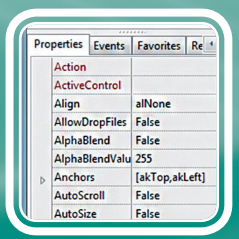
rede e-Tec Brasil

Programação em Ambiente Gráfico

Geraldo Nunes da Silva Júnior

Curso Técnico em Informática



INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA PIAUÍ

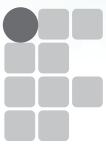
UNIVERSIDADE FEDERAL DE SANTA CATARINA

PDE | PRONATEC
PROGRAMA NACIONAL DE ACESSO AO ENSINO TÉCNICO E EMPREGO

Ministério da Educação

Programação em Ambiente Gráfico

Geraldo Nunes da Silva Júnior



INSITUTO FEDERAL DE EDUCAÇÃO
CIÊNCIA E TECNOLOGIA
PIAUÍ

Teresina – PI
2013

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação Profissional e Tecnológica

Este Caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia do Piauí e a Universidade Federal de Santa Catarina para a Rede e-Tec Brasil.

Equipe de Elaboração
Instituto Federal de Educação, Ciência e Tecnologia do Piauí – IFPI

Coordenação Institucional
Francieric Alves de Araujo/IFPI

Coordenação do Curso
Thiago Alves Elias da Silva/IFPI

Professor-autor
Geraldo Nunes da Silva Júnior/IFPI

Comissão de Acompanhamento e Validação
Universidade Federal de Santa Catarina – UFSC

Coordenação Institucional
Araci Hack Catapan/UFSC

Coordenação do Projeto
Silvia Modesto Nassar/UFSC

Coordenação de Design Instrucional
Beatriz Helena Dal Molin/UNIOESTE e UFSC

Coordenação de Design Gráfico
Juliana Tonietto/UFSC

Design Instrucional
Gustavo Pereira Mateus/UFSC

Web Master
Rafaela Lunardi Comarella/UFSC

Web Design
Beatriz Wilges/UFSC
Mônica Nassar Machuca/UFSC

Diagramação
Juliana Tonietto/UFSC
Roberto Gava Colombo/UFSC

Revisão
Júlio César Ramos/UFSC

Projeto Gráfico
e-Tec/MEC

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina

S586p Silva Júnior, Geraldo Nunes da
Programação em ambiente gráfico / Geraldo Nunes da Silva Júnior. – Teresina : Instituto Federal de Educação, Ciência e Tecnologia do Piauí, 2013.
126 p.: il., tabs.

Inclui bibliografia
ISBN: 978-85-67082-04-2

1. Linguagem de programação (Computadores). 2. Computação gráfica.
3. Programação (Computadores). I. Título.

CDU 681.31.06

Apresentação e-Tec Brasil

Bem-vindo a Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino, que por sua vez constitui uma das ações do Pronatec - Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de o acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (SETEC) e as instâncias promotoras de ensino técnico como os Institutos Federais, as Secretarias de Educação dos Estados, as Universidades, as Escolas e Colégios Tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e educação técnica, - é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Março de 2013

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor **9**

Apresentação da disciplina **11**

Projeto instrucional **13**

Aula 1 – A linguagem Lazarus e seu ambiente gráfico **15**

1.1 Introdução à linguagem Lazarus 15

1.2 Ambiente gráfico do Lazarus 16

1.3 Conhecendo comandos mais aplicados na linguagem Lazarus 21

Aula 2 – Recursos, componentes e *unit* ***29***

2.1 Recursos Lazarus 29

2.2 Criando uma aplicação 33

2.3 Entendendo o código 38

2.4 Analise da estrutura da *unit* – Código fonte do arquivo (.pas) 39

Aula 3 – Conceitos básicos de linguagem Pascal e paleta de componentes **43**

3.1 *Object* pascal 43

3.2 Identificadores 43

3.3 Palavra reservada 44

3.4 Comentários 44

3.5 Variáveis 44

3.6 Constantes 44

3.7 Operadores 45

3.8 Tipos de dados 45

3.9 Blocos de instruções 47

3.10 Uma aplicação usando tipo de dado *string* ***53***

3.11 Uma aplicação usando tipo de dado inteiro 61

3.12 Uma aplicação usando tipo de dado real 65

Aula 4 – Estrutura de decisão, repetição e novos componentes **73**
4.1 Estrutura de decisão 73
4.2 Estrutura de repetição 84

Aula 5 – Procedimentos, funções, *array* e *record* ***97***
5.1 Procedimento (*procedure*) 97
5.2 Funções (*function*) 105
5.3 *Arrays* (vetores e matrizes) 111
5.4 *Record* (registro) 117

Referências **125**

Currículo do professor-autor **126**

Palavra do professor-autor

Caro estudante!

Bem-vindo à disciplina de Programação em Ambiente Gráfico. O objetivo desta disciplina é apresentar o surgimento da Linguagem de Programação Lazarus e os seus principais conceitos, alguns recursos do IDE Lazarus. Exporemos conceitos básicos da Linguagem Pascal. Apresentaremos implementações de estrutura de decisões, de repetições no IDE Lazarus, bem como as implementações da estrutura de procedimento com e sem parâmetro, e estrutura de funções e do tipo de dados homogêneo (*array*) e de dados heterogêneo (*record*).

Aproveite o curso e a disciplina, pesquise, troque informações com seus colegas e tire dúvidas com seu tutor, pois ele tem muito mais informações para compartilhar com você.

Bons estudos!
Prof. Geraldo Nunes da Silva Júnior

Apresentação da disciplina

Na sociedade em que vivemos, sabemos a importância das automatizações das atividades através dos softwares. Uma linguagem de programação é um método padronizado para transmitir instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como esses dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

Um dos principais objetivos das linguagens de programação é permitir que programadores tenham uma maior produtividade expressando suas intenções mais facilmente do que utilizando a simples linguagem de máquina que o computador compreende.

Assim, linguagens de programação em ambientes gráficos são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. São ferramentas importantes para que estes desenvolvam programas mais organizados e com mais rapidez. Esperamos que ao final desta disciplina você tenha uma compreensão melhor sobre todas essas questões pertinentes a ela e ao curso como um todo.

Projeto instrucional

Disciplina: Programação em Ambiente Gráfico (carga horária: 75 h).

Ementa: Ambiente Lazarus: comandos, recursos, paleta de componentes, *forms* e *units*. Propriedades e eventos, estrutura de decisão, estrutura de repetição. Procedimento, função, *arrays* e *record*.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. A linguagem Lazarus e seu ambiente gráfico	Conceituar linguagem Lazarus. Conhecer o ambiente gráfico do Lazarus. Conhecer os principais comandos do Lazarus.	Vídeos; AVEA; *Software* Lazarus; Webconferência; *HotPotatoes*.	15
2. Recursos, componentes e *unit*	Conhecer os recursos oferecidos pela linguagem Lazarus. Conhecer algumas abas da paleta de componentes. Reconhecer as extensões dos arquivos gerados pelo Lazarus. Analisar as seções dentro da estrutura de uma *unit*.	Vídeos; AVEA; *Software* Lazarus; Webconferência; *HotPotatoes*.	15
3. Conceitos básicos da linguagem Pascal, tipos de dados, paleta de componentes	Conhecer conceitos básicos de linguagem Pascal. Compreender aplicação com o uso de tipos de dados: *string*, inteiro e real. Conhecer alguns componentes da paleta de componentes, propriedades e eventos.	Vídeos; AVEA; *Software* Lazarus; Webconferência; *HotPotatoes*.	15
4. Tomadas de decisão e estrutura de repetição	Implementar as estruturas de decisões no IDE Lazarus. Implementar as estruturas de repetições no Lazarus. Conhecer novos componentes da paleta de componentes e suas respectivas propriedades e alguns eventos.	Vídeos; AVEA; *Software* Lazarus; Webconferência; *HotPotatoes*.	15
5. Procedimentos, funções, *array* e *record*	Implementar a estrutura de procedimento com e sem parâmetro. Implementar a estrutura de funções. Implementar tipo de dados homogêneo *array*. Implementar tipo de dados heterogêneo *record*.	Vídeos; AVEA; *Software* Lazarus; Webconferência; *HotPotatoes*.	15

Aula 1 – A linguagem Lazarus e seu ambiente gráfico

Objetivos

Conceituar linguagem Lazarus.

Conhecer o ambiente gráfico do Lazarus.

Conhecer os principais comandos do Lazarus.

1.1 Introdução à linguagem Lazarus

O Projeto Lazarus surgiu em 1º de fevereiro de 1999, criado por três pessoas: Cliff Baeseman, Shane Miller e Michael A. Hess. Em agosto de 1999 Marc Weustink se junta ao grupo, seguido então por Mattias Gaertner em setembro de 2001.

Figura 1.1: Logotipo do *software* Lazarus
Fonte: Lazarus, 2011

O Lazarus é um ambiente de desenvolvimento gratuito de código livre para o compilador Free Pascal de *object* Pascal (Pascal orientado a objetos). O Ambiente Lazarus IDE é produtivo, estável e rico para o desenvolvimento de aplicações gráficas, console e internet. Ele atualmente roda em sistemas operacionais Linux, MS Windows (95 ou superior), Mac OS X, *BSD e Solaris, e provê um editor de código personalizável, além de um ambiente de criação visual de formulários (janelas) acompanhado de um gerenciador de pacotes, depurador (*debugger*) e completa integração da **GUI** com o compilador Free Pascal. Ele garante a qualidade que somente uma ferramenta **RAD** pode proporcionar.

Para saber mais sobre este programa, consulte o *site* http://www.lazarusbrasil.org

RAD
Rapid Application Development (desenvolvimento rápido de aplicação) é o termo usado para definir ferramentas que estimulam o *design* visual da interface gráfica e a alta reutilização de componentes, de forma a produzir uma aplicação no menor tempo possível.

GUI
Graphical User Interface é a interface gráfica de usuário que permite que os usuários interajam com dispositivos eletrônicos com imagens em vez de comandos de texto.

Para saber mais sobre Free Pascal, consulte o *site* http://www.freepascal.eti.br/

Para saber mais sobre como usar as licenças GPL ou a LGPL, consulte o *site* http://www.gnu.org/licenses/gpl-howto.pt-br.html

Tanto a **LCL** do Lazarus, quanto a FCL (*Free component Library)* ou a Biblioteca de componentes livre do Free Pascal são licenciadas sob a **LGPL**, que permite a você criar aplicações comerciais e vender livremente. Porém, a IDE, o Lazarus em si, é licenciado sob **GPL**.

Quando você ativa o Lazarus pela primeira vez, uma série de janelas flutuantes separadas irão aparecer em seu *desktop* (ambiente gráfico).

1.2 Ambiente gráfico do Lazarus

O ambiente de desenvolvimento do Lazarus, conforme Figura 1.2, possui as características **IDE**.

A-Z

LCL
Lazarus component library e significa Biblioteca de Componentes do Lazarus.

GPL
General Public License (Licença Pública Geral) GNU GPL ou simplesmente GPL É a designação da licença para software livre idealizada por Richard Matthew Stallman em 1989, no âmbito do projeto GNU da *Free Software Foundation* (FSF).

LGPL
Lesser General Public License, escrita por Richard Stallman e Eben Moglen em 1991 (e atualizada em 1999), é uma licença de *software* livre aprovada pela FSF e escrita como um meio-termo entre a GPL e licenças mais permissivas. A principal diferença entre a GPL e a LGPL é que esta permite também a associação com programas que não estejam sob as licenças GPL ou LGPL, incluindo *Software* proprietário.

IDE
Integrated Developer Environment e é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de *software* com o objetivo de agilizar esse processo. Geralmente os IDEs facilitam a técnica de RAD.

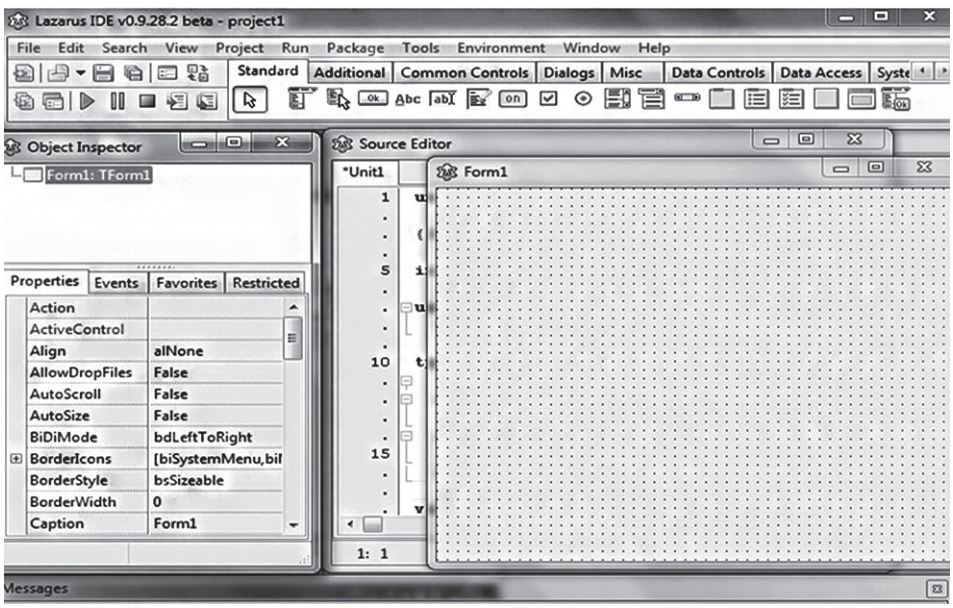


Figura 1.2: Ambiente gráfico do Lazarus
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

O Lazarus possui um conjunto de ferramentas que facilitam e agilizam a construção de programas, permitindo uma melhor interação entre o programador e o computador.

1.2.1 Janela principal

A janela principal controla o funcionamento do Lazarus. É através dessa janela que mantemos o Lazarus aberto, identificamos qual projeto está ativo, executamos os comandos de compilação, entre outras funcionalidades. Essa janela pode ser dividida em três partes:

a) Menu principal

Contém as opções de utilização do Lazarus, ou seja, assim como toda aplicação Windows, no menu principal estão presentes os comandos de abertura, criação, compilação, configuração do ambiente e chamadas a ferramentas auxiliares, conforme Figura 1.3.

Figura 1.3: Janela principal
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Barra de ferramentas

Contém botões que agilizam determinadas funções do Lazarus, ou seja, as principais opções do menu principal. É utilizada para acessar os comandos de forma mais rápida, conforme Figura 1.4.

Figura 1.4: Janela de ferramentas (*SpeedTools*)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

c) Paleta de componentes

Contém diversas páginas de componentes que podem ser utilizados na construção do projeto. Componentes são objetos que representam os *buttons, Labels, Edits*, dentre outros. Cada página (aba) apresenta um conjunto de ícones que representam as classes cujos objetos serão utilizados nos formulários das aplicações, conforme Figura 1.5.

Para adicionar componentes em um formulário, basta selecioná-lo e, em seguida, posicioná-lo no formulário. Os componentes podem ser visuais ou não visuais.

- Os componentes visuais são aqueles que podem ser redimensionados e aparecem no formulário quando o projeto está sendo executado.

- Os componentes não visuais são visíveis apenas em tempo de edição e não podem ser redimensionados.

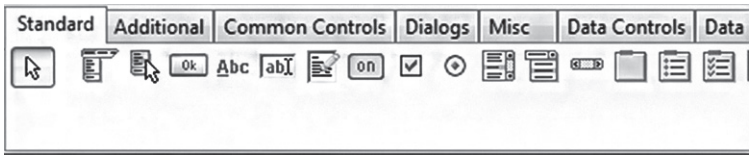


Figura 1.5: Janela de paleta de componentes
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

1.2.2 *Object inspector* (inspetor de objetos)

A janela *Object Inspector* contém propriedades e eventos dos componentes inseridos em um formulário e também do próprio formulário. Nela a hierarquia dos componentes existentes no projeto corrente está dividida em quatro páginas: *Properties, Events, Favorites e Restricted*, mostradas na Figura 1.6.

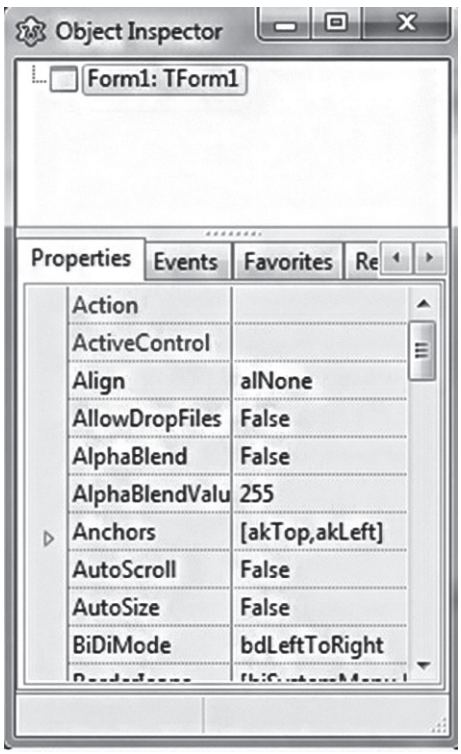


Figura 1.6: Janela do inspetor de objetos
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

As colunas (*abas*) de propriedades e eventos serão as mais utilizadas nas aplicações neste caderno.

- A página *Properties* (Propriedades) permite que se alterem os valores das propriedades dos componentes contidos no formulário ou ainda as propriedades do próprio formulário.

- A página *Events* (Eventos) estabelece ações a serem tomadas pelo componente a partir de um evento associado ao *mouse*, teclado, sistema operacional, etc.

1.2.3 O *form designer* (formulário)

O Formulário (*Form*) é a janela onde o desenvolvedor constrói sua aplicação. A partir de um formulário, conforme Figura 1.7, é que se estabelece a interação usuário-computador, por meio de botões, rótulos e outros componentes, estabelecendo-se funções, métodos ou eventos que serão ativados. Os componentes são dispostos na área útil do *Form*.

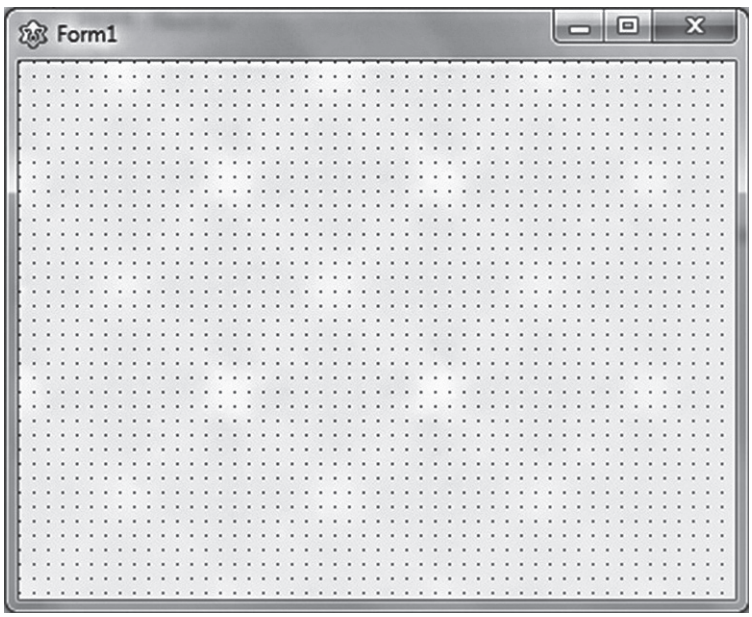


Figura 1.7: Janela do *Form* (Formulário)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Existem quatro tipos de formulários:

a) Normal – é o *form* padrão.

b) MDIForm – são os *forms* do tipo MDI (*Multiple Document Interface*), que podem conter outros *forms* do tipo *MDIChild*.

c) *MDIChild* – são *forms* que sempre estão subordinados a um *form MDIForm*.

d) *Splash* – são *forms* de abertura de um programa.

e) *StayOnTop* – são *forms* que sempre ficam visíveis na tela.

1.2.4 O *code editor* (editor de código)

A janela *Code editor* (Editor de código) é o local onde se desenvolve o programa-fonte. É nesse editor que se encontra a estrutura sintática propriamente dita da linguagem. Cabe ressaltar, no entanto, que boa parte do código escrito é gerada automaticamente.

No topo do editor de código, conforme mostrado na Figura 1.8, encontra-se um controle de múltiplas páginas, em que cada página corresponde a um módulo do programa (*unit*). Geralmente as *units* estão relacionadas aos formulários da aplicação. A cada vez que se adiciona um novo formulário na aplicação, o Lazarus adiciona também uma nova página ao editor de código.

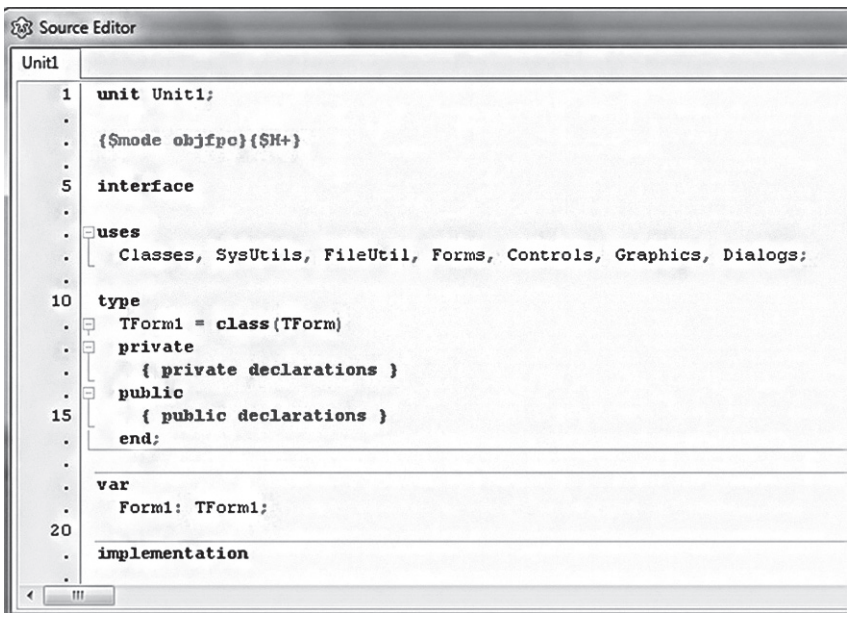


Figura 1.8: Janela Editor de código (*Code editor*)
Fonte: Print screen do Lazarus IDE v0.9.28.2

1.2.5 Mensagens

A janela Mensagens, mostrada na Figura 1.9, exibe as mensagens do compilador, erros ou relatórios de progresso em seu projeto.

Figura 1.9: Janela mensagem
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

1.3 Conhecendo comandos mais aplicados na linguagem Lazarus

No ambiente gráfico, o idioma padrão (inglês) no IDE Lazarus pode ser alterado para o português. Para isso, basta adotar o procedimento a seguir.

Ao abrir o ambiente gráfico do Lazarus: acesse Options do menu *Environment*, conforme Figura 1.10 a seguir.

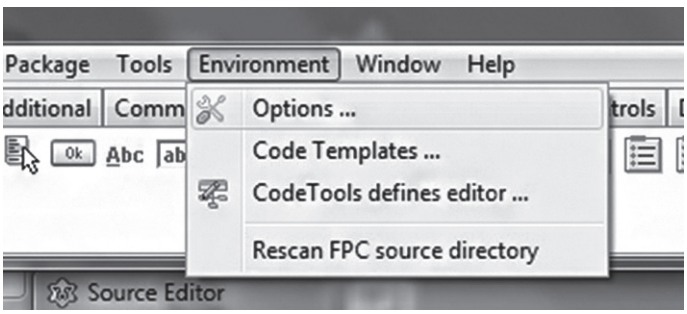


Figura 1.10: Menu Ambiente (*Environment*)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Após clicar em *Options*, aparecerá a janela, conforme Figura 1.11.

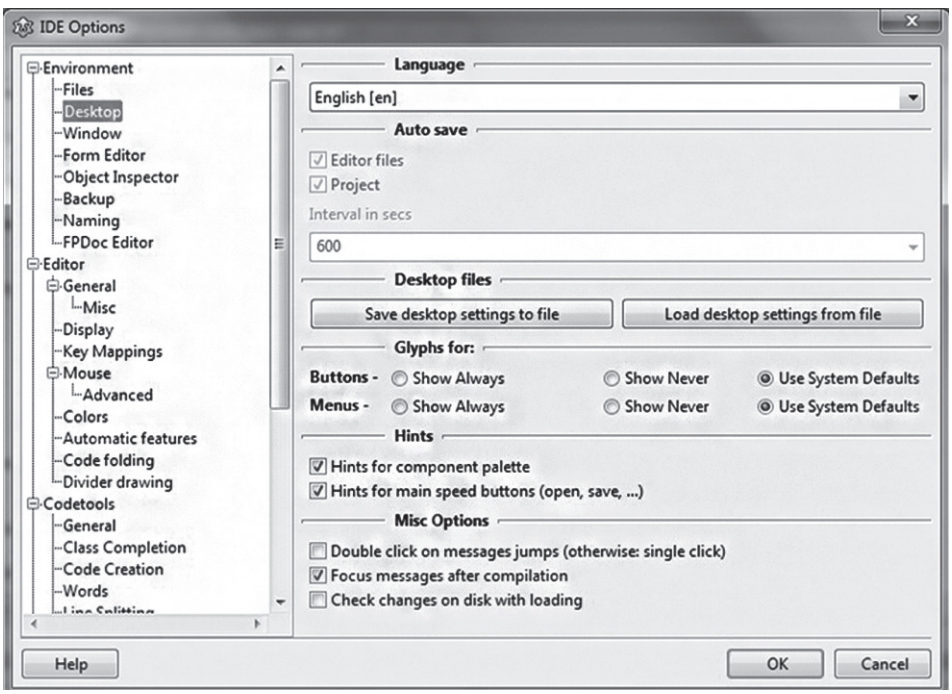


Figura 1.11: Janela de opções do menu Ambiente
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Selecione a opção *Desktop*, conforme Figura 1.11 e escolha o idioma Português do lado direito da referida janela e aperte o botão **OK**. O Ambiente estará no idioma português apenas após o fechamento do ambiente e sua reinicialização.

a) Menu *file* (arquivo)

Nesse menu, de acordo com Figura 1.12, aplicamos: uma Nova Unidade, um Novo Formulário, Salvar, Salvar Como, Salvar Tudo, entre outros.

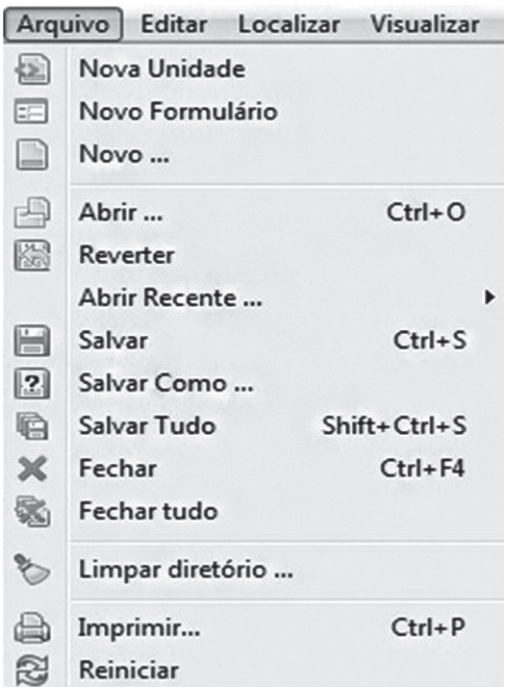


Figura 1.12: Menu *file* (arquivo)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Menu *view* (visualizar)

Nesse menu, como mostrado na Figura 1.13, podemos visualizar as janelas: Inspetor de Objetos (*Object Inspetor*), Mensagens (*Messages*), acessamos as *units*, os *forms*, alternamos entre o *form* e a sua *unit*, visualizamos a paleta de componentes, entre outras opções.

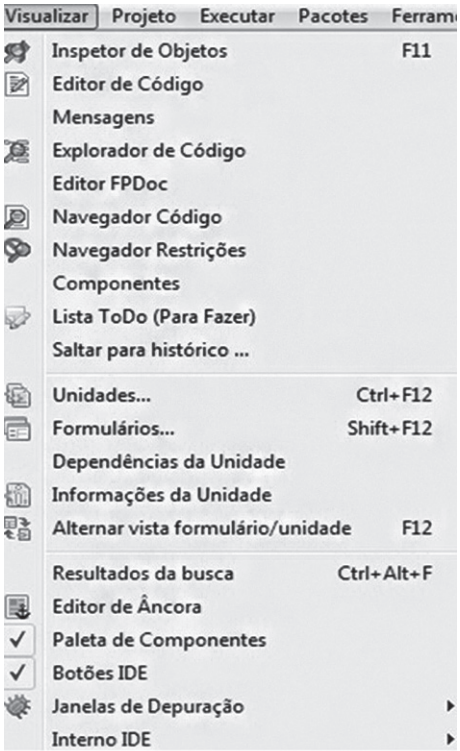


Figura 1.13: Menu *view* (visualizar)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

c) Menu *project* (projeto)

Nesse menu, conforme Figura 1.14, os mais utilizados são: Novo Projeto e Abrir Projeto e Salvar Projeto Como.

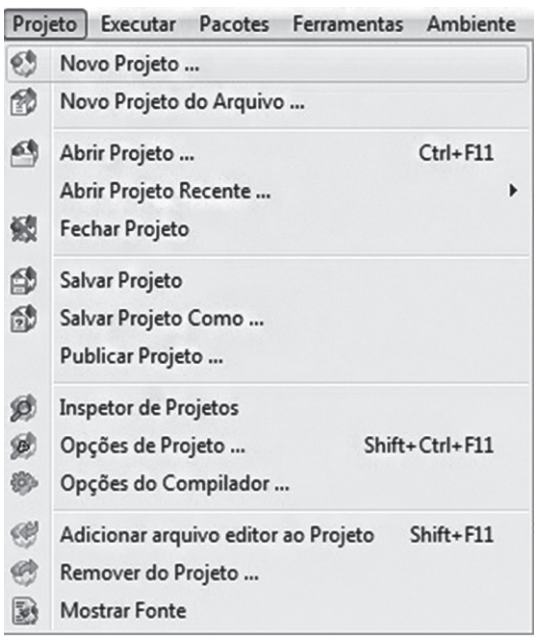


Figura 1.14: Menu *project* (projeto)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

d) Menu *run* (executar)

Nesse menu, como pode ser visto na Figura 1.15, executamos o projeto corrente, geramos o executável, depuramos o programa-fonte, entre outras funções.

Figura 1.15: Menu *run* (executar)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

e) Menu *package* (pacote)

Nesse menu, como mostrado na Figura 1.16 adicionamos e configuramos a paleta de componentes por meio dos pacotes.

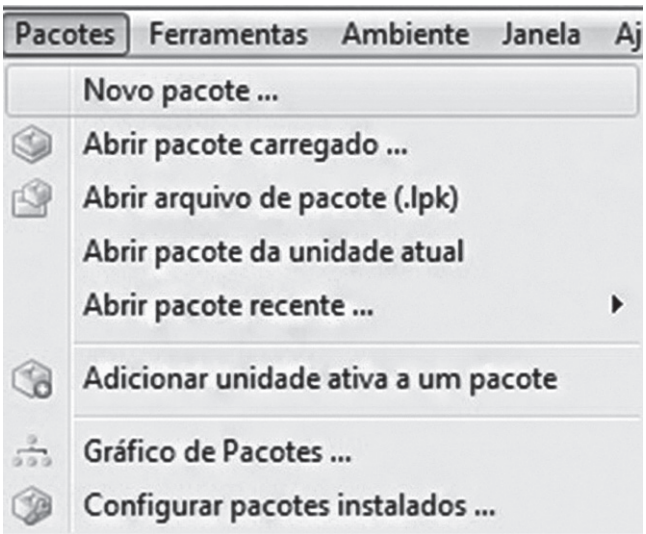


Figura 1.16: Menu *package* (pacote)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

f) Menu *enviroment* (ambiente)

Esse menu, de acordo com a Figura 1.17, é usado para a configuração do ambiente gráfico.

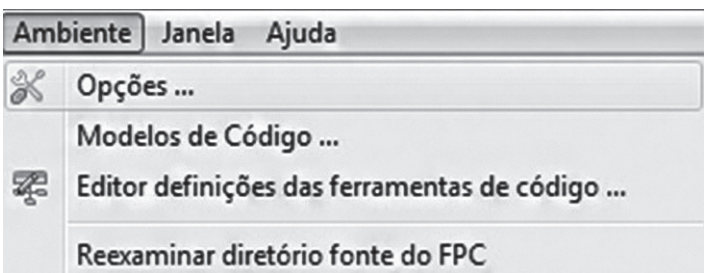


Figura 1.17: Menu *environment* (ambiente)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

g) Menu *help* (ajuda)

Esse menu, como mostrado na Figura 1.18, é usado para adquir orientação acerca da programação Lazarus, quando acessamos Ajuda Online.

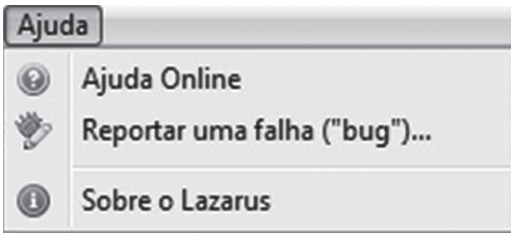


Figura 1.18: Menu *help* (ajuda)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Eu sei que você vai querer saber um pouco sobre a barra de menu do ambiente de programação da linguagem Lazarus. Para isso, acesse o endereço http://wiki.lazarus.freepascal.org/Lazarus_Tutorial/pt

Resumo

Nesta aula abordamos conceitos da Linguagem de Programação Lazarus e conhecemos seu ambiente gráfico e comandos principais, os quais serão aplicados nas aulas subsequentes.

Atividades de aprendizagem

1. De acordo com os conceitos básicos de Lazarus visto nesta Aula 1, o que você entende por IDE e RAD?

2. Ao abrir o Ambiente de Desenvolvimento Integrado do Lazarus, observamos diversas janelas que compõem esse IDE. Quais são essas janelas? Descreva cada uma delas.

3. Como se caracterizam os componentes visuais e não visuais?

4. Nesta aula conhecemos diversos menus compostos por ações utilizadas no desenvolvimento de projetos no IDE Lazarus. Localize e responda qual é o menu e atalho para as respectivas ações relacionadas abaixo, as quais serão mais utilizadas nesta disciplina:

   a) Salvar Tudo
   b) Inspector de Objetos
   c) Visualizar as Unidades
   d) Visualizar os Formulários
   e) Alternar entre *Form/Unit*
   f) Contruir o Projeto
   g) Executar o Projeto

Poste suas respostas no AVEA.

Aula 2 – Recursos, componentes e *unit*

Objetivos

Conhecer os recursos oferecidos pela linguagem Lazarus.

Conhecer algumas abas da paleta de componentes.

Reconhecer as extensões dos arquivos gerados a partir do Lazarus.

Analisar as seções dentro da estrutura de uma *unit*.

2.1 Recursos Lazarus

O Lazarus possui vários recursos úteis. Na sequência, destacamos alguns deles.

2.1.1 Linhas guias

Quando inserimos mais de um componente e ele possui a propriedade *Top*, *Height*, *Left* e/ou *Width* igual à de outro componente no mesmo formulário (ou mesmo contêiner), aparecem linhas guias apenas no modo de *designer*, como podemos ver na Figura 2.1:

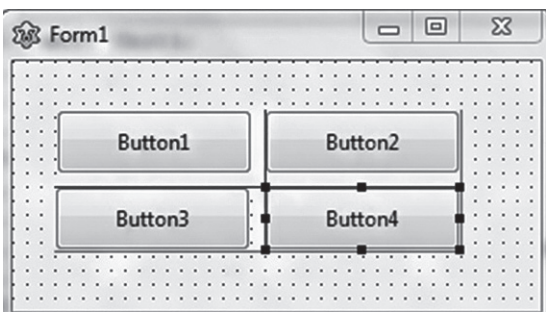


Figura 2.1: *Form* e inserções de botões e linha guias
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

2.1.2 Cópia e colagem de componentes

O Lazarus também possui a utilidade de cópia e colagem de componentes em tempo de *design*. Selecione um componente, ou vários, arrastando o *mouse*, e pressione *Ctrl+C e* logo em seguida pressione *Ctrl+V.*

2.1.3 Mudança na posição e tamanho pelo teclado

É possível mudar a posição e o tamanho dos componentes apenas com o teclado. Para mudar o tamanho, segure *Shift* e pressione uma das teclas direcionais, conforme indicado na Figura 2.2; para mudar a posição, segure *Ctrl* e pressione uma das teclas direcionais.

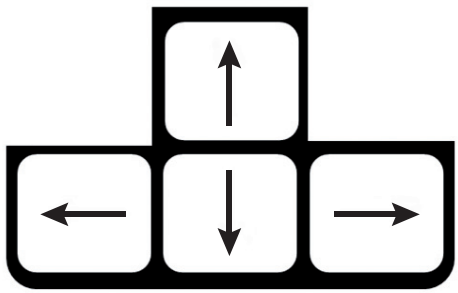


Figura 2.2: Teclas direcionais de um teclado de computador
Fonte: Elaborada pelo autor

2.1.4 Visualização rápida do código-fonte dos componentes

É possível visualizar o código-fonte dos componentes no Lazarus. Basta clicar com o botão direito do *mouse* no componente desejado na paleta de componentes; aparecerão dois menus: *Open Package* (Nome do Pacote}, e *Open Unit* (Caminho da *Unit*); clique no segundo para abrir edição e você verá o código-fonte do componente.

2.1.5 Visualização rápida de outros códigos-fonte

Segurando a tecla *Ctrl* e clicando em alguma *unit*, na cláusula *Uses*, por exemplo, você é automaticamente "redirecionado" ao código-fonte do arquivo.

2.1.6 Limpeza de diretório

O Lazarus possui um recurso de limpeza de diretório. Basta acessar o menu *File -> Clean Directory* ou Arquivo -> Limpar Diretório e especificar as extensões que você deseja deletar (Não use extensões **.pas* ou **.pp* ou o seu código-fonte será jogado fora).

2.1.7 *Form, object inspector* e componentes

Vamos estudar a fundo o *form*, o *object inspector* e os componentes, pois são eles os responsáveis diretos pelo visual e eventos de uma aplicação:

a) *Form* (formulário)

O Sistema Operacional Windows é conhecido como o sistema das janelas, justamente por serem elas a sua principal forma de comunicação com o usuário. Ai você se pergunta: O que o *form* tem a ver com isso? E a resposta é muito simples: Tudo!

Qual a diferença entre *forms* e janelas?

Um programa tem dois estados diferentes, sendo o primeiro deles o estado de desenvolvimento. É aqui que entra o *form*, que é o local onde o programador "desenha" o formato da aplicação. Já o segundo estado é o de execução, que seria a janela, em seu formato final, com todas as suas funcionalidades.

Se você não entendeu, vamos resumir: o *form* é a janela em tempo de desenvolvimento, e a janela é o *form* em tempo de execução.

Já sabendo o que é o tal do *form*, vamos agora nos aprofundar um pouco mais em suas propriedades:

b) *Object inspector* (inspetor de objetos)

- **Aba *properties* (propriedades)**
  Como dito anteriormente, o *object inspector* serve para alterar as várias propriedades de um objeto (componente ou *form*); as propriedade seriam nada mais do que estado e aparência. Vamos, então, checar as principais propriedade e eventos de um *form*, de acordo com o Quadro 2.1.

Quadro 2.1: Algumas propriedades do *form*

Propriedades	Funções
Caption	É o título da janela (texto que aparece na sua borda superior). Esse título geralmente é utilizado para descrever o que a janela faz.
Color	É a propriedade responsável pela cor de fundo do *form*.
Font	Aqui você pode alterar o tipo de fonte, cor, estilo, etc.
Name	É o nome do componente, que não pode ser repetido no projeto.
Position	Indica onde a janela ira aparecer quando o programa for executado. Sempre é importante alterar esta propriedade que, por *default* (*poDesigned*), coloca a janela no mesmo lugar onde ela foi construída, ou seja, na sua origem. Se a resolução do usuário for diferente, a sua janela poderá aparecer fora do vídeo, ou melhor, poderá não aparecer. Uma das propriedades mais utilizadas é o *PoScreenCenter*, que coloca o *form* sempre no centro da tela.
WindowState	Determina como a janela irá aparecer: maximizada (*wsMaximized*), minimizada (*wsMinimized*) ou normal(*wsNormal*).

Fonte: Elaborado pelo autor

- **Aba *events* (eventos)**

  Evento é algo que acontece, certo? No Lazarus não é diferente; por exemplo, o que pode acontecer com um *form*? Ele pode ser criado, destruído, mostrado, clicado, minimizado, etc. Você pode definir a sua aplicação, ou seja, o que fazer, caso aconteça algum evento. No Quadro 2.2 vamos ver alguns eventos.

Quadro 2.2: Alguns eventos do *form*

Eventos	Funções
OnActivate	Acontece sempre que o *form* é ativado.
OnClick	Acontece sempre que o *form* é clicado.
OnClose	Acontece sempre que o *form* é fechado.
OnShow	Acontece sempre que o *form* é mostrado

Fonte: Elaborado pelo autor

c) *Componentes*

O Sistema Operacional Windows introduz uma série de componentes visuais para facilitar a comunicação homem-máquina. Esses componentes são comuns a todos os programas encontrados no Windows.

O Lazarus possui uma coleção completa desses componentes visuais. Estão todos na paleta de componentes.

Veremos agora algumas abas da paleta de componentes que serão utilizadas em nossas aplicações, para que possamos nos familiarizar, e na Aula 3 veremos alguns dos componentes, tais como: *button*, *edit*, *label* e outros mais. Uma das abas mais utilizada é a *Standard* (veja a Figura 2.3); outra bastante aplicada é a *Additional* (vista na Figura 2.4) e a aba *Common Controls* (vista na Figura 2.5).

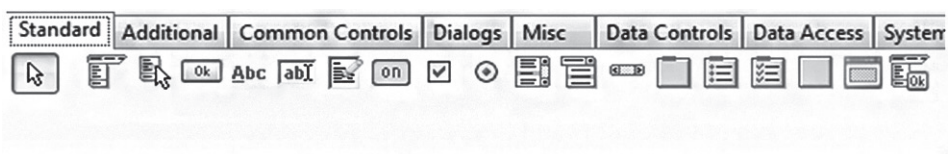


Figura 2.3: Aba *Standard*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

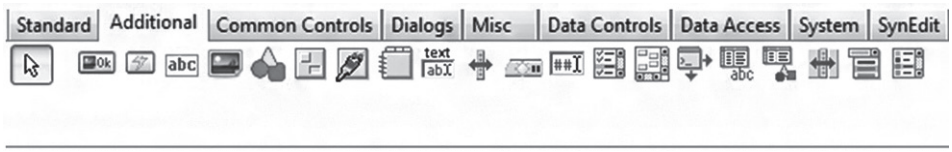


Figura 2.4: Aba *Additional*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Figura 2.5: Aba *Common Controls*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Eu sei que você vai querer conhecer a paleta de componente do ambiente de programação da linguagem Lazarus. Para isso, acesse o endereço http://wiki.lazarus.freepascal.org/Lazarus_Tutorial/pt

Vamos praticar um pouco do que acabamos de estudar? Tenho certeza de que você é capaz! Selecione alguma aba da paleta de componentes, insira alguns deles e acesse algumas de suas propriedades, tais como: name, *caption, width e height*.

2.2 Criando uma aplicação

Já conhecendo um pouco sobre componentes Lazarus e seu ambiente de trabalho, partiremos para uma aplicação.

Para que sejamos mais organizados no computador, criaremos na unidade C:, conforme Figura 2.6, uma pasta chamada Programacao_Ambiente_Grafico_Lazarus e uma subpasta chamada Programas, que servirão para salvar todas as aplicações do nosso material didático. Em cada aplicação desenvolvida, criaremos uma subpasta chamada Aplicacao_(número da referida aplicação), por exemplo uma subpasta chamada *Aplicacao_01*, *Aplicacao_02* e assim por diante.

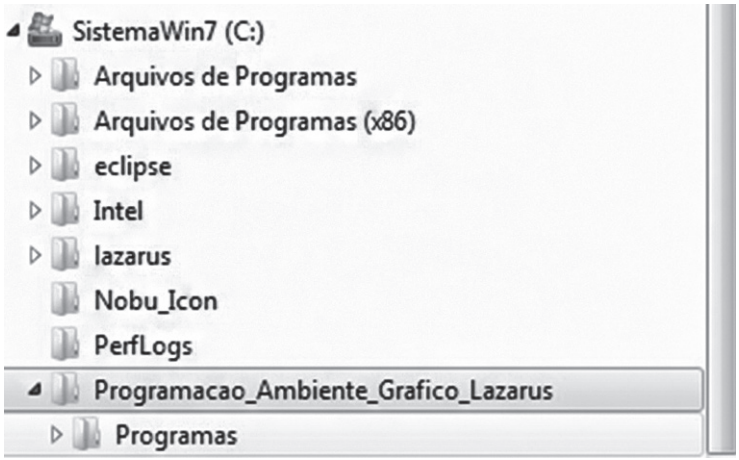


Figura 2.6: Pasta criada para os projetos no computador
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Vamos começar nossa primeira aplicação, a qual ajudará a nos familiarizar um pouco mais com as propriedades dos componentes.

Siga os passos abaixo e monte o formulário de acordo com a Figura 2.7.

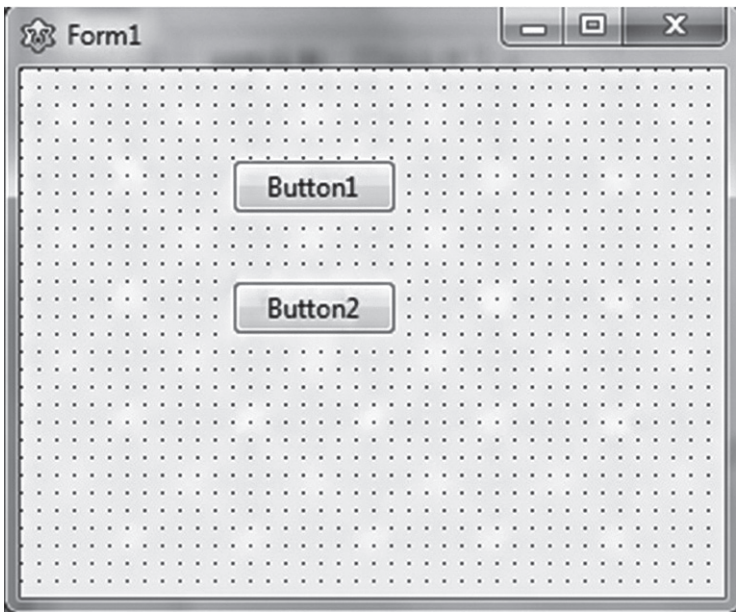


Figura 2.7: Janela principal da primeira aplicação
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

a) Coloque dois *buttons* no *form* (veja na Figura 2.7); eles estão localizados na paleta ou aba *Standard*).

b) Clique sobre o *button1* e vá até o *Object inspector*. Procure a propriedade *caption* do *button1* e altere-a para LIGADO; faça o mesmo com o *button2*, porém alterando para DESLIGADO, conforme Figura 2.8.

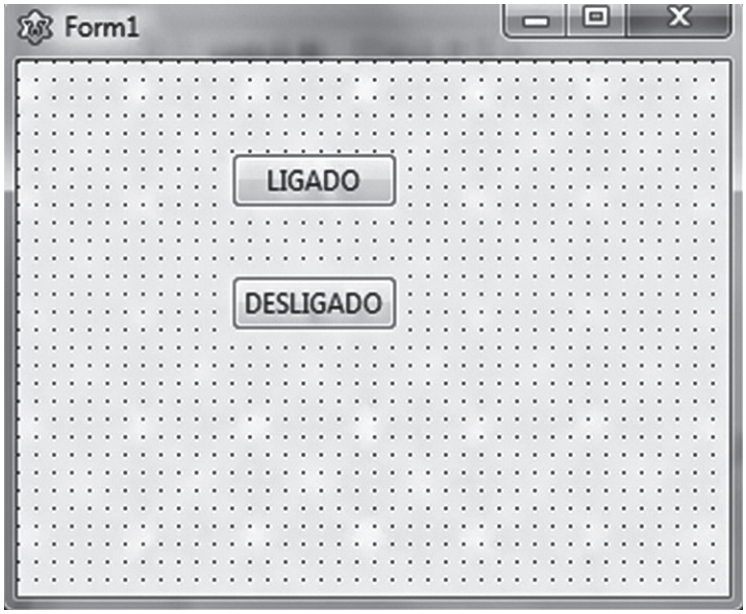


Figura 2.8: Janela principal com *buttons* renomeados
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

c) Procure também a propriedade *Enabled* do *button2* e altere-a para *False*.

d) Selecione o *button1*, segure a tecla *Shift* e selecione também o segundo botão. Vá ao *Object inspector* e procure a propriedade *Font*. Dê um clique no botão com reticências ao lado da propriedade e altere para *Arial-Regular-12*. Você deve ter notado que foram alterados os dois botões, deixando assim a dica do *shift* que serve para selecionar mais de um componente (caso os componentes sejam diferentes, serão exibidas somente suas propriedades comuns).

e) Ainda com os dois botões selecionados, altere a propriedade *Width de 75 para 100*.

f) Por último, e não menos importante, procure a propriedade *name* e a altere para *btn1*, no caso do primeiro botão, e *btn2*, para o segundo botão.

g) Salve a aplicação clicando no menu Arquivo>Salvar tudo; localize e selecione o diretório desejado (c:\Programacao_Ambiente_Grafico_Lazarus\Programas) e gere a subpasta Aplicacao_01 dentro da pasta Programas. Na janela "Salve *Unit1*", substitua o nome para *uAplicacao_01* e clique em salvar. Aparecerá uma janela, conforme a Figura 2.9; clique em Manter nome. A partir de agora a *Unit1* deixou de ter o nome padrão e foi substituída por *uAplicacao_01*. Na janela "Salvar Projeto *Project1*" digite *prj_Aplicacao_01* e clique em salvar. Pronto, seu projeto está salvo.

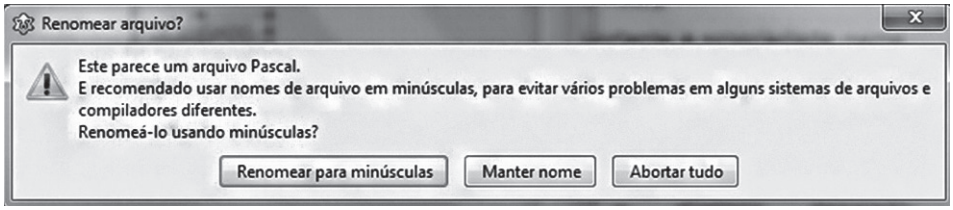


Figura 2.9: Janela emitida após salvar a *unit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

h) No Quadro 2.3 vamos ver agora os arquivos criados no momento de salvar. Abra a subpasta *Aplicacao_01* e nela nós podemos observar:

Quadro 2.3: Arquivos gerados do Lazarus

Extensões	Significados
prj_aplicacao_01.lpi	Contém configurações do Lazarus que ele salva automaticamente. Não deve ser modificado manualmente. Está escrito no formato *xml*.
prj_aplicacao_01.lpr	Contém o código-fonte principal do seu projeto. O próprio Lazarus cria e mantém este programa. Apesar da extensão **.lpr* também é um arquivo texto. Basicamente o que esse programa faz é mostrar o *form* principal e ficar esperando que esse *form* ser fechado. Não mexa aqui.
uAplicacao_01.lfm	Contém a definição do *form*. Tudo o que o *form* tem – botões, *labels, edits*, etc. – está definido aqui dentro.
uAplicacao_01.lrs	Contém os recursos utilizados, como imagens, desenhos, sons, vídeos, etc.
uAplicacao_01.pas	Finalmente encontramos uma coisa na qual você pode mexer. Este vai ser o seu programa, é aqui que você irá colocar o código em *Object* Pascal. O Lazarus escreve as partes principais para você, porém a parte mais difícil é por sua conta.
prj_aplicacao_01.exe	Este aqui é bem óbvio. É o Programa executável. Este só aparecerá quando o projeto for compilado.

Fonte: Elaborado pelo autor

Para fechar nossa aplicação, clique em Arquivo > Fechar tudo; se tiver feito algum tipo de alteração, será necessário salvá-la. Já para reabri-la, clique em Arquivo -Abrir, localize a pasta Aplicacao_01 e selecione *prj_Aplicacao_01.lpr*, clique em abrir.

i) Agora passaremos a conhecer um pouco mais a *unit* que passou a chamar-se *uAplicacao_01*; visualize o *Form1* e selecione o **btn1** e no *Object inspector* selecione a aba *Events*. Clique duas vezes sobre o espaço em branco na frente do Evento *OnClick*. A *uAplicacao_01* deve ser chamada e deve ficar semelhante à da Figura 2.10.

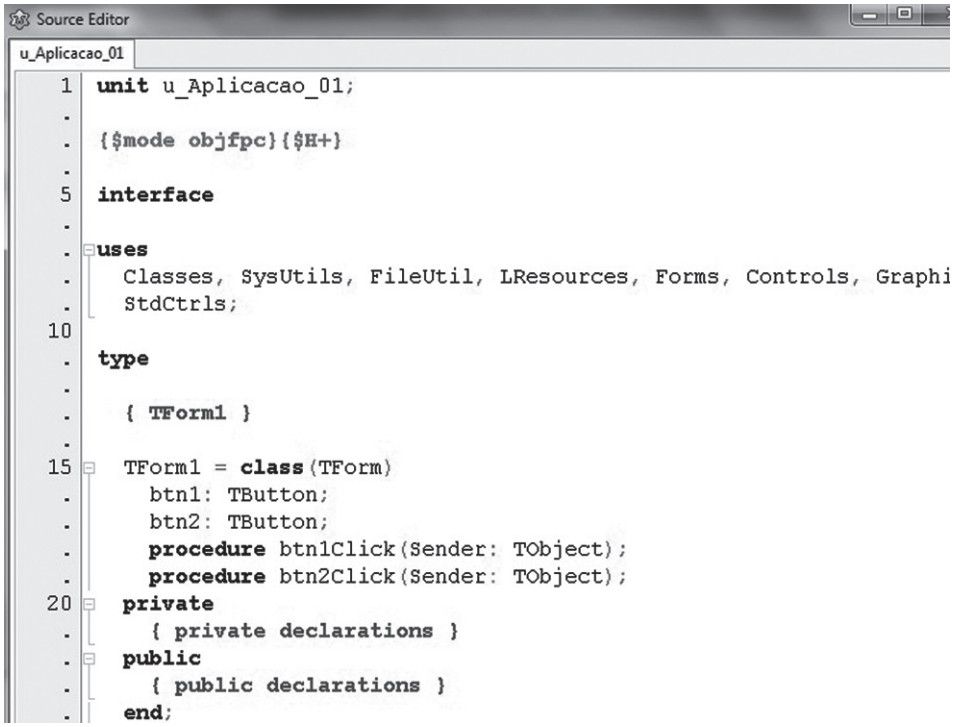


Figura 2.10: *Unit* com o procedimento após clicar em *btn1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

j) Observe que o Lazarus já se encarregou de declarar a *procedure*, e montar seu "esqueleto" automaticamente. Agora, dentro do espaço entre *begin* e *end*, digite o código conforme Figura 2.11.

```
if btn1.Enabled = true then
begin
     btn1.Enabled := false;
     btn2.Enabled := true;
     btn1.Caption := 'DESLIGADO';
     btn2.Caption := 'LIGADO';
end;
```

Figura 2.11: Código do button1
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

2.3 Entendendo o código

a) Quando o *botão1* for clicado, o evento *OnClick*, se ele estiver habilitado, então irá se desabilitar e mudar seu título, e o botão2 será habilitado e por sua vez também terá seu título alterado.

b) Agora vamos repetir o procedimento para o segundo botão: selecione-o e em seguida vá até a aba *Events* do *Object Inspector* e dê um clique duplo sobre o espaço em branco ao lado do evento OnClick; digite o código entre begin e end, conforme Figura 2.12.

```
if btn2.Enabled = true then
begin
     btn2.Enabled := false;
     btn1.Enabled := true;
     btn2.Caption := 'DESLIGADO';
     btn1.Caption := 'LIDADO';
end;
```

Figura 2.12: Código do *button2*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Note que o código é o mesmo anterior, porém invertido. Agora então salve o projeto clicando em Arquivo>Salvar Tudo, e em seguida teste compilando-o com a opção do menu Executar>Executar ou teclando F9.

c) Agora vamos testar o seu projeto: clique nos botões e veja o resultado.

Para conhecer melhor o ambiente Lazarus, insira componente *Edits*, *labels* e *Buttons* e acesse através do evento *OnClick*; realize a visibilidade dos componentes supracitados através da propriedade *visible*.

2.4 Analise da estrutura da *unit* – Código fonte do arquivo (.pas)

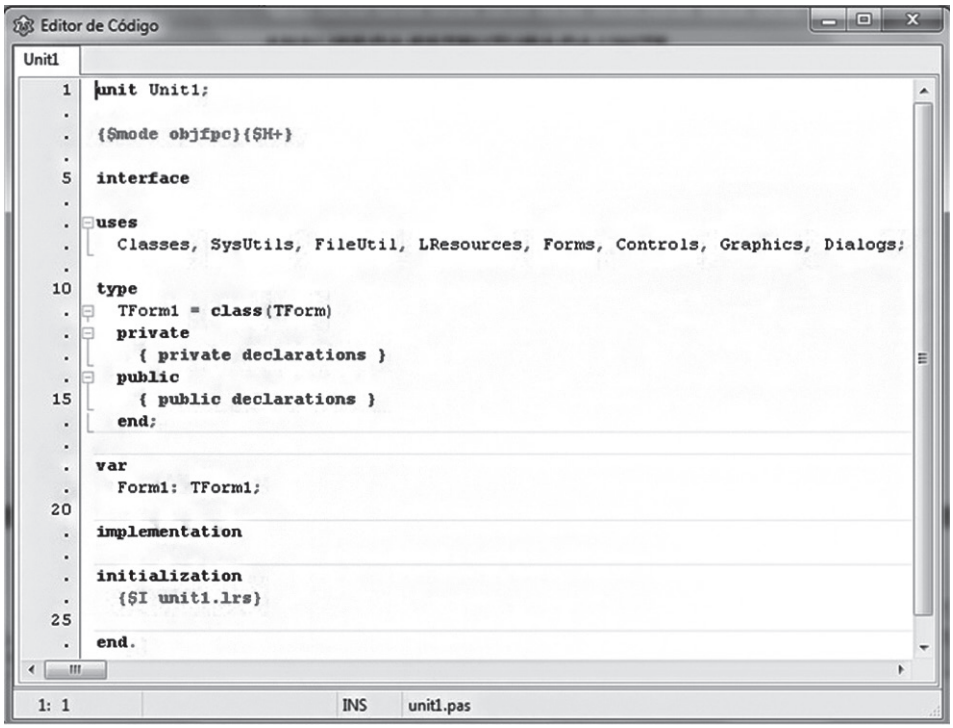


Figura 2.13: *Code editor*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Descreveremos cada seção da estrutura da *unit*, de acordo com a Figura 2.13.

2.4.1 Seção *unit*

Declaramos o nome da *unit*.

2.4.2 Seção *interface*

Nessa seção estão as declarações de constantes, tipos de variáveis, funções e *procedures* gerais da *Unit/Form.* Esta seção é formada pelo seguinte código:

a) *Interface* – palavra que inicia a seção;

b) *Uses* – contém as *units* (bibliotecas) do Lazarus, predefinidas ou definidas pelo desenvolvedor.

Algumas bibliotecas predefinidas:

- Classes: elementos de baixo nível do sistema de componentes;
- *SysUtils*: utilitários do sistema (*strings*, data/hora, gerar arquivos);

- *Forms*: componentes de forma e componentes invisíveis de aplicativos;
- *Control*: elementos de nível médio do sistema de componentes;
- *Graphics*: elementos gráficos;
- *Dialogs*: componentes de diálogo comuns.

2.4.3 Seção *type*

Declara os tipos definidos pelo usuário e as subseções:

a) *Private*: declarações privativas da *unit*.

b) *Public*: declarações públicas da *unit*.

2.4.4 Seção *var*

Declara as variáveis privadas utilizadas.

2.4.5 Seção implementation

Contém os corpos das funções e *procedures* declaradas nas seções *Interface* e *Type*. Nessa seção também estão definidos todos os procedimentos dos componentes que estão incluídos no *form*. As declarações desta seção são visíveis apenas para ela mesma.

2.4.6 Seção *inicialization*

Nesta seção, que é opcional, pode ser definido um código para proceder às tarefas de inicialização da *unit* quando o programa começa.

Resumo

Nesta aula apresentamos alguns recursos do IDE Lazarus que nos auxiliam no desenvolvimento de aplicações de forma otimizada, bem como as propriedades e eventos aplicados em formulários e outros objetos da paleta de componentes. Abordamos as extensões dos arquivos criados a partir do *Software* Lazarus e descrevemos as seções presentes na estrutura de uma *unit*.

Atividades de aprendizagem

3. Nesta Aula 2 vimos alguns dos recursos que o IDE do Lazarus oferece para proporcionar o desenvolvimento de projetos de forma mais rápida e automatizada. Cite alguns dos recursos abordados e descreva-os.

4. No decorrer da disciplina, perceberemos a importância do *form* e do Inspetor de Objetos. Descreva essa importância.

5. Quando começamos a desenvolver um projeto no IDE Lazarus, é necessário salvá-lo. No ato de salvar, percebemos a geração de diversos arquivos, os quais possuem extensões diferentes. Quais extensões são criadas e o que são esses arquivos?

6. Nesta aula vimos a estrutura do código fonte, ou seja, a *unit*. Como ela é estruturada? Descreva a seção.

Poste suas respostas no AVEA.

Aula 3 – Conceitos básicos de linguagem Pascal e paleta de componentes

Objetivos

Conhecer os conceitos básicos de linguagem Pascal.

Compreender aplicação com o uso de tipos de dados: *string*, inteiro e real.

Conhecer alguns componentes da paleta de componentes, propriedades e eventos.

3.1 *Object* pascal

A linguagem *Object* Pascal é proveniente da linguagem Pascal, que durante a década de 1980 fez grande sucesso comercial através do Turbo Pascal.

Como a linguagem de programação Lazarus utiliza a linguagem *Object* Pascal, mostraremos nesta Aula 3 um resumo das estruturas aplicadas no *Object* Pascal, tais como: estruturas de bloco, de controle (condicionais), de repetições (laços ou *loops*), funções e procedimentos, bem como criação de identificadores: variáveis, constantes e tipos de dados e operadores.

Para saber mais sobre *Object* Pascal, consulte o *site* http://www.freepascal.eti.br/

Todo o resumo será de suma importância para nos auxiliar na construção das aplicações colocadas neste material.

3.2 Identificadores

São os nomes atribuídos às constantes e variáveis que receberão as informações. Um identificador válido na linguagem Pascal é qualquer sequência de caracteres que obedeçam às seguintes regras:

a) devem começar por um caractere alfabético – (a, b, ..., z);

b) podem ser seguidos por mais caracteres e/ou numéricos, ou o caractere _;

c) não é permitido o uso de caracteres especiais;

d) têm que ser diferentes das palavras reservadas do Pascal, como os comandos, funções, etc.

Exemplos:

- Identificadores válidos: A, Nota, P1, Meu_Identificador.
- Identificadores inválidos: 1ª ,E(13), A:B.

3.3 Palavra reservada

Identifica uma função determinada.

Exemplos:

absolute, and, array, begin, case, const, div, do, downto, else, end, for, function, if, mod, not, or.

3.4 Comentários

Como ponto inicial, você deve saber como fazer comentários em seu programa Pascal. O Lazarus suporta os tipos de comentários abordados abaixo:

{Comentários utilizando chaves}
(* Comentários usando parênteses e asterisco *)
// Comentários no estilo C++

3.5 Variáveis

A linguagem *Object* Pascal, diferentemente da linguagem C e C++, não considera maiúsculas e minúsculas como formas diferentes. Por exemplo, 'X' e 'x' se referem a uma mesma variável.

3.6 Constantes

São valores que não são modificados durante toda a execução do programa. São armazenados na memória para ser usados em cálculos e processamentos, mas não são modificados.

Exemplo:

CONST
PI = 3.1415926;

3.7 Operadores

Os operadores são símbolos que habilitam a manipulação de todos os tipos de dados. Em *Object* Pascal os operadores estão definidos basicamente em: operador de atribuição (Quadro 3.1), operadores lógicos (Quadro 3.2), operadores relacionais (Quadro 3.3) e operadores aritméticos (Quadro 3.4).

Quadro 3.1: Operador de atribuição

Operador	Símbolo	Aplicação
de atribuição	:=	A:= B +1;

Fonte: Elaborado pelo autor

Quadro 3.2: Operadores lógicos

Operadores lógicos	Símbolos	Aplicação
E	And	A and B
Ou	Or	A or B
Não	Not	Not A

Fonte: Elaborado pelo autor

Quadro 3.3: Operadores relacionais

Operadores Relacionais	Símbolos	Aplicação
Comparação	=	A = B
Diferença	<>	A <> B
Maior	>	A > B
Menor	<	A < B
Maior ou igual	>=	A >= B
Menor ou igual	<=	A <= B

Fonte: Elaborado pelo autor

Quadro 3.4: Operadores aritméticos

Operadores aritméticos	Símbolos	Aplicação
Adição	+	A + B
Subtração	-	A –B
Multiplicação	*	A * B
Divisão real	/	A / B
Divisão inteira	Div	A div B
Resto da divisão	Mod	A mod B

Fonte: Elaborado pelo autor

3.8 Tipos de dados

A definição de tipos de dados corretos para as variáveis é muito importante, pois permitem a economia de memória.

3.8.1 Numéricos inteiros

Integer, byte, shortint, word, smallint e longint.

3.8.2 Numéricos reais

Real, single, double e extended.

3.8.3 Booleanos

Boolean, bytebool, wordbool, longbool.

3.8.4 *Caracter*

Char, ansichar, widechar.

3.8.5 *String* (conjunto de caracteres)

String, shortstring, ansistring, widestring.

3.8.6 Tipos definidos pelo usuário

Números inteiros, *strings* e ponto flutuante frequentemente não são suficientemente adequados para representar variáveis nos problemas do mundo real. Nesse caso, deve-se criar tipos complexos para melhor representar variáveis de um determinado problema. Em Pascal, os tipos definidos pelo usuário geralmente são da forma de registros (*records*) ou objetos.

a) *Arrays*

A linguagem *Object* Pascal permite criar *arrays* (matrizes) de qualquer tipo. Por exemplo, uma variável declarada como *array* com oito números inteiros:

```
Var
  A: array[0..7] of integer;
```

b) *Records*

O tipo *record* permite que se agrupe em uma única variável, um conjunto de tipos de dados diferentes. A aplicação desse tipo de dado é bastante semelhante ao conceito de registro, ou seja, uma variável do tipo *record* é composta por vários campos.

record é tipo estruturado em Pascal, equivalente ao *struct* do C.
{ Pascal }

```
Type
      Meu_Registro = record
        I: integer;
        D: double;
end
```

3.9 Blocos de instruções

3.9.1 Estruturas de controle ou decisão

No *Object* Pascal existem dois comandos para tomar decisões durante a execução de um programa. São as instruções *if* e *case*.

a) comando *if*

A instrução *if* permite controlar a execução de outras instruções (ou bloco de instruções) do programa, dependendo do valor de uma variável ou expressão booleana.

A instrução *if* comporta-se conforme as sintaxes abaixo:

- Sintaxe 1:

```
if expressão booleana then instrução;
```

- Sintaxe 2:

```
if expressão booleana then
    instrução1
else
    instrução2;
```

- Sintaxe 3:

```
if expressão booleana then
     begin
        instrução;
     end
else
     begin
        instrução;
     end;
```

b) O comando *case*

O comando *case* é equivalente a uma série de comandos *if*. Se a variável de controle é igual a uma das constantes de uma lista, a instrução ou bloco de instruções associadas à lista são executados. Se a variável de con-

trole não é igual a nenhuma das constantes, a declaração *case* não terá nenhum efeito e o computador passa à instrução seguinte do programa.

- Sintaxe 1:

```
case variável-controle of
                constante1: instrução1;
                constante2: instrução2;
...
else
                instrução1;
end;
```

- variável-controle: é uma variável do tipo inteiro, *caracter* ou definida pelo usuário, mas não pode ser do tipo real.

- constantes: pode ser um único valor, vários valores isolados separados por vírgula ou uma faixa de valores indicados pelo valor mínimo e máximo separados por dois pontos finais (..).

3.9.2 Estruturas de repetição

Na linguagem Pascal estão disponíveis três estruturas de repetição que fornecem uma grande flexibilidade na representação de um processo repetitivo. Um *loop* executa uma instrução (ou bloco de instruções) repetidamente.

As três estruturas de repetição em Pascal são representadas pelas palavras reservadas *for*, *while* e *repeat*.

a) comando – *for*

O comando *for* é usado para repetir várias vezes a mesma instrução ou bloco de instruções. Durante a execução, uma variável usada como índice assume uma série de valores, desde um dado valor inicial até um valor final. Na forma *for-to*, o índice assume valores crescentes; na forma *for-downto*, o índice assume valores decrescentes.

A forma geral da declaração é a seguinte:

- Sintaxe 1:

```
for índice := inicial to final do instrução
```

- Sintaxe 2:

```
for índice := inicial to final do
begin
    instrução1;
    instrução2;
    ...
end;
```

- Sintaxe 3:

```
for índice := inicial downto final do
begin
    instrução1;
    instrução2;
    ...
end;
```

- **Índice:** O índice é uma variável ordinal. O índice pode ser do tipo inteiro, caracter, *byte* ou definido pelo usuário. Não pode ser uma variável real.

- **Inicial:** Valor inicial da variável índice.

- **Final:** Valor final da variável índice.

- **Instrução:** Qualquer instrução Pascal.

É importante notar que se **inicial** for maior ou igual a **final**, as instruções contidas no loop do tipo *for-to* serão simplesmente ignoradas. O mesmo acontecerá em um *loop* do tipo *for-downto* se **inicial** for menor que **final.**

b) O comando *repeat*

A instrução *repeat* faz com que uma instrução (ou bloco de instruções) seja executada repetida vezes, até que certa condição seja satisfeita. Observe que na forma composta da declaração, as palavras *begin* e *end* não são usadas para delimitar as declarações que fazem parte do *loop*; este sempre começa pela palavra *repeat* e termina na palavra *until*.

- Sintaxe 1:

```
repeat instrução until condição;
```

- Sintaxe 2:

```
repeat
    instrução1;
    instrução2;
    ...
until condição;
```

- Condição é uma expressão booleana.
- Instrução é qualquer instrução Pascal.

Como a *condição* é verificada no final do *loop*, todo *loop* do tipo *repeat* é executado pelo menos uma vez. Se a condição nunca assume o valor *True*, o *loop* só para se o programa for interrompido manualmente.

c) O comando *while*

O loop *while* é semelhante ao loop *repeat*. A principal diferença está no fato de que o loop *repeat* é sempre executado pelo menos uma vez; o loop *while* pode não ser executado nenhuma vez.

- Sintaxe 1:

```
while condição do instrução
```

- Sintaxe 2:

```
while condição do
begin
    instrução1;
    instrução2;
    ...
end;
```

- Condição é uma expressão booleana.
- Instrução é uma instrução Pascal. Na forma composta do *loop while* não é preciso colocar um ponto e vírgula depois da última instrução.

3.9.3 Procedimentos e funções

Procedimentos (*procedures*) e Funções (*functions*) são partes do programa que podem executar uma tarefa específica quando chamamos um dos módulos do programa. Após a execução do procedimento ou função, o controle da execução retorna ao módulo do programa que executou a chamada.

Os procedimentos e funções são executados de forma bastante semelhante, porém, uma função retorna um valor de retorno, enquanto que um procedimento apenas executa um bloco de comandos.

Observe o procedimento **ReajustaSalario**, conforme Figura 3.1.

```
procedure RajusteSalario;
begin
     if salario > 600 then
        salario: = salario * 1.05
     else
         salario := salario *1.10
end;
```

Figura 3.1: Procedimento ReajustaSalario
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Agora outra etapa do programa, só que usando a *function*, conforme Figura 3.2.

```
function HouveReajuste: boolean
begin
      if salario > 600 then
         result:= false
      else
         result:= true
end;
```

Figura 3.2: Função HouveReajuste
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Note que na implementação da função é necessário indicar o tipo de retorno da função.

Abaixo duas formas de chamar o procedimento e a função, como mostrado respectivamente na Figura 3.3 e Figura 3.4.

```
begin
      While salario <600 do
            ReajustaSalario;
end;
```

Figura 3.3: Chamando o procedimento ReajustaSalario
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

```
begin
     if HouveReajuste then
        ShowMessage('Resuste')
end;
```

Figura 3.4: Chamando a função HouveReajuste
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

A Linguagem Pascal permite que se utilizem parâmetros na implementação de procedimentos e funções. O Pascal permite passagem de parâmetros por valor ou por referência. Os parâmetros passados podem ter um tipo básico ou um tipo definido pelo usuário ou *arrays abertos* (que serão abordados posteriormente neste material).

3.10 Uma aplicação usando tipo de dado *string*

Com o resumo da linguagem Pascal feito no início desta Aula, podemos desenvolver os aplicativos para implementar os conceitos até agora abordados.

Para desenvolver nossos futuros projetos, teremos que reconfigurar nosso IDE Lazarus, pois inicialmente ele é configurado para que abra sempre ao iniciar o último projeto. Algumas vezes essa ação irá nos atrapalhar.

Então, vamos clicar no menu Ambiente e acessar opções, onde aparecerá a janela conforme Figura 3.5, e desmarcar a opção: abrir o último projeto ao iniciar.

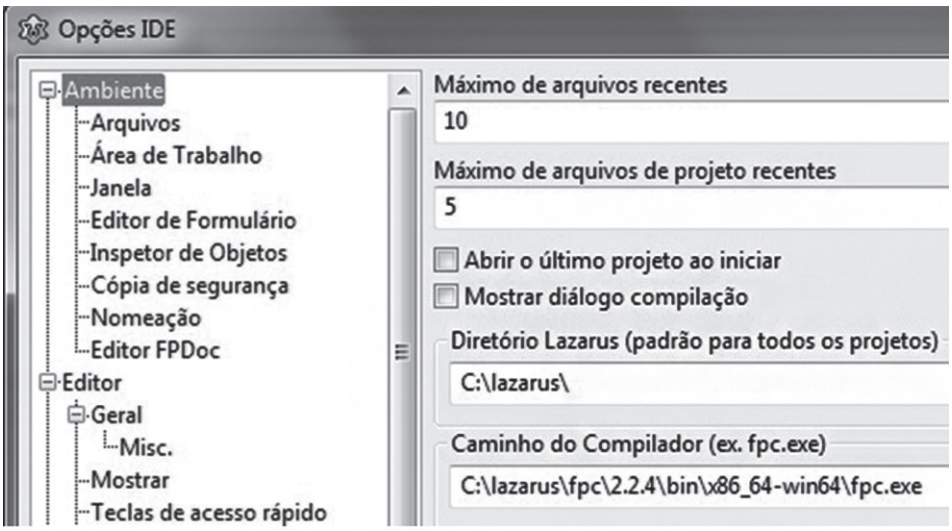


Figura 3.5: Janela Opções do menu Ambiente
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Com a nova configuração, vamos realizar nossa Aplicação de número 2 (dois). Para isso, abriremos a nossa pasta Programação_Ambiente_Gráfico_Lazarus, acessaremos a subpasta Programas e criaremos uma pasta chamada Aplicacao_02 para armazenarmos os arquivos dessa aplicação gerados do Lazarus.

Com a pasta *Aplicacao_02* criada, clique no ícone do Lazarus presente na sua Área de trabalho. Quando o IDE estiver aberto, acesse o menu projeto e clique em Novo Projeto e consequentemente será aberta uma outra janela, conforme Figura 3.6; mantenha o item Aplicação selecionado e clique em **OK**.

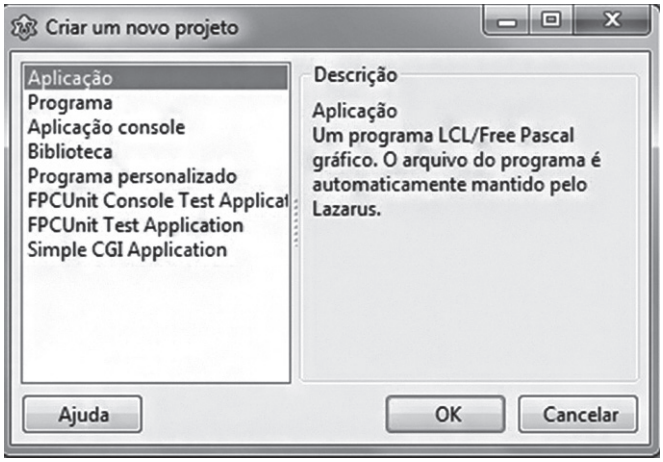


Figura 3.6: Janela selecionando uma nova aplicação
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Após selecionar Novo Projeto, coloque no *Form1* os seguintes objetos, de acordo com a Figura 3.7. Importante ressaltar que todos os componentes ou objetos estão localizados na paleta ou aba *Standard*.

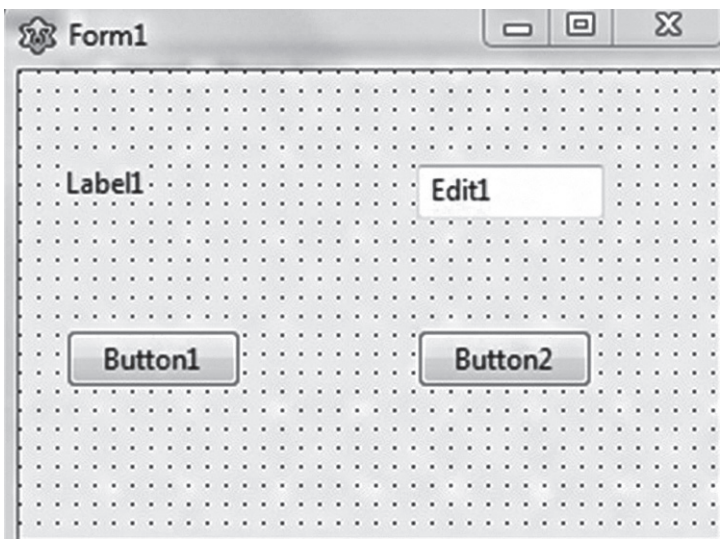


Figura 3.7: Modelo do projeto Aplicacao_02
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Agora partiremos para o próximo passo: alterar algumas propriedades dos componentes e, no final, o projeto terá um novo visual, conforme Figura 3.8.

Figura 3.8: Novo visual do projeto Aplicacao_02
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Essa aplicação tem como principal objetivo mostrar como o Lazarus lida com o tipo de dados *String* e como o manipula.

a) No formulário, clique no *label1*, no *Object inspector*, localize a propriedade *caption* e altere *label1* para: **Digite seu nome:.**

b) Selecione o *Edit1*, e no *Object inspetor*, procure a propriedade *Text* e delete a informação presente.

c) Clique no *button1* e altere o *caption* para **OK.**

d) Agora faça a mesma coisa com *button2*, mudando o *caption* para **Limpar.**

e) Clique no *Form1* ou selecione no *Object inspector* o *Form1*, conforme Figura 3.9 e altere seu *caption* para Aplicação com String.

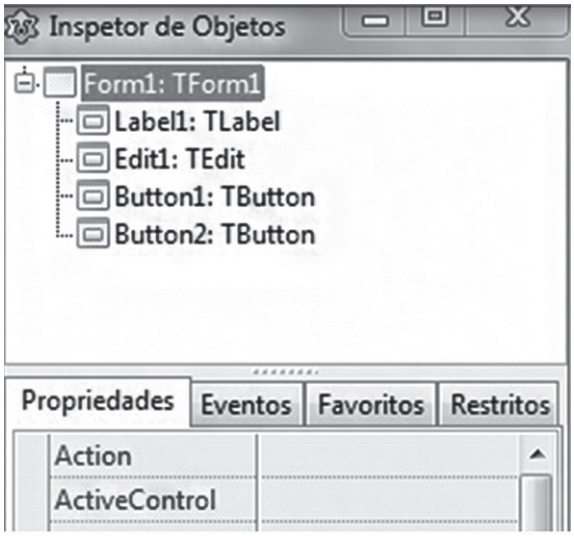


Figura 3.9: Visualização dos componentes do Inspetor de Objetos
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

f) Com o novo visual estabelecido, iremos salvar nossos arquivos. Para isso, acesse o menu Arquivo > Salvar Tudo. Fique atento às teclas de atalhos que nos ajudarão a acelerar nosso trabalho ao desenvolver as aplicações.

g) Ao clicar em Salvar Tudo, será aberta a janela Salvar *Unit1*, conforme Figura 3.10; nomeie como *uAplicacao_02*; essa letra inicial não é obrigatória, porém adotaremos essa padronização.

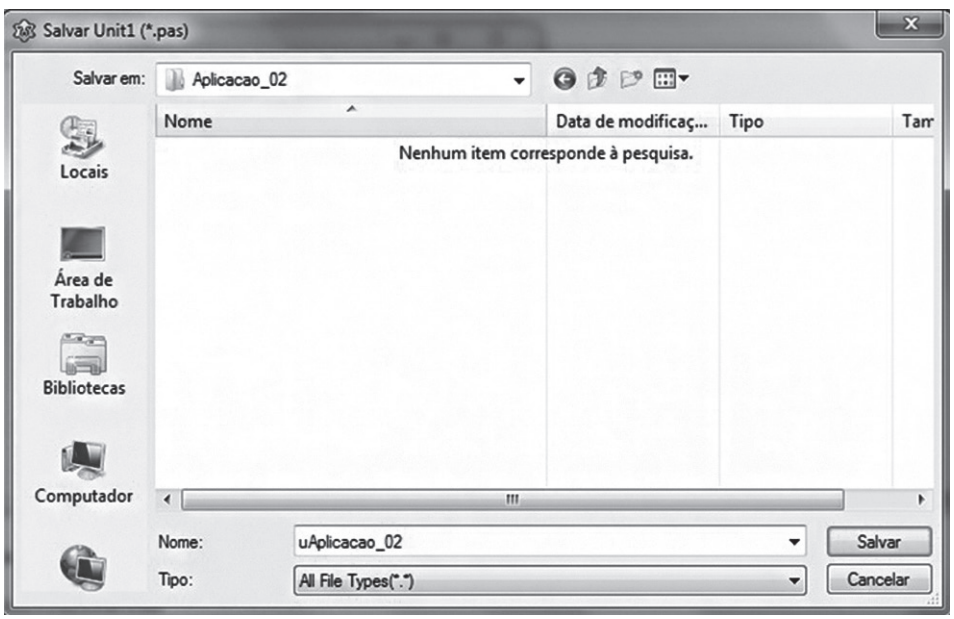


Figura 3.10: Janela do Salvar *Unit1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

h) Dando um nome a *Unit1* e clicando em Salvar, aparecerá uma janela conforme Figura 3.11; clique em **Manter nome**.

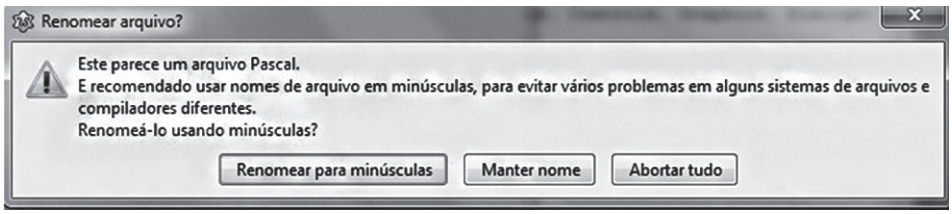


Figura 3.11: Janela do Renomear o Arquivo da Unit
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

i) Clicando em Manter nome, abrirá outra janela para *Salvar Project1*, conforme Figura 3.12; dê o nome de *prj_Aplicacao_02*. Como padronização, iniciaremos os nomes de nossos futuros projetos como prj_(nome do projeto).

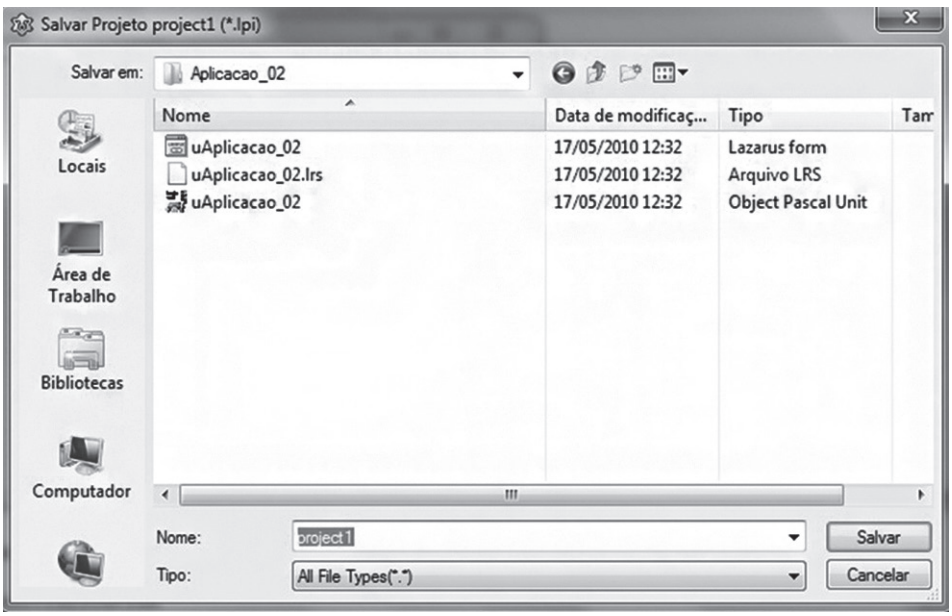


Figura 3.12: Janela de Salvar Projeto *Project1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

j) Agora com o projeto salvo, podemos executá-lo, porém ele não exercerá nenhuma função, pois os *buttons* ainda não foram implementados. São os botões que exercem função de disparar o fornecimento da solução do problema. Para executar o projeto, localize o menu *Executar* e escolha o submenu *Executar* ou clique apenas em F9 de seu teclado. Executando o aplicativo, o projeto buscará algum erro de compilação, sintaxe; caso não ocorram erros, surgirá a janela de Mensagens, conforme Figura 3.13 e aparecerá a janela compilada, conforme Figura 3.14.

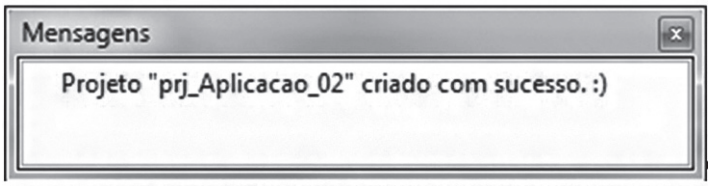


Figura 3.13: Janela Mensagens emitindo execução com sucesso
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Figura 3.14: Projeto em execução
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

k) Agora a etapa mais importante do projeto, a implementação dos *buttons*; para isso, selecione o botão **OK** ou *button1*, localize no *Object inspector* a aba *Events* e dê um duplo clique no evento *OnClick* que abrirá a *Unit* chamada *uAplicacao_02*, conforme Figura 3.15. Podemos realizar a mesma ação apenas dando um duplo clique no botão **OK**.

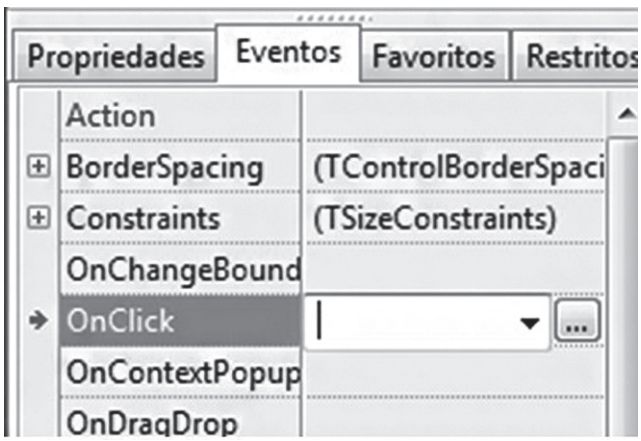


Figura 3.15: Evento *OnClick*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

l) Com a janela da *Unit* aberta, agora podemos digitar o código. Perceba que quando a *Unit* surgiu, o procedimento foi criado automaticamente pelo Lazarus, conforme Figura 3.16.

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Figura 3.16: Procedimento do OK
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

m) Agora apenas dentro do procedimento, ou seja, dentro do corpo (*begin ...end*), digite o código abaixo, e pronto, o Botão **OK**, já exercerá sua função.

```
if Edit1.Text= '' then
ShowMessage('Digite seu nome! ')
else
ShowMessage('Saudações ' + Edit1.Text + ', estamos usando uma STRING.');
```

n) Clicaremos agora no botão **Limpar** para implementar a função apagar a informação fornecida pelo usuário dentro do *Edit1*. Para isso dê um duplo clique no botão **Limpar** ou *button2* e digite o código dentro do procedimento gerado automaticamente, conforme Figura 3.17.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
end;
```

Figura 3.17: Procedimento Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Para que o comando realmente funcione, ou seja, limpe o componente Edit, é necessário que a informação atribuída ao componente esteja entre aspas (sem nenhum caractere interno às aspas, de acordo com a Figura 3.17.

o) Com as implementações concretizadas, podemos executar o projeto. Clique em F9 e aparecerá a janela em execução de acordo com a Figura 3.18.

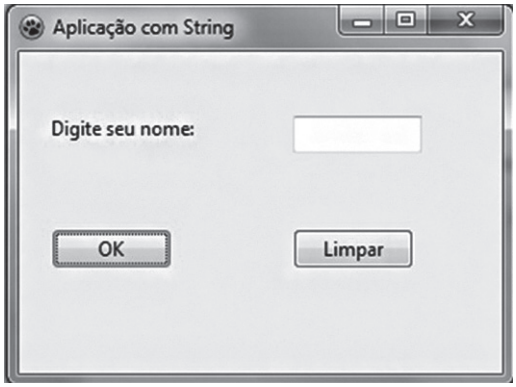


Figura 3.18: Procedimento OK
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

p) Sem digitar nada, clique diretamente no botão **OK** e analisaremos o código do botão **OK**:

```
if Edit1.Text= ' ' then
ShowMessage('Digite seu nome!')
else
ShowMessage('Saudações ' + Edit1.Text + ', estamos usando uma STRING.');
```

O código acima começa com uma estrutura de controle analisando se o *Edit1* não possui informação; se essa condição for *true*, será emitida uma mensagem solicitando que o usuário forneça essa informação, conforme Figura 3.19.

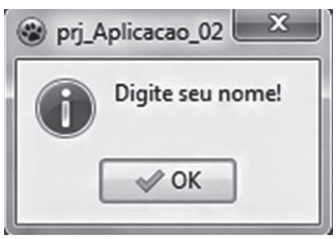


Figura 3.19: Mensagem emitida sem a digitação do nome
Fonte: *Print screen* do Lazarus IDE v0.9.28.2 realizado pelo autor

E caso o usuário tenha fornecido a informação, o programa emitirá outra mensagem fornecido na cláusula (*else*), conforme Figura 3.20.

Figura 3.20: Mensagem emitida com a digitação do nome
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Vamos praticar um pouco? Crie um novo projeto e solicite o curso que você deseja cursar.

3.11 Uma aplicação usando tipo de dado inteiro

Agora ressalto que essa aplicação tem como principal objetivo, mostrar como o Lazarus manipula o tipo de dado Inteiro.

Para desenvolver nossa próxima aplicação, criaremos a subpasta Aplicacao_03 na mesma pasta das aplicações anteriores. Clique no ícone do Lazarus presente na sua Área de trabalho. Quando o IDE estiver aberto, acesse o menu projeto e clique em Novo Projeto, de acordo com o procedimento colocado na *prj_Aplicacao_03*.

a) Inicialmente, coloque no *Form1* os componentes de acordo com a Figura 3.21. Importante ressaltar que ainda trabalharemos a aba *Standard*.

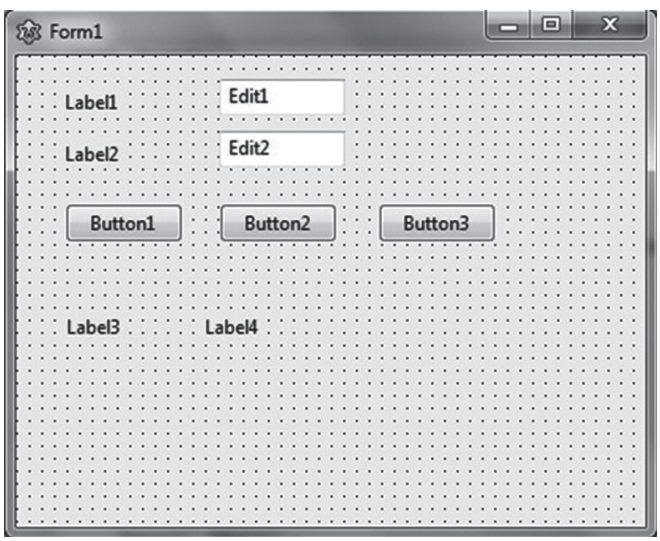


Figura 3.21: Janela inicial da Aplicação 03
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Agora partiremos para o próximo passo: alterar algumas propriedades de cada componente e, no final, o projeto terá o visual, conforme Figura 3.22.

Figura 3.22: Janela após reconfiguração da Janela inicial
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

c) Clique no *label1* e altere no *Object* Inspector a propriedade *caption* para **Digite um valor:.**

d) Faça o mesmo procedimento como *label2*, alterando a propriedade *caption* para **Digite outro valor:.**

e) A mesma coisa com o *label3*, alterando *caption* para O **somatório é:.**

f) No *label4*, na propriedade *caption*, apague a informação *label4*; observe que na Figura 3.22 ficou invisível o *label4*, porém quando o programa fornecer a resposta se tornará visível.

g) Selecione o *Edit1*, delete a informação *Edit1* da propriedade *Text*, deixe-a sem nada.

h) O mesmo procedimento com o *Edit2*, delete *Edit2* da propriedade *Text*:, deixe-a sem nada.

i) Altere a propriedade *caption* dos *buttons*, conforme informações abaixo:

- *button1* para **Calcular;**
- *button2* para **Limpar;**
- *button3* para **Fechar**.

j) Agora altere a propriedade *caption* do *Form1* para **Somatório.**

k) O próximo passo é salvar nossos arquivos. Para isso, acesse o menu *Arquivo* > Salvar Tudo; ao clicar em Salvar Tudo, localize a subpasta Aplicacao_03 e na janela Salvar *Unit1* forneça o nome *uAplicacao_03*.

l) Dando um nome à *Unit1* e clicando em Salvar, aparecerá mais uma vez outra janela; clique em Manter nome.

m) Clicando em Manter nome, abrirá outra janela para Salvar *Project1*; agora dê o nome de *prj_Aplicacao_03*.

n) Agora com o projeto salvo, podemos executá-lo; clique em F9. O projeto será executado, conforme Figura 3.23, porém ele não exercerá nenhuma função, pois os *buttons* ainda não foram implementados.

Executado o projeto, apesar da ausência das suas funcionalidades, houve a execução sem erros sintáticos; até então, podemos concluir que os futuros erros que ocorrerem, não foram ocasionados por códigos até o momento inseridos.

Figura 3.23: Janela de execução da Aplicação 03
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

o) Implementaremos agora os códigos dos botões: **Calcular, Limpar** e **Fechar:** dê um duplo clique no botão **Calcular** e insira o código conforme Figura 3.24; não esqueça que o procedimento é gerado automaticamente. Atenção, insira o código entre *begin* e *end*; atente também para declaração da variáveis na cláusula *var*.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   num1, num2, result: integer;
begin
   num1:=StrtoInt(Edit1.Text);
   num2:=StrtoInt(Edit2.Text);
   result:= num1 + num2;
   Label4.Caption:= IntToStr(result);
end;
```

Figura 3.24: Procedimento do *button1* ou Calcular
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Todos os componentes do Lazarus são do tipo *string*, mas para manipular outros tipos de dados é necessário que sejam utilizadas algumas funções de conversão.

De acordo com a Figura 3.24, o procedimento utilizou duas funções de conversões, a função (**StrToInt**) para o sistema compreender a informação digitada nos *Edits*, que transforma a informação do componente, ora *string*, para o tipo Inteiro, para que o sistema possa utilizar as operações aritméticas, e outra (**IntToStr**), que faz a ação contrária, ou seja, com o resultado do cálculo aritmético realizado pelo sistema, teremos que fornecer o resultado em outro componente do Lazarus, no caso o *Label4*, para que o usuário visualize o resultado.

p) Agora implemente o código do botão **Limpar**, dando um duplo clique no botão referente à funcionalidade **Limpar**, conforme a Figura 3.25.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   Edit1.Text:= '';
   Edit2.Text:='';
   Label4.Caption:='';
end;
```

Figura 3.25: Procedimento do *button2* ou Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

q) Agora insira o código do botão **Fechar**, dando um duplo clique no respectivo botão para ser gerado automático o procedimento *OnClick* e digite o código, conforme a Figura 3.26.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  close;
end;
```

Figura 3.26: Procedimento do *button3* ou Fechar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

r) Neste momento salve tudo. Não esqueça que sempre devemos salvar qualquer alteração realizada e desejada; por último, teste seu projeto.

3.12 Uma aplicação usando tipo de dado real

Essa aplicação tem como principal objetivo, mostrar como o Lazarus lida com o tipo de dado real e o e manipula.

Para desenvolver nossa próxima aplicação, criaremos a subpasta Aplicacao_04. Abra o IDE do Lazarus e gere um *Novo Projeto.*

a) Coloque no *Form1* os componentes de acordo com a Figura 3.27. Importante ressaltar que nessa aplicação todos os componentes estão na aba *Standard*, exceto os *BitBtns* que são fornecidos na aba *Additional*:

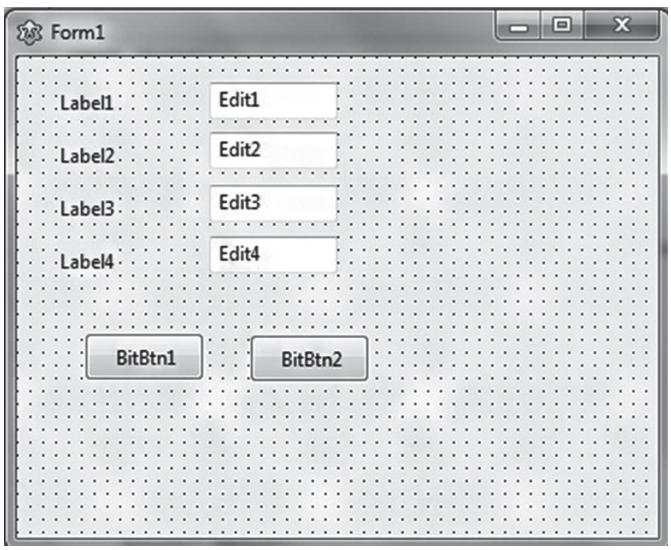


Figura 3.27: Janela modelo da Aplicação 04
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Agora partiremos para o próximo passo: alterar as propriedades dos componentes e, no final, o projeto terá o visual conforme Figura 3.28.

Figura 3.28: Janela após reconfiguração da aplicação 04
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

c) Agora altere a propriedade *caption* do *Form1* para **Média Aritmética.**

d) Mantendo a seleção do *Form1*, altere a propriedade Height (comprimento) de 400 para 300.

e) Localize a propriedade Width (altura) do *Form1* para 250.

f) Clique no *label1* e altere a propriedade *caption* para **Digite um valor:.**

g) Faça o mesmo procedimento como *label2*, alterando a propriedade *caption* para **Digite outro valor:.**

h) Faça o mesmo com o label3, alterando *caption* para **Outro valor:.**

i) No label4, altere a propriedade *caption* para **Média:.**

j) Selecione todos os *labels* segurando a tecla *Shift* e clicando em cada *Label* até selecionar todos. Após a seleção, localize a propriedade *Font* e altere para *Cambria*, Estilo para *Negrito* e *Tamanho 11*, conforme Figura 3.29.

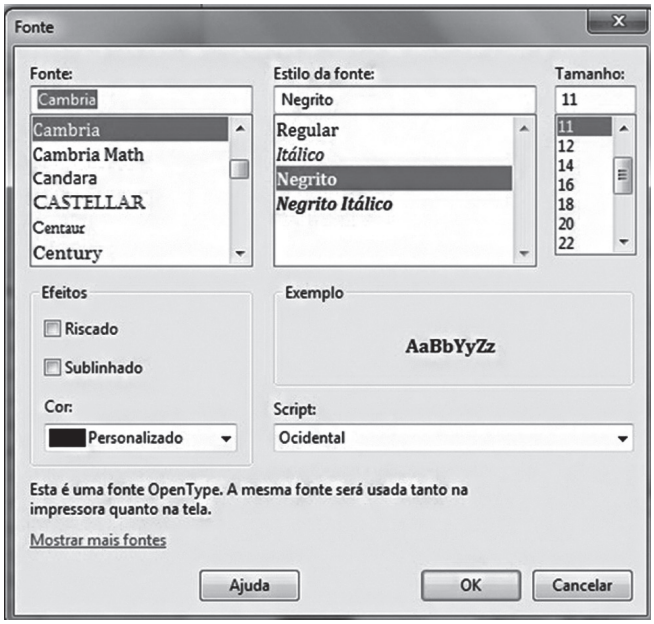


Figura 3.29: Janela para configurações da Fonte
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

k) Selecione o *Edit1*, delete a informação *Edit1* da propriedade *Text*, deixando-a sem nada.

l) Faça o mesmo procedimento com o *Edit2*: delete *Edit2* da propriedade *Text*, deixando-a sem nada.

m) Faça o mesmo procedimento com o *Edit3*: delete *Edit3* da propriedade *Text*, deixando-a sem nada.

n) Delete *Edit4* da propriedade *Text*, deixando-a sem nada.

o) O *Edit4* será um componente especial, pois servirá também para fornecer a resposta. Para tanto, o usuário não poderá ter acesso a esse componente. Para isso, conheceremos a propriedade *Enabled*, que desabilita o acesso a esse componente.

p) Clique no *BitBtn1* e selecione na propriedade *Kind*: *bkOk*.
A diferença entre *Button* e *BitBtn*, é que o *BitBtn* pode incorporar um ícone no componente.

q) Faça a mesma coisa com o *BitBtn2*, selecionando na propriedade *Kind*: *bkClose*. Esta seleção já fornece a função Fechar encapsulada (conceito de Orientação a Objeto), ou seja, se abrirmos o código-fonte não veremos, mas ao executar o projeto e clicar nesse botão, ele exercerá a função de fechar.

r) Agora para melhor interação com o usuário, iremos fornecer uma *DICA*, que é colocada na propriedade ***Hint***.

s) Clique no *BitBtn1*, localize a propriedade *Hint* e digite **Clique aqui para executar a média... .**

Para podermos visualizar o efeito, teremos que alterar outra propriedade chamada *ShowHint de False* para *True*, ou seja, que permite visualizar a *DICA*. Se desejarmos visualizá-la, basta executar o projeto e passar o *mouse* sobre o *BitBtn1*.

t) O próximo passo é salvar nossos arquivos. Para isso, acesse o menu Arquivo > Salvar Tudo. Ao clicar em Salvar Tudo, localize a subpasta Aplicacao_04, e na janela Salvar *Unit1* forneça o nome *uAplicacao_04*.

u) Quando solicitado para Salvar *Project1*, dê o nome de *prj_Aplicacao_04*.

v) Agora com o projeto salvo, podemos executá-lo. Clique em F9 e o projeto será executado; porém, ele não exercerá nenhuma função, exceto pelo *BitBtn2* (**Fechar**).

w) Implementaremos agora o código do *BitBtn1* (**OK**). Dê um duplo clique no *BitBtn1* e insira o código conforme Figura 3.30. Atenção, insira o código entre os bloco de comando *begin* e *end*. Atente também para a declaração da variáveis na cláusula *var*.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
   valor1, valor2, valor3, media: real;
   confirma: integer;
   resposta: String;
begin
     val(Edit1.Text, valor1, confirma);
     val(Edit3.Text, valor2, confirma);
     val(Edit3.Text, valor3, confirma);
     media:= (valor1 + valor2 + valor3) /3;
     Str(Media:3:2, resposta);
     Edit4.Text:= resposta;

end;
```

Figura 3.30: Procedimento *BitBtn1* (OK)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Na aplicação anterior, ou seja Aplicacao_03, vimos que os componentes do Lazarus são todos do tipo *STRING*, e que para manipular outros tipos de dados são necessárias funções de conversão. Nesta aplicação, utilizamos mais duas: a função (**Val**) para o sistema compreender que a informação digitada nos Edits seja convertida de Strings para o tipo *real*, para que o sistema possa utilizar as operações aritméticas, e a função (Str), que faz a ação contrária a fim de que o resultado do cálculo aritmético realizado pelo o sistema possa ser visualizado no *Edit4*.

x) Compreendendo melhor o código do *Bitbtn* (Ok), conforme Figura 3.30.

Na cláusula *var*, declaramos **valor1, valor2 e valor3**, cada um deles para receber do usuário um valor que será fornecido pelos *Edit1*, *Edit2* e *Edit3*. Como já é notório, os componentes do Lazarus reconhecem como tipo *String* todas as informações digitadas dentro deles, impedindo o sistema de realizar qualquer operação aritmética. Portanto a função **Val** é composta por três argumentos, conforme abaixo descrito:

Val(1º Argumento, 2º Argumento, 3º Argumento.)

- *1º Argumento*: origem da informação, ou seja, em qual componente do Lazarus ela é digitada;

- *2º Argumento*: variável criada para receber a informação convertida de *string* para real;

- *3º Argumento*: variável controladora, declarada aqui como (*confirma*) responsável em analisar se cada dígito fornecido ao componente é inteiro. Importante relembrar que no número real com padrão americano, separa-se o parte inteira da fracionária com o ponto(.).

Após a conversão, teremos os componentes *Edit1*, *Edit2* e *Edit3* com informações do tipo *string*, mais as variáveis **valor1**, **valor2** e **valor3** do tipo real. Sendo assim, podemos utilizar as variáveis do tipo real para realizar a operação aritmética (somar os três valores, dividir por três) e armazenar o resultado do cálculo na variável **Media**, que também foi declarado como real;

Em posse do resultado armazenado na variável **Media**, teremos que torná-la visível ao usuário e utilizar um componente, no caso, colocando o *Edit4* para esse propósito; mas para que isso ocorra, teremos que converter o resultado armazenado na variável **Media** do tipo *real* para *string*.

Já a função (**Str**) éformada por dois argumentos, conforme abaixo:

Str(1° Argumento, 2° Argumento)

- *1° Argumento*: variável que tem a informação em do tipo real;
- *2° Argumento:* valor convertido do *1° argumento* armazenado na variável criada para receber o tipo *string.*

De acordo com o código ***Str***(***Media*:3:2, *respost***a), a variável ***Media:3:2***, tem o valor em real com a formatação de duas casas decimais para parte real. Então a ***Str*** pega informação de ***Media*** e converte o respectivo valor em tipo *string* e armazena na variável (resposta) do tipo *string*.

Como a variável (**resposta**) e componente *Edit4* são do mesmo tipo, ou seja, *string*, podemos mostrar o resultado para o usuário através do comando ***Edit4.Text:= resposta***;

y) Neste momento salve tudo e teste o projeto.

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa e solicite como resultado as operações aritméticas básicas.

Resumo

Nesta aula abordamos conceitos básicos da Linguagem Pascal, tais como o uso de identificadores, estrutura de controle, repetição, funções e procedimentos e tipos de dados. Desenvolvermos aplicações que nos permitiram conhecer componentes da paleta de componentes e suas propriedades e eventos e aprendemos como utilizar as funções de conversões que manipulam os tipos de dados: *string*, inteiro e real.

Atividades de aprendizagem

1. Nesta aula vimos a importância das propriedades de um objeto, pois a familiaridade com elas é de suma relevância para que possamos desenvolver nossos projetos. Então, descreva algumas propriedades abaixo:

a) *Caption*
b) *Text*
c) *Font*
d) *Enabled*
e) *Visible*
f) *Top*
g) *Left*
h) *Width*
i) *Height*
j) *Name*
k) *Hint*
l) *ShowHint*
m) *ReadOnly*
n) *CharCase*

2. Nesta aula percebemos que os objetos da paleta de componentes suportam informações do tipo de dado *STRING,* e, em nossos códigos colocados nas *units*, trabalhamos com funções de conversão. Em nossas aplicações, vimos *Str*, *IntToStr*, *StrToInt*, *Val*. Descreva-as.

3. Nesta aula desenvolvemos aplicações utilizando alguns componentes e o evento *Onclick*. Abaixo colocamos uma *Unit* através da qual podemos inferir quantos componentes há dentro do formulário, e por meio dos procedimentos podemos saber que ações eles realizam. Então, de acordo a *Unit* abaixo, identifique quantos componentes existem no formulário e quais são eles? Quais ações esses procedimentos realizam.

```
unit UAplicacao_04;
{$mode objfpc}{$H+}
interface
uses
Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
StdCtrls, Buttons;
type
 { TForm1 }
 TForm1 = class(TForm)
```

```
 BitBtn1: TBitBtn;
 BitBtn2: TBitBtn;
 Edit1: TEdit;
 Edit2: TEdit;
 Edit3: TEdit;

 Edit4: TEdit;
 Label1: TLabel;
 Label2: TLabel;
 Label3: TLabel;
 Label4: TLabel;
 procedure BitBtn1Click(Sender: TObject);
 private
 { private declarations }
 public
 { public declarations }
 end;

var
 Form1: TForm1;
implementation
{ TForm1 }
procedure TForm1.BitBtn1Click(Sender: TObject);
var
 valor1, valor2, valor3, media: real;
 confirma: integer;
 resposta: String;
begin
 val(Edit1.Text, valor1, confirma);
 val(Edit3.Text, valor2, confirma);
 val(Edit3.Text, valor3, confirma);
 media:= (valor1 + valor2 + valor3) /3;
 Str(Media:3:2, resposta);
 Edit4.Text:= resposta;
end;
initialization
 {$I UAplicacao_04.lrs}
end.
```

Poste suas respostas no AVEA.

Aula 4 – Estrutura de decisão, repetição e novos componentes

Objetivos

Implementar as estruturas de decisões no IDE Lazarus.

Implementar as estruturas de repetições no Lazarus.

Conhecer novos componentes da paleta de componentes e suas respectivas propriedades e alguns eventos.

Estrutura de decisão

4.1 Estrutura de decisão

4.1.1 Estrutura condição simples

a) Monte um formulário de acordo com a Figura 4.1, inserindo os componentes e configurações.

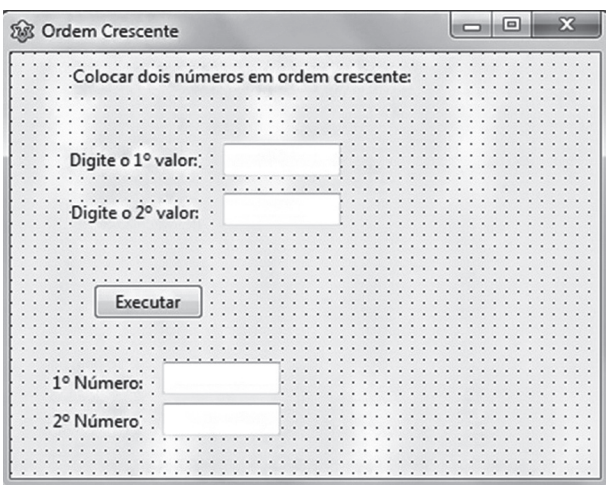


Figura 4.1: Modelo para prj_Aplicacao_05
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere a propriedade *caption* do *Form1* para **Ordem Crescente**.

c) Insira cinco *labels* da aba *Standard* e altere a propriedade *caption*:

- *label1* – *caption* para colocar dois números em ordem crescente;
- *label2* – *caption* para digite o 1º valor;
- *label3* – *caption* para digite o 2º valor;
- *label4* – *caption* para 1º número;
- *label5* – *caption* para 2º número.

d) Acesse a Aba *standard* e insira quatro *Edits* e altere suas respectivas propriedades – *Text*, deixando-as vazias, ou seja, apague a informação presente.

e) Insira um componente *button* da aba *standard* e altere a propriedade *caption* para **Executar**.

f) Altere a propriedade *Name* do *button1* para **btn_Executar**.

g) Antes de darmos continuidade, salvaremos agora nosso projeto. Para a *unit1* salve com o nome de *u_Aplicacao_05* e para o projeto com o nome *prj_Aplicacao_05*.

h) Agora vamos inserir o código botão **Executar**.

i) Dê um duplo clique do **btn_Executar** para gerar o evento *OnClick* e insira o código de acordo a Figura 4.2.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor1, valor2: integer;
begin
     valor1:=StrToInt(Edit1.Text);
     valor2:=StrToInt(Edit2.Text);
     if (valor1>=valor2) then
        begin
             Edit3.Text:= IntToSTr(Valor2);
             Edit4.Text:= IntToStr(Valor1);
        end;


end;
```

Figura 4.2: *OnClick* do *btn_Executar*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

O objetivo desse projeto é colocar em ordem crescente dois valores fornecidos pelo usuário nos componentes *Edit3* e *Edti4*.

Para compreender melhor o código da Figura 4.2, foram criadas duas variáveis: **valor1** e **valor2** do tipo inteiro, que são tratadas na estrutura condicional simples. Se a variável **valor1** for maior que, ou igual, à variável **valor2**, faremos com que o componente Edit3 receba o menor número e, consequentemente, o *Edit4* receberá o maior número.

j) Salve tudo e execute o projeto teclando F9.

4.1.2 Estrutura condição composta

Agora daremos continuidade com a estrutura condicional composta aprimorando o projeto *prj_Aplicacao_05*.

a) Gere o formulário de acordo com a **Figura 4.3**, inserindo os componentes e configurações.

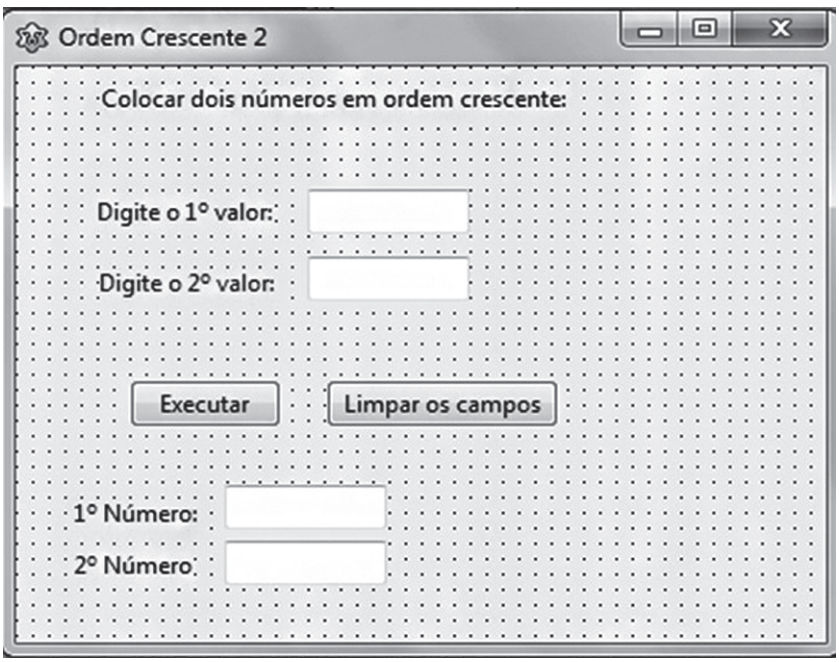


Figura 4.3: Formulário do *prj_Aplicacao_06*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere a propriedade *caption* do *Form1* para **Ordem Crescente 2.**

c) Insira cinco *labels* da aba *standard* e altere a propriedade *caption*:

- *label1 – caption* para **colocar dois números em ordem crescente;**
- *label2 – caption* para **digite o 1º valor;**
- *label3 – caption* para **digite o 2º valor;**
- *label4 – caption* para **1º Número;**
- *label5 – caption* para **2º Número.**

d) Agora na Aba *standard* insira quatro *Edits* e altere suas respectivas propriedades *Text*, deixando-as vazias, ou seja, apague a informação presente.

e) Insira um componente *button* da aba *standard* e altere a propriedade *caption* para **Executar**.

f) Insira mais componente *button* no *Form1* da aba *standard* e altere a propriedade *caption* para **Limpar os campos**.

g) Agora salvaremos nosso projeto. Para a *Unit1*, salve com o nome de *u_Aplicacao_06* e para o projeto com o nome *prj_Aplicacao_06*.

h) Agora vamos inserir o código botão **Executar**.

i) Dê um duplo clique do *button1* para gerar o procedimento *OnClick* e insira o código de acordo a Figura 4.4.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor1, valor2: integer;
begin
     valor1:=StrToInt(Edit1.Text);
     valor2:=StrToInt(Edit2.Text);
     if (valor1>=valor2) then
        begin
             Edit3.Text:= IntToSTr(Valor2);
             Edit4.Text:= IntToStr(Valor1);
        end
     else
         begin
              Edit3.Text:= IntToStr(valor1);
              Edit4.Text:= IntToStr(valor2);
         end;

end;
```

Figura 4.4: *OnClick* do *Button1* da *prj_aplicacao_06*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

O objetivo do projeto é aperfeiçoar o projeto *prj_Aplicacao_05*, quando as entradas dos valores serão tratadas agora em uma estrutura condicional composta; se a variável **valor1** for maior que, ou igual, a variável **valor2**, faremos com que o componente *Edit3* receba o menor número, armazenado na variável (**valor2**) e consequentemente o *Edit4* receberá o maior número, armazenado em (**valor1**). Se essa condição for falsa, então o componente *Edit3* guardará o menor valor que está armazenado na variável (**valor1**) e *Edit4* guardará o maior valor que está armazenado na variável (**valor2**).

Para inserir o código do *button2*, clique no *button2* e no Inspetor de Objetos, clique na Aba dos Eventos, localize o evento *OnClick*, dê um duplo clique para abrir o procedimento e insira o código abaixo de acordo com a Figura 4.5.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
end;
```

Figura 4.5: *OnClick* do *Button2* da *prj_Aplicacao_06*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Essa ação fará com que todos os componentes *Edits* fiquem vazias, ou seja, sem informações para futuras inserções de novos valores.

Salve tudo e execute o projeto teclando **F9.**

Vamos praticar um pouco do que acabamos de estudar? Tenho certeza de que você é capaz! Crie um novo programa e exiba como resultado a ordenação decrescente.

4.1.3 Estrutura condicional aninhada

Construiremos nosso projeto de acordo com a Figura 4.6.

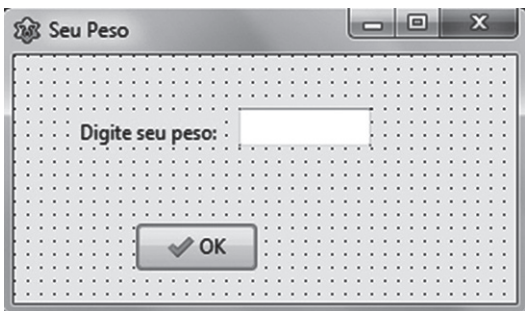


Figura 4.6: Formulário do prj_Aplicacao_07
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

a) Selecione o *Form1* e altere a propriedade *caption* para **Seu Peso.**

b) Ainda com o *Form1* selecionado, mude a propriedade *WindowState* para *wsMaximized*.

c) Insira, por meio da aba *standard*, um componente *label* e altere o *caption* para **Digite seu Peso.**

d) Insira um *Edit* da Paleta *standard* e Limpe a propriedade *Text*, deixando-a vazia.

e) Com Aba *Additional*, insira o componente *BitBtn* e altere a propriedade *Kind* para *bkOk*.

f) Salve o projeto. Para a *Unit1*, salve com o nome de *u_Aplicacao_07*, e para o projeto, com o nome *prj_Aplicacao_07*;

Nesse projeto o usuário fornecerá um peso, o qual é do tipo real; conheceremos outra função de conversão.

g) Agora, de acordo com a Figura 4.7, insira o código através do *OnClick* do componente *Bitbtn*.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
   peso: real;
begin
     peso:= StrToFloat(Edit1.Text);
     if (peso <20) then
        ShowMessage('Você tem ' + FloatToStr(peso))
     else
       if (peso >=20) and (peso <60) then
        ShowMessage('Tem ' +FloatToStr(peso)+ ' entre 20 e 60')
       else
        ShowMessage('Tem ' +FloatToStr(peso)+ ' acima de 60')

end;
```

Figura 4.7: *OnClick* do *BitBtn*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observando o código, Figura 4.7, foi criada a variável **peso** para ser tratada na estrutura condicional aninhada e a inserção da função *StrToFloat* que converte a informação do tipo *string* armazenada no *Edit1* para real; se variável **peso** for menor que 20, como resultado será emitido: "você *tem 20*"; caso contrário, entrará em outra condição. Se o **peso** for maior ou igual a 20 e menor que 60, então outra mensagem será emitida, que essa pessoa está na faixa de peso entre 20 e 60. Porém se essa pessoa tiver com peso acima de 60, será emitida a mensagem afirmando que ela está acima de 60. Para a exibição do resultado, foi usada a função *ShowMessage* e a conversão de real para String através da função *FloatToStr*.

h) Vamos tratar agora o componente *Edit* para que o usuário digite apenas os dígitos de 0(zero) a 9 (nove), a divisão (ponto) da parte inteira e fracionária e o *Backspace*. Para isso, selecione o componente *Edit*, localize o evento *OnKeyPress* no Inspetor de Objetos e dê um duplo clique para abrir o procedimento e colocar o código abaixo, de acordo com a Figura 4.8.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: char);
begin
   if not (Key in ['0'..'9']) and not (Key in [#8]) and
   not (Key in [#46])  then
    Key:=#0;

end;
```

Figura 4.8: *OnkeyPress* do *Edit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

De acordo com o código da Figura 4.8, usamos os operadores lógicos *And* e *Not*. Se a condição da *if* for verdadeira, o *Edit1* recebe *(#0)*, ou seja, não vai inserir nada (vazio), permitindo apenas que no *Edit* sejam digitados: dígitos de 0 a 9, o (ponto) e deletar as informações através da tecla (*Backspace*).

Salve tudo e execute o projeto teclando **F9**.

Estrutura condicional com *Radiobutton*

Veja agora um exemplo de estrutura condicional usando o componente *RadioButton*.

a) Para isso montaremos nosso projeto de acordo com a Figura 4.9.

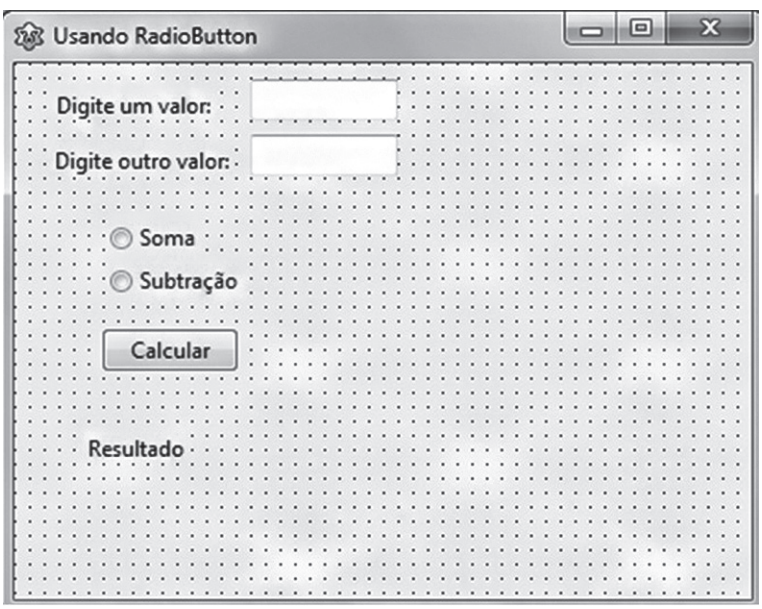


Figura 4.9: Formulário do *prj_Aplicacao_08*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Mude a propriedade *caption* do *Form1* para Usando ***RadioButton***.

c) Insira três *labels* e altere a propriedade *caption*:

- *label1* para **Digite um valor:**
- *label2* para **Digite outro valor:**
- *label3* para**Resultado:**

d) Insira dois *Edits* e limpe a propriedade *Text*.

e) Na aba *Standard* insira dois componentes *RadioButtons* e altere a propriedade *caption*:

- *RadioButton1* para **Soma.**
- *RadioButton2* para **Subtração.**

f) Insira um *button* e altere seu *caption* para **Calcular**.

g) Agora salve *Unit1* com o nome *u_Aplicacao_08* e o projeto com *prj_Aplicacao_08*.

h) Implementaremos agora o código do botão **Calcular**, de acordo com a Figura 4.10.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor1, valor2, resultado:real;
begin
     RadioButton1.Checked:=true;
     valor1:=StrToFloat(Edit1.Text);
     valor2:=StrToFloat(Edit2.Text);
     if RadioButton1.Checked then
        resultado:= valor1 + valor2;
     if RadioButton2.Checked then
        resultado:= valor1 - valor2;

     Label3.Caption:=FloatToStr(resultado);


end;
```

Figura 4.10: *OnClick* do Botão Calcular
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Quando o *Form* possui diversos componentes *RadioButton* inseridos, é necessário que o usuário selecione uma opção, apenas uma opção, através da propriedade *Checked*.

Nesse projeto o usuário fornecerá dois valores para que seja selecionada a Operação Aritmética (*Soma* ou *Subtração*) através da propriedade *Checked* do componente *RadioButton*. De todos os *RadioButtons* inseridos no *Form*, apenas um pode estar selecionado, quando eles forem checados através da estrutura condicional – *if*. Se *RadioButton* da Soma estiver selecionado, efetuará a operação (soma) e será atribuído o somatório na variável **resultado**. Se *RadioButton* da Subtração estiver selecionado, o resultado da operação (subtração) será atribuído na variável **resultado**. O resultado será exibido no *label3*. Ressalvo que o *label* não possui a propriedade *Text*. O resultado será colocado na propriedade *caption*, conforme código inserido na Figura acima.

Salve tudo e execute o projeto teclando **F9.**

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa e exiba como resultado, por meio do *RadioButton*, o quadrado ou o cubo de um determinado número.

4.1.4 Estrutura *case*

Neste projeto utilizaremos a estrutura *case*.

a) Para isso montaremos o formulário de acordo com a Figura 4.11.

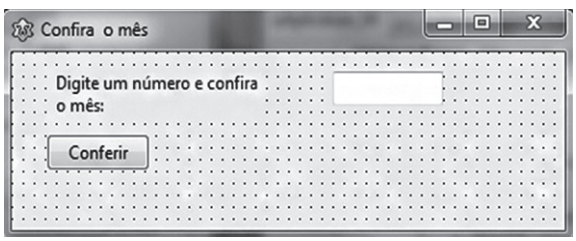


Figura 4.11: Formulário do prj_Aplicacao_09
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Selecione o *Form1* alterando o *caption* para **Confira o mês.**

c) Insira um componente *label* e clique nos três pontos da propriedade *caption*, conforme Figura 4.12.

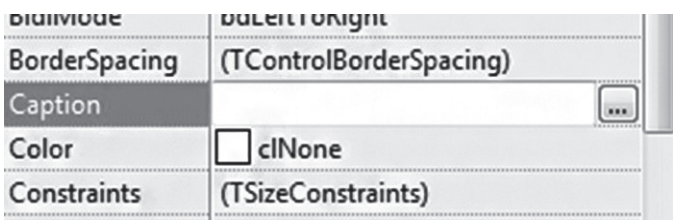


Figura 4.12: caption do *label*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

d) Ao clicar nos três pontos da propriedade *caption*, surgirá a janela de Editor de Caracteres; digite o texto conforme Figura 4.13.

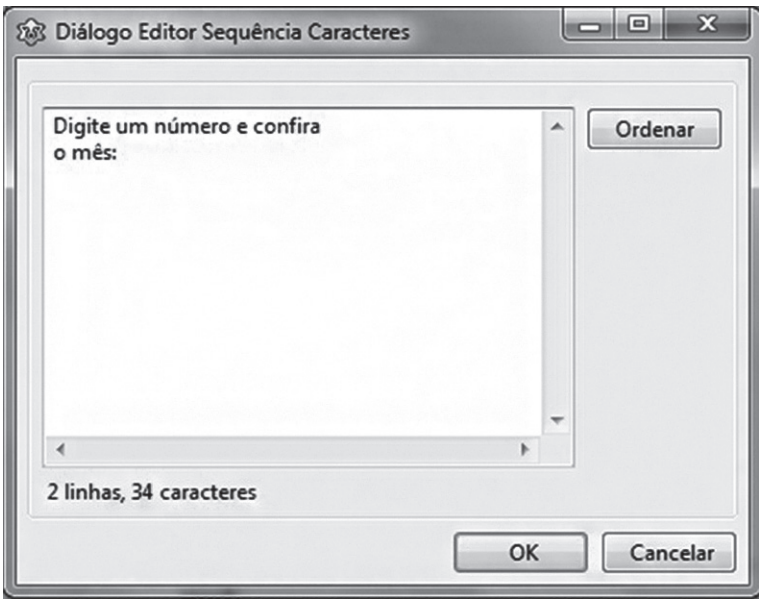


Figura 4.13: Editor de caracteres do *caption*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

e) Insira agora o componente *Edit* e limpe sua propriedade *Text*.

f) Insira o componente *button* alterando sua propriedade *caption* para **Conferir**.

g) Agora salve *Unit1* com *u_Aplicacao_09* e o projeto com *prj_Aplicacao_09*.

h) Por último, implementaremos o código do botão Conferir, através do evento *OnClick* conforme Figura 4.14.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  mes: integer;
begin
     mes:= StrToInt(Edit1.Text);
     case mes of
     1: Label2.Caption:='JANEIRO';
     2: Label2.Caption:='FEVEREIRO';
     3: Label2.Caption:='MARÇO';
     4: Label2.Caption:='ABRIL';
     5: Label2.Caption:='MAIO';
     6: Label2.Caption:='JUNHO';
     7: Label2.Caption:='JULHO';
     8: Label2.Caption:='AGOSTO';
     9: Label2.Caption:='SETEMBRO';
     10: Label2.Caption:='OUTUBRO';
     11: Label2.Caption:='NOVEMBRO';
     12: Label2.Caption:='DEZEMBRO';
     ELSE
        Label2.Caption:='Valor Inválido!';
     end;
end;
```

Figura 4.14: *OnClick* do botão Conferir
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

O código da Figura 4.14 aplica a estrutura *case*. Criamos a variável **mes** do tipo inteiro para ser aplicado nas opções do case; essa variável será lida por meio da entrada de dados realizado pelo usuário através do componente *Edit1*. Uma das opções será comparada à lista de opções de meses do ano; caso seja digitada uma opção não existente, cairá na instrução *else* da estrutura *case*, emitindo uma mensagem de valor inválido.

Salve tudo e execute o projeto teclando **F9.**

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa e exiba como resultado o dia da semana através da Estrutura *Case*, em que o usuário fornecerá um numero de 1 a 7.

4.2 Estrutura de repetição

4.2.1 Estrutura *while*

Neste próximo projeto aplicamos a estrutura de repetição (*while*).

a) Para isso, construa o formulário de acordo com a Figura 4.15.

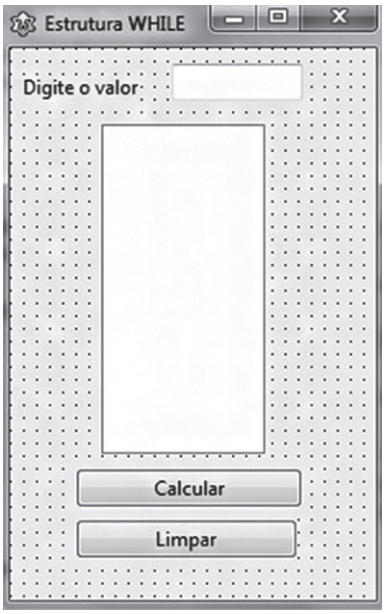


Figura 4.15: Formulário do *prj_Aplicacao_10*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Estrutura *while.***

c) Insira um componente *label* e altere a propriedade *caption* para **Digite o valor.**

d) Agora insira o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

e) Insira o componente *ListBox* localizado na Aba *standard* e altere as propriedades *Height* e *Width* para 200 e 100 respectivamente.

f) Agora coloque dois *buttons* e altere a propriedade *caption*:

- *button1* para *caption* para **Calcular.**
- *Button2* para caption para **Limpar**.

g) Agora salve *Unit1* com *u_Aplicacao_10* e o projeto com *prj_Aplicacao_10*.

O respectivo projeto tem por finalidade mostrar no *ListBox* o resultado dos dez primeiros múltiplos do número fornecido pelo usuário no componente *Edit1*, conforme Figura 4.16.

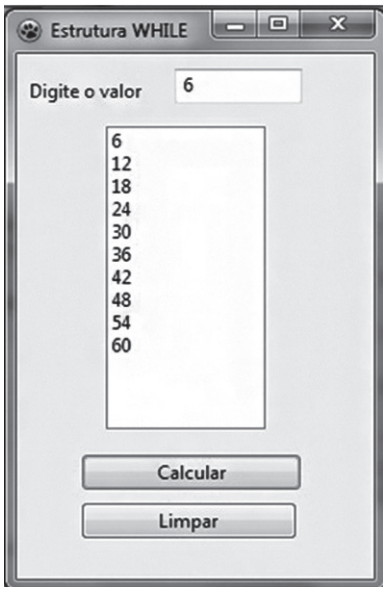


Figura 4.16: O *prj_Aplicacao_10* em execução
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

h) Agora implementaremos o código do evento *OnClick* do botão **Calcular**, conforme a Figura 4.17.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor, contador, Resultado: Integer;
begin
     ListBox1.Items.Clear;
     valor:= StrToInt(Edit1.Text);
     contador:=1;
     while (contador<=10) do
     begin
          resultado:=valor* contador;
          ListBox1.Items.Add(InttoStr(resultado));
          contador:=contador +1;
     end;
end;
```

Figura 4.17: *OnClick* do botão Calcular
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Nesse procedimento, foi a variável **valor** para receber o número do usuário; foi criada também a variável **contador**, que servirá para incrementar os valores de um a dez dentro da nossa estrutura de repetição *while*; observe que na Estrutura *while*, a condição dar-se-á no início da estrutura; por isso a importância de inicializar a variável **contador** antes da estrutura de repetição.

O código da Figura 4.17 utilizou o comando *ListBox1.Items.Clear* para limpar os itens presente na lista, como resultado após a execução, e utilizou também o comando *ListBox1.Items.Add* para realizar a inserção dos valores.

Para esclarecer melhor o código, percebemos no início do procedimento o comando *ListBox1.Items.Clear* usado para limpar todos itens ou linhas da nossa lista antes de serem inseridos novos valores. Para inserção dos valores no componente *ListBox* foi usado o comando *ListBox1.Items.Add* convertendo os valores armazenados na variável **resultado** do tipo de dado inteiro para *String*. Para isso foi usada a função de conversão (*IntToStr*). Por último a linha de comando de incremento da variável **contador**, até que a condição se torne falsa, ou seja, até que a variável seja maior que 10.

i) Agora implementaremos o evento *OnClick* no botão **Limpar**; para isso, dê duplo clique no respectivo botão e insira o código, conforme Figura 4.18.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  ListBox1.Items.Clear;
end;
```

Figura 4.18: *OnClick* do botão Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste código, Figura 4.18, observamos que o *Edit1* ficará limpo, pois foi solicitado que a sua propriedade *Text* fique vazia. E o comando *Listbox1.Items.Clear*, como já conhecido, limpará os valores antes armazenados através da última execução.

j) Salve tudo e execute o projeto teclando **F9**.

4.2.2 Estrutura de repetição *repeat*

Neste projeto iremos aprimorar a exibição dos resultados no *ListBox* do *prj_Aplicacao_10* e trocaremos a estrutura de repetição *while* pela estrutura de repetição *repeat*.

a) Para isso, montaremos o formulário de acordo com a Figura 4.19.

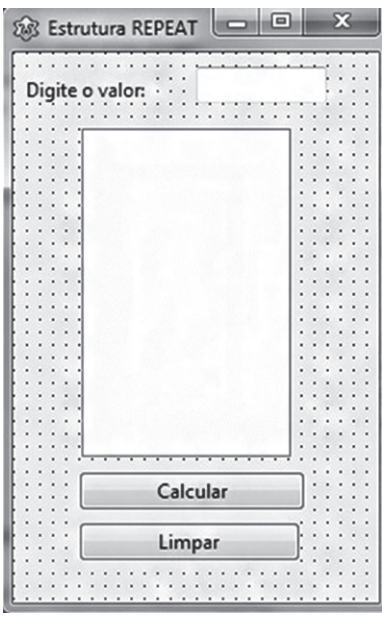


Figura 4.19: Formulário do *prj_Aplicacao_11*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Estrutura *repeat*.**

c) Insira um componente *label* e altere a propriedade *caption* para **Digite o valor.**

d) Agora insira o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

e) Insira o componente *ListBox* localizado na Aba *Standard* e altere as propriedades *Height* e *Width* para 250 e 100, respectivamente.

f) Ainda com *ListBox1* selecionado, acesse a propriedade *Font*, altere conforme Figura 4.20 e clique em OK.

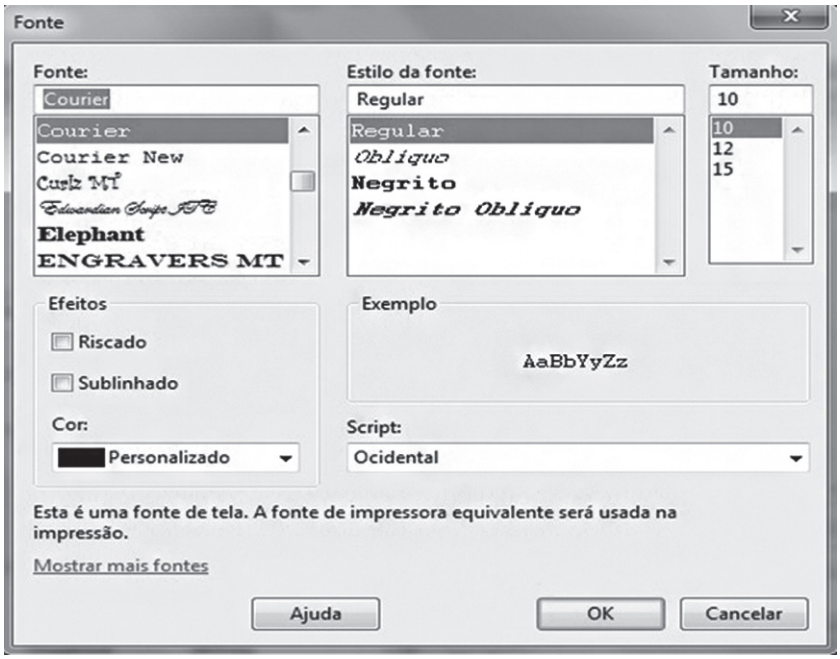


Figura 4.20: Janela da propriedade *Font*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

g) Agora colocaremos dois *buttons* e alteraremos a propriedade *caption*.

- *button1* para *caption* para **Calcular.**
- *button2* para *caption* para **Limpar.**

h) Agora salve *Unit1* com o nome *u_Aplicacao_11* e o nome do projeto com *prj_Aplicacao_11*.

O respectivo projeto tem por finalidade mostrar no *ListBox* a tabuada dos dez primeiros múltiplos do número fornecido pelo usuário no *Edit1*, conforme Figura 4.21.

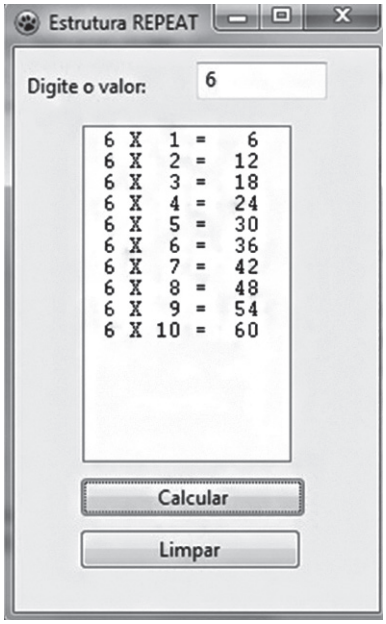


Figura 4.21: O *prj_Aplicacao_11* em execução
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

i) Agora partiremos para a inserção do código do evento *OnClick* do botão **Calcular**, conforme a Figura 4.22.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor, contador, resultado: Integer;
   V_Str, C_Str, r_STr: string;
begin
     ListBox1.Items.Clear;
     valor:= StrToInt(Edit1.Text);
     contador:=1;
     repeat
          resultado:=valor* contador;
          Str(valor:2, V_Str);
          Str(contador:2, C_Str);
          Str(Resultado:3, r_Str);
          ListBox1.Items.Add(V_Str+ ' X ' +
              C_Str+ ' = ' + r_Str);
          contador:=contador +1;
     until (contador>10);
end;
```

Figura 4.22: *OnClick* do botão Calcular
Fonte: *Print screen* do Lazarus IDE v0.9.28.

A diferença do *prj_Aplicacao_11* para *prj_Aplicacao_10* é que agora para inserir os valores no *ListBox* precisaremos converter os valores de inteiro para *String* antes do comando *ListBox1.Items.Add* com a função Str, no caso, *Str*(**Contador***:2, c_Str*), a função pega o valor do tipo inteiro armazenado na variável **contador** com formatação de organização dos dígitos e armazena o valor convertido para o tipo *string* na variável **c_Str.**

Em relação à estrutura de repetição (*repeat*), observamos agora que a condição é colocada no final de sua estrutura, ou seja, *until* (**contador** > *10*). Enquanto a condição for falsa, ela continuará no *loop*, até que se torne verdadeira, finalizando a estrutura de repetição.

j) Agora implementaremos o evento *OnClick* no botão **Limpar**; para isso, dê duplo clique no respectivo botão e insira o código, conforme Figura 4.23.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  ListBox1.Items.Clear;
end;
```

Figura 4.23: *OnClick* do botão Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

k) Salve tudo e execute o projeto teclando **F9**.

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa de acordo com o projeto anterior e apenas troque por outra a operação aritmética básica.

4.2.3 Estrutura de repetição (*for*)

Neste projeto trocaremos a Estrutura repeat do *prj_Aplicacao_11* pela estrutura de repetição *for*.

a) Para isso, montaremos o formulário de acordo com a Figura 4.24.

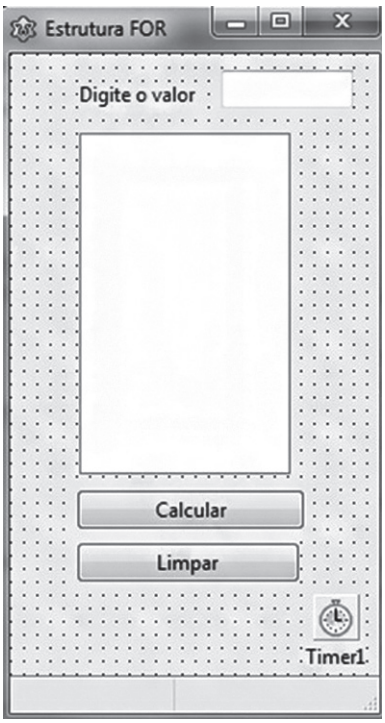


Figura 4.24: Formulário do *prj_Aplicacao_12*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para E**strutura de repetição *for*.**

c) Insira um componente *label* e altere a propriedade *caption* para **Digite o valor.**

d) Agora insira o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

e) Insira o componente *ListBox* localizado na Aba *Standard* e altere as propriedades *Height* e *Width* para 250 e 100, respectivamente;

f) Ainda com *ListBox1* selecionado, acesse a propriedade *Font*, altere conforme Figura 4.25 e clique em OK.

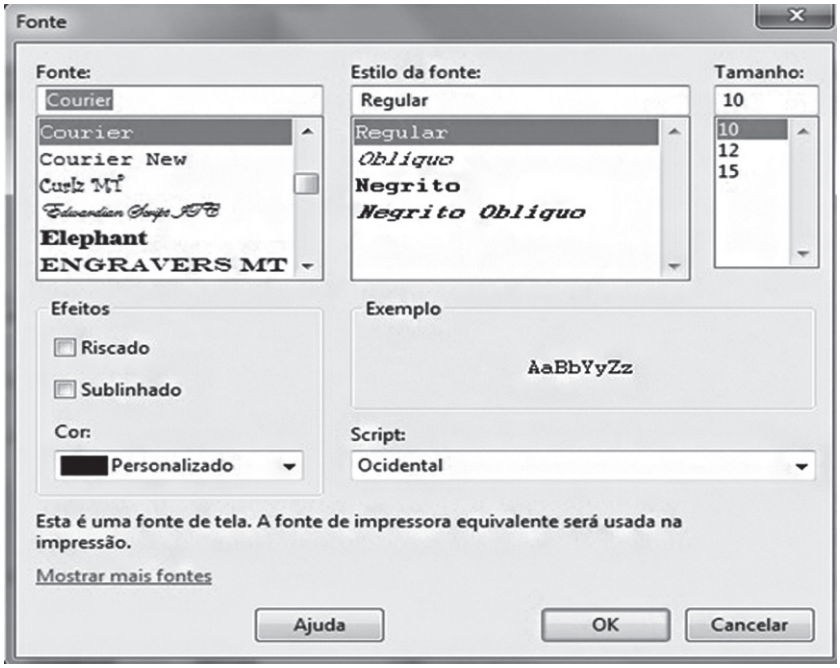


Figura 4.25: Janela da propriedade *Font*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

g) Agora colocaremos dois *buttons* e alteraremos a propriedade *caption*:

- *button1* para *caption* para **Calcular.**
- *button2* para *caption* para **Limpar**.

h) Insira o componente *Timer* da aba *System*.

i) Ao executar o projeto, você perceberá que o componente *Timer* é um componente não visual, pois ele não aparece no formulário em tempo de execução (*Runtime*).

j) Insira também o componente *StatusBar* da aba Common *Controls*.

k) Agora salve Unit1 com o nome *u_Aplicacao_12* e o projeto com *prj_Aplicacao_12*.

l) Vamos inserir o código do botão **calcular**, conforme Figura 4.26.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   valor, contador, resultado: Integer;
   v_Str, C_Str, r_STr: string;
begin
     ListBox1.Items.Clear;
     valor:= StrToInt(Edit1.Text);
     for contador:=1 to 10 do
     begin
       resultado:=valor* contador;
       Str(valor:2, v_Str);
       Str(contador:2, c_Str);
       Str(Resultado:3, r_Str);
       ListBox1.Items.Add(v_Str+ ' X '
                          + c_Str+ ' = ' + r_Str);
     end;
end;
```

Figura 4.26: *OnClick* do Button1
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observamos que diferença do código do *prj_Aplicacao_12* para *prj_Aplicacao_11* é a substituição do *repeat* para a estrutura de repetição *for*, uma estrutura que na sua sintaxe exige que a variável seja inicializada e seja incrementada ou decrementada automaticamente até um valor final.

m) Agora implementaremos o evento *OnClick* no botão **Limpar**; para isso, dê duplo clique no respectivo botão e insira o código, conforme Figura 4.27.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:='';
  ListBox1.Items.Clear;
end;
```

Figura 4.27: *OnClick* do botão Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

n) Da mesma forma que nos projetos anteriores, essas ações servirão para limpar os campos para inserções de novos valores.

o) Agora vamos configurar nosso *StatusBar* para que sejam inseridas a data e hora do sistema operacional. Para isso, dê um duplo clique no componente *StatusBar1*, quando surgirá a janela, conforme a Figura 4.28.

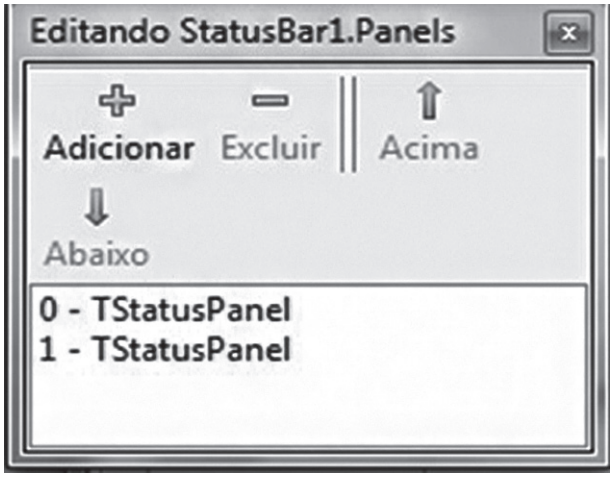


Figura 4.28: Editor do *StatusBar*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

p) Selecione a opção (*0- TStatusPanel*) e no Inspetor de Objetos altere a propriedade *Width* para valor 100, conforme Figura 4.29.

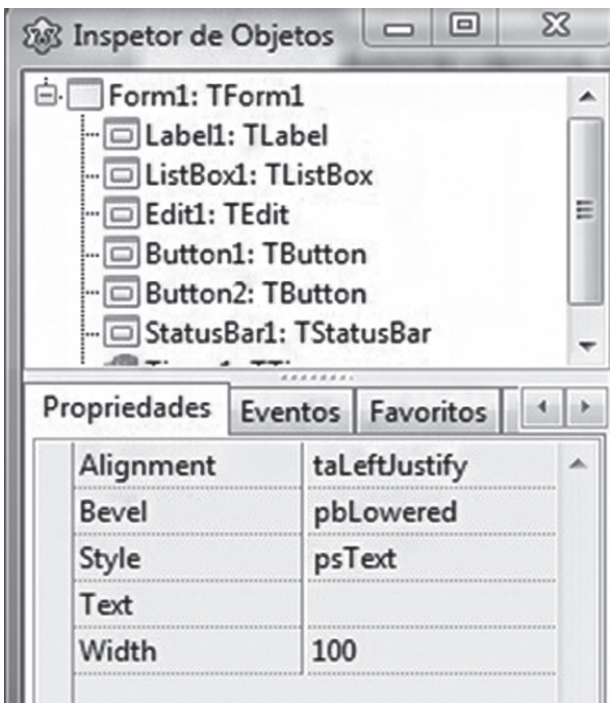


Figura 4.29: Propriedade *Width* do (*0-TStatusPanel*)
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

q) Faça a mesma coisa com a opção (*1- TStatusPanel*).

r) Agora criaremos o código para exibir a data e a hora no *StatusBar1*; para isso, dê um duplo clique no componente *Timer* para gerar o evento *On-Timer* e insira o código, conforme Figura 4.30.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  StatusBar1.Panels[0].Text:=DateToStr(Date);
  StatusBar1.Panels[1].Text:=TimeToStr(Time);
end;
```

Figura 4.30: Evento *OnTimer*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observe que o *StatusBar* foi dividido em dois painéis, onde o índice começa por 0(zero). Portanto *StatusBar1.Panels[0].Text* recebe a data convertida para *string* através da função *DateToStr*, e *StatusBar1.Panels[1].Text* recebe a hora através da função *TimeToStr*.

s) Salve tudo e execute o projeto teclando **F9**.

Resumo

Nesta aula implementamos as estruturas condicionais: simples, composta, aninhada e a estrutura *case*. Implementamos também as estrutura de repetição: *while*, *repeat* e *for*. Conhecemos alguns eventos: *OnKeyPress*, *OnTimer*. Conhecemos também novos componentes: *BitBtn, RadioButton, ListBox, StatusBar, Timer*, bem como algumas propriedades pertencentes aos componentes supracitados.

Atividades de aprendizagem

Todas as questões devem ser desenvolvidas no IDE Lazarus utilizando o *Object* Pascal.

1. O sistema deve receber um número inteiro do usuário e deverá emitir o resultado se esse número é par ou ímpar.

2. O sistema deve receber três valores inteiros e emitir para o usuário o maior valor.

3. O sistema receberá a entrada de valor para a altura e o sexo de uma pessoa e através dessas duas informações fornecerá para o usuário o seu peso ideal.

   Fórmulas:

   - Para homens: (72.7 * H) - 58
   - Para mulheres: (62.1 * H) - 44.7
   - (H = altura)

4. Fornecer ao Sistema dois números inteiros, x e y, e emitir como resultado o quociente e o resto da divisão inteira entre eles.

5. Faça a tabuada da soma de um número inteiro fornecido pelo usuário. Observação: o resultado deverá ser exibido em componente *ListBox* e aplicado à estrutura de repetição – *repeat*.

6. O sistema receberá dois valores inteiros do usuário e, através da estrutura de repetição *while*, mostrará como resultados os valores dentro da faixa de valores fornecidos. Observação: o resultado deverá ser exibidos em um *ListBox*.

Poste suas respostas no AVEA.

Aula 5 – Procedimentos, funções, *array* e *record*

Objetivos

Implementar a estrutura de procedimento com e sem parâmetros.

Implementar a estrutura de funções.

Implementar tipo de dado homogêneo *array*.

Implementar tipo de dado heterogêneo *record*.

5.1 Procedimento (*procedure*)

O *Integrated Development Environment* (IDE) Lazarus disponibiliza dois tipos de aplicações com procedimentos: **sem parâmetros** e **com parâmetros.**

Sintaxe:

```
procedure <Nome> [(parâmetros)]
        <definições>
begin
        <comandos>;
end;
```

Uma *procedure* é um tipo de sub-rotina que é ativada pela colocação de seu Nome em alguma parte do programa. Dessa forma, assim que o Nome de uma *procedure* é encontrado, ocorre um desvio no programa, para que os comandos dessa sub-rotina sejam executados. Ao término da sub-rotina, a execução retornará ao ponto subsequente à chamada da *procedure*.

Na estrutura os [***parâmetro***s] são *comandos opcionais*.

5.1.1 Procedimento sem parâmetros

Nesta aplicação utilizamos um exemplo de solução com o uso de procedimento sem parâmetros.

a) Para isso, montaremos o formulário de acordo com a Figura 5.1.

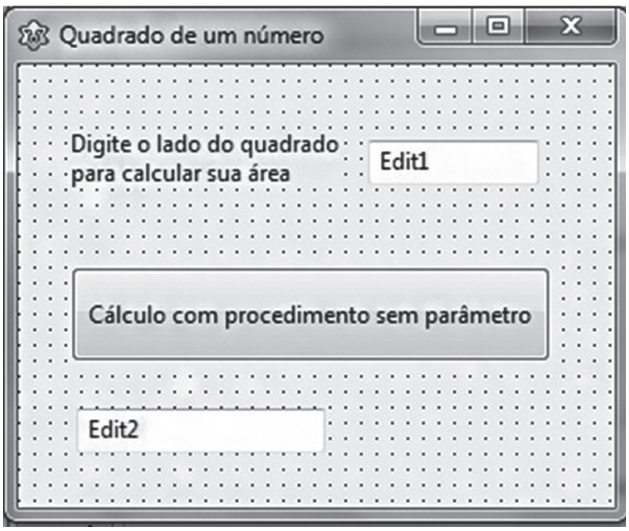


Figura 5.1: Modelo do Formulário do *prj_Aplicacao_13*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Quadrado de um número.**

c) Insira um componente *label* e altere a propriedade *caption* conforme Figura 5.2.

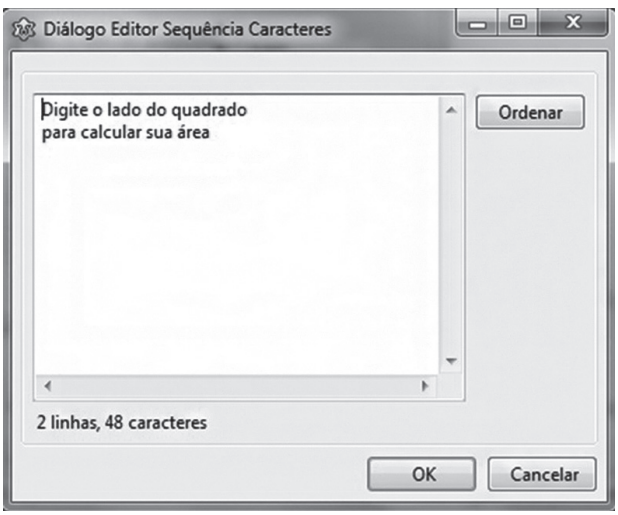


Figura 5.2: Janela do Editor de Caracteres do *Label1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

d) Agora, após a inserção do componente *Edit*, altere a propriedade *Text*, deixando-a vazia.

e) Após a inserção do componente *Edit*, posicione-o conforme a Figura 5.2 e altere a propriedade *Text*, deixando-a vazia, e altere a propriedade *Enabled* para *False*.

f) Colocado o componente *button*, altere a propriedade *caption* para **Cálculo com procedimento sem parâmetros.**

g) Agora salve *Unit1* com o nome *u_Aplicacao_13* e o projeto com *prj_Aplicacao_13*.

h) Vamos inserir o código do evento *OnClick* do *button1*, conforme Figura 5.3.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     ProcedimentoSemParametro;
end;
```

Figura 5.3: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

No evento *OnClick* fica claro que o único comando presente é a chamada do procedimento, o qual é chamado de **ProcedimentoSemParametro.**

i) Agora vamos inserir o código do procedimento **ProcedimentoSemParametro**, conforme Figura 5.4.

```
procedure ProcedimentoSemParametro;
var
   area, lado: real;

begin
     lado:=StrToFloat(Form1.Edit1.Text);
     area:=lado * lado;
     Form1.Edit2.Text:=(FloatToStr(Area));

end;
```

Figura 5.4: *Procedure* ProcedimentoSemParametro
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Digite o código do procedimento da Figura 5.4 nas linhas acima do evento *OnClick* da Figura 5.3. Lembre-se que se formulário estiver acima da *Unit*, utilize a tecla de alternância entre *Form/Unit*, ou seja, use a tecla F12.

O referido procedimento é chamado no evento *OnClick* do *Button*, Figura 5.3, e executa as linhas de comando, conforme Figura 5.4. Na *procedure* foram declaradas duas variáveis: **area e lado** do tipo real, pois o aplicativo

retorna o valor da área do uma determinada figura geométrica. As funções de conversão aplicadas foram *StrToFloat* e *FloatToStr*.

Importante ressaltar a presença para *Form1* antes do componente Edit1.Text, e Edit2.Text, pois a procedure **ProcedimentoSemParametro** está fora da rotina *TForm1.Button1.Click*.

j) Ao término da sub-rotina, a execução retornará ao ponto subsequente à chamada da *procedure*. Salve tudo e execute o projeto teclando F9.

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa e exiba como resultado o produto de dois números inteiros através do procedimento sem parâmetro.

5.1.2 Procedimento com parâmetros

Já sabemos que para ativar uma sub-rotina colocamos seu Nome em alguma parte do programa para que ela possa ser executada.

Nos procedimentos com parâmetros há duas subdivisões: **passagem de parâmetro por valores e passagem de parâmetro por referência**.

a) Passagem de parâmetro por valores

Sintaxe:

```
procedure <Nome> (<variável> : <tipo >);
    <Definições>;
begin
    <comandos>;
end;
```

Utilizaremos o próximo projeto para exemplificar o uso de procedimento por **passagem de valores.** Para isso, montaremos o formulário de acordo com a Figura 5.5.

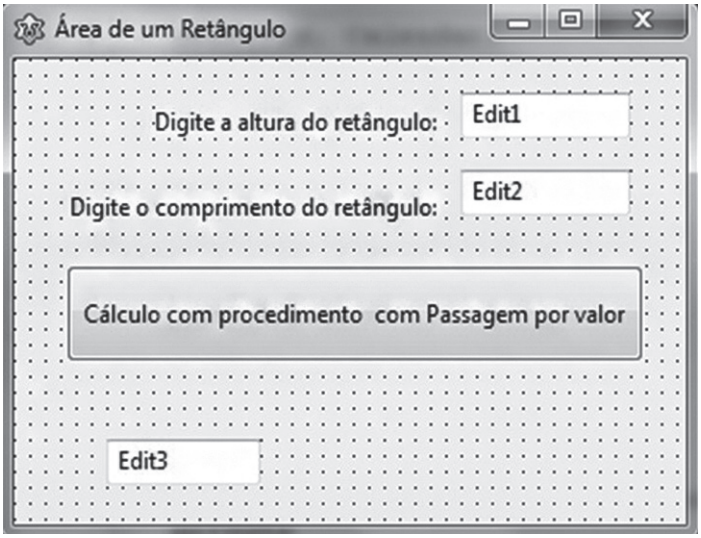


Figura 5.5: Modelo do Formulário *prj_Aplicacao_14*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

- Altere o *caption* do *Form1* para **Área de um Retângulo.**

- Selecione o componente *label* e altere a propriedade *caption* para **Digite a altura do retângulo**.

- Agora faça a mesma coisa com o segundo componente *label* e acesse e altere a propriedade *caption* para **Digite o comprimento do retângulo.**

- Selecione o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

- Faça a mesma coisa com o segundo componente *Edit* e posicione-o conforme a Figura 5.5 e altere a propriedade *Text*, deixando-a vazia.

- Após a inserção do terceiro componente *Edit*, posicione-o conforme a Figura 5.5 e altere a propriedade *Text*, deixando-a vazia e altere sua propriedade *Enabled* para *False*.

- Selecione o componente *button* e altere a propriedade *caption* para **Cálculo com procedimento com Passagem por valor.;**

- Agora salve *Unit1* com o nome *u_Aplicacao_14* e o projeto com prj_Aplicacao_14.

- Vamos inserir o código do evento *OnClick* do *button1*, conforme Figura 5.6.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    altura, comprimento: real;
begin
     altura:=StrToFloat(Edit1.Text);
     comprimento:=StrToFloat(Edit2.Text);
     ProcParamValor(altura, comprimento);
end;
```

Figura 5.6: OnClick do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste evento foram declaradas as variáveis: **altura e comprimento**. As respectivas variáveis do tipo real foram lidas através dos *Edits* e seus respectivos valores foram encaminhados através da chamada do procedimento o qual chamamos de **ProcParamValor(altura, comprimento)** , conforme Figura 5.7.

- Agora digite o código do procedure **ProcParamValor**, conforme Figura 5.7 nas linhas acima da *TForm1.Button1Click* da Figura 5.6.

```
procedure ProcParamValor(Alt, comp: real);
var
   area: real;
begin
     area:=Alt *comp;
     Form1.Edit3.Text:=FloatToStr(Area);
end;
```

Figura 5.7: *Procedure* ProcParamValor
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Com o chamado do respectivo procedimento, os valores das variáveis **altura e comprimento** foram armazenados nas variáveis **Alt e comp** declaradas como variáveis nos Parâmetros do procedimento, conforme Figura 5.7.

Observamos que no procedimento **ProcParamValor**, declaramos a variável local **area**, a qual recebe o produto de **alt e comp** e, para exibirmos o resultado armazenado em **área**, é preciso redirecionar a informação para o componente *Edit3*, presente no *Form1*. Para isso foi utilizada a referência por meio do comando *Form1.Edit3.Text*.

- Salve tudo e execute o projeto teclando **F9**.

b) Passagem de parâmetro por referência

Neste próximo projeto exemplificaremos o uso de procedimento com passagem por referência.

- Para isso, montaremos o formulário de acordo com a Figura 5.8.

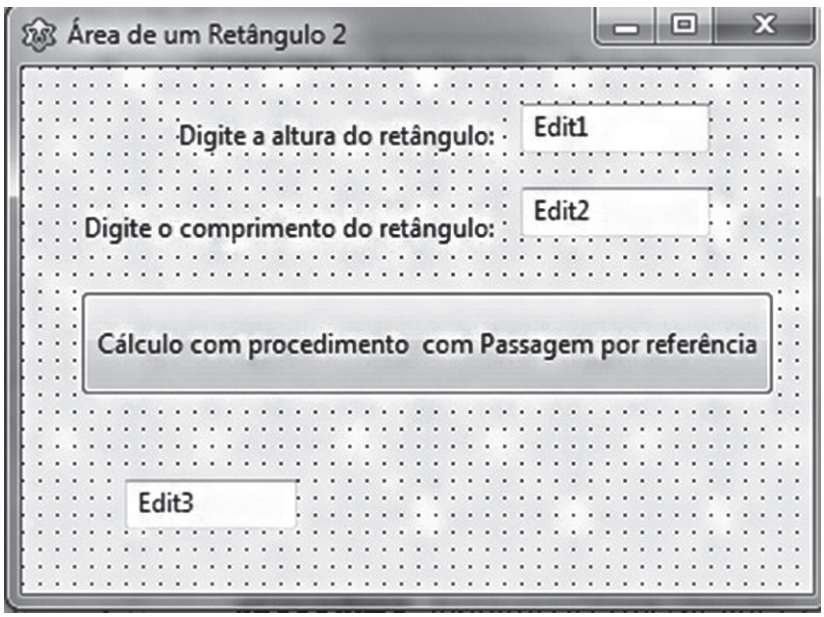


Figura 5.8: Modelo do Formulário do *prj_Aplicacao_15*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

- Altere o *caption* do *Form1* para **Área de um Retângulo 2.**

- Selecione o primeiro componente *label* e altere a propriedade *caption* para **Digite a altura do retângulo.**

- Agora faça o mesmo procedimento com o segundo componente *label*, alterando a propriedade *caption* para **Digite o comprimento do retângulo**.

- Acesse o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

- Agora faça a mesma coisa com o segundo componente *Edit* e posicione-o conforme a Figura 5.8 e altere a propriedade *Text*, deixando-a vazia.

- Selecione o terceiro componente *Edit* e posicione-o conforme a Figura 5.8 e altere a propriedade *Text*, deixando-a vazia e altere sua propriedade *Enabled* para *False*.

- Através do componente *button*, altere a propriedade *caption* para **Cálculo com procedimento com Passagem por referência.**

- Salve *Unit1* com o nome *u_Aplicacao_15* e o projeto com *prj_Aplicacao_15*.

- Insira o código do evento *OnClick* do *button1*, conforme Figura 5.9.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    area, altura, comprimento: real;
begin
     altura:=StrToFloat(Edit1.Text);
     comprimento:=StrToFloat(Edit2.Text);
     ProcParamReferencia(altura, comprimento, area);
     Edit3.Text:=FloatToStr(Area);
end;
```

Figura 5.9: *OnClick* do Button1
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste evento percebemos mais uma vez a declaração das variáveis: **altura, comprimento e area.** Também fica claro a leitura dos valores e o armazenamento nas variáveis **altura e comprimento.**

Na linha de comando **ProcParamReferencia (altura, comprimento, area)** ocorre a chamada do referido procedimento, e a passagem dos valores das duas primeiras variáveis (**altura e comprimento**) para procedimento **ProcParamReferencia.** A terceira variável (**area**) servirá para receber o retorno, ou seja, o resultado da execução do procedimento **ProcParamReferencia** (passagem por referência).

- Insira agora o código do procedure **ProcParamReferencia** da Figura 5.10 nas linhas acima do código da Figura 5.9.

```
procedure ProcParamReferencia(Alt, comp: real; var resultado:real);
begin
     resultado:=Alt *comp;
end;
```

Figura 5.10: *Procedure* ProcParamReferencia
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Ao declarar o procedimento **ProcParamReferencia**, conforme Figura 5.10, percebemos a declaração dos parâmetros **alt e comp** para receber respectivamente os valores de **altura e comprimento**, e na declaração dos parâ-

metros colocamos a cláusula *var* para declararmos o parâmetro **resultado**, variável que guardará o resultado da solução do projeto.

Quando a variável **resultado** tiver o valor final, ela encaminhará o respectivo valor para a variável **area** declarada na linha de comando: **ProcParamReferencia(altura, comprimento, area)** , comando este inserido na *TForm1.Button1Click*.

Com o valor armazenado na variável **area**, podemos exibir o resultado final para o usuário através do componente *Edit3*, conforme Figura 5.11.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    area, altura, comprimento: real;
begin
     altura:=StrToFloat(Edit1.Text);
     comprimento:=StrToFloat(Edit2.Text);
     ProcParamReferencia(altura, comprimento, area);
     Edit3.Text:=FloatToStr(Area);
end;
```

Figura 5.11: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

- Salve tudo e execute o projeto teclando **F9**.

5.2 Funções (*function*)

Essa sub-rotina possui as mesmas características de uma *procedure* no que se refere à passagem de parâmetros, mas possui uma diferença, que é o retorno de um valor ao fim de sua execução, ou seja, uma *function* sempre terá que retornar um valor armazenado no nome da própria função, como será visto no próximo projeto.

Na definição de uma *function*, devemos informar o tipo do valor retornado, que pode ser diferente dos tipos de dados dos valores encaminhados através da passagem por valores.

Sintaxe:

```
function <Nome> [(Parâmetros)]: < Tipo do valor retornado>;
        <Definições>;
begin
        <Comandos>;
end;
```

Observando a sintaxe acima, a declaração do tipo de retorno fica ao fim da primeira linha, após a declaração dos parâmetros.

a) Para fazer uso da *function*, criaremos o projeto e montaremos o formulário de acordo com a Figura 5.12.

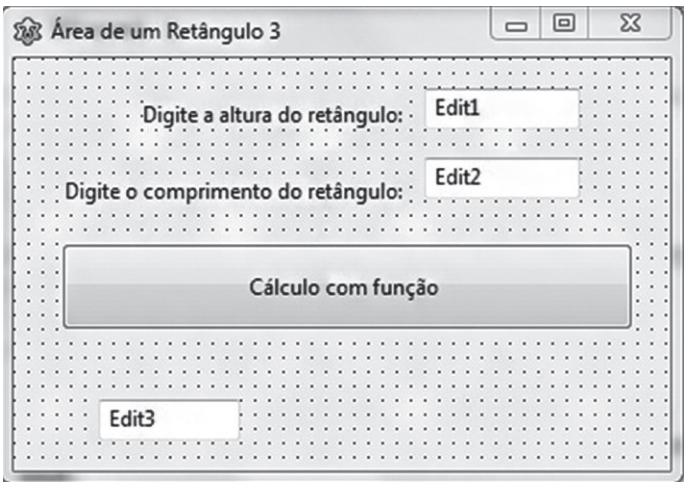


Figura 5.12: Modelo do Formulário do *prj_Aplicaca_16*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste exemplo teremos como retorno um valor com os mesmos tipos de dados dos valores passados para a função.

b) Altere o *caption* do *Form1* para **Área de um Retângulo 3.**

c) Selecione o primeiro componente *label* e altere a propriedade *caption* para **Digite a altura do retângulo**.

d) Acesse o segundo componente *Label* e altere a propriedade *caption* para **Digite o comprimento do retângulo.**

e) *Agora selecione o componente* Edit e altere a propriedade *Text*, deixando-a vazia.

f) Realize o mesmo procedimento com segundo componente *Edit*, posicione-o conforme a Figura 5.12 e altere a propriedade *Text*, deixando-a vazia.

g) Selecione o terceiro componente *Edit* e posicione-o conforme a Figura 5.12, altere a propriedade *Text*, deixando-a vazia, e altere sua propriedade *Enabled* para *False*.

h) Altere a propriedade *caption* do componente *button* para **Cálculo com Função.**

i) Salve *Unit1 com o nome u_Aplicacao_16* e o projeto com *prj_Aplicacao_16*.

j) Agora insira o código do evento *OnClick* do *button1*, conforme Figura 5.13.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    altura, comprimento: real;
begin
     altura:=StrToFloat(Edit1.Text);
     comprimento:=StrToFloat(Edit2.Text);
     Edit3.Text:=FloatToStr(FuncaoArea(altura, comprimento));
end;
```

Figura 5.13: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

A função **FuncaoArea** é chamada na linha de comando *Edit3.Text:=* *FloatToStr*(F**uncaoArea(altura, comprimento**)), ou seja, como sempre haverá retorno como a function, esse retorno será exibido no componente *Edit3* através da função *FloatToStr.*

k) Agora insira o código da Função, Figura 5.14 nas linhas acima do procedimento *TForm1.Button1Click* , conforme Figura 5.13.

```
function FuncaoArea(Alt, comp: real):real;
begin
     FuncaoArea:=Alt *comp;
end;
```

Figura 5.14: *Function* FuncaoArea
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

O retorno da função deste projeto é igual aos tipos de dados dos valores encaminhados para a *function* através das variáveis **altura e comprimento**, conforme Figura 5.14. Analisando as variáveis: **altura e comprimento** são do tipo real e o retorno da Função também é do tipo real, conforme declarado na sintaxe da *function*.

O retorno do valor é realizado com o próprio nome da *Function*, ou seja, **FuncaoArea:= Alt * comp.**

l) Salve tudo e execute o projeto teclando **F9**.

Vamos praticar um pouco do que acabamos de estudar? Crie um novo programa e exiba como resultado o produto de dois números inteiros através de uma função.

Neste próximo projeto usamos uma função com o tipo de retorno diferente dos tipos de dados dos valores passados para função.

a) Para isso criaremos o projeto e montaremos o formulário de acordo com a Figura 5.15.

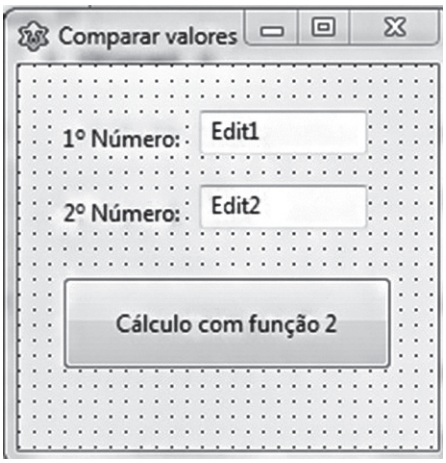


Figura 5.15: Modelo do Formulário do *prj_Aplicacao_17*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Compara Valores.**

c) Através do primeiro componente *label*, altere a propriedade *caption* para **1º Número.**

d) Faça mesma coisa com o segundo componente *label* e altere a propriedade *caption* para **2º Número.**

e) Agora selecione o componente *Edit* e altere a propriedade *Text*, deixando-a vazia.

f) Através do segundo componente *Edit*, posicione-o conforme a Figura 5.15 e altere a propriedade *Text*, deixando-a vazia.

g) Acesse o componente *button* e altere a propriedade *caption* para **Cálculo com Função 2.**

h) Salve *Unit1* com o nome *u_Aplicacao_17* e o projeto com *prj_Aplicacao_17*.

i) Selecione o evento *OnClick* do *button1* e digite o código, conforme Figura 5.16.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    numero1, numero2: Integer;
begin
     numero1:=StrToInt(Edit1.Text);
     numero2:=StrToInt(Edit2.Text);
     if (MesmoValor(numero1, numero2)) then
        ShowMessage('Valores iguais!')
     else
        ShowMessage('Valores diferentes!');

end;
```

Figura 5.16: *OnClick* do *Buttom1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

j) Agora digite o código da função **MesmoValor,** Figura 5.17, nas linhas acima do procedimento *TForm1.Button1Click* Figura 5.16.

```
function MesmoValor(valor1, valor2: integer):boolean;
begin
    MesmoValor:= valor1=valor2;
end;
```

Figura 5.17: *Function* MesmoValor
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observe que no corpo da estrutura condicional *if*, Figura 5.16, a função **MesmoValor** é chamada, passando os valores armazenados nas variáveis **numero1 e numero2** para as variáveis **valor1 e valor2,** Figura 5.17, todas as variáveis do tipo de dado inteiro, porém o retorno da função será do tipo *boolean*, conforme a Figura 5.17.

O propósito do projeto é comparar dois valores, conforme códigos inseridos na Figura 5.18. Se os valores forem iguais, retornará verdadeiro (*true*); caso contrário, retornará falso (*false*). Conforme o resultado retornado, esse valor substituirá a função conforme linha de comando *if* (**MesmoValor**) *then*; se *true*, será exibido através do *ShowMessage*, **Valores iguais**, e se for *false* surgirá a mensagem **Valores diferentes.**

```
procedure TForm1.Button1Click(Sender: TObject);
var
    numero1, numero2: Integer;
begin
     numero1:=StrToInt(Edit1.Text);
     numero2:=StrToInt(Edit2.Text);
     if (MesmoValor(numero1, numero2)) then
        ShowMessage('Valores iguais!')
     else
        ShowMessage('Valores diferentes!');

end;
```

Figura 5.18: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

As possíveis respostas serão exibidas conforme Figura 5.19 para valores iguais e conforme Figura 5.20 para valores diferentes.

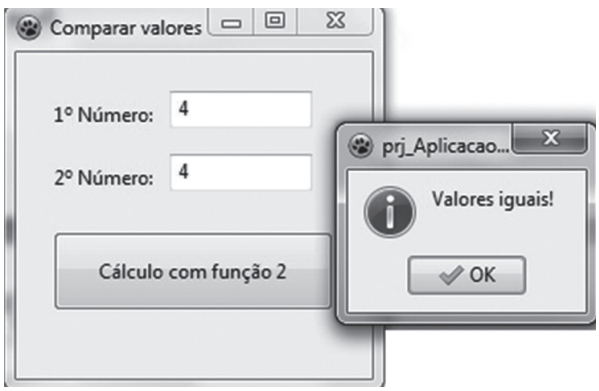


Figura 5.19: Janela de Execução do *prj_Aplicacao_17* para valores iguais
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

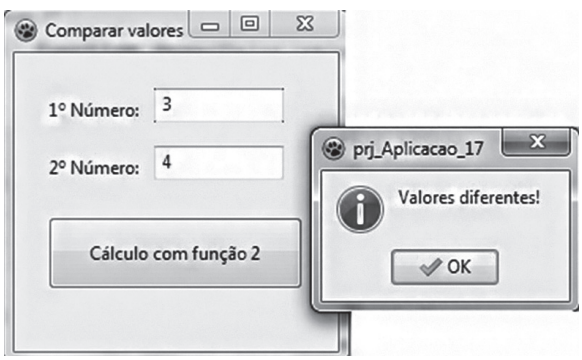


Figura 5.20: Janela de Execução do *prj_Aplicacao_17* para valores diferentes
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

k) Salve tudo e execute o projeto teclando **F9.**

5.3 *Arrays* (vetores e matrizes)

Arrays são estruturas de dados homogêneas. Na programação, um *array* é conhecido como vetor (*arrays* unidimensionais) ou matriz (*arrays* multidimensionais). Os *arrays* são compostos por conjuntos de elementos de dados com o tamanho definido e com todos seus elementos com o mesmo tipo de dados, por isso chamado de estrutura **homogênea**.

Os elementos individuais são acessados por sua posição através de um índice. O índice é uma sequência de números inteiros.

Sintaxe na declaração de *array* unidimensional:

Variável: *array*[valor **inicial** .. valor **final]** *of* [tipo do dados];

Exemplo: nomes: *array*[1.. 5] *of string*[30];
Acesso a cada elemento:
nomes[contador]:= Nome_pessoas;

Onde a variável **contador** será o índice do vetor, ou seja, serão geradas as posições 1 a 5 para armazenar os valores.

Neste próximo projeto utilizaremos variáveis comuns já conhecidas e no *prj_Aplicacao_19*, faremos o aperfeiçoamento do *prj_Aplicacao_18* com o uso da estrutura *array*.

a) Para criar este projeto, montaremos o formulário de acordo com a Figura 5.21.

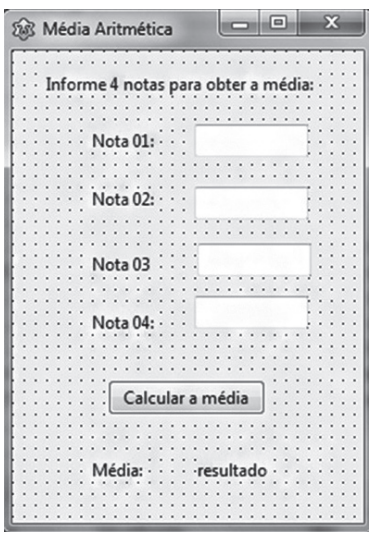


Figura 5.21: Modelo de Formulário do *prj_Aplicacao_18*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Média Aritmética.**

c) Altere a propriedade *caption* do primeiro componente *label* e: **Informe 4 notas para obter a média.**

d) Realize o mesmo procedimento para 2º, 3º, 4º e 5º componentes *label*, alterando a propriedade *caption respectivamente para* **Nota 01, Nota 02, Nota 03 e Nota 04.**

e) Selecione o sexto componente *label* e altere a propriedade *caption* para **Média** e posicione conforme Figura 5.21 e realize o mesmo procedimento com o sétimo componente *Label* alterando *caption* para **Resultado** e também posicione conforme Figura 5.21.

f) Selecione todos os quatro componentes *Edits* e altere suas respectivas propriedades *Text*, deixando-as vazias e coloque na posição de acordo com a Figura 5.21.

g) Altere a propriedade *caption* para **Calcular a Média** do componente *button*.

h) Salve *Unit1* com o nome *u_Aplicacao_18* e o projeto com *prj_Aplicacao_18*.

i) Acesse o evento *Onclick* do *button1* e insira o código, conforme Figura 5.22.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   N1, N2, N3, N4: real;
   media: real;
   confirma: integer;
   respMedia: String;
begin
     val(Edit1.Text, N1, confirma);
     val(Edit2.Text, N2, confirma);
     val(Edit3.Text, N3, confirma);
     val(Edit4.Text, N4, confirma);
     media:= (N1 + N2 + N3 + N4)/4;
     Str(media:3:2, respMedia);
     Label7.Caption := respMedia;

end;
```

Figura 5.22: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Conforme o evento *OnClick* da Figura 5.22, observamos que foram declaradas quatro variáveis para receber quatro notas e que as referidas variáveis são todas do tipo real. E para exibir o resultado da média aritmética, criou-se a variável **média**, a qual irá receber o resultado da operação, ou seja, a soma das quatro notas dividida por quatro e exibida no componente *Label7*.

Ao executar o projeto, surgirá o *Form* para inserção dos dados e exibição do resultado, conforme a Figura 5.23.

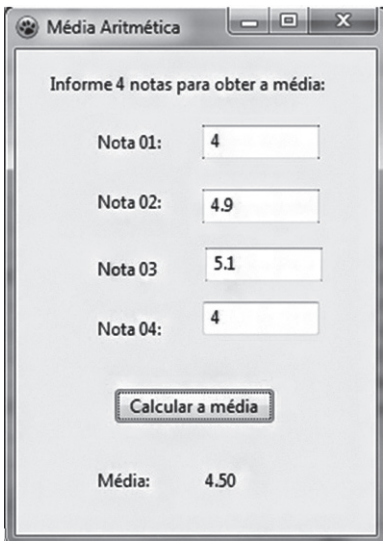


Figura 5.23: Janela de execução do *prj_Aplicacao_18*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

j) Salve tudo e execute o projeto teclando **F9**.

5.3.1 *Array* unidimensional

Agora, para aprimorar o *prj_Aplicacao_18*, mudaremos as quatro variáveis responsáveis por receber as notas do usuário em uma estrutura de *array*.

a) Para isso, criaremos este projeto e montaremos o formulário de acordo com *prj_Aplicacao_18*, conforme a Figura 5.24.

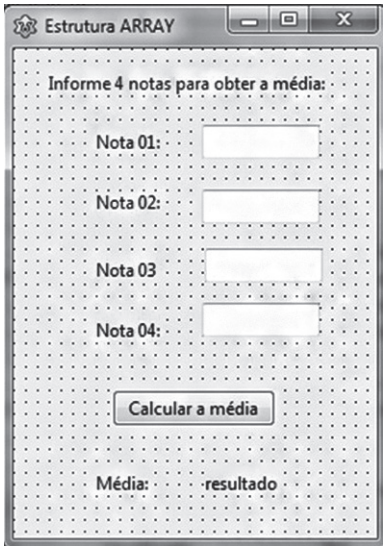


Figura 5.24: Modelo do Formulário do *prj_Aplicacao_19*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Estrutura *array*.**

c) Para montar a parte visual deste projeto, siga os passos estabelecidos no *prj_Aplicacao_18*, mantendo todos os passos, exceto na inserção do evento *OnClick*.

d) Agora salve *Unit1* com o nome *u_Aplicacao_19* e o projeto com *prj_Aplicacao_19*.

e) Vamos inserir o código do evento *OnClick* do *button1*, conforme Figura 5.25.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Notas: array[1..4] of real;
   soma, media: real;
   contador, confirma: integer;
   respMedia: String;
begin
     val(Edit1.Text, Notas[1], confirma);
     val(Edit2.Text, Notas[2], confirma);
     val(Edit3.Text, Notas[3], confirma);
     val(Edit4.Text, Notas[4], confirma);
     soma := 0;
     for contador := 1 to 4 do
         soma := soma +Notas[contador];

     Media:= soma/4;
     Str(media:3:2, respMedia);
     Label7.Caption := respMedia;
end;
```

Figura 5.25: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Conforme a Figura 5.25, percebemos que a variável chamada **Notas** é declarada como um *array* e que os seus quatro elementos são do tipo **real.**

A referida variável **Notas** possui quatro índices para armazenar seu conjunto de valores, conforme **Notas[1], Notas[2], Notas[3] e Notas[4].** As quatro notas são somadas e o resultado atribuído na variável **soma;** logo após, a variável **Media** recebe a variável **soma** dividida por 4 e o resultado da média Aritmética exibido no componente *Label7*.

f) Salve tudo e execute o projeto teclando **F9.**

5.3.2 *Array* multidimensional

Neste projeto trabalharemos com o uso de uma matriz (*array* com mais de uma dimensão).

a) Para isso, monte seu projeto conforme a Figura 5.26.

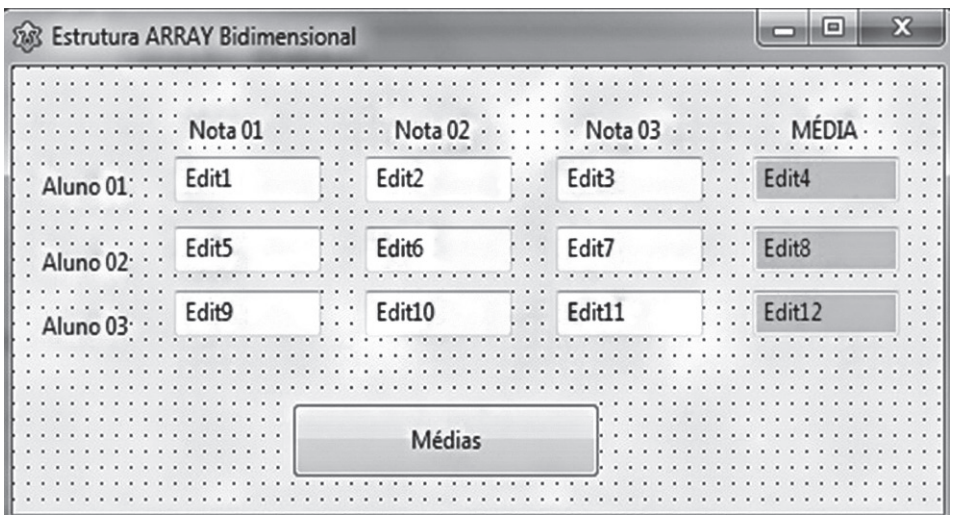


Figura 5.26: Modelo do Formulário do *prj_Aplicacao_20*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para Estrutura *array* bidimensional.

c) Insira três componentes *label*, altere as suas propriedades *caption* e posicione conforme Figura 5.26.

- *label1* para **Aluno1**
- *label2* para **Aluno2**
- *label3* para **Aluno3**

d) Agora insira mais quatro componentes *label* e altere a propriedade *caption* e posicione conforme Figura 5.26.

- *label4* para **Nota 01**
- *label5* para **Nota 02**
- *label6* para **Nota 03**
- *label7*: **Média**

e) Insira 12 componentes *Edit* e altere a propriedade *Text*, deixando-a vazia e colocando-a nas posições de acordo com a Figura 5.26.

f) Ainda nos componentes *Edit4*, *Edit8* e *Edit12*, altere a propriedade *Enabled* para *false*, a fim de que o usuário não consiga armazenar informações. E altere a propriedade *color* para *clAqua*.

g) Agora colocaremos o componente *button* e alteraremos a propriedade *caption* para **Média.**

h) Agora salve *Unit1* com o nome *u_Aplicacao_20* e o projeto com *prj_Aplicacao_20*.

i) Vamos inserir o código do evento *OnClick* do *button1*, conforme Figura 5.27.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Notas:array[1..3, 1..3] of real;
   cont_1, cont_2, confirma: integer;
   media: array[1..3] of real;
   soma:real;
   MediaStr: String;
begin
   val(Edit1.Text, Notas[1,1], confirma);
   val(Edit2.Text, Notas[1,2], confirma);
   val(Edit3.Text, Notas[1,3], confirma);
   val(Edit5.Text, Notas[2,1], confirma);
   val(Edit6.Text, Notas[2,2], confirma);
   val(Edit7.Text, Notas[2,3], confirma);
   val(Edit9.Text, Notas[3,1], confirma);
   val(Edit10.Text, Notas[3,2], confirma);
   val(Edit11.Text, Notas[3,3], confirma);
   for  cont_1 := 1 to 3 do
   begin
        soma:=0;
        for cont_2 := 1 to 3 do
            soma := soma + Notas[cont_1,cont_2];
        media[cont_1] := soma/3;
   end;
   Str(media[1]:3:2,MediaStr);
   Edit4.Text :=mediaStr;
   Str(media[2]:3:2,MediaStr);
   Edit8.Text :=mediaStr;
   Str(media[3]:3:2,MediaStr);
   Edit12.Text :=mediaStr;

end;
```

Figura 5.27: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observando o formulário, percebemos que cada *Edit* é uma posição em nossa matriz, ou seja, **Notas[1,1], Notas[1,2] e Notas[1,3]** para calcular a média do Aluno 01 e assim acontecerá a cada aluno, ou seja, Aluno 02 e Aluno 03.

Declaramos também outra variável chamada ***media***, também um *array*, porém com uma única dimensão para armazenar os resultados das médias dos três respectivos alunos.

Em um *array* multidimensional a quantidade de índice será a quantidade de dimensões. Neste projeto foi declarado um *array* com duas dimensões; portanto, foram criadas duas variáveis que controlam os índices, ou seja, **cont_1**, responsável pelo índice da linha, e **cont_2**, responsável pelo índice da coluna.

Ao ser executado, o projeto terá o seguinte formato, conforme Figura 5.28. Nele percebemos que os resultados das médias ficam exibidos nos componentes: *Edit4*, *Edit8* e *Edit12*.

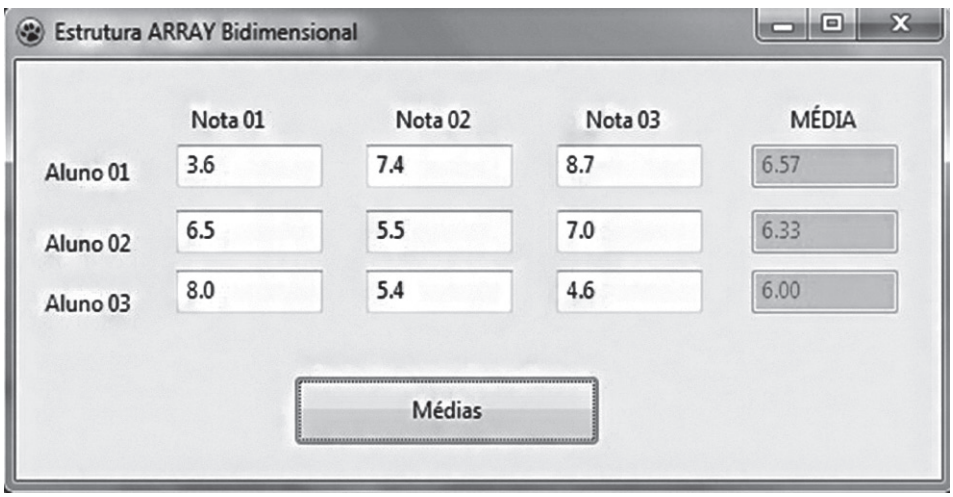


Figura 5.28: Janela em execução do *prj__Aplicacao_20*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Eu sei que você vai querer saber um pouco mais sobre estruturas homogêneas e heterogêneas. Para isso, acesse o endereço http://lazarus.codigolivre.org.br/index.php/P%C3%A1gina_principal

j) Salve tudo e execute o projeto teclando **F9**.

5.4 *Record* (registro)

Os tipos de dados mais comumente usados foram vistos com maior ênfase nas linguagens de programação: *integer, real, string* e boolean.

Outra forma de definir uma variável em *Object* Pascal é através do tipo heterogêneo *record*, ou seja, registros. Esse tipo é diferente dos demais, pois podemos declarar diversas variáveis, as quais podem ser de tipos de dados diferentes.

Imagine que desejamos armazenar informações de um determinado aluno, tais como: **Nome, Notas1, Nota2 e Media**. Na forma tradicional, seria necessário definir uma variável para cada tipo de informação, ou seja:

```
var
        Nome        : string;
        Nota1       : real;
        Nota2       : real;
        Media       : real;
```

Utilizando o tipo *record*, a definição seria a seguinte:

```
var
Aluno       : record
        Nome        : string;
        Nota1       : real;
                Nota2       : real;
                Media       : real;
        end;
```

Quando definimos uma variável do tipo *record*, devemos definir também quais serão os campos dessa variável (***Nome, Notas1, Nota2, Media***), junto com seus respectivos tipos.

Neste projeto trabalharemos com a aplicação de um registro simples (*record*).

a) Para isso monte seu projeto conforme a Figura 5.29.

Figura 5.29: Modelo do Formulário do *prj_Aplicacao_21*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

b) Altere o *caption* do *Form1* para **Cadastro de Alunos.**

c) Através do primeiro componente *label*, altere sua propriedade *caption* para **Aluno**.

d) Acesse os três componentes *label* e altere as suas propriedades *caption* e posicione conforme Figura 5.29.

- *label2* para **Média** 1
- *label3* para **Média 2**
- *label4* para **Média Geral**

e) Agora faça o mesmo procedimento com os três *Edits* e altere a propriedade *Text*, deixando-a vazia e colocando-a nas posições de acordo com a Figura 5.29.

f) Altere a propriedade *Alignment* dos respectivos *Edit2* e *Edit3*, conforme Figura 5.30.

Figura 5.30: Propriedade *Alignment* do *Edit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

g) Selecione *Label4* e altere a propriedade *Visible* para *False*, conforme Figura 5.31, para quando for executado o projeto, não aparecer para o usuário, ou seja, na inserção dos dados realizado no botão **Cadastrar,** torná-los visíveis apenas na apresentação dos dados realizada pelo botão **Apresentar;**

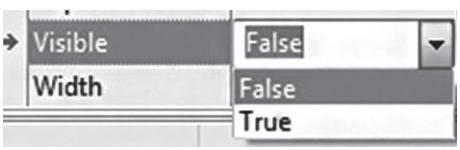


Figura 5.31: Propriedade *Visible* dos componentes *Label* e *Edit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

h) Realize o mesmo procedimento com *Edit4*, tornando Invisível; e altere a propriedade *Enabled* para *False*.

i) Agora colocaremos dois componentes *buttons* e alteraremos suas respectivas propriedades *caption*:

- *button1* para **Cadastrar**.
- *button2* para **Apresentar.**

j) Salve *Unit1* com o nome *u_Aplicacao_21* e o projeto com *prj_Aplicacao_21*.

k) Antes de inserirmos o evento *OnClick* dos botões, é necessário criarmos nossa estrutura *record*. Para esse procedimento, acesse nosso editor de código clicando em *F12*; localize a cláusula *type* e insira a estrutura registro, conforme Figura 5.32.

```
Cad_Alunos = Record
               nome : String[30];
               Media1, Media2, MediaGeral: real;
end;
```

Figura 5.32: Declaração do *RECORD Cad_Alunos*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Conforme a Figura 5.32, o registro criado se chama **Cad_Alunos** e possui quatro campos: **nome, Media1, Media2 e MediaGeral**, ou seja, com isso criamos um novo tipo de dado, chamado **Cad_Alunos.**

Para termos acesso a cada um de seus campos, é necessário declararmos uma variável, a qual deverá ser do novo tipo criado pelo desenvolvedor, conforme Figura 5.33, declarada na cláusula *var*.

```
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;
  alunos: Cad_Alunos;

implementation
```

Figura 5.33: Declaração da variável Alunos na cláusula VAR
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

A variável criada na cláusula *var* se chama **Alunos** e para termos acessos aos campos é importante respeitar a sua sintaxe, ou seja, **alunos.nome, alunos.Media1, alunos.Media2, alunos.MediaGeral**. Observe que a variável **alunos** é separada de seu campo por um ponto (.).

l) Agora vamos inserir o código do evento *OnClick* do *button1* (botão **Cadastrar**), conforme Figura 5.34.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Alunos.nome:= Edit1.Text;
     Alunos.Media1:= StrToFloat(Edit2.Text);
     Alunos.Media2:= StrToFloat(Edit3.Text);
     Alunos.MediaGeral:= (Alunos.Media1 + Alunos.Media2)/2;
     Limpar;


end;
```

Figura 5.34: *OnClick* do *Button1*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Conforme o evento *OnClick*, os valores foram armazenados através dos *Edit1* (**alunos.nome**), *Edit2* (**alunos.Media1**) e *Edit3* (**alunos.Media2**).

Na última linha de comando do evento *OnClick* do botão **Cadastrar**, chamamos o procedimento **Limpar**, que tem o intuito de quando clicar no botão armazenar os valores fornecidos pelo usuário nas variáveis: **Alunos.nome, Alunos.Media1 e Alunos.Media2** e limpe os seus respectivos componentes: *Edit1, Edit2, Edit3* e *Edti4.*

m) Acesse o editor de código e após cláusula *implemantion* insira o procedimento **Limpar**, conforme Figura 5.35.

```
procedure Limpar;
begin
  Form1.Edit1.Text := '';
  Form1.Edit2.Text := '';
  Form1.Edit3.Text := '';
  Form1.Edit4.Text := '';
end;
```

Figura 5.35: *Procedure* Limpar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Observe que no procedimento para reconhecer a existência dos *Edit1, Edit2* , *Edit3* e *Edit4* foi preciso colocar antes do componente a palavra *Form1*, pois o procedimento está fora do *TForm1.Button1Click*.

Ao executar o botão **Cadastrar**, teremos a seguinte visão para a inserção dos dados, conforme Figura 5.36.

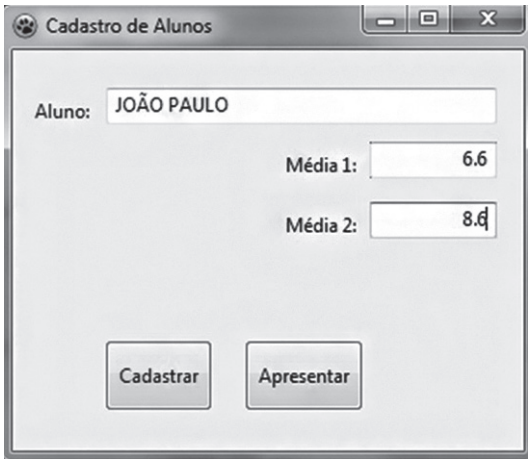


Figura 5.36: Execução do *prj_Aplicacao_21* para o botão Cadastrar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

n) Agora vamos inserir o código do evento *OnClick* do *button2* (botão **Apresentar**), conforme Figura 5.37.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  resposta: String;
begin
   Edit1.Text := Alunos.nome;
   Edit2.Text := FloatToStr(Alunos.Media1);
   Edit3.Text := FloatToStr(Alunos.Media2);;
   Str(Alunos.MediaGeral:3:2, resposta);
   Label4.Visible:= true;
   Edit4.Visible:= true;
   Edit4.Text := resposta;

end;
```

Figura 5.37: *OnClick* do *Button2*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste evento iremos apresentar os valores antes armazenados nas variáveis: *Edit1* (**Alunos.nome**), *Edit2* (**Alunos.Media1**) e Edit3 (**Alunos.Media2**) e tornar visível os componentes *Label4* (**Média Geral**) e *Edit4* que exibirão o resultado, que é a Média Geral.

Ao executar o projeto, teremos o seguinte resultado, conforme Figura 5.38 e o valor inserido pelo o usuário.

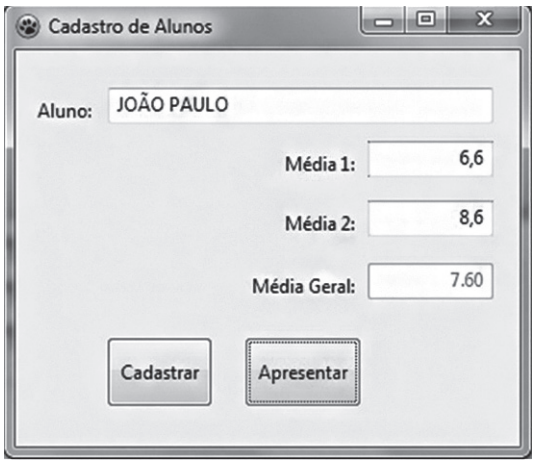


Figura 5.38: Execução do *prj_Aplicacao_21* para o botão Apresentar
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Após execução do projeto, percebemos a visibilidade dos componentes *Label4* e *Edit4*, conforme presente no código do evento do botão **Apresentar**: *Label4.Enabled* := *True* e *Edit4.Enabled* := *True*.

Agora trataremos da inserção de dados nos *Edits*, tornando possível apenas o armazenamento de caracteres numéricos, o ponto (.) separação da parte inteiro da fracionária, já que são valores do tipo **real**, e claro, permitem a ação do ***backspace***.

A-Z

Backspace
Tecla do teclado usada para apagar os caracteres que antecedem o cursor do mouse.

```
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: char);
begin
     if not(key in['0'..'9']) and not(key in [#8]) and
        not(key in [#46]) then
        Key := #0;
end;
```

Figura 5.39: Evento *OnKeyPress* do *Edit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2

Neste evento a condição *if*, quando verdadeira, insere no componente *[#0]* que representa a inserção de "nada", tornando sem efeito a inserção. O símbolo *[#8]* representa a inserção do ponto (.) e o símbolo *[#46]* representa a permissão do uso do *backspace*.

p) Repita o mesmo procedimento, ou seja, o evento (*OnKeyPress*) para o componente *Edit3*.

q) Salve tudo e execute o projeto teclando **F9.**

Resumo

Nesta aula abordamos introdução e aplicação de procedimento, destacando os procedimentos **com** e **sem parâmetros**; criamos projetos com o uso de *functions*, estrutura homogênea (*array*) com uma dimensão e multidimensão e a aplicação da estrutura heterogênea *record*.

Atividades de aprendizagem

1. Crie um projeto usando procedimento sem parâmetros para calcular o volume de um cubo.

   Formula= lado ^{3}

2. Desenvolva um projeto da questão anterior aplicando o procedimento com parâmetro e passagem por valor.

3. Desenvolva um projeto para calcular a área de um triângulo. Nesse projeto deverá ser aplicado o uso do procedimento com parâmetro e passagem por referência.

   Formula Area= (Altura X base) / 2

4. Crie um projeto aplicando o conhecimento de *array* que leia três valores inteiros do usuário e exiba para o usuário como resultado os números em ordem crescente.

5. Crie um projeto para Cadastrar um registro, conforme a Figura 5.40 a seguir.

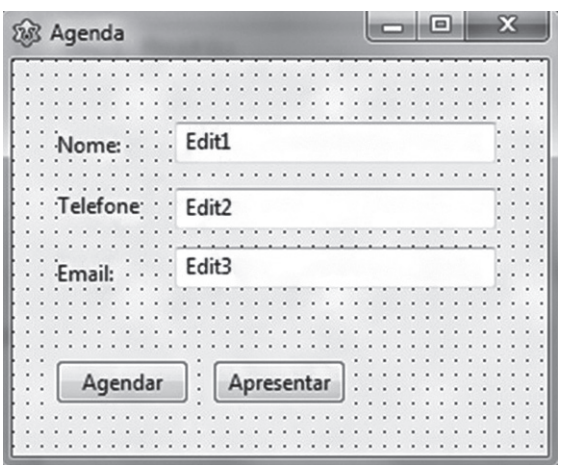


Figura 5.40: Evento *OnKeyPress* do *Edit*
Fonte: *Print screen* do Lazarus IDE v0.9.28.2 realizado pelo autor

Poste suas respostas no AVEA.

Referências

LAZARUS. Disponível em: <http://www.lazarus.freepascal.org>. Acesso em: 4 dez. 2011.

LAZARUS TUTORIAL. Disponível em: <http://wiki.lazarus.freepascal.org/Lazarus_Tutorial/pt acessado em 04/12/2011>. Acesso em: 4 dez. 2011.

Software Lazarus IDE v0.9.28.2

Currículo do professor-autor

Geraldo Nunes da Silva Júnior é bacharel em Ciência da Computação, licenciado em Letras/Inglês e pós-graduado *lato sensu* em Docência do Ensino Superior pela Universidade Estadual do Piauí. É professor do quadro efetivo do Instituto Federal de Ciência e Tecnologia do Piauí desde 2009, Campus Piripiri, professor de Informática do quadro efetivo da Rede Pública Municipal de Educação. Foi professor da Universidade Estadual do Piauí por sete anos, lecionando as disciplinas: Lógica de Programação, Linguagens de Programação, Estrutura de Dados, Banco de Dados. Faz parte da Equipe de Educação a Distância do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, atuando como professor regente e como conteudista.

rede e-Tec Brasil

ISBN 978-85-67082-04-2