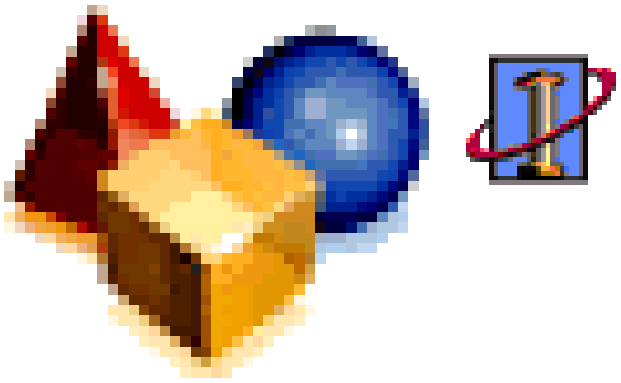


DELPHI 5

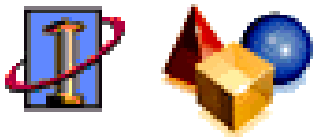
Escola Técnica Machado de Assis
Curso Técnico em Informática

Linguagem de Programação
Módulo I

Professor: José Luís Schiavo

SUMÁRIO

1 - Introdução



1.1 - Visão Geral do Delphi

Desde que a primeira versão do Delphi foi lançada, em 1995, esta ferramenta tem se mostrado como a melhor escolha no desenvolvimento para Windows. Numa relação com outros ambientes de programação, podemos dizer que o Delphi tem o poder do C++, e a facilidade do Visual Basic.

A principal vantagem do Delphi está na linguagem usada, Object Pascal, que é uma evolução do Pascal padrão. O Pascal surgiu no final dos anos 60 e, até hoje, é usada como uma das primeiras linguagens de programação para estudantes de computação. Em 1984, a Borland lançou o Turbo Pascal, que se firmou como o melhor compilador de Pascal do mercado e, a partir de então, passou a incluir novos recursos nesta linguagem, como Units e Objetos, até a ascensão do Windows, quando foi lançado o Turbo Pascal for Windows e, depois, o Borland Pascal, cuja linguagem é considerada a primeira versão da Object Pascal. Na sua atual versão, usada pelo Delphi, a Object Pascal é uma linguagem poderosa, sólida e respeitada, sem perder sua peculiar facilidade.

No Delphi, a criação de aplicativos começa com a montagem de componentes em janelas, como se fosse um programa gráfico, o usuário também pode utilizar componentes desenvolvidos por terceiros ou criar seus próprios componentes.

O Delphi vem com todas as ferramentas necessárias para a criação de bancos de dados dBase e Paradox, além de uma versão do Interbase, permitindo a criação de aplicativos com banco de dados sem a necessidade de aquisição de outro programa. O Delphi também tem acesso a bases de dados como Foxpro, Access, InFormix, SYBASE, Oracle, SQL Server e DB2, além de qualquer outro banco de dados para Windows compatível com ODBC.

1.2 - Principais Características

- Compilador/otimizador de código mais rápido do mercado, gerando executáveis rápidos e puros, sem run-time
- Baseado em componentes, com facilidade de criação de componentes nativos, além de controles ActiveX, inclusive com disponibilidade do código fonte dos componentes padrão
- Programação com a utilização de métodos visuais ou diretamente sobre o código
- Suporte a manipulação de exceções, que permite criar aplicações mais robustas e com maior segurança
- Acesso rápido e seguro a bancos de dados através do Borland Database Engine, com facilidades de manipulação
- Criação de relatórios no próprio executável, com utilização de componentes nativos
- Facilidade de upsizing para bancos de dados cliente/servidor
- Capacidade de criação de aplicações multi-tier, com objetos distribuídos
- Suporte a código in-line, em assembly
- Capacidade de criação de outros tipos de utilitários, como DLL's, Screen Saver's e aplicações CGI..
- Literatura diversificada
- Fluxo de programação baseado em eventos

1.3 - Versões

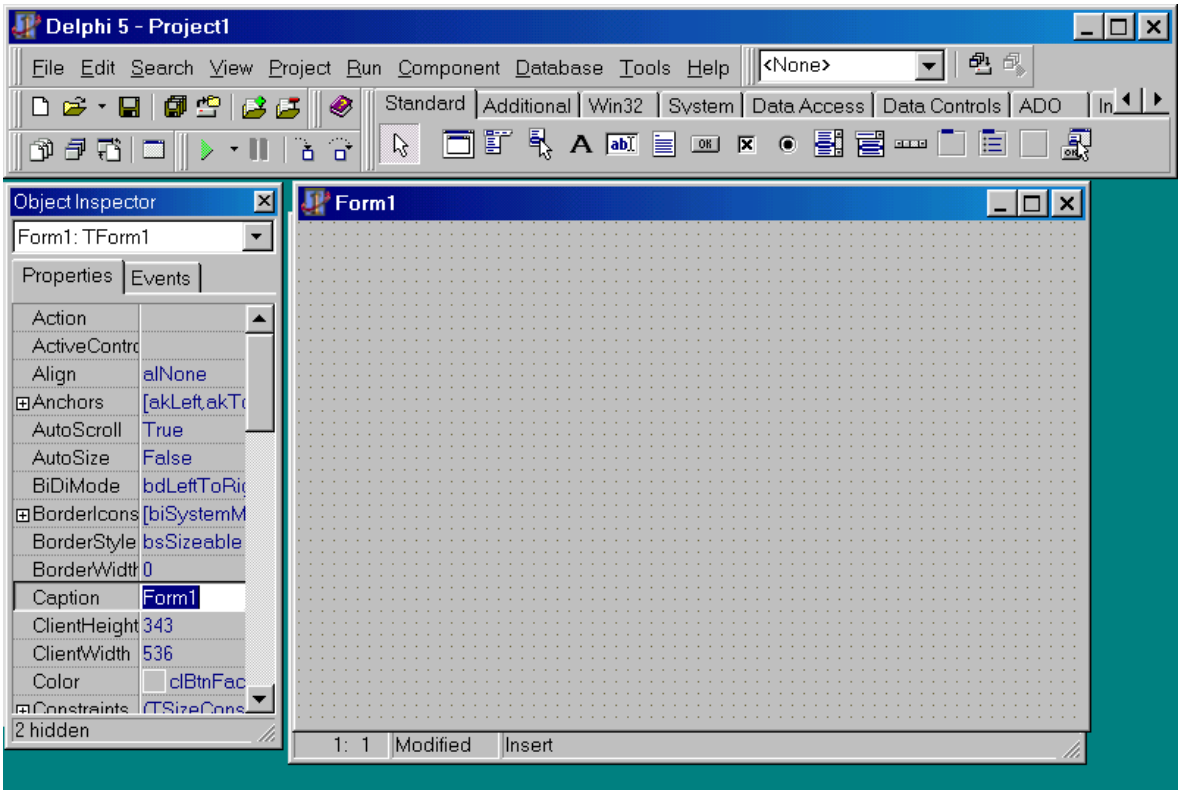
- Delphi 5 Standard, para estudantes, com poucas ferramentas de apoio
- Delphi 5 Professional, com a maioria das ferramentas, mas sem suporte a arquitetura cliente/servidor
- Delphi 5 Enterprise, versão completa, com todas as ferramentas de apoio

2 - O Ambiente de Programação

2.1 - Janela Principal

A janela principal do Delphi é composta pela barra de menus, barra de ferramentas e paleta de componentes. Para personalizar a barra de ferramentas, basta clicar em Properties no menu de contexto. Na paleta de componentes estão os controles usados nas aplicações, agrupados em guias por categorias.

Logo que você inicia o ***DELPHI***, sua tela ser parecerá com a seguinte:



2.2 - Barra de Menus

Você pode executar um comando no ambiente ***DELPHI*** de três maneiras:

1. Usando a barra de menus.
2. Usando a ***SpeedBar*** (Barra de Ferramentas).
3. Usando um ***SpeedMenu*** (Menu Rápido acionado através do clique com o botão direito do mouse).

2.3 - Menu File

No Menu File estão reunidas as funções de tratamento de arquivos. Estas funções estão divididas em três grupos, os dois primeiros relacionados ao Projeto (New, New Application, New Form, New Frame, Open, Open Project, Reopen), e outro de funções relacionadas aos Arquivos de Código-fonte (Save, Save As, Save Project As, Save All, Close, Close All, Use Unit, Print e Exit).

2.4 - Menu Edit

Além dos comandos característicos de um Menu Edit (Undo, Redo, Cut, Copy, Paste, etc.), este possui uma série de comandos relacionados aos componentes dispostos ao Formulário (Align to Grid, Bring to Front, Send to Back, Align, Size, Scale, Tab Order, Creation Order, Lock Controls).

O comando Lock Controls trava os componentes do formulário, impedindo que se altere o posicionamento deles.

2.5 - Menu Search

O Menu Search possui comandos para a localização e/ou substituição de trechos de texto no arquivo de código-fonte. Um comando interessante é o Incremental Search, que permite que você procure diretamente na área de mensagem do editor. Quando você digita a primeira letra, o editor se desloca para a primeira ocorrência desta letra, depois para a segunda, e assim por diante.

2.6 - Menu View

Mais da metade dos comandos do Menu View servem para exibir as janelas do ambiente DELPHI, como por exemplo: Project Manager, Object Inspector, Alignment Palette, etc.
Os comandos Toggle Form / Unit, Units... e Forms... são utilizados para alternar entre um formulário e o editor de código-fonte, New Edit Window duplica o editor de código-fonte permitindo assim que se possa visualizar duas ou mais páginas de código ao mesmo tempo, já que elas (as janelas) não são sincronizadas. Os dois últimos comandos (SpeedBar e Component Palette) são usados para mostrar / ocultar a barra de ferramentas e a paleta de componentes do DELPHI.

2.7 - Menu Project

Possui comandos de manipulação e compilação de um projeto. Compile e Build All constróem o executável da aplicação com a diferença de que Compile só verifica os códigos que foram alterados e Build All verifica todos. Para somente verificar a sintaxe dos arquivos-fonte do seu projeto utilize Sintaxe Check, ele não cria o executável.
Comando Information... exibe informações sobre a última compilação do projeto. Por fim, o comando Optinons... configura as opção do projeto, tais como: título da aplicação, opções de compilação e linkagem, entre outras.

2.8 - Menu Run

O Menu Run possui comandos para executar e depurar a aplicação. Você pode executar passo a passo, verificar variáveis e objetos, determinar paradas; permitindo assim um maior controle sobre a execução de sua aplicação a procura de possíveis falhas que podem estar ocorrendo.

2.9 - Menu Component

Uma das principais características do Menu Component é customizar a paleta de componentes através das opções de criação (New...), instalação (Install...), e da paleta de componentes (Configure Palette). A opção Create Componente Template customiza um componente e salva-o como um modelo (Template) com novo nome, página na paleta e ícone.

2.10 - Menu Database

Como o próprio nome já diz, este menu agrupa as ferramentas de banco de dados do DELPHI. Dependendo da edição do DELPHI (desktop, professional ou enterprise), este menu pode possuir mais ou menos comandos: Explore, SQL Monitor e Form Wizard.

2.11 - Menu Tools

Relaciona uma série de ferramentas e programas para facilitar o trabalho, tais como: Image Editor, Database Desktop e Collection Editor e outros.

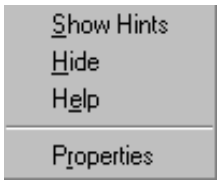
2.12 - Menu Help

O Menu Help trás várias opções: as três primeiras chamadas as Janelas Tópicos de Ajuda e DELPHI Help já posicionadas em uma determinada opção. As seguintes acessam as páginas da Borland na Internet (caso você esteja conectado). E, finalmente, About consiste apenas em uma janela descrevendo os créditos do DELPHI.

2.13 - Barra de Ferramentas (SpeedBar)

A barra de ferramentas do DELPHI oferece uma forma rápida e fácil de acessar os principais comandos do ambiente. Ela é totalmente customizável, permitindo que o usuário inclua ou exclua botões nela. Usando a técnica do drag-drop, arraste um ícone da caixa de diálogo para a ***SpeedBar*** para incluí-lo, e para fora da ***SpeedBar*** para removê-lo. Da mesma forma, você pode rearranjar os ícones na própria ***SpeedBar*** inserindo espaços para agrupar os botões, o que facilita seu trabalho.

2.14 - SpeedMenus



SpeedMenus são menus de contexto que permitem trabalhar com um elemento específico da interface. Você acessa um SpeedMenu pressionando o botão direito do mouse sobre o elemento ou através das teclas <ALT><F10>.

2.15 - Paleta de Componentes

A Paleta de Componentes contém as ferramentas que você usa para desenhar os componentes no formulário. Cada ferramenta corresponde a um componente, e estes estão agrupados por função ou tipo de utilização.

Página	Componentes
Standard	Componentes padrão do Windows, tais como: *Button, Label, Listbox, Panel,* etc.
Additional	Componentes especializados do Windows, como: *Speed Button, Mask Edit, Image, Bitbtn,* etc.
Win32	Componentes característicos de aplicações 32-bits. Exemplos: *TabControl, PageControl, ImageList,* etc.
System	Componentes pertencentes ao Windows System Tecnology. Exemplos: *Timer, PaintBox, MediaPlayer*, etc.
Internet	Oferece uma variedade de protocolos de acesso à internet (disponível somente na versão *Enterprise*). Exemplos: *ClientSocket, ServerSocket, WebDispatcher, PageProducer*, etc.
Data Access	Componentes para acesso a banco de dados, como: *Data Source, Table, Query*, etc.
Data Controls	Componentes utilizados para visualizar informações em bancos de dados como, por exemplo: *DBGrid, DBNavigator, DBListBox* e mais.
Decision Cube	Refere-se a um conjunto de componentes para suporte à decisão que permitem gerar gráficos de tabelas-cruzadas para obter visualizações dos dados a partir de perspectivas variantes, tais como: *DecisionCube, DecisionQuery, DecisionPivot*, etc.
Oreport	Componentes utilizados para criação de relatórios. Por exemplo: *QuickRep, QRGroup, QRShape*, etc.
Dialogs	Caixas de Diálogo padrão Windows, como: *OpenDialog, SaveDialog, PrintDialog, FontDialog, OpenPictureDialog,* etc.
Win 3.1	Componentes utilizados para compatibilidade com aplicações Windows 3.1. Exemplos: *DBLookupList, TabSet, NoteBook,* etc.
Samples	Exemplos de componentes customizados que você pode criar e adicionar à paleta. Exemplos: *Gauhe, ColorGrid, SpinEdit,* etc.

ActiveX	Objetos *ActiveX*, tais como: *Chaftx, VSSpell, FLBook, VtChart* e *Graph*.

2.16 - Object Inspector

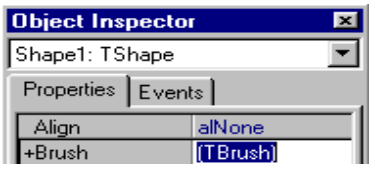
No Object Inspector podemos manipular, em tempo de projeto, as propriedades e eventos dos componentes. Você também pode selecionar um componente usando o Seletor de Objetos, no topo do Object Inspector.

Quando mais de um componente está selecionado, somente as propriedades e eventos comuns a todos são exibidos no Object Inspector. E em caso de alterações, elas serão válidas para todos os componentes selecionados.

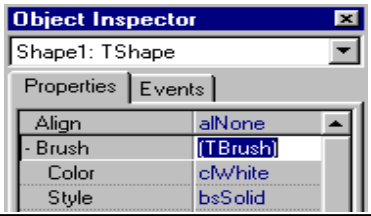
Para selecionar mais de um componente ao mesmo tempo, mantenha pressionada a tecla ***<SHIFT>*** enquanto clica sobre os demais componentes, ou arraste um retângulo envolvendo os componentes que deseja selecionar.

Para alterar uma propriedade, basta que você dê um clique na área a direita do nome da propriedade e informe o novo valor. Dependendo da propriedade, a maneira de informar esse valor pode mudar. Por exemplo: para configurar a propriedade ***Font*** de um botão, você pode clicar sobre o botão com reticências que lhe será apresentada a caixa de diálogo ***Fonte*** padrão; outro exemplo é a propriedade ***Cursor*** do mesmo botão que pode ser configurada através de uma caixa combo que lista todos os valores possíveis para esta propriedade.

Algumas propriedades podem ser expandidas em subpropriedades que podem ser configuradas individualmente. Por exemplo: a propriedade Brush de um componente Shape possui duas subpropriedades (Color e Style) que devem ser informadas caso se queira alterar a cor ou o estilo do objeto de desenho. Uma propriedade com subpropriedades pode ser identificada pelo Sinal de Mais (+) à esquerda do seu nome. Para expandir este tipo de propriedade, dê um clique duplo sobre o seu nome.



Propriedade ***Brush*** **do componente *Shape***



Propriedade ***Brush*** **expandida permitindo visualizar as**

A página ***Events*** lista todos eventos aplicáveis ao(s) componente(s) selecionado(s). Quando for necessário fazer o tratamento, através do código de um determinado evento, deve-se selecionar o componente para o qual o evento acontece, mudar-se para a página ***Events*** no Object Inspector e dar um clique duplo na área em branco à direita do nome do evento que se quer tratar. O DELPHI abre o editor de código já posicionado na procedure que trata esse mesmo evento.

Uma outra forma de abrir o editor de código é dar um clique duplo sobre o componente. Neste caso o editor de código abre-se no evento padrão do componente.

💣 Para que a janela do Objetct Inspector fique sempre visível, clique com o botão direito sobre ela e escolha a opção Stay on top no SpeedMenu, Mesmo que outra janela seja arrastada sobre o Object Inspector, ele sempre ficará por cima.

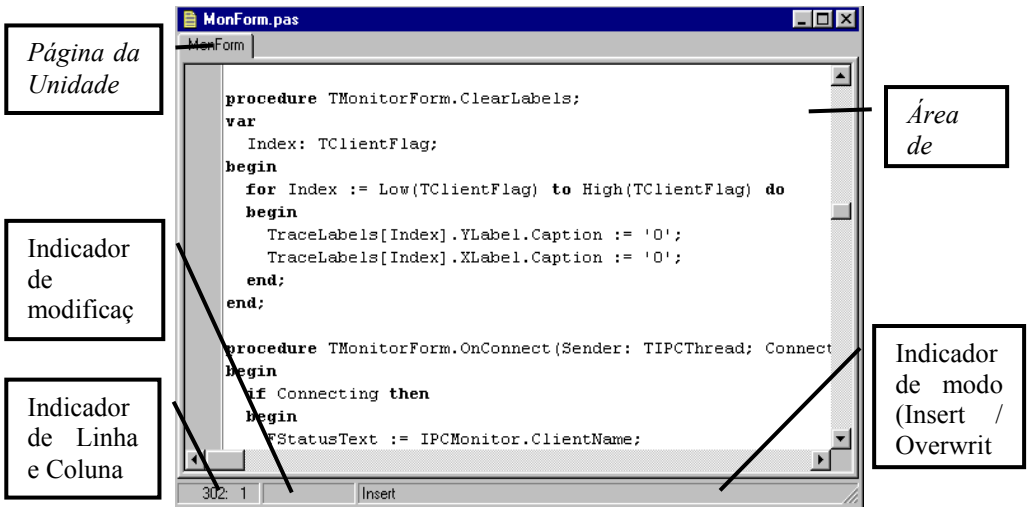
2.17 - Form Designer

O Form Designer é onde são desenhados os Forms das aplicações, com a inserção de componentes. No menu de contexto do Form, você pode clicar em View as Text para editar a descrição textual do Form e de seus componentes no Editor de Código, essas informações são

gravadas em um arquivo binário com a extensão DFM, para voltar ao modo de exibição normal, escolha View as Form no menu de contexto do Editor de Código.

2.18 - O Editor de Códigos

Para escrever o código, usamos o Editor de Código do Delphi. Para cada Form é criado um código, que é gravado em arquivos chamados Units, nesses arquivos é definida a classe do Form e seus métodos de eventos. Para alternar entre o Form e sua Unit podemos clicar em Toggle Form/Unit no menu View, ou no botão corresponde da Barra de Ferrramentas, ou ainda, usar a tecla F12. Para cada Form aberto é criado um Form Designer e uma nova guia no Editor de Código.

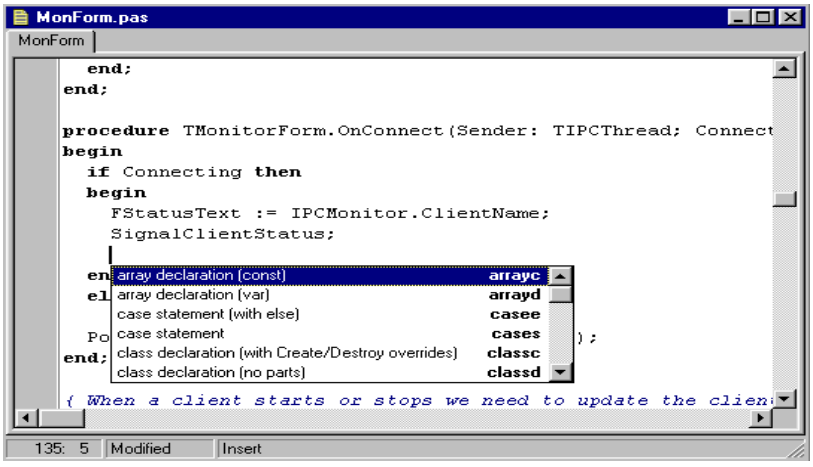


Uma característica interessante do ***Editor de Códigos*** é o fato dele destacar partes da sintaxe com cores.

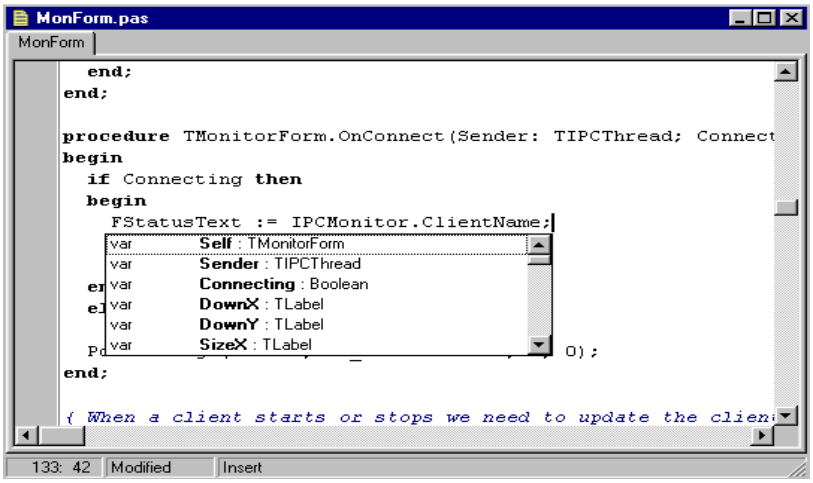
2.19 - Code Insight

Numa tradução grosseira, Code Insight significa Código Perspicaz. E é essa mesma a idéia dos Code Insight, ele oferece modelos, complementação de código, parâmetros de métodos e funções, e avaliador de expressões. Para configurar o Code Insight, escolha Evironment Options no Menu Tools e selecione a ficha Code Insight.

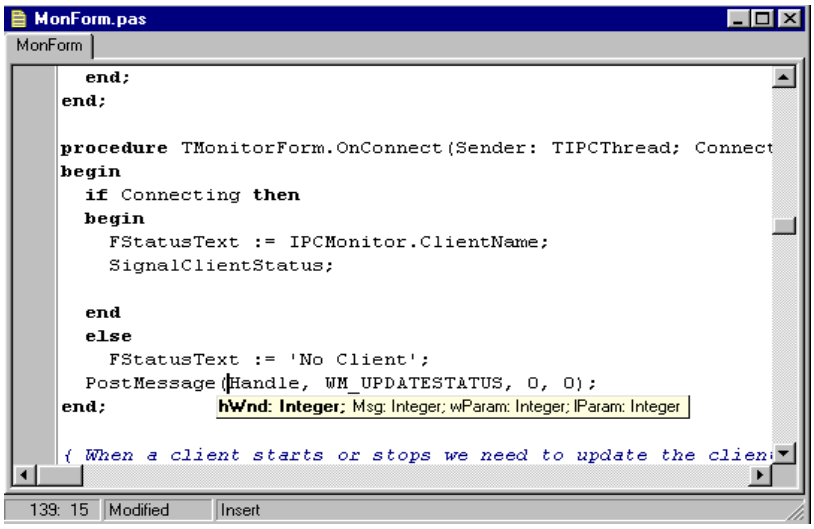
- **Code Templates:** quando estiver digitando o código, você pressiona ***<CTRL><J>*** e obtém uma lista de ***Code Transplates*** (modelos de códigos) para escolher. O editor digita o ***Template*** para você. Ë possível também criar seus próprios modelos e adicioná-los à lista.



- **Code Completion:** se você digita um nome de objeto e depois o ponto preparando-se para digitar um método ou propriedade, um ***Listbox*** aparece com todas as possibilidades disponíveis. Você pode invocar a lista com ***<CTRL><SPACE>***.



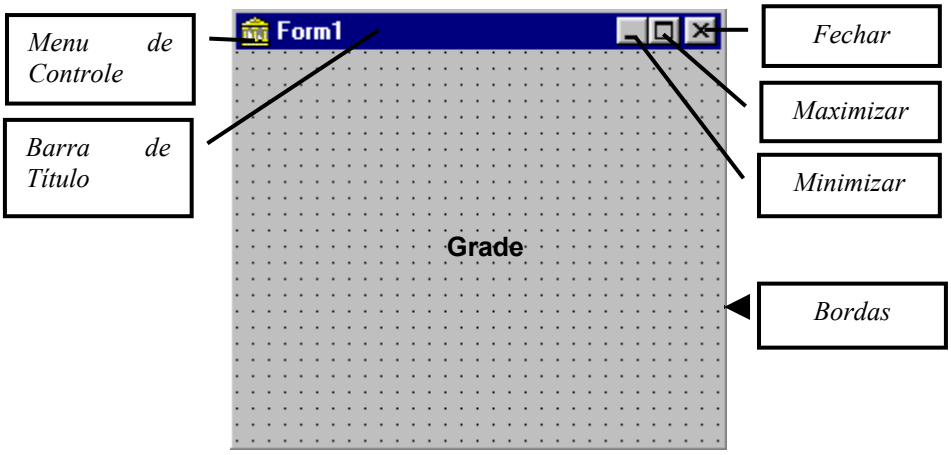
- **Code Parameters:** digite um nome de procedure, função ou método e abra parênteses, uma tarja amarela ***(ToolTip)*** surgirá mostrando a lista de parâmetros como ela aparece na declaração das rotinas: nome, tipo e qualquer modificador. O parâmetro de que você necessita estará em negrito e, à medida que você entra com mais parâmetros, ele se move.



- **Expression Evaluator:** quando o depurador está no controle da aplicação, você posiciona seu mouse sobre expressões envolvendo variáveis ou propriedades e o valor aparece em um ***ToolTip*** (tarja amarela).

2.20 - A Janela de Formulário

O formulário é o elemento principal para a criação da interface com o usuário. Quando o ambiente de trabalho é iniciado ou inicia-se uma nova aplicação, um formulário padrão é criado pelo Delphi, com o nome de **Form1** a partir do qual você pode começar a construir a interface visual do seu programa.



O formulário em tempo de projeto, possui um menu suspenso que aparece quando o botão direito do mouse for clicado sobre qualquer área dele, inclusive sobre seus componentes. Alguns componentes podem incluir comandos neste menu para permitir sua melhor configuração em tempo de design.

Como qualquer janela de uma aplicação desenvolvida para ambiente windows, uma janela de formulário apresenta:

- Uma barra de títulos.
- Um menu de sistema.
- Botões de Minimização, Maximização e Finalização.
- Uma área cliente, na qual serão incluidos os controles e componentes.

Este Formulário possui uma grade, representada por pontos dispostos no Formulário. Para aumentar ou diminuir o tamanho da grade (distância entre os pontos) faça o seguinte:

1. Escolha a opção ***Environment Options...*** no ***Menu Tools***;
2. Selecione a página ***Preferences***;
3. Digite os valores nas caixa de diálogo: ***Grid size X*** e ***Grid size Y***.

Quanto menor o valor digitado, menor será a distância entre os pontos e vice-versa.

2.21 - Configuração do Ambiente

Grande parte das opções de configuração do ambiente podem ser acessadas através do item Environment Options do menu Tools. A maioria das opções desse diálogo são bastante claras e através delas podemos definir, desde as opções do Form Designer, até o Editor de Código e o caminho das bibliotecas. No menu Tools, podemos escolher também Configure Tools, para permitir abrir aplicações externas a partir do ambiente do Delphi, como o Image Editor e o Database Desktop.

2.22 - Estrutura de Projetos

Um projeto em Delphi é dividido em módulos, chamados Units, seguindo a estrutura de arquivos descrita na tabela abaixo.

Extensão	**Descrição**
DPR	Arquivo de projeto, onde são indicados as Units e o código de inicialização do programa
PAS	Código fonte de uma Unit do projeto
DCU	Unit compilada
DFM	Definição visual de um Form. O código fonte está em uma Unit com o mesmo nome
DOF	Opções de configuração para o projeto
RES	Recursos do projeto, com o ícone do programa
~PA, ~DF, ~DP	Arquivos temporários (backup)
DSK	Configurações de Desktop

2.23 - Project Manager

Para ajudar no gerenciamento de projetos, podemos usar o Project Manager pelo menu View. O Project Manager lista as Units, os Forms existentes nessas Units e o path, se a Unit não estiver na pasta do projeto. Através dos botões do Project Manager você pode adicionar, excluir e visualizar Units e Forms que compõem o projeto.

2.24 - Project Options

Através do item Options, do menu Project, podemos escolher diversos aspectos de um projeto:

Forms

Nessa página, podemos definir o Form principal da aplicação e a os Forms que serão criados automaticamente. Se um Form não for criado automaticamente, você terá que instanciar esse Form explicitamente.

Application

Nessa página podemos definir o título, o arquivo help e o ícone da aplicação.

Compiler

Usamos essa página para definir as opções de compilação, para o projeto atual. Essas opções irão interferir diretamente no executável gerado.

Linker

Essa página é muito pouco usada, mas somente através dela podemos modificar a memória exigida por uma aplicação.

Directories/Conditionals

Aqui você pode especificar pastas de saída para os arquivos gerados na compilação do projeto e opções de compilação condicional.

Version InFormation

InFormações da versão do executável.

Packages

Nesta página você pode especificar parte do código para ser incluído em *Packages*, fora do executável, permitindo compartilhamento de componentes entre várias aplicações Delphi.

2.25 - Gerenciamento de Projetos

Segue uma descrição das mais importantes opções de menu para o gerenciamento de projetos, algumas dessas opções tem um botão correspondente na barra de ferramentas.

File	
New	Abre um diálogo com novos itens que podem ser adicionados ao projeto
Open	Abrir projetos, pode abrir também Units, Forms e texto no editor de código
Save	Salva o arquivo aberto no editor de código
Save Project As	Salva o projeto com outro nome ou local
Use Unit	Faz com que a Unit atual possa usar outra Unit do projeto
Add to Project	Adiciona uma Unit em disco ao projeto
Remove from Project	Remove uma Unit do projeto
View	
Project Manager	Mostra o gerenciador de projeto
Project Source	Mostra o código do projeto (programa principal)
Object Inspector	Mostra o Object Inspector
Toggle Form/Unit	Alterna entre o Form e a Unit
Units	Mostra o código fonte de uma Unit ou do Projeto a partir de uma lista
Forms	Seleciona um Form a partir de uma lista
Project	
Compile	Compila o projeto
Options	Opções do projeto, como ícone do executável, nome da aplicação e opções de compilação

Run	
Run	Compila e executa o projeto

2.26 - Ajuda

O sistema de ajuda do Delphi é a referência mais completa, seguida pelos manuais do usuário cedidos com o sistema. O Help do Delphi, como a maioria dos Helps Windows, utiliza-se da tecnologia de hipertextos que permite uma leitura não seqüencial do texto. Os hipertextos funcionam da seguinte forma: quando aparecer uma palavra-chave (geralmente na cor verde), podemos clicar sobre ela para obter explicações sobre aquele determinado termo, e assim por diante.

Para acessar o Help do DELPHI escolha Contents no Menu Help, ou pressione a tecla <F1>. A ajuda obtida através da tecla <F1> é sensitiva ao contexto, isto é, no momento em que você a pressionar um objeto ou termo que estiver selecionado, o Help exibirá um texto explicativo sobre ele. O mesmo pode ser feito com propriedades e eventos, no Object Inpector e comandos, no editor de código.

Apesar do Help do DELPHI ser em inglês, você encontrará muitos exemplos que o ajudarão a solucionar problemas que surgirão durante o desenvolvimento de suas aplicações.

3 - Visão Geral dos Componentes

Os *Componentes* são a base da criação de aplicativos DELPHI. Pode-se dizer que Componentes são objetos disponíveis na paleta de componentes que podem ser conectados diretamente à aplicação. Estes componentes foram desenvolvidos visando a sua utilização em qualquer tipo de aplicação, bastando apenas configurar suas propriedades ou ainda, adaptá-los à aplicação de acordo com as necessidades do programador.

A maior ***"Sacada"*** dos Componentes é a facilidade de serem reutilizados, ou seja, sempre que um determinado componente for inserido no formulário é como se fosse feita uma cópia do mesmo tornando-a parte integrante da aplicação que está sendo construída. Cada componente possui um conjunto de características próprias, mas também, muitas delas são comuns a vários outros componentes como no caso dos componentes visuais, todos possuem as propriedades *width e height* as quais determinam o seu comprimento e a altura, respectivamente. O Delphi também nos permite adicionar novas características a estes componentes permitindo uma melhor adaptação dos mesmos às nossas necessidades, mas para se aventurar neste campo é necessário que o programador tenha conhecimentos mais profundos de programação orientada a objetos, no caso do Delphi o Object Pascal.

Os Componentes estão organizados hierarquicamente em uma estrutura chamada VCL (Visual Component Library). Nem todos os Componentes estão disponíveis na Paleta de Componentes, é o caso do Componente TApplication que encapsula uma Aplicação Windows.

Todos os Componentes são objetos, mas nem todos os objetos são Componentes. Por exemplo, as classes TPrinter, TClipboard e TFont são objetos constantes da VCL e não são Componentes.

3.1 - Componentes Visuais

De uma maneira bem simplificada, Componentes Visuais são todos aqueles que aparecem em tempo de execução. São também chamados de Controles, quando em tempo de projeto você insere um Controle em seu formulário ele se apresentará da mesma forma que em tempo de execução.

Normalmente, os termos Componentes e Controle são confundidos, mas nem todo Componente é um Controle.

3.2 - Componentes Não-visuais

Os Componentes Não-Visuais não são controles e não aparecem, eles mesmos em tempo de execução, mas gerenciam algo que é visual. Por exemplo: o Componente TSaveDialog é usado para exibir uma Caixa de Diálogo Salvar Como na tela. Eles são representados em tempo de projeto por um pequeno ícone o qual deve ser disposto no formulário. São exemplos de Componentes Não-Visuais: TTimer, TSaveDialog, TOpenDialog, etc.

3.3 - Propriedades

Uma *Propriedade* é um atributo de um objeto e define um estado dele. Por exemplo, uma *Propriedade* pode determinar qual será a posição do objeto dentro do formulário, outra pode definir qual será sua cor, etc. Além disso, *Propriedades* determinam também comportamentos para o objeto, exemplo: determinada propriedade diz se o objeto estará habilitado ou não.
Uma *Propriedade* pode ser: somente de leitura, somente de escrita, de tempo de projeto, e/ou de tempo de execução, podendo ser usada, dependendo do caso, em operações de leitura e gravação, e expressões, mas não pode ser passada como parâmetro.
Algumas *Propriedades* executam *Métodos* quando seus valores são alterados, por isso não se pode confundi-las com *Campos de Dados* (os quais simplesmente guardam um determinado dado e nada mais).

3.4 - Métodos

Um *Método* pode ser um procedimento ou uma função relacionada a um objeto que executa uma ação. Por exemplo, executamos o *Método SetFocus* para um controle quando queremos que receba o foco de entrada. Alguns *Métodos* bastante comuns são: *Show, Hide,* etc.

3.5 - Eventos

Um *Evento* pode se gerado de diversas maneiras:

- Por uma ação praticada pelo usuário – quando ele clica com o mouse sobre um componente, está gerando o *Evento* ***OnClick***; quando passa com o cursor do mouse sobre um componente dispara o *Evento* ***OnMouseMove***, etc.
- Pelo sistema – quando executamos um método ou alteramos uma determinada propriedade, exemplo: ao mudarmos o foco de um componente para outro, um perde o foco e executa o *Evento* ***OnExit*** e outro ganha o foco e executa o *Evento* ***OnEnter***.

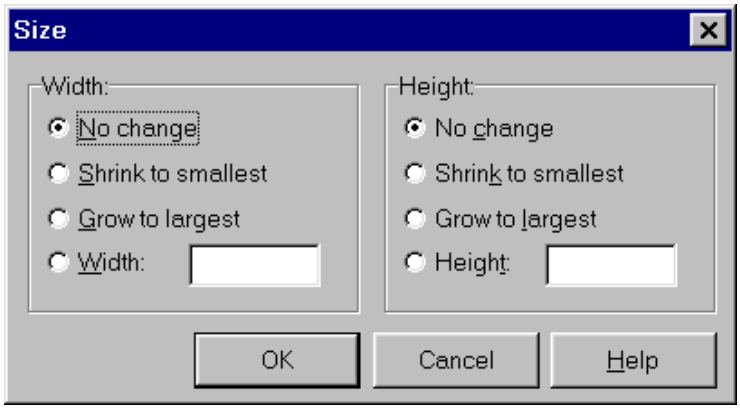
Um *Evento* pode ser executado como uma mensagem ***Windows*** enviada a um componente. Este componente então responde a esta mensagem.

3.6 - Selecionando Componentes

Você pode selecionar um componente, em tempo de projeto, simplesmente clicando sobre ele. Se desejar selecionar mais de um componente ao mesmo tempo, clique do segundo elemento em diante com a tecla ***<SHIFT>*** pressionada.
Caso os elementos a serem selecionados estejam todos numa mesma região do formulário, é possível arrastar o mouse sobre o formulário desenhando um retângulo que abranja todos os elementos que ser quer selecionar.
Para selecionar o formulário, basta clicar numa área vazia dele, isto é, em qualquer posição que não possua um outro componente.

OBS: Qualquer componente pode ser selecionado a partir da caixa combo Object Selector, que fica na parte superior do Object Inspector.

3.7 - Redimensionando um componente



Quando você seleciona um objeto, aparecem quadrinhos ao seu redor, que chamamos de alças (gerenciadores) de dimensionamento. Arraste um desses gerenciadores e as dimensões do componente se alterarão.
Utilize a combinação de teclas ***<SHIFT><SETA DIREÇÃO>*** para alterar o tamanho do componente selecionado sem o uso do mouse. Normalmente esta técnica é utilizada quando se quer uma maior precisão no redimensionamento do componente.
Podemos também redimen-sionar um componente através da caixa de diálogo *Size*, que é acessada através do comando ***Size*** do ***Menu Edit***.

3.8 - Movendo um Componente

Para movimentar um componente utilize a técnica de drag-drop (arrastar-soltar), em que você clica sobre o componente, mantém o botão pressionado enquanto arrasta o mouse, e solta o componente na posição desejada.
A combinação das teclas ***<CTRL><SETAS DIREÇÃO>*** também movimente o componente selecionado, acompanhando a grade do formulário.

3.9 - Apagando Um Componente

Para apagar um componente do formulário, basta selecioná-lo e pressionar a tecla ***<DEL>*** ou ***<DELETE>***. Se mais de um componente estiver selecionado, todos serão eliminados ao mesmo tempo.
Quando eliminamos um componente que já possua código associado a qualquer dos seus eventos, esse código permanente na *Unit* até que você o apague diretamente no *Editor de Códigos*.

3.10 - Padronizando Prefixos em Nomes de Componentes

Quando você adiciona um ***Componente (ou Formulário)*** ao ***Projeto***, o *DELPHI* automaticamente configura sua propriedade ***Name*** para um valor padrão. Por exemplo: toda vez que você insere um ***Botão*** no formulário, sua propriedade ***Name*** é configurada para ***Button*n**, onde **n** é ***1, 2, 3***, etc.
À medida que são colocados os eventos para um determinado componente, o *DELPHI* utiliza o seu nome para compor os nomes das procedures que tratam esses eventos. Exemplo: o evento ***OnClick*** de um botão chamado ***Button1*** inserido em um formulário chamado ***Form1*** possuiria uma procedure de nome ***TForm1.Button1Click***. Notamos claramente que estes nomes são inadequados para se identificar os componentes, nem mesmo a pessoa que criou esta aplicação saberia dizer para que serve esta procedure uma semana após tê-la criado.
Agora imagine que essa mesma pessoa tenha nomeado o formulário para ***frmClientes*** e o botão para ***btnIncluir***. Então a procedure ***OnClick*** passaria a se chamar ***frmClientes.btnIncluirClick***. Muito mais claro, não !?

Visando a facilitar a legibilidade do *Código*, criaremos uma padronização para os nomes dos *Componentes*. Lembre-se de que todo componente possui um nome que é atribuído à sua propriedade ***Name***.

Nosso padrão consistirá em informar um prefixo de três (03) letras minúsculas no início do nome do *Componente*. Assim, um *Componente* ***TButton*** poderia ser nomeado para ***bntOK***, um *componente* ***TEdit*** para ***edtNome***, e assim por diante.

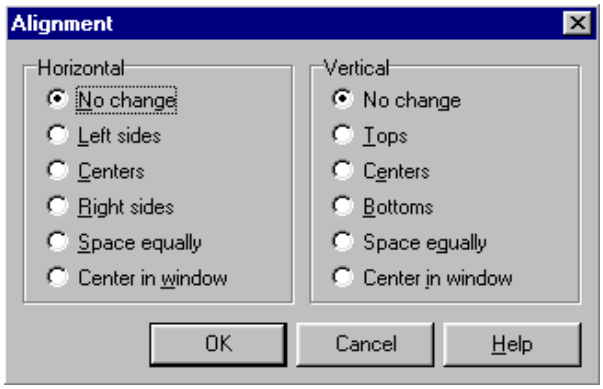
Segue tabela com prefixos de alguns ***Componentes***:

COMPONENTE	*PREFIXO*	*EXEMPLOS*
Tbutton	**btn**	btnFecha, btnOK, btnCancela, etc.
Tedit	**edt**	edtNome, edtValor, edtNasc, etc.
Tshape	**shp**	shpFigura, shpDesenho, etc.
TradioButton	**rbt**	rbtLigado, rbtDesligado, etc.
TgroupBox	**grp**	grpOpcoes, grpEstado, etc.
Tform	**frm**	frmClientes, frmGeo, frmLista, etc.
TcheckBox	**chk**	chkNegrito, chkItalico, etc.
Tlabel	**lbl**	lblNome, lblValor, lblNasc, etc.
TlistBox	**lst**	lstNomes, lstPaises, etc.
TscrollBar	**scr**	scrHorizontal, scrCores, etc.
TspeedButton	**sbt**	sbtGrava, sbtImprime, etc.

A padronização é importantíssima em programação, principalmente no trabalho em equipes. Padrões devem ser criados e estabelecidos para facilitar a comunicação entre os membros de uma equipe de programação para facilitar a manutenção do código.

3.11 - Alinhamento dos Componentes no Formulário

Para alinhar os componentes dentro do formulário, você pode utilizar a caixa de diálogo ***Alignment*** através da opção ***Align*** do ***Menu Edit*** ou através da paleta ***Align***, disponibilizada pela opção ***Alignmente Palette*** do ***Menu View***:



OBS: ***Antes de escolher qualquer uma destas opções de alinhamento, os componentes a serem alinhados devem estar selecionados. O alinhamento será feito com base no primeiro componente selecionado.***

4 - Criando Menus

Sabemos que toda aplicação ***Windows*** que se presa deve possuir uma ***Barra de Menu***. Normalmente, as tarefas mais complexas de uma aplicação são executadas através de ***Comandos de Menus***.

4.1- Padronização de Menus

Existem alguns ***Padrões*** que devem ser seguidos quando da criação de um ***Menu*** para a aplicação. Esses ***Padrões*** são utilizados pelas grandes Softwarehouses, tornando-os conhecidos mundialmente. Portanto, segui-los é sinal de bom senso e garantia de que, pelo menos no quesito usabilidade, o seu software terá boa aceitação por parte dos usuários.

PADRÃO:	***DESCRIÇÃO:***
Item Cinza	Significa que aquela opção não está disponível naquele momento.
Reticências	Abrirá uma caixa de diálogo, que deve ser preenchida para que o comando possa ser executado.
Tique	Indica o estado ou condição (ativado/desativado) do item.
Triângulo	Abrirá um segundo *Menu* (submenu) com mais opções.
Itens de Rádio	Aplica-se a um conjunto de opções de *Menu*, onde somente uma pode estar ativa por vez.
Separador	Traço que separa grupos de opções que finalizam tarefas diferentes dentro do mesmo *Menu*.
HotKey	As *HotKeys (Letras Quentes)* servem como atalho para acessar determinada opção de *Menu* colocando-se um *E comercial ("&")* à esquerda da letra. Normalmente a primeira letra é definida como atalho.
Teclas de Atalho	Tecla ou combinação de teclas que aparecem ao lado do item de *Menu*. Possibilitam um acesso rápido ao comando sem que se tenha que tirar as mãos do teclado.

Uma outra característica das *Barras de Menus* é o posicionamento e o formato dos seus ***Menus***. Iniciando com o ***Menu Arquivo***, seguindo de ***Editar, Visualizar (ou Exibir)*** e outros específicos da aplicação e terminando sempre com ***Ferramentas, Janelas*** e ***? (Help)***. Quando utilizados, esses menus também possuem um formato ***Padrão***, por exemplo, no ***Menu Arquivo*** estão os comandos relacionados com arquivos ***(Novo, Salvar, Imprimir, Sair***, etc.). É obvio que estes padrões só podem ser seguidos se fizerem sentido dentro da aplicação que está sendo desenvolvida. Fica a critério do programador.

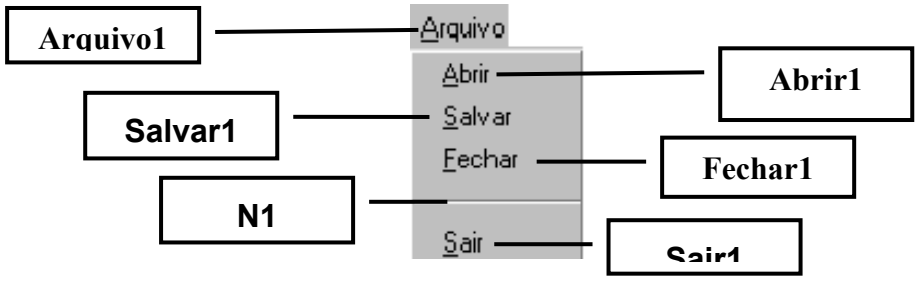
4.2 - Menu Designer

O Menu Designer permite que você crie e adicione facilmente ***Menus*** ao seu formulário. Você digita os ***Items de Menu*** diretamente na ***Janela do Menu Designer***. As alterações feitas na ***Barra de Menus*** são mostradas diretamente no formulário em tempo de projeto.
Você pode também criar um ***Menu*** a partir de um modelo. E ainda, criar seus próprios modelos para serem utilizados em outros formulários da mesma aplicação ou de outras.
Para executar o Menu Designer insira um componente de Menu (MainMenu ou PopupMenu***)*** no seu formulário – não se preocupe com o seu posicionamento – e dê
um clique duplo sobre ele.

OBS: *Procure observar os principais aplicativos Windows: como são estruturados seus Menus, as nomenclaturas utilizadas, os padrões, etc.*

4.3 - Editando Itens

Assim que você entrar no Menu Designer, uma janela em branco aparece com m campo de entrada no seu canto superior esquerdo. Comece por digitar o título do primeiro Menu Suspenso, você perceberá que o título será digitado na propriedade Caption no Object Inspector. Criado o Menu Suspenso, abre-se um campo para informar o primeiro Item de Menu, e assim por diante.



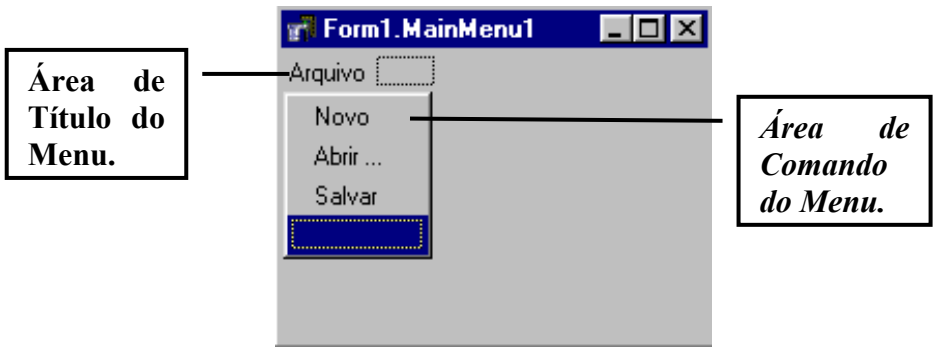
Para apagar um ***Item*** do *Menu Suspenso*, selecione-o pressione a tecla <*DEL / DELETE*>***;*** apagando o *Menu Suspenso* eliminam-se todos os seus *Items*.

Para trocar um Item ou Menu de posição, selecione-o e arraste para a nova posição, mesmo que esteja em outro Menu.

Para inserir um novo Item entre dois já existentes, clique com o botão direito do mouse sobre o ***Item*** que será deslocado para baixo e escolha a opção ***Insert***, ou pressione a tecla <*INS/INSERT*>. Use as setas de direção para navegar dentro do Menu, designe e escolha o ***Item*** que deseja alterar. Para cada *Menu* e *Item de Menu*, o *Menu Designer* cria um *Objeto* no *Object Inspector.* Estes objetos possuem propriedades que podem ser configuradas tanto em tempo de projeto como de execução. Um exemplo é a propriedade ***ShortCut*** que determina as *Teclas de Atalho* do *Item de Menu*.

4.4 - Como Funciona o Menu Designer

O Menu Designer cria componentes para cada Menu e Item de Menu, até mesmo para os ***Separadores***. Os ***Nomes*** para esses componentes são criados automaticamente removendo-se *caracteres especiais, espaços, hífen* e *"&"*. Caso a *Legenda* não comece com uma letra, é atribuída a letra ***N*** ao nome do componente; e, por fim, um número é adicionado ao final do nome começando em ***1*** para o primeiro elemento com aquele título, incrementando em caso de repetição.



Por exemplo: imagine que você tenha criado um menu chamado ***Arquivo***, com os seguintes itens: *Abrir, Salvar, Fechar* e *Sair*, mais um separador entre *Fechar* e *Sair*. O *Menu Designer* cria os seguintes componentes – *Arquivo1, Abrir1, Salvar1, Fechar1, N1* e *Sair1*.

OBS: *Deve-se tomar cuidado com Items de Menus que contenham Acentuação. O Menu Designer não os reconhece, simplesmente omitindo-os no Nome do Componente, por exemplo, o nome dado ao item chamado Edição será Edio1.*

Para fazer um Separador no meio dos Items de Menu, use a propriedade Caption e digite apenas um Hífen (-).

4.5 - Codificando os Itens de Menu

Todos os Items de Menu possuem somente um evento OnClick. Para adicionar código a ele, feche o Menu Designer; você perceberá que o menu criado por ele já aparece no seu formulário. Clique sobre o Item de Menu desejado e insira o código correspondente.

4.6 - Criando um Menu Pop-Up

Para criarmos um Menu Pop-up (menu de contexto – SpeedMenu), realizamos os mesmo procedimentos da Criação de um MainMenu exceto por utilizar a ferramenta Pop-up Menu da paleta de ferramentas.
Utilizamos o Menu Designer dando um duplo clique sobre o componente Pop-up Menu e informamos as opções do menu.
Depois de criado o Menu, devemos associá-lo ao elemento da interface através da sua propriedade Pop-up Menu, e então codificar suas opções. Desta forma quando o usuário clicar com o botão direito do mouse sobre o elemento, o seu SpeedMenu aparecerá.

5 - OBJECT PASCAL

5.1 - A Estrutura de Uma Unit

Uma ***Unit*** é constituída de ***Constantes, Tipos de Dados, Variáveis, Procedures*** e ***Funções***. A ***Unit*** é a base da programação modular do ***Object Pascal***. É na ***Unit*** que são inseridas as procedures e funções que vão controlar a execução da aplicação (possui a extensão ***.PAS***).
Para cada formulário existe uma ***Unit*** associada, o contrário, porém, não é verdadeiro – podemos ter ***Units*** sem formulário.

Vamos examinar o código gerado para um novo Form, identificando as principais seções de uma Unit típica. Crie uma nova aplicação e observe na Unit principal as seguintes cláusulas.

Unit: A primeira declaração de uma unit é seu identificador, que é igual ao nome do arquivo.
Interface: Seção interface, onde ficam declarações que podem ser usadas por outras Units.
Uses: Na cláusula uses fica a Lista de Units usadas.
Type: Na cláusula type fica a definição de tipos, aqui temos a declaração da classe do Form.
Var: Na cláusula var são declaradas as variáveis, aqui temos a declaração da instância do Form.
Implementation: Na seção implementation ficam as definições dos métodos.
Initialization:
Finalization:
End.: Toda Unit termina com um "*end.*" a partir de onde qualquer texto é ignorado.

Vamos dar uma olhada na ***Unit Figuras*** :

```
Unit Figuras;  // nome da unidade

Interface      // determina o que é visível/acessível para qualquer
               programa (ou unit)}

uses           // lista todas as units usadas por esta unit
   Windows, Messages, SysUnits, Classes, Graphics, Controls,   Forms,
   Dialogs, StdCtrls, ExtCtrls;
```

```
type // lista o formulário e todos os seus componentes e procedures
  TfrmGeo = class(TForm)
    shpObjeto: TShape;
    rdgOpcoes: TRadioGroup;
    rbtRetangulo: TRadioButton;
    rbtCincunferencia: TRadioButton;
    rbtQuadrado: TRadioButton;
    btnFechar: TButton;
    procedure rbtRetanguloClick(Sender: TObject);
    procedure rbtCircunferenciaClick(Sender: TObject);
    procedure rbtQuadradoClick(Sender: TObject);
    procedure btnFecharClick(Sender: TObject);
  private
          {   Private declarations   }
          // declarações privadas
  public
          {   Public declarations   }
          // declarações públicas
end;
var       // declaração de variáveis Globais
   frmGeo: TfrmGeo

implementation      // código da lógica das rotinas utilizadas

{$R * .DFM}

// Implementação das procedures e funções utilizadas p/ controlar a
execução da aplicação

procedure TfrmGeo.rbtRetanguloClick(Sender: TObject);
begin
   shpObjeto.Shape := stRectangle;
end;

procedure TfrmGeo.rbtCircunferenciaClick(Sender: TObject);
begin
   shpObjeto.Shape := stCircle;
end;

procedure TfrmGeo.rbtQuadradoClick(Sender: TObject);
begin
   shpObjeto.Shape := stSquare;
end;

procedure TfrmGeo.btnFecharClick(Sender: TObject);
begin
   close;
end;
end.
```

O cabeçalho de uma ***Unit*** começa com a palavra reservada ***Unit***, seguido pelo nome da unidade (um identificador). O próximo item em uma ***Unit*** é a palavra reservada ***interface*** que indica o

começo da parte de interface da ***Unit, ou seja,*** a parte visível a outras ***Units*** *que* especificarem esta ***Unit*** na sua clausula ***uses***. A cláusula ***uses*** pode aparecer em dois lugares:

Imediatamente depois da palavra reservada **interface*.***
Qualquer ***Unit*** pode usar as declarações feitas na parte de interface da ***Unit*** *atual*.
Por exemplo, suponha você está escrevendo código um uma ***Unit A***, e você quer chamar um procedimento declarado na parte de interface da ***Unit B***. Uma vez que você adiciona o nome da ***Unit B*** para cláusula uses na parte de interface da ***Unit A***, qualquer código na ***Unit A*** pode chamar os procedimentos declarados na ***Unit B***.

Se a ***Unit B*** está na cláusula uses na parte de interface da ***Unit A***, a ***Unit A*** não pode estar na cláusula uses na parte de interface da ***Unit B***. Isto cria uma referência circular entre as ***Units***, e quando você tenta compilar, o *DELPHI* gera uma mensagem de erro.

Imediatamente depois da palavra reservada **implementation*.***
☞ Somente declarações na ***Unit*** atual podem usar as declarações nas partes de interface das ***Units*** especificadas aqui. Qualquer outra ***Unit*** que usa esta ***Unit*** não poderá usar as declarações das Units listadas nesta cláusula uses.

Cláusula Interface
A parte ***Interface*** de uma ***Unit*** começa com a palavra reservada ***interface*** que aparece depois do cabeçalho da ***Unit*** e antes da palavra ***implementation***. A ***interface*** determina o que é ***"visível***" (acessível ao público) para qualquer aplicação de outra ***Unit*** que usa esta ***Unit***.
Na parte ***Interface*** da ***Unit***, você pode declarar constantes, tipos de dados, variáveis, procedimentos e funções.
Os ***Procedimentos e Funções Visíveis*** a qualquer ***Unit*** ou aplicação que usa a ***Unit*** é declarada na parte de ***interface***. Os corpos atuais destes – ***a implementação*** – são encontrados na parte de implementação. A parte de ***interface*** lista todos os ***cabeçalhos de procedimentos e funções***; a parte de implementação contém a lógica codificada dos procedimentos e funções, incluindo todos os manipuladores de evento.
Uma cláusula ***uses*** opcional pode aparecer na interface e tem que seguir imediatamente após a palavra reservada ***interface***.

Cláusula Implementation (Implementação)
A parte de ***implementação*** de uma ***Unit*** começa com a palavra reservada ***implementation***. A ***implementação*** pode ter declarações adicionais de si própria, embora estas declarações não são acessíveis a qualquer outra aplicação ou ***Unit***. Estas declarações são usadas pelos procedimentos, funções, e manipuladores de evento declarados nesta mesma ***Unit***.
Uma cláusula ***uses*** opcional pode aparecer na parte de implementação e tem que aparecer imediatamente após a palavra reservada ***implementation***.
☞ Os corpos das rotinas declaradas na parte de interface têm que aparecer na parte de implementação. O ***procedimento*** ou ***cabeçalho de função*** que aparecem na parte de ***implementação*** devem ser idênticos à declaração feita na parte de interface.

Cláusula Initialization
Se você quer inicializar qualquer tipo de dado ou variável que a ***Unit*** usa ou torna disponível na parte de interface para a aplicação, você pode adicionar uma parte de ***inicialization (inicialização)*** para esta ***Unit*** e uma parte ***finalization*** após a palavra reservada ***initializaton*** e antes da palavra ***end.*** .

```
initialization
   { Seu código de inicialização vai aqui }
end.
```

Quando uma aplicação usa uma ***Unit***, o código dentro da parte de ***inicialização*** da ***Unit*** é chamado antes de qualquer execução de código da aplicação. Se a aplicação usa mais de uma ***Unit***, a parte de inicialização de cada ***Unit*** é chamada antes do resto da execução da aplicação.

Cláusula Finalization
Se sua ***Unit*** precisa executar qualquer ***cleanup*** (algo como liberar memória de variáveis e objetos alocados durante a execução) quando a aplicação termina, como liberar qualquer recurso alocado na parte de inicialização, você pode adicionar uma parte de ***finalization*** para sua ***Unit***. A parte de ***finalization*** vem após a parte de inicialização, mas antes da palavra final ***end.***

Por exemplo:

```
initialization
   { Seu código de inicialização vai aqui }
finalization
    { Seu código de finalização vai aqui }
end.
```

Quando uma aplicação termina, executa as seções de ***finalization*** de suas ***Units*** na ordem oposta às inicializações.

5.2 - Estrutura do Arquivo de Projeto

O ***Arquivo de Projeto*** possui a extensão ***.DPR***, e para que você possa visualizá-lo escolha a opção ***View Source*** no ***Menu Project***.

```
program Geo;
uses
   Forms,
   UFiguras in 'UFiguras.pas' {frmGeo};
  {$R * .RES}

begin
   Application.Initialize;
   Application.CreateForm(TfrmGeo, frmGeo);
   Application.Run;
end.
```

5.3 - O Arquivo de Formulário

As propriedades do formulário e seus componentes são armazenados num arquivo de descrição de formulário (com extensão .**DFM**). Para ver o arquivo de descrição do formulário clique com o botão direito do mouse sobre o formulário, e no SpeedMenu, escolha a opção **View as Text**. Para retornar ao formulário, acesse novamente o ***SpeedMenu*** (clicando sobre o editor) e escolha a opção ***View as Form***.

5.4 - Variáveis

No Delphi, toda variável tem que ser declarada antes de ser utilizada. As declarações podem ser feitas após a palavra reservada *var*, onde são indicados o nome e o tipo da variável. Os nomes de variáveis não podem ter acentos, espaços ou caracteres especiais como &, $ ou % e o primeiro

caractere de um nome de variável tem que ser uma letra ou um sublinhado. O Delphi não faz distinção entre letras maiúsculas e minúsculas.

5.4.1 - Variáveis Globais

As variáveis abaixo são globais, declaradas da *Interface* da Unit. Podem ser acessadas por qualquer Unit usuária.

```
Var
  FrmPrincipal : TfrmPrincipal;
  I : Integer;
  Usuario : string;
  A, B, Soma : Double;
  Ok : Boolean;
```

5.4.2 - Variáveis Locais

As variáveis abaixo são locais ao método, ou seja elas só existem dentro do método, não podem ser acessadas de fora, mesmo que seja na mesma Unit. Na verdade essas variáveis são criadas quando o método é chamado e destruídas quando ele é encerrado, seu valor não é persistente.

```
procedure TFrmExemplo.BtnTrocarClick(Sender: TObject);
var  Aux: string;
begin
  Aux :=  EdtA.Text;
  EdtA.Text := EdtB.Text;
  EdtB.Text := Aux;
end;
```

5.4.3 - Encapsulamento

Os principais níveis de visibilidade dos atributos e métodos de uma classe são mostrados abaixo.

Nível	Visibilidade
Private	Os itens declarados nesse nível só podem ser acessados na mesma unit.
Public	Nesse nível, qualquer unit usuária poderá acessar o item.
Protected	Os itens só poderão ser acessados em outra unit se for em uma classe descendente
Published	É o nível default, igual ao Public, mas define propriedades e eventos usados em tempo de projeto. Ex.: Quando da criação de um componente, as propriedades que deverão aparecer no Object Inspector em tempo de projeto deverão ser declaradas como Published.

5.4.4 - Constantes

São declaradas na seção *const*, podem ser usadas como variáveis, mas não podem ser alteradas. Geralmente o nome das constantes é escrito em letras maiúsculas e na declaração dessas constantes não é indicado o tipo.

```
const
  G = 3.94851265E-19;
  NUM_CHARS = '0123456789';
  CR = #13;
  SPACE = ' ';
  MAX_VALUE = $FFFFFFFF;
```

5.4.5 - Constantes Tipadas

Na verdade, constantes tipadas são variáveis inicializadas com valor persistente, que podem ser alteradas normalmente, como qualquer variável. A única diferença de sintaxe entre constantes tipadas e simples é que o tipo da constante é indicado explicitamente na declaração. Se uma constante tipada for declarada localmente, ela não será destruída quando o método for encerrado. Para diferenciar das constantes normais, costuma-se declarar estas com letras de case variável, como abaixo.

```
const
  Cont: Integer = 1;
  Peso: Double = 50.5;
  Nome: String = 'JOÃO';
```

5.4.6 - Tipos de Dados Padrão

O Delphi trata vários tipos de dados padrão, segue uma descrição sucinta desses tipos.

Tipos Inteiros

São tipos numéricos exatos, sem casas decimais. O tipo *Integer* é o tipo inteiro padrão.

Tipo	**Tamanho em Bytes**	**Valor Mínimo**	**Valor Máximo**
ShortInt	1	-128	127
SmallInt	2	-32768	32767
Longint	4	-2147483648	2147483647
Byte	1	0	255
Word	2	0	65535
Integer	4	-2147483648	2147483647
Cardinal	4	0	2147483647

Tipos Reais

São tipos numéricos com casas decimais. O tipo *Double* é o tipo real padrão.

Tipo	**Tamanho em Bytes**	**Valor Mínimo**	**Valor Máximo**	**Dígitos Significativos**
Real	6	10^{-39}	10^{38}	11-12
Single	4	10^{-45}	10^{38}	7-8
Double	8	10^{-324}	10^{308}	15-16
Extended	10	10^{-4932}	10^{4932}	19-20
Comp	8	-10^{18}	10^{18}	19-20
Currency	8	-10^{12}	10^{12}	19-20

Tipos Texto

Os tipos texto podem operar com caracteres simples ou grupos de caracteres. O tipo texto padrão é o tipo string.

Tipo	**Descrição**
Char	Um único caractere ASCII
String	Texto alocado dinamicamente, não há limite para o número de caracteres, mas pode ser limitado a 255 caracteres conforme config.

Pchar	String terminada em nulo (#0), usada geralmente nas funções da API do Windows

O operador **+** pode ser usado para concatenar strings e você pode usar uma variável do tipo string como uma lista de caracteres.
ShowMessage('5ª letra do título da janela: ' + Caption[5]);
Label1.Text := '2ª letra do Edit: ' + Edit1.Text[2];

Existem várias funções de manipulação de strings, veja algumas das mais importantes mostradas abaixo.

Função	**Descrição**
AnsiCompareText	Compara 2 strings sem sensitividade de maiúsculas/minúsculas
AnsiLowerCase	Converte todas as letras de uma string para minúsculas
AnsiUpperCase	Converte todas as letras de uma string para maiúsculas
Copy	Retorna parte de uma string
Delete	Apaga parte de uma string
Insert	Insere uma string em outra
Length	Número de caracteres de uma string
Pos	Posição de uma string em outra
Trim	Remove todos os espaços de uma string
TrimLeft	Remove os espaços à esquerda de uma string
TrimRight	Remove os espaços à direita de uma string
Format	Formata uma string com uma série de argumentos de vários tipos

Por exemplo, para comparar o texto de dois Edits, poderíamos usar a função AnsiCompareText.

```
if AnsiCompareText(EdtA.Text, EdtB.Text) = 0 then
  ShowMessage('Os textos dos dois Edits são iguais.');
```

Um detalhe que deve ser observado é que as propriedades dos objetos não podem ser usadas como variáveis em funções. Veja a declaração do procedimento Delete no help.

```
procedure Delete(var S: string; Index, Count:Integer);
```

Digamos que você deseje apagar as 5 primeiras letras de um Edit, como a string do Delete é variável, não poderia usar o código abaixo.

```
Delete(Edit1.Text, 1, 5);
```

Para você poder fazer a operação desejada, teria que usar uma variável auxiliar.

```
var
  S: string;
begin
  S := Edit1.Text;
  Delete(S, 1, 5);
  Edit1.Text := S;
end;
```

Tipos Ordinais

Tipos ordinais são tipos que tem uma seqüência incremental, ou seja, você sempre pode dizer qual o próximo valor ou qual o valor anterior a um determinado valor desses tipos. São tipos ordinais o Char, os tipos inteiros, o Boolean e os tipos enumerados. Algumas rotinas para ordinais são mostradas abaixo.

Função	Descrição
Dec	Decrementa variável ordinal
Inc	Incrementa variável ordinal
Odd	Testa se um ordinal é ímpar
Pred	Predecessor do ordinal
Succ	Sucessor do ordinal
Ord	Ordem de um valor na faixa de valores de um tipo ordinal
Low	Valor mais baixo na faixa de valores
High	Valor mais alto na faixa de valores

Por exemplo, use o código abaixo no evento OnKeyPress de um Edit e veja o resultado.

```
Inc(Key);
```

Boolean

Variáveis do tipo Boolean podem receber os valores lógicos True ou False, verdadeiro ou falso. Uma variável Boolean ocupa 1 byte de memória.

TDateTime

O tipo TDateTime guarda data e hora em uma estrutura interna igual ao tipo Double, onde a parte inteira é o número de dias desde 31/12/1899 e a parte decimal guarda a hora, minuto, segundo e milissegundo. As datas podem ser somadas ou subtraídas normalmente.

Existem várias rotinas de manipulação de datas e horas, usadas com o tipo TDateTime, veja algumas abaixo.

Rotina	Descrição
Date	Retorna a data do sistema
Now	Retorna a data e hora do sistema
Time	Retorna a hora do sistema
DayOfWeek	Retorna o dia da semana de uma data especificada
DecodeDate	Decodifica um valor TDateTime em Words de dia, mês e ano
DecodeTime	Decodifica um valor TDateTime em Words de hora, minuto, segundo e milissegundos
EncodeDate	Retorna um TDateTime a partir de Words de dia, mês e ano
EncodeTime	Retorna um TDateTime a partir de Words de hora, minuto, segundo e milissegundos

No help de cada uma das funções acima você vai encontrar alguns exemplos, veja os colocados abaixo.

```
if DayOfWeek(Date) = 1 then
  ShowMessage('Hoje é Domingo!')
else
  ShowMessage('Hoje não é Domingo!');
```

```
var
  A, M, D: Word;
begin
  DecodeDate(Date, A, M, D);
  ShowMessage('Dia '+IntToStr(D)+' do mês '+ IntToStr(M)+' de '+
              IntToStr(A), [D, M, A]));
end;
```

Variant

Tipo genérico, que pode atribuir e receber valores de qualquer outro tipo. Evite usar variáveis do tipo Variant, pois o uso dessas variáveis podem prejudicar a performance do programa, além de diminuir a legibilidade do código fonte e a integridade do executável, veja o trecho de código abaixo e note como esse tipo de variável tem um comportamento estranho.

```
var
  V1, V2, V3: Variant;
begin
  V1 := True;
  V2 := 1234.5678;
  V3 := Date;
  ShowMessage(V1 + V2 + V3);
end;
```

5.5 - Conversões de Tipo

Freqüentemente você vai precisar converter um tipo de dado em outro, como um número em uma string. Para essas conversões você pode usar duas técnicas, o TypeCasting e as rotinas de conversão de tipos.

TypeCasting

TypeCast é uma conversão direta de tipo, usando o identificador do tipo destino como se fosse uma função. Como o Delphi não faz nenhuma verificação se a conversão é válida, você deve tomar um certo cuidado ao usar um TypeCast para não criar programas instáveis.

```
var
  I: Integer;
  C: Char;
  B: Boolean;
begin
  I := Integer('A');  // I será = 65
  C := Char(48);   // C será = 0
  B := Boolean(0); // B será = False
  Application.MessageBox(PChar('Linguagem de Programação' + #13 +
                         'Delphi 5'), 'BIGMASTER', MB_ICONEXCLAMATION);
end;
```

5.6 - Rotinas de Conversão

As principais rotinas de conversão estão listadas na tabela abaixo. Caso você tente usar uma dessas rotinas em uma conversão inválida, pode ser gerada uma exceção.

Rotina	Descrição
Chr	Byte em Char
StrToInt	String em Integer

IntToStr	Integer em String
StrToIntDef	String em Integer, com um valor default caso haja erro
IntToHex	Número em String Hexadecimal
Round	Arredonda um número real em um Integer
Trunc	Trunca um número real em um Integer
StrToFloat	String em Real
FloatToStr	Real em string
FormatFloat	Número real em string usando uma string de formato
DateToStr	TDateTime em string de data, de acordo com as opções do Painel de Controle
StrToDate	String de data em TDateTime
TimeToStr	TDateTime em Strind de Hora
StrToTime	String de hora em TDateTime
DateTimeToStr	TDateTime em string de data e hora
StrToDateTime	String de data e hora em TDateTime
FormatDateTime	TDateTime em string usando uma string de formato
VarCast	Qualquer tipo em outro usando argumentos do tipo Variant
VarAsType	Variante em qualquer tipo
Val	String em número, real ou inteiro
Str	Número, real ou inteiro, em String

Veja alguns exemplos de como usar essas rotinas. Conversão de dados é uma operação muito comum na programação em Object Pascal, seria interessante dar uma olhada no help de cada uma das funções acima.

```
var
  I: Integer;
  D: Double;
  S1, S2: string;
begin
  D := 10.5;
  I := Trunc(D);
  S1 := FloatToStr(D);
  S2 := IntToStr(I);
  ShowMessage(S1 + #13 + S2);
end;

var A, B, Soma: Bouble;
begin
  A := StrToFloat(EdtA.Text);
  B := StrToFloat(EdtB.Text);
  Soma := A + B;
  ShowMessage(Format('%f + %f = %f', [A, B, Soma]);
end;
```

5.7 - Expressões

Uma expressão é qualquer combinação de operadores, variáveis, constantes, valores literais e chamadas de funções que resultem em um valor de determinado tipo. Uma expressão é usada sempre que precisamos de um valor que possa ser obtido por uma expressão.

```
A + 12 * C;
Date - 4;
StrToInt(Edit1.Text + Edit2.Text);
StrToDate(Edit2.Text) - StrToDate(Edit1.Text);
12 * A / 100;
A < B;
```

5.8 - Operadores

Os operadores são usados em expressões e a ordem em que as expressões são executadas depende da precedência desses operadores. Veja abaixo a lista de operadores em ordem descendente de precedência.

Operador	**Descrição**
Operadores Unários	
@	Endereço
Not	*Não* booleano ou bit voltado para *não*
Operadores Multiplicativos e de direção de Bit	
*	Multiplicação ou interseção de conjuntos
/	Divisão de Real
div	Divisão de Inteiro
mod	Resto de divisão de Inteiros
as	TypeCast seguro quanto ao tipo (RTTI)
and	*E* booleano ou bit voltado para *e*
shl	Deslocamento de bits à esquerda
shr	Deslocamento de bits à direita
Operadores Aditivos	
+	Adição ou união de conjuntos
-	Subtração ou diferença de conjuntos
or	*Ou* booleano ou bit voltado para *ou*
xor	*Ou* exclusivo booleano ou bit voltado para *ou* exclusivo
Operadores Relacionais	
=	Igual
<>	Diferente
<	Menor
>	Maoir
<=	Menor ou igual
>=	Maior ou igual
in	Pertinência a conjuntos
is	Compatibilidade de tipos (RTTI)

Para forçar uma expressão de menor precedência a ser executada antes, você pode usar os parênteses, como mostrado abaixo.

```
(5 - 2) * 3;
(A > B) and (A < C);
```

Para fazer potenciação, use a função *Power* declarada na Unit Math, abaixo temos que A é igual a A elevado a 4.

```
A := Power(A, 4);
```

5.9 - Instruções

Os programas são compostos por instruções, que são linhas de código executável. Exemplos de instruções simples são atribuições, mensagens entre objetos, chamadas de procedimentos, funções e métodos, como mostradas abaixo. As instruções podem ser divididas em várias linhas, o que indica o fim de uma instrução é o ponto e vírgula no final. Quando uma instrução é quebrada, costuma-se dar dois espaços antes das próximas linhas, para melhorar a leitura do código.

```
Caption := 'Primeiro Sistema';
Form2.ShowModal;
Application.MessageBox('Você executou uma operação ilegal, o programa
                        será finalizado.','Falha geral', MB_ICONERROR);
```

Você pode usar várias instruções agrupadas em uma instrução composta, como se fosse uma só instrução. Uma instrução composta delimitada pelas palavras reservadas *begin* e *end*. Toda instrução, simples ou composta, é terminada com um ponto-e-vírgula.

```
if CheckBox1.Checked then
begin
  ShowMessage('O CheckBox será desmarcado.');
  CheckBox1.Checked := False;
end;
```

5.10 - Padronização de Código

Fora as regras de sintaxe da linguagem, você é livre para escrever seu ***Código*** da maneira que bem entender. Mas, alguns princípios podem e devem ser adotados para facilitar o processo de criação e, principalmente, o de manutenção do *Código*.

5.10.1 - Comentários

Com o *Object Pascal* você pode inserir comentários em seu programa de três formas diferentes: ***entre chaves, entre parênteses*** e ***asteriscos***, ou após ***duas barras de divisão***.

Exemplo:

```
{comentário inserido entre chaves }
(* uma outra opção de comentário *)
// comentário até o final da linha
```

Comentar o ***Código*** é um hábito saudável que deve ser adotado por todos os programadores.

5.10.2 - Endentações

A ***Edetação*** consiste em deslocar dois espaços para a direita as instruções que estiverem dentro de um comando composto.

Exemplo:

```
if ... then
   <instruções>;

if ... then
begin
   <instrução1>;
   <instrução2>;
end;
```

```
if ... then
begin
  if ... then
     <instrução1>;
  <instrução2>;
end;
```

Como você mesmo pode ver, com esta técnica fica mais fácil de identificar quais instruções estão dentro de cada estrutura ***If-Then***.

5.10.3 - Letras Maiúsculas e Espaços em Branco

Apesar da linguagem ***Pascal*** não ser ***Case-Sensitive***, isto é, não diferenciar letras maiúsculas de minúsculas, é um bom hábito digitar nomes de identificadores (*objetos, variáveis*, etc.) combinando letras maiúsculas e minúsculas.

Exemplo:

```
ExameMedico
ValorDolar
FichaCliente
```

O *Compilador Object Pascal* simplesmente ignora espaços em branco adicionais, linhas em branco e tabulações. Portanto, você pode usar livremente estes recursos para melhorar a legibilidade de seu *Código*.

5.11 - Estruturas de Decisão

5.11.1 - If

O if é uma estrutura de decisão usada para executar instruções em determinadas condições. O if é considerado uma só instrução, por isso, só encontramos o ponto-e-vírgula no final. O else é opcional.

```
if OpenDialog.Execute then
  Imagem.Picture.LoadFromFile(OpenDialog.FileName);

if Nota < 5 then
  ShowMessage('Reprovado')
else
  ShowMessage('Aprovado');
```

5.11.2 - Case

Permite que o fluxo da execução seja desviado em função de várias condições de acordo com o valor do argumento, que tem que ser ordinal, caso o valor do argumento não corresponda a nenhum dos valores listados, podemos incluir um else.

```
case Ch of
  ''                : ShowMessage('Espaço');
  '0'..'9'          : ShowMessage('Dígito');
  '+', '-', '*', '/': ShowMessage('Operador');
  else
    ShowMessage('Caractere especial');
end;
```

```
case CbbBorda.ItemIndex of
  0: BorderStyle := bsDialog;
  1: BorderStyle := bsSingle;
  2: BorderStyle := bsSizeable;
end;
```

5.12 - Estruturas de Repetição

5.12.1 - While

O laço while executa uma instrução até que uma condição seja falsa.

```
I := 10;
while I >= 0 do
begin
  ShowMessage(IntToStr(I));
  Dec(I);
end;
```

5.12.3 - For

O laço for executa uma instrução um número determinado de vezes, incrementando uma variável de controle automaticamente a cada iteração. Caso seja preciso que a contagem seja decremental, pode-se usar *downto* em vez de *to*.

```
for I := 1 to ComponentCount do
  ShowMessage('O ' + IntToStr(I) + ' Componente é ' + Components[I -
1].Name);

for I := Length(Edit1.Text) downto 1 do
  ShowMessage(Edit1.Text[I]);
```

5.12.4 - Repeat

O laço repeat executa instruções até que uma condição seja verdadeira.

```
I := 1;
repeat
  S := InputBox('Acesso', 'Digite a senha', '');
  Inc(I);
  if I > 3 then
    Halt;
until S = 'inter';
```

5.13 - Quebras de Laço

Em qualquer um dos laços mostrados podemos usar o procedimento **Break** para cancelar a repetição e sair do laço, podemos também forçar a próxima iteração com o procedimento usando **Continue**, encerrar a execução de um procedimento usando **exit** ou ainda, abortar a execução do programa com o comando **Halt**.

```
I := 1;
while true do
begin
  Inc(I);
  if I < 10000000 then
```

```
    Continue
  else begin
    ShowMessage('Chegamos a dez milhões');
    Break;
  end;
end;
```

5.14 - Sets

São conjuntos de dados de um mesmo tipo, sem ordem, como os conjuntos matemáticos. Conjuntos podem conter apenas valores ordinais, o menor que um elemento pode assumir é zero e o maior, 255.

```
TBorderIcons = set of TBorderIcon;
BorderIcons := [biSystemMenu, biMinimize];

if MesAtual in [Jul, Jan, Fev] then
  ShowMessage('Férias');

Os conjuntos podem ser definidos como constantes ou constantes tipadas,
como abaixo:
DIG_HEXA = ['0'..'9', 'A'..'F'];  // Constantes
ou
DIG_HEXA: set of Char = ['0'..'9', 'A'..'F']; // Constantes tipadas
```

Para testar se um determinado elemento pertence ou está contido em um conjunto usa-se o operador **in**, veja um exemplo:

```
if '9' in DIG_HEXA then
    ...comandos ...

ou ainda, sem declarar variável:

if '9' in ['0'..'9'] then
    ...comandos ...
```

Anotações

6 Bibliografia

Delphi 5 Passo a Passo Lite
Editora MAKRON Books

Delphi 3 Total
Maurício B. Longo
Ronaldo Smith Jr.
Editora Brasport

Dominando o Delphi 5 "A Bíblia"
Marco Cantù
Editora MAKRON Books

Aplicações em Delphi
Adelize Generini de Oliveira
Editora Visual Books

Help do Delphi 5

Sites da Internet:

http://www.drdelphi.com.br/
http://www.clubedelphi.com.br/
http://www.ramosdainformatica.com.br/
http://www.delphibrasil.com/
http://www.delphibr.com.br/
http://www.borland.com.br/

Fórum de Discussão:

http://ferramentas.abril.com.br/aberto/forum/delphi.shl