Muitos anos se passaram desde os primórdios da computação, mas apesar de já termos vivido vários paradigmas de programação, existe uma base de conhecimento que não mudou e não mudará, a lógica de programação.

E a lógica, como vai?

Existe a ilusão de que apenas conhecer uma boa linguagem credencia um bom desenvolvedor de sistema. Todavia, a lógica de programação tem papel fundamental na qualidade do programa.

Qual é a melhor linguagem? Qual é o software mais usado no mercado hoje?

É comum ouvirmos perguntas deste tipo de pessoas que desejam estar na vanguarda da informática. Existe uma grande preocupação de atualizar-se em termos do melhor software de programação. Todavia, alguns, entusiasmados pelo software, esquecem daquilo que os move: a lógica computacional.

Balance Line?

Você já fez um "Balance Line"? Sabe o que é? Este termo (em desuso) significa fazer a consolidação de um ou mais arquivos seqüenciais, pré-ordenados, gerando um novo arquivo. Este era um problema típico a ser resolvido por um programador de Main Frame, quando o processamento Batch – em lotes – era largamente utilizado.

Programas como este exigiam do programador muito mais de sua lógica do que de seu conhecimento da linguagem de programação, ou seja, o maior esforço ficava por conta do raciocínio lógico para resolver o problema do que a codificação da solução em uma linguagem de programação.

O re-trabalho

Quando efetuamos um trabalho e este não está bem feito, será preciso refazê-lo. Um programa mal construído, que não atinge o propósito a que foi concebido, está credenciado a ser refeito, mesmo que contenha requintes da linguagem. Programar "bonito", explorando bem os recursos que a linguagem proporciona não é garantia de que o programa desempenhará o papel esperado.

O que é um programa?

Uma das definições de programa é: "uma seqüência de instruções logicamente ordenadas a fim de solucionar um problema". Isto significa que, se desejamos resolver um problema, é preciso estabelecer quais passos precisamos efetuar e, depois disto, ordená-los de forma lógica a fim de que desempenhem o que deles se espera.

Elaborar regras?

Na verdade, o raciocínio lógico utilizado para resolver um problema deve elaborar uma regra (ou várias regras) a ser seguida a fim de que o problema seja resolvido. A(s) regra(s) deve indicar por qual caminho a solução será alcançada.

E o software?

Uma vez com a regra pronta, sabendo como chegar à solução, a próxima etapa é "traduzir" a regra em código, ou seja, programar a solução utilizando uma linguagem de programação.

E quem programa "direto"?

Você pode estar se perguntando: "mas e quem começa direto pela programação?" Ora, isto é comum nos programadores com maior experiência, mas, mentalmente, a solução foi arquitetada. Não há impeditivo a isto, mas é importante ter-se a solução antes da programação.

Paulo Sergio Borba, 33 anos, é Analista de Sistemas há 13 anos e Professor da Faculdade Radial São Paulo, onde leciona Linguagem e Técnicas de Programação no curso de Processamento de Dados.

Algoritmo

Um algoritmo é formalmente uma seqüência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo a passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por seqüências lógicas. Por exemplo: chupar uma bala e pegar um ônibus.

```
PEGAR A BALA
RETIRAR O PAPEL
CHUPAR A BALA
JOGAR O PAPEL NO LIXO
```

e

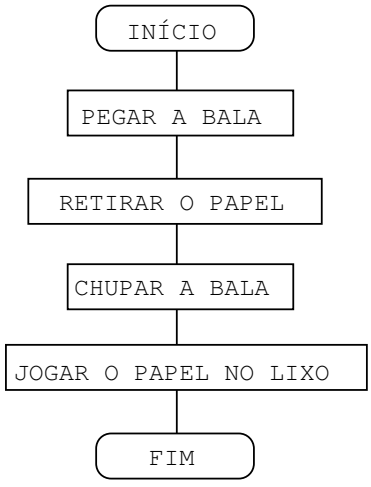
```
ANDAR ATÉ O PONTO DE ÔNIBUS
AGUARDAR O ÔNIBUS
AO AVISTAR O ÔNIBUS CORRETO, FAZER SINAL
ENTRAR NO ÔNIBUS
PAGAR PASSAGEM
SE TIVER LUGAR LIVRE, SENTAR
AO CHEGAR NO LOCAL DESEJADO, DAR SINAL DE DESCIDA
AO PARAR, DESCER DO ÔNIBUS PELA PORTA DIANTEIRA
```

Mas observe ainda o seguinte, se mudarmos a ordem das 2 últimas linhas no algoritmo de chupar uma bala, o mesmo continua a funcionar, isto é, temos várias soluções verdadeiras para um mesmo problema.

O algoritmo é a solução lógica do problema e antes de pensar na solução, procure o maior número de informações possíveis sobre o problema, pois só podemos resolve-lo após o entendimento completo deste. Sempre faça isso fora da ferramenta de trabalho e até mesmo fora do computador, faça funcionar e depois vá melhore tudo com calma.

Diagrama de Bloco

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento. Com o diagrama podemos definir uma seqüência de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.



Conceitos básicos

Para o desenvolvimento de programas, precisaremos de poucos conceitos, que devem se bem fixados, mas de forma gradativa:

Conhecer as variáveis/constantes
Identificadores (letras, palavras, etc) que armazenam valores, fixos ou não, sendo: texto, número (inteiro ou real) e lógico;

Escrever de forma seqüencial
Uma operação somente ocorre após a anterior terminar e sucessivamente;

Usar condições
Para permitir a tomada de decisões e seus caminhos;
Usar repetições
Facilitando e agilizando os processos, tornado-os flexíveis;

Modular sempre
Dividindo um problema em pedaços e resolvendo cada um deles, obtendo no final uma solução total.

Algoritmos em "Portugol"

Durante nosso curso iremos aprender a desenvolver nossos algoritmos em uma pseudolinguagem conhecida como "Portugol" ou Português Estruturado. O "Portugol" é derivado da aglutinação de Português + Algol. Algol é o nome de uma linguagem de programação estruturada usada no final da década de 50. Abaixo alguns exemplos da pseudolinguagem:

```
Variável (não usar espaços e não iniciar com números):
  Nome: Texto;
  Nota_Final: Real;
  Falta1, Falta2: Inteiro;

Condição:
  SE
    algo acontecer
  ENTÃO
    faça isso
  SENÃO
    faça aquilo;

Repetição:
  ENQUANTO
    algo acontecer
  FAÇA
    isso;

  REPITA
    isso
  ATÉ
    algo acontecer;

  DE 1 ATÉ 10 FAÇA
    isso;
```

Algoritmo não computacional

Devemos aprender a pensar logicamente e para isso inicialmente, usaremos algoritmos não computacionais, resolvendo problemas simples de nosso dia a dia. Vamos criar um algoritmo não computacional com objetivo de usar um telefone público.

```
INÍCIO
  TIRAR O TELEFONE DO GANCHO;
  OUVIR O SINAL DE LINHA;
  COLOCAR O CARTÃO;
  TECLAR O NÚMERO DESEJADO;
  SE CHAMAR ENTÃO
    CONVERSAR;
    DESLIGAR;
    RETIRAR O CARTÃO;
  SENÃO
    REPETIR (DO INÍCIO);
FIM;
```

Neste algoritmo não nos preocupamos em achar o telefone, saber se o cartão tinha unidades, entre outras coisas, pois devemos solucionar o problema de forma simples, aumentando sua complexidade a cada nova etapa satisfeita.

Após a resolução de algoritmos não computacionais (mais livres de regras), partimos para algoritmos computacionais, que nada mais é que o primeiro passo para desenvolvimento de programas.

Algoritmo computacional

O desenvolvedor deve montar os algoritmos da forma mais clara possível, podendo ser entendido por qualquer um, sem deixar dúvidas pelo caminho e usando uma linguagem padronizada, o "Portugol". Com um algoritmo computacional bem elaborado, basta levá-lo para a ferramenta de desenvolvimento, com poucos ajustes.

Exercícios propostos. Temos 2 implementações, sendo uma usando o Portugol (sempre acima) e outra usando a própria linguagem do Delphi (sempre abaixo). Atenção, não existe somente uma solução, ao contrário, podem existir diversas:

1. Ler 2 números e mostrar o maior deles.

```
Var
  A,B: Inteiro;
Inicio
  Ler(A,B);
  Se A>B então Mostrar(A) senão Mostrar(B);
Fim;
```

```
//Para começarmos, coloque um botão no formulário e dê duplo clique neste,
//criando o procedimento abaixo, após, insira o código.
//Esta duas barras (//), indicam comentários de uma linha, use sempre
{Sinal de chaves também podem ser usadas para comentários,
mas usando várias linhas. Ao abrir uma chave, feche a mesma.}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A,B: Integer;
begin
  //Aqui esta uma das partes mais complicadas, comentada abaixo
  A:=StrToInt(InputBox('Solicitação','Digite o 1° número',''));
  B:=StrToInt(InputBox('Solicitação','Digite o 2° número',''));
  if A>B then ShowMessage(IntToStr(A)) else ShowMessage(IntToStr(B));
end;

//Tecle F9 (Run) para compilar e executar o programa.

{Este 1° exemplo requer algumas explicações adicionais, não quanto ao algoritmo, mas
sim quanto a nomenclatura dos termos usados.
A 1ª explicação refere-se ao comando InputBox. Este comando apresenta uma janela onde
digitamos os números desejados (um de cada vez), pois trabalhamos em uma plataforma
(Microsoft Windows) totalmente visual.
A 2° explicação refere-se aos comandos StrToInt e logo após, IntToStr. Estes comandos
convertem um texto em um número inteiro (string to integer) e vice-versa, um número
inteiro para texto (integer to string).
Devemos usar estes procedimentos, pois sempre que capturamos algo no teclado
(InputBox) ou mostramos na tela (ShowMessage), teremos um texto (string) e para fazer
os cálculos (if A>B then), usamos números.
Isso pode ser um pouco complicado, mas como uso constante, torna-se usual.}
```

2. Ler 3 números e mostrar o maior deles.

```
Var
  A,B,C: Inteiro;
Inicio
  Ler(A,B,C);
  Se (A>B) e (A>C) então Mostrar(A);
  Se (B>A) e (B>C) então Mostrar(B);
  Se (C>A) e (C>B) então Mostrar(C);
Fim;

procedure TForm1.Button2Click(Sender: TObject); //Ao clicar no 2° botão
var
  A,B,C: Integer;
begin
  A:=StrToInt(InputBox('Solicitação','Digite o 1° número',''); //Finalize
  B:=StrToInt(InputBox('Solicitação','Digite o 2° número',''); //sempre com ;
  C:=StrToInt(InputBox('Solicitação','Digite o 3° número','');
  if (A>B) and (A>C) then ShowMessage(IntToStr(A)); //Ao usar condições
  if (B>A) and (B>C) then ShowMessage(IntToStr(B)); //múltiplas, separá-las
  if (C>A) and (C>B) then ShowMessage(IntToStr(C)); //com parênteses.
end;

Var
  A,B,C: Inteiro;
Inicio
  Ler(A,B,C);
  Se (A>B) e (A>C) então Mostrar(A)
  senão
    Se B>C então Mostrar(B)
    senão
      Mostrar(C);
Fim;

procedure TForm1.Button2Click(Sender: TObject); //Ao clicar no 2° botão
var
  A,B,C: Integer;
begin
  A:=StrToInt(InputBox('Solicitação','Digite o 1° número','');
  B:=StrToInt(InputBox('Solicitação','Digite o 2° número','');
  C:=StrToInt(InputBox('Solicitação','Digite o 3° número','');
```

```
  if (A>B) and (A>C) then ShowMessage(IntToStr(A))
  else
    if B>C then ShowMessage(IntToStr(B))
    else ShowMessage(IntToStr(C));
    //Perceba que usamos ; somente na ultima condição,
    //pois somente ali ela se encerra
end;
```

3. Ler um número, se este for maior que 100, some 30 a esse. Mostre o resultado final.

```
Var
  A: Inteiro;
Inicio
  Ler(A);
  Se A>100 então Mostrar(A+30) senão Mostrar(A);
Fim;

procedure TForm1.Button3Click(Sender: TObject);
var
  A: Integer;
begin
  A:=StrToInt(InputBox('Solicitação','Digite um número',''));
  if A>100 then ShowMessage(IntToStr(A+30)) else ShowMessage(IntToStr(A));
end;

Var
  Meu_Numero: Inteiro;
Inicio
  Ler(Meu_Numero);
  Se Meu_Numero>100 então Meu_Numero:=Meu_Numero+30;
  Mostrar(Meu_Numero);
Fim;

procedure TForm1.Button3Click(Sender: TObject);
var
  Meu_Numero: Integer; //Não use espaços em variáveis
begin
  Meu_Numero:=StrToInt(InputBox('Solicitação','Digite um número',''));
  //Meu_Numero recebe ele mesmo, mais 30, se for maior que 100
  if Meu_Numero>100 then Meu_Numero:=Meu_Numero+30;
  ShowMessage(IntToStr(A));
end;
```

4. Ler um número, mostrar este somente se for menor que 20 ou, maior ou igual a 180.

```
Var
  Numero_Digitado: Inteiro;
Inicio
  Ler(Numero_Digitado);
  Se (Numero_Digitado<20) ou (Numero_Digitado >=180) então Mostrar(A);
Fim;

procedure TForm1.Button4Click(Sender: TObject);
var
  Numero_Digitado: Integer;
begin
  Numero_Digitado:=StrToInt(InputBox('Solicitação','Digite um número',''));
  if (Numero_Digitado<20) or (Numero_Digitado>=180) then //Podemos quebrar em
    ShowMessage(IntToStr(Numero_Digitado)); //várias linhas, se desejado
end;

//Reparou que usamos IntToStr e StrToInt em todos os exercícios?
//Use sempre 2 espaços (não use TAB) para organizar seus códigos
```

Os exercícios abaixo não possuem o código fonte, isto é, o desenvolvimento do mesmo no Delphi. Todos eles se encontram no Anexo I.

5. Ler 4 notas, calcular a média. Sendo a média maior ou igual a 6, mostrar aprovado, se não mostrar reprovado.

```
Var
  A,B,C,D: Inteiro;
  Media: Real;
Inicio
  Ler(A,B,C,D);
  Media:=(A+B+C+D)/4; //Devemos respeitar as leis da matemática
  Se Media>=6 então Mostrar('Aprovado') senão Mostrar('Reprovado');
Fim;
```

6. Ler 4 números, calcular e exibir os resultados das operações:
6.1. Subtração do 1° número pelo 2° número;
6.2. Multiplicação do 2° número pelo 3° número;
6.3. Divisão do 3° número pelo 4° número;
6.4. Soma dos 3 resultados anteriores.

```
Var
  A,B,C,D,X,Y: Inteiro;
  Z: Real;
Inicio
  Ler(A,B,C,D);
  X:=A-B;
  Mostrar(X);
  Y:=B*C;
  Mostrar(Y);
  Z:=C/D;
  Mostrar(Z);
  Mostrar(X+Y+Z);
Fim;
```

7. Mostrar os números de 1 até 10 (usar para/for, enquanto/while e repetir/repeat).

```
Var
  X: Inteiro;
Inicio
 Para X:=1 a 10 faça Mostrar(X);
Fim;
```

```
Var
  X: Inteiro;
Inicio
  X:=0;
  Enquanto X<10 faça
  Inicio
    X:=X+1;
    Mostrar(X);
  Fim;
Fim;
```

```
Var
  X: Inteiro;
Inicio
  X:=0;
  Repetir
    X:=X+1;
    Mostrar(X);
  Até x=10 faça
Fim;
```

8. Mostrar os 100 primeiros números pares.

```
Var
  X: Inteiro;
Inicio
  Para X:=0 a 99 faça Mostrar(X*2);
Fim;
```

9. Ler 33 números e mostrar quantos são: positivos, negativos e zeros.

```
Var
  X,A,P,N,Z: Inteiro;
Inicio
  P:=0;
  N:=0;
  Z:=0;
  Para X:=1 a 33 faça
  Início
    Ler(A);
    Se A>0 então P:=P+1;
    Se A<0 então N:=N+1;
    Se A=0 então Z:=Z+1;
  Fim;
  Mostrar(P,N,Z);
Fim;
```

10. Ler a idade de 50 pessoas, calcular e exibir:
10.1. A soma de todas das idades;
10.2. A média das idades de todas a pessoas;

```
Var
  X,SI: Inteiro;
  MI: Real;
Inicio
  SI:=0;
  Para X:=1 a 50 faça
  Início
    Ler(I);
    SI:=SI+I;
  Fim;
  Mostrar(SI);
  Mostrar(SI/50);
Fim;
```

10.3. Alterar para mostrar a soma das idades a cada 10 pessoas.

```
Var
  X,Y,SI: Inteiro;
  MI: Real;
Inicio
  Y:=0;
  SI:=0;
  Para X:=1 a 50 faça
  Início
    Ler(I);
    SI:=SI+I;
    Y:=Y+1;
    Se Y=10 então
    Início
      Mostrar(SI);
      Y:=0;
    Fim;
  Fim;
  Mostrar(SI);
  Mostrar(SI/50);
Fim;
```

11. Ler a altura, o peso e a idade de 40 pessoas e mostrar:
11.1. Quantas tem idade maior ou igual a 18 anos;
11.2. Quantas tem idade menor que 18 anos;
11.3. A média do peso;
11.4. A média da altura;
11.5. A média da idade.

12. Ler a idade de 45 jogadores de basquete de uma escola e classificá-los de acordo com a tabela abaixo:
12.A. Maior que 11 e menor ou igual a 13 anos, classificar como pré-mirim;
12.B. Maior que 13 e menor ou igual a 15 anos, classificar como mirim;
12.C. Maior que 15 e menor ou igual a 17 anos, classificar como infantil;
12.D. Maior que 17 e menor ou igual a 18 anos, classificar como infanto-juvenil;

Calcular e mostrar:
12.1. Quantos alunos estão em cada categoria;
12.2. Quantos alunos não se encaixam em nenhuma categoria;
12.3. Quais categorias poderão competir, sendo que para isso elas tenham no mínimo 10 jogadores.

13. Calcular o salário de um operário, sabendo que ele recebe R$ 15,00 por hora de trabalho. Se a quantidade de horas trabalhadas exceder 50 horas, o valor será de R$ 20,00 por hora excedente. Considere ainda se o salário ultrapassar R$ 800,00, deve ser descontado 8% de imposto deste. Mostre:
13.1. O salário bruto (sem desconto do imposto);
13.2. O imposto;
13.3. O salário líquido com desconto do imposto).

14. Ler 3 números, verificar se formam um triângulo isósceles (2 lados iguais), um triângulo equilátero (3 lados iguais), um triângulo escaleno (3 lados diferentes).

15. Ler 3 números distintos e colocá-los em ordem crescente ou decrescente.

16. Ler um nome e mostrar este invertido.

Programa 1. **Agenda Telefônica**

Vamos criar um programa de agenda de telefones, contendo também o registro das ligações... Usaremos o Delphi 6 e inicialmente tabelas Paradox, migrando posteriormente.

Primeiramente vamos analisar as tabelas que o programa necessita, tendo uma tabela para cadastro dos nomes e dados pessoais e outra para registro das ligações. Os campos são:

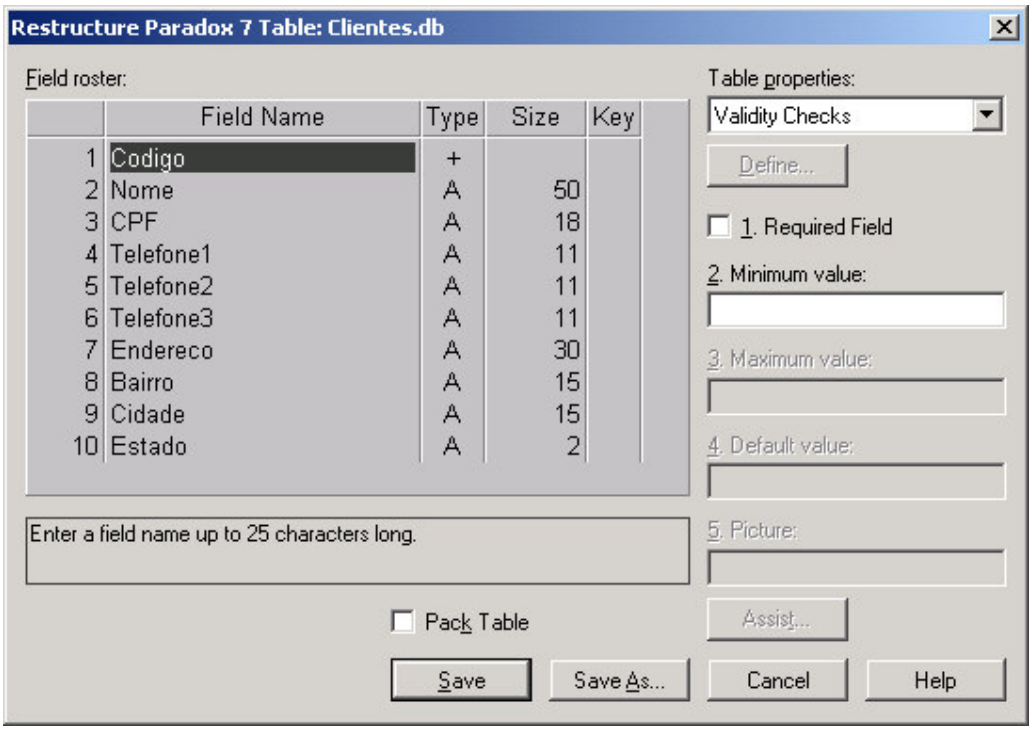
Tabela de Clientes

Nome do Campo	Tipo	Tamanho	
Código	*+*		– Campo auto-incremento (tipo inteiro que incrementa automaticamente e nunca se repete).Este campo é necessário para uso de relacionamentos, explicado mais a frete
Nome	*A*	*50*	– Campo alfanumérico (String) com até 50 caracteres, o máximo é de 250 caracteres
CPF	*A*	*18*	– Não será usado inteiro, pois possui caracteres de . e – (ex.: 145.254.785.87)
Telefone1	*A*	*11*	– Mesmo motivo acima (ex.:22-38513496)
Telefone2	*A*	*11*	
Telefone3	*A*	*11*	
Endereço	*A*	*30*	
Bairro	*A*	*15*	
Cidade	*A*	*15*	
Estado	*A*	*2*	

Tabela de Ligações

Nome do Campo	Tipo	Tamanho	
Chave	*+*		– Não necessário, mas podemos precisar ordenar esta tabela ou outros motivos
Codigo_Cliente	*I*		– Conterá o código do cliente que gerou a ligação, explicado mais à frente
Data_Ligacao	*D*		

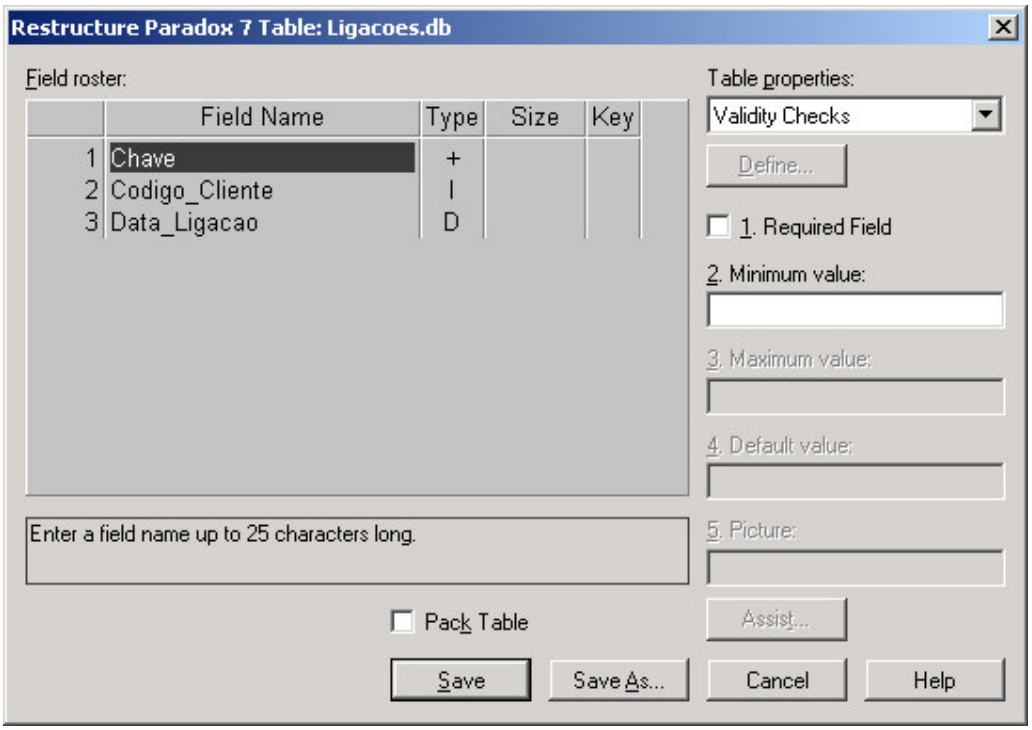
Vamos criar as tabelas, mais tarde criaremos novos campos nestas tabelas.
Abra o Database Desktop e navegue até *File – New – Table*, selecione *Paradox 7* e clique em **OK**. Digite os dados da tabela de clientes, sendo *FieldName* o nome do campo, *Type* o tipo e *Size* o tamanho.



Dica: Os campos podem conter acentuação gráfica ou espaços, mas não é recomendado o uso dos mesmos, pois o Delphi eliminará estes caracteres, podendo gerar confusão (ex.: Digitação SEÇÃO ou Delphi converterá em SEO).

Após digitar os campos da tabela de clientes, clique em Save As..., escolha o diretório e chame o arquivo de *CLIENTES*. Recomendamos criar um diretório exclusivo para suas tabelas. Assim como um exclusivo para o código fonte.
Isto é válido, pois cada tabela pode ter até 6 arquivos e cada formulário do programa, mais uns 4 arquivos, então serão certas vezes centenas de arquivos criados, complicando a manutenção e entendimento do sistema.

Repita os passos para a tabela de ligações, dando o nome de *LIGACOES*. Evite acentuação gráfica e espaços. Os espaços podem dar problemas no uso da linguagem SQL.



Dica: Sempre que desejar criar um espaço, use _ (ex.:Nome_cliente). Atenção, _ difere de -, use somente _.

Pronto, as tabelas foram criadas, depois criaremos índices e será explicado detalhadamente sobre estes.

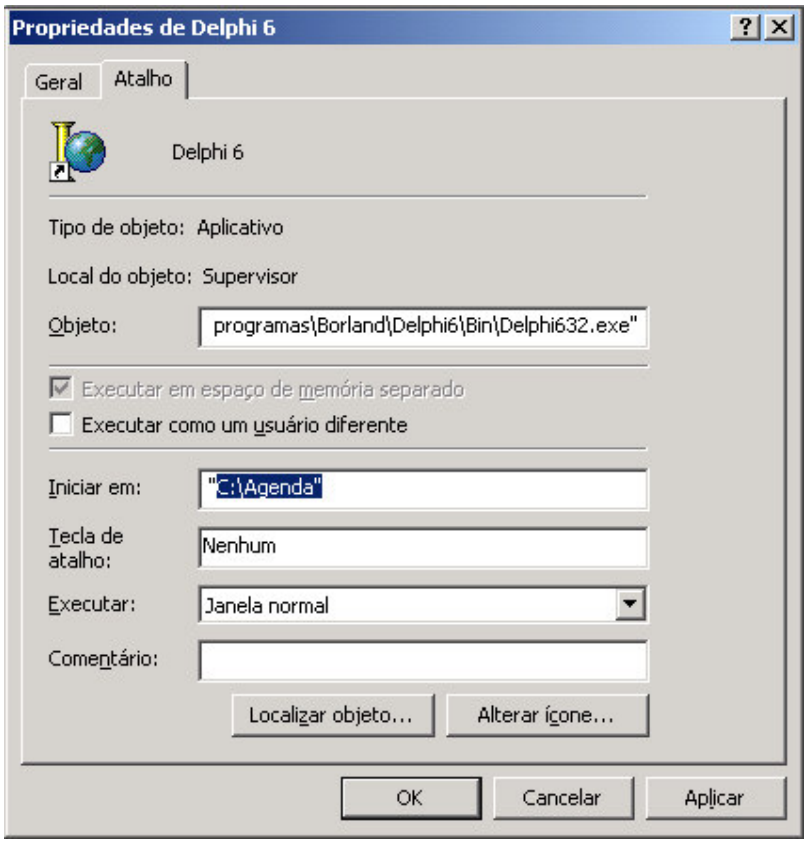
Agora partiremos para o Delphi, em cada detalhe.

Abra o Delphi pelo seu atalho e vamos configurar o formulário. Se uma aplicação não existir, navegue até *File - New - Application*.
Tecle **F12** para que o formulário apareça na tela (outra vez, aparece a unit).
Tecle agora **F11** para visualizar o *Object Inspector* e mudar as propriedades do formulário.
Mude as propriedades para:

Heigth	*480*	Altura
Width	*640*	Largura
Position	*poScreenCenter*	Ficará sempre no centro da tela
Name	*Fagenda*	Sem espaços ou acentos
Caption	*Agenda de Telefones*	Pode conter espaços e acentos
BorderStyle	*bsSingle*	Não permite mudar o tamanho do formulário quando o mesmo estiver sendo executado, apenas se usar a opção de maximização

Vamos salvar a aplicação. Sempre é recomendado salvar logo no início, para já definir o diretório a nome do projeto. Veja dica acima sobre como criar os diretórios acima.
Navegue até *File - Save All* (sempre use *Save All* por segurança).
Será solicitado salvar duas vezes, pois temos que salvar a unit/form e também o arquivo de projeto.
Se a janela for *Save Unit1 As*, digite o nome para sua unit/form, que chamaremos de *AgendaU* (U de unit).
Se a janela for *Save Project1 As*, digite o nome de seu projeto (será o nome do executável criado), que será *Agenda*.
Não é necessário colocar a extensão do arquivo.

Dica: O Delphi sempre indica o diretório C:\Arquivos de programas\Borland\Delphi6\Projects como padrão. Para mudar este, navegue até o atalho do Delphi, clique com o botão direito neste e escolha *Propriedades*. Em *Iniciar em...* coloque o diretório que desejar e o seu Delphi será indicará este como padrão.



Dica: Caso tenha salvado seus arquivos em um diretório errado, navegue até *File – Save As...* para salvar novamente a unit e *File – Save Project As...* para salvar novamente o arquivo de projeto.

Vamos continuar arrumando o projeto.
Navegue até *Project – Options... – Application* e em *Title* escreva Agenda de Telefones (nome que aparecerá na barra de tarefas do Windows), clique em **Load Icon**... e escolha o ícone de seu projeto.
Clique na guia *Directories/Conditionals* e em *Output Directory* selecione o diretório onde deseja salvar seu executável (não é necessário preencher este campo). Navegue até *Version Info* para colocar informações no seu executável quando clicar com o botão direito no mesmo e depois *Propriedades*.
Sempre salve tudo (*Save All*).

Coloque 3 componentes Panel em seu formulário. e modifique as propriedades:
Panel1 com... *Align* *AlTop* – sempre no topo
Panel3 com... *Align* *AlBotton* – sempre na base
Panel2 com... *Align* *AlClient* – em todo o formulário restante
Remova a propriedade *Caption* dos mesmos (selecione qualquer um deles, aperte **SHIFT** e selecione os outros dois, na propriedade *Caption* no *Object Inspector* e apague seu conteúdo). Com os Panels ainda selecionados, mude a propriedade *BevelInner* para *bvLowered*.
Selecione o Panel1 (ao topo) e a propriedades *Caption* para *Agenda de Telefones* e *Heigth* para *40*. Clique no sinal de mais em *Font* e modifique:
Color clBlue Efetue duplo clique na cor para escolher
Name *Times News Roman*
Size *18*

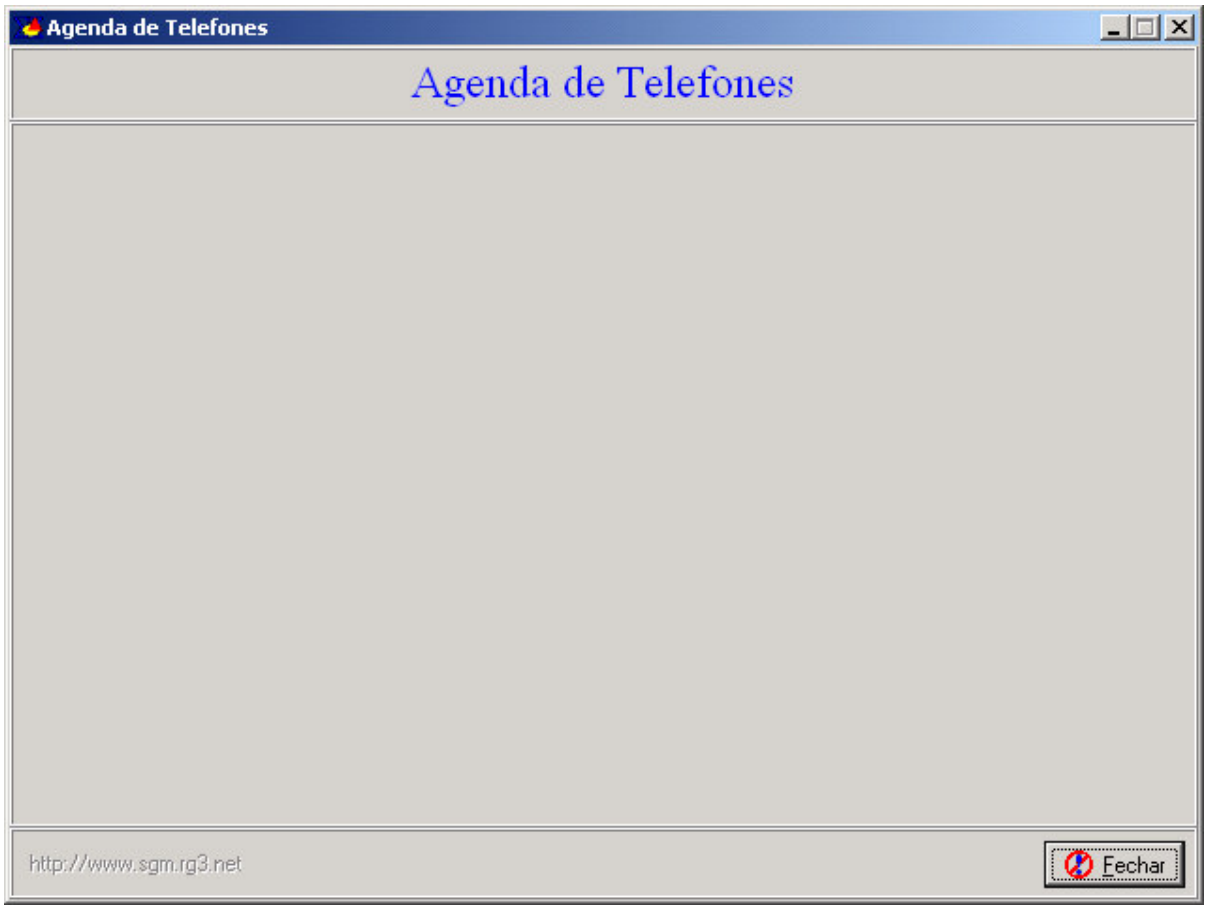
No **Panel3** (a base) mude a propriedade *Heigth* para *38*. Arrume seus componentes pelo *Object Inspector* para obter maior precisão.
No **Panel3** ainda, coloque um **BitBtn** (na paleta Additional) e modifique suas propriedades:
Caption *&Fechar*
Glyph Clique nos 3 pontinhos [...] e escolha um botão em C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons

Top *7* Posição vertical
Left *550* Posição horizontal
Name *BtFechar*
Efetue um duplo clique no botão e escreva: Application.Terminate;

Dica: Use sempre dois espaços para endentar o código. Use CRTRL+espaço para autopreencher (ex.: App e **CTRL+Espaço**, procure *Application* e **ENTER**)
Salve tudo sempre.

Coloque um **Label** no **Panel3** (a base) e mude o *Caption* para *http://www.sgm.rg3.net* :0) e *Enabled* para *False* (duplo clique no *True* funciona também). Ajuste a posição do mesmo com *Left* em *10* e *Top* em *12*, por exemplo.

Tecle **F9** e veja a cara de seu programa.



Dica: Quando teclamos **F9** o Delphi compila e executa o programa, mas o mesmo continua na tela. Uma boa opção é navegar até *Tools - Environment Options...* e a opção marcar *Minimize on run*. Teste novamente com **F9** e veja a diferença.

Após o **F9**, observe agora que seu formulário esta no centro da tela e vemos ainda que a opção de maximização está disponível. Vamos testar a mesma, clique nela. Ops, observe que o botão fechar ficou fora de posição, como arrumar isso?
Feche seu programa, mas atenção, o programa é executado com **F9**, não feche seu projeto no Delphi. Verifique se o mesmo está rodando na barra de tarefas do Windows (abaixo) ou na barra de título do Delphi (acima) como o nome do programa e o termo *[Running]*.

Voltando ao botão. Selecione o mesmo (**F12** para ver o formulário) e vamos até a propriedade *Anchors*, clicando no sinal de mais. Temos 4 opções:
AkLeft *True*
AkTop *True*
AkRight *False*

AkBotton False

Cada opção desta ancora o botão em um dos lados (ancorar é travar). Como o padrão é ancorar a esquerda (*akLeft*) e ao topo (*aktop*) de seu componente pai (**Panel3**), quando maximizamos o formulário, o botão vai para o topo do **Panel3** (não muda nada) e a esquerda, como visto. O botão não vai ao topo do **Panel3**, pois mudamos a propriedade *Align* do mesmo para *AlBotton*, que muda suas ancoras para *AkLeft*, *AkRigth* e *AkBotton* e não é puxado pra cima, mantendo a mesma largura original.
Veja as ancoras do **Panel1** que colocamos *Align* com *alTop* e do **Panel2** com *Align* em *AlClient*.
As propriedades *Anchors* e *Align* estão ligadas uma as outras e certas vezes são complicadas de arrumar, sendo uma boa opção desligar a maximização do formulário da seguinte forma: clique em qualquer **Panel** e tecle *ESC* para focar o **Form**, na propriedade *BorderIcon*, desligue *biMaximize* com duplo clique em *True*. Veja como fica com **F9** e escolha a opção que desejar.

Salve tudo. A partir de agora salvar tudo é com você, ok?

Vamos agora colocar nossas tabelas no programa.
Uma boa opção é criar um formulário para conter estas tabelas e outros componentes não visuais. O nome deste formulário é **Data Module**, mas sendo que prefiro usar um formulário comum. O **Data Module** é complicado de arrumar às vezes, sem opção de **Labels**, etc.
Navegue até *File - New - Form,* salve o mesmo como o nome *DMAgendaU* e no mesmo diretório dos arquivos fontes anteriores. Atenção neste detalhe.
Mude a propriedade *Name* para *DMAgenda* e outras que desejar. Este formulário nunca será visto quando o programa estiver em execução, ele será um formulário não visual, apenas usado para armazenar componentes.

Agora passaremos para os componentes de banco de dados, mas antes uma explicação sobre relacionamento. No nosso projeto, temos duas tabelas relacionadas, o que é isto?
A tabela de ligações está relacionada à tabela de clientes, pois toda ligação é feita obrigatoriamente para um cliente. Registramos na tabela de ligações, a data da ligação e também o código do cliente que ligamos, mas por que somente o código?
Isso é feito por dois motivos, o 1° para economizar espaço em disco (e velocidade de acesso), pois se armazenarmos todos os dados do cliente na tabela de ligações, teríamos centenas ou milhares de informações repetidas sem necessidade. Exemplo:

Tabela de Ligações

Chave	Codigo_Cliente	Data_Ligacao	Nome	CPF	Telefone1	Telefone2	Telefone3	Endereco	Bairro	Cidade	Estado
1	7	28/09/2003	José	456.987.635-87	22-38513251			Rua Paulo II	Centro	Niterói	RJ
2	15	28/09/2003	Ana	688.988.630-82	21-22351541	22-98635741					
3	7	28/09/2003	José	456.987.635-87	22-38513251			Rua Paulo II	Centro	Niterói	RJ
4	3	28/09/2003	Pedro	015.965.325-95	22-38535654						

Vimos que temos várias vezes a mesmas informações e sem necessidade. Sendo que as informações primordiais são: *Codigo_Cliente* e *Data_Ligacao*. Vamos eliminar as outras informações então.

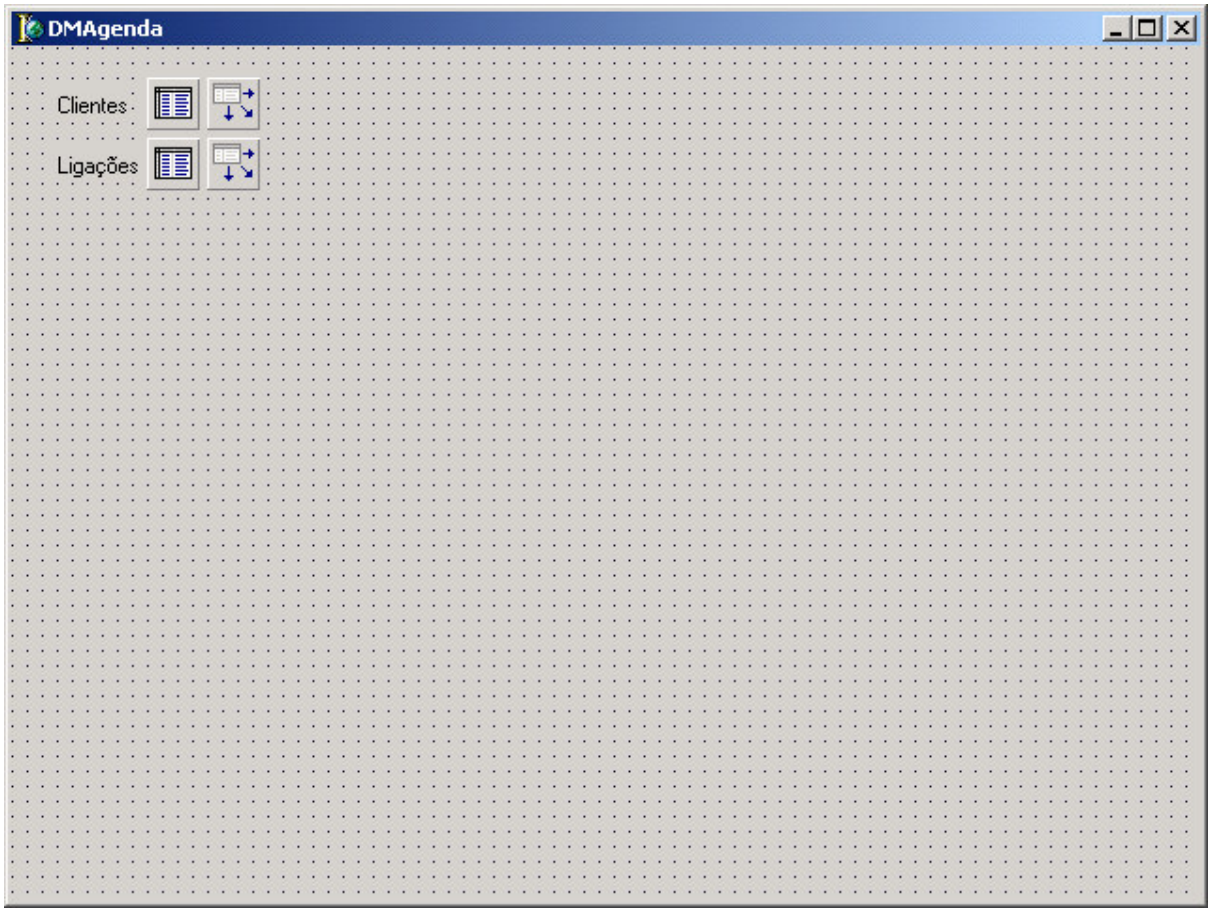
Chave	Codigo_Cliente	Data_Ligacao
1	7	28/09/2003
2	15	28/09/2003
3	7	28/09/2003
4	3	28/09/2003

O campo chave não é prioritário, mas manteremos o mesmo e poderíamos achar o campo telefone discado importante, mas analisaremos isso posteriormente.
Ocorreu uma redução no número de informações desta tabela, deixando-a mais leve e rápida. Mas ai você pode perguntar, para que liguei? Qual o endereço deste cliente?
Neste caso entra o relacionamento, que funciona da seguinte forma: quando selecionamos uma ligação, esta contém além de outras coisas o código do cliente,

então pegamos este código e verificamos o mesmo na tabela de clientes. Por exemplo: a 2° ligação for efetuada para o cliente de código 15, basta percorrermos a tabela de cliente e acharmos o mesmo.
Outra vantagem deste relacionamento, é que mesmo que os dados dos clientes mudem, eles serão sempre atualizados, pois a informação é uma só e não várias e várias.
Este exemplo possuía apenas quatro registros de ligações, imagine uma empresa com um sistema há anos e com milhões de registro. O relacionamento minimiza custo e otimiza o sistema, deixando-o mais ágil.
O campo Codigo na tabela de clientes é chamado de Chave Primária (PK – Primary Key) e o campo Codigo_Cliente na tabela de ligações, de Chave Estrangeira (FK – Foren Key). Os dois campos estão relacionados.
Observe ainda que a FK na tabela de ligações se repete (sem problemas, pois um cliente gerará várias ligações), mas a PK na tabela de clientes NUNCA poderá se repetir, pois teremos sempre dúvida para quem ligamos. Uma PK sempre, SEMPRE, é única.
Este relacionamento é chamado 1:n (um para n), pois um cliente tem várias ligações. Temos ainda 1:1 (um para um) e N:M (muito para muitos) em outros casos, mas não nesse aqui.

Vamos colocar os bancos de dados no Delphi agora?

Localize a paleta BDE e coloque dois componentes **Table** no formulário **DMAgenda** em qualquer posição, agora mude as propriedades *Name* para *TbClientes* e *TbLigacoes*. Estes componentes fazem a ligação externa do programa com as tabelas.
Para definir qual tabela pertence a cada componente, primeiro precisamos indicar o diretório onde estão as tabelas na propriedade *DatabaseName* de cada componente. Para facilitar, localize a mesma no Windows Explorer, copie o endereço e cole nesta propriedade. No meu caso: *C:\Agenda\Tabelas*.
Feito isso, na propriedade *TableName*, clique na setinha que aparece e informe a tabela correta para cada componente. Mude a propriedade *Active* para *True* das duas tabelas e você já estará conectado ao banco de dados.
Até agora só fizemos a ligação do nosso programa com as tabelas, precisamos agora importar os campos das tabelas também (opcional, depende de como o programador trabalha). Dê um duplo clique na componente *Table* (qualquer um dos dois) e aperte **CTRL+A** (ou botão direto e *Add fields...*). Todos já estão selecionados, de **OK**. Faça o mesmo com a outra tabela.
Mas um outro passo ainda é necessário. O delphi não permite acesso aos campos da tabela diretamente, tendo que usar um componente exclusivo para isso chamado **DataSource**.
Na paleta *DataAccess* localize este componente e coloque dois destes no formulário, próximos as tabelas para facilitar a manutenção. Mude *Name* para *DSClientes* e *DataSet* para *TbClientes* e na outra tabela, mude *Name* para *DSLigacoes* e *DataSet* para *TbLigacoes*.



Vamos colocar os componentes visuais agora, componentes para digitações dos dados, botões, etc.

Tecle **CTRL+F12** (ou 1° ícone da 2ª linha) e escolha *AgendaU* (ou ainda **SHIFT+F12** e *FAgenda*). Tecle **F12** para ver o formulário. Na paleta *Win32* coloque um componente **PageControl** no **Panel2** (central) mudando a propriedade *Align* para *AlClient*.
Clique com o botão direito no *PageControl* e selecione *New Page* quatro vezes.

Dica: Agora dentro do componente **PageControl**, temos quatro componentes **TabSheet**, cada componente **TabSheet** tem suas propriedades e diferem umas das outras, assim como diferem das propriedades do **PageControl**.

Clique no **TabSheet1** (na palavra *TabSheet1* mesmo) e depois clique no centro do mesmo. Mude o *Caption* para *Consulta* e o *Name* para *TSConsulta*.
Clique no **TabSheet2** e mude o *Caption* para *Cadastro* e o *Name* para *TSCadastro*.
Clique no **TabSheet3** e mude o *Caption* para *Ligações* e o *Name* para *TSLigacoes*.
Clique no **TabSheet4** e mude o *Caption* para *Relatórios* e o *Name* para *TSRelatorios*.

Dica: Para mudar esta ordem, mude a propriedade *PageIndex* dos **TabSheet**, experimente.

Coloque um **Panel** dentro de um dos **TabSheet** quaisquer. Mude a propriedade *Align* para *AlClient*, *BevelInner* para *bvLowered* e apague o *Caption* do mesmo. Clique neste **Panel**, aperte **CTRL+C**, nos outros **TabSheet** (não esqueça de clicar nos centros destes) aperte **CTRL+V**.

Vamos fazer em ordem...

Coloque mais um **Panel** dentro do **TSConsulta** (dentro do **Panel** que já esta lá também). Apague o *Caption* e mude *BevelInner* para *bvLowered*, *Align* para *AlTop* e *Heigth* para *30*. Coloque dentro deste novo **Panel** um **BitBtn** com *Heigth* em *25*, *Width* em *25* e *Caption* com *A*. Faça o mesmo com mais vinte e dois **BitBtn**, colocando sempre um à direita do outro, até a letra *Z*.

Exemplo:
A *Top* *2*, *Left* *2*
B *Top* *2*, *Left* *27*
C *Top* *2*, *Left* *52*
...
Z *Top* *2*, *Left* *552*

Coloque um último **BitBtn** e escolha um *Glyph* para o mesmo, deixe *Caption* em branco:
Top *2*, *Left* *577*, *Heigth* *24* e *Width* *36*

Verifique a propriedade *TabOrder* de cada um, sendo:

A *TabOrder* *0*
B *TabOrder* *1*
C *TabOrder* *2*
...
Z *TabOrder* *22*
TabOrder *23*

O *TabOrder* indica a ordem que a tabulação seguirá ao teclar **TAB**.

Na paleta *DataControl*, selecione um **DBGrid** e coloque no **Panel**, mudando a propriedade *Align* para *AlClient* (preencherá o **Panel** por completo). Neste apareceram os clientes filtrados por letra inicial de nome. O último botão mostrará todos os clientes sem filtro.

Vamos deixar a consulta agora um pouco de lado e vamos ao cadastro. Clique no **TSCadastro** e coloque no mesmo dez componentes **DBEdit** localizados na paleta *DataControls*. Estes componentes serão ligados a tabela de clientes para inserção e alteração dos dados.
Como vimos anteriormente, o componente **DBEdit** não pode se ligar diretamente a tabela de clientes e então usamos o componente **DataSource** para este fim. Selecione o **DBEdit1** e mude a propriedade *DataSource* do mesmo para *DMControle.TbClientes*.

Observe que você não consegue localizar a tabela de clientes na propriedade *DataSource* do **DBEdit**, por que? Isto ocorre porque temos o **DBEdit** em um formulário e as tabelas em outro e eles não se enxergam automaticamente, nós mesmo que teremos que autorizar este vínculo, de forma muito simples.
Tecle **ALT+F11** (ou *File – Use Unit...*) e selecione a unit desejada, no caso a unit *DMAgendaU* (única disponível). Pronto, agora o formulário/unit *Fagenda/AgendaU* enxerga o nosso **Data Module**. Mas atenção, o inverso não ocorre e não é necessário ainda.

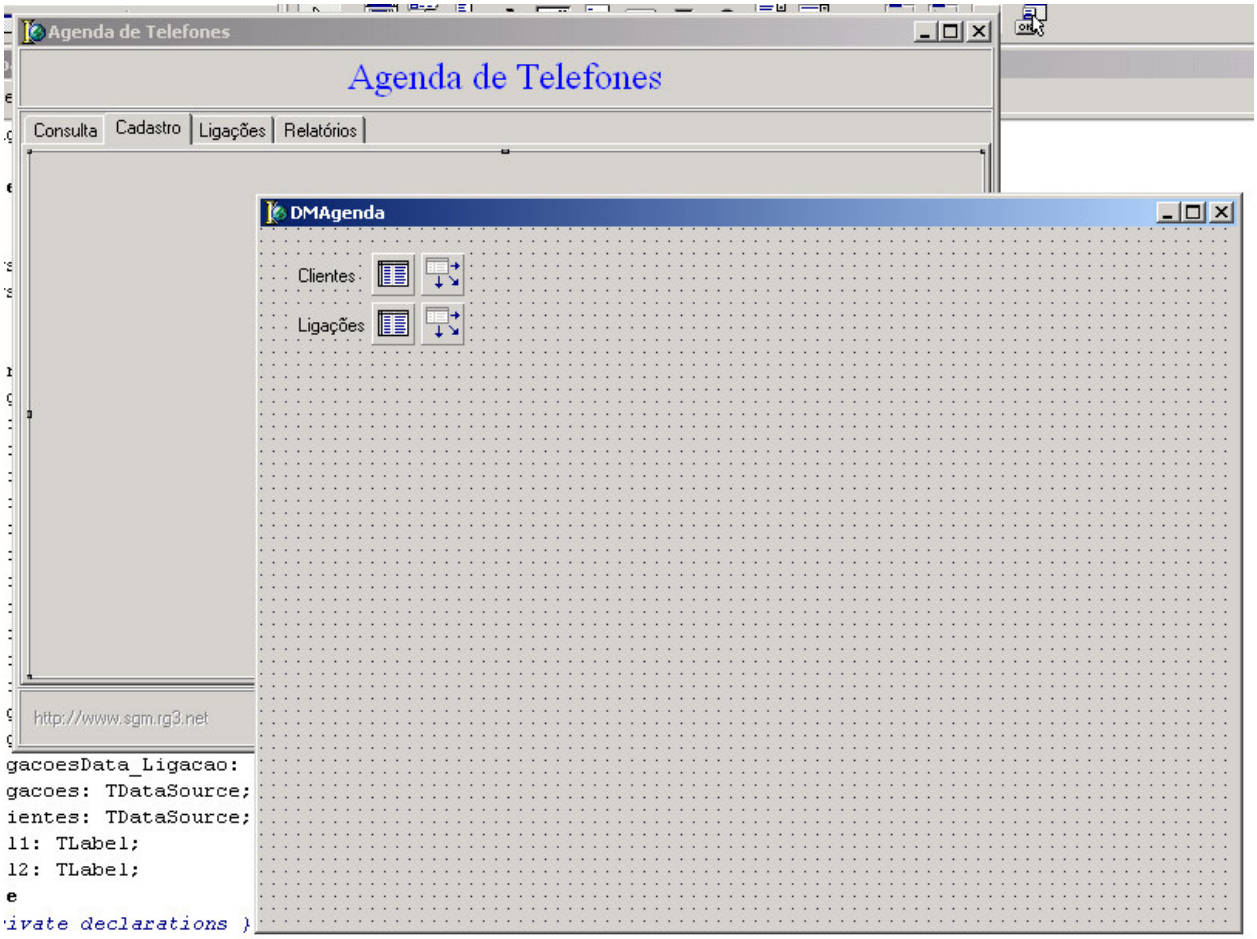
Dica: Observe na unit *AgendaU* que logo abaixo de *implementation* foi declarada a cláusula uses *DMAgendaU*, que é o resultado de teclamos **ALT+F11**. Esta cláusula pode ser criada ou apagada manualmente também.

Novamente selecione o **DBEdit1** e mude a propriedade *DataSource* para *DMControle.TbClientes*, que agora estará disponível. Com isso, o **DBEdit1** está vinculado à tabela de clientes, mas falta ainda informar a qual campo, então selecione na propriedade *DataField* o campo desejado, sendo o campo *Codigo*.
Pronto, finalizamos os vínculos, faça isso para os outros nove **DBEdits** no formulário, sendo 1° o *DataSource* (tabela) e depois o *DataField* (campos). Posicione os componentes como quiser, aumentando ou diminuindo a largura dos mesmos, coloque também vários **Labels** para indicar o conteúdo do campo.

Dica: Para vincular todos os *DataSource* juntos, selecione um qualquer, tecle **SHIFT** e selecione os outros, depois mude a propriedade *DataSource*. Para *DataField* isto não funciona.

Obs.: É recomendado trocar os nomes dos componentes a medida que aumentamos nosso programa, pois mais tarde, este nomes facilitarão o entendimento do mesmo. Por padrão, pode-se renomear os componentes usando duas iniciais para o tipo de componente e o nome de referência do mesmo.
Exemplo: mude o **DBEdit1** para **EdCodigo**, o **DBEdit2** para **EdNome**. O botão deste **TabSheet** pode se chamar **BtConfirmar** ainda.

Dica: Uma opção mais simples seria arrastar os *fields* da tabela para o **Panel5**. Para testar apague todos os **DBEdits** deste painel e siga os seguintes passos, atentamente:
Tecle **CTRL+F12** e selecione **DMAgendaU**, posicione os formulários de maneira que os dois permaneçam visíveis, conforme figura abaixo.



Dê um duplo clique na tabela de clientes e selecione todos os campos da mesma, para isso clique no 1°, aperte **SHIFT** e clique no último. Agora arraste estes campos para o **Panel5** no formulário principal **Fagenda**. Se tudo deu certo, para cada campo na tabela aparecerá um **DBEdit** e um **Label** já preenchido, arrume como desejar.

Dica: Uma propriedade dos componentes que temos que ter atenção, é a de nome *TabOrder*. Está propriedade define qual será a ordem de tabulação destes e sempre deve ser de maneira a facilitar o trabalho do usuário final.
Para acertar, clique no **EdCodigo** e coloque *TabOrder* como 0 (zero), coloque o **EdNome** com *TabOrder* como 1 (um) e assim por diante.
Outra opção é clicar com o botão direto em qualquer componente e acessar *Tab Order...* Fique sempre de olho nesta ordem, pois isso pode atrapalhar o funcionamento de um programa.

Agora vamos começar a codificar nosso programa.
Dê um duplo clique no botão do **Panel5**, o **BtConfirmar**. Estaremos acionando o evento *OnClick* do botão. Digite DMAgenda (ou DM e **CTRL+Espaço**) e .TbClientes.Insert;

Veja:

```
procedure TFAgenda.BtConfirmarClick(Sender: TObject);
begin
  DMAgenda.TbClientes.Post;
end;
```

Dica: Deixe sempre o Delphi ajudá-lo com a codificação, usando **CTRL+Espaço** para aparecer o *Code Completion*. Use de maiúsculas ou minúsculas não interfere no programa neste caso, mas tenha sempre uma regra para facilitar.

Temos um problema de lógica agora, como podemos efetuar um *Post* se não estamos em modo de edição ou inserção, um erro acontecerá. Vamos testar?

Tecle **F9** e após a execução, clique no **BtConfirmar**. Um erro surgirá, dê **OK** e tecle **F9** novamente. A mensagem "TbClientes: Dataset not in edit or insert mode.", dê **OK** e feche o programa.

Dica: Duas telas de erro apareceram, sendo que a 1ª só ocorre quando estamos com o projeto aberto e a 2ª aparece para o usuário. Para não ver a 1ª quando estiver projetando, navegue até *Tools – Debugger Options... – Language Exceptions* e marque **Stop on Delphi Exceptions**, mas cuidado com esta configuração, ok?

Voltando ao erro no programa, para efetuarmos um *Post*, devemos antes efetuar um *Insert* ou *Edit*. No nosso caso agora, queremos um *Insert*, mas onde colocar isso?
Um bom lugar talves seja ao entrar no **EdCodigo** se este estiver em branco, mas lembre-se que colocamos este campo como auto-incremento, então não podemos mudá-lo.
Selecione o **EdCodigo** e coloque a propriedade *ReadOnly* como *True*.
Vamos colocar então este *Insert* ao entrar no **EdNome**. Selecione o mesmo e no *Object Inspector*, selecione a guia *Events*, procure *OnEnter* e dê um duplo clique na caixa de edição a doreita deste. Veja abaixo o código:

```
procedure TFAgenda.EdNomeEnter(Sender: TObject);
begin
  if EdNome.Text='' then DMAgenda.TbClientes.Insert;
end;
```

Mas podemos melhorar este código, pois erros ainda acontecem se a tabela já estiver em modo de inserção ou edição e ainda mais, se o usuário digitar espaços em branco no **EdNome**. Coloque então:

```
procedure TFAgenda.EdNomeEnter(Sender: TObject);
begin
  if  (DMAgenda.TbClientes.State<>dsInsert)  or  (DMAgenda.TbClientes.State<>dsEdit)
then //Verifica se já esta em inserção ou edição
  begin
  if Trim(EdNome.Text)='' then DMAgenda.TbClientes.Insert; //Função Trim – remove
espaços antes de depois do nome
  end;
end;
```

ou

```
procedure TFAgenda.EdNomeEnter(Sender: TObject);
begin
  if  (DMAgenda.TbClientes.State<>dsInsert)  or  (DMAgenda.TbClientes.State<>dsEdit)
then if Trim(EdNome.Text)='' then DMAgenda.TbClientes.Insert;
end;
```

Observe que as condições do 1º **if** estão entre parenteses, esta é uma obrigação no Delphi. Tecle **CTRL+F9** para compilar o programa, para verificar se existe algum erro de sintaxe até agora.
Se o erro "Undeclared identifier: 'dsInsert'" aparecer, é devido ao Delphi não ter declarado a cláusula DB no seu **uses** dá **unit**. Para isso vá no topo da **unit** e declare **DB** em qualque local do **uses**:

```
unit AgendaU;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, Buttons, ComCtrls, Grids, DBGrids, Mask, DB,
  DBCtrls;
```

Compile (**CTRL+F9**) novamente para ver mais algum erro.

Vamos colocar um bloqueio no **BtConfirmar** também:

```
procedure TFAgenda.BtConfirmarClick(Sender: TObject);
begin
  if Trim(EdNome.Text)<>'' then
  begin
    DMAgenda.TbClientes.Post;   //Confirma
    DMAgenda.TbClientes.Insert; //e já deixa pronto para novo cliente
    EdNome.SetFocus;            //Manda o foco para EdNome
  end;
end;
```

Tecle **F9** (compilar e executar) e teste, mas devemos fazer mais acertos. Dê um duplo clique em qualquer guia do **PageControl1** (em Consulta, Cadastro, Ligações ou Relatórios) e o seguinte código surge:

```
procedure TFAgenda.PageControl1Change(Sender: TObject);
begin
end;
```

Este evento sempre é executado ao navegarmos no **PageControl1** (clicado nos **TabSheet**). Acrescente neste evento:

```
  if TSCadastro.CanFocus then //Se o TabSheet selecionado for de cadastro
  begin
    if (DMAgenda.TbClientes.State<>dsInsert) or ((DMAgenda.TbClientes.State<>dsEdit)
then DMAgenda.TbClientes.Insert;
  end;
```

Hummm, você viu um erro? Abrimos 2 parenteses depois do **or**, para cada parenteses aberto deve existir um fechado, caso contrário este erro ocorre, hehe, tá acordado?

Mais...

```
procedure TFAgenda.PageControl1Change(Sender: TObject);
begin
  if TSCadastro.CanFocus then //Se o TabSheet selecionado for de cadastro
  begin
    if (DMAgenda.TbClientes.State<>dsInsert) or ((DMAgenda.TbClientes.State<>dsEdit)
then DMAgenda.TbClientes.Insert;
  end
  else
  begin
    if  (DMAgenda.TbClientes.State=dsInsert)  or  (DMAgenda.TbClientes.State=dsEdit)
then DMAgenda.TbClientes.Cancel; //Não confirmou, então cancela
  end;
```

Coloque o **TSConsulta** como padrão na abertura do programa (clique nele) e teste tudo. Aqui não achei mais nenhum erro, só meu **DBGrid1** (agora com o nome de **GrClientes**) que estava com a propriedade *DataSource* vazia, então coloquei *DMAgenda.DSClientes*, às vezes acontece.
Vamos colocar umas máscaras agora por baile, ops... Máscaras são usadas em campos como CPF e telefone, que possuem caractares especiais de formatação. Faça o seguinte: **CTRL+F12** e **DMAgendaU**, duplo clique em **TbClientes** e selecione *CPF*, mude a propriedade *EditMask* para *999.999.999-99;1;***espaço** (tecle um espaço) e nos telefones coloque *99-99999999;1;***espaço**. Você pode usar números 0 ou 9 nestes *EditMask*, onde 0 torna a digitação obrigatória (podendo gerar até erros, por exemplo, 99-00000000) e 9 a digitação opcional.

✓ **Sobre EditMask**

!	Sei não, hehe, você sabe? Fala ai;
0	Número obrigatório;
9	Número opcional;
L	Letra obrigatória;
l	Letra opcional;
A	Alfanumérico obrigatório;
a	Alfanumérico opcional;
C	Qualquer obrigatório;
c	Qualquer opcional;
#	Número ou símbolo opcional;

**** Para usar qualquer caractere, \(999\) → (000);
> Fomata como maiúscula;
< Formata como minúscula;

No 2° termo, quando passa para Label, por exemplo:
0 Não salva os caracteres extras, como / ou (e), etc. Usar quando precisar somente dos números, para operações, por exemplo;
1 Salva todos os caracteres;

Vamos agora deixar o cadastro um pouco de lado e voltar à consulta... Antes de tudo, você percebeu que pode inserir, editar ou excluir (**CTRL+Delete**) através do **GrClientes**? Para desabilitar isso, selecione o mesmo e no *Object Inspector* (nome complicado de escrever) mude a propriedade *ReadOnly* para *True* e em *Options*, *dgEditing* para *False*.
Dê um duplo clique nn **GrClientes** e clique no 3° ícone, apague tudo, deixando somente *0 - Codigo* e *1 - Nome*. Selecione *0 - Codigo*, mude a propriedade *Title - Caption* para *Código*.
Vamos fazer os filtros, mas deixe explicar algo. Teríamos que fazer um filtro para cada letra e um código para cada botão, farei apenas um código para todos os botões, ok? O filtro funcionará da seguinte forma, ao clicar na letra desejada, apareceram todos os clientes com aquela inicial e usaremos a seguinte máscara: *letra + **, onde o asterisco é um caracter coringa, tipo você usa no Windows mesmo para localizar arquivos (exemplo: *.doc, aparece todos os arquivos do MSWord, que estou apanhando dele agora, acho que farei um em Delphi, hehe).
Oura coisa, por padrão o Dephi faz diferença de letras maiúsculas e minusculas em comparação de *String* e este filtro é uma *String*. Para desabilitar esta propriedade nos filtro, selecione a tabela **TbClientes**, mude a propriedade *FilterOptions - foCaseInsensitive* para *True*, agora no filtro (somente no filtro desta tabela) ele não fará mais distinção em maiúsculas e minúsculas.

Duplo clique no botão da letra A que ficará com o seguinte código:

```
procedure TFAgenda.BitBtn2Click(Sender: TObject);
var
  Mascara: String;
begin
  Mascara := QuotedStr(TBitBtn(Sender).Caption+'*');  //Exemplo: "A*"
  DMAgenda.TbClientes.Filter   := 'Nome = '+Mascara;  //Nome  =  "A*"
  DMAgenda.TbClientes.Filtered := True;               //Liga o filtro
end;
```

Antes de explicar, selecione o botão da referente à letra B, aperte **SHIFT** e selecione todos até a letra Z (se quiser, selecione o B, aperte **CTRL+SHIFT** e arraste o mouse sobre os outros, difícil?). Com todos (B à Z) selecionados, vá na guia *Events* no *Object Inspector* e em *OnClick*, selecione *BitBtn2Click* (selecione, não de duplo clique).
O que você fez? Você ligou todos on eventos de clicar nos botões ao cique do botão A, isto é, todos os botões então no mesmo *OnClick*. De duplo cique nos botões para ver isso. Já que todos os botões estão com o mesmo procedimento, economizamos 22 procedimentos e ficamos com um só. Neste que sobrou poderíamos colocar várias codições para saber com qual letra filtramos, mas vamos usar o poder do Delphi de novo.
A segredo está aqui: `Mascara := QuotedStr(TBitBtn(Sender).Caption+'*');`

Por partes...
Mascara - variável *String* comum;
QuotedStr - para usarmos um filtro com um campo Alfa (texto), temos que colocar o nome desejado entre aspas e QuotedStr faz isso. Seria o mesmo que usar aspas duplo no início e fim (Mascara:='"'+TBitBtn(Sender).Caption+'*"')), mas com o problema com palavras como Caixa d'água, que já possui um apóstrofe e ocorreria um erro. A função QuotedStr resolve este tipo de problema.
TBitBtn(Sender).Caption - aqui esta o mais importante. Não usamos o nome do botão (**BitBit2**) e sim sua classe, o *Sender* diz de onde originou este código e o *caption* é a letra que desejamos. Se clicarmos na letra S, por exemplo, o *Sender* sabe que o botão clicado foi o **BitBtn17** (supondo que estajam em seqüencia), que possui caption igual a S.
* - máscara, igual do exemplo *.doc

Tendo conhecimento desta funcionalidade, podemos simplificar e muito nos códigos fontes. Basta fazer os testes agora.
Por último, duplo cique no **BitBtn25** (último botão, depois do Z) e coloque:
DMAgenda.TbClientes.Filtered:=False; *//Desliga o filtro*

Apostila ainda em desenvolvimento...

1. Funcionamento local em Paradox (com uso de SQL)
2. Relatórios com QuickReport
3. Instalação completa
4. Instalando o RxLib
5. Funcionamento em rede
6. Migração para Firebid
7. Migração para Linux
8. Espero conseguir tempo pra tudo, rs.
9. Quem paga uma cerveja?

Entre em contato e de sugestões:

Sávio Cler
ICQ: 142428832
saviocler@ig.com.br
Tel.: (22)9812-0656
Santo Antônio de Pádua - RJ

Acesse:

www.sgm.rg3.net
www.deephi.rg3.net - mais de 1.500 dicas

Anexo I

Apresentamos aqui a resolução completa de todos os exercícios propostos anteriormente, sendo que todos foram desenvolvidos usando apenas um formulário e um botão para cada problema. Mas antes disso, vamos brincar um pouco...

Faremos aqui, um programa para nos sentir mais a vontade com a interface do Delphi. Este terá 3 funções: deixar o mouse maluco, fazer um botão fugir do mouse e apresentar imagens na tela.

Inicialmente, com o Delphi aberto e com uma nova aplicação (File – New – Application), coloque no formulário 3 botões, um componente Image (paleta Additional), um Timer (paleta System) e um OpenPictureDialog (paleta Dialogs).

1° passo, deixar o mouse maluco...
Dê um duplo clique no Timer e digite o seguinte código:

```
  SetCursorPos(Random(800),Random(600));
```

Pronto, só teclar **F9**... Gostou? Conseguiu fechar o programa? Hehe. Vamos piorar um pouquinho. Selecione o componente Timer e tecle **F11** (Object Inspector), mude a propriedade Enabled para False e a propriedade Interval para 100. Dê um duplo clique no botão 1 e digite:

```
Timer1.Enabled:=True;
```

Tecle **F9** e depois clique no botão 1, ah, boa sorte.

2° passo (se você sobreviveu ao 1° passo), fazer um botão fujão.
Selecione o botão 2, tecle **F11** (Object Inspector), na guia Events procure OnMouseMove e de um duplo clique na caixa branca do lado direito. Digite o código:

```
  Button2.Left := Random(300);
  Button2.Top  := Random(300);
```

Atenção, Button2 é o nome do botão, podendo variar. Verifique a propriedade Name no Object Inspector.

3° passo, um leitor de imagens.
Dê um duplo clique no botão 3 e digite o código:

```
  if OpenPictureDialog1.Execute then
            Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
```

Selecione o componente Image, tecle **F11** e mude a propriedade AutoSize para True. Basta testar teclando **F9**.

Resolução completa do exercícios apresentados.

```
unit e_logicaU;

{* * * Elaborado e desenvolvido por Sávio Cler em 2004 * * *}
{* * *       www.sgm.rg3.net - www.deephi.rg3.net      * * *}
{* * *               saviocler@ig.com.br               * * *}


interface

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

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Button1: TButton;
    SpeedButton1: TSpeedButton;
    BitBtn3: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Close; //Fecha aplicação

  //Neste botão, passando o mouse sobre o mesmo (quando em tempo de execução - depois
do F9),
  //aparece uma mensagem, que pode ser configurada na propriedade (no Object
Inspector)
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  A,B: Integer; //Variáveis não podem ter espaços, acentuação gráfica
  X,Y: String;  //ou começar por números
begin
  //A propriedade Caption (no Object Inspector) do botão foi alterado para '1º
exercício'
  //Estas linhas com // no início, são apenas comentários, sendo ignoradas pelo
Delphi

  //Sempre teremos que converter o que foi digitado na tela, para números
  //Então neste exercício, usaremos mais 2 variáveis adicionais, X e Y, do tipo
String (texto)
```

```
  //para demonstrar com mais detalhes a conversão
  X := InputBox('1° exercício','Digite o 1° número:','');
  Y := InputBox('1° exercício','Digite o 2° número:','');

  //Convertemos X e Y, para números e armazenamos em A e B
  //Atenção, pois se digitar letras os espaços e não números, um erro surgirá
  //Analisaremos estes erros, mais adiante
  A := StrToInt(X); //StrToInt = String to Integer = Texto para Inteiro
  B := StrToInt(Y);

  //Fazemos a comparação e apresentamos o resultado
  //Observe que a conversão foi feita (de número para texto agora), na própria
exibição do valor
  //sem usarmos X e Y
  if A>B then ShowMessage( IntToStr(A) ) else ShowMessage( IntToStr(B) );
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
var
  A,B,C: Integer;
begin
  //Alteramos a propriedade Caption e também adicionamos uma imagem neste botão
(BitBtn)
  //pela propriedade Glyph, no Object Inspector

  //Não usaremos mais as variáveis adicionais, fazendo a conversão sempre direta

  A := StrToInt( InputBox('2° exercício','Digite o 1° número:','') ); //A conversão
foi direta
  B := StrToInt( InputBox('2° exercício','Digite o 2° número:','') ); //Os espaços
dados somente visam facilitar
  C := StrToInt( InputBox('2° exercício','Digite o 3° número:','') ); //o
etendimento, não sendo obrigatórios

  //Fazemos a comparação e apresentamos o resultado
  if (A>B) and (A>C) then ShowMessage( IntToStr(A) ); //IntToStr = Integer to String
= Inteiro para Texto
  if (B>A) and (B>C) then ShowMessage( IntToStr(B) );
  if (C>B) and (C>A) then ShowMessage( IntToStr(C) );
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  Meu_Numero: Integer; //As variáveis podem, e devem, ter nomes mais explicativos
begin
  //Este botão (SpeedButton) possui ainda a propriedade Flat, onde o mesmo fica
  //transparente se igual a True

  Meu_Numero := StrToInt( InputBox('3° exercício','Digite um número:','') );

  if Meu_Numero>100 then Meu_Numero := Meu_Numero + 30; //Ele mesmo, mais 30 (muito
utilizado em programação)

  ShowMessage( IntToStr(Meu_Numero) );
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
var
  A: Integer;
begin
  A := StrToInt( InputBox('4° exercício','Digite um número:','') );

  //Sempre que usamos and ou or, devemos separar as condições com parenteses,
obrigatoriamente
  if (A<20) or (A>=180) then ShowMessage( IntToStr(A) );
end;
```

```
end.
```