selecione o objeto **TLabel** através da Caixa de Objeto da janela Object Inspector, e altere a propriedade **Alignment** para **taCenter**, para que o texto no TLabel fique centralizado. Altere também a propriedade **AutoSize** para **False**, e no Formulário aumente a largura do TLabel.



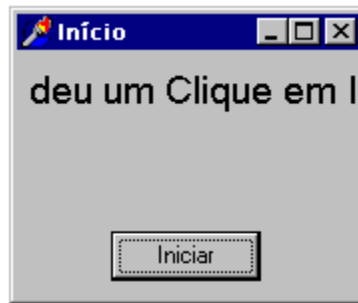


Além das propriedades descritas acima, com padrões pré-definidos, existem outras que possuem inúmeras escolhas, neste caso, ao invés de uma seta, observaremos três pontos, este é o caso da propriedade Font.

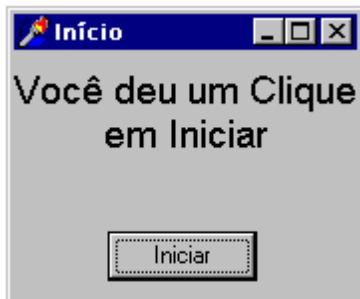


Quando selecionamos os três pontos, aparece um Quadro de Diálogo onde escolheremos o formato da fonte que será apresentada a mensagem.

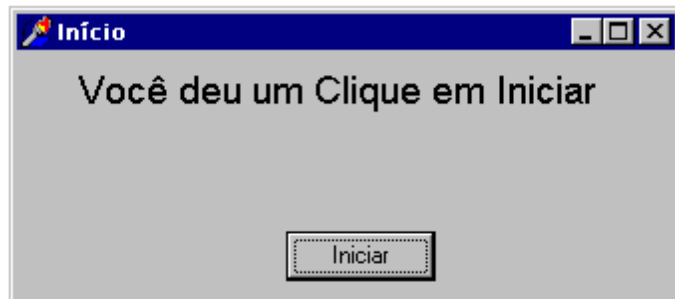
No seu projeto **Iniciar**, teste as alterações de fonte e observe as mudanças. Na figura a seguir, foi utilizada a fonte Arial com tamanho de 14 pontos.



Observe na figura acima que o texto não coube na área de exibição do TLabel e nem do Formulário, nós temos duas opções para que este texto apareça integralmente. A primeira, é alterar para True, a propriedade **WordWrap** do **TLabel**, esta propriedade insere uma mudança de linha quando o texto atinge a margem direita do objeto. A segunda, é redimensionar os tamanhos da TLabel e do Formulário. Como mostram as figuras a seguir.



Primeira opção



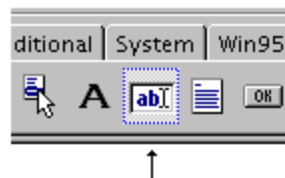
Segunda opção

EXEMPLO I - CALCULADORA

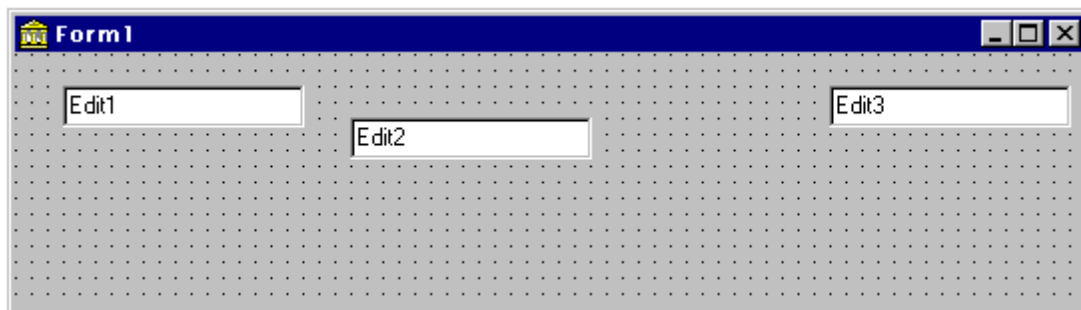
Para iniciar um novo projeto, escolha a opção **New Application** do menu **File**. Não salvando o projeto Iniciar anterior.

Dimensione e insira os controles, 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.

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 mostra.

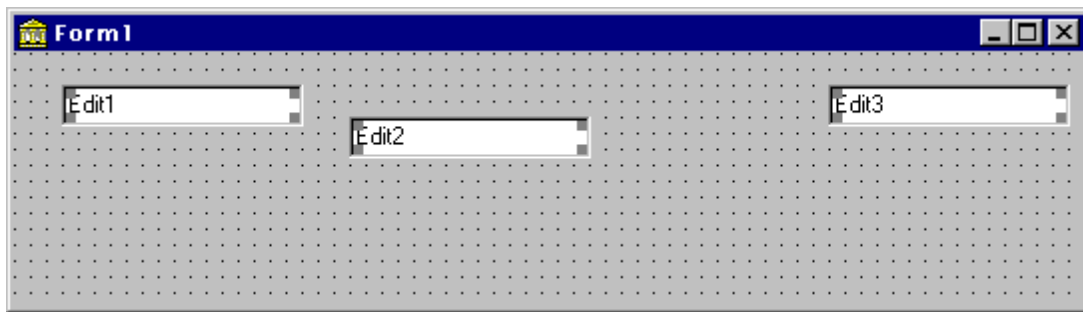


Logo após, basta ir inserindo o objeto (no caso, o Edit) dentro do Formulário, sem se preocupar com a estética. Após inserir todos os componentes repetidos, clique no cursor bitmap em forma de seta, isto desativará o botão selecionado anteriormente.

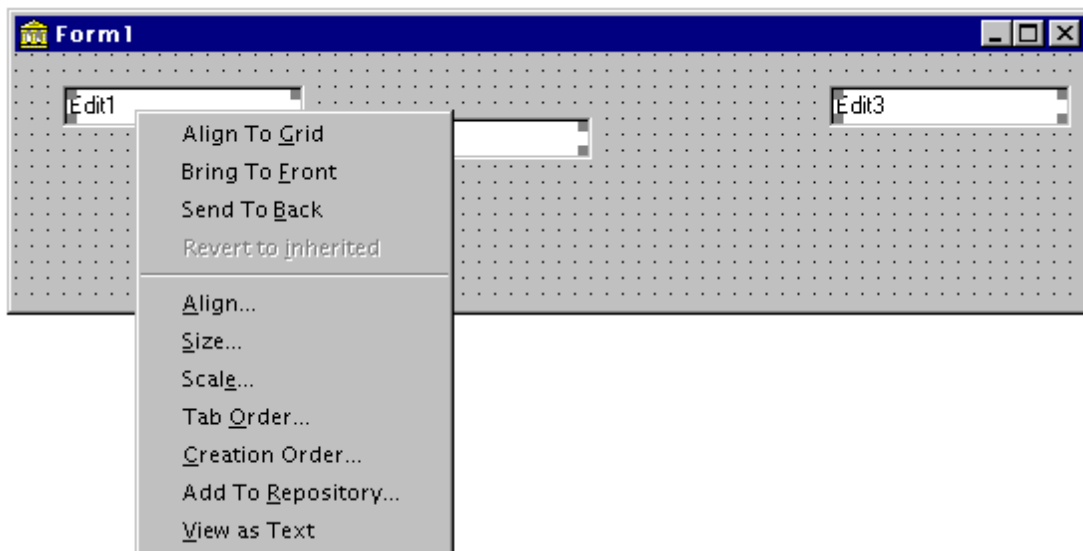


Observe que os objetos TEdit estão desalinhados, o Delphi nos oferece um recurso para realizar rapidamente um alinhamento entre objetos.

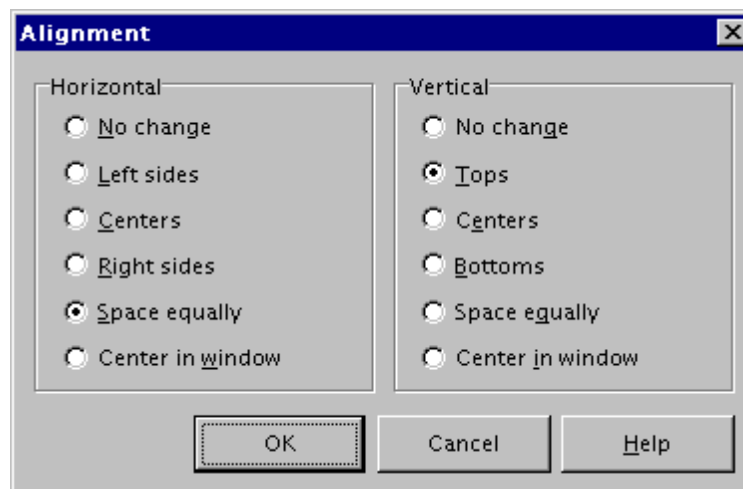
Primeiro deveremos selecionar os objetos que queremos alinhar. Pressione a tecla **Shift** enquanto você dá um clique em cada um dos TEdit, selecionando todos ao mesmo tempo como mostra a figura a seguir.



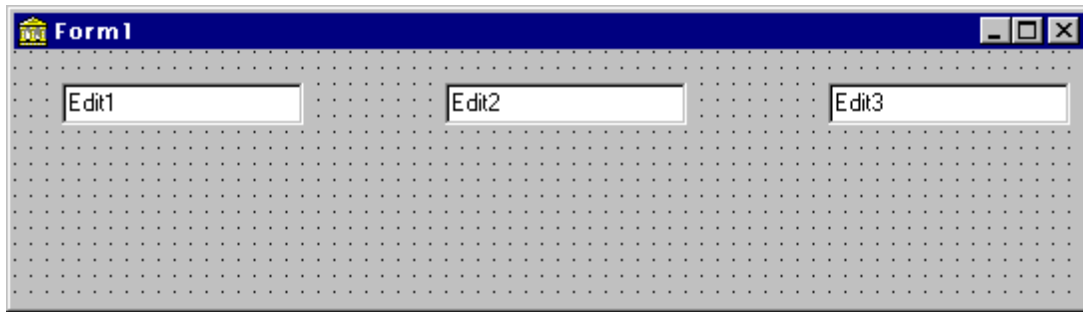
Depois disto, pressione o botão direito do mouse em cima de um TEdit, para aparecer o pop-menu. Escolha **Align...**, aparecendo a janela **Alignment**.



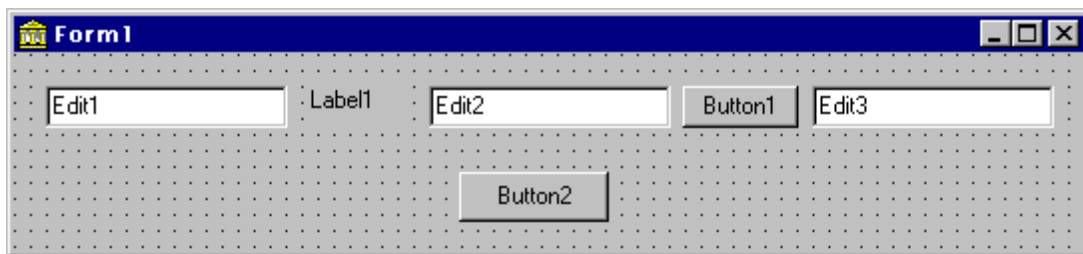
Escolha **Space equally** (igualmente espaçado) para alinhamento horizontal, e **Tops** (topo) para o alinhamento vertical.



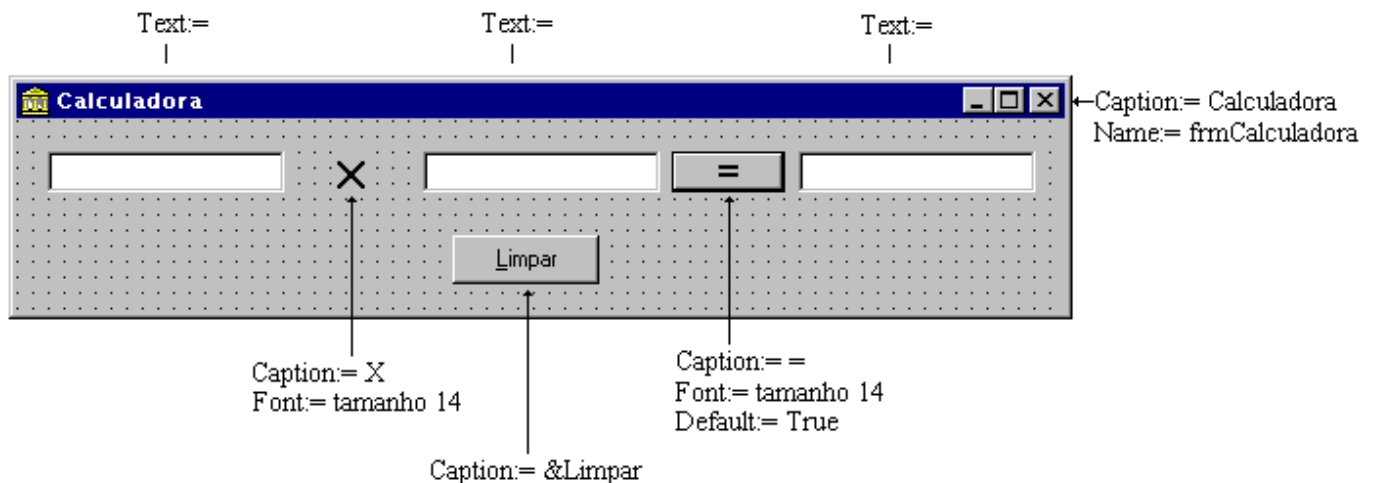
Após o alinhamento, nosso Formulário estará conforme a figura abaixo:



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



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

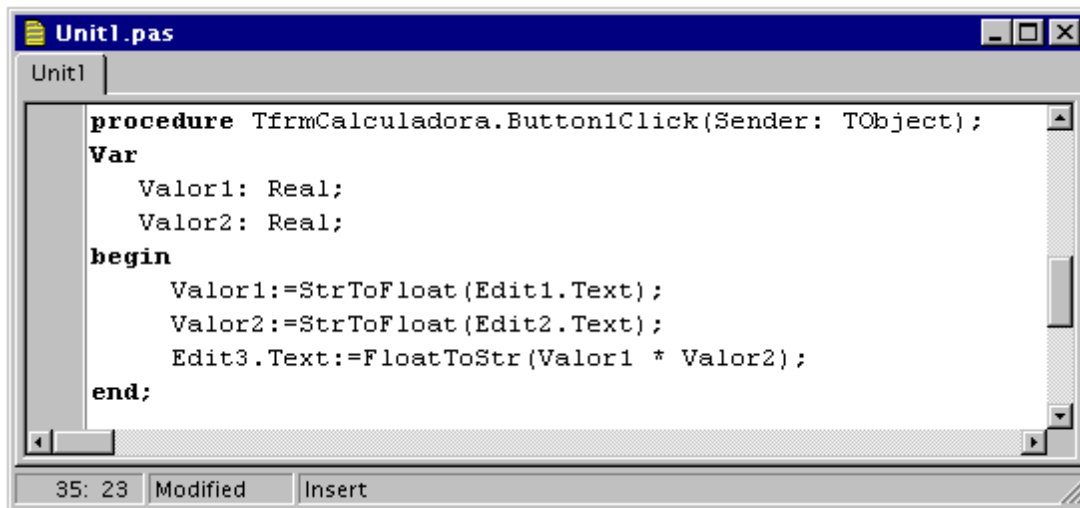


Neste exemplo de projeto, digitaremos um número em Edit1, outro em Edit2, e quando for dado um Click em Button1, o resultado da multiplicação aparecerá em Edit3. Para limpar os Quadros de texto, usaremos o Button2.

O projeto irá trabalhar basicamente com dois eventos :

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

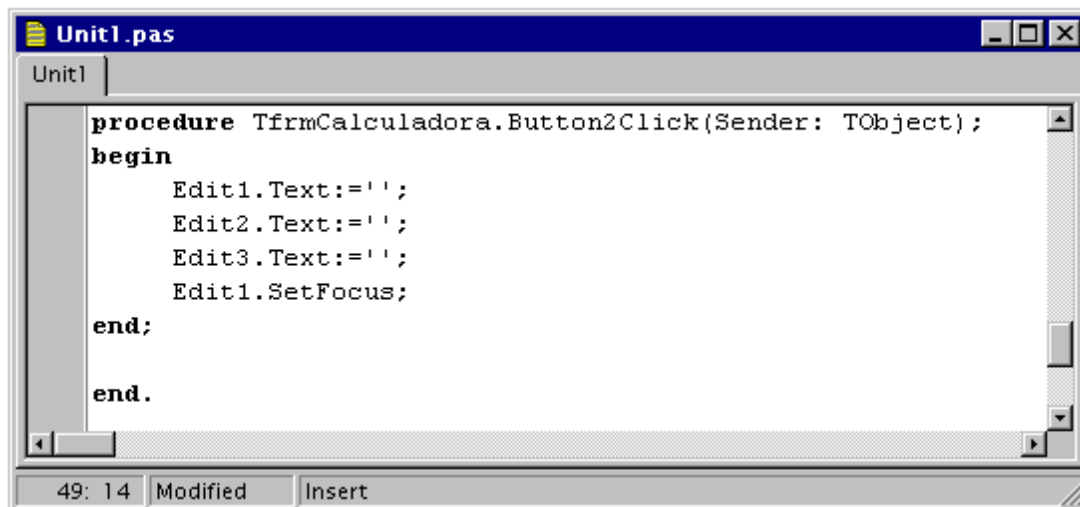
Então, para escrevermos o código, daremos dar um Duplo Click no Button1 - janela Unit será mostrada. Entre com o código mostrado na figura a seguir:



```
Unit1.pas
Unit1
procedure TfrmCalculadora.Button1Click(Sender: TObject);
Var
    Valor1: Real;
    Valor2: Real;
begin
    Valor1:=StrToFloat(Edit1.Text);
    Valor2:=StrToFloat(Edit2.Text);
    Edit3.Text:=FloatToStr(Valor1 * Valor2);
end;
```

35: 23 Modified Insert

Altere para o procedimento do Button2. E entre com os comandos a seguir:



```
Unit1.pas
Unit1
procedure TfrmCalculadora.Button2Click(Sender: TObject);
begin
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit1.SetFocus;
end;
end.
```

49: 14 Modified Insert

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

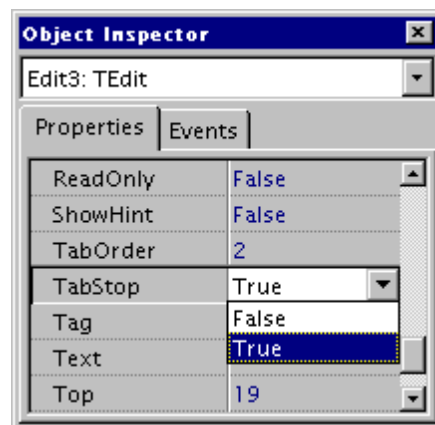
Note que alternamos os campos ativos com a tecla **Tab**. A ordem de tabulação corresponderá à ordem em que os controles foram colocados no formulário. Esta ordem é determinada pela propriedade **TabOrder** dos controles, caso o seu projeto não esteja, coloque-o na seguinte ordem:

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.

Execute o projeto e observe a alteração.

Note que podemos alterar o valor de Edit3 mesmo após a multiplicação ter sido efetuada. Para evitar isso, defina a propriedade **TabStop=False** para Edit3 e verá que o usuário não terá mais acesso com a tecla **Tab** ao Edit3.

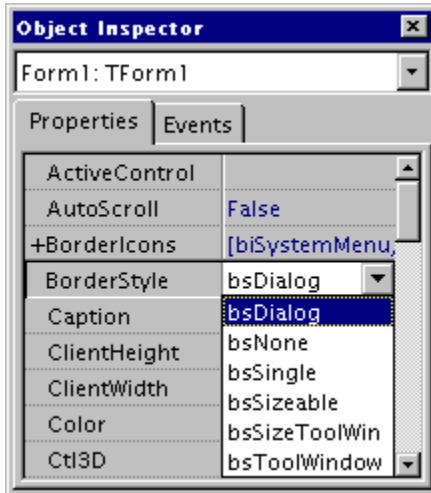


Existem, nas aplicações para Windows, botões de comando que são acionados com a tecla **Enter** ou com um **Click** neles. No nosso projeto este botão será o **Button1**, por isso, a propriedade **Default** foi selecionada para **True**. Fazendo aparecer um contorno mais espesso no botão, dando a indicação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

PROPRIEDADES**BorderStyle**

Retorna ou dá o estilo de borda de um objeto;

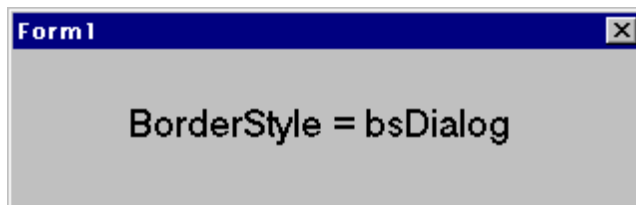
objeto.BorderStyle := [valor]



Existem 6 tipos de bordas:

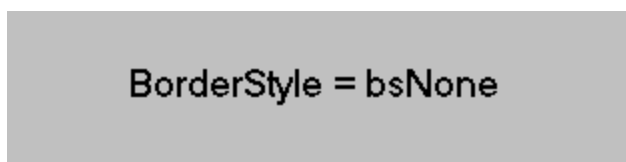
1 - bsDialog

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



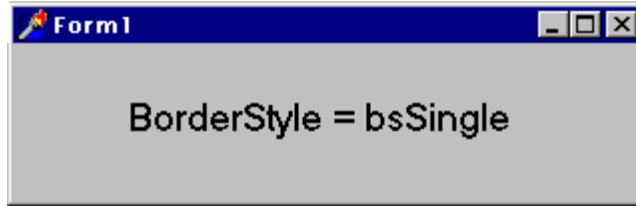
2 - bsNone

Nenhuma



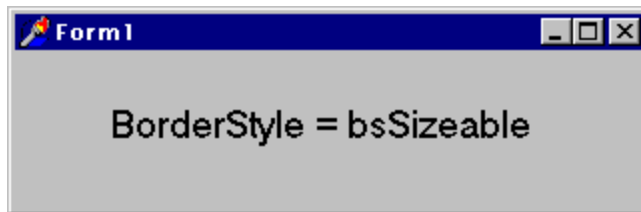
3 - bsSingle

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



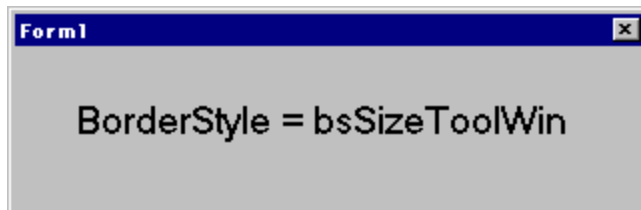
4 - bsSizeable

Redimensionável



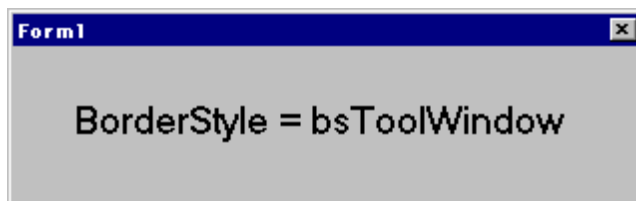
5 - bsSizableToolWindow

Semelhante à bsToolWindow, mas é dimensionável.



6 - bsToolWindow

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



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 conterà 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 visíveis.

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.

Tipos Inteiros	Número de Bytes	Faixa
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
Tipos Booleanos		
Boolean	1	1 byte booleano
ByteBool	1	Byte - sized Booleano
WordBool	2	Word - sized Booleano
LongBool	4	Double - word - sized Booleano
Tipos Reais		
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

```
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 dado a ser exibido, usaremos a função;

`FormatFloat(formato , expressão)`, onde:

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

Formatando números:

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

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

Formatando Data e Hora:

Para formatar data e hora usamos a função:

```
FormatDateTime (formato , data);
```

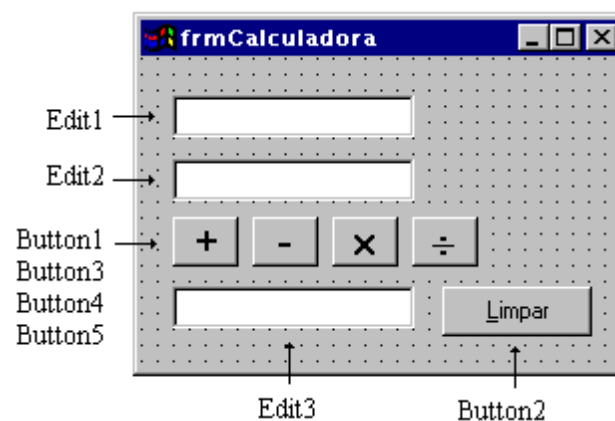
Formato	Exibido
d/m/yy	10/7/96
dd-mm-yyyy	01-Jun-1996
dd-ddd	02-dom
hh:mm AM/PM	08:50 AM
h:mm:ss a/p	8:50:20 a
d/m/yy h:mm	03/12/95 9:30

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 na janela Unit. Selecione a janela Unit, e observe que a procedure TfrmCalculadora.Button1Click continua no mesmo lugar.

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

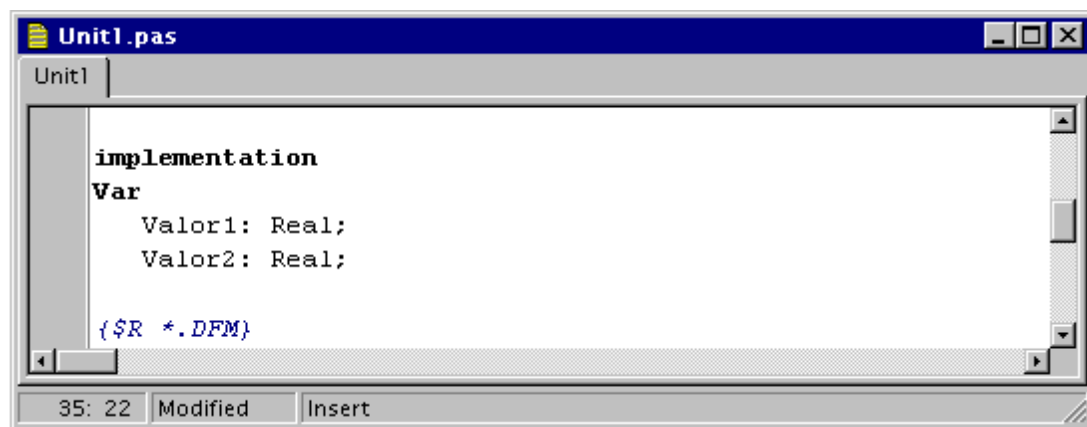
Na figura aparecem as propriedades Name de cada objeto



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 utilize o recurso de copiar e colar na propriedade Caption deste botão de comando. Provavelmente o caractere que aparecerá na caixa de propriedade não será o mesmo do botão, 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.

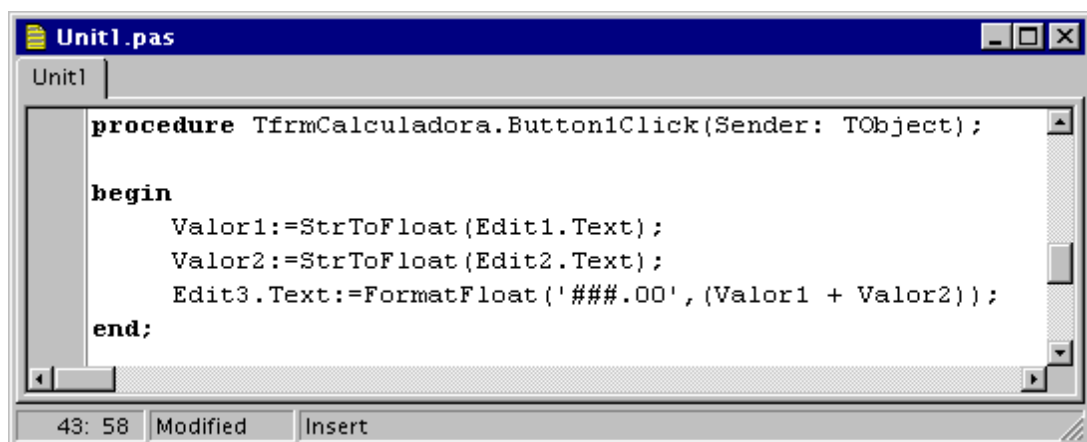


```
Unit1

implementation
Var
  Valor1: Real;
  Valor2: Real;

{$R *.DFM}
```

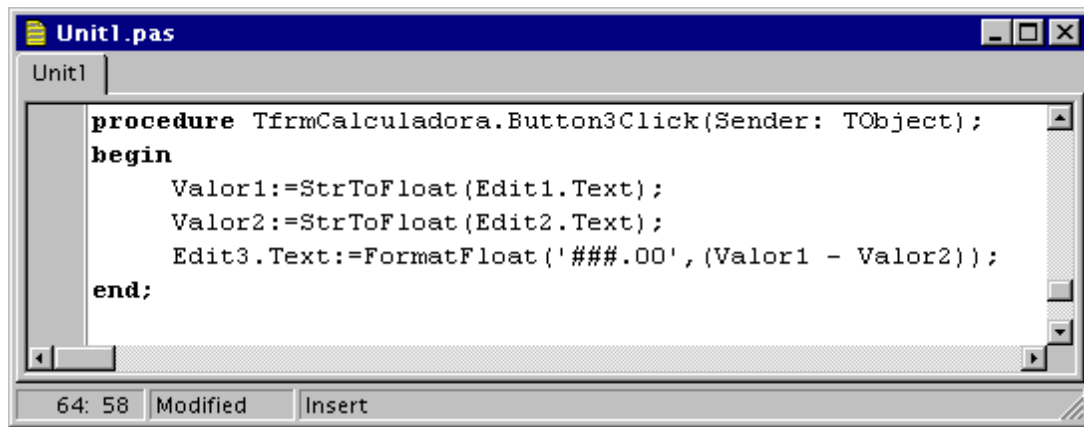
Substitua o tipo de operação em **Button1Click**, e nos demais botões, utilizando as ferramentas de Copiar e Colar.



```
Unit1

procedure TfrmCalculadora.Button1Click(Sender: TObject);

begin
  Valor1:=StrToFloat(Edit1.Text);
  Valor2:=StrToFloat(Edit2.Text);
  Edit3.Text:=FormatFloat('###.00', (Valor1 + Valor2));
end;
```

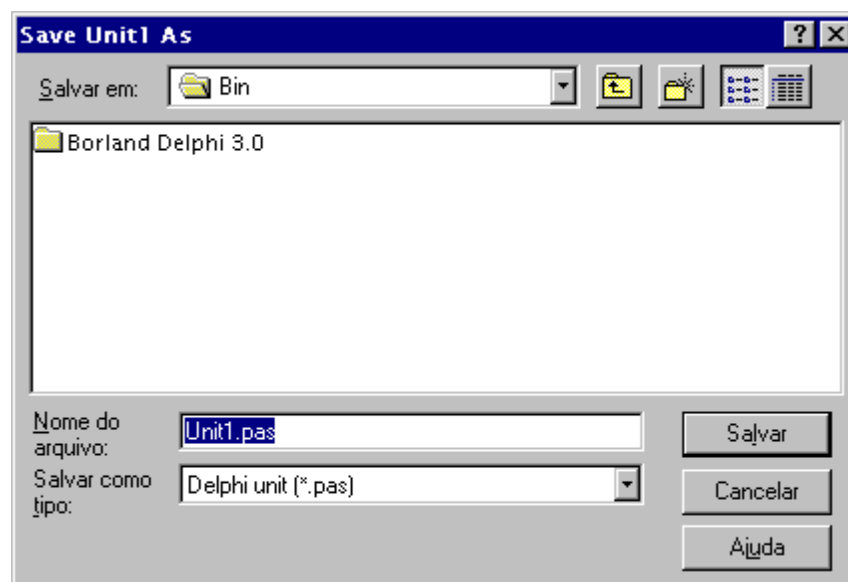


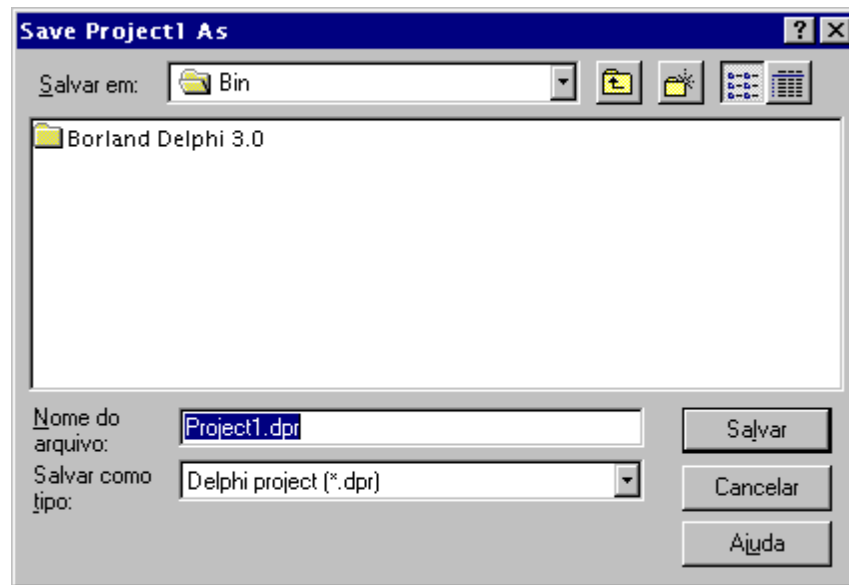
Teste os vários formatos de apresentação dos números, alterando o formulário de apresentação da função **FormatFloat**.

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

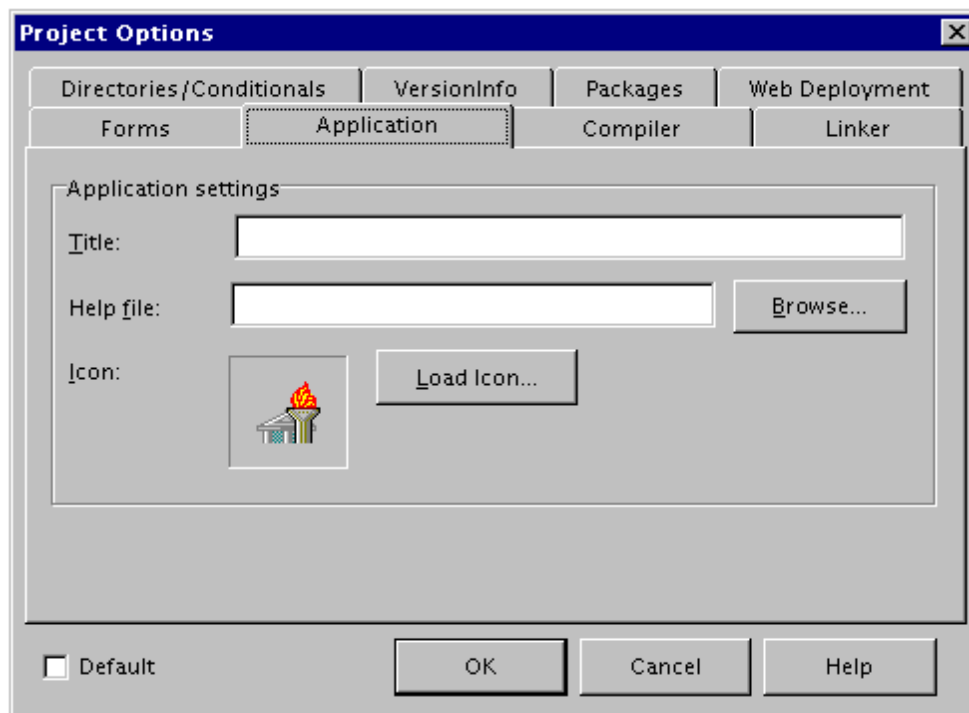
Vamos salvar o nosso projeto Calculadora.

No menu, selecione **File / Saveall** 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 *calculadora* 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 *calculador.dpr* e clique em **Salvar**. Os nomes dos arquivos da Unidade e do Projeto deverão ser diferentes, apesar da extensão já o ser.

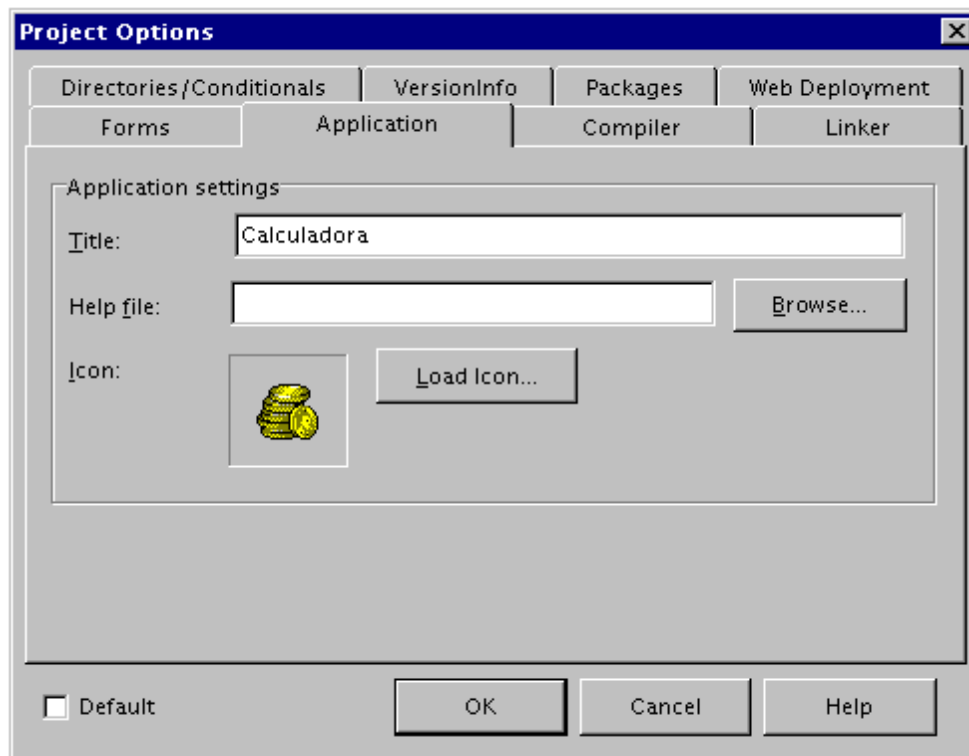




O nosso projeto está salvo. Agora, precisamos escolher um ícone que o representará na tela do Windows. No menu principal do Delphi, selecione **Project / Options...**, e escolha a página **Application**:



Clique no botão **Load Icon...** , e procure o ícone **Finance.ico**, no diretório **Images\Icons** do Delphi, escolha **Abrir** para voltar à janela *Project Options*. Dê um título ao programa e pronto, o programa já possui um ícone e um título associados. Verifique no Explorando.



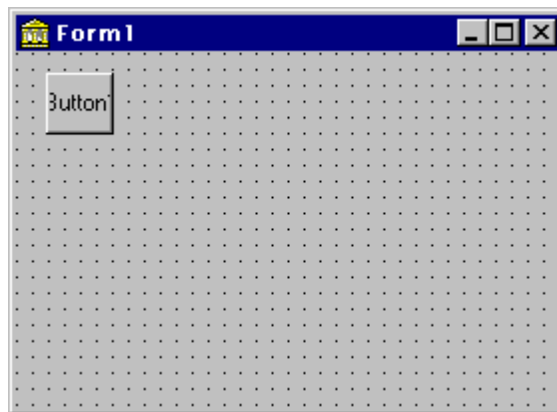
Execute novamente o projeto para o Delphi gerar um novo arquivo executável contendo as últimas alterações realizadas. Agora, você tem um programa executável em qualquer microcomputador que possua o sistema Windows 95, sem necessariamente ter o DELPHI instalado.

EXEMPLO II - JOGO DA VELHA

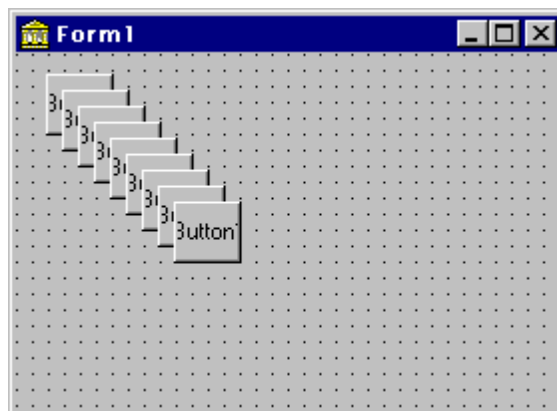
Para iniciar um novo projeto, selecione **New Application** do menu **File**. Caso você ainda não tenha salvo o seu projeto corrente, o Delphi abrirá as janelas necessárias para salvar o projeto. E só então iniciará o novo projeto.

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.

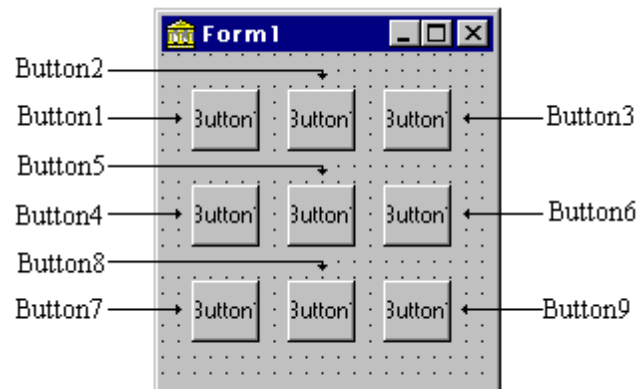
Insira um botão de comando no Formulário dimensionando-o como um quadrado, como mostra a figura.



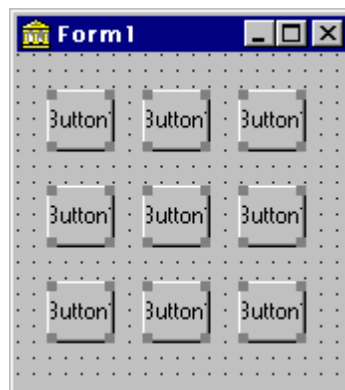
Selecione o Botão, Copie-o (Ctrl+C) e Cole (Ctrl+V) mais oito botões iguais. Formando um total de nove botões.



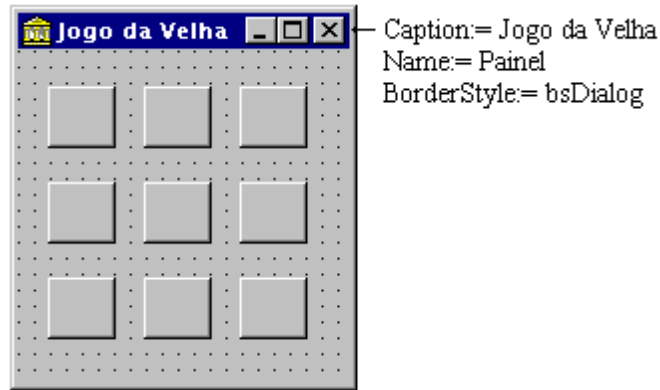
Estes botões de comando estão nomeados de Button1 a Button9, de cima para baixo. Arraste-os de forma que fiquem como a figura abaixo.



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.



Uma Unit possui as seguintes seções:

- Cabeçalho
- Interface
- Implementação
- Inicialização

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.

```

UNIDADE <nome da unidade>;

INTERFACE
    Seção de interface
IMPLEMENTATION
    Seção de implementação
FIM.
  
```

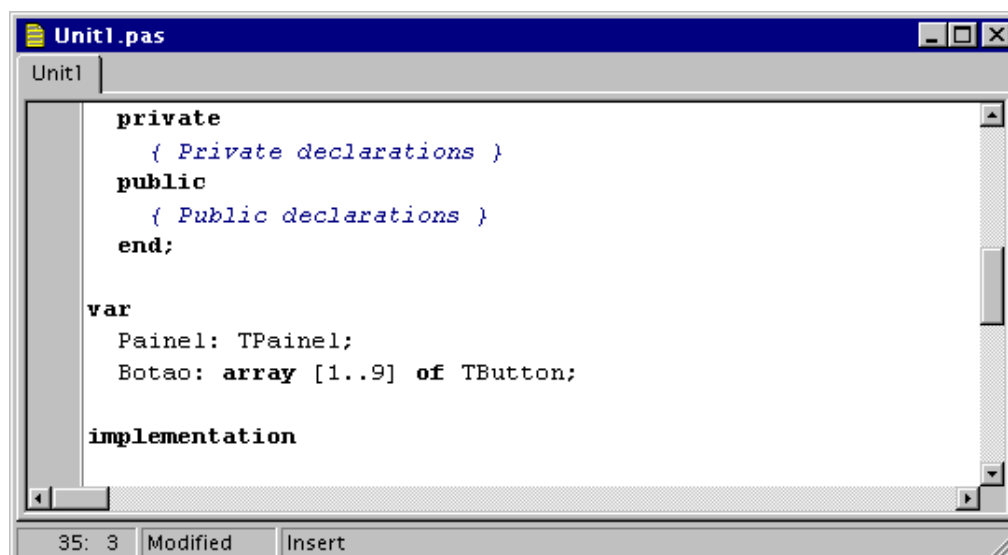
Na seção de `Interface` da nossa unidade está sendo definido o tipo `TPainel` (o nosso Formulário) como sendo um Formulário (`TForm`) e, a seguir, são listados todos os controles, seus tipos, e procedimentos pertencentes a este Formulário.

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

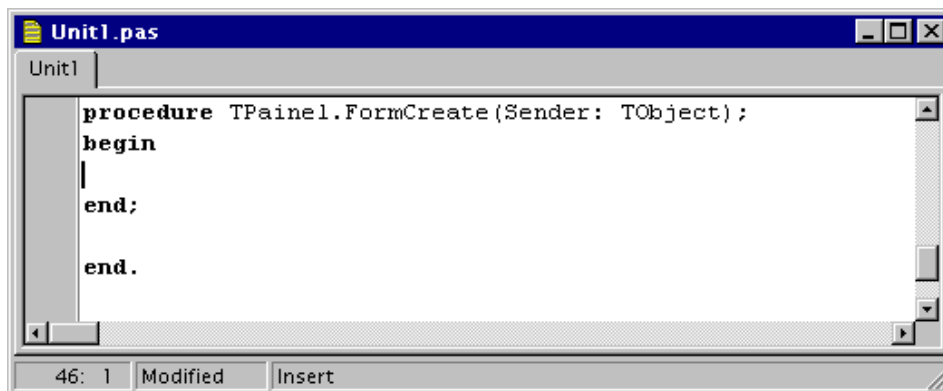
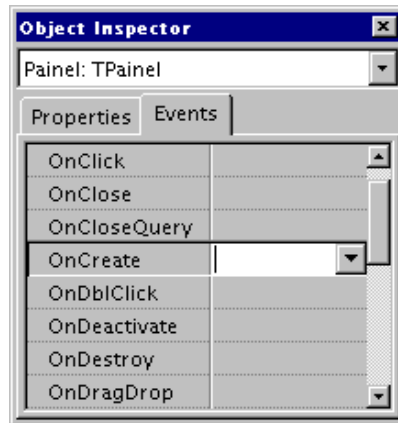
```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
  Dialogs;  
  
type  
  TPainel = class(TForm)  
    Button1: TButton;  
    Button2: TButton;  
    Button3: TButton;  
    Button4: TButton;  
    Button5: TButton;  
    Button6: TButton;  
    Button7: TButton;  
    Button8: TButton;  
    Button9: TButton;  
    procedure JogoNovo;  
    procedure Verificar;
```

Os procedimentos devem primeiro ser declarados para depois serem implementados, declare os procedimentos *JogoNovo* e *Verificar* como mostrado acima. A implementação de um procedimento nada mais é que a construção do seu código.

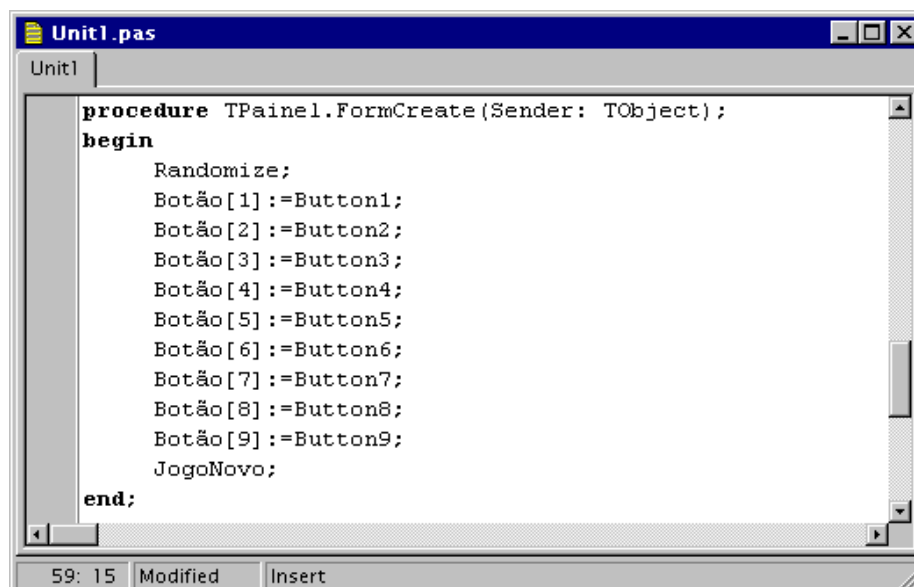
Devemos declarar também a variável **Botao** na seção **Implementation**. Esta variável é uma matriz unidimensional com nove elementos do tipo TButton, ou seja, cada variável dessa possui as mesmas propriedades de um botão de comando. Esta matriz poderia conter números inteiros (**array [1..9] of Integer;**) 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 (Unit), ou selecione o Formulário, vá até a janela **Object Inspector** escolhendo o indicador **Events**, e dê um duplo clique em **OnCreate**. Aparecendo a janela Unit com o cabeçalho do procedimento.

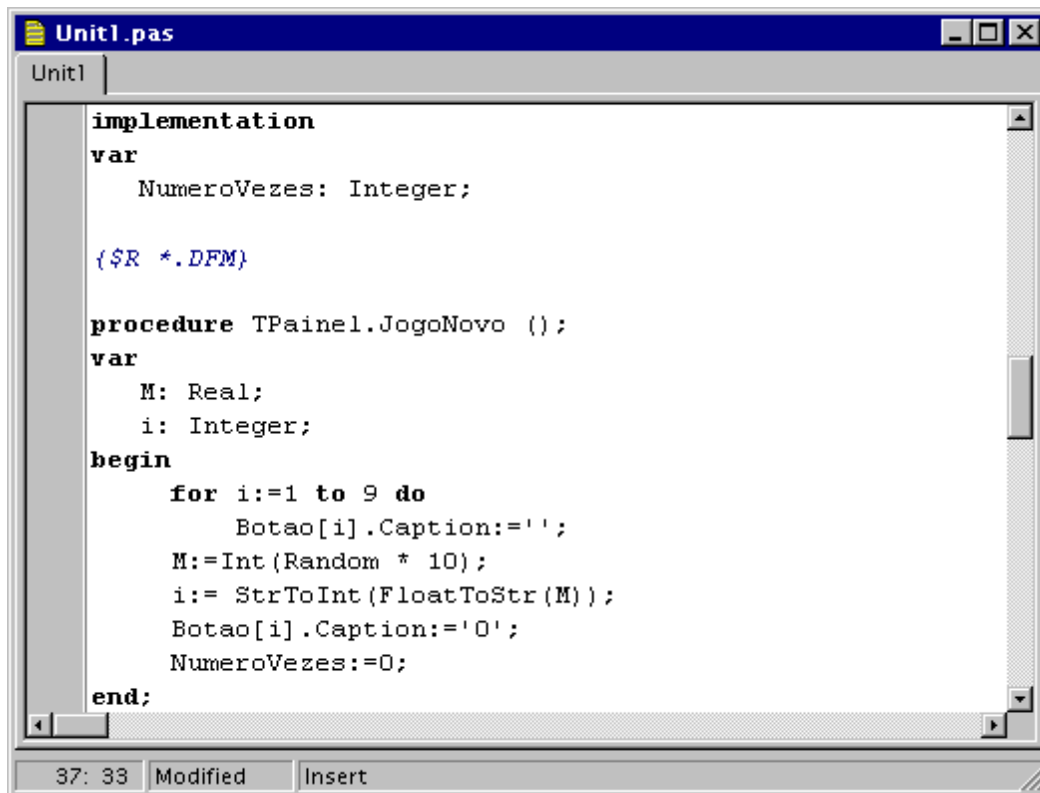


Entre com o código mostrado na figura abaixo:



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.

Digite a procedure JogoNovo na seção Implementation, como segue. Declare também a variável NumeroVezez que poderá ser utilizada por todos os procedimentos desta Unidade.



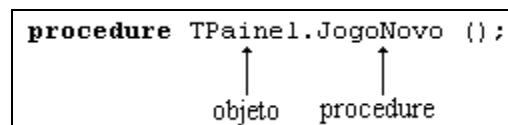
```

implementation
var
    NumeroVezez: Integer;

    {$R *.DFM}

procedure TPainel.JogoNovo ();
var
    M: Real;
    i: Integer;
begin
    for i:=1 to 9 do
        Botao[i].Caption:='';
    M:=Int(Random * 10);
    i:= StrToInt(FloatToStr(M));
    Botao[i].Caption:='0';
    NumeroVezez:=0;
end;
  
```

No cabeçalho, observamos que o nome da procedure é composto do nome do objeto que contém a procedure e o nome da procedure, declarada anteriormente.



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

As variáveis M: Real; e i: Integer; são de uso exclusivo deste procedimento. Findo o procedimento, elas deixarão 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 :

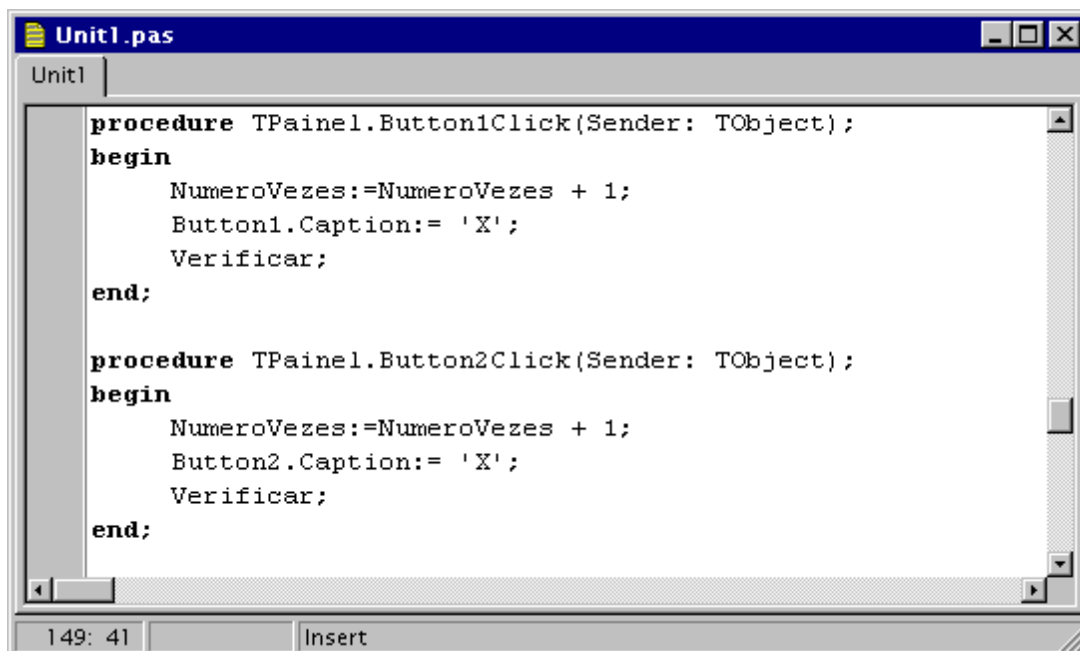
```
for i:=1 to 3 do
  begin
    Botao[i].Caption:='';
    Botao[i+1].Caption:='';
    Botao[i+2].Caption:='';
  end;
```

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.

Primeiro convertemos este número em um String usando a função **FloatToString**, e depois convertemos o String em um número do tipo Inteiro com a função **StringToInt**.

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.



```
Unit1
procedure TPainel.Button1Click(Sender: TObject);
begin
  NumeroVezez:=NumeroVezez + 1;
  Button1.Caption:= 'X';
  Verificar;
end;

procedure TPainel.Button2Click(Sender: TObject);
begin
  NumeroVezez:=NumeroVezez + 1;
  Button2.Caption:= 'X';
  Verificar;
end;
```

O evento `OnClick` nos botões é que dará partida ao programa. O procedimento associado a este evento, incrementará o número de jogadas (`NumeroVezes`), 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 TPainel.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:=StrToInt(FloattoStr(Int (Random *10))) until
      Botao[i].Caption=' ' ;

    Botao[i].Caption:='0';
    if (Button1.Caption='0') and (Button2.Caption='0')
      and (Button3.Caption='0') then goto 2;
    if (Button4.Caption='0') and (Button5.Caption='0')
      and (Button6.Caption='0') then goto 2;
    if (Button7.Caption='0') and (Button8.Caption='0')
      and (Button9.Caption='0') then goto 2;
    if (Button1.Caption='0') and (Button4.Caption='0')
      and (Button7.Caption='0') then goto 2;
    if (Button2.Caption='0') and (Button5.Caption='0')
      and (Button8.Caption='0') then goto 2;
    if (Button3.Caption='0') and (Button6.Caption='0')
      and (Button9.Caption='0') then goto 2;
    if (Button1.Caption='0') and (Button5.Caption='0')
      and (Button9.Caption='0') then goto 2;
    if (Button3.Caption='0') and (Button5.Caption='0')
      and (Button7.Caption='0') 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:= ''`.

O Botão encontrado anteriormente será preenchido com “0”, 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 aviso 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.

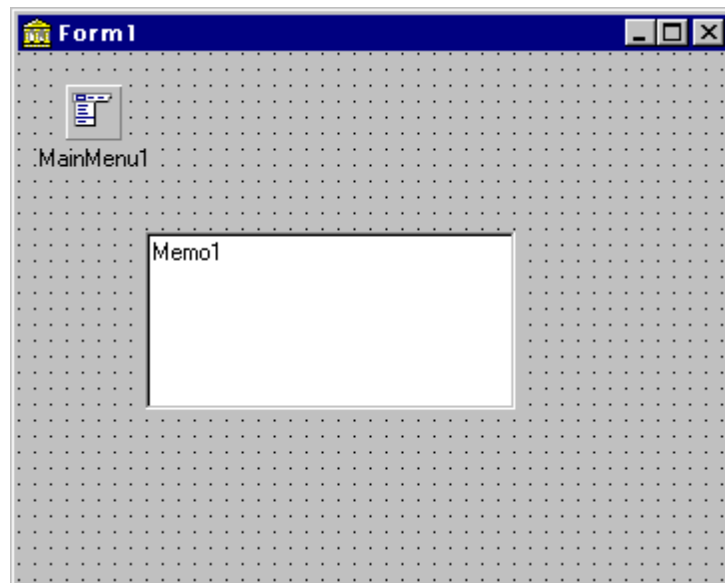
Valor	Significado
1	Botão OK foi pressionado
2	Botão Cancelar foi pressionado
3	Botão Anular foi pressionado
4	Botão Repetir foi pressionado
5	Botão Ignorar foi pressionado
6	Botão Sim foi pressionado
7	Botão Não foi pressionado

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

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 ela seja redimensionada.

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.

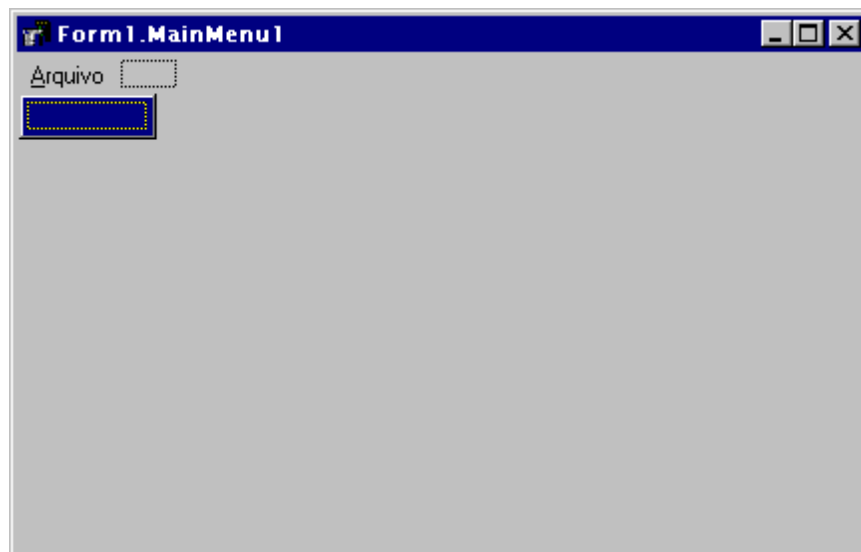
O nosso exemplo de Bloco de Notas, usará um menu e quatro caixas de diálogo.

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.

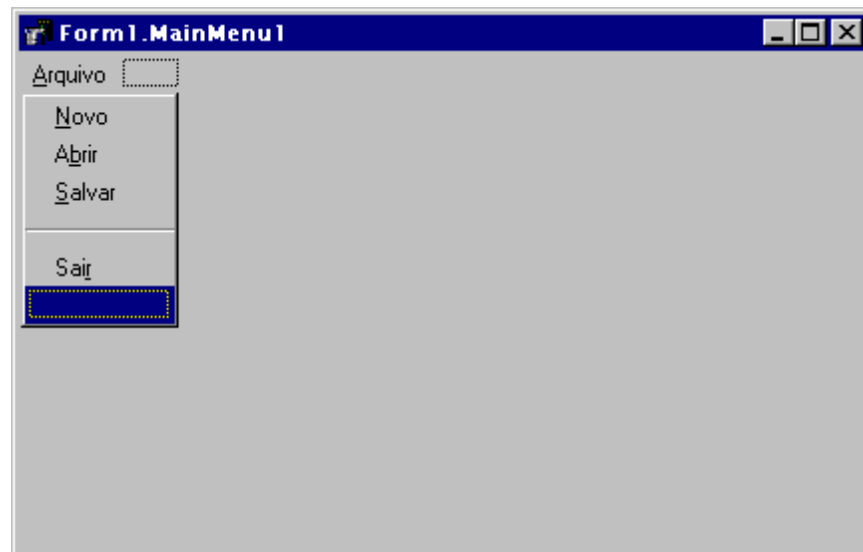


É nesta janela que iremos construir o menu do nosso exemplo. Observe que o primeiro título já está selecionado.

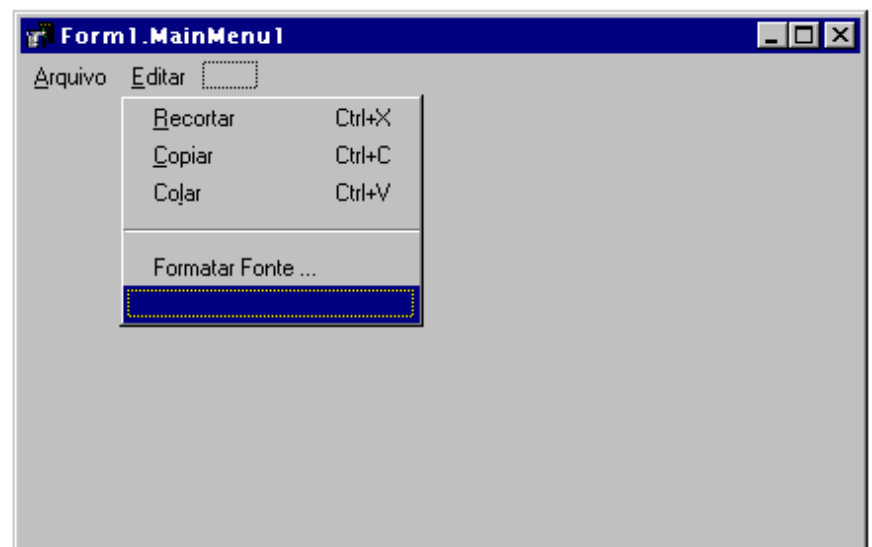
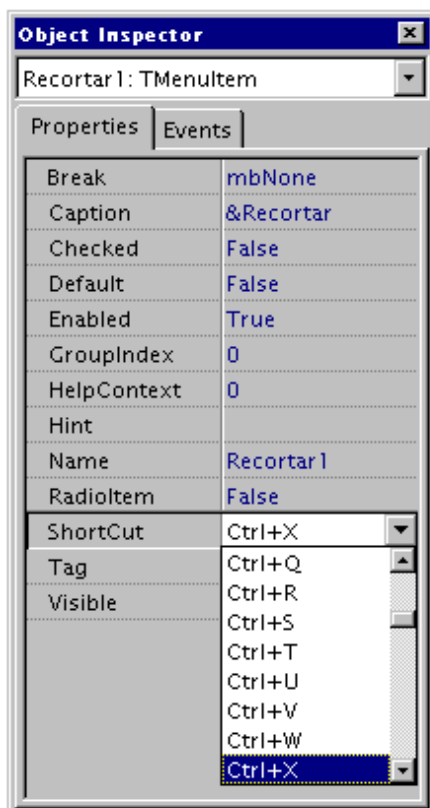
Vá até a janela **Object Inspector** e mude a propriedade **Caption** para **&Arquivo** e pressione Enter - para acesso via teclado, usamos o “&” comercial antes da letra que queremos que seja o atalho. Este procedimento, cria o menu Arquivo e move a barra de destaque para baixo, para podermos digitar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



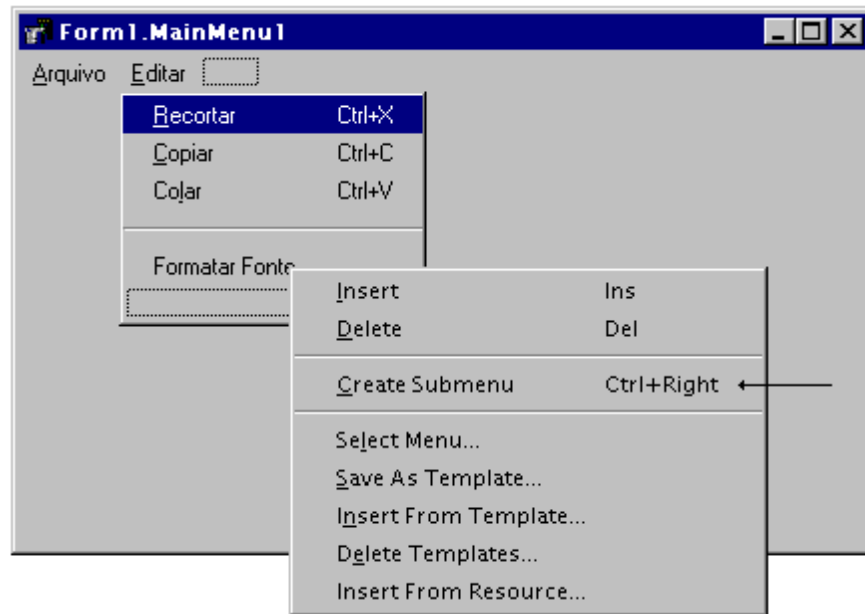
Monte as demais opções do nosso menu seguindo a janela Menu Designer mostrada abaixo. Para criar um separador no menu, digite apenas um sinal de menos (-) na propriedade Caption do item abaixo de Salvar.



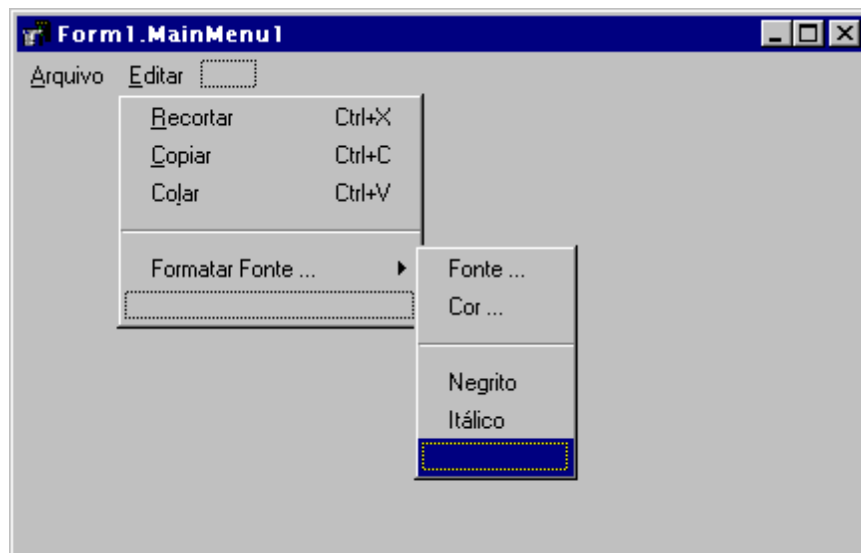
Terminado o menu Arquivo, inicie o menu Editar, como segue abaixo. Defina as teclas de atalho na propriedade **ShortCut** dos itens de menu.



Quando chegarmos ao item **Formatar Fonte...**, exiba o menu local (dê um clique com o botão direito em cima do item) e escolha **Create Submenu**, 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 acentuados e nem brasileiros, o nome que ele dará para o item Itálico será Itlico, suprimindo a letra á acentuada (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).



Acabamos de desenhar o nosso Formulário, colocando todos os componentes a ele pertencentes. Tanto os visíveis como 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**. Quando usamos o método Execute, ele responde True se o usuário tiver selecionado 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 nosso 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 abaixo:

```
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 de 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, e não será tratada pelo compilador do Delphi.

```
procedure TPrincipal.Cor1Click(Sender: TObject);  
begin  
    ColorDialog1.Color:=Mem1.Color;  
    if ColorDialog1.Execute then  
        Mem1.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  
    Mem1.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.Mem1Change(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  
    Mem1.CutToClipboard;  
    Colar1.Enabled:=True; {habilita o item Colar}  
end;
```

```
procedure TPrincipal.Copiar1Click(Sender: TObject);  
begin  
    Mem1.CopyToClipboard;  
    Colar1.Enabled:=True;  
end;
```

```
procedure TPrincipal.Colar1Click(Sender: TObject);  
begin  
    Mem1.PasteFromClipboard;  
end;
```

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

Comece declarando o procedimento Fonte na seção de definição do tipo TPrincipal - Nosso Formulário. E depois, implemente-a na seção Implementation.

```
procedure Novo1Click(Sender: TObject);  
procedure Abrir1Click(Sender: TObject);  
procedure Salvar1Click(Sender: TObject);  
procedure FontelClick(Sender: TObject);  
procedure Cor1Click(Sender: TObject);  
procedure FormCreate(Sender: TObject);  
procedure Mem1Change(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;
```

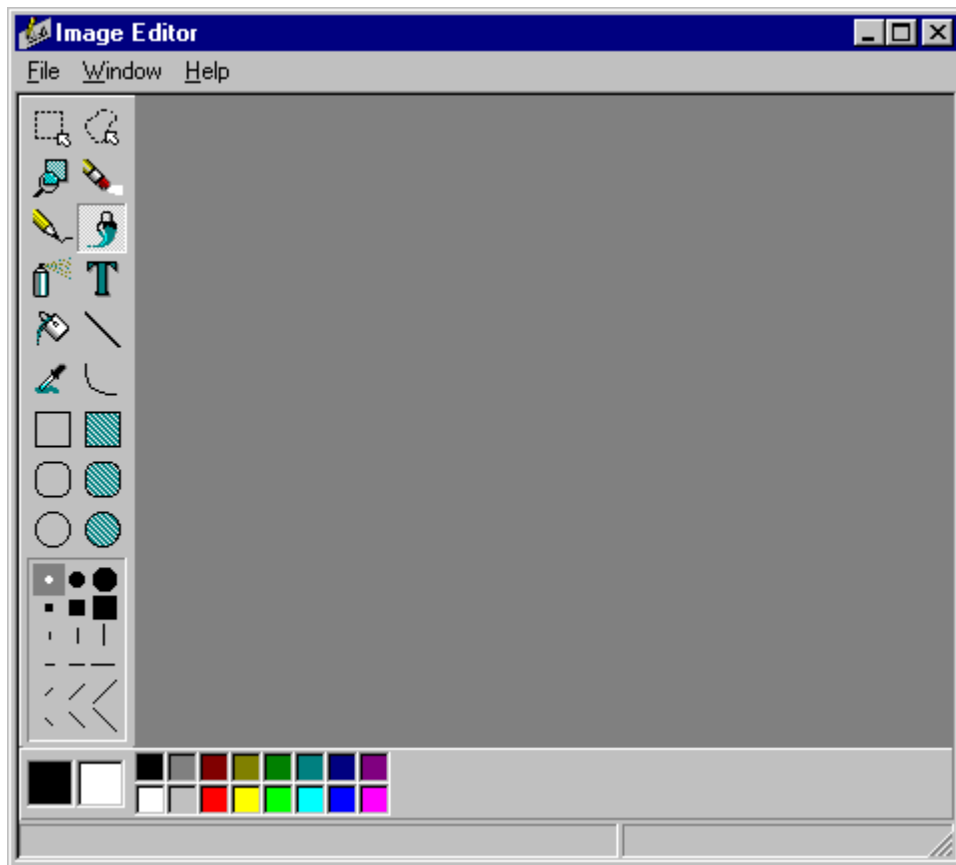
```
procedure TPrincipal.ItlicolClick(Sender: TObject);
begin
  Itlicol.Checked:= Not Itlicol.Checked;
  Fonte;
end;
```

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.

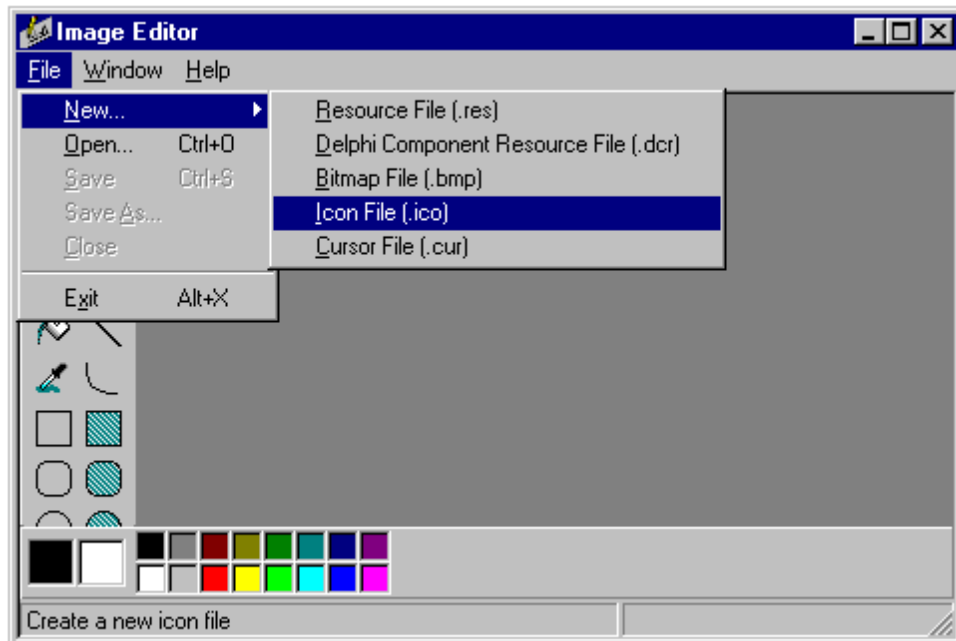
```
procedure TForm1.Sair1Click(Sender: TObject);
begin
  Close;
end;
```

Salve seu trabalho, e teste o programa pressionando F9.

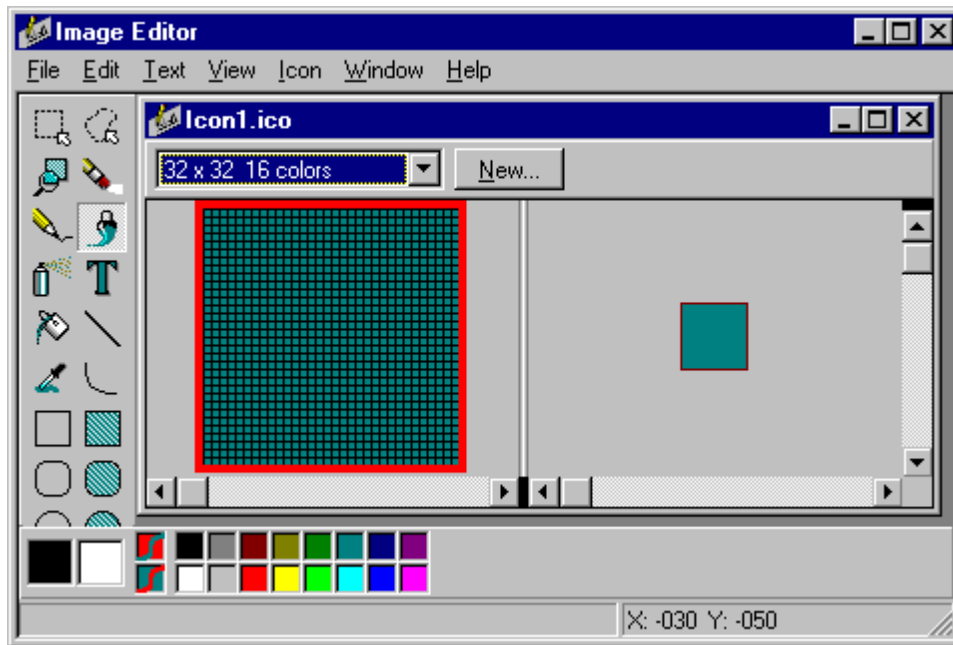
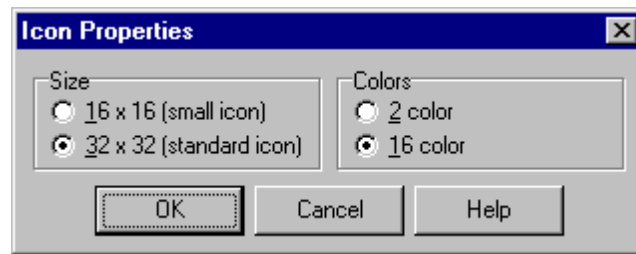
Nós já aprendemos a associar um ícone ao nosso projeto, mas agora iremos aprender a desenhar um utilizando o aplicativo Image Editor do Delphi. Para inicializá-lo, vá até a opção Tools do menu principal do Delphi, e escolha Image Editor.



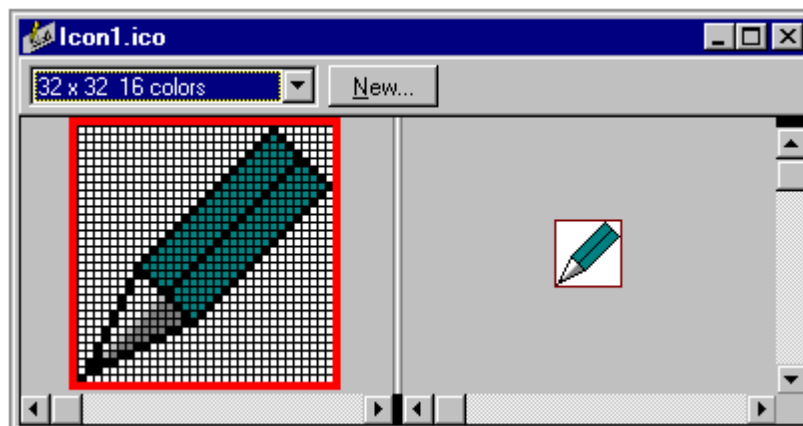
Escolha a opção **New** do menu **File**, e **IconFile**.



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

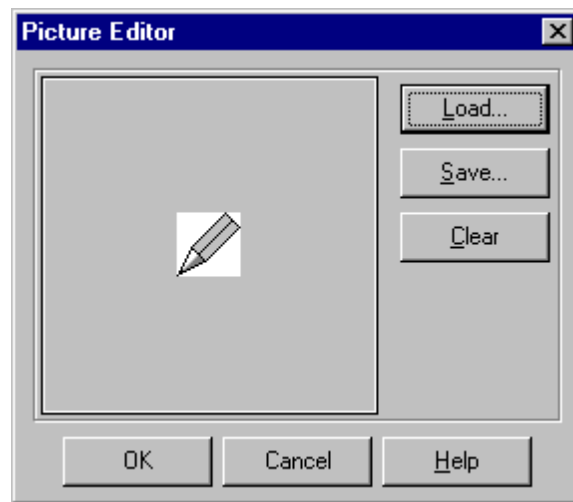
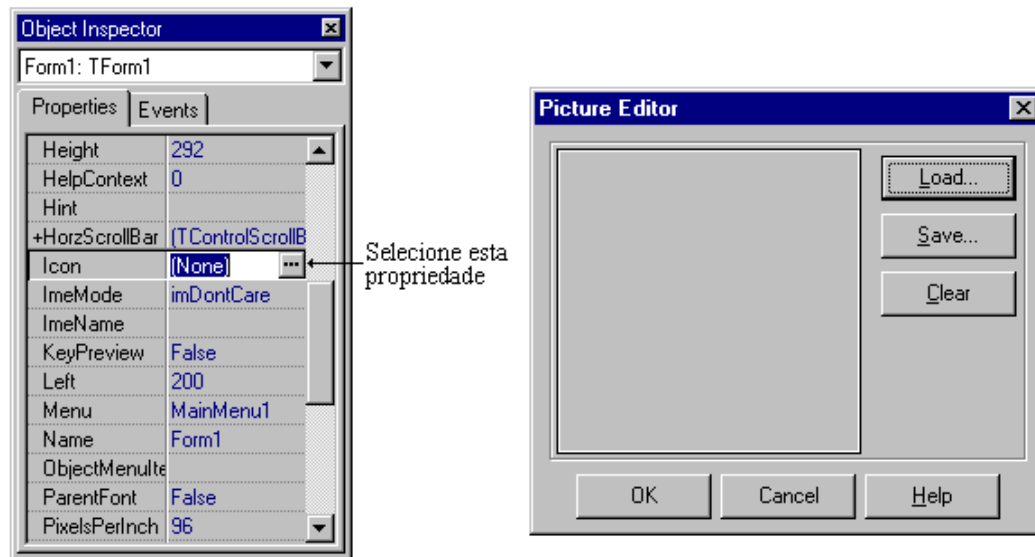


Desenhe um ícone semelhante ao da figura abaixo, utilizando recursos semelhantes ao programa Paint do Windows.

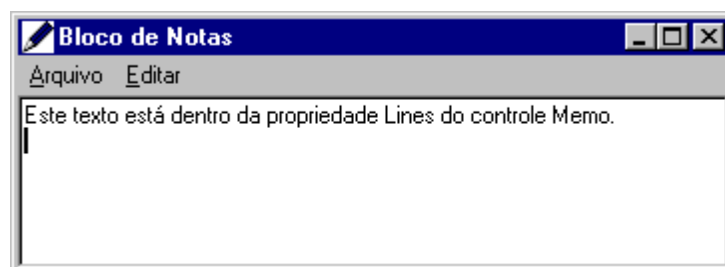


Salve o seu ícone e memorize o nome completo incluindo o caminho. E feche o aplicativo Image Editor.

O nosso 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 mesmo ícone para representar o Bloco de Notas no Windows, da mesma forma descrita para a Calculadora.

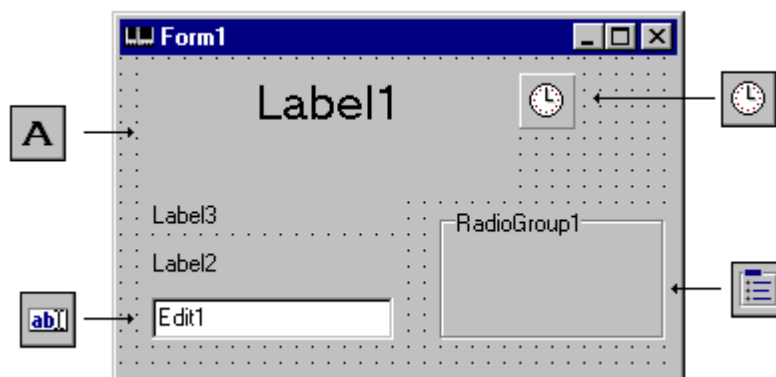


EXEMPLO IV - RELÓGIO DESPERTADOR

Este projeto contém outro componente não visível - o TTimer. 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.

Construa o Formulário como mostrada abaixo.



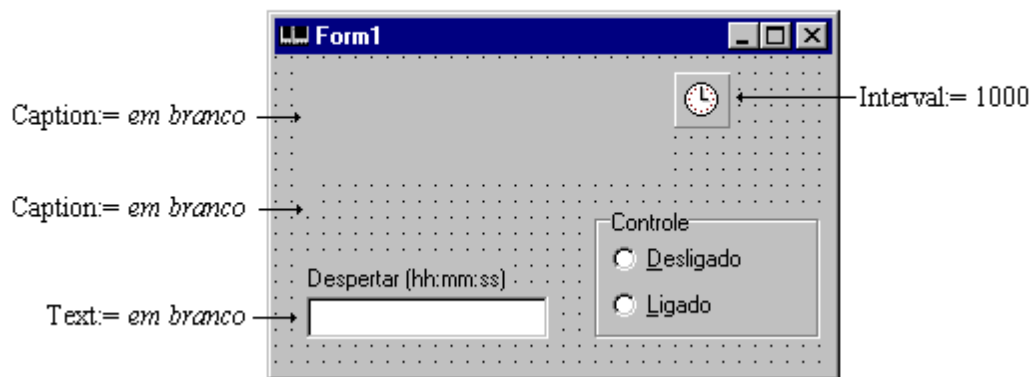
A propriedade Caption dos Label1 e Label3 deverá estar em branco. No Label1, será apresentada a hora do sistema, e em Label3 a data atual.

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, onde poderemos editar os botões de opção do grupo. Entre com as duas opções, como mostrado a seguir.



Forma pronta



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, selecionaremos o botão Desligado, utilizando a sintaxe "RadioGroup1.ItemIndex:= 0", a propriedade ItemIndex, indica qual o botão do grupo estará selecionado.

Colocaremos a data do sistema em Label3 usando a função FormatDateTime, de acordo com a tabela mostrada no projeto da Calculadora.

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    RadioGroup1.ItemIndex := 0;  
    Label3.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 evento KeyPress ocorre. 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 excluído do Edit.

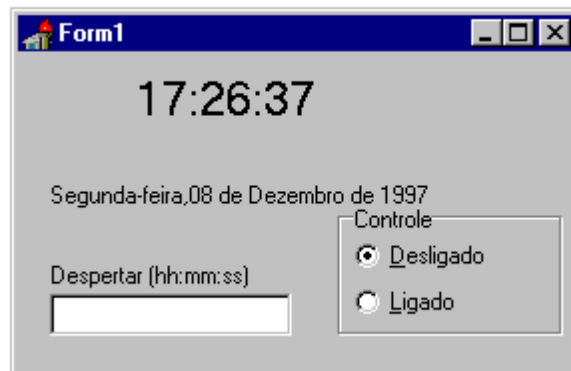
O evento KeyPress envia para a procedure, mostrada abaixo, a variável Key do tipo Char, e será esta variável que o programa verificará.

```
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 no 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;
```

Programa 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 tais 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:

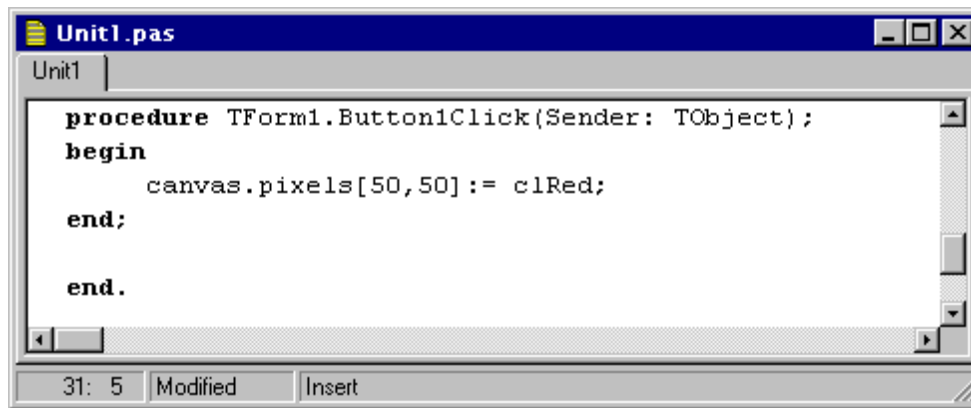
Canvas - é o objeto do tipo TCanvas, declarado desta forma, o ponto será desenhado na área do cliente - Formulário.

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 nas coordenadas (50,50) do formulário na cor vermelha, quando o botão for selecionado.





CORES

Para determinarmos a cor de um gráfico, temos 3 formas diferentes de atribuir valores para o parâmetro *cor*, 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, and 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, and clWindowText.

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

DESENHO DE LINHA

Para desenharmos no Windows, usamos a propriedade Pen, 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.

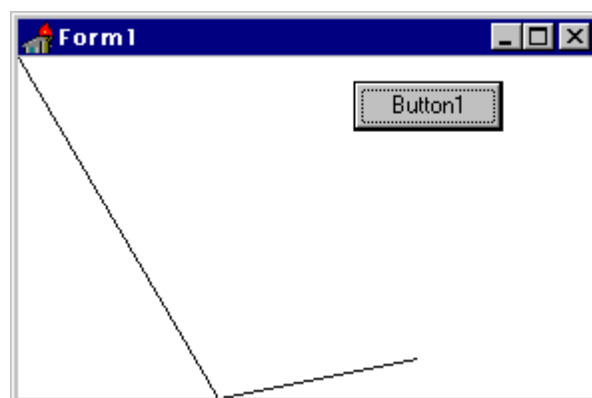
Uma linha é desenhada utilizando-se os métodos MoveTo e LineTo para mover a Pen (caneta). MoveTo posiciona a caneta em um determinado lugar no Formulário. LineTo traça uma linha, que vai da posição corrente (PenPos) até as coordenadas especificadas.

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
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**. 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.

No exemplo acima, utilizamos este recurso com o objeto TCanvas 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;

```

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;

```

Começamos declarando uma matriz - Estilo - com o 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.

```

Canvas.Pen.Style:= psDot;
Canvas.Pen.Style:= psDot;

```

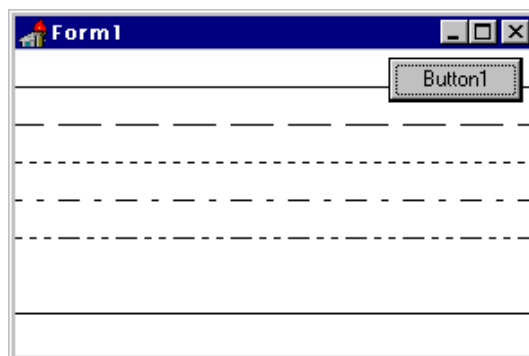
ou

```

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

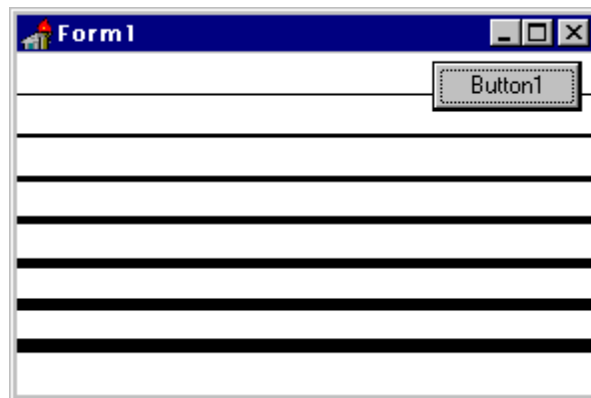
```

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:



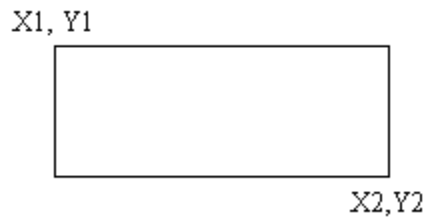
Altere o procedimento anterior, para serem mostradas as várias espessuras de linha.

```
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}  
            {Os comentários ficam entre chaves}  
            for i:= 0 to 6 do  
                begin  
                    Pen.Width:= i+1;  
                    MoveTo(0, (i*SY)+20);  
                    LineTo(SX, (i*SY)+20);  
                end;  
            end;  
    end;
```



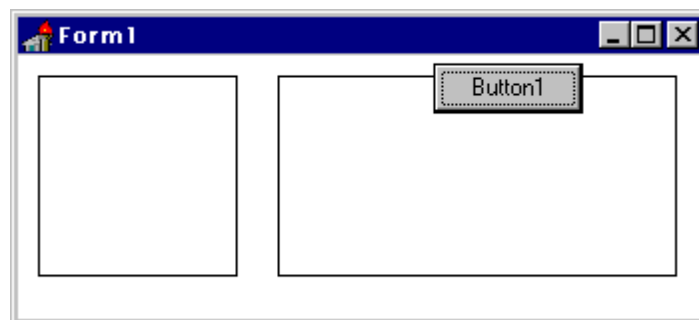
Para desenharmos retângulos usamos o método `Rectangle` do `TCanvas`. Como parâmetros são utilizadas as coordenadas do canto superior esquerdo e inferior direito do retângulo.

Canvas.Rectangle (X₁, Y₁, X₂, Y₂);



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.

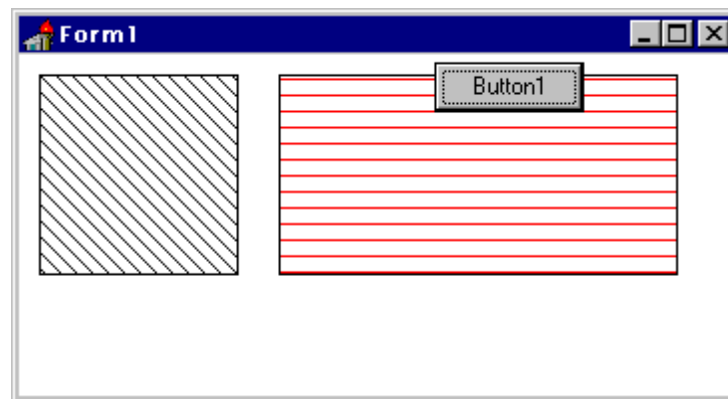
Para preenchermos este retângulos, usamos o tipo **TBrush** que como TPen, está contido em TCanvas. As propriedades mais comuns do TBrush são:

- 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.

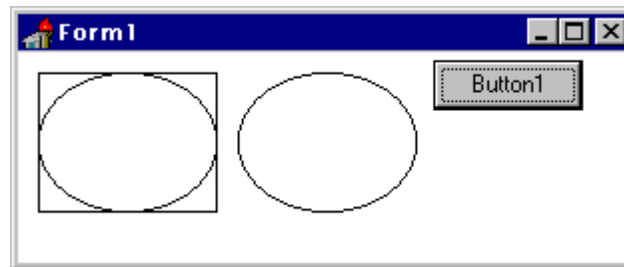
Faça o exemplo a seguir.

```
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;
```



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.

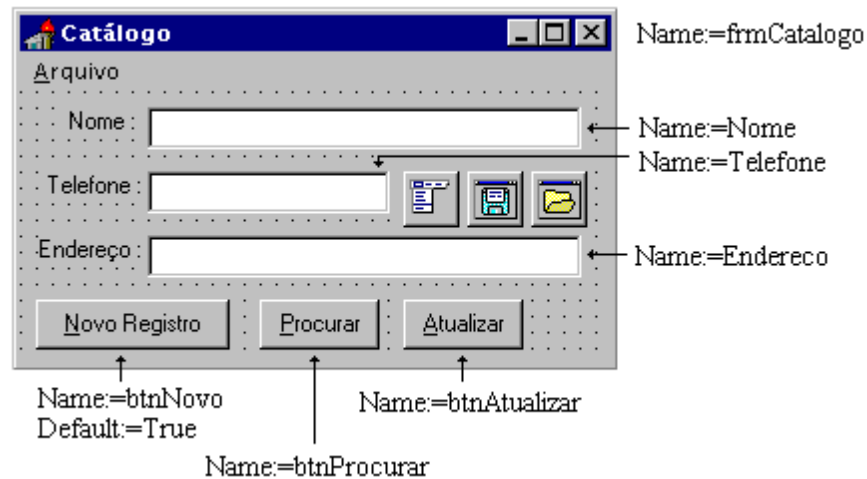
```
procedure TForm1.Button1Click(Sender: TObject);  
  
begin  
  With Canvas do  
    begin  
      Rectangle (10,10,100,80);  
      Ellipse (10,10,100,80);  
      Ellipse (110,10,200,80);  
    end;  
end;
```



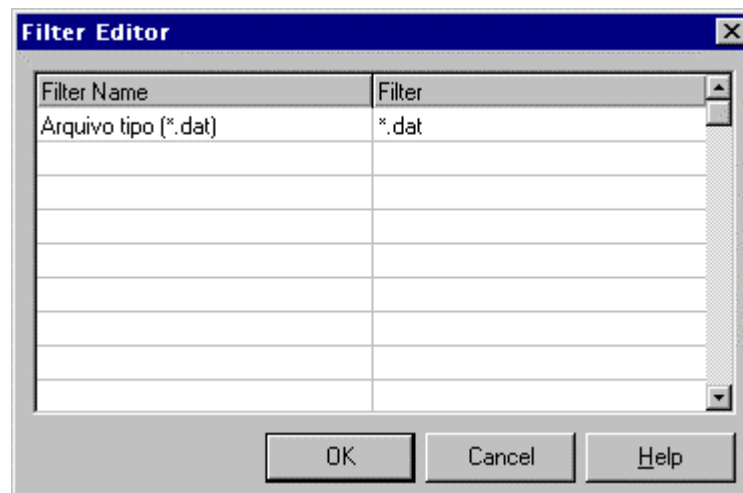
EXEMPLO V - CATÁLOGO

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.




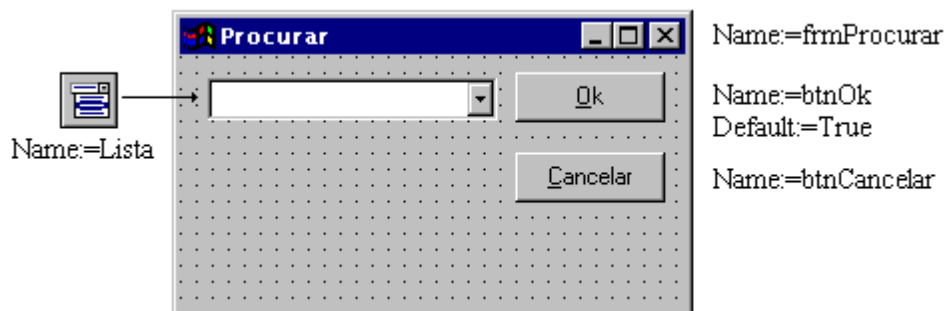
Altere a propriedade Filter do controle SaveDialog. Será então mostrada a janela Filter Editor, edite-a como mostra o modelo abaixo.



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 no 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.

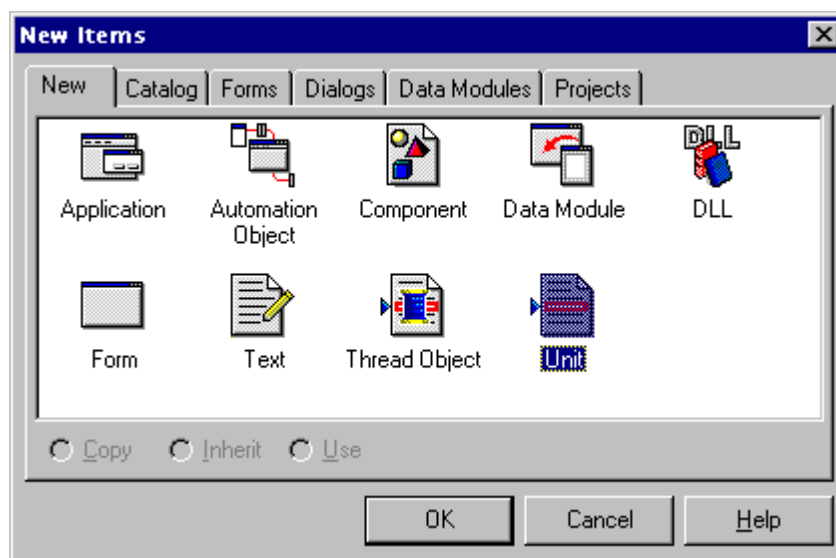
csOwnerDrawFixed - Cria uma lista drop-down com uma caixa de texto, para entrada de texto manualmente. Mas cada item do ComboBox terá seu comprimento em caracteres determinado pela propriedade ItemHeight.

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

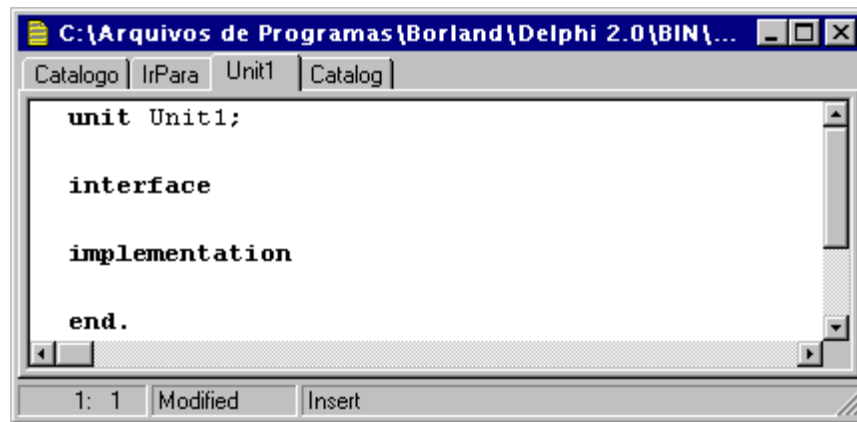
Vamos alterar a propriedade Style do nosso quadro combo, para csDropDownList, pois não queremos que o usuário altere o conteúdo do combo. Evitando que o usuário entre com um nome de pessoa inexistente.

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** 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

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, e 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 Unit nova (ArqUnit - nome salvo em disco).

```
unit ArqUnit;

interface

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

TPessoas = file of TPessoasReg;

var
  PesArq: Pessoas;
```

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 Tpeessoas, 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 declaramos uma variável do tipo desse arquivo de registros, essa variável conterà 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:Pessoas; var NomeArq:String;
                  var N:Boolean);
procedure FecharArq(var A:Pessoas);
procedure GravarArq(var A:Pessoas;var RegN:Integer;
                   var Reg:PessoasReg);
procedure LerArq(var A:Pessoas;var RegN:Integer;
                 var Reg:PessoasReg);
```

```
implementation

procedure AbrirArq(var A:Pessoas; var NomeArq:String;
var N:Boolean);
begin
  Assign (A, NomeArq);
  if N then
    Rewrite (A)
  else
    Reset (A);
end;

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

procedure GravarArq (var A:Pessoas;var RegN:Integer;
var Reg:PessoasReg);
begin
  Seek (A,RegN);
  Write (A,Reg);
end;

procedure LerArq (var A:Pessoas;var RegN:Integer;
var Reg:PessoasReg);
begin
  Seek (A,RegN);
  Read (A,Reg);
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, abriremos o arquivo, que pode ser de duas formas:

- 1) Reset - abre o 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 o registro.

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

PONTEIRO DE REGISTROS

O **ponteiro** do arquivo indica qual o 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 nomeada uma unidade na seção **uses** de outra unidade em que esteja trabalhando, você poderá referenciar qualquer coisa na seção de interface da unidade nomeada. Poderemos acessar as variáveis e procedures da outra unidade.

O nosso formulário `frmCatalogo` (Unit `Catalogo`), terá acesso aos procedimentos da Unit `IrPara` e `ArqUnit`. Então, declaramos estas duas unidades logo após às unidades já colocadas pelo Delphi.

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


Declare as variáveis que iremos utilizar, como mostrado abaixo.

```
private
  { Private declarations }
  ArqNovo : String;
  ArqAtual: String;
  Novo: Boolean;
  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;
  ArqUnit.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;
      ArqUnit.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: ArqUnit.AbrirArq(...);

```
procedure TfrmCatalogo.btnNovoClick(Sender: TObject);
var
  Reg: ArqUnit.PessoasReg;
  RegN: Integer;
begin
  Nome.SetFocus;
  if ArquivoAberto=false then
    begin
      ShowMessage ('Abra primeiro um arquivo');
      Exit;
    end;
  if btnNovo.Caption='&Novo Registro' 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);
      ArqUnit.GravarArq (PesArq, RegN, Reg);
      btnNovo.Caption:='&Novo Registro';
    end;
end;
```

```
procedure TfrmCatalogo.btnAtualizarClick(Sender: TObject);
var
  Reg : PessoasReg;
  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;
  ArqUnit.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;
      ArqUnit.FecharArq (PesArq);
      RenameFile(ArqAtual,ArqNovo);{a função RenameFile,}
      Novo:=False;                {renomeia um arquivo}
      ArqUnit.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 onde podemos alternar com 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 ArqUnit.

```

unit IrPara;

interface

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

```

Declare as variáveis ReN e Reg.

```

private
  { Private declarations }
  RegN: Integer;
  Reg: PessoasReg;

```

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
  frmProcurar.Close;
end;
```

```
procedure TfrmProcurar.btnOkClick(Sender: TObject);
begin
  RegN:=Lista.ItemIndex;
  ArqUnit.LerArq (PesArq,RegN,Reg);
  With frmCatalogo do
    begin
      Nome.Text:=Reg.CNome;
      Endereco.Text:=Reg.CEndereco;
      Telefone.Text:=Reg.CTelefone;
    end;
  frmProcurar.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 Catalogo na seção Implementation. Porque a unidade IrPara também se referencia à Catalogo, mas se Catalogo for declarada na seção **uses** da interface, o Delphi gerará um erro de referência circular.

```
implementation

uses
  Catalogo;

{$R *.DFM}
```

Catálogo

Arquivo

Nome :

Telefone :



Endereço :

Catalog

Abra primeiro um arquivo

Procurar

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. Substitua o controle ComboBox por um ListBox no exemplo de Catálogo, com pesquisa pelo número do telefone.
14. No último exemplo, crie mais uma tabela de Estoque, e a cada venda realizada os valores de estoque deverão ser atualizados.
15. Crie mais duas tabelas, uma de Fornecedores e outra de Compras, semelhantes a Clientes e Vendas, e acrescente os campos ao seu critério.