PROGRAMAÇÃO EM DELPHI PARA INICIANTES

BY YURIY KALMYKOV

embarcadero

Sobre o autor

Yuriy Kalmykov é um especialista renomado em desenvolvimento de software e autor de muitos manuais e publicações sobre programação, incluindo "Teaching Delphi Programming in Schools". Esse livro é o resultado de 25 anos de orientação de estudantes como membro da faculdade Informatics and Control Processes da National Research Nuclear University MEPhI (Moscow Engineering Physics Institute) e lecionando nas principais escolas preparatórias em Moscou.

Introdução

Esse livro foi feito para quem está iniciando no estudo de programação. Porém, isso não quer dizer que todos aqueles que já utilizam Delphi (ou outra linguagem de programação combinada com uma IDE RAD) não possam aproveitar da experiência de Yuri Kalmykov.

O Delphi surgiu em 1995 para plataforma Windows e, desde então, se consolidou com uma linguagem dinâmica, moderna e atualizada. E quantas coisas mudaram desde 1995? No Brasil, por exemplo, só em 1995 começou a ser distribuído o acesso à internet. O acesso a computadores era bem restrito ainda, com pouquissimas pessoas possuindo computadores pessoais. Celulares eram um luxo e ainda nem existia a rede 3G (que só foi lançada, no Japão, em 2001). O Delphi acompanhou essa evolução, integrando a seus componentes tudo que foi surgindo ao longo dos anos.

Hoje Delphi é utilizado em diversas aplicações de diversos segmentos, tanto no lado cliente como no lado servidor, em aplicações multicamadas e com compatibilidade para diversos bancos de dados. De aplicações médicas, aplicativos para celulares (em Delphi é possível, com um único código, criar aplicativos para Android, IOS e Windows!) passando por estúdios de áudio, modelagem de aplicações geológicas e, até mesmo pesquisa nuclear (afinal, essa é a área do autor do livro!), tudo pode ser desenvolvido em Delphi utilizando sua IDE RAD.

A evolução tecnológica não para. Todos os dias surgem novidades, novas ideias e conceitos. O Delphi irá acompanhar essa evolução, sempre proporcionando uma forma simples e intuitiva de ser utilizada.

Se você ainda não conhece Delphi, seja bem vindo. Temos certeza que a simplicidade, praticidade e atualidade estarão ao seu lado.

Conteúdo

MÓDULO 1

Apresentando o Delphi. Componentes simples

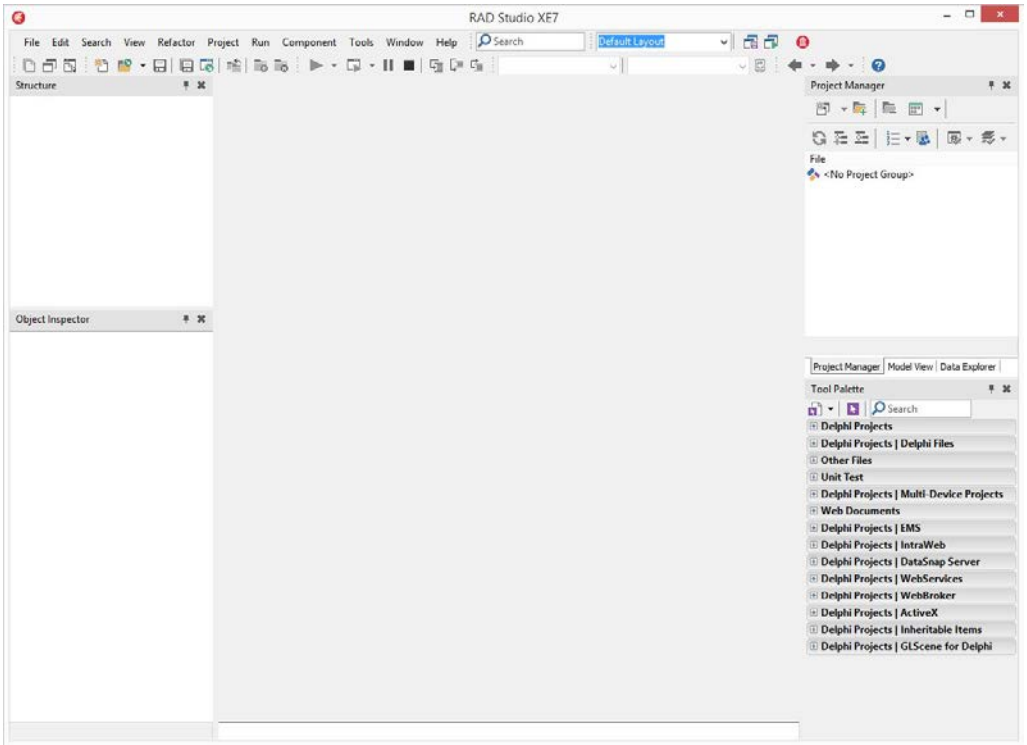
O desenvolvimento de computadores desktop pessoais levaram a introdução de sistemas operacionais multitarefa e multiusuário como o Microsoft Windows. No entanto, como resultado, o processo de criação de software se tornou muito mais complicado. Os ambientes de desenvolvimento integrado (IDE) visuais e os sistemas de desenvolvimento de aplicativo rápido (RAD) foram criados por líderes de mercado para facilitar a interação com sistemas operacionais, reduzir o tempo de codificação e melhorar a qualidade do código.

A programação visual é um processo de criação de aplicativos de software em que um usuário pode projetar, editar, depurar e testar automaticamente o aplicativo usando o IDE visual. Em outras palavras, a programação visual é uma combinação de dois processos interconectados: criação de uma janela de aplicativo visualmente e escrita do código.

O Delphi, um poderoso compilador Pascal com várias melhorias significativas para a criação de aplicativos Windows, foi disponibilizado pela primeira vez em 1996. Ele é um sistema de desenvolvimento de software de alto nível com um conjunto de ferramentas integrado para desenvolver aplicativos completos, é uma ferramenta para o desenvolvimento rápido de aplicativos. Os conceitos de programação baseada em eventos e design visual são os pilares da ideologia do Delphi. O uso desses conceitos melhora significativamente os processos de design de aplicativo e reduz bastante o tempo de desenvolvimento.

O design visual permite que os usuários projetem o aplicativo e vejam os resultados do processo antes mesmo de iniciar o programa de fato. Há outros benefícios também: para atribuir um valor de propriedade a um controle ou elemento do aplicativo, não é necessário escrever várias linhas de código. É necessário apenas alterar esse valor em uma janela Properties. Essa alteração gerará ou atualizará o código automaticamente.

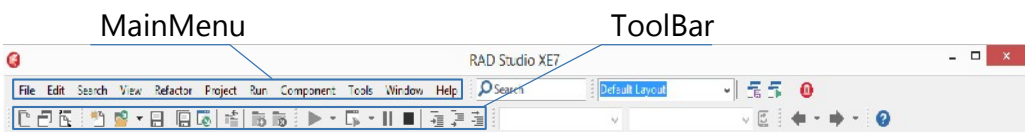
Executando o Delphi



O ambiente de desenvolvimento integrado Delphi é um sistema de várias janelas configurado, definindo os elementos da interface do usuário.

Para qualquer ação do programador na janela, o Delphi faz alterações automaticamente no código do programa subjacente.

A janela principal do Delphi tem um MainMenu e ToolBar.

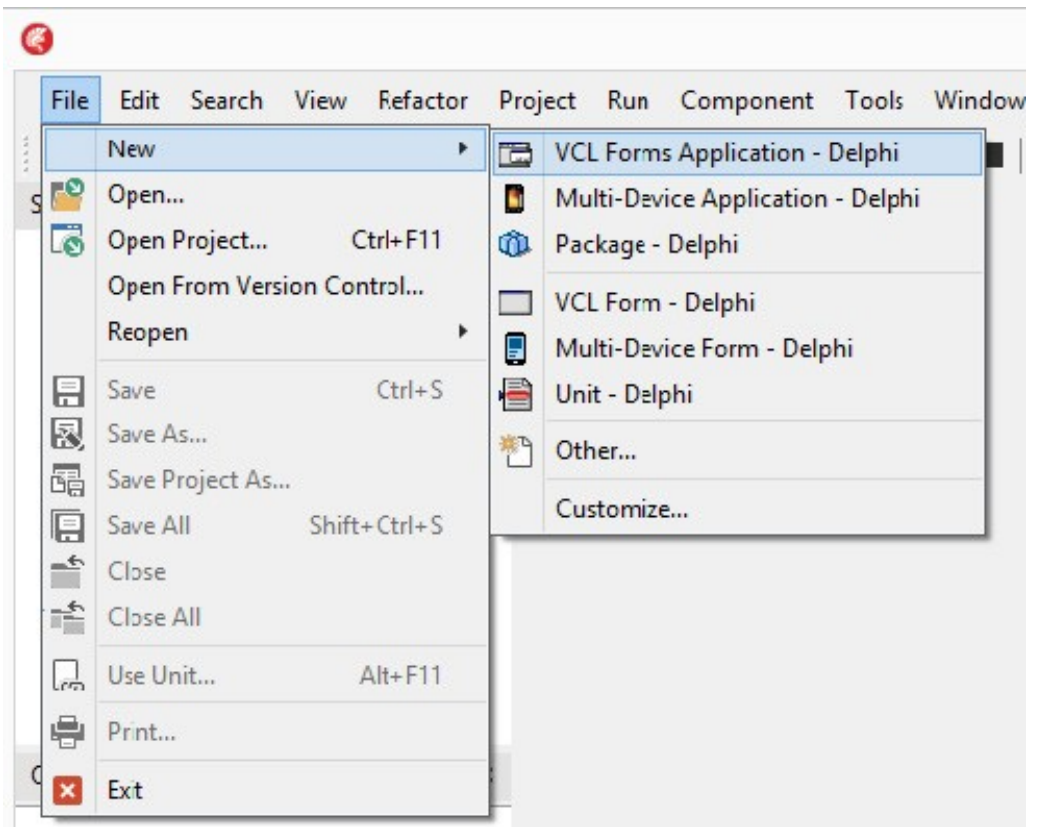


MainMenu contém tudo o que é necessário para gerenciar o projeto.

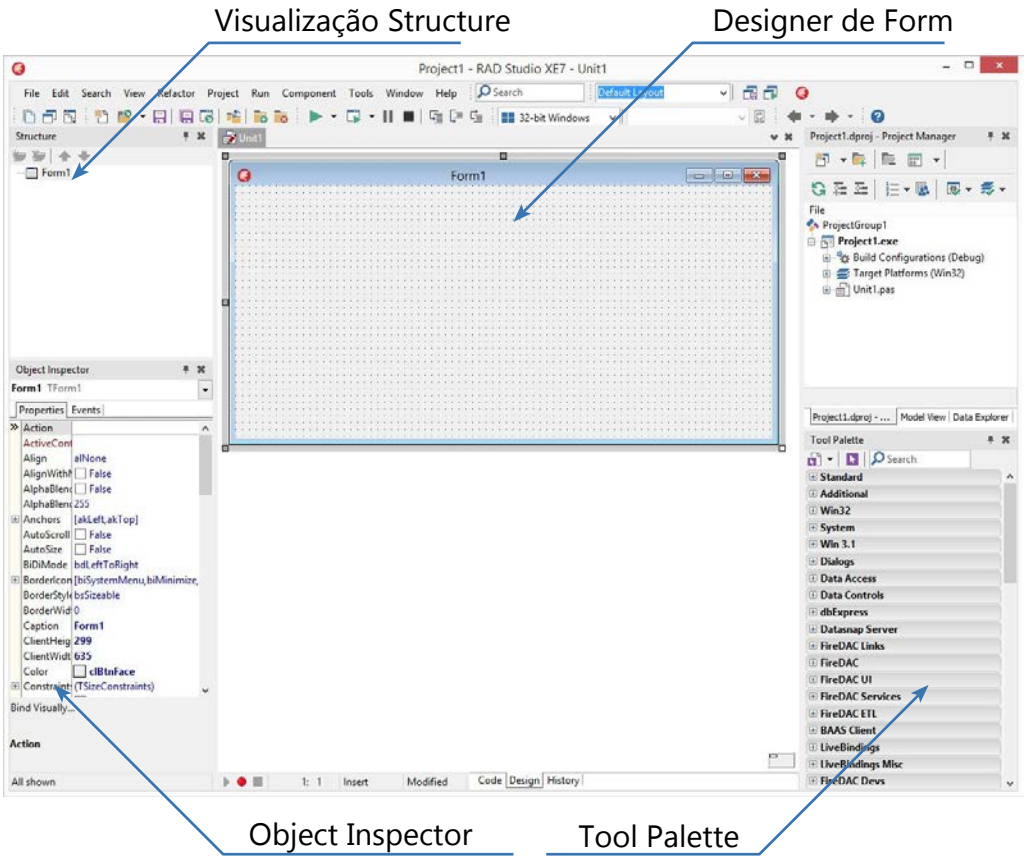
ToolBar contém botões para o acesso rápido às opções mais populares do MainMenu.

Por exemplo, para executar um programa, podemos escolher a opção ***Run*** no MainMenu ***Run*** suspenso, pressionar a tecla F9 ou clicar em um triângulo verde na ToolBar.

Para começar a escrever um novo programa, é necessário selecionar o item ***New*** no menu ***File*** e, na lista aberta, escolher ***VLC Forms Application — Delphi***.

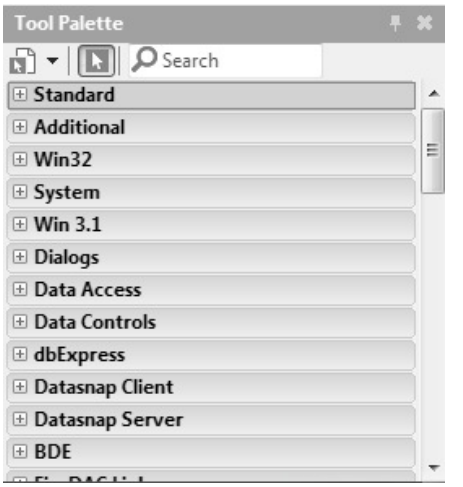


O seguinte layout de janelas do Delphi aparecerá:



Vamos analisar as janelas.

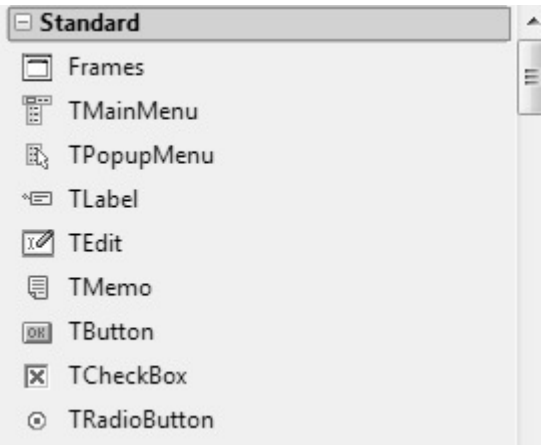
O Delphi é uma linguagem de programação orientada a objeto. Um objeto é uma entidade autocontida que possui propriedades (características ou sinais distintivos) e um conjunto de ações ou comportamentos. O objeto criado pode ser movido de um programa ou outro. O Delphi inclui centenas de objetos prontos para uso (componentes), que são apresentadas na Tool Palette. Esses componentes são agrupados em guias.



As principais guias da Tool Palette são:

1. Standard
2. Additional
3. System
4. Data Access
5. Data Controls
6. Dialogs

Por enquanto, trabalharemos com a guia Standard na Tool Palette. Ela contém elementos de interface padrão que a maioria não pode ficar sem.



A janela Form é um projeto para o futuro aplicativo. A programação visual permite que você crie programas manipulando componentes e colocando-os em um form.

Quando você executa o Delphi pela primeira vez, ele automaticamente sugere um novo projeto com a janela Form chamada Form1. Esse form contém todos os elementos de janela básicos: o título *Form1*, os botões minimizar, fechar e maximizar e um botão de menu do sistema de janela. Essa é a janela principal do nosso programa. Adicionaremos itens gráficos a partir da Tool Palette aqui.

As caixas de texto, botões de comando e outros controles de Form são chamados de componentes (componentes de Form) no Delphi. No programa, o form e os componentes são considerados objetos. Portanto, a janela com as propriedades dos componentes é chamada ***Object Inspector***.

A janela ***Object Inspector*** tem duas guias. Na primeira guia, você pode ver todas as propriedades disponíveis do componente selecionado. A coluna da esquerda tem uma lista e a coluna da direita contém os valores padrão atuais.

A segunda guia, *Events*, tem possíveis manipuladores de evento para o componente selecionado. As colunas da esquerda contêm nomes, a da direita tem as propriedades ou os procedimentos relevantes.

A visualização Structure mostra todos os objetos (componentes) colocados no form.

Parar a execução do aplicativo

- Clique no botão para fechar
- Menu ***Run*** — comando *Program Reset* ***(Ctrl+F2)***

Salvar o projeto

Qualquer programa Delphi inclui uma grande quantidade de diversos arquivos como o arquivo do projeto, uma ou várias unidades etc. O arquivo do projeto é gerado automaticamente pelo ambiente Delphi e não está disponível para edição manual. É por essa razão que ele tem uma única extensão (*.dpr) e ela não é exibida no Code Editor.

O projeto deve ser salvo usando o comando Save All. Cada projeto deve ser salvo em uma pasta separada. A pasta e os arquivos do projeto devem ser nomeados adequadamente. Você não deve aceitar os nomes padrão. Os nomes de arquivo podem conter somente caracteres latinos. Os numerais podem ser usados em nomes de arquivo começando no segundo caractere. Outros símbolos são inválidos para a nomeação do arquivo.

Você deve atribuir nomes únicos aos arquivos ***Project1.dpr*** e ***Unit1.pas*** (eles são diferentes). Aos outros arquivos serão atribuídos nomes padrão.

Ao renomear os arquivos ***Project1.dpr*** e ***Unit1.pas***, siga o padrão de código do Delphi, de acordo com o qual todos os arquivos devem ter um prefixo XXX, por exemplo: ***prjMyTask1.dpr*** e ***untMyTask1.pas*** (XXX são as letras correspondentes ao nome abreviado do que você está salvando: ***Project1 — prj*** e ***Unit1 — unt***). Essa regra também se aplica ao renomear componentes simples.

Isso nos leva a alguns dos componentes simples disponíveis no ambiente Delphi.

Esses componentes são considerados simples não apenas porque é mais fácil de usá-los em comparação com outros componentes. Eles são chamados assim porque, durante a criação da interface gráfica do usuário, eles normalmente são usados em combinações bastante complexas com outros componentes. Vamos analisar os seguintes componentes simples: form, rótulo, caixa de texto e botão.

Form

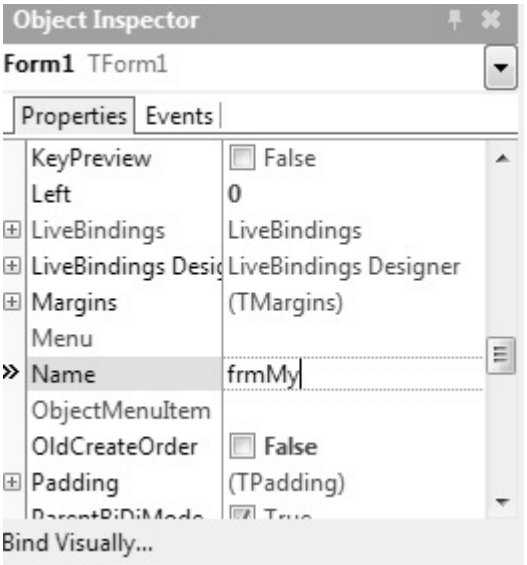
De todos esses componentes, o form é o único objeto que não está na Tool Palette. Um form é gerado automaticamente ao criar um novo aplicativo. Se você precisar usar vários forms em um aplicativo, pode adicionar mais forms ao projeto. Aprenderemos a fazer isso posteriormente. Ao trabalhar no Delphi, imagine que você é um artista e que o form é sua tela. Você "pinta" a interface gráfica do usuário do aplicativo colocando os componentes no form. Em seguida, alterando as propriedades dos componentes, você altera sua aparência.

Depois disso, você trabalhará na interação do componente com o usuário. Interação do usuário significa que os componentes do form reagem às ações do usuário.

As propriedades dos componentes são suas características. O nome, a legenda, o tamanho ou a cor do componente e o tamanho ou a cor do rótulo do componente são exemplos de propriedades.

Vamos revisar algumas propriedades do objeto Form. Muitas propriedades de diferentes componentes são iguais, mas o Form tem algumas propriedades únicas.

A primeira propriedade que examinaremos é a propriedade ***Name.*** Todos os componentes devem ter essa propriedade, pois o nome é necessário para chamar o componente no programa. Para o nome, use o prefixo de trigrama descrevendo o tipo de componente. Para o Form, por exemplo, usaremos o prefixo frm (nome mais curto de Form). Se quisermos chamar o Form de *My*, devemos atribuir o valor *frmMy* à propriedade Name.



Vamos examinar três componentes simples da guia Standard da Tool Palette.

TLabel

O componente ***TLabel*** é usado para emitir o texto que pode ser alterado pelo usuário (certamente, ele também pode ser alterado por um programa). Vejamos como trabalhar com TLabels em um exemplo concreto. Realize as seguintes etapas:

1. Crie um novo projeto.

2 Coloque o TLabel no form. Para fazer isso, você precisa clicar duas vezes no ícone TLabel na Tool Palette. Outra maneira de colocar o TLabel no form é clicar uma vez no ícone TLabel e clicar em qualquer lugar no form. Isso é mais conveniente porque o TLabel é colocado onde você deseja. Para excluir o TLabel do form, você precisa selecioná-lo (clique nele com o botão do mouse, pequenos quadrados pretos mostram que ele está selecionado) e pressionar a tecla <Delete>. Para cancelar a seleção do TLabel, você precisa clicar com o botão do mouse em algum lugar fora do TLabel. Experimente com a colocação e a remoção de TLabels.

3. Mova o TLabel arrastando e soltando. Para fazer isso, mova o ponteiro até o TLabel e pressione e segure o botão do mouse para "pegar" o TLabel. "Arraste" o TLabel até o local desejado movendo o ponteiro para o novo local. "Solte" o objeto soltando o botão.

4. Altere a propriedade Name do TLabel para lblMyLabel (por padrão, ele chamava Label1). Clique na propriedade Name no Object Inspector e digite ***lblMyLabel.*** Certifique-se de alterar a propriedade TLabel, não a propriedade Form (esse é um erro comum de iniciantes). Para este fim, o TLabel precisa ser selecionado. O título da lista na parte superior do Object Inspector dirá ***Label1 TLabel*** (assim que você alterar o nome, isso será alterado para ***lblMyLabel TLabel).*** Após digitar o novo nome, salve-o pressionando a tecla <***Enter***>.

5. Verifique se a legenda do TLabel foi alterada para IblMyLabel. Isso acontece porque, por padrão, a legenda do TLabel é seu nome. Altere a legenda. Para fazer isso, selecione a propriedade Caption no Object Inspector, digite a nova legenda "It's my first label!" e pressione a tecla <***Enter***>. O novo texto aparecerá em um form.

6. Altere a cor do plano de fundo do TLabel. Selecione a propriedade Color, clique na seta, selecione yellow na lista suspensa e clique nele.

7. Altere a fonte e a cor do texto do TLabel. Selecione a propriedade Font e clique nos três pontos (reticências). Na janela Font, altere a fonte para Arial, o estilo para Bold Italic e o tamanho para 20. Selecione red na lista suspensa e clique no botão OK.

8. Adicione mais um TLabel ao form. Tente utilizar outra maneira dessa vez: clique no ícone TLabel na Tool Palette, mova o ponteiro até qualquer lugar no form e clique novamente. Um novo TLabel deve aparecer.

9. Altere a propriedade Name do novo TLabel para ***lblAnother*** e a propriedade ***Caption*** para "Another label".

10. Agora selecione o form. Você pode fazer isso de duas maneiras: clicar em qualquer lugar fora dos rótulos ou selecionar ***Form1*** na visualização Structure. Se o form estiver visível, a primeira maneira é mais conveniente, claro, mas se houver muitos forms no projeto e o form que você precisa estiver coberto por outros, a segunda maneira é melhor.

11. Altere as propriedades do form: configure o valor da propriedade Name como ***frmLabelExample*** e o valor da propriedade Caption para "Example of Label"**.**

12. Você acaba de criar um aplicativo simples, que, para dizer a verdade, ainda não faz nada. Execute-o. Você pode fazer isso usando uma das três maneiras: clicar no ícone Run (o triângulo verde), selecionar o comando Run no menu Run ou pressionar a tecla ***<F9>***.

13. Clique no botão X no canto superior direito do form para sair do aplicativo.

TEdit

O componente TEdit armazena o texto que pode ser colocado no componente durante a criação do aplicativo e durante a execução. O texto visto na caixa de texto vem da propriedade ***Text***. A propriedade ***MaxLength*** define o número máximo de símbolos na caixa de texto. O valor ***MaxLength*** 0 significa que não há limite para o número de caracteres. A fonte do texto é configurada usando a propriedade ***Font***. Se você definir o ***valor da propriedade Readonly como True***, o usuário não poderá alterar o texto de TEdit. O passo-a-passo a seguir ajudará você a obter um melhor controle da funcionalidade TEdit.

1. Crie um novo projeto.

2. Coloque o ***TEdit*** no form. Exatamente como o ***TLabel***, você pode fazer isso de duas maneiras: clicando duas vezes no ícone TEdit na Tool Palette ou clicando no ícone TEdit e, em seguida, em qualquer lugar no form.

3. Altere o tamanho do TEdit. Coloque o ponteiro do mouse em um dos pequenos quadrados pretos, pressione e segure o botão esquerdo do mouse e arraste o pequeno quadrado preto (e a borda do TEdit com ele) na direção necessária. Quando estiver no tamanho necessário, solte o botão. Se não houver nenhum pequeno quadrado preto em torno do TEdit, o TEdit não está selecionado. Nesse caso, selecione TEdit clicando nele primeiro.

4. Mova o TEdit para outro lugar usando o recurso de arrastar e soltar. Coloque o ponteiro do mouse no TEdit, pressione e segure o botão esquerdo do mouse e arraste o TEdit para o novo local. Quando TEdit chegar ao local necessário, solte o botão do mouse.

5. Atribua o valor ***edtMyText*** à propriedade ***Name***. Clique na propriedade Name no Object Inspector e digite ***edtMyText.*** Como para TLabel, certifique-se de alterar a propriedade TEdit, não o form. Na visualização Structure, você verá ***Edit1***: ***TEdit*** (assim que você alterar o nome ***edtMyText: TEdit*** aparecerá).

6. Selecione a propriedade ***Text*** no Object Inspector e insira seu novo valor: "This is a text edit control". Registre o novo nome pressionando a tecla <***Enter***>. Observe que conforme você insere o texto no Object Inspector, o texto TEdit no form também é alterado.

7. Altere a cor do texto de TEdit para azul. Clique no sinal de + ao lado da propriedade Font. Uma lista de propriedades Font adicionais aparecerá. Selecione a propriedade Color e clique na seta: a lista de cores disponíveis aparecerá. Encontre a cor azul e clique nela.

8. Selecione o form. Você pode fazer isso de uma das duas maneiras: clicar em qualquer lugar no form não coberto pela caixa de texto ou selecionar um nome de form na visualização Structure. Altere a propriedade Name para ***frmEditBoxExample*** e a propriedade Caption para "Text Box Example".

9. Pressione a tecla <***F9***> para executar o novo aplicativo. Experimente com a caixa de texto digitando qualquer texto nela.

10. Para sair do aplicativo, clique no botão X no canto superior direito do form.

11. Defina a propriedade ***Readonly*** como ***True***.

12. Pressione a tecla ***F9*** para executar o aplicativo novamente. Tente alterar o texto: como você pode ver, agora é impossível. Você pode imaginar porque precisaria de uma caixa de texto em que não é possível inserir nada. No entanto, posteriormente no manual, você verá que ela é uma ferramenta bastante útil porque você pode alterar um valor da propriedade Readonly programaticamente, permitindo ou não que o usuário insira os dados.

13. Clique no botão X para sair do aplicativo.

TButton

O TButton normalmente é usado para iniciar a execução de alguma parte do código ou todo o programa. Em outras palavras, ao clicar no controle TButton o usuário solicita que o programa execute uma determinada ação. Nesse momento, o botão muda para parecer como se estivesse pressionado.

Você pode atribuir combinações de teclas de atalho aos botões. Durante a execução, pressionar essas teclas de atalho é equivalente a usar o botão esquerdo do mouse no botão.

Execute um conjunto de ações:

1. Crie um novo projeto.

2. No Object Inspector, altere a propriedade Name do form para ***frmButtonExample*** e ***Caption*** para "Button Example".

3. Coloque o botão no form. Clique duas vezes no ícone TButton na Tool Palette ou clique em seu ícone e clique em qualquer lugar no form.

Altere a propriedade TButton para ***btnMyButton***. Clique na propriedade ***Name*** no Object Inspector e digite ***btnMyButton.*** Certifique-se de alterar a propriedade TButton, não o form. Na visualização Structure, você verá ***Button1***: ***TButton*** (assim que você alterar o nome ***btnMyButton: TButton*** aparecerá).

4. Altere a propriedade Caption para ***&Run.*** Observe que a letra após & está sublinhada (nesse caso é R).

5. Altere o tamanho e o local do botão.

6. Pressione a tecla <***F9***> para executar o aplicativo.

7. Clique no botão. O botão aparece pressionado.

8. Ative o botão pressionando a tecla <R>. Como você pode ver, o botão não parece pressionado ao usar a tecla de atalho para ativá-lo. Enquanto isso, como não há nenhum código do programa conectado ao botão, não vemos nenhuma reação ao botão. No entanto, você pode acreditar que ele está ativado.

9. Clique no botão X no canto superior direito do form para sair do aplicativo.

A legenda *btnMyButton* aparece como **R**un, não **Run**. Colocar o símbolo & antes de qualquer caractere da propriedade Caption atribui uma tecla de atalho ao botão. Na legenda do botão, o caractere após & está sublinhado. Isso diz que há uma tecla de atalho conectada ao botão. Os usuários podem ativar o botão pelo teclado pressionando Alt e a tecla com a letra sublinhada.

O que fazer se a própria legenda precisar ter &? Depois de tudo, se você colocá-lo na legenda, ele sublinhará a próxima letra e não será visível. Para resolver esse problema, use a seguinte regra: O símbolo & será visto na legenda se a propriedade Caption tiver dois & juntos, isto é, &&. Por exemplo, para criar a legenda *This &* That, escreva *This &&* That na propriedade ***Caption***. As combinações de teclas de atalho não são atribuídas nessa situação.

Como você pode ver, no Delphi é possível criar um form (e um aplicativo) simples rapidamente. As interfaces gráficas do usuário mais complicadas podem ser criadas da mesma forma.

Agora você está familiarizado com as propriedades dos componentes mais populares do Delphi: forms, rótulos, caixas de edição e botões. Para aprimorar seu novo conhecimento, experimente com esses componentes. Tente alterar suas propriedades.

Manipulando um evento de pressionamento de botão

Os componentes visuais podem gerar e manipular diversos tipos de eventos: clique do mouse, pressionamento de botão, pressionamento do teclado, abertura de janelas etc.

Quando o usuário pressiona qualquer botão do mouse, o Windows envia uma mensagem *"This mouse button was clicked" para o aplicativo.* Esse programa responderá a esse evento se um programador tiver programado uma reação para ele.

A resposta para um evento é o resultado de um evento de sistema ocorrendo.

Uma resposta do programa a um evento é programada como uma lista de ações a serem realizadas. Essa lista é inserida na janela ***Code Editor***. Para um programa reagir a um clique de botão do mouse, é necessário escrever um fragmento de programa chamado *manipulador de eventos*.

Janela Code Editor

Essa janela tem o título Unit1 no início.

Você pode abrir vários arquivos de código-fonte na janela do editor. Todos os arquivos abertos ocupam suas próprias guias e os rótulos de suas guias exibirão os nomes dos seus arquivos de código-fonte. Haverá três unidades se o programa tiver três janelas. Todas as três unidades serão exibidas no Code Editor fonte.

Eventos simples contêm apenas fonte de eventos, apontadas pelo parâmetro Sender no ***procedimento de manipulador de eventos***. Os ***procedimentos de manipulador de eventos*** para manipular eventos complexos precisam de parâmetros adicionais, como as coordenadas do ponteiro do mouse.

A seleção de um elemento de controle gera um evento ***OnClick***, também chamado de ***evento de pressionamento***. Normalmente ele é originado do clique do mouse em um componente. O evento ***OnClick*** é um dos eventos

usados com mais frequência no desenvolvimento de um aplicativo. Para alguns componentes, o evento ***OnClick*** pode ser originado de diferentes ações no elemento de controle.

Como um exemplo, reescreveremos o manipulador do evento de pressionamento do botão de comando ***btnMy***, colocado no form ***frmMy*:**

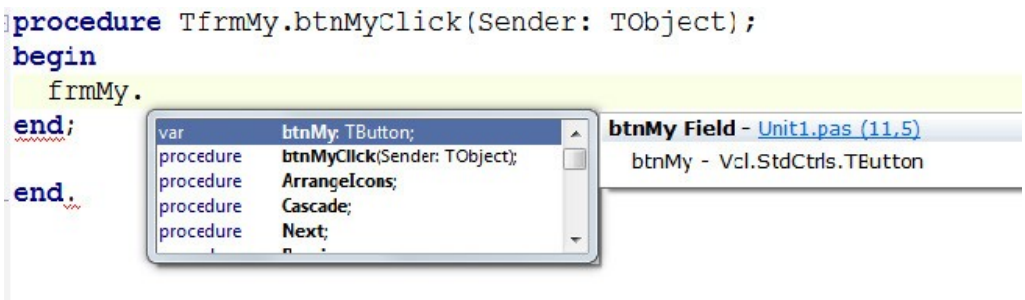
Clique duas vezes nesse botão.

O Delphi gera o código automaticamente para minimizar a entrada do teclado, fornecendo o corpo completo do procedimento de pressionamento do botão do mouse.

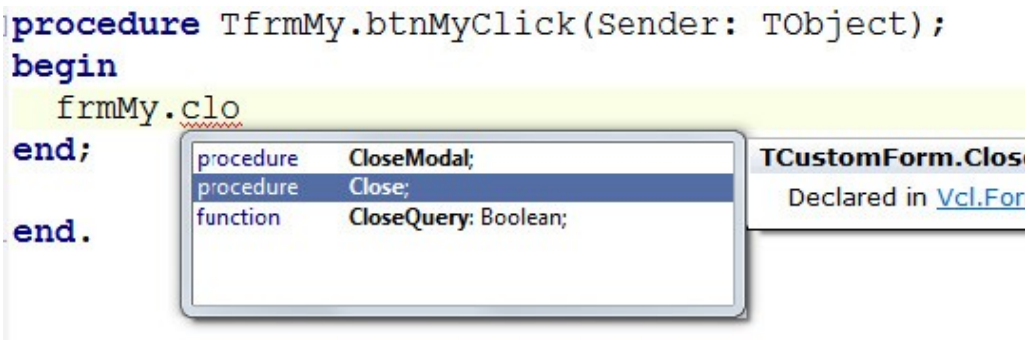
```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
end;
```

O cursor estará localizado entre as palavras-chave inicial e final. Aqui escreveremos uma instrução para fechar programaticamente a janela do form, um análogo completo do botão do sistema no título da janela. Como essa janela é a principal e a única no aplicativo, todo o programa será encerrado.

Digite ***frmMy*** e adicione um ponto ('.'). Após alguns momentos, o Delphi mostra as opções de conclusão do código. Essas opções são os métodos e propriedades disponíveis para o componente ***frmMy***.



Podemos encontrar a propriedade ou método adequado navegando na lista. Como a lista é relativamente grande, podemos acelerar a pesquisa inserindo alguns caracteres do nome e o número de elementos na lista diminuirá consideravelmente.



Finalizamos a entrada da conclusão do código pressionando Enter.

O código final para a resposta ao pressionamento de botão do comando ***btnMy*** tem a aparência semelhante a seguinte:

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
   frmMy.Close;
end;
```

O programa será fechado se você pressionar o botão do comando de pressionamento ***btnMy*** após o programa ser executado. Neste exemplo, usamos o método (procedimento) Close. Chamar um método mostra o que será feito com o componente. Você pode ver uma sintaxe de referência do método no exemplo: o nome do componente vem primeiro, em seguida, há um delimitador (ponto) e, por fim, o nome do método especificado após o delimitador.

```
<nome_do_componente>.<método>;
```

Todos os componentes têm seu próprio conjunto de métodos. Às vezes, é possível encontrar métodos idênticos em diferentes componentes.

Vamos demonstrar mais um método, ***Clear***. Esse método limpa o conteúdo e pode ser aplicado, por exemplo, ao componente ***Edit***.

```
Edit1.Clear
```

Como mencionamos acima, os componentes têm métodos e propriedades. Propriedades são atributos de componentes. As propriedades podem ser alteradas ou especificadas da seguinte forma: primeiro, referenciamos o nome do componente, em seguida, colocamos o delimitador (ponto) e, após o delimitador, especificamos o nome da propriedade (o Delphi nos permite escolher em uma lista de conclusão do código). Depois, colocamos o operador da atribuição, por fim, especificamos o valor da propriedade.

```
<nome_do_componente>.<propriedade>:=<valor>;
```

A maioria das propriedades de componente corresponde às exibidas na janela Object Inspector. Também há propriedades que não são refletidas ali, vamos examiná-las posteriormente. Por enquanto, vamos escrever um programa em que o pressionamento do botão do comando preencherá a janela do form com um plano de fundo branco. Nesse programa, precisamos usar a propriedade ***Color***.

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
     frmMy.Color:=clWhite;
end;
```

Podemos não apenas especificar propriedades do componente, mas lê-las também. Um uso dessa capacidade é atribuir um valor de propriedade de um componente a alguma propriedade de outro componente. A sintaxe é semelhante a essa:

```
     <nome_do_componente1>.<propriedade1>:=<nome_do_
componente2>.<propriedade2>;
```

Por exemplo:

```
lblMy.Caption:=edtMy.Text;
```

Isso instrui o programa a colocar o texto da caixa de texto ***edtMy*** na legenda do rótulo ***lblMy***.

Várias ações são separadas por ponto e vírgula.

Por exemplo:

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
   edit1.Clear;
   edit1.Color:=clBlue;
end;
```

Duas ações serão executadas após o botão ser pressionado: a caixa de texto será limpa e seu plano de fundo será preenchido com azul.

Exercícios

Exercício 1.

Crie um form que contém um rótulo e dois botões. O primeiro botão habilita o rótulo e o segundo desabilita.

Exercício 2.

Crie um form que contém um rótulo, uma caixa de texto e um botão. Um pressionamento de botão move o texto da caixa de texto para o rótulo e limpa a entrada da caixa de texto.

Exercício 3.

"Semáforo": Crie um form que contém três rótulos e três botões. Cada botão deve habilitar um rótulo apropriado e desabilitar os outros rótulos.

Exercício 4.

Crie um form que contém um rótulo e quatro grupos de botões. Cada grupo contém três botões. O primeiro grupo (três botões) altera a cor do plano de fundo do rótulo. O segundo grupo (três botões) altera a cor da fonte do rótulo. O terceiro grupo (três botões) altera o tamanho da fonte do rótulo. E o grupo de botões final (três botões) muda o nome da família da fonte do rótulo.

Variáveis e tipos de variáveis. Conversão de tipo

Para calcular um valor para uma expressão única com certas entradas, e fazer isso apenas uma vez, não é necessário escrever um programa de software completo. No entanto, se o processo dos cálculos precisar ser realizado frequentemente e com diferentes entradas a cada vez, faz sentido automatizar esse processo.

Operador de atribuição

Se você deseja converter graus (DEGR) para radianos (RAD), a fórmula de conversão terá a seguinte aparência no Delphi:

```
RAD:= (DEGR *3.14) / 180;
```

O operador de atribuição é uma instrução muito básica e importante em qualquer linguagem de programação. Ele atribui um valor de uma expressão calculada à direita do símbolo de atribuição **:=** a uma variável à esquerda do símbolo de atribuição. O símbolo de atribuição é composto por dois símbolos separados (**:** e **=**), no entanto, eles são interpretados como um único elemento no Delphi.

A maneira como funciona internamente é que o valor da expressão à direita do operador de atribuição é calculado primeiro e o valor resultante é atribuído à variável à esquerda do operador de atribuição.

O formato do operador de atribuição é o seguinte:

```
<nome da variável>:= <expressão>;
```

Agora, vamos falar sobre as variáveis. Uma variável é um local na memória que tem um nome. O tamanho da memória alocada para uma variável depende do valor a ser escrito nela. Em teoria, seria possível alocar somente grandes "slots" de memória para todas as variáveis. No entanto, se esse fosse o caso, ficaríamos sem memória disponível rapidamente.

Para saber exatamente a quantidade de memória a ser alocada para uma variável, precisamos conhecer seu ***tipo de dados***.

Você pode usar variáveis no Delphi para diversas tarefas e motivos. As variáveis precisam ser definidas antes de poderem ser usadas. A palavra-chave ***var*** é usada para iniciar uma seção de definições da variável, que está localizado após a definição do procedimento e antes da palavra-chave ***begin***. A seguir está o formato da definição da variável:

```
Var
    <nome da variável>: <tipo de dados>;
```

O nome de uma variável é uma sequência de caracteres com a seguinte sintaxe:

- Pode conter letras, números (0 a 9) ou sublinhado;
- Deve começar com uma letra.

O nome da variável pode ter qualquer comprimento (o número de caracteres não é limitado) e não diferencia maiúsculas de minúsculas.

Ao pensar sobre a nomenclatura de uma variável, observe que você deve evitar nomes de um caractere, exceto para variáveis de contador de loop ou temporárias.

As variáveis de contador de loop são os nomes I e J. Outras instâncias de nomes de variáveis de um caractere são S (string) ou R (raio). Os nomes de variáveis de um caractere sempre devem estar em letra maiúscula, mas é sempre preferível usar nomes mais significativos. É recomendável que você não use a letra I como um nome de variável para evitar confusão com o número 1 (um).

Exemplo:

```
procedure TfrmMy.btnMyClick(Sender: TObject); Var
   k: Integer;
begin
end;
```

Neste exemplo, uma variável ***k*** do tipo ***Integer*** é declarada.

Há muitos tipos de dados no Delphi. Vamos analisar alguns deles.

Os inteiros podem ser descritos com os seguintes tipos de dados:

Tipo de dados	Intervalo	Representação
Integer	–2.147.483.648 a 2.147.483.647	32 bits, com sinal
Cardinal	0 a 4.294.967.295	32 bits, inteiros positivos
ShortInt	–128 a 127	8 bits, com sinal
SmallInt	–32.768 a 32.767	16 bits, com sinal
LongInt	–2.147.483.648 a 2.147.483.647	32 bits, com sinal
Int64	-2^{63} a $2^{63}-1$	64 bits, com sinal
Byte	0 a 255	8 bits, sem sinal
Word	0 a 65535	16 bits, sem sinal
LongWord	0 a 4.294.967.295	32 bits, sem sinal

Usaremos principalmente ***Integer*** ao longo desse manual.

Números reais também podem ser representados por vários tipos de dados. Usaremos o tipo de dados ***Real***.

Tipo de dados	Intervalo	Dígitos significativos (precisão)	Bytes
Real48	$\pm 2{,}9 \times 10^{-39}$ a $\pm 1{,}7 \times 10^{38}$	11 a 12	6
Real	$\pm 5{,}0 \times 10^{-324}$ a $\pm 1{,}7 \times 10^{308}$	15 a 16	8
Single	$\pm 1{,}5 \times 10^{-45}$ a $\pm 3{,}4 \times 10^{38}$	7 a 8	4
Double	$\pm 5{,}0 \times 10^{-324}$ a $\pm 1{,}7 \times 10^{308}$	15 a 16	8
Extended	$\pm 3{,}6 \times 10^{-4951}$ a $\pm 1{,}1 \times 10^{4932}$	19 a 20	10
Comp	-2^{63} a $2^{63}-1$	19 a 20	8
Currency	–922337203685477,5808 a +922337203685477,5807	19 a 20	8

Você conhece a notação científica normalizada (também conhecida como notação exponencial) para representação de números reais. Por exemplo:

$3{,}28 \cdot 10^{17}$ $1{,}4 \cdot 10^{-9}$ $-5{,}101 \cdot 10^{4}$

No Delphi, esses números são registrados como:

3.28e+17	1.4e-09	-5.101e+4
328e15	0.14e-8	-5101e+1
0.328e+18	140e-11	-510100e-1

Expressões aritméticas

Vamos analisar os operadores de atribuição novamente. O tipo de dados à esquerda deve corresponder ao tipo de dados do valor da expressão aritmética à direita.

Uma expressão aritmética é uma expressão matemática legal composta usando constantes, variáveis, funções, operadores aritméticos (como +, *, —, /, exponente e parênteses (e)) .

Operadores usados em expressões aritméticas:

Operador	Nome do operador	Tipos de dados dos operandos	Tipo de dados do resultado
+	Adição	Pelo menos um dos operandos é real	Real
		Integer	Integer
-	Subtração	Pelo menos um dos operandos é real	Real
		Integer	Integer
*****	Multiplicação	Pelo menos um dos operandos é real	Real
		Integer	Integer
/	Divisão	Real, Integer	Real
Div	Divisão de integer	Integer	Integer
Mod	Resto da divisão de integer	Integer	Integer

Exemplos:

13 div 4 = 3	13 mod 4 = 1
–13 div 4 = –3	–13 mod 4 = –1
13 div –4 = –3	13 mod –4 = 1
–13 div –4 = 3	–13 mod –4 = –1
0 div 2 = 0	1 mod 2 = 1
2 div 5 = 0	2 mod 5 = 2

Se o valor da expressão for Integer, ele poderá ser atribuído à variável do tipo de dados Real, mas não vice-versa.

Adicionando dois números

Crie o form ***frmCalc*** com duas caixas de testo TEdit ***edtA*** e ***edtB***, um botão ***btnAdd*** e o rótulo ***lblResult***.

Defina a propriedade ***Caption*** do botão e deixe as propriedades ***Text*** das caixas de texto em branco. Clique duas vezes no botão, a janela de código será aberta e poderemos começar a codificar um manipulador de evento de clique do botão.

Primeiro, declararemos as variáveis:

```
Var
   a, b, c: Single;
```

Obteremos os valores a e b das caixas de texto edtA e edtB respectivamente. Observe que inserimos valores de string nas caixas de texto, portanto precisamos convertê-los para o tipo de dados numérico. Usaremos a função StrToFloat (string) para esse fim.

```
  Begin
  a:=StrToFloat(edtA.Text); //coloque o número A
  b:=StrToFloat(edtB.Text); //coloque o numero B
  c:=a+b; //faça a adição de A e B
  lblResult.Visible:=true; //Deixe o label que receberá
a resposta visível
  lblResult.Caption:=FloatToStr(c); //Insira o valor da
resposta no label (convertendo para o tipo string antes)
  end;
```

O resultado da adição de duas variáveis é armazenado em uma terceira variável (c) e esse resultado é convertido novamente em uma string usando FloatToStr (number) e exibido no rótulo lblResult. Para deixar o código ainda mais bonito, vamos escrevê-lo da seguinte forma:

```
lblAnswer.Caption:=edtA.Text+'+'
                +edtB.Text+'='+FloatToStr(c);
```

Ao trabalhar com **StrToFloat** e **FloatToStr** no Delphi, é importante se atentar à formatação adequada do separador decimal. No programa (Unidade), somente ponto (".") é permitido como separador decimal. No entanto, nas janelas de entrada e saída no form, o separador decimal depende das configurações regionais do Windows. Por exemplo, em muitos países europeus, a vírgula é usada para separar as partes inteiras e fracionárias de um número decimal.

As funções **IntToStr(integer)** e **StrToInt(string)** são usadas para converter inteiros em strings e strings em inteiros respectivamente.

Exercícios

Exercício 1.

Usando duas caixas de texto, um rótulo e quatro botões, crie uma calculadora para quatro operações aritméticas básicas.

Exercício 2.

Escreva um programa para converter Fahrenheit em Celsius e vice-versa. (Tf=9/5*Tc+32).

Exercício 3.

Escreva um programa para converter velocidades de quilômetros por hora em metros por segundo e vice-versa.

Funções matemáticas padrão

Em expressões, é possível usar funções matemáticas padrão com operadores e operações padrão. Você deve se atentar ao argumento e tipos de resultado para obter os resultados corretos.

Funções padrão na linguagem Delphi

Função	Computação	Parâmetros de entrada	Tipo de resultado	Exemplos
`ABS(X)`	Retorna o valor absoluto de um número.	X — valor REAL ou INTEGER	Igual ao tipo de parâmetro	`ABS(2.0) = 2.0000e+00;`
`SQR(X)`	Retorna o quadrado de um número.	X — valor REAL ou INTEGER	Igual ao tipo de parâmetro	`SQR(3) = 9;` `SQR(-2.0)= 4.0000e+00;`
`SQRT(X)`	Retorna a raiz quadrada de um número.	X — valor REAL ou INTEGER	REAL	`SQRT(16) =4.0000e+00;` `SQRT(25.0)=5.0000e+00;`
`EXP(X)`	Retorna a exponenciação de um número.	X — valor REAL ou INTEGER	REAL	`EXP(0)=1.00000e+00;` `EXP(-1.0)=3.67879e-01;`
`LN(X)`	Retorna o logaritmo natural de um número.	X — valor REAL ou INTEGER	REAL	`LN(1) = 0.00000e+00` `LN(7.5) = 2.01490e+00`
`SIN(X)`	Retorna o seno do parâmetro.	X — valor REAL ou INTEGER, em radianos	REAL	`SIN(0)=0.00000e+00;` `SIN(1.0)=8.41471e-01`
`COS(X)`	Retorna o cosseno do parâmetro.	X — valor REAL ou INTEGER, em radianos	REAL	`COS(0)=1.00000e+00;` `COS(1.0)=8.41471e-01`

`ARCTAN(X)`	Retorna o arco tangente do parâmetro.	X — valor REAL ou INTEGER	REAL	`ARCTAN(0)=0.0000e+00`
				`ARCTAN(-1.0)=`
				`-7.8539e-01`

Função	Computação	Parâmetros de entrada	Tipo de resultado	Exemplos

`ROUND(X)`	Arredonda um valor real para o valor inteiro mais próximo (por valor absoluto).	X — REAL	INTEGER	ROUND(3.1) = 3;
				ROUND(–3.1)= –3;
				ROUND(3.8) = 4;
				ROUND(–3.8)= –4;
				Observação: números com partes fracionárias iguais a 0,5 são arredondados para o número par mais próximo.
				`ROUND(3.5) = 4;`
				`ROUND(2.5) = 2;`

`TRUNC(X)`	Retorna a parte inteira do parâmetro eliminando sua parte fracionária.	X — REAL	INTEGER	`TRUNC(3.1) = 3;`
				`TRUNC(-3.1)= -3;`
				`TRUNC(3.8) = 3;`

`INT(X)`	Retorna a parte inteira do parâmetro eliminando sua parte fracionária.	X — REAL	REAL	`INT(3.1) = 3.00000E+00`
				`INT(-3.1) = -3.00000E+00`
				`INT(3.8) = 3.00000E+00`

Exercícios

Exercício 1.

Dado um número real, mostre suas partes inteiras e fracionárias separadamente.

Exercício 2.

Supondo que a Terra é uma esfera ideal com o raio R=6350 km, escreva um programa para calcular a distância para a linha do horizonte do ponto de uma determinada altura a partir da superfície da Terra.

Exercício 3.

Calcule e exiba a soma e o produto de três números inseridos usando o teclado.

Exercício 4.

O triângulo é especificado pelas coordenadas de seus vértices. Calcule e exiba o perímetro e a área do triângulo.

Exercício 5.

Calcule a altura da árvore dada a distância até a árvore e seu ângulo de visão. O resultado deve ser exibido da seguinte forma:

A altura da árvore é igual a 2 m e 87 cm

MÓDULO 5

Expressões lógicas, Variáveis do tipo Boolean e Operações lógicas

O lado direito da instrução de atribuição pode não ser apenas uma expressão aritmética, mas uma expressão de outro tipo, por exemplo, uma expressão lógica.

Uma expressão lógica (Boolean) é uma expressão que resulta em um valor de TRUE ou FALSE. O nome *Boolean* foi escolhido em homenagem ao matemático inglês George Boole que estabeleceu os fundamentos da lógica matemática. Os termos *Boolean e lógico* normalmente são usados como sinônimos.

O valor de uma expressão lógica pode ser atribuído a uma variável Boolean.

Exemplo de variável lógica:

```
Var
Exist: Boolean;
```

As expressões lógicas podem incluir: expressões aritméticas, operadores relacionais e operadores lógicos.

Operadores relacionais

Os operadores relacionais são usados para comparar dois valores. O resultado da comparação tem o valor **TRUE** ou **FALSE**.

=	—	igual a
<>	—	diferente de
<	—	menor que
<=	—	menor ou igual a
>	—	maior que
>=	—	maior ou igual a

Exemplo:

```
Var
   X: Real;
   Exist, Ok: Boolean;
begin
   X:= 2.5;
   Ok:= X > 0;
   Exist:= X = 3–27;
end.
```

Como resultado dessa execução do programa, a variável **Ok** armazena o valor **TRUE** e a variável **Exist** armazena **FALSE**.

Operadores lógicos

Os operadores lógicos são usados com valores lógicos e eles também retornam um valor lógico.

Vamos examinar os seguintes operadores lógicos:

NOT

AND

OR

Operações lógicas e valores

X	Y	Not X	X And Y	X Or Y
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

O valor da expressão é calculado em uma certa ordem.

Tabela de precedência de operadores

Tipo de expressão	Operador
Avaliação em parênteses	**()**
Avaliação de função	**funções**
Operadores unários	**not, unary '-'**
Operadores como multiplicação	*** / div mod and**
Operadores como adição	**+ — or**
Operadores relacionais	**= <> < > <= >=**

Os operadores com a mesma precedência são avaliados da esquerda para a direita de acordo com sua ordem na expressão.

Como exemplo, vamos examinar a ordem dos operadores e encontrar o valor da próxima instrução:

```
(a * 2 > b) or not (c = 7) and (d – 1 <= 3), if a = 2,
b = 4, c = 6, d = 4.
```

```
(2 *2 > 4) or not (6 = 7) and (4 – 1 <= 3)
(4 > 4) or not (6 = 7) and (3 <= 3)
false or not false and true
false or true and true
false or true
True
```

A instrução matemática $-4 < x \leq 18+3$ no Delphi será escrita como:

```
(x > –4) and (x <= 18 + 3)
```

Exercícios

Exercício 1.

Crie um form com um rótulo e um botão que ativa e desativa o rótulo.

Exercício 2.

Semáforo: Crie um form com três rótulos e três botões. Cada botão ativa seu próprio rótulo (vermelho, amarelo, verde) com sua cor correspondente e desativa os demais.

Exercício 3.

Crie um form com duas caixas de texto e um botão. Quando o botão é pressionado, o rótulo True deve aparecer se o número na primeira caixa de texto for maior que o número na outra e, caso contrário, o rótulo False deve aparecer.

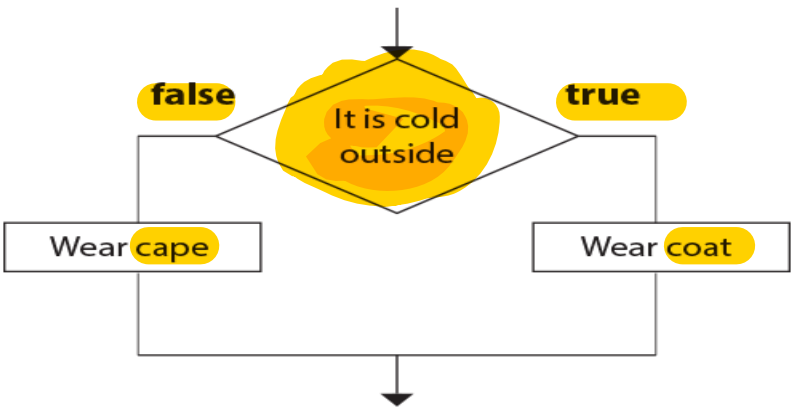
Exercício 4.

Um inteiro de três dígitos é inserido na caixa de texto. Escreva um programa que mostra o rótulo True se a soma de quaisquer dois dígitos for igual ao terceiro dígito, caso contrário, False.

Execução condicional no programa. Instrução IF...THEN...ELSE

Até o momento, escrevemos programas com um curso sequencial de ações. Nesses programas, as ações foram executadas conforme foram escritas, uma após a outra. Essa estrutura de programa é chamada de *estrutura de programa linear*.

Na vida real, frequentemente nos deparamos com diversas escolhas. Utilizamos casacos se estiver frio, caso contrário usamos capas. Essa situação pode ser apresentada como o fluxograma a seguir:



Dependendo da veracidade da afirmação "está frio lá fora" na caixa em forma de diamante, executamos as ações "vestir capa" ou "vestir casaco".

Esta estrutura é chamada de *estrutura de ramificação*.

Escolher uma de duas ou mais alternativas é bastante comum em programação. Na linguagem Delphi, a seleção é realizada utilizando uma instrução ***If… Then…Else***.

Sintaxe:

```
If <expressão lógica> Then
   <instrução 1>
Else
   <instrução 2>;
```

A ordem da execução de ***If…Then…Else***:

Primeiro, a *expressão lógica* é avaliada,

Se a *expressão lógica* for avaliada como ***True***, a *instrução 1* será executada,

Caso contrário (a *expressão lógica* for avaliada como ***False***), a *instrução 2* será executada.

Exemplo 1.

Dados dois números inteiros, calcule o valor máximo deles.

Solução:

```
procedure TfrmMy.btnMaxClick(Sender: TObject);
var
   a, b, m: integer;
begin
   a:=StrToInt(edtA.Text);
   b:=StrToInt(edtB.Text);
   if a>b then
        m:=a
   else
        m:=b;
   lblMax.Caption:=IntToStr(m)
end;
```

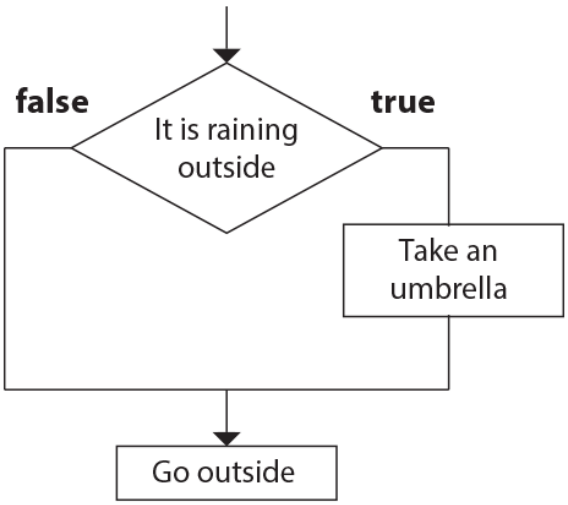
Deve-se observar que a instrução ***If…Then…Else*** não precisa de um ponto e vírgula antes da palavra-chave ***Else***, porque ela é uma instrução complexa única e não pode haver ***Else*** sem ***If***.

Você pode usar instruções compostas quando a linguagem Delphi permite apenas uma única ação, mas é necessário executar várias ações. Uma instrução composta consiste em várias instruções delimitadas nos colchetes do operador ***begin…end***.

Por exemplo, se for necessário executar mais de uma única instrução nas ramificações ***Then*** ou ***Else***, essas instruções são delimitadas nos colchetes do operador ***begin…end***:

```
…
If (x < 0) Then
begin                    ┐
   y:= 1;                │
   k:= k + 1;            │  instrução composta
end                      ┘
Else
   begin                 ┐
   y:= 2 * x;            │  instrução composta
   k:= x;                ┘
end;
```

Às vezes as opções são diferentes:



Essa estrutura é representada pela instrução ***If…Then*** mais curta.

Sintaxe:

```
If <expressão lógica> Then
   <instrução 1>;
```

A versão curta da instrução condicional é executada exatamente como a versão longa. Se a *expressão lógica* for avaliada como ***True***, a *instrução 1* será executada. Caso contrário, nenhuma instrução é executada durante a execução da instrução ***If…Then***.

Exercícios

Exercício 1.

Dois inteiros m e n são inseridos pelo usuário.

Se n for divisível por m, o resultado da divisão de m por n deverá ser exibido. Caso contrário, o programa deve exibir a mensagem "n cannot be divided by m".

Exercício 2.

Dado um inteiro de três dígitos, decida se ele é um ***palíndromo***, isto é, se ele é lido da esquerda para a direita e da direita para a esquerda.

Exercício 3.

Encontre o máximo e o mínimo de três números reais diferentes.

Exercício 4.

Uma casa contém ***n*** aposentos numerados sequencialmente, começando com o número ***a***. Descubra se a soma de todos os cômodos é par ou não.

Instrução If...Then...Else aninhada. Praticando resolução de tarefas

As instruções após ***Then*** e ***Else*** também podem ser instruções condicionais. Nesse caso, a instrução If...Then...Else é chamada de uma instrução aninhada.

Por exemplo:

```
If <expressão lógica 1> Then
   <instrução 1>
Else
   If <expressão lógica 2> Then
   <instrução 2>
   Else
   <instrução 3>;
```

Cada uma das instruções (instrução 1, instrução 2, instrução 3) pode ser composta, isto é, em Then ou Else é possível colocar mais de uma instrução, colocando-os entre colchetes em um bloco **begin... end**.

Por exemplo:

```
If <expressão lógica 1> Then
   begin
   <instrução 1>
   If <expressão lógica 2> Then
   begin
   <instrução 2>
   <instrução 3>
end
   Else
   <instrução 4>;
end
Else
   If <expressão lógica 3> Then
   <instrução 5>
   Else
   begin
   <instrução 6>;
   <instrução 7>;
   end;
```

Vamos examinar o próximo fragmento de código:

```
If <expressão lógica 1> Then
   If <expressão lógica 2> Then
   <instrução 1>
Else
   <instrução 2>;
```

Pode parecer ambíguo, uma vez que não é claro a qual ***Then Else*** pertence (ao primeiro ***Then*** ou ao segundo ***Then***). Há uma regra em que o ***Else*** pertence ao ***If*** mais próximo.

Para evitar confusão e possíveis erros, é desejável colocar ***If… Then… Else*** aninhada no bloco ***begin***… ***end***.

Exercícios

Exercício 1.

Encontre a raiz quadrada da equação quadrática $Ax^2+Bx+C=0$ ($A\neq0$). Se a equação não tiver raízes válidas, mostre a mensagem correspondente.

Exercício 2.

Descubra se um determinado ano é um ano bissexto. Um ano é um ano bissexto se ele for divisível por 4, mas entre os anos que também são divisíveis por 100, os anos bissextos são apenas os que são divisíveis por 400. Por exemplo, 1700, 1800 e 1900 não são anos bissextos, mas 2000 é bissexto.

Exercício 3.

Os números reais X, Y (X≠Y) são indicados. Substitua o número menor por sua média aritmética e substitua o número maior com o produto triplo.

Exercício 4.

O inteiro k (1 <= k <= 180) é indicado. Descubra qual dígito está na posição k da sequência 10111213 … 9899 de todos os números de dois dígitos em ordem.

Procedimentos

Não é suficiente conhecer os operadores de uma linguagem de programação para escrever programas com facilidade, usando o mínimo de tempo e introduzindo o mínimo de erros possível. Em vez disso, é necessário se tornar proficiente nos princípios e habilidades de programação procedimental ou estruturada.

A aplicação prática desses princípios permite que você:

- escreva programas facilmente
- crie programas legíveis e compreensíveis
- encontre os erros do programa mais rápido e
- altere e melhore os programas mais rapidamente

Um dos principais princípios da programação estruturada é chamado de ***programação top-down***. Esse princípio indica que a tarefa principal de qualquer programa deve ser dividida em várias subtarefas. Em seguida, cada subtarefa deve ser dividida em subtarefas ainda mais simples e esse processo deve continuar até acabar com um conjunto de pequenas tarefas simples que são fáceis de implementar.

As subtarefas separadas são chamadas ***procedimentos*** na linguagem do Delphi.

Vamos resolver a tarefa de encontrar as raízes da equação quadrática. A equação será especificada como três coeficientes.

Primeiro criamos um novo form chamado ***frmQuadEq*** e inserimos "Solution of quadratic equation" na propriedade ***Caption***.

Criaremos as caixas de texto ***edtA***, ***edtB*** e ***edtC*** para os valores das variáveis A, B e C correspondentes. Aceitamos a propriedade ***Text*** dessas caixas para a entrada mais rápida dos coeficientes.

Um rótulo ***lblCoefs*** é colocado acima dessas caixas de texto, sua propriedade Caption deve conter "Enter coefficients of quadratic equation".

Também precisamos que os rótulos ***lblX1*** e ***lblX2*** mostrem os valores das raízes da equação. O rótulo ***lblNo*** contém o texto "No real roots are found" na

propriedade ***Caption***. Definimos a propriedade ***Visible*** desses rótulos como False para que não sejam exibidos na inicialização do programa.

Também precisamos de um botão ***btnFind*** para executar o processo da solução.

Vamos clicar duas vezes e começar a escrever o código.

Primeiro documentaremos nossa solução usando comentários. Os comentários no Delphi podem ser colocados dentro de chaves ou, mais convenientemente, após duas barras //:

```
//Entre com os coeficientes da equação quadrática A, B
e C
//Compute a discriminante △
//SE △ >= 0, Então
      //Compute as raízes X1 e X2 da equação
      //Mostre as raízes
//Senão
      //Notifique o usuário que não há solução
```

Essa visão geral da solução nos fornece o número de variáveis, procedimentos e funções necessários. Se uma única linha de uma solução precisar ser expressa em várias instruções do Delphi, então escreveremos um procedimento, caso contrário uma única instrução será suficiente.

Dada a consideração acima, vamos escrever a solução em Delphi:

```
procedure TfrmQuadEq.btnFindClick(Sender: TObject); var
   A, B, C, D, X1, X2: real;
begin
   coefInput(A, B, C); //leia dos coeficientes A, B, C
   D:=sqr(B)-4*A*C; //compute the discriminante Δ
if D>=0 then //Se a discriminante >= 0, então
begin
   calc(A, B, D, X1, X2); //compute as raizes x1, x2 de
   //A, B e Δ
   prn(X1, X2); //mostre as raízes x1 e x2
end
else //senão
   lblNo.Visible:=true; //Notifique o usuário da falta
//de solução
end;
```

No corpo do procedimento, vemos os nomes das ações que o programa precisa executar para resolver a tarefa. Essas ações são chamadas de procedimentos, os valores dentro dos parênteses são chamados de parâmetros de procedimento.

Quando o Delphi encontra o nome de uma ação que não é uma palavra-chave ou uma instrução, ele assume que é a chamada de procedimento. Ele então transfere o controle para o procedimento com um certo nome (em nosso caso, os nomes são coefInput, calc e prn). Após executar as instruções do procedimento, a execução retorna à rotina de chamada e o programa executa a próxima instrução após a chamada de procedimento.

Sintaxe da chamada de procedimento:

```
<nome do procedimento> (<lista de parâmetros de chamada>);
```

Os procedimentos devem ser definidos na seção ***Implementation*** antes de se referir a eles na chamada. Você deve escrever o procedimento antes de chamá-lo.

Sintaxe da definição do procedimento:

```
procedure <nome do procedimento> (<lista de parâmetros formais>);
    <declarações>
    begin
      <instruções>
    end;
```

Como vemos, a chamada de procedimento usa os parâmetros de chamada e a definição de procedimento usa parâmetros formais. A lista de parâmetros formais especifica tipos para todos os parâmetros. Há uma correspondência de um para um entre os parâmetros de chamada e parâmetros formais.

O número, a ordem e os tipos de chamada e os parâmetros formais devem ser idênticos.

Vamos definir nossos procedimentos, começando com o procedimento prn.

Nesse procedimento, os parâmetros formais Xf e Xs correspondem aos parâmetros de chamada X1 e X2. Os parâmetros formais Xf e Xs são parâmetros de valor, eles são transmitidos do autor da chamada para Prn e não são retornadas para o autor da chamada.

```
Procedure Prn(Xf, Xs: real);
begin
    frmQuadEq.lblX1.Visible:=true;
    frmQuadEq.lblX1.Caption:='x1='+FloatToStr(xf);
    frmQuadEq.lblX2.Visible:=true;
    frmQuadEq.lblX2.Caption:='x2='+FloatToStr(xs);
end;
```

Aqui tornamos os rótulos com os valores de raiz visíveis e colocamos as respostas lá. Receberíamos um erro se escrevêssemos ***lblX1.Visible:=true***. Nos procedimentos definidos pelo usuário, precisamos escrever o nome completo do componente (especificando a qual form ele pertence, porque pode haver vários forms no programa, em geral).

Vamos escrever um procedimento para inserir coeficientes. Precisamos obter os coeficientes das caixas de texto e colocá-las nas variáveis correspondentes. Para os valores de variáveis serem retornados do procedimento para o programa principal, precisamos usar parâmetros de variável. Os parâmetros de variável não apenas transmitem dados para procedimentos, eles também colocam dados nas variáveis do programa principal. Para definir parâmetros como variáveis, precisamos usar a palavra-chave ***var*** antes delas. Os parâmetros de chamada devem ter variáveis para os parâmetros de variável do procedimento:

```
Procedure coefInput(var k1, k2, k3: real);
begin
   k1:=StrToFloat(frmQuadEq.edtA.Text);
   k2:=StrToFloat(frmQuadEq.edtB.Text);
   k3:=StrToFloat(frmQuadEq.edtC.Text);
end;
```

Os parâmetros de variáveis formais *k1*, *k2* e *k3* correspondem aos parâmetros de chamada *a*, *b* e *c*. Observe o uso da função StrToFloat. Precisamos usá-la porque a propriedade Text de uma caixa de texto é do tipo string e nossas variáveis são do tipo Real. Precisamos converter os tipos usando essa função e o uso da função também permite a conclusão do código para reconhecer adequadamente sua intenção.

Vamos definir o procedimento final, como calcular as raízes das equações. Precisamos retornar valores de raiz, portanto os parâmetros formais Xf e Xs serão parâmetros de variável. Não precisamos retornar o primeiro e o segundo coeficientes e valores discriminantes. Eles serão transmitidos como parâmetros de valor.

```
procedure Calc(k1, k2, dis: real; var Xf, Xs:
Real);
begin
   Xf:=(-k2+Sqrt(dis))/(2*k1);
   Xs:=(-k2-Sqrt(dis))/(2*k1);
end;
```

Após executarmos o programa e inserirmos os coeficientes, veremos um ou outro conjunto de rótulos no form. Alguns rótulos permanecerão visíveis se inserirmos outro conjunto de coeficientes para gerar outro conjunto de rótulos. Para melhorar o comportamento do programa, definiremos outro procedimento que colocará tudo em um estado inicial limpo. Esse será o procedimento sem parâmetros que chamaremos de ***Init***:

```
procedure Init;
begin
   frmQuadEq.lblX1.Visible:=false;
   frmQuadEq.lblX2.Visible:=false;
   frmQuadEq.lblNo.Visible:=false;
end;
```

Exercícios

Exercício 1.

Crie um programa que troque os valores entre as variáveis **A** e **B** e troca os valores das variáveis **C** e **D**. Defina um procedimento que troque os valores de duas variáveis.

Exercício 2.

Dados os comprimentos dos lados de dois triângulos, encontre a soma de seus perímetros e seus quadrados. Defina um procedimento para calcular o perímetro e a área de um triângulo se você souber os comprimentos de seus lados.

Exercício 3.

Encontre o quadrado de um anel com o raio interno R1 e o raio externo R2. Defina e use esse procedimento para calcular a área de um círculo.

Exercício 4.

Encontre uma área do quadrilátero convexo especificado pelas coordenadas de vértices.

Funções

Agora, vamos examinar as funções. Funções são iguais aos procedimentos, exceto que elas retornam um valor além de executar instruções.

O Delphi tem muitas funções integradas, mas há muitas situações em que é necessário escrever suas próprias funções. Por exemplo, precisamos resolver uma tarefa usando a função de tangente Tg(A) várias vezes, mas não existe essa função integrada. A seguir está um exemplo simples:

Há um triângulo retângulo. O ângulo (em graus) e o comprimento do lado adjacente são indicados. Calcule o outro lado.

Clique duas vezes no botão para entrar no editor de código:

```
procedure TfrmCatheti.btnRunClick(Sender: TObject); var
   a, b.alfa: real;
begin
   alfa:=StrToFloat(edtAlfa.Text); //Set angle alfa
   a:=StrToFloat(edtAlfa.Text); //Set adjacent leg a
   b:=a*tg(alfa); //Calculate another leg b
   lblB.Caption:= FloatToStr(b); //Output b
end;
```

Precisávamos apenas descrever nossa função. As funções são descritas na seção *Implementation*. Quando você declara uma função, especifica seu nome, número e o tipo de parâmetros necessários e o tipo de seu valor de retorno.

Formato da declaração da função:

```
Function <nome da função> (<lista de parâmetros >):
<tipo de valor de retorno>;
   <declarações locais>
begin
   <instruções>
end;
```

As funções têm parâmetros de valor e parâmetros de variável. Os parâmetros são governados pelas mesmas regras que se aplicam aos parâmetros do procedimento.

As funções devem ter pelo menos um operador de atribuição, que atribui um valor ao nome da função.

```
Function tg(x: real): real;
var
   y: real;
Begin
   y:= x / 180 * pi;
   tg:= sin(y) / cos(y);
End;
```

Aqui usamos uma variável adicional chamada de variável local, ela é declarada dentro da função.

Exercícios

Exercício 1.

Calcule o valor da expressão: $x = \frac{\sqrt{6}+6}{2} + \frac{\sqrt{13}+13}{2} + \frac{\sqrt{21}+21}{2}$, usando a função $y = y = \sqrt{x} + x$

Exercício 2.

Calcule o valor da expressão $x = \frac{15+\sqrt{8}}{8+\sqrt{15}} + \frac{6+\sqrt{12}}{12+\sqrt{6}} + \frac{7+\sqrt{21}}{21+\sqrt{7}}$, usando a função $y = a + \sqrt{b}$

Exercício 3.

Dadas as coordenadas de três vértices de um triângulo, calcule seu perímetro.

Exercício 4.

O usuário inserirá as variáveis a, b, c. Calcule o valor da expressão $t = \frac{\max(a, b, c) - \min(a, b, c)}{2 + \max(a, b, c) \cdot \min(a, b, c)}$, em que as funções min(a, b, c) e max(a, b, c) retornam, respectivamente, o mínimo e o máximo de três números.

Exercício 5.

Calcule o valor da expressão, em que

Exercício 6.

As coordenadas dos pontos A, B e C, situados no plano, são inseridas pelo teclado. Descubra se é possível construir um triângulo não degenerado nesses pontos. Se for possível, calcule os comprimentos de seus lados, alturas, área, perímetro e ângulos em graus. Emita os resultados na tela, tendo especificado quais ângulos, lados e alturas foram calculados. Por exemplo, lado AB=..., ângulo BCA=... e assim por diante. Ao escrever o programa, defina funções para calcular o comprimento dos lados, tamanhos de ângulos e alturas.

Se o triângulo não puder ser criado, emita a mensagem correspondente.

Gráficos

O Delphi tem uma ampla gama de recursos gráficos. Discutiremos alguns deles neste capítulo. Os desenhos serão colocados no objeto ***TPaintBox***.

Selecione a guia *Additional* na paleta de componentes. No menu de ícone dessa guia, selecione ***TPaintBox***. Como um pintor estica a tela sobre a moldura, esticamos nossos componentes no form.

Nossa mídia de pintura é a propriedade ***Canvas*** do componente. A origem é no canto superior esquerdo do objeto, o X aponta horizontalmente para a direita e o eixo Y aponta verticalmente para a parte inferior. As dimensões da imagem podem ser visualizadas no balão ao lado do cursor do mouse quando ele está sobre o objeto ou elas podem ser consultadas programaticamente pelas propriedades ***Width*** e ***Height*** do objeto. Podemos alterar o nome do objeto para ***pbxEx*** e calcular o centro do componente ***TPaintBox***:

```
x0:= pbxEx.Width div 2;
y0:= pbxEx.Height div 2;
```

A imagem ***Canvas*** é criada usando várias ferramentas de desenho. Podemos desenhar usando uma caneta, preencher os desenhos com pincéis e exibir o texto usando fontes.

Para alterar a cor e a largura das linhas de qualquer figura, usamos o objeto ***Pen***:

```
pbxEx.Canvas.Pen.Color:=clRed;
pbxEx.Canvas.Pen.Width:=3;
```

As linhas de figuras serão desenhadas usando vermelho e uma largura um pouco maior do que a largura de linha padrão.

O objeto ***Brush*** é usado para preencher o espaço interno de figuras fechadas.

```
pbxEx.Canvas.Brush.Color:=clGreen;
```

A cor do pincel será verde a partir de agora.

Um exemplo simples de como definir a cor do pincel para branco e preencher toda a ***Canvas*** com a cor branca:

```
pbxEx.Canvas.Brush.Color:= clwhite;
pbxEx.Canvas.FillRect(ClientRect;
```

Os estilos de linha e preenchimento podem ser definidos na propriedade ***Style***. Deixaremos para o leitor descobrir os detalhes de trabalhar com o estilo sozinho.

Vamos examinar os primitivos gráficos padrão que podem ser colocados na ***Canvas***.

Ponto

Para definir um ponto (X, Y) para uma cor específica, usamos a propriedade ***Pixels[X, Y]***:

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

Em outras palavras, somente um pixel nos cordões X e Y é configurado para uma cor específica, vermelho aqui.

Linha

As linhas podem ser desenhadas com o método ***LineTo***:

```
pbxEx.Canvas.LineTo(X, Y)
```

Uma linha será desenhada da posição atual da caneta para até as coordenadas específicas. O cursor será movido para as coordenadas especificadas.

A caneta pode ser movida até o ponto (X, Y) sem nenhum desenho usando o método ***MoveTo***:

```
pbxEx.Canvas.MoveTo(X, Y)
```

Todas as outras figuras não movem a caneta.

Retângulo

Para desenhar um retângulo, podemos usar o método ***Rectangle***:

```
pbxEx.Canvas.Rectangle(X1, Y1, X2, Y2
```

Essa chamada desenhará um retângulo com lados alinhados ao eixo e uma diagonal de (X1, Y1) a (X2, Y2).

Os estilos de linha e o pincel de preenchimento atuais serão usados. Por padrão, a cor da linha é preta e a cor de preenchimento é branca.

Elipse

Uma elipse é especificada pelo retângulo de circunscrição. Os eixos de elipse são paralelos aos eixos X e Y. Como com o retângulo, a elipse é desenhada preenchida.

```
pbxEx.Canvas.Ellipse(X1, Y1, X2, Y2)
```

Exercícios

Exercício 1.

Desenhe um boneco de neve.

Exercício 2.

Desenhe uma casa.

Exercício 3.

Pense sobre sua própria imagem e desenhe-a em ***Canvas***. Use diferentes primitivos, linhas, larguras de linha e cores de preenchimento.

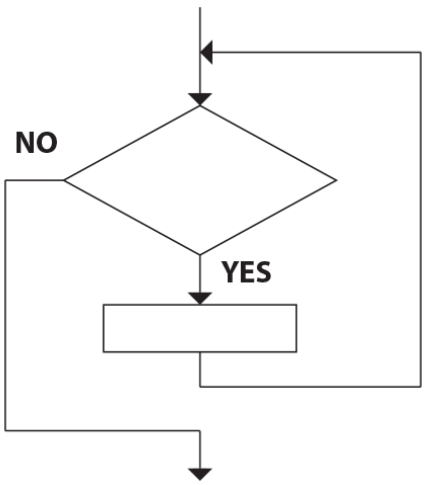
Loops

O **Loop** é uma sequência de instruções executada repetidamente. Há dois tipos de loop com um loop de precondição (loop ***While...do***...) e um loop de pós-condição (loop ***Repeat...Until...***)

O loop de precondição ***While...do***... no Delphi tem o seguinte formato.

```
While <expressão lógica> do
    <instrução>;
```

O loop ***While...do*** executará repetidamente a mesma instrução se a expressão condicional for True. Uma expressão condicional normalmente é alterada dentro do loop e é calculada antes da execução da instrução. Se o valor da expressão for *False* desde o início, a instrução não será executada.



Fluxograma de ***While...do***

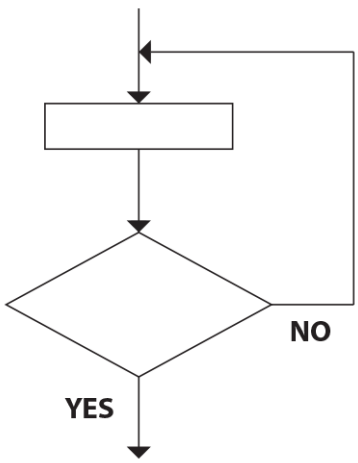
Se você precisar executar várias instruções, é necessário usar ***begin... end*** após ***do***:

```
While <expressão lógica> do
begin
   <instrução 1>;
   <instrução 2>;
   <instrução 3>;
end;
```

O loop de pós-condição ***Repeat...Until...*** tem o formato:

```
Repeat
    <instrução 1>;
    <instrução 2>;
    ...
    <instrução n>;
Until <expressão lógica>
```

Nesse tipo de loop, as instruções do corpo do loop (a parte do programa entre ***Repeat*** e ***Until)*** são executadas primeiro. Em seguida, a expressão lógica é avaliada: se for ***False***, a execução das instruções será repetida, caso contrário (se a expressão lógica for True), o loop será encerrado. Portanto, em ***Repeat...Until*** o corpo do loop sempre é executado pelo menos uma vez e a expressão lógica é a condição da conclusão do loop.



Fluxograma de *Repeat… Until*

Se o número de execuções do loop do corpo for conhecido, você pode usar o loop com um contador ***For***.

Formato:

```
For <contador do loop>:= <valor inicial> To <valor
final> do
    <instrução>;
```

or

```
For <contador do loop>:= <valor inicial> DownTo <valor
final> do
    <instrução>;
```

Em que:

<*contador do loop*>: variável do tipo Integer, aceitando um valor com inicial a final com um passo de +1 no caso de ***To*** e -1 no caso de ***DownTo***. O valor do contador do loop muda automaticamente

<*valor inicial*>: qualquer expressão Integer, o valor inicial do contador do loop

<*valor final*>: qualquer expressão Integer, o valor final do contador do loop

<*instrução*>: a instrução é executada até o contador do loop não ser maior que (no caso de ***To)*** ou menor que (no caso de ***DownTo***) o valor final. Se você precisar executar várias instruções, é necessário usar um bloco ***begin...end*** após ***do.***

Os valores inicial e final são calculados somente uma vez na entrada do loop.

O loop ***FOR*** também é um loop de pré-condição.

Exercícios

Exercício 1.

Desenhe N linhas horizontais à mesma distância umas das outras. O número N é definido através da caixa de texto.

Exercício 2.

Desenhe N quadrados com o lado S localizado horizontalmente à mesma distância uns dos outros. Os números N e S são definidos usando caixas de texto.

Exercício 3.

Desenhe o campo quadriculado composto por N linhas e M colunas. Os números N e M são definidos usando caixas de texto.

Exercício 4.

Desenhe o tabuleiro de xadrez composto por N linhas e M colunas. Os números N e M são definidos usando caixas de texto.

Exercício 5.

Desenhe N círculos concêntricos. O raio do menor círculo é r e o raio do maior círculo é R. Os números N, r e R são definidos usando caixas de texto.

Exercício 6.

Desenhe o círculo composto por N pontos de raio R com as coordenadas definidas do centro Xc, Yc. Todos os parâmetros são inseridos em caixas de texto. As coordenadas de um ponto no círculo são definidas da seguinte maneira:

$$X = Xc + R\cos(\alpha) \qquad Y = Yc — R \operatorname{sen} \alpha$$

O ângulo α muda de 0 até 2π com o passo de 2π/n.

Exercício 7.

Dados três números representando os comprimentos dos segmentos, descubra se é possível criar um triângulo com esses segmentos. Se a resposta for sim, crie o triângulo, se for não, forneça uma mensagem de erro adequada.

Exercício 8.

Desenhe N quadrados aninhados, tendo definido um procedimento para desenhar os quadrados a partir das linhas.

Exercício 9.

Desenhe um relógio analógico simplificado. O horário é inserido em uma caixa de texto. Quando um botão é pressionado, os ponteiros mostram o horário.

Strings

Já temos um pouco de experiência com strings no Delphi. Por exemplo, as propriedades como ***Caption*** ou ***Text*** podem ter apenas valores de string. O que são strings exatamente e como você trabalha com elas?

Uma string é uma sequência de símbolos dentro de aspas simples. Para declarar uma variável de string, é necessário usar o tipo ***String***:

```
Var
    s: String;
```

Essa definição significa que o programa pode usar uma variável de string com um comprimento ilimitado.

As strings podem ser concatenadas. A concatenação de strings é denotada pelo sinal de mais. Por exemplo:

```
Var
    s, st: String;
Begin
    s:='We learn'; //We put "We learn" into first
variable
    st:' Delphi; //We put " Delphi" into second
variable
    s:=s+st; //Concatenate these two strings.
End;
```

Quando o programa for executado, obteremos a string "We learn Delphi" na variável *S*.

As strings podem ser comparadas.

A comparação é realizada símbolo por símbolo usando a relação de comparação de símbolos (comparando sua representação interna). Os caracteres latinos têm ordenação alfabética e dígitos têm uma ordenação natural em que 0 <1 <...<9.

Exemplo:

'AB' > 'AA' 'A' < 'AB' 'DC' > 'ABCDE' 'ABCE' > 'ABCD'

Vários procedimentos e funções padrão funcionam com strings. A tabela a seguir os resume.

Procedimentos e funções de string padrão

Procedimentos de string		
Nome e parâmetros	Tipos de parâmetro	Semântica
Delete(St, Pos, N)	St: string; Pos, N: integer;	Excluir os símbolos N a partir da string St, começando na posição Pos.
`Insert(St1, St2, Pos)`	`St1, St2: string;` Pos: integer;	Insira símbolos da string St1 na string St2 a partir da posição Pos.
Funções de string		
Nome e parâmetros	Tipos de parâmetro	Semântica
Copy(St, Poz, N)	Tipo de resultado: string; St: string; Pos, N: integer;	O resultado é a cópia de N símbolos da string St a partir da posição Pos.
Length(St)	Tipo de resultado: integer; St: string;	Calcula o comprimento (contagem de símbolo) da string St.
`Pos(St1, St2)`	`Tipo de resultado: integer; St1, St2: string;`	Procura a primeira ocorrência da substring St1 na string St. O resultado é a posição do primeiro símbolo da substring. Se a substring não foi encontrada, retorna 0.

Exemplos:

Procedimento **Delete**:

Valor de St	Instrução	Valor de St após a execução da instrução
'abcdef'	Delete(St,4,2)	'abcf'
'Turbo-Pascal'	Delete(St,1,6)	'Pascal'

Procedimento **Insert**:

Valor de St1	Valor de St2	Instrução	Valor de St2 após a execução da instrução
'Turbo'	'-Pascal'	Insert(St1, St2,1)	'Turbo-Pascal'
'-Pascal'	'Turbo'	Insert(St1, St2,6)	'Turbo-Pascal'

Função **Copy**:

Valor de St	Instrução	Valor de Str após a execução da instrução
'abcdefg'	Str:=Copy(St,2,3);	'bcd'
'abcdefg'	Str:=Copy(St,4,4);	'defg'

Função **Length**:

Valor de St	Instrução	Valor de N após a execução da instrução
'abcdefg'	N:=Length(St);	7
'Turbo-Pascal'	N:=Length(St);	12

Função **Pos**:

Valor de St2	Instrução	Valor de N após a execução da instrução
'abcdef'	N:=Pos('de', St2);	4
'abcdef'	N:=Pos('r', St2);	0

Exemplo estendido:

A entrada do procedimento é uma string que contém duas palavras delimitadas por um único símbolo de espaço.

Esse procedimento deve alterar a ordem das palavras.

Observação. O símbolo ‘␣’ significa espaço. Ele é usado para legibilidade.

```
Procedure Change(var s: String);
var
   s1: string;
begin
   s1:= copy(s,1, pos(‘␣’, s)-1);
   // copie a primeira palavra em s (palavra é qualquer
   // coisa antes de um espaço)
   delete(s,1, pos(‘␣’, s));
   // delete a primeira palavra (s continuará a conter
   // a segunda palavra)
   s:=s+’ ‘+s1;
   // adicione um espaço e a primeira palavra ao final
   // de s
 end;
```

Exercícios

Exercício 1.

Escreva um programa que, após pressionar um botão, troca a segunda e a terceira palavras do texto na caixa do texto. O conteúdo da caixa de texto deve conter exatamente três palavras, separadas por um espaço.

Exercício 2.

Quando o botão for pressionado, troque todos os espaços em uma caixa de texto por pontos de exclamação.

Exercício 3.

Conte o número de períodos na string da caixa de texto.

Exercício 4.

Conte o número de ocorrências da string "abc" no conteúdo da caixa de texto.

Exercício 5.

Uma string com um parêntese de abertura e um de fechamento é inserido em uma caixa de texto. Emita uma string entre esses parâmetros em uma caixa de texto diferente.

Exercício 6.

Conte o número de palavras separadas por um espaço em uma string inserida em uma caixa de texto.

Exercício 7.

Substitua todas as ocorrências da palavra "dog" pela palavra "cat" na caixa de texto.

Exercício 8.

Reverta uma string em uma caixa de texto.

Exercício 9.

Conte o número de palavras separadas por espaço na caixa de texto. Troque a segunda palavra e a próxima a última.

Strings e conversão para e de tipos numéricos

O procedimento ***Val(st, X, code)*** transforma a string ***st*** em um valor inteiro ou real armazenado na variável ***X***. O tipo resultante depende do tipo de ***X***. O parâmetro de variável ***Code*** indica o sucesso ou erro.

Após o cálculo, se a transformação tiver sido bem-sucedida, a variável ***Code*** contém zero e o resultado é colocado em ***X***. Caso contrário, ele contém o índice da posição de um símbolo errado na string ***st*** e o valor de ***X*** permanece inalterado.

A string ***st*** pode conter espaços à esquerda ou à direita. Para valores reais, as partes inteiras e fracionárias são delimitadas por um ponto.

`Exemplo:`

Para verificar a exatidão de uma entrada de dados numéricos pelo teclado e sua transformação nas variáveis ***X*** e ***Y***, usaremos o procedimento ***Val***.

As variáveis ***cx*** e ***y*** será definido como zero, se as strings ***sx*** e ***sy*** contêm representações de números válidas. Caso contrário, elas terão valores diferentes de zero.

```
Val (sx, x, cx);
Val (sy, y, cy);
If (cx=0) and (cy=0) Then
   begin
        <instruções>
   end
Else
   begin
        <instrução>
   end
```

O procedimento ***Str(X [: Width [: Decimal]], st)*** transforma o número inteiro ou real ***X*** em uma string ***st***. Os parâmetros ***Width*** e ***Decimal*** controlam a transformação.

O parâmetro ***Width*** define o comprimento completo da representação de string resultante de ***X***. O parâmetro ***Decimal*** define o número de símbolos para a parte fracionária do número e é aplicável apenas aos valores reais.

Exemplo:

```
var
   x: integer;
   y: real;
   s: string;
begin
y:=3.5;
Str(y, s);
... .;
... .
end;
```

A variável ***s*** conterá **3.500000000000000E+0000**.

No entanto, se aplicarmos uma transformação como essa ***str(y:6:2, s);***, o resultado será ںں**3.50**. Os símbolos à esquerda (zero) ausentes da parte inteira são substituídos por espaços (denotada pelo sinal ں).

Outro exemplo é:

```
Str(12345.6:5:2, s).
```

Nesse caso, a parte inteira é emitida completamente porque a restrição não pode ser satisfeita.

```
O resultado é 12345.6.
```

Um outro exemplo é:

```
Str(12.3456:5:2, s).
```

Nesse caso, a parte fracionária será arredondada para o número especificado de casas decimais.

O resultado é **12.35.**

Exercícios

Exercício 1.

Uma caixa de texto contém uma string. Exclua todos os dígitos da string e mostre o resultado em um rótulo separado.

Exercício 2.

Uma caixa de texto contém uma string. Calcule a soma de todos os dígitos na string.

Exercício 3.

Uma caixa de texto contém uma string com palavras e números, que são separados por um único espaço. Calcule a soma de todos os números.

Exercício 4.

Uma caixa de texto contém uma string na forma de **number1+number2=**. Coloque a soma de **number1** e **number2** após a string igual.

Exemplo:

```
Input string: 12.35+7.123=
Resulting string: 12.35+7.123=19.473
```

Exercício 5.

Uma caixa de texto contém uma string com um parêntese de abertura e um de saída. Há vários dígitos entre os parênteses. Calcule a média dos dígitos. Substitua os dígitos dentro dos parênteses pelo resultado calculado.

Exercício 6.

Duas caixas de texto contêm strings na forma de **Nome espaço Número**, em que **Número** é a altura de um homem em centímetros. Emita o nome do homem mais alto. Se as alturas forem iguais, emita os dois nomes.

Exercício 7.

Uma caixa de texto contém um número. Emita esse número em binário.

Exercício 8.

Uma caixa de texto contém um número. Emita esse número em hexadecimal.

Controle TMemo

TMemo é uma caixa de edição que gerencia várias linhas de texto.

É possível criar o conteúdo de ***TMemo*** dinamicamente usando suas propriedades e métodos.

As propriedades ***Font*** e ***Readonly*** de ***TMemo*** são iguais às propriedades de ***TEdit***. A propriedade de texto tem todo o texto de ***TMemo***, mas essa propriedade está disponível apenas quando o aplicativo está em execução.

Propriedade Lines

O texto do controle ***TMemo*** é armazenado na propriedade ***Lines*** e representa o conjunto numerado de linhas (a numeração começa em 0), a primeira linha tem um índice 0, a segunda 1 etc.

Exemplo:

```
Var
    s: string;
Begin
    s:= memEx.Lines[2];
    //o valor da segunda linha é atribuido a variável s
    memEx.Lines[3]:='Hi';
    //string constant 'Hi'is assigned to the 4rd line
end;
```

Um programa pode acessar as linhas do controle ***TMemo*** dessa forma apenas se as linhas tiverem sido criadas pelo Object Inspector, inseridas pelo teclado durante a execução do programa ou criadas pelo método correspondente. Uma tentativa de acessar uma linha inexistente causará um erro de tempo de execução.

Você pode preencher o ***TMemo*** usando a propriedade ***Lines***. Para fazer isso, pressione o botão com as reticências na propriedade ***Lines*** no Object Inspector. A janela do editor ***Lines*** abrirá. Você pode inserir as linhas na janela pressionando **<*Enter*>** no final de cada linha.

Ao terminar, pressione o botão **<*OK*>**.

Propriedade Count

A propriedade ***Count*** retorna o número total de linhas.

```
k:=memEx.Lines.Count;
```

Count é ***ReadOnly***, isto é, não é possível alterá-la/editá-la.

Propriedade WordWrap

Se ***WordWrap*** for True, o controle de edição quebra o texto na margem direita para que ele caiba na área do cliente.

Propriedade MaxLength

MaxLength define o número máximo de caracteres que pode ser inserido no controle de edição. Se o ***MaxLength*** for 0, o ***TMemo*** não limita o comprimento do texto inserido.

Propriedade ScrollBars

A propriedade ***ScrollBars*** determina se o controle ***TMemo*** tem barras de rolagem. O ***TMemo*** ignorará a propriedade ***MaxLength*** se ela for vertical ou se as duas barras de rolagem estiverem ativadas.

Propriedade Alignment

A propriedade ***Alignment*** altera a maneira como o texto é formatado no controle de edição de texto.

Métodos

Vamos examinar alguns métodos.

Método Delete

Este método exclui a linha com o índice especificado. Outras linhas serão automaticamente movidas e renumeradas.

```
memEx.Lines.Delete(0);
//deleta a primeira linha, p.e. linha com index 0
```

Exemplo:

Deixar apenas as linhas pares.

```
k:=memEx.Lines.Count;
For i:=k-1 Downto 1 Do
   if (i mod 2) <>0 then
        memEx.Lines.Delete(i);
```

Método Exchange

O ***Exchange*** troca as posições de duas linhas com os índices especificados.

```
memEx.Lines.Exchange(0,1);
// troca a linha 0 com a linha 1
```

Método Move

Move uma linha para uma nova posição.

Exemplo:

```
memEx.Lines.Move(1,5);
```

Durante a execução deste método, a linha com o índice 1 (isto é, a segunda linha, uma vez que a numeração começa em 0) será removida do texto. Todas as linhas após ela serão movidas uma posição para cima e a quinta linha substituirá a linha.

Exercícios

Exercício 1.

Preencha um ***TMemo***. Escreva um programa que, ao clicar no botão, deixe apenas o seguinte no ***TMemo***:

- As quatro primeiras linhas, se houver mais de quatro linhas.
- A primeira linha, se houver menos de quatro linhas.

Exercício 2.

Preencha um ***TMemo*** e edite o texto.

Escreva um programa que, ao clicar no botão, emita para o rótulo:

a) O índice da linha no ***TMemo***, que é igual à linha na edição do texto, ou a mensagem de que não existe tal linha.

b) Os índices das linhas que contêm uma combinação de símbolos que foram inseridos na edição do texto ou a mensagem de que não existem tais linhas.

Exercício 3.

Preencha um ***TMemo*** com o texto de algumas linhas. Uma linha deve conter "cat" e outra deve conter "dog". Escreva um programa para trocá-los.

Exercício 4.

Preencha um ***TMemo***.

Escreva um programa para encontrar a linha mais longa e colocá-la no topo.

Exercício 5.

Preencha um ***TMemo*** com números, um por linha.

Escreva um programa que duplique todos os números pares e os move para o segundo controle ***TMemo***. Divida todos os números ímpares por dois e mova-os para o terceiro ***TMemo***.

Controle TMemo (continuação)

Vamos examinar mais alguns métodos e propriedades úteis.

Clear

Limpa completamente o conteúdo ***TMemo***.

```
memEx.Lines.Clear
```

Métodos Append e Add

Se o ***TMemo*** estiver vazio ou ***Clear*** tiver sido chamado, é possível preencher um ***TMemo*** chamando o método ***Append.*** O método ***Append*** adiciona uma nova linha ao final de um ***TMemo***.

Exemplo:

Preencha um ***TMemo*** com números de 1 a 10, um número por linha.

```
...
memEx.Lines.Clear;
For i:=1 to 10 do
begin
   memEx.Lines.Append (IntToStr(i));
end;
```

O método ***Add*** funciona como o ***Append***.

```
memEx.Lines.Add ('Esse exemplo usa a Lista de' +
                 'string A.');
```

Como você pode ver, ambos os métodos usam string como parâmetro.

Add retorna um índice da nova string**.**

```
b:=memEx1.Lines.Add(edtEx1.Text);
lblEx1.Caption:=IntToStr(b);
```

Neste exemplo, o texto da caixa de texto edtEx1 foi adicionado ao ***TMemo*** chamado memEx1. O número da nova linha foi emitido para o rótulo lblEx1**.**

Você pode remover, salvar e recuperar fragmentos de texto do ***TMemo*** usando ***Clipboard.***

Método Insert

Ele insere uma linha com um índice especificado. Outras linhas serão movidas automaticamente.

```
memEx.Lines.Insert(2,'');
// adiciona uma linha vazia ao lugar da linha com index
//2, que é movida e não deletada
```

Classificação de linhas no TMemo

A classificação de linhas pode ser realizada usando o seguinte algoritmo.

Encontre o índice de uma linha com o valor mínimo (as strings são comparadas caractere por caractere A<B<C...<Z etc.)

A primeira linha e a linha mínima são trocadas.

Repita essas instruções (N-1) vezes, em que N é o número de linhas no ***TMemo***.

Vamos escrever esse programa.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
   begin
   //para toda i-linha de TMemo
   //encontra o index de linha mínima começando de i
   //muda essa linha com a linha atual
end.
```

Descrevemos as variáveis necessárias, os operadores e a função de encontrar a linha mínima.

Como resultado, obteremos o seguinte programa:

```
function PlMin(numStr: integer): integer; var
   k, m: integer;
begin
   m:=NumStr;
   //Deixa a linha na posição mínima num_str
   for k:=m+1 to frmEx1.memEx1.Lines.Count do
   //Para todas as linhas começando da posição num_str
   if frmEx1.memEx1.Lines[k]< frmEx1.memEx1.Lines[m]
   then
   //Se a linha presente for menor que o mínimo, então
   m:=k;//mude a posição mínima
   PlMin:=m;//Assinale a posição para a linha mínima
end;
```

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   i, j: integer;
begin
   for i:=0 to memEx1.Lines.Count do
   //Para toda a linha de TMemo
   begin
   j:=plmin(i);
   //encontre a posição mínima da linha começando em i
   memEx1.Lines.Exchange(i, j);//Faça a troca
end;
end;
end.
```

Exercícios

Exercício 1.

Preencha um ***TMemo*** com números, um número por linha.

Escreva um programa para colocar todos os números positivos no segundo ***TMemo*** e todos os números negativos em um terceiro ***TMemo.***

Exercício 2.

Preencha um ***TMemo*** com uma lista de nome de classe. Escreva um programa para classificar a lista.

Exercício 3.

Preencha dois ***TMemo***s.

Escreva um programa que, ao clicar no botão, emita as linhas idênticas do primeiro e do segundo ***TMemo***s para o terceiro ***TMemo***, caso contrário, emita a mensagem de que não há linhas idênticas.

Exercício 4.

Preencha um ***TMemo*** com números, um número em cada linha.

Escreva um programa que classifique os números.

Números, constantes, tipos criados pelo usuário e matrizes aleatórias

Função RANDOM

Podemos usar a função ***Random*** para obter números aleatórios distribuídos uniformemente. Essa função pode ser chamada de duas maneiras:

- Chamar ***Random*** sem nenhum parâmetro retornará um número **real** aleatório no intervalo de 0 (inclusive) a 1 (não inclusivo);
- Chamar ***Random(N)*** com um parâmetro inteiro ***N*** retornará um número inteiro aleatório no intervalo de 0 a ***N***-1.

Se você precisar obter um número aleatório no intervalo de *a* a *b* (ambos inclusivos), você precisa usar a seguinte fórmula:

```
Random(b-a+1)+a.
```

Se você precisar obter um número real aleatório no intervalo de *a* (inclusivo) a *b* (não inclusivo), você pode usar a seguinte fórmula:

```
(b - a) * Random + a.
```

Cada execução do programa produzirá a mesma sequência de números aleatórios.

Você chamará o procedimento Randomize para obter diferentes conjuntos de números aleatórios em seu programa. Para garantir que você não reinicie acidentalmente um gerador de números aleatórios (possivelmente) do mesmo ponto, organize seu programa de uma maneira que Randomize seja chamado apenas antes de quaisquer chamadas para ***Random***.

Constantes nomeadas

Constantes são entidades que não podem alterar seus valores durante uma execução do programa. Exatamente como as variáveis, as constantes têm tipos. Elas podem ser anônimas ou nomeadas.

Exemplos de constantes anônimas:

100	—	Integer
2.7	—	Real
7.12457e-03	—	Real
TRUE	—	Boolean
14.0	—	Real
-37	—	Integer
‘asderc’	—	String

É muito útil ter constantes nomeadas. Constantes são declaradas na parte de declaração de um programa, juntamente com seu valor.

Sintaxe da definição da constante nomeada:

```
Const
    <nome da constante> = <valor>;
```

As regras controlando nomes de constantes são as mesmas para os nomes de variáveis. O tipo da constante nomeada é derivado automaticamente de seu valor.

As constantes nomeadas normalmente são usadas para aprimorar a legibilidade e programar a manutenção, bem como para denotar os parâmetros críticos.

Tipos definidos pelo usuário

O Delphi nos permite definir nossos próprios tipos.

A seguir está um exemplo de definição de tipo do usuário e seu uso:

```
type
   kg = integer;
   cm = integer;
var
   Weight: kg;
   Size: cm;
```

Os tipos do usuário melhoram a legibilidade do programa. A manutenção do programa também melhora, sempre podemos alterar a definição de um tipo (por exemplo, de ***Integer*** para ***Real***) se, durante o desenvolvimento do programa, descobrirmos uma inadequação na definição de tipo atual. A alteração precisa acontecer apenas em um lugar, não em todas as definições de todas as variáveis de um tipo inadequado.

Exercícios

Exercício 1.

Vamos jogar um jogo: "Adivinhe o número".

Deixe o computador escolher um número aleatório de um intervalo especificado por duas caixas de texto. A terceira caixa de texto será usada para inserir nossos palpites.

Escreva um programa que mostre o resultado de um convidado após o pressionamento de um botão. O resultado pode ser uma confirmação do palpite correto ou uma comparação entre o número selecionado pelo computador e nosso palpite (maior ou menor).

Adicione um botão que permite que os usuários olhem rapidamente o número escolhido pelo computador.

Matriz estática unidimensional

Uma ***matriz*** é uma coleção de elementos (valores ou variáveis do mesmo tipo), que são identificados por um índice de matriz ou chave com um nome geral. Por exemplo, uma lista de classes. Nessa matriz, o nome de cada elemento é "Student" e cada nome de estudante tem uma posição na lista (índice). Nesse caso, o nome é o valor do elemento.

Formato da matriz:

```
<nome da matriz>: Array [<índice inicial>..<índice final>] Of <tipo de elemento>;
```

O formato da matriz inclui:

Nome da matriz.

Palavra-chave: ***Array***.

O valor inicial e o final do índice: eles definem o intervalo do índice, por exemplo, 1..40, —2..2, 0..10. O limite inferior é o mínimo valor possível de um índice e o limite superior mostra o maior valor. O limite inferior não pode ser maior que o limite superior. Eles são separados por dois pontos. O tipo de índice deve ser uma enumeração, isto é, um tipo com o elemento anterior e o próximo elemento predefinidos. Agora conhecemos os seguintes tipos de enumeração: todos os tipos inteiros e o tipo Boolean. Somente constantes podem ser usadas como um índice, isto é, o tamanho de uma matriz é determinado durante a escrita do programa e não pode ser alterado durante o tempo de execução.

Tipo do elemento.

A seguir está um exemplo de matriz:

```
A: array[1..10]of real;
```

A matriz A é composta por 10 elementos e cada elemento é um número real.

É possível acessar um elemento de matriz por um índice. Um índice pode ser uma expressão do tipo adequado.

Por exemplo:

```
X:=5;
A[x+2]:=4.5
```

Aqui, o valor de 4.5 é atribuído ao elemento da matriz com um índice 7 (5+2).

O acesso por um índice permite que as tarefas sejam resolvidas de forma mais eficiente. Por exemplo, para atribuir 0 a todos os elementos da matriz, é suficiente escrever o loop:

```
For i:=1 To 10 do
A[i]:=0;
```

Para usar uma matriz como um parâmetro de um procedimento, seu próprio tipo deve descrever a matriz, por exemplo:

```
Type
MyArray = array[1..30] of integer;
.....
.....
Var
A: MyArray;
```

Vamos examinar a tarefa de preencher uma matriz de elementos N. Os valores de elementos de matriz devem estar no intervalo especificado. Emita os valores de matriz para um ***TMemo*** e encontre o elemento máximo.

Escreva os algoritmos nos comentários.

```
Begin
//Defina o tamanho do intervalo do array
//Preencha o array com números na posição específica
//Coloque o array no TMemo
//Encotnre o maior elemento Amax
//Mostre Amax
End.
```

Vamos escrever uma instrução para cada frase:

```
Begin
   SetArrayRange(rMin, rMax);
   //Defina o tamanho dos intervalo do array
   FillArray(a, rMin, rMax);
   //Preencha o array com números do intervalo
   OutputArray(a, n);
   //Coloque o array no TMemo
   Amax:=max(a); //Encontre o maior elemento Amax
   lblAmax.Caption:=IntToStr(Amax); //Mostre Amax
End.
```

Antes de escrevermos esses procedimentos e funções, observe que uma matriz deve ser definida como um parâmetro do procedimento, descrita por seu próprio tipo.

```
Const
   n=20;
Type
   array_n_elements=array[1..n] of integer;
procedure SetArrayRange(var ch1, ch2: integer);
begin
   ch1:=StrToInt(frmArr.edtCh1.Text);
   ch2:=StrToInt(frmArr.edtCh2.Text);
end;
```

```
  procedure FillArray(var a: array_n_elements; rMin,
rMax: integer);
  var
     i: integer;
  begin
     randomize;
     For i:=1 to n do
     a[i]:=random(rMax-rMin)+rMin;
  end;
  procedure OutputArray(a: array_n_elements; n:
integer);
  var
     i: integer;
  begin
     frmArr.mem_ish.Lines.Clear;
     for i:=0 to n-1 do
     frmArr.memIsh.Lines.Append(IntToStr(a[i+1]));
  end;
  function max(a: array_n_elements): integer;
  var
     i, m: integer;
  begin
     m:=a[1];
     for i:=1 to n do
     if a[i]>m then
          m:=a[i];
          max:=m;
     end;
```

```
procedure TfrmArr.btnClick(Sender: TObject);
var
   rMin, rMax: integer;
   Amax: integer;
   A: array_n_elements;
begin
   SetArrayRange(rMin, rMax);
   //Defina o intervalo dos valores do array
   FillArray(a, rMin, rMax);
   //Preencha o array com números do intervalo
   OutputArray(a, n); //Mostre o array
   Amax:=max(a); //Encontre o elemento máximo Amax
   lblAmax.Caption:=IntToStr(Amax); //Mostre Amax
end;
end.
```

Exercícios

Exercício 1.

Preencha uma matriz com números aleatórios do intervalo. Insira os limites do intervalo por meio de caixas de texto.

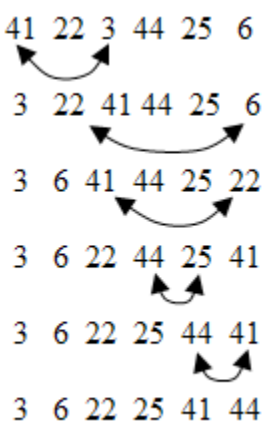
- Emita a matriz para um ***TMemo***.
- Para os rótulos, conte e emita:
- A soma dos elementos.
- A média dos elementos.
- A quantidade de elementos positivos e negativos.
- Os elementos máximo e mínimo.
- Divida a matriz em duas matrizes, uma com elementos positivos e outra com os elementos negativos. Emita as matrizes para dois novos ***TMemo***s.

Classificação da matriz e classificação da seleção

O processo de classificação pode ser definido como um processo de ordenação de um conjunto de objetos comparando seus atributos. A classificação da matriz é uma área bem conhecida em ciência da computação e há muitos algoritmos que são usados para classificar as matrizes. Examinaremos um dos algoritmos de classificação de matriz mais simples e compreensíveis, a classificação de seleção.

Um algoritmo encontra o menor elemento em uma matriz de elementos N e o troca pelo primeiro elemento na matriz. Em seguida, um algoritmo encontra o menor elemento nos elementos restantes (N-1) e o troca pelo segundo elemento na matriz. A mesma operação é realizada com elementos N-2, elementos N-3 e assim por diante, até haver apenas um elemento restante. Esse elemento é o maior.

Um exemplo de execução de algoritmo é mostrado abaixo. Os arcos abaixo do texto mostram as trocas de elementos:



Vamos examinar a classificação de uma matriz em ordem ascendente. Primeiro, escrevemos o algoritmo nos comentários.

```
//Insira o valor do intervalo, chB e chE.
//Preencha o array A com números no intervalo de chB
// até chE (inclusive).
//Coloque o array A no primeiro campo do memo.
//Organize o array A em ordem ascendente.
//Coloque o array A no segundo campo do memo.
```

Em seguida, escrevemos o algoritmo como uma sequência das chamadas de procedimento.

```
procedure TfrmArr.btnGoClick(Sender: TObject);
var
   chB, chE: integer;
   A: array_n_elements;
begin
   Input(chB, chE);
   //Insira o valor do intervalo do array, de from chB
   //para chE (inclusive).
   FillArr(a, chB, chE);
   //Preencha o array com números aleatórios no
  // intervalo de chB até chE
   OutputArray1(a, n);
   //Coloque o array A no TMemo 1.
   SortArr(a);
   //Organize o array A em ordem ascendente.
   OutputArray2(a, n); //Colque o array A no TMemo 2.
end;
```

Os três primeiros procedimentos foram discutidos no módulo anterior. Aqui, escreveremos apenas o procedimento para classificar a matriz. É quase igual ao procedimento para classificar um ***TMemo***.

```
procedure SortArr(var a: array_n_elements);
var
   i: integer;
begin
   for i:=1 to n-1 do //Para cada elemento no array
   change(a[i], a[NumMin(a, i)]);
   //troque o elemento atual pelo último elemento no
   // intervalo até a última posição de troca
end;
```

Antes desse procedimento, escreveremos um procedimento para trocar os dois elementos, dados seus índices e a função, para encontrar o menor elemento, começando na posição especificada:

```
function NumMin(a: array_n_elements; start:
integer): integer;
var
   i, m: integer;
begin
   m:=start;
   for i:=m+1 to n do
   if a[i]<a[m] then
   m:=i;
   NumMin:=m;
end;
procedure change(var one, two: integer);
var
   temp: integer;
begin
   temp:=one;
   one:=two;
   two:=temp;
end;
```

Exercícios

Exercício 1.

Insira um intervalo de valores pelo teclado e preencha a matriz com os valores inteiros do intervalo inserido.

Emita a matriz em um ***TMemo***.

Classifique a matriz em ordem descendente e emita-a em um segundo ***TMemo***.

Exercício 2.

Insira um intervalo de valores pelo teclado e preencha a matriz com os valores inteiros do intervalo inserido.

Emita a matriz em um ***TMemo***.

Coloque todos os elementos positivos no início da matriz e todos os elementos zero e negativos no final da matriz (**não use a classificação**). Emita a matriz em um segundo ***TMemo***.

Exercício 3.

Insira um intervalo de valores pelo teclado e preencha a matriz com os valores reais do intervalo inserido.

Emita a matriz em um ***TMemo***.

Troque a primeira metade da matriz pela segunda metade. Emita a matriz em um segundo ***TMemo***.

Controle StringGrid

StringGrid mostra e coleta dados da string no formato de grade.

StringGrid está na guia ***Additional***. Usamos o prefixo ***sgd*** para os nomes desse componente. Para nossos exemplos, usaremos o nome ***sgdMy***.

Uma grade pode ter células fixas. As células fixas precisam exibir cabeçalhos de colunas e linhas e para alterar manualmente seus tamanhos.

Normalmente, a coluna da esquerda e a linha superior são fixas. No entanto, usando as propriedades ***FixedCols*** e ***FixedRows***, é possível definir outros números de colunas e linhas fixas. Se essas propriedades forem definidas como 0, a grade não terá parte fixa.

De maneira geral:

```
sgdMy.FixedCols:=1; //Número fixo de colunas = 1
sgdMy.FixedRows:=1; //Número fixo de posições = 1
```

As células não fixas de uma grade podem conter qualquer quantidade de colunas e linhas e o número pode ser alterado programaticamente. Se uma grade for maior do que a janela do componente, as barras de rolagem serão adicionadas automaticamente. Quando a grade é rolada, as linhas e colunas fixas não são roladas para fora da visão, mas seu conteúdo muda (os cabeçalhos de linhas e colunas).

Cells é a principal propriedade de ***StringGrid***. A propriedade ***Cells*** é a coleção de células e todas as células podem ter texto. As células têm duas coordenadas: a coordenada da coluna da célula e a coordenada da linha da célula. A primeira linha é a linha zero e a primeira coluna é a coluna zero.

Cells tem o tipo ***String***.

As células podem ser preenchidas manualmente durante o tempo de execução. Para permitir isso, você deve pressionar + no Object Inspector para abrir a propriedade ***Options*** e definir ***True*** para a propriedade ***goEditing***.

O valor da célula pode ser definido programaticamente usando um operador de atribuição. Use os índices para acessar a célula. Lembre-se de que **o primeiro índice é o número da coluna e o segundo ícone é o número da linha.**

Aqui está um exemplo:

```
sgdMy.Cells [1,1]:= 'A célula superior a esquerda não'
+ ' fixa';
sgdMy.Cells [0,0]:= 'Números: ';
```

Os números de ***String*** serão definidos para a primeira célula da grade, isto é, para a primeira célula da parte fixa.

Os valores das propriedades ***ColCount*** e ***RowCount*** definem o tamanho da grade.

ColCount e ***RowCount*** podem ser alterados durante a criação do programa e o tempo de execução. No entanto, seus valores devem ser maiores em pelo menos mais do que um dos valores correspondentes de ***FixedCols*** e ***FixedRows***.

Vamos definir a contagem de colunas como três e a contagem de linhas como cinco em sgdMy.

```
sgdMy.ColCount:=3;
sgdMy.RowCount:=5;
```

A propriedade ***FixedColor*** define a cor das células fixas e a propriedade ***Color*** define a cor das outras células.

Vamos escrever um programa como exemplo. Ao clicar no primeiro botão, o ***StringGrid*** é criado. O número de colunas, linhas, colunas fixas e linhas fixas é inserido em quatro caixas de texto. Ao clicar no segundo botão, as células fixas são pintadas na cor verde. Ao clicar no terceiro botão, as outras células são pintadas na cor vermelha.

```
  Procedure GetGridParam(var n1, n2, n3, n4:
integer);
  Begin
     n1:=StrToInt(frmGrid.edtLine.Text);
     n2:=StrToInt(frmGrid.edtStolb.Text);
     n3:=StrToInt(frmGrid.edtFline.Text);
     n4:=StrToInt(frmGrid.edtFstolb.Text);
  End;
  Procedure CreateGrid(n1, n2, n3, n4: integer);
  Begin
     frmGrid.sgdMy.RowCount:=n2;
     frmGrid.sgdMy.ColCount:=n1;
     frmGrid.sgdMy.FixedCols:=n4;
     frmGrid.sgdMy.FixedRows:=n3;
  end;
  procedure TfrmGrid.btnTablClick(Sender: TObject);
  var
     nl, ns, nfl, nfs: integer;
  begin
     GetGridParam(nl, ns, nfl, nfs);
     //Insire os parametros do grid
     CreateGrid(nl, ns, nfl, nfs);
     //Cria com os parâmetros definidos
     end;
  procedure TfrmGrid.btnCelRedClick(Sender: TObject);
  begin
     frmGrid.sgdMy.Color:=clRed;
  end;
```

```
procedure TfrmGrid.btnFCGreenClick(Sender: TObject);
begin
   frmGrid.sgdMy.FixedColor:=clGreen;
end;
```

Algumas propriedades úteis de StringGrid

Propriedade	Descrição
BorderStyle: TBorderStyle;	Especifica a aparência da borda da grade: bsNone — sem borda, bsSingle — borda de 1 pixel.
ColCount: Longlnt;	Contém o número de colunas da grade.
DefaultColWidth: Integer;	Contém a largura de coluna padrão.
RowCount: Longlnt;	Contém o número de linhas da grade.
DefaultRowHeight: Integer;	Contém a altura de linha padrão.
Color: TColor;	Controla a cor das células.
FixedCols: Integer;	Controla o número de colunas fixas.
FixedRows: Integer;	Controla o número de colunas fixas.
FixedColor: TColor;	Controla a cor das células fixas.
GridHeight: Integer;	Contém a altura da grade.
GridWidth: Integer;	Contém a largura da grade.
GridLineWidth:	Especifica a largura das linhas da grade.

Exercícios

Exercício 1.

Crie um ***StringGrid***.

Nas caixas de texto, defina:

- O número de linhas.
- O número de colunas.
- O número de linhas fixas.
- O número de colunas fixas.

Crie três botões para alterar a cor das células fixas (um para uma cor).

Crie três botões para alterar a cor das células não fixas (um para uma cor).

Crie uma caixa de texto e dois botões.

Após clicar no primeiro botão, o ***TMemo*** mostra a coluna com um índice, que é especificado na caixa de texto. Após clicar no segundo botão, o ***TMemo*** mostra a linha com um índice, que é especificado na caixa de texto.

Exercício 2.

Crie um ***StringGrid***.

Os alunos estão realizando experimentos em um laboratório de física. Os dados a seguir são fornecidos: o número de experimentos, o intervalo e os valores da escala, que são medidos nos intervalos de escala.

Insira o número de experimentos (5 a 10) e os intervalos de escala nas caixas de texto.

Os valores medidos são adicionados à tabela pelo teclado. Para todos os experimentos, adicione os valores à tabela e calcule os valores mínimo e máximo.

Por exemplo, o número de experimentos = 5 e os intervalos de escala = 10. Então, a tabela terá a seguinte aparência:

Número de experimentos	Intervalos de escala	Valores medidos	Resultado	Máx./mín.
1	10	83	830	Máx.
2	10	51	510	
3	10	67	670	
4	10	49	490	Mín.
5	10	75	750	

Prática StringGrid

Encontrar os números de elementos com um atributo específico.

Os problemas de encontrar o número de elementos de grade específicos são resolvidos praticamente da mesma forma para um ***TMemo*** ou uma matriz unidimensional. Você precisa apenas adicionar o segundo loop para o segundo índice.

Exercício 1.

Encontre o número de elementos negativos em cada linha da matriz inteira.

Solução

Preencha um ***StringGrid*** usando o gerador de números aleatórios. Lembre-se de que a numeração começa em 0. O resultado será emitido para o ***TMemo***.

Vamos escrever um programa ***Delphi***.

Colocaremos duas caixas de texto no form para definir o número de linhas e colunas, ***StringGrid*** e ***TMemo***, bem como dois botões:

1: Cria e preenche o ***StringGrid***.

2: Calcula o número de elementos negativos em cada linha.

Vamos escrever dois procedimentos para manipular o evento de clique dos botões.

```
  procedure TForm1.Button1Click(Sender: TObject);
  var
     i, j, n, m: integer;
  begin
     n:=strtoint(edit1.Text);
     m:=strtoint(edit2.Text);
     stringgrid1.RowCount:=n;
     stringgrid1.ColCount:=m;
     for i:=0 to n-1 do
     for j:=0 to m-1 do
     stringgrid1.Cells[j,
i]:=inttostr(random(100)-50);
  end;
  procedure TForm1.Button2Click(Sender: TObject);
  var
     i, j, n, m, k: integer;
  begin
     n:=stringgrid1.RowCount;
     m:=stringgrid1.ColCount;
     for i:=0 to n-1 do
  begin
     k:=0;
     for j:=0 to m-1 do
     if strtoint(stringgrid1.Cells[j, i])<0 then
     k:=k+1;
     memo1.Lines.Append(inttostr(k));
  end
  end;
```

Exercício 2.

Substitua todos os elementos negativos de uma matriz pelos opostos.

Possíveis soluções:

- Multiplicar o elemento negativo por (–1).
- Encontrar os valores absolutos dos elementos negativos.
- Colocar um sinal de menos antes do elemento negativo.

Programa 1

```
procedure TForm1.Button2Click(Sender: TObject);
var
   i, j, n, m: integer;
begin
   n:=stringgrid1.RowCount;
   m:=stringgrid1.ColCount;
   for i:=0 to n-1 do
begin
   for j:=0 to m-1 do
   if strtoint(stringgrid1.Cells[j, i])<0 then
        stringgrid1.Cells[j, i]:=inttostr((–
1)*strtoint(stringgrid1.Cells[j, i])); end
end;
```

Programa 2

```
procedure TForm1.Button2Click(Sender: TObject);
var
   i, j, n, m: integer;
begin
   n:=stringgrid1.RowCount;
   m:=stringgrid1.ColCount;
   for i:=0 to n-1 do
begin
   for j:=0 to m-1 do
   stringgrid1.Cells[j, i]:=inttostr(abs(strtoint
(s tringgrid1.Cells[j, i])));
end
end;
```

Programa 3

```
procedure TForm1.Button2Click(Sender: TObject);
var
   i, j, n, m: integer;
begin
   n:=stringgrid1.RowCount;
   m:=stringgrid1.ColCount;
   for i:=0 to n-1 do
begin
   for j:=0 to m-1 do
   if strtoint(stringgrid1.Cells[j, i])<0 then
   stringgrid1.Cells[j, i]:=inttostr(-
strtoint(stringgrid1.Cells[j, i]));
end
end;
```

Exercício *3.*

Descubra se uma matriz quadrada é simétrica ou não, em relação à diagonal principal.

Solução

Usar a regra: a matriz será simétrica se para cada ***i* = *1, 2, …, n*** e ***j* = *1, 2,…, n,*** em que ***i>j, StringGrid1.Cells[i, j]= StringGrid1.Cells [j, i]***. Portanto, podemos escrever o programa:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i, j, n: integer;
begin
   n:=strtoint(edit1.Text);
   stringgrid1.RowCount:=n;
   stringgrid1.ColCount:=n;
   for i:=0 to n-1 do
   for j:=0 to n-1 do
   stringgrid1.Cells[j, i]:=inttostr(random(2));
   label2.Caption:='';
end;
```

```
  procedure TForm1.Button2Click(Sender: TObject);
  var
     i, j, n, m: integer;
     simm: boolean;
  begin
     n:=stringgrid1.RowCount;
     simm:= True;
     {Assume that the matrix is symmetric}
     i:=2;
     While simm and (i < n) Do
     Begin
          j:=1;
          While (j < i) and (stringgrid1.Cells[j, i]
= stringgrid1.Cells[i, j]) Do
          j:=j+1;
          simm:=(j=i);
          i:=i+1
  End;
  if simm then
     label2.Caption:='Matrix is symmetric'
  else
     label2.Caption:='Matrix is not symmetric'
  end;
```

Exercício 4.

Preencha a matriz n x m de uma maneira especial:

```
12345 … n
… n+2 n+1
```

Solução

Para preencher a matriz dessa forma, precisamos definir a regra de preenchimento. Nesse caso, a regra será:

Se a linha for ímpar, então ***stringgrid1.Cells[j, i]:= inttostr(i*m + j+1)***, caso contrário, (se a linha for par) ***stringgrid1.Cells[j, i]:= inttostr((i+1)*m-j).***

Escreveremos um programa de acordo com essa regra:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   i, j, n, m: integer;
begin
   n:=strtoint(edit1.Text);
   m:=strtoint(edit2.Text);
   stringgrid1.RowCount:=n;
   stringgrid1.ColCount:=m;
   for i:=0 to n-1 do
   for j:=0 to m-1 do
   if i mod 2 =0 then
   stringgrid1.Cells[j, i]:= inttostr(i*m + j+1)
else
   stringgrid1.Cells[j, i]:= inttostr((i+1)*m-j)
end;
```

Exercícios

Exercício 1.

Preencha a matriz n x m com 0 e 1 de acordo com o esquema a seguir (por exemplo, para uma matriz 5 x 5):

1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0

Exercício 2.

Preencha a matriz n x m de acordo com o esquema a seguir (por exemplo, para uma matriz 5 x 5):

0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 1 1 1 0
1 1 1 1 1

Exercício 3.

Preencha a matriz m x n com valores aleatórios do intervalo. Os valores mais alto e mais baixo do intervalo serão inseridos pelo teclado.

Exercício 4.

Preencha a matriz m x n com valores aleatórios do intervalo. Os valores mais alto e mais baixo do intervalo serão inseridos pelo teclado.

Encontre o menor elemento em cada linha e troque-o pelo elemento da diagonal principal.

Exercício 5.

Preencha a matriz m x n com valores aleatórios do intervalo. Os valores mais alto e mais baixo do intervalo serão inseridos pelo teclado.

Para cada coluna, encontre e emita o produto dos elementos das linhas ímpares.

Exercício 6.

Preencha a matriz m x n com valores aleatórios do intervalo. Os valores mais alto e mais baixo do intervalo serão inseridos pelo teclado.

Encontre e emita o produto dos elementos positivos das colunas ímpares.

Matrizes bidimensionais

Como já sabemos, uma matriz é uma coleção de elementos do mesmo tipo ordenada, com um nome e diferentes índices. Em qualquer matriz, seus elementos ficam próximos uns dos outros (na memória).

Primeiro elemento	Segundo elemento	Terceiro elemento	...	Nº elemento

Usamos uma matriz unidimensional quando trabalhamos com sequências de dados simples, sejam números, strings ou outros tipos.

Deve-se observar que todas as matrizes podem ser vistas, essencialmente, como unidimensionais. Por conveniência, as matrizes bi e multidimensionais são necessárias na manipulação e representação de dados.

As matrizes bidimensionais têm dois índices e as matrizes N-dimensionais têm N índices.

É conveniente usar uma matriz bidimensional no processo de resolução de problemas que envolvem dados tabulares.

Por exemplo, os dados de entrada devem ser representados e manipulados como uma tabela com três colunas e cinco linhas.

Coluna um					Coluna dois					Coluna três				
el 1	el 2	el 3	el 4	el 5	el 6	el 7	el 8	el 9	el 10	el 11	el 12	el 13	el 14	el 15

Elementos da matriz

Aqui, temos uma matriz composta por três colunas e cada uma delas composta por cinco linhas. Podemos dizer que temos uma matriz de matrizes.

Esse tipo de matriz será colocado em um ***StringGrid*** da seguinte forma:

	Coluna um	Coluna dois	Coluna três
Linha um	Elemento 1	Elemento 6	Elemento 11
Linha dois	Elemento 2	Elemento 7	Elemento 12
Linha três	Elemento 3	Elemento 8	Elemento 13
Linha quatro	Elemento 4	Elemento 9	Elemento 14
Linha cinco	Elemento 5	Elemento 10	Elemento 15

Vamos declarar essa matriz.

Primeiro, vamos declarar a matriz da coluna.

```
Const
   m=3;
Type
   col_array=array[1..m] of row_array;
   Adicione a declaração da matriz de
   linhas.
Const
   m=3;
   n=5;
Type
   row_array=array[1..n] of integer;
   col_array=array[1..m] of row_array;
```

Vamos examinar outro exemplo, em que os dados de entrada devem ser representados como uma matriz com três linhas, cada uma com cinco colunas.

Linha um					Linha dois					Linha três				
el 1	el 2	el 3	el 4	el 5	el 6	el 7	el 8	el 9	el 10	el 11	el 12	el 13	el 14	el 15

Elementos da matriz.

Como podemos ver, temos uma matriz composta por três linhas, cada uma com cinco colunas. Novamente, temos uma matriz de matrizes.

Nesse caso, a matriz será colocada no ***StringGrid*** da seguinte forma:

	Coluna um	Coluna dois	Coluna três	Coluna quatro	Coluna cinco
Linha um	el 1	el 2	el 3	el 4	el 5
Linha dois	el 6	el 7	el 8	el 9	el 10
Linha três	el 11	el 12	el 13	el 14	el 15

Vamos definir o tipo de matriz.

Primeiro, vamos declarar a matriz de linhas.

```
Const
   n=3;
Type
   row_array=array[1..n] of col_array;
Em seguida, adicionamos a declaração da matriz de
colunas:
Const
   n=3;
   m=5;
Type
   col_array =array[1..m] of integer;
   row_array =array[1..n] of col_array;
```

Como você pode ver nesses exemplos, as matrizes bidimensionais (e N-dimensionais) podem ser representadas como uma matriz de matrizes. As matrizes bidirecionais podem ser uma matriz de linhas ou uma matriz de colunas.

Podemos tratar dos elementos da matriz A da seguinte forma:

A[1][4], A[3][5] ou A[I][j], em que I é um índice de uma matriz bidirecional e j é um índice de uma matriz unidimensional (um elemento de uma matriz externa bidimensional).

Vamos resolver um problema de exemplo.

Precisaremos preencher uma matriz bidimensional em M colunas de N elementos inteiros com números aleatórios no intervalo de -10 a 10, inclusive. Precisamos emitir essa matriz em um elemento de controle ***StringGrid***. O número de linhas e colunas de células fixas de ***StringGrid***s deve ser 0.

Primeiro, declaramos o tipo de matriz.

```
Const
   n=3;
   m=6;
Type
   array_1=array[1..n] of integer;
   array_2=array[1..m] of array_1;
```

Agora, podemos escrever a solução:

```
Var
   C: array_2;
Begin
   FillArray(c);
   //Preencha o array C
   TableOutput(c); //Coloque o C na tabela de texto
end;
```

Agora, podemos escrever os procedimentos. Começamos com o procedimento para preencher uma matriz de colunas.

```
procedure FillArray(var arr: array_2);
   var i: integer;
begin
Randomize;
   for i:=1 to m do //de i no intervalo de 1 até M
   FillColumn(arr[i]);
   //preencha a coluna i-th do array
end;
```

Agora, precisamos de um procedimento para preencher a coluna com números aleatórios no intervalo de -10 a 10.

```
procedure FillColumn(var col: array_1);
var
   j: integer;
begin
   for j:=1 to n do //de j no intervalo de 1 até n
   ma_1[j]:=Random(21)-10; //ao elemento j-th do array é
   // atribuido um valor aleatório no intervalo de –10
   // até +10 (inclusive)
end;
```

O procedimento para emitir a matriz resultante para o ***StringGrid*** é definida abaixo.

```
procedure TableOutput(arr: array_2);
var
   i, j: integer;
begin
   frm2arr.sgdMy.FixedCols:=0; //Número de colunas fixo
   frm2arr.sgdMy.FixedRows:=0; //Número de células fixo
   frm2arr.sgdMy.ColCount:=m; //Colunas
   frm2arr.sgdMy.RowCount:=n; //células
   for i:=1 to m do
   for j:=1 to n do
   frm2arr.sgdMy.Cells[i-1, j-1]:=IntToStr(arr[i][j]);
   end;
```

Vamos ver o que mudaria se emitíssemos a matriz linha por linha, como se a transpuséssemos. Nesse caso, teríamos três linhas, cada uma com seis colunas. Precisamos lembrar que as células do ***StringGrid*** são indexadas primeiro por uma coluna e depois por uma linha.

O procedimento para emitir uma matriz para o ***StringGrid*** será semelhante ao seguinte:

```
procedure TableOutput(arr: array_2);
var
   i, j: integer;
begin
   frm2arr.SgdMy.ColCount:=n;
   //n – números de células do array
   frm2arr.SgdMy.RowCount:=m;
   //m – número de colunas do array
   for i:=1 to n do
   for j:=1 to m do
   frm2arr.SgdMy.Cells[i-1, j-1]:=IntToStr(arr[j][i]);
end;
```

Exercícios

Exercício 1.

Preencha com números aleatórios uma matriz bidimensional, composta por M colunas e N linhas cada.

O intervalo deve ser inserido em caixas de texto separadas.

Emita a matriz em uma tabela ***StringGrid***. O número das colunas e linhas das células fixas deve ser definido como 0.

Os valores a seguir devem ser escritos no componente ***Memo***:

Soma de todos os elementos da matriz.

Elementos máximo e mínimo entre os elementos da matriz.

A soma dos elementos de cada coluna.

Tudo deve ser escrito com um comentário.

Por exemplo:

Soma dos elementos = 234.

Elemento máximo = 68.

Elemento mínimo = 5.

Soma dos elementos de uma coluna = 94.

Soma dos elementos de duas colunas = 43.

Exercício 2.

Preencha uma matriz bidirecional de números reais com valores aleatórios. A matriz deve ter N linhas e M colunas.

O intervalo de valores aleatórios deve ser inserido a partir de componentes ***Edit*** separados.

Crie um componente ***StringGrid***. Defina o número de colunas e linhas nas células fixas.

Emita a matriz em um componente ***StringGrid***.

Calcule a soma dos elementos de cada linha da matriz e emita em uma célula separada em cada linha do componente ***StringGrid***.

Data e hora

Para trabalhar com um valor de data e hora, há um tipo de dados especial ***TDateTime*** no ***Delphi.***

As variáveis são declaradas da seguinte maneira:

```
Var
  a, b: TDateTime;
```

TDateTime é um duplo que contém o valor de data e hora. A parte inteira do valor de ***TDateTime*** é o número de dias que passaram desde 30 de dezembro de 1899.

A parte fracionária do valor de ***TDateTime*** é a hora do dia.

Para encontrar o número fracionário de dias entre duas datas, basta subtrair os dois valores, a menos que um dos valores de ***TDateTime*** seja negativo. De forma semelhante, para aumentar o valor de data e hora em um determinado número de dias fracionário se o valor de ***TDateTime*** for positivo, adicione o número fracionário ao valor de data e hora.

Ao trabalhar com valores de ***TDateTime*** negativos, os cálculos devem manipular a parte da hora separadamente. A parte fracionária reflete a fração de um dia de 24 horas, sem considerar o sinal do valor de ***TDateTime***. Por exemplo, 6h de 29.12.1899 é –1,25, não –1 + 0,25, que seria –0,75. Não há valores de ***TDateTime*** entre -1 e 0.

Procedimentos para trabalhar com datas

Vamos considerar os seguintes procedimentos:

DecodeDate(Date: TDateTime; var Year, Month, Day: Word); extrai os valores de ano, mês e dia de um determinado valor do tipo ***Date TDateTime***.

Por exemplo, se a variável A tiver a data 1º de março de 2008, ***DecodeDate*** (A, y, m, d) retornará y=2008, M=3, d=1.

DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word); extrai os valores de hora, minuto, segundo e milissegundo de um determinado valor do tipo ***Time TDateTime***.

Funções para trabalhar com datas

Vamos supor que temos três variáveis do tipo ***TDateTime***:

```
Var
   a, b, c: TDateTime;
Begin
   a:=Date; //Returns the current date
   b:=Time; //Returns the current time
   c:=Now; //Returns the current date and time
End;
```

Essas funções obtêm a data e a hora das configurações do computador. Portanto, se as configurações do computador estiverem incorretas, essas funções retornarão a data e a hora incorretas.

Há funções que convertem a data e a hora em uma string.

TimeToStr(t: TDateTime): String; converte Time de um valor de ***TDateTime*** em uma string de hora formatada. A hora é formatada usando o valor de ***LongTimeFormat***, que, por sua vez, usa o valor de ***TimeSeparator***.

DateToStr(t: TDateTime): String; converte ***Date*** de um valor de ***TDateTime*** em uma string de data formatada.

A data é formatada usando o valor de ***ShortDateFormat***, que, por sua vez, usa o valor de ***DateSeparator***, por exemplo, '***03:04:2015***'.

DateTimeToStr(t: TDateTime): String; converte DateTime de um valor de TDateTime em uma string de data e hora formatada. A string inclui:

Data em ***ShortDateFormat***

Um espaço em branco

Hora em ***LongTimeFormat***

Além disso, há funções que convertem uma string em data, hora ou data e hora.

```
StrToTime(S: String): TDateTime;
StrToDate(S: String): TDateTime;
StrToDateTime(S: String): TDateTime;
```

Aqui estão algumas características do formato da string:

- Para a hora, é possível ignorar os segundos. Por exemplo, '9:10' será convertida como '9:10:00' por padrão.
- A data deve estar correta.
- Se o ano for especificado apenas por dois dígitos, ele será convertido como um ano do intervalo de 1930 a 2030.
- Se o ano não for especificado, ele será convertido como o ano atual.
- O dia e o mês devem ser específicos.

Consideraremos algumas outras funções para trabalhar com datas e horas.

DayOfWeek(Date: TDateTime): Word retorna um número de índice para o dia da semana. 1 = domingo, 2 = segunda-feira etc.

DecodeDateFully(Date: TDateTime; var Year, Month, Day, DOW: Word): Boolean extrai o ano, o mês, o dia e o DOW (dia da semana). Retorna True para um ano bissexto;

EncodeDate(var Year, Month, Day: Word): TDateTime gera um valor de retorno TDateTime dos valores de ano, mês e dia passados;

EncodeTime(Hour, Min, Sec, Msec: Word): TDateTime gera um valor de retorno TDateTime dos valores de hora, minutos, segundos e ms (milissegundo) passados.

Operações com datas

- É possível subtrair duas datas e, assim, a parte inteira de uma diferença é o número de dias entre as datas.
- Não faz sentido adicionar as duas datas, mas podemos adicionar ou subtrair um inteiro. Nesse caso, obtemos uma nova data, que difere no número de dias correspondente, para trás ou para frente.

A hora pode ser somada da seguinte maneira:

Por exemplo, há duas maneiras de saber que horas serão em 1,5 hora:

```
Time+1,5/24
    or
Time+StrToTime('1:30')
```

Exercícios

Exercício 1.

Um ***StringGrid*** está preenchido com os sobrenomes, datas de nascimento e datas das concessões do Prêmio Nobel. Determine o sobrenome do premiado mais jovem.

Exercício 2.

Vamos supor que uma escola tem aulas de duração igual. Os intervalos nessa escola também têm durações iguais, que são diferentes da duração das aulas. Crie um cronograma escolar usando: a hora de início da primeira aula, a duração das aulas, a duração dos intervalos e o número de aulas. Todas as quantidades são inseridas em caixas de texto. O cronograma deve ser apresentado da seguinte maneira:

	Hora de início	Hora de término
Aula 1	9:00	9:40
Intervalo	9:40	9:50
Aula 2	9:50	10:30
Intervalo	10:30	10:40
. . .		

Exercício 3.

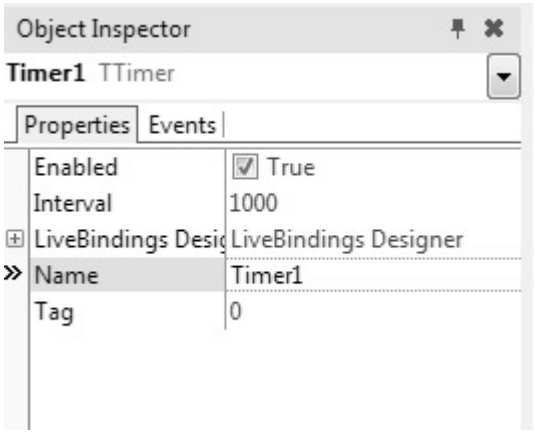
Uma data é inserida em uma caixa de texto. Calcule quantos anos, meses e dias inteiros separam essa data do dia de hoje. Emita o resultado em diferentes rótulos.

Timer

Às vezes, é necessário que algumas instruções em um programa sejam executadas em um determinado período de tempo. Consideraremos a seguinte tarefa: pintar o form de vermelho em cinco segundos após a execução do programa.

Para resolver essa tarefa, usaremos o componente ***Timer*** da guia ***System***.

Coloque o ***Timer*** em qualquer lugar no form. Ele não será visualizado durante o tempo de execução. Vamos conhecer essas propriedades de componentes no Object Inspector.



Além de ***Name***, estamos interessados nas propriedades ***Enabled*** e ***Interval***. Se ***Enabled*** for ***True***, o timer estará em execução, caso contrário, ele estará desativado. A propriedade ***Interval*** especifica o tempo de reação do timer em milissegundos. Por padrão, o ***Interval*** é 1.000 ou um segundo.

Definiremos o valor de ***Interval*** como 5.000 (cinco segundos).

Clicaremos no timer no form e será criado um manipulador de evento do timer. Nele, adicionaremos uma instrução para alterar a cor do form. Aqui está:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=clRed;
end;
```

Após a execução do programa, veremos que, após um pouco, o form será pintado na cor vermelha.

Vamos corrigir um pouco o programa. Colocaremos o botão no form. Adicionaremos a seguinte instrução ao manipulador de evento de clique do botão:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   timer1.Enabled:=true;
end;
```

Definiremos o valor da propriedade ***Enabled*** como ***False***. Agora, não acontecerá nada após a execução do programa. No entanto, após clicar no botão, o form será pintado na cor vermelha em cinco segundos. Então, após o botão ser clicado, a propriedade ***Enabled*** será definida como true, o timer será ativado e, em cinco segundos, o manipulador do timer será executado.

Vamos alterar o manipulador do timer da seguinte forma:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=256*256*random(256)+256*random(256)+ran
dom(256);
end;
```

Aqui, atribuímos uma cor aleatória.

Alteraremos o manipulador do evento de clique do botão também:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   timer1.Enabled:=true;
   timer1.Interval:=1500;
end;
```

Aqui, definimos o intervalo do timer para 1,5 segundo.

Vamos executar o programa e clicar no botão. O que veremos? A cada 1,5 segundo, a cor do form muda aleatoriamente. O timer em execução chama o manipulador de evento do timer constantemente no intervalo definido. Para parar o timer, a propriedade ***Enabled*** deve ser definida como ***False***.

Adicionaremos mais botões ao form e escreveremos em seu manipulador de evento de clique:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   timer1.Enabled:=false;
end;
```

Agora, clicando no primeiro botão, iniciaremos a alteração de cor. Vamos pará-la clicando no segundo botão.

Se desejarmos que o timer seja disparado apenas uma vez, precisamos definir ***False*** para a propriedade ***Enabled*** no manipulador de evento do timer. Portanto, se quisermos que isso ocorra após os 1,5 segundo em que a cor muda uma vez, o manipulador de evento do timer deverá ser alterado da seguinte maneira:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=256*256*random(256)+256*random(256)
+random(256); timer1.Enabled:=false;
end;
```

Para desativar um timer em um determinado intervalo, você deve adicionar outro timer. Por exemplo, para parar a mudança de cores em 20 segundos, adicionaremos o segundo timer (não se esqueça de definir a propriedade ***Enabled*** como ***False***). Escreveremos no segundo manipulador de evento do timer:

```
procedure TForm1.Timer2Timer(Sender: TObject);
begin
   timer1.Enabled:=false;
   timer2.Enabled:=false;
end;
```

Alteraremos o primeiro manipulador de evento de clique do botão e o primeiro manipulador de evento de timer da seguinte maneira:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   timer1.Enabled:=true;
   timer1.Interval:=1500;
   timer2.Enabled:=true;
   timer2.Interval:=20000;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=256*256*random(256)+256*random(256)
+random(256);
end;
```

Executaremos o programa. Após o primeiro botão ser clicado, a cor do form começará a mudar, com um intervalo de 1,5 segundo. A alteração de cor vai parar em 20 segundos.

Usando tempos, é possível criar objetos em movimento. Vamos considerar mais um exemplo.

Vamos colocar qualquer componente na parte esquerda do form, por exemplo, uma caixa de texto. Corrigiremos o primeiro manipulador de evento de timer da seguinte maneira:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   edit1.Left:=edit1.Left+15;
end;
```

Execute o programa e clique no primeiro botão. Veremos que a caixa de texto se moverá da esquerda para a direita.

Os timers nos permitem medir intervalos de tempo durante o tempo de execução, alteram os parâmetros dinamicamente e criam objetos em movimento.

Exercícios

Exercício 1.

7Usando um timer, escreva um programa para 'pintar' um céu estrelado. As estrelas são iluminadas e, após um pouco, apagam.

Exercício 2.

Escreva um programa que cria um cronômetro de penalidade.

O intervalo de tempo é definido como ***TEdit***. Após clicar no botão, o timer começa a contagem regressiva em segundos. Assim que o tempo for 0, a cor do plano de fundo de ***TEdit*** fica vermelho e o timer começa a contar o tempo de penalidade.

Exercício 3.

Escreva um programa para criar um relógio digital, que mostra a hora, a data e o dia da semana.

Exercício 4.

Escreva um programa que imite um relógio com ponteiros. O segundo ponteiro deve se mover a cada segundo, minuto, cada minuto, hora e cada hora (é possível fazer o ponteiro da hora se mover suavemente, em proporção à parte passada da hora).

Arquivos de texto

Gravando dados no arquivo de texto

Qualquer programa contemporâneo precisa executar um pouco de trabalho nos arquivos.

Um **arquivo** é uma parte nomeada de um disco ou outro meio de informações que possui algumas informações gravadas nele.

Esses arquivos podem ter objetivos especiais, como arquivos de bibliotecas padrão ou podem ser arquivos de dados para ou de um programa. Todos os arquivos têm seus formatos. As especificações do formato de arquivo dependem da codificação de dados no arquivo.

Um dos formatos de arquivo é o formato de **arquivo de texto**.

Um ***arquivo de texto*** contém uma ou mais linhas de comprimento arbitrário e cada linha pode conter caracteres arbitrários. Cada ***arquivo de texto*** termina com um caractere de fim de linha especial e o arquivo termina com um caractere de fim de arquivo. Você pode editar um ***arquivo de texto*** com o editor ***Notepad*** ou qualquer outro editor de arquivo de texto.

Um ***arquivo de texto*** pode ser visto como um agregado de dados combinados, que tem um nome comum. Os nomes de ***arquivo de texto*** normalmente têm a extensão ***.txt***, mas essa extensão não garante que o arquivo é um ***arquivo de texto***. Além disso, os nomes de ***arquivos de texto*** podem ter extensões diferentes de ***.txt***. Aqui estão exemplos de extensões padrão para ***arquivos de texto***: ***.ini***, ***.log***, ***.inf***, ***.dat***, ***.bat***. Usaremos a extensão ***.txt***.

Os ***arquivos de texto*** são arquivos de acesso sequencial e podem ser acessados apenas em na ordem sequencial.

Em arquivos de acesso sequencial, os registros de dados são colocados na ordem em que são gravados no arquivo, sequencialmente um após o outro. Para encontrar um registro de dados específico, o programa precisa analisar sequencialmente o arquivo do início até os dados necessários. Consequentemente, o tempo necessário para acessar o registro de dados linearmente depende de sua posição no arquivo.

Tais arquivos devem ser usados se um programa processa (praticamente) todos os dados contidos nele e o conteúdo do arquivo raramente muda. Portanto, a principal desvantagem desses arquivos é que eles são usados para atualizar dados existentes, alterar registros de dados para novos e inserir novos registros.

Vejamos como trabalhar com arquivos de texto no ***Delphi***.

Um programa não pode acessar um disco diretamente. Ele deve usar variáveis para manter os dados na RAM. Assim, precisamos de uma variável para manter uma referência para um arquivo. Tal variável é chamada de ***variável de arquivo*** e pode ser definida da seguinte forma:

```
Var
   f: TextFile;
```

Antes de usar um arquivo, o programa precisa associar a ***variável de arquivo*** e o arquivo a ser usado, quer ele exista ou precise ser criado. Para fazer isso, o programa precisa usar o procedimento ***AssignFile***, que tem dois parâmetros: a ***variável de arquivo*** e a string com um local ou caminho completo para o arquivo.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
begin
   AssignFile(f,'my.txt');
end;
```

Neste exemplo, associamos a ***variável de arquivo f*** e o arquivo chamado ***my.txt***, que está (ou estará) localizado no diretório atual (diretório do qual o programa foi executado). O nome do arquivo, dado como um segundo parâmetro, é um nome de arquivo local. Devemos usar o caminho completo do arquivo se ele não estiver localizado no diretório atual.

Atenção: a associação da ***variável de arquivo*** a um nome de arquivo não verifica se o arquivo existe ou não. Isso não abre um arquivo existente ou cria um novo.

Um ***arquivo de texto*** pode ser lido ou gravado. É impossível ler de um ***arquivo de texto*** e gravar nele simultaneamente.

Para realizar as operações de leitura ou gravação no arquivo, precisamos abri-lo. Podemos abrir um arquivo para ser lido ou gravado.

Primeiro, analisaremos o processo de gravação do arquivo.

Para abrir um arquivo para gravação, precisamos usar o operador ***Rewrite(f)***, em que ***f*** é uma ***variável de arquivo.***

Essa operação cria um arquivo vazio com o nome associado ao ***AssignFile***. O conteúdo do arquivo será apagado se esse arquivo já existir.

Agora podemos gravar informações no arquivo, usando os operadores, ***write*** e ***writeln***. Eles funcionam como seus equivalentes para imprimir no console. A única diferença é que precisamos especificar a ***variável de arquivo*** no início dos argumentos, indicando que a saída vai para o arquivo e não para o console.

A seguir está um procedimento de exemplo que grava várias informações em um ***arquivo de texto "example.txt".***

```
  Procedure ZapFile;
  var
     f: TextFile;
     x, y: integer;
     Ok: boolean;
  begin
     AssignFile(f, 'example.txt');
     ReWrite(f);
     x:=10;
     y:=7;
     Writeln(f, x); //Escreve 10 na primeira lina do
    // arquivo. O cursor vai para a segunda linha.
     Write(f, x+2);); //Esceve 12 (10 + 2) na segunda
    //linha. O cursor permanece na segunda linha
     Write(f, Hello'););
    //No fim da segunda linha (posição do cursor) será
    // escrito a palavra 'Hello'. O cursor permanece no
    // final da segunda linha.
     Write(f, x, y); //Escreve 10 e depois 7 sem espaço
    // na posição do cursor. O cursor permanece no fim da
    // segunda linha.
     Writeln(f, x, y);); //Escreve 10 e depois 7 sem
espaço na posição do cursor. O cursor se move para a
terceira linha
     Ok:=5>7;
     Writeln (f, Ok); //Escreve False (valor da variável
     OK) na terceira linha. O cursor se move para a
     quarta linha
     Writeln(f, x,' ', y); //Escreve 10, depois dois
    // espaço, depois 7 na quarta linha. O Cursor vai
    // para a quinta linha
     Writeln(f,'x=', x); //Escreve "x=10" na quinta linha
do arquivo.
     CloseFile(f);
  end;
```

Você pode ver que encerramos o arquivo com a instrução ***CloseFile(f)***. O procedimento ***CloseFile*** **deve ser chamado após ter terminado de trabalhar com o arquivo**. Ele encerra o arquivo e conclui a gravação. Sem essa chamada, algumas informações podem ser perdidas.

Como dissemos acima, a chamada para ***reescrever*** um procedimento sempre cria um novo arquivo. No entanto, há casos em que é necessário adicionar informações em um arquivo existente. Nesses casos, o arquivo pode ser aberto com o procedimento ***Append***. Esse procedimento não cria um arquivo novo, mas abre um existente e coloca o cursor no final do arquivo. Consequentemente, todas as instruções ***write*** e ***writeln*** têm a informação da posição do cursor nelas.

É possível usar ***Append(f)*** apenas em um arquivo existente. Caso contrário, o sistema do tempo de execução exibirá um erro e o programa será encerrado. Após terminar de trabalhar com um arquivo que foi aberto com ***Append(f)***, precisamos fechá-lo com uma chamada de ***CloseFile(f)***.

O argumento do arquivo nos procedimentos deve ser sempre declarado como um ***argumento de variável***, usando a palavra-chave ***var***.

A seguir está um exemplo de um programa que cria dois arquivos, "test1.txt" e "test2.txt", e escreve várias linhas com zeros nelas. O primeiro arquivo deve conter três linhas com cinco zeros e um segundo arquivo deve conter quatro linhas com três zeros. Os zeros são separados por dois espaços.

```
procedure ZapZero(var fl: TextFile; n, m: integer);
var
   i, j: integer;
begin
   Rewrite(fl);
   for i:=1 to n do
   begin
        for j:=1 to m do
        write(fl,0:3);
        writeln(fl)
   end;
   CloseFile(fl)
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
   f, g: TextFile;
begin
   AssignFile(f, 'test1.txt');
   AssignFile(g, 'test2.txt');
   ZapZero(f,3,5);
   ZapZero(g,4,3);
end;
```

Lendo dados de um arquivo de texto

Vamos analisar o processo de leitura de dados de um arquivo. Abrimos o arquivo para leitura com a chamada para ***Reset(f)***.

Atenção: o arquivo que está prestes a ser aberto para leitura deve existir em um disco. Caso contrário, um erro de tempo de execução exibirá uma mensagem de erro e encerrará a execução do programa.

Se o arquivo for aberto com sucesso, o cursor será posicionado no início da primeira linha. Podemos ler essa linha chamando o procedimento ***ReadlLn(f, s)***.

Após a instrução ***ReadLn*** ser concluída, a variável ***s*** terá o conteúdo da primeira linha e o cursor será posicionado no início da próxima linha. Se repetirmos a instrução, a variável será atribuída a um conteúdo da segunda linha, o cursor será posicionado no início da terceira linha e assim por diante. Após terminarmos de trabalhar com o arquivo, precisamos fechá-lo, usando a chamada de procedimento ***CloseFile(f)***, exatamente como ao gravar arquivos.

Abaixo está um exemplo de um procedimento que gravará a primeira linha de um arquivo ***'my.txt'*** no componente ***memEx1 Memo***.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
  // arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   ReadLn(f, s); //Lê a primeira linha do arquivo
   memEx1.Lines.Append(s); //Coloca a linha no Memo
   CloseFile(f); //Fecha o arquivo
end;
```

Esse procedimento sempre colocará apenas a primeira linha do arquivo. Podemos usar os loops para ler várias linhas do arquivo. O exemplo abaixo lerá e colocará cinco linhas no ***TMemo*** do arquivo ***'my.txt'.***

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   i: integer;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
   // arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   For i:=1 to 5 do //Repete 5 vezes
   begin
        ReadLn(f, s); //Lê a linha do arquivo
        memEx1.Lines.Append(s); //Coloca a linha no Memo
   end;
   CloseFile(f); //Fecha o arquivo
end;
```

Como já observamos, os ***arquivos de texto*** são arquivos de acesso sequencial, isto é, não é possível acessar uma linha específica sem ler as linhas anteriores. Por exemplo, precisamos ler quatro linhas se desejarmos acessar a quinta linha do arquivo. Podemos usar a instrução ***ReadLn(f)*** sem especificar nenhuma variável.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   i: integer;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
// arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   For i:=1 to 4 do //Rpete 4 vezes
     ReadLn(f); // pula a linha
     Readln(f, s); //Lê a quinta linha do arquivo.
     memEx1.Lines.Append(s); //Coloca alinha no Memo
   CloseFile(f); //Fecha o arquivo
end;
```

No exemplo acima, ***ReadLn(f)*** apenas move o cursor para o início da próxima linha, sem ler nenhum dado.

O número de linhas em um arquivo não é sempre é conhecido a priori. Então, é possível ler todas as linhas de um arquivo sem saber sua contagem? Podemos usar a função Boolean ***EOF(f)*** (**EOF** significa **E**nd **O**f **F**ile). Essa função retorna ***True*** se o cursor estiver no fim do arquivo, caso contrário, é ***False***. Aqui está um exemplo de um procedimento que lê todas as linhas de um ***arquivo de texto*** e as coloca em um componente ***Memo***:

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
  // arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   While not EOF(f) do
   //Repete enquanto não chegar no fim do arquivo
begin
   ReadLn(f, s); //Lê a linha do arquivo
   memEx1.Lines.Append(s); //Coloca a linha no TMemo
   end;
   CloseFile(f); //Fecha o arquivo
end;
```

Observe que, em todos os nossos exemplos, usamos a instrução ***ReadLn***, não a ***Read***. O último exemplo não concluirá seu trabalho se alterarmos ***ReadLn*** para ***Read***. O motivo disso é que ***Read*** não move o cursor além do final da linha atual e lerá linhas vazias quando o cursor estiver no final da linha. Isso resultará em um loop infinito, que lerá uma quantidade infinita de linhas vazias e as colocará no ***Memo***. Isso ocorre porque o cursor nunca chegará ao fim do arquivo. Portanto, se lermos as linhas do arquivo, precisamos usar uma instrução ***ReadLn***.

Também é possível saber se o cursor aponta para o fim da linha. Podemos usar a função Boolean ***EOLN(f)*** para isso. Essa função retorna ***True*** quando o cursor está no fim da linha e retorna ***False*** se não estiver.

Agora sabemos como trabalhar com um ***arquivo de texto*** quando lemos dados de um arquivo linha por linha. Podemos ler também valores inteiros ou reais de um arquivo, que são separados por um ou vários espaços. Nesse caso, podemos ler as variáveis diretamente. Vamos observar um exemplo de como fazer isso.

Vamos supor que temos um ***arquivo de texto*** que contém números, que são separados por espaços (qualquer número de espaços diferente de zero). Um exemplo é fornecido na figura à direita.

Logo após abrir o arquivo, vamos usar a instrução ***Read(f, x)***, em que ***f*** é uma ***variável de arquivo*** e ***x*** é uma variável inteira. Após executar essa instrução, a variável ***x*** será definida como 10 e o cursor será posicionado no delimitador (espaço) logo após 0. Após executar ***Read(f, x)*** uma segunda vez, o cursor ignorará todos os delimitadores (espaços). Em seguida, o valor 7 será lido na variável ***x*** e o cursor parará no delimitador (espaço) logo após 7. Executar ***Read(f, x)*** uma terceira vez ignorará todos os delimitadores (espaços) antes do número 25. O número 25 será lido na variável ***x*** e o cursor parará após 25. Falando de modo geral, a ***Read(f, x)*** para uma instrução ***x*** de variável inteira ignora todos os delimitadores antes do número, lê o número na variável e posiciona o cursor no símbolo não numérico após o número.

Aqui está um exemplo:

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   x, i, k: integer;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
  // arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   k:=0;
   For i:=1 to 5 do //Repete cinco vezes
         begin
         Read(f, x); //Lê outro número
         k:=k+x; // Adiciona a k
         end;
   CloseFile(f); //Fecha o arquivo
end;
```

Após executar esse procedimento, a variável ***k*** conterá ***57***, a soma dos cinco primeiros números ***10, 7, 25, 14*** e ***1.*** Observe que ***Read(f, x)*** lê os dados numéricos em uma variável inteira, ignorando todos os símbolos de delimitador. Os delimitadores para ***Read*** são os espaços e símbolos de fim de linha.

No exemplo acima, o que acontecerá se trocarmos a instrução ***Read(f, x)*** por uma instrução ***ReadLn(f, x)***? A principal diferença entre ***Read*** e ***ReadLn*** é que ***ReadLn*** ignorará todos os dados na linha até o cursor chegar à próxima linha.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   x, i, k: integer;
begin
   AssignFile(f,'my.txt'); //Associa o nome do
  // arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   k:=0;
   For i:=1 to 5 do //repete cinco vezes
        begin
        ReadLn(f, x); //Lê ou número
        k:=k+x; // Adiciona a k
        end;
   CloseFile(f); //Fecha o arquivo
end;
```

Após executar o procedimento acima, a variável ***k*** conterá o número ***86***, a soma dos números no início das cinco linhas: ***10, 14, 24, 27 u 11.***

Um problema encontrado com frequência é o de ler todos os números da linha do arquivo. Além disso, a quantidade de números não é conhecida com antecedência, o que pode ser problemático. Parece lógico usar a função ***EOLN(f)*** para verificar a situação do fim da linha e principalmente repetir um dos exemplos acima. Nesse caso, o procedimento terá uma aparência semelhante a seguinte:

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   x, k: integer;
begin
   AssignFile(f,'my.txt');
   //Associa um arquivo com uma variável
   Reset(f); //Abre o arquivo para leitura
   k:=0;
   While not EOLN(f) do
   //Repete enquanto não terminar a linha
   begin
       Read(f, x); //Lê outro número
       k:=k+x; // Adiciona a soma
   end;
   CloseFile(f); //Fecha o arquivo
end;
```

Após a execução desse exemplo, a variável ***k*** deverá conter o número ***42***, a soma dos números da primeira linha, ***10, 7, 25.*** No entanto, esse não será o caso. A resposta variará de acordo com a existência ou não de espaços após o último número na linha lida. Se não houver espaços, a resposta esperada será um.

Vamos analisar o que estaria acontecendo se houvesse espaços após o número ***25*** em nosso arquivo de exemplo. Nesse caso, após ler o ***25***, o cursor será posicionado no espaço log após o dígito cinco. A chamada para ***EOLN(f)*** retornará ***False*** porque o espaço não é um símbolo de fim de linha. O loop será executado mais uma vez e ***Read(f, x)*** ignorará todos os delimitadores, incluindo o símbolo de fim de linha, até o próximo número ser lido. Como resultado, a variável ***k*** conterá uma soma de números de várias linhas. O número de linhas e a quantidade de números lidos dependem do número de linhas consecutivas que contêm espaços antes do fim da linha.

Para evitar esse problema, é necessário usar a função ***SeekEOLN(f),*** em vez da ***EOLN(f),*** **ao ler números**. Essa função primeiro ignora todos os espaços e somente então verifica a existência do símbolo de fim da linha. No exemplo acima, se alterarmos a chamada para ***EOLN(f)*** para a chamada para ***SeekEOLN(f)***, nosso exemplo funcionará corretamente, independentemente do número de espaços antes do símbolo de fim da linha.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   x, k: integer;
begin
   AssignFile(f,'my.txt');
   //Associa o arquivo com uma variável
   Reset(f); //Abre o arquivo para leitura
   k:=0;
   While not SeekEOLN(f) do
   //Repete enquanto não terminar a linha
   begin
        Read(f, x); //Lê o próximo número
        k:=k+x; // Adiciona a soma
   end;
   CloseFile(f); //Fecha o arquivo
end;
```

O ***Delphi*** também tem a função ***SeekEOF(f),*** que funciona exatamente como a ***SeekEOLN(f)***, mas ignora os espaços e os símbolos do fim da linha. Você deve usá-la ao ler números de um arquivo usando a instrução ***Read(f, x)***.

Um procedimento para calcular a soma de todos os números em um arquivo pode ter a seguinte aparência:

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: string;
   x, k: integer;
begin
   AssignFile(f,'my.txt');
   //Associa o arquivo a uma variável
   Reset(f); //Abre o arquivo para leitura
   k:=0;
   While not SeekEOF(f) do
   //Repete enquanto não termina a linha
    begin
        Read(f, x); //Lê o número
        k:=k+x; // Adiciona a soma
   end;
   CloseFile(f); //Fecha o arquivo
 end;
```

Use a regra a seguir para se guiar quando precisar que seu programa leia dados de um ***arquivo de texto***: se você ler dados como **linhas**, use as funções ***EOLN(f)*** e ***EOF(f)***, e se ler dados como **números**, use as funções ***SeekEOLN(f)*** e ***SeekEOF(f)***.

Quando um programa tentar ler um número de dados não numéricos (***EOF*** ou apenas texto), o sistema do tempo de execução exibirá uma mensagem de erro e encerrará a execução do programa. ***Abort*** também será executado quando um programa tentar ler um número real em uma variável inteira ou formato de número em um arquivo que não é compatível com o formato do ***Delphi*** (um separador de vírgula para a parte fracionária em vez de um ponto).

Exercícios

Exercício 1.

Escreva um programa que encontra a menor linha em um arquivo e grava seu conteúdo em um componente ***Label***. Se houver várias linhas igualmente curtas, o programa deverá gravar a última linha encontrada. O nome do arquivo deve ser lido da caixa de texto.

Para testar o comportamento do programa, use um ***arquivo de texto*** que foi criado usando um editor de texto externo (***Notepad*** sendo um deles).

Exercício 2.

Escreva um programa que lê linhas de um arquivo e emita as linhas de palíndromo para o ***TMemo***. O nome do arquivo deve ser inserido na caixa de texto (é possível usar uma caixa de diálogo).

Para testar o comportamento do programa, use um ***arquivo de texto*** que foi criado usando um editor de texto externo (***Notepad*** sendo um deles).

Exercício 3.

Um ***arquivo de texto*** contém várias linhas. Cada linha contém vários números inteiros, que são separados por um ou mais espaços. Escreva um programa que lerá o arquivo e gravará em outro arquivo as linhas que têm apenas uma soma par dos números. Os nomes de arquivo são inseridos em caixas de texto (é possível usar caixas de diálogo).

Para testar o comportamento do programa, use um ***arquivo de texto*** que foi criado usando um editor de texto externo (***Notepad*** sendo um deles).

Exercício 4.

Um ***arquivo de texto*** contém informações sobre crianças e escolas que participaram das Olimpíadas. A primeira linha contém o número dos registros das crianças. Cada uma das linhas subsequentes tem o seguinte formato:

```
<sobrenome> <iniciais> <número da escola>
```

em que <sobrenome> é uma string que contém no máximo 20 símbolos, <iniciais> é uma string que contém quatro símbolos (caractere, ponto, caractere, ponto), <número da escola> é um número de um ou dois dígitos. <sobrenome> e <iniciais> e <iniciais> e <número da escola> são separados por um único espaço. A seguir está um exemplo de uma string de entrada:

```
Smith J. C. 57
```

Escreva um programa que emitirá para um rótulo o número da escola com o menor número de participantes. Os números de escola devem ser listados e, se houver várias escolas com o mesmo menor número de participantes, eles devem ser separados por uma vírgula. O número de linhas é maior que 1.000.

MÓDULO 25

Caixas de diálogo de arquivo padrão

No módulo anterior, revisamos a maneira padrão e a típica para trabalhar com arquivos. No entanto, o ***Delphi*** tem a opção de trabalhar com arquivos com janelas de ***diálogo de arquivo***, isto é, selecionar arquivos para ler ou gravar por meio de janelas de ***diálogo de abertura de arquivo*** e ***salvamento de arquivo***.

Os componentes das janelas de diálogo de arquivo padrão são localizados na guia ***Dialogs*** na paleta. Quando colocados no form, esses componentes não são visíveis durante a execução do programa.

Usaremos o componente ***OpenDialog*** (ícone de arquivo aberto) para ler as informações do arquivo (deve haver um arquivo) e ***SaveDialog*** (ícone de disco) para gravar no arquivo existente ou em um novo arquivo. O nome do novo arquivo será inserido pelo teclado.

Os componentes têm a propriedade ***FileName*** e o método Boolean ***Execute***.

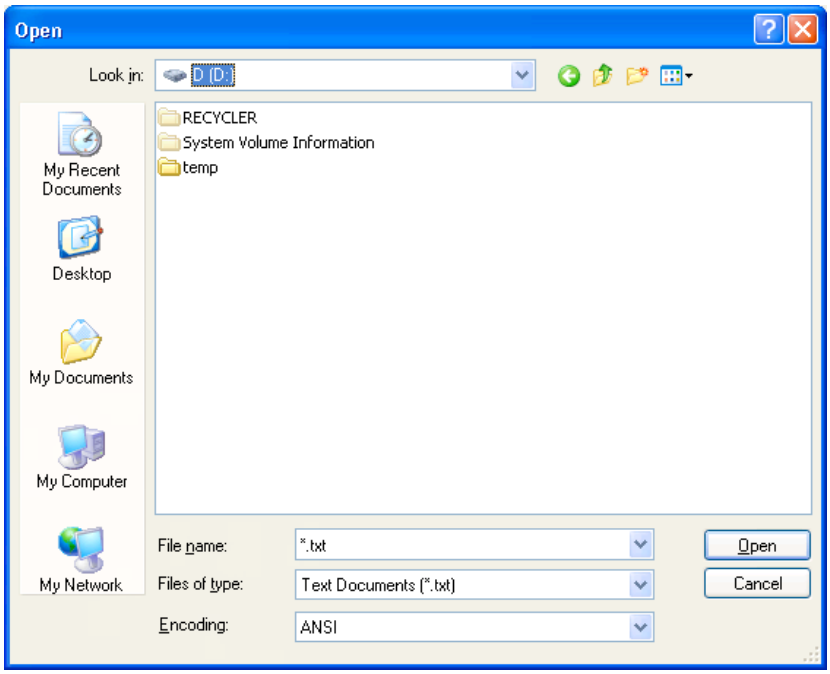
A seguir está um algoritmo de trabalho com janelas de diálogo de arquivo:

```
Var
   Okf1, Okf2: Boolean;
   StF1, StF2: string;
Begin
//Arquivo para leitura foi selecionado?
Begin//Se o arquivo para leitura foi selecionado, então
//Encontre o nome do arquivo no Dialog
//Associe o arquivo a uma variável
//Abra o arquivo para leitura
<process File information: read, process, etc>
End; //Feche o arquivo
//Arquivo para escrita selecionado?
//Se o arquivo para escrita foi selecionado, então
Begin
//Encontre o nome do arquivo no Dialog
//Associe o arquivo a uma variável
//Abra o arquivo para escrita
<write information to File>
//Feche o arquivo
End;
end;
```

A primeira linha do programa será:

```
Okf1:= OpenDialogEx1.Execute;
```

Quando executarmos o programa, a seguinte ***OpenDialog*** de arquivo padrão será exibida. Selecionaremos o arquivo necessário nessa janela de diálogo.



Pressionar o botão ***Open*** abrirá o arquivo para leitura. O método de execução retornará ***True*** e o nome do arquivo será atribuído à propriedade ***FileName*** do componente ***OpenDialogEx1***.

Vamos continuar o programa:

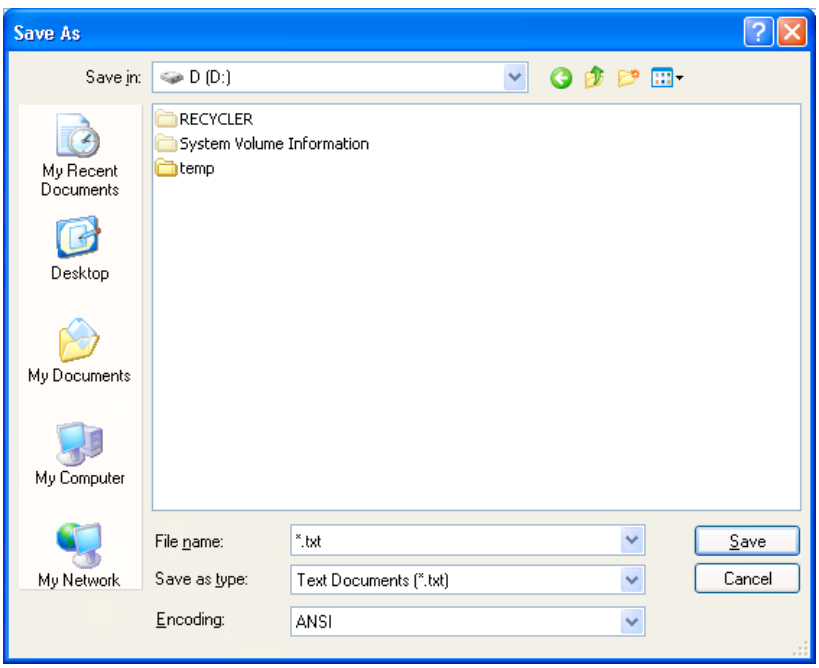
```
If Okf1 then
//se o arquivo para leitura está selecionado, então
Begin
Stf1:= OpenDialogEx2.FileName;
//Pegue o arquivo do Dialog e associe a variável StF1
AssignFile(f, Stf1); //Associe o arquivo a variável
Reset(f); //Abra o arquivo para leitura
……; //Processe os dados no arquivo
Closefile(f); // Feche o arquivo
End;
```

Neste exemplo, após o arquivo ser aberto, podemos substituir o "....." por operadores que são necessários para processar os dados para resolver os problemas, ler informações do arquivo, processar os dados etc. Já testamos isso nos módulos anteriores.

Para gravar as informações no arquivo:

```
Okf2:=SaveDialogEx2.Execute;
//O arquivo está selecionado para escrita?
```

Quando este comando for executado, o ***diálogo de salvamento de arquivo*** padrão será exibido. Nele, podemos selecionar um arquivo existente ou digitar o nome de um novo arquivo na caixa de texto "File Name". O nome do arquivo selecionado (e o caminho para ele) serão atribuídos à propriedade ***SaveDialogEx1.FileName***.



A operação será cancelada se você pressionar o botão ***Cancel***.

O código para registrar informações no arquivo será semelhante a esse:

```
Okf2:=SaveDialogEx2.Execute;
//Is File selected for Write?
If Okf2 then
//Se o arquivo para escrever está selecionado, então
Begin
Stf2:= SaveDialogEx2.FileName;
//Selecione o nome do arquivo do Dialog
AssignFile(g, Stf2);
//Associe o arquivo para a variável
Rewrite(g); //Abra o arquivo para escrita
…… .; //Escreva dados no arquivo, processa dados
Closefile(g); //Fecha o arquivo
End;
End;
```

Agora você deve saber como trabalhar com arquivos no modo ***Dialog***.

Exercícios

Exercício 1.

Crie um ***arquivo de texto*** no ***Notepad.*** Usando ***caixas de diálogo de arquivo padrão***, escreva um programa que copiará todas as strings de comprimentos pares desse arquivo para File 2 e todas as strings de comprimentos ímpares para File 3.

Exercício 2.

Crie um ***arquivo de texto*** no ***Notepad*** para que cada linha do arquivo contenha inteiros separados uns dos outros por um ou mais espaços. Usando ***caixas de diálogo de arquivo padrão***, escreva um programa que calculará a soma de todos os números e registrará essa soma em um novo arquivo.

Exercício 3.

Crie um ***arquivo de texto*** no ***Notepad*** para que cada linha do arquivo contenha inteiros aleatórios e palavras aleatórias separados uns dos outros por um ou mais espaços. Usando ***caixas de diálogo de arquivo padrão***, escreva um programa que calculará a soma de todos os números e registrará essa soma em um novo arquivo.

Aplicações com Banco de Dados

Introdução

Um banco de dados é uma coleção organizada de dados. Na linguagem de computação, um banco de dados é uma coleção de registros (também chamados de linhas - representam um único item de dados, estruturados implicitamente em uma tabela) armazenados em um computador de forma sistemática, de modo que um programa de computador possa consultá-los para responder a perguntas. Para uma melhor recuperação e classificação, cada registro é, normalmente, organizado como um conjunto de elementos de dados (fatos). Os itens recuperados em resposta a consultas tornam-se informações que podem ser usadas para tomar decisões. O programa de computador usado para gerenciar e consultar um banco de dados é conhecido como um sistema de gerenciamento de banco de dados (DBMS).

O conceito central de um banco de dados é o de uma coleção de registros ou pedaços de conhecimento. Tipicamente, para uma determinada base de dados, não há uma descrição estrutural do tipo de fatos realizada na base de dados: essa descrição é conhecida como um esquema. O esquema descreve os objetos que são representados no banco de dados e as relações entre eles. Há diferentes formas de organizar um esquema, ou seja, de modelar a estrutura de banco de dados, conhecidas como modelos de banco de dados (ou modelos de dados).

Hoje, o modelo em uso mais comum é o relacional, que representa todas as informações sob a forma de várias tabelas relacionadas, cada uma composta de linhas e colunas (a verdadeira definição utiliza a terminologia matemática). Este modelo representa as relações através da utilização de valores comuns a mais de uma tabela. Outros modelos, como o hierárquico e o de rede, usam uma representação mais explícita de relacionamentos.

Linguagem SQL

SQL (Structured Query Language) é a linguagem mais popular usada para criar, modificar e recuperar dados de sistemas de gerenciamento de banco de dados relacionais. A linguagem evoluiu além do seu propósito original para suportar sistemas de gerenciamento de banco de dados objeto-relacional. É um padrão ANSI/ISO.

O componente DML (Data Manipulation Language) compreende 4 opções básicas:

- UPDATE – Para modificar linhas de uma tabela
- DELETE – Para remover linhas de uma tabela
- INSERT – Para adicionar linhas em uma tabela

Os comandos DDL (Data Definition Language) incluem o seguinte:

- CREATE - Para criar um novo banco de dados, tabela, índice ou consulta armazenada
- ALTER - Modifica a estrutura de uma tabela existente e outros objetos
- DROP - Para destruir um banco de dados, tabela, índice ou consulta armazenada já existente

Outros comandos comuns:

- SELECT - Para obter dados do banco de dados e impor ordenação sobre ele
- REVOKE e GRANT - Para dar e receber direitos de acesso a objetos de banco

Por que Usar Bancos de Dados em Aplicativos?

As bases de dados são utilizadas em muitas aplicações, abrangendo praticamente toda a gama de software de computador. Bancos de dados são o método preferido de armazenamento para aplicações multiusuário, onde é necessária a coordenação entre muitos usuários. Mesmo os usuários individuais os acham convenientes e muitos programas de correio eletrônico e organizadores pessoais são baseados na tecnologia de banco de dados padrão.

Ao projetar um aplicativo de banco de dados, você deve entender como os dados são estruturados. Com base nessa estrutura, você pode, então, projetar uma interface para exibir dados para o usuário e permitir ao usuário inserir novos dados ou modificar os existentes.

Drivers para acessar bancos de dados estão disponíveis para a maioria das plataformas de banco de dados para que o seu software possa usar a interface de programação (API) para recuperar as informações armazenadas em um banco de dados.

Banco de Dados Locais e Remotos

Servidores de banco de dados relacionais variam na forma como eles armazenam informações e, também, na maneira de permitir que vários usuários tenham acesso a essas informações simultaneamente. O RAD Studio fornece suporte para dois tipos de servidores de banco de dados relacionais:

Bancos de Dados Locais

Residem em sua unidade local. Eles têm muitas vezes APIs proprietárias para acessar os dados. Quando compartilhados por vários usuários, eles usam mecanismos de bloqueio com base em arquivo. Devido a essa característica, às vezes são chamados bancos de dados baseados em arquivos. As aplicações que utilizam bases de dados locais são chamadas de aplicativos de uma camada, porque a aplicação e o banco de dados residem no mesmo sistema de arquivos.

Bancos de Dados Remotos

Residem em uma máquina separada. Às vezes, os dados de um servidor de banco de dados remoto nem sequer residem em uma única máquina, mas são distribuídos ao longo de vários servidores. Embora os servidores de banco de dados remotos variem na forma como armazenam informações, eles fornecem uma interface lógica comum aos clientes.

Essa interface comum é a Structured Query Language (SQL). Devido ao fato de você acessá-los usando SQL, eles às vezes são chamados de servidores SQL (outro nome é o sistema de gerenciamento de banco de dados remoto, ou RDBMS).

Além dos comandos comuns que compõem SQL, a maioria dos servidores de banco de dados remotos suporta um "dialeto" único de SQL. Exemplos de servidores SQL incluem InterBase, Oracle, Microsoft SQL Server, Sybase, Informix, DB2, entre tantos outros.

As aplicações que utilizam servidores de banco de dados remotos são chamadas de aplicativos de duas camadas porque o aplicativo e o banco de dados operam em sistemas independentes.

Transações e Concorrência

Uma transação é um conjunto de ações que devem ser realizadas com sucesso em uma ou mais tabelas de um banco de dados antes de serem confirmadas (tornadas permanentes). Se qualquer uma das ações do grupo falhar, então todas as ações são revertidas (desfeitas). Em sistemas de banco de dados, as capacidades de lidar com transações permitem ao usuário garantir que a integridade de um banco de dados seja mantida.

Uma única operação pode exigir várias consultas, cada qual com leituras/gravações no banco de dados. Quando isso acontece, geralmente, é importante ter certeza de que o banco de dados vai completar todas as operações realizadas. Por exemplo, ao fazer uma transferência de dinheiro, se o dinheiro foi debitado de uma conta, é importante ele também ser creditado na conta de depósito. Além disso, as transações não devem interferir umas nas outras.

Uma simples transação, geralmente, é enviada para o sistema de banco de dados em uma linguagem, como SQL, da seguinte forma:

- Início da Transação (Begin Transaction);
- Executar várias consultas e atualizações (embora todas as atualizações do banco de dados não sejam visíveis para o mundo exterior até o momento);
- Consolidar a transação (Commit Transaction). Neste momento, as atualizações tornam-se visíveis se a operação for bem-sucedida.

Se uma das operações falhar, o sistema de banco de dados pode reverter toda a operação ou apenas a consulta que falhou. Este comportamento é dependente do SGBD em uso e de como ele está configurado. A transação também pode ser revertida manualmente a qualquer momento antes de sua confirmação.

Idealmente, o software de banco de dados deve cumprir as regras ACID (Atomicidade, Consistência, Isolamento, Durabilidade) que garantam sua total integridade.

Em caso de sucesso, a transação deve ter todas as suas operações executadas. Em caso de falha, nenhum resultado de alguma operação deve ser refletido sobre a base de dados. Isto é referido como Atomicidade;

A execução de uma transação deve levar o banco de dados de um estado consistente a um outro estado consistente, ou seja, uma transação deve respeitar as regras de integridade dos dados (como unicidade de chaves, restrições de integridade lógica, etc.). Isso é referido como a Consistência;

Em sistemas multiusuários, várias transações podem acessar simultaneamente o mesmo registro (ou parte do registro) no banco de dados. O isolamento é um conjunto de técnicas que tentam evitar que transações paralelas interfiram umas nas outras. Isso é referido como o Isolamento;

Os efeitos de uma transação em caso de sucesso (commit) devem persistir no banco de dados mesmo em casos de quedas de energia, travamentos ou erros. Garante que os dados estarão disponíveis em definitivo. Isso é referido como a Durabilidade.

Integridade Referencial, Stored Procedures e Triggers

Todos os bancos de dados relacionais têm certas características em comum, que permitem aos aplicativos armazenar e manipular dados. Além disso, bancos de dados, muitas vezes, fornecem outros recursos específicos que podem ser úteis para garantir relações consistentes entre as tabelas em um banco de dados.

Integridade Referencial

A integridade referencial fornece um mecanismo para impedir que relações mestre/detalhes entre as tabelas sejam quebradas. A integridade referencial é geralmente executada pela combinação de uma chave primária e uma chave estrangeira. Para que a integridade referencial funcione, qualquer campo de uma tabela que é declarada como uma chave estrangeira poderá conter apenas os valores do campo de chave primária de uma tabela pai. Quando o usuário tenta excluir um campo de uma tabela mestre, que resultaria em linhas filhas órfãs, regras de integridade referencial podem impedir sua supressão ou eliminar automaticamente os registros órfãos.

Stored Procedures

Os procedimentos armazenados são conjuntos de instruções SQL, nomeados e armazenados em um servidor SQL. Procedimentos armazenados podem executar tarefas comuns, relacionadas com o banco de dados no servidor, e, às vezes, retornar um conjunto de registros (conjuntos de dados).

Usos típicos para procedimentos armazenados incluem a validação de dados. Procedimentos armazenados utilizados para este fim são, frequentemente, chamados de gatilhos (ou triggers). Ou ainda podem encapsular uma API para algum processamento de dados grandes e complexos, tais como manipulação de um grande conjunto de dados para produzir um resultado resumido.

Triggers

Triggers são conjuntos de instruções SQL executadas automaticamente em resposta a um comando particular.

Gatilhos de banco de dados são iniciados quando um INSERT, UPDATE ou DELETE para uma tabela é executado. Um gatilho pode conter várias instruções SQL. Você pode usar gatilhos para verificar se os valores são adequados antes de introduzi-los em uma tabela ou para salvar os valores antes de mudá-los. Ao programar um gatilho, você pode acessar tanto os valores novos, quanto antigos de uma coluna em particular. Se a execução de um gatilho falhar, você pode usar ROLLBACK para redefinir o INSERT associado, UPDATE ou DELETE.

Suporte para Banco de Dados no RAD Studio

O RAD Studio possui uma série de Frameworks que permitem acesso aos mais diversos bancos de dados. Ao longo das diversas versões do produto, novas tecnologias foram introduzidas, sendo a mais atual e completa de todas o FireDAC.

Existem também frameworks de terceiros, entre produtos comerciais e de código aberto. Dentre as tecnologias pré-existentes no produto para acesso a dados, podemos citar:

- BDE: primeira tecnologia de acesso a dados do RAD Studio, foi amplamente utilizado no passado e, hoje, é uma tecnologia formalmente descontinuada. Aplicações utilizando este framework devem ser migradas para FireDAC.
- dbGo: representa uma camada de abstração para o ADO e, apesar de suportar diversas tecnologias de banco de dados, é normalmente aplicado apenas ao acesso de soluções baseadas em SQL Server.
- IBX (Interbase Expresss): trata-se de um conjunto de componentes que permite acesso nativo ao banco de dados Interbase. Possui ótima performance e é amplamente utilizado apenas para aplicações baseadas neste banco de dados.

- DBX (dbExpress): trata-se de um conjunto de drivers de banco de dados leve que fornecem acesso rápido a servidores de banco de dados SQL. Para cada banco de dados suportado, dbExpress provê um driver que se adapta ao software específico do servidor para um conjunto de interfaces dbExpress uniformes. Quando você implanta um aplicativo de banco de dados que usa dbExpress, você deve incluir uma DLL (o driver específico do servidor) na distribuição de sua aplicação.

FireDAC

FireDAC é um conjunto único e universal de componentes para acesso a dados, para desenvolver aplicações de banco de dados multidispositivo para Delphi e C++ Builder. Com sua poderosa arquitetura comum, FireDAC permite o acesso direto de alta velocidade nativa para InterBase, SQLite, MySQL, SQL Server, Oracle, PostgreSQL, IBM DB2, SQL Anywhere, Access, Firebird, Informix e muito mais.

Baseado em 10 anos de experiência em escrever drivers nativos para banco de dados, o FireDAC foi construído como uma camada de acesso poderosa que suporta todos os recursos necessários para construir aplicações reais de alta performance. FireDAC oferece uma API comum para acessar banco de dados de diferentes plataformas, sem comprometer o acesso a recursos específicos de um determinado banco de dados ou, ainda, seu desempenho.

Entre os inúmeros benefícios deste Framework, podemos citar:

- FireDAC é o fruto de 10 anos de experiência no desenvolvimento de aplicações de banco de dados reais;
- FireDAC permite que o desenvolvedor se concentre no desenvolvimento do aplicativo e não sobre as complexidades de interação com o banco;
- FireDAC fornece um moderno conjunto, rico em recursos de componentes para abordar todos os principais ambientes RDBMS;
- FireDAC usa um conjunto de componentes para tratar todos os tipos de RDBMS suportados;
- FireDAC reduz o custo total de propriedade.

Prática com Banco de Dados

Neste ponto, você já sabe o que é um banco de dados e alguns detalhes sobre sua estrutura. Assim, para melhor visualizar e entender como trabalhar com um banco de dados, você vai criar um banco de dados simples para testar as operações básicas, tais como inserir, editar e excluir.

Se você já tem o Delphi ou o C++ Builder instalados, então seu sistema já conta com uma edição do Interbase para desenvolvimento disponível. O Interbase possui diversas edições e suporte a múltiplas plataformas como iOS, Android, OS X, Linux, além do próprio Windows, distribuídas entre versões Server e embutidas. Para conhecer mais sobre todas as possibilidades, visite este link:

https://www.embarcadero.com/products/interbase

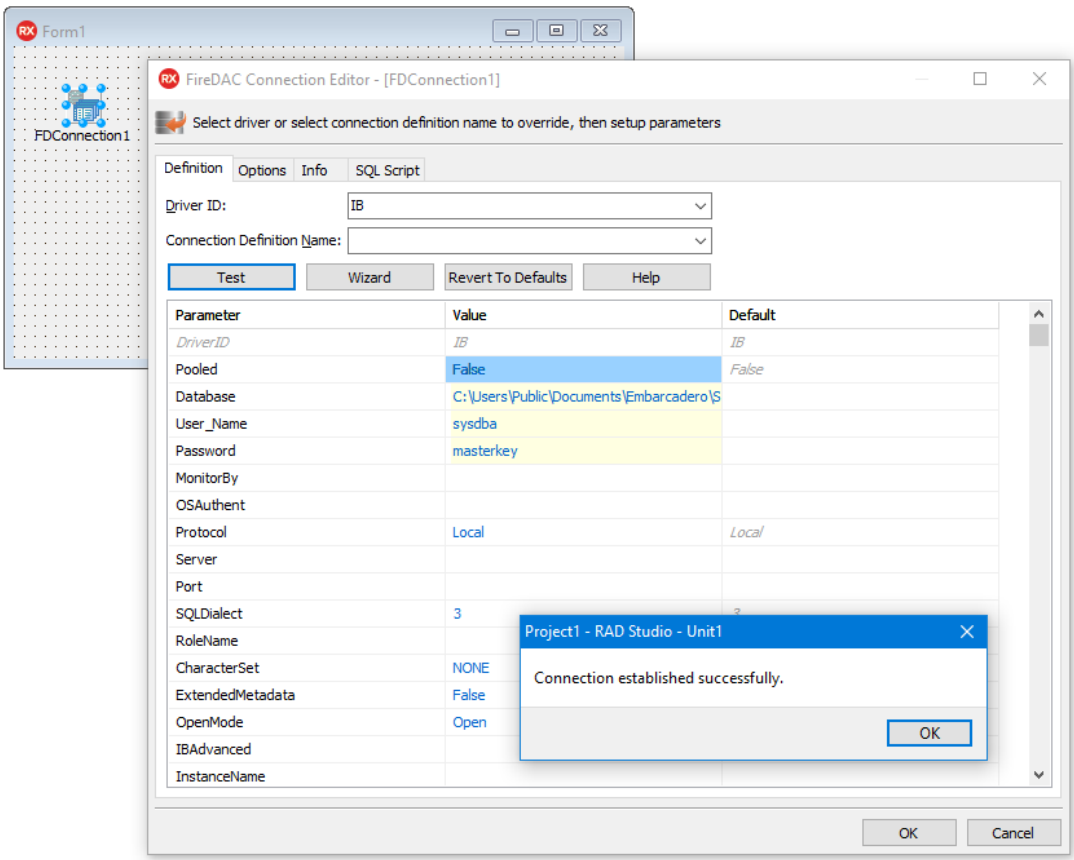
Inicialmente, verifique se o serviço do Interbase está devidamente ativo, através do gerenciador de serviços do Windows. Se tudo estiver OK, podemos seguir em frente!

Este tutorial está dividido em três partes principais:

- Estabelecer a conexão com o banco de dados: como usar Delphi ou C++ Builder para criar um aplicativo que irá conectar-se a um banco de dados;
- Selecionar linhas do banco de dados: ligar os dados até um *Gris* e exibi-los em tempo de design;
- Preparar sua aplicação para distribuição: descrever os passos necessários para fazer com que um aplicativo execute de maneira autônoma em qualquer computador.

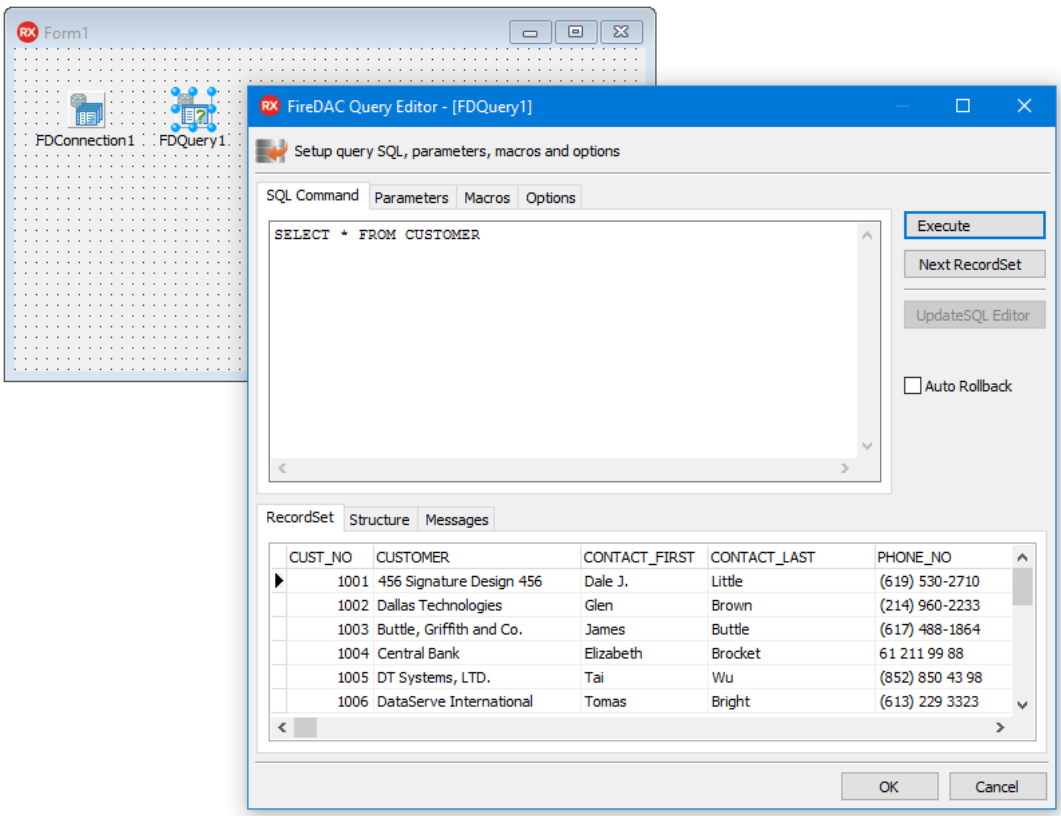
Inicie pela criação de uma nova aplicação do tipo “VCL Forms Application”. Arraste um componente TFDConnection para o formulário selecionado a partir da página "FireDAC" do Tool Palette. Este componente é responsável por estabelecer e controlar a conexão com o banco de dados. Através de um double-click sobre este componente, vamos definir a conexão com uma base de dados de exemplo do Interbase, que faz parte dos exemplos instalados em conjunto com o produto. Tudo o que você precisa fazer é:

- Selecionar o Driver "IB" (responsável pela conexão ao Interbase);
- Informar a propriedade *DataBase*, apontando-a para o banco de dados de exemplo "EMPLOYEE.GDB", localizado na pasta *Samples* de seu produto (normalmente a pasta *Samples* encontra-se abaixo de "Documentos" ou "Documentos Públicos", de acordo com o idioma de seu Windows);
- Informar as propriedades *User_Name* e *Password*. Por padrão, caso não tenha sido alterado durante a instalação do produto, o usuário e senha serão os mesmos apresentados na imagem abaixo;
- Se tudo estiver OK, ao clicar no botão Test, uma conexão com o banco de dados será estabelecida.



Vamos agora selecionar alguns dados para exibir em sua primeira aplicação com banco de dados:

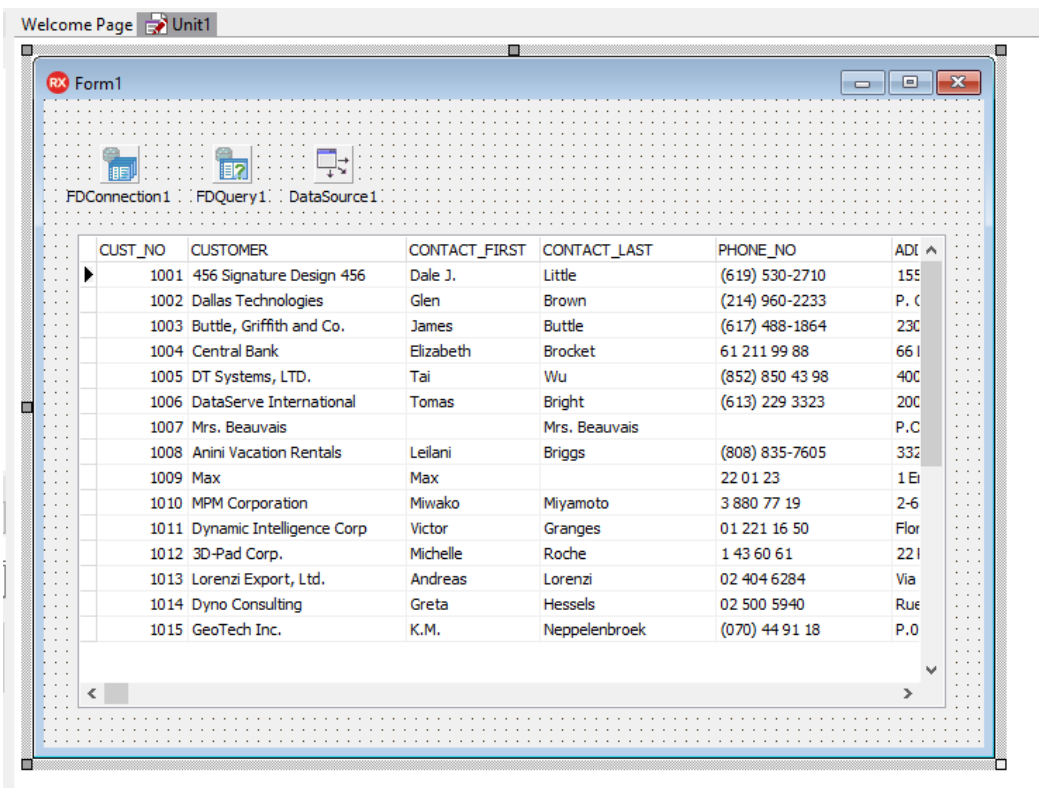
- Arraste um componente TFDQuery a partir da paleta "FireDAC" para o formulário. Este componente é responsável pela execução de comandos SQL, para buscar linhas do banco de dados e postar dados alterados de volta para o DB. Defina sua propriedade Connection para “FDConnection1”, caso não tenha sido automaticamente definida.
- Através de um *double-click* no componente “FDQuery1”, digite e teste o seguinte comando SQL:



- Se você observou dados sendo exibidos (similar à figura acima), podemos passar agora à configuração da porção visual de sua aplicação.

Para exibir os dados para seus usuários, vamos utilizar um componente do tipo Grid. Siga os seguintes passos para finalizar sua primeira aplicação com banco de dados:

- Localize e arraste para o formulário um componente "TDataSource" (paleta "Data Access");
- Configure sua propriedade DataSet para o "FDQuery1";
- Arraste um componente "TDBGrid" (paleta "Data Controls") para o formulário e configure sua propriedade DataSource para o "DataSource1";
- Finalmente, configure a propriedade Active do "FDQuery1" para **True**. Os dados serão exibidos no controle "DBGrid1".



Para encerrar, devemos preparar nossa aplicação para distribuição. Para tal, siga os seguintes passos:

- Arraste um componente "TFDPhysIBDriverLink" (paleta "FireDAC Links") para o formulário, este componente contém o driver para a conexão com o Interbase;
- Arraste um componente "TFDGUIxWaitCursor" (paleta "FireDAC UI") para o formulário, este componente é responsável por exibir o cursor apropriado no momento em que uma operação de banco de dados está sendo executada;
- Compile sua aplicação e ela está pronta para ser distribuída!

Com isso, encerramos este capítulo sobre aplicações com banco de dados. Para saber mais sobre FireDAC e como conectar outros bancos de dados suportados, você pode acessar esta página que é parte da documentação do produto:

http://docwiki.embarcadero.com/RADStudio/Berlin/en/Overview_(FireDAC)