

Alinhamentos III-T

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Alinhamentos III-T



Média III-T



$$M.III-T = LV (25\%) + LM (25\%) + LF (25\%) + TF (25\%)$$



Boas práticas de programação



Os trabalhos devem ser feitos em duplas
(Programação em Par)

Onde Publicar os Trabalhos?

Onde os **trabalhos** devem ser **publicados** e **documentados**?



GITHub.com



Onde Encontrar o Material?

Onde encontrar o slide e os conteúdos?
[GITHub.com/RaffaelSchemmer/A2](https://github.com/RaffaelSchemmer/A2)



GITHub.com

Cronograma

➤ Setembro:

- 1. Revisão VCL + Recuperação II-T.
- 2. Vetores + LV.
- 3. Matrizes + LM.
- 4. Procedimentos e Funções + LF.

Cronograma

➤ Outubro:

- 1. Desenvolvimento do TF (Telas).
- 2. Desenvolvimento do TF (Cadastro + Visualização).
- 3. Desenvolvimento do TF (Remoção + Edição).
- 4. Geração de Instalador e Repositório.

Boas práticas de programação

- Entre programadores:
 - É legal definir algumas regras comuns.
 - Regras baseadas no que diz o coletivo (consenso entre a comunidade).
 - Vamos aprender algumas convenções que tornam o código mais agradável.
 - Respeitando as regras do Object Pascal (Delphi) e do RAD (VCL).
- Regra [1] : Comentários
 - Lembre que o Delphi possui 3 tipos de comentários sendo eles.
 - // Sou um comentário de linha
 - (* Comentário de bloco *) ou { Comentário de bloco }



Boas práticas de programação

- Regra [2]: Últimas dicas
 - Indentação e uso de TABs.
 - Não mudam a lógica mas facilitam o entendimento e o debug do código.
 - Utilize o atalho Ctrl + D para indentar automaticamente o código.
 - Utilize um espaçamento de linhas correto entre cada função e cada bloco.
 - Utilize BEGIN e END; sempre que possível:
 - Isso estrutura os blocos condicionais (if) e os laços de repetição (while/for).
 - Crie códigos estruturados:
 - A legibilidade favorece o entendimento das coisas.



Atalhos do DELPHI

Crtl + Space

O atalho acima deve ser utilizado SEMPRE
Ele permite completar o código!

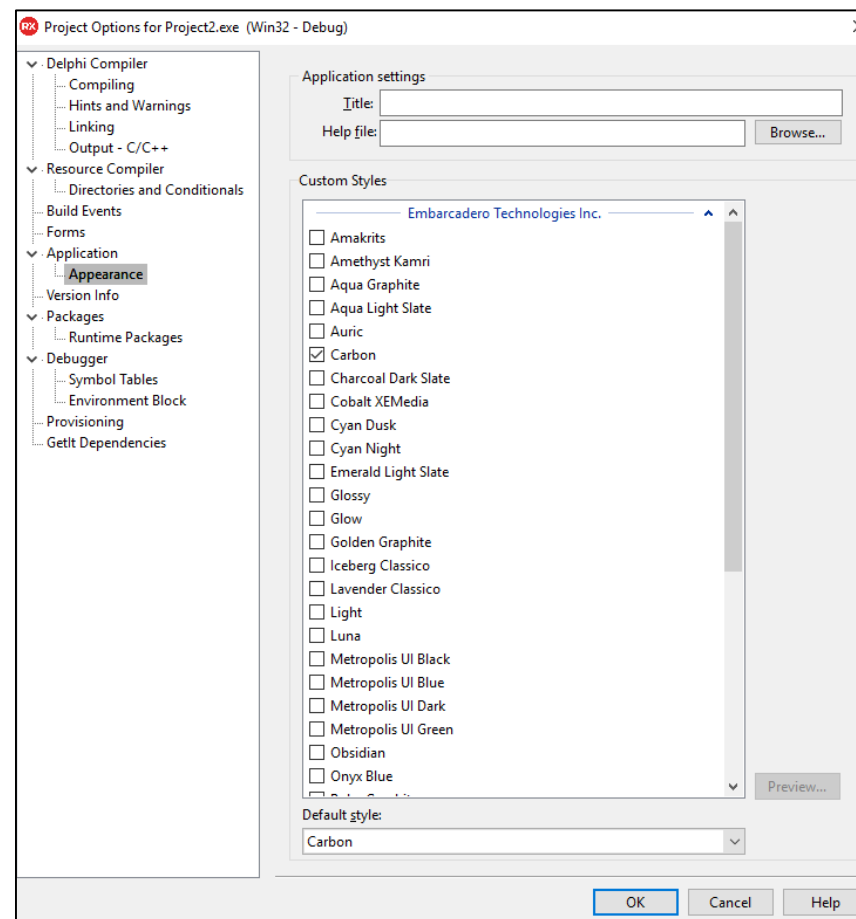


Componentes VCL (I e II Trimestre)

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Componentes VCL (I e II Trimestre)

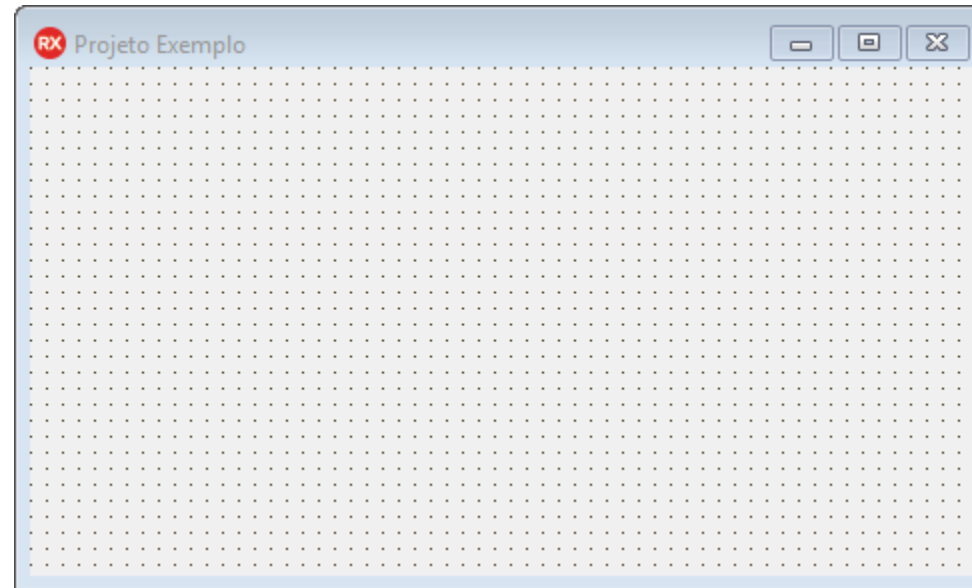


Estilizando todo o projeto (Apperance)



Escolha um estilo de projeto

[1] TForm (Formulário): Definição



Componente TForm

[1] TForm (Formulário) – Propriedades I

- **Align**: Define a posição do componente no TForm.
- **Caption**: Troca o rótulo (frase) do Form (cabeçalho).
- **ClientHeight**: Altura do FORM em pixel.
- **ClientWidth**: Largura do FORM em pixel.
- **BorderStyle**: bsSingle (Desabilita a edição do W e do H do FORM).
- **Position**: poDesktopCenter (Centraliza o FORM na tela).
- **Name**: Troca o nome do formulário (Dentro do Código Delphi).
- **Color**: Define a cor de fundo de um formulário.



[1] TForm (Formulário) – Propriedades II



- **BorderWidth**: Define a espessura da borda do FORM utilizando um número inteiro.
- **BorderIcon**: Permite remover os ícones de minimizar/maximizar do formulário.
- **Icon**: Permite modificar o ícone do formulário (32x32 pixels).
- **Visible**: TRUE ou FALSE (Permite fazer o TForm aparecer ou não).
- **FormStyle**: Permite definir se o Form será fixo na tela (fsStayOnTop).
- **WindowState**: Permite definir se o Form será maximizado (wsMaximized).



[1] TForm (Formulário): Eventos

- **onCreate**: Executa toda vez que o form for criado.
- **OnClose**: Executa toda vez que o form for finalizado.
- **onResize**: Executa toda vez que o form for redimensionado.
- **onShow**: Executa toda vez que o form for pedido para ser visualizado.
- **onClick**: Executa toda vez que um click for feito no form.
- **onDblClick**: Executa toda vez que dois clicks forem feitos no form.
- **onMouseEnter**: Executa toda vez que o mouse entrar no form.
- **onMouseLeave**: Executa toda vez que o mouse sair no form.



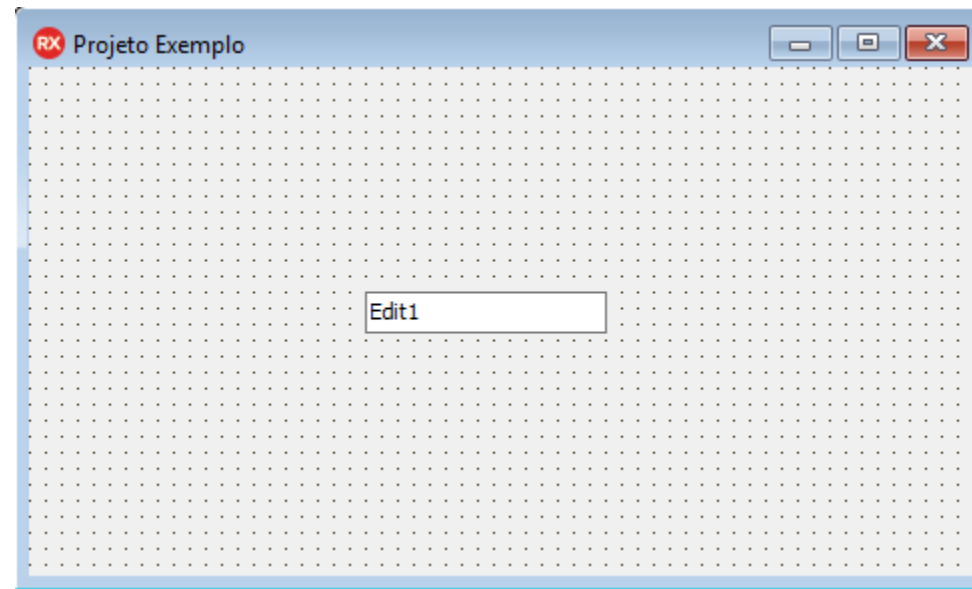
[1] TForm (Formulário): Exemplo

```
procedure TForm1.ChamaFormClick(Sender: TObject);  
begin  
  
    FrmAgenda.Caption := 'Agenda de Contatos';  
    FrmAgenda.ClientHeight := 500;  
    FrmAgenda.ClientWidth := 500;  
    FrmAgenda.Position := poDesktopCenter;  
  
    FrmMenu.Hide;  
    FrmAgenda.Show;  
  
end;
```

Preparando FrmAgenda com 500x500 centralizado
Oculta FrmMenu (Hide) e Mostra FrmAgenda



[2] TEdit (Diálogo): Definição



Componente TForm com um TEdit

[2] TEdit (Diálogo): Propriedades I

- **Align**: Define a posição do componente no TForm.
- **Text**: Permite manipular o conteúdo do TEdit.
- **TextHint**: Define um texto de dica para o componente (caso Text não existir).
- **Height**: Altura do TEdit em pixel.
- **Width**: Largura do TEdit em pixel.
- **ReadOnly**: Permite que o conteúdo do TEdit seja somente lido.
- **MaxLength**: Comprimento máximo do TEdit em caracteres.
- **PasswordChar**: Permite ocultar o texto com (*).
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.

[2] TEdit (Diálogo): Propriedades II

- **Visible**: Torna o componente ocultável.
- **Name**: Troca o nome do TEdit (Dentro do Código Delphi).
- **Color**: Define a cor de fundo de um TEdit.
- **Hint**: Define uma caixa de dica para o componente.
- **ShowHint**: Habilita a dica.
- **Bevel**: (Inner/Kind/Outer): Permite estilizar as bordas do TEdit.
- **CharCase**: Permite definir se as letras serão em CAPS ou não.
- **NumbersOnly**: Permite aceitar apenas números.

[2] TEdit (Diálogo): Eventos

- **onChange**: Executa toda vez que o TEdit for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TEdit.
- **onExit**: Executa toda vez que o TAB sair dentro do TEdit.
- **onClick**: Executa toda vez que um clique for feito no TEdit.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TEdit.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TEdit.
- **onMouseLeave**: Executa toda vez que o mouse sair no TEdit.



[2] TEdit (Diálogo): Exemplo

```
procedure TForm1.ChamaFormClick(Sender: TObject);  
begin  
  
    Entrada.TextHint := 'Entre com a Senha';  
    Entrada.Hint := 'Informe aqui a senha do usuário com 5 chars!';  
    Entrada.ShowHint := TRUE;  
    Entrada.TabOrder := 0;  
    Entrada.Color:= clGreen;  
    Entrada.MaxLength := 5;  
    Entrada.PasswordChar := '*';  
  
end;
```

Preparando TEdit Entrada com TextHint e Hint, TabOrder, MaxLength e PassWordChar



[3] TLabel (Rótulo): Definição



Componente TForm com um TLabel

[3] TLabel (Rótulo): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Caption**: Permite manipular o conteúdo do TLabel.
- **TextHint**: Define um texto de dica para o componente (caso Text não existir).
- **Height**: Altura do TLabel em pixel.
- **Width**: Largura do TLabel em pixel.
- **Visible**: Torna o componente ocultável.
- **Name**: Troca o nome do TLabel (Dentro do Código Delphi).
- **Font**: Define o tamanho, a cor e a fonte de um TLabel.
- **Hint**: Define uma caixa de dica para o componente.
- **ShowHint**: Habilita a dica.

[3] TLabel (Rótulo): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TLabel.
- **onExit**: Executa toda vez que o TAB sair dentro do TLabel.
- **onClick**: Executa toda vez que um clique for feito no TLabel.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TLabel.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TLabel.
- **onMouseLeave**: Executa toda vez que o mouse sair no TLabel.

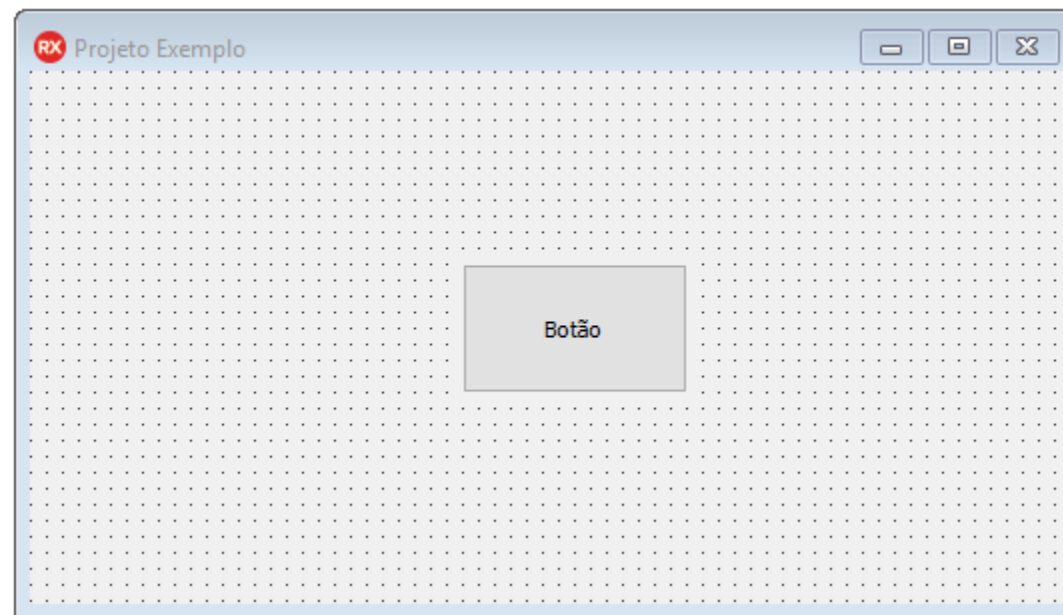


[3] TLabel (Rótulo): Exemplo

```
procedure TForm18.LblClick(Sender: TObject);  
begin  
  
    Lbl.Caption := 'Raffael';  
    Lbl.Font.Color := ClBlue;  
    Lbl.Hint := 'Nome do Usuário!';  
    Lbl.ShowHint := TRUE;  
  
end;
```

Tratamento do evento do onClick do Label
Define o conteúdo do Label, modifica a cor e habilita a dica.

[4] TButton (Botão): Definição



Componente TForm com um TButton

[4] TButton (Botão): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o rótulo do botão.
- **Font**: Define a fonte e o tamanho do texto do botão
- **Enabled**: Define se o botão será habilitado
- **TabOrder**: Define a ordem de chamada do TAB no botão.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.



[4] TButton (Botão): Eventos

- **onClick**: Executa toda vez que um click for feito no botão.
- **onMouseEnter**: Executa toda vez que o mouse entrar no botão.
- **onMouseLeave**: Executa toda vez que o mouse sair do botão.
- **onEnter**: Executa toda vez que o TAB entrar dentro do botão.
- **onExit**: Executa toda vez que o TAB sair dentro do botão.

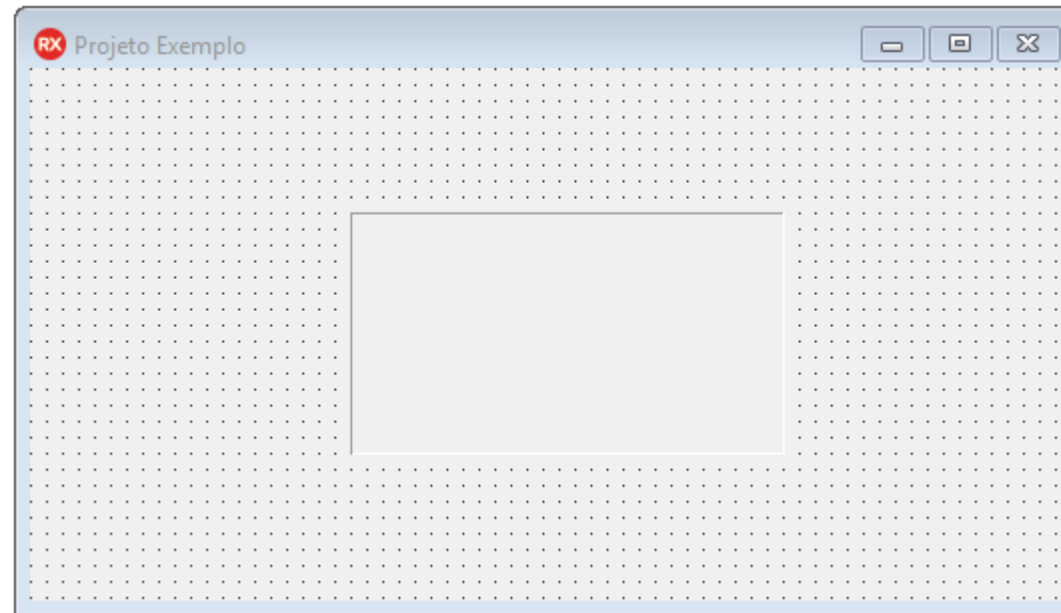


[4] TButton (Botão): Exemplo

```
procedure TForm18.BtnAtivarClick(Sender: TObject);  
  
var state : Boolean;  
begin  
  
    BtnAtivar.Caption := 'Fui clicado';  
    BtnAtivar.Font.Size := 15;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do rótulo do botão e muda o tamanho da fonte

[5] TBevel (Moldura): Definição



Componente TForm com um TBevel

[5] TBevel (Moldura): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Height**: Altura do bevel em pixel.
- **Width**: Largura do bevel em pixel.
- **Hint**: Define uma dica para o bevel.
- **Name**: Define o nome do componente na Unit.
- **ShowHint**: Habilita a dica.
- **Visible**: Torna o componente ocultável.
- **Style**: Define o estilo (para dentro ou para fora) do bevel.
- **Shape**: Define o formato do bevel.

[6] TImage (Imagem): Definição



Componente TForm com um TImage

[6] TImage (Imagem): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Height**: Altura do TImage em pixel.
- **Width**: Largura do TImage em pixel.
- **Hint**: Define uma dica para o TImage.
- **Name**: Define o nome do componente na TImage.
- **ShowHint**: Habilita a dica.
- **Visible**: Torna o componente ocultável.
- **AutoSize**: Define o tamanho do TImage conforme imagem.
- **Picture**: Permite definir uma imagem para TImage.

[6] TImage (Imagem): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TImage.
- **onExit**: Executa toda vez que o TAB sair dentro do TImage.
- **onClick**: Executa toda vez que um clique for feito no TImage.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TImage.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TImage.
- **onMouseLeave**: Executa toda vez que o mouse sair no TImage.

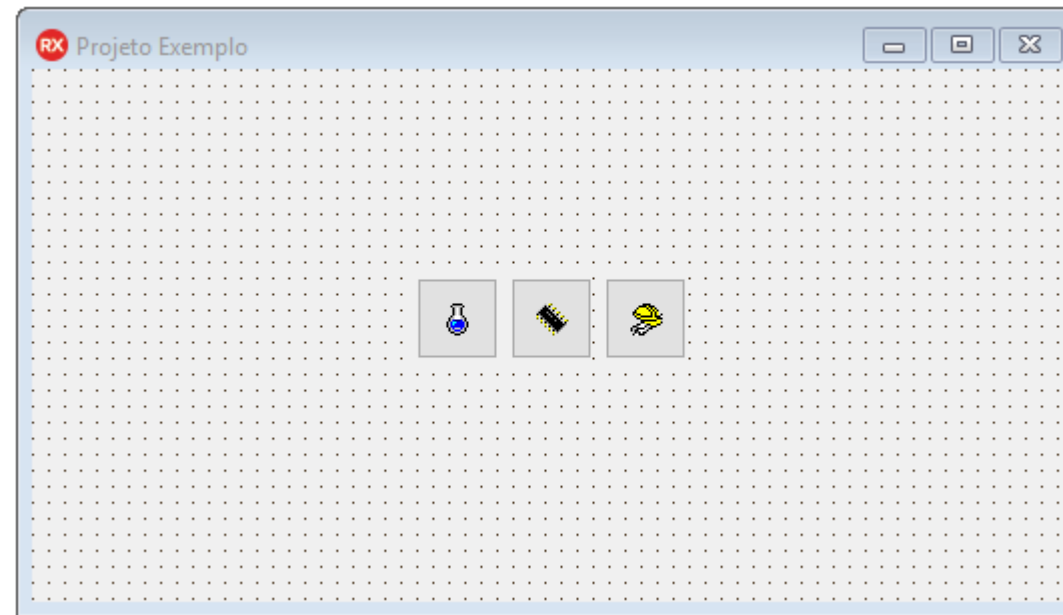


[6] TImage (Imagem): Exemplo

```
procedure TForm18.ImgClick(Sender: TObject);  
begin  
  
    Img.Width := 100;  
    Img.Height := 100;  
    Img.Hint := 'Imagem do Usuário!';  
    Img.ShowHint := TRUE;  
    Img.AutoSize := TRUE;  
    Img.Visible := TRUE;  
  
end;
```

Tratamento do evento do onClick da imagem
Define o tamanho da imagem como 100px x 100px com dica e visibilidade.

[7] TBitBtn (Botão Editável): Definição



Componente TForm com um TBitBtn

[7] TBitBtn (Botão Editável): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Font**: Define a fonte e o tamanho do texto do TBitBtn
- **Enabled**: Define se o TBitBtn será habilitado
- **Caption**: Define o rótulo do TBitBtn.
- **Glyph**: Define uma imagem (BMP para o botão).
- **Height**: Altura do TBitBtn em pixel.
- **Width**: Largura do TBitBtn em pixel.
- **Hint**: Define uma dica para o TBitBtn.
- **Name**: Define o nome do componente na Unit.
- **ShowHint**: Habilita a dica.
- **Kind**: Habilita certos tipos de ícones específicos como bkOK e bkCancel.
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.
- **Visible**: Torna o componente ocultável.

[7] TBitBtn (Botão Editável): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TBitBtn.
- **onExit**: Executa toda vez que o TAB sair dentro do TBitBtn.
- **onClick**: Executa toda vez que um clique for feito no TBitBtn.
- **onDblClick**: Executa toda vez que dois clicks forem feitos no TBitBtn.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TBitBtn.
- **onMouseLeave**: Executa toda vez que o mouse sair no TBitBtn.

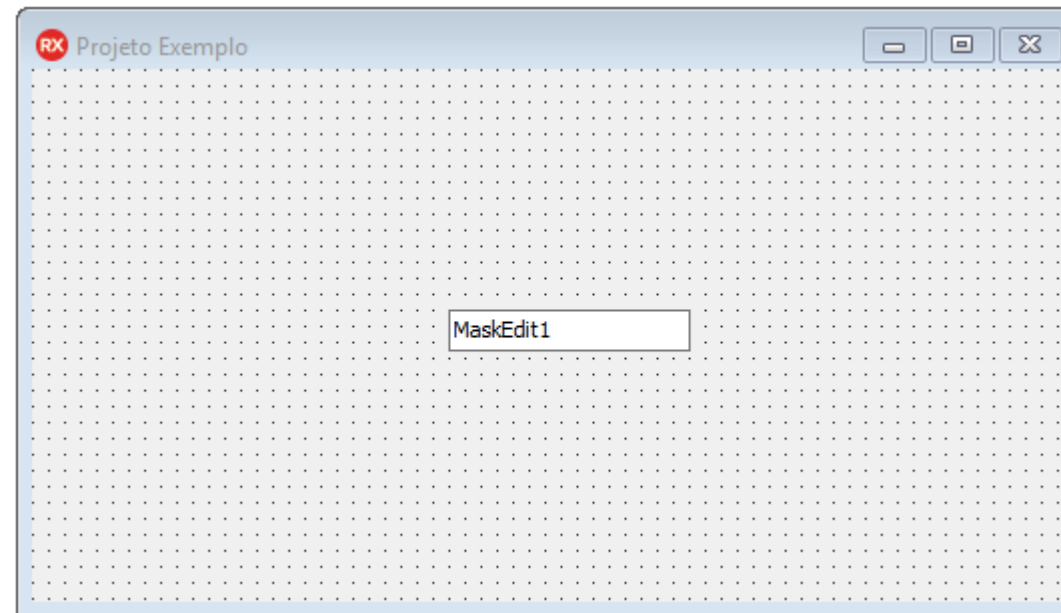


[7] TBitBtn (Botão Editável): Exemplo

```
procedure TForm2.BtnClick(Sender: TObject);  
begin  
  
    Btn.Font.Color := ClGreen;  
    Btn.Font.Size := 10;  
    Btn.Caption := 'Raffael';  
    Btn.Visible := TRUE;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do rótulo do botão e muda o tamanho da fonte e a cor.

[8] TMaskEdit (Diálogo): Definição



Componente TForm com um TMaskEdit

[8] TMaskEdit (Diálogo): Propriedades I

- **Align**: Define a posição do componente no TForm.
- **Text**: Permite manipular o conteúdo do TMaskEdit.
- **TextHint**: Define um texto de dica para o componente (caso Text não existir).
- **Height**: Altura do TMaskEdit em pixel.
- **Width**: Largura do TMaskEdit em pixel.
- **ReadOnly**: Permite que o conteúdo do TMaskEdit seja somente lido.
- **MaxLength**: Comprimento máximo do TMaskEdit em caracteres.
- **PasswordChar**: Permite ocultar o texto com (*).
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.

[8] TMaskEdit (Diálogo): Propriedades II

- **Visible**: Torna o componente ocultável.
- **Name**: Troca o nome do TMaskEdit (Dentro do Código Delphi).
- **Color**: Define a cor de fundo de um TMaskEdit.
- **Hint**: Define uma caixa de dica para o componente.
- **ShowHint**: Habilita a dica.
- **Bevel**: (Inner/Kind/Outer): Permite estilizar as bordas do TMaskEdit.
- **CharCase**: Permite definir se as letras serão em CAPS ou não.
- **EditMask**: Define uma máscara fixa para o TMaskEdit.



[8] TMaskEdit (Diálogo): Eventos

- **onChange**: Executa toda vez que o TMaskEdit for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TMaskEdit.
- **onExit**: Executa toda vez que o TAB sair dentro do TMaskEdit.
- **onClick**: Executa toda vez que um clique for feito no TMaskEdit.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TMaskEdit.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TMaskEdit.
- **onMouseLeave**: Executa toda vez que o mouse sair no TMaskEdit.



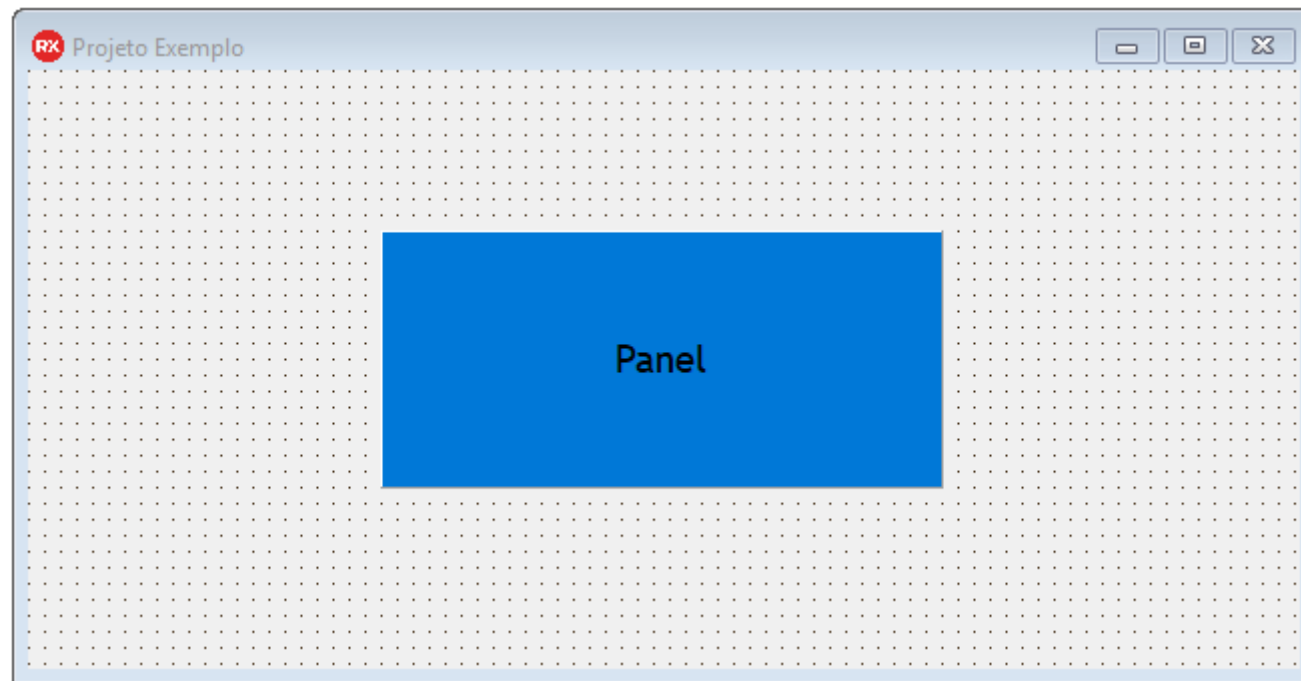
[8] TMaskEdit (Diálogo): Exemplo

```
procedure TForm2.BtnClick(Sender: TObject);  
begin  
  
    MskEdt.MaxLength := 5;  
    MskEdt.ReadOnly := TRUE;  
    MskEdt.Width := 10;  
    MskEdt.Height := 50;  
    MskEdt.PasswordChar := '*';  
  
end;
```

Tratamento do evento do onClick do botão

Define o conteúdo do TMaskEdit com tamanho de 5 posições e PasswordChar char com '*'

[9] TPanel (Container): Definição



Componente TForm com um TPanel

[9] TPanel (Containner): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Permite obter ou modificar o nome do TPanel
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Color**: Define a cor interna do componente.
- **Visible**: Torna o componente ocultável
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.
- **Enabled**: Habilita o acesso ao componente.



[9] TPanel (Container): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TPanel.
- **onExit**: Executa toda vez que o TAB sair dentro do TPanel.
- **onClick**: Executa toda vez que um clique for feito no TPanel.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TPanel.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TPanel.
- **onMouseLeave**: Executa toda vez que o mouse sair no TPanel.

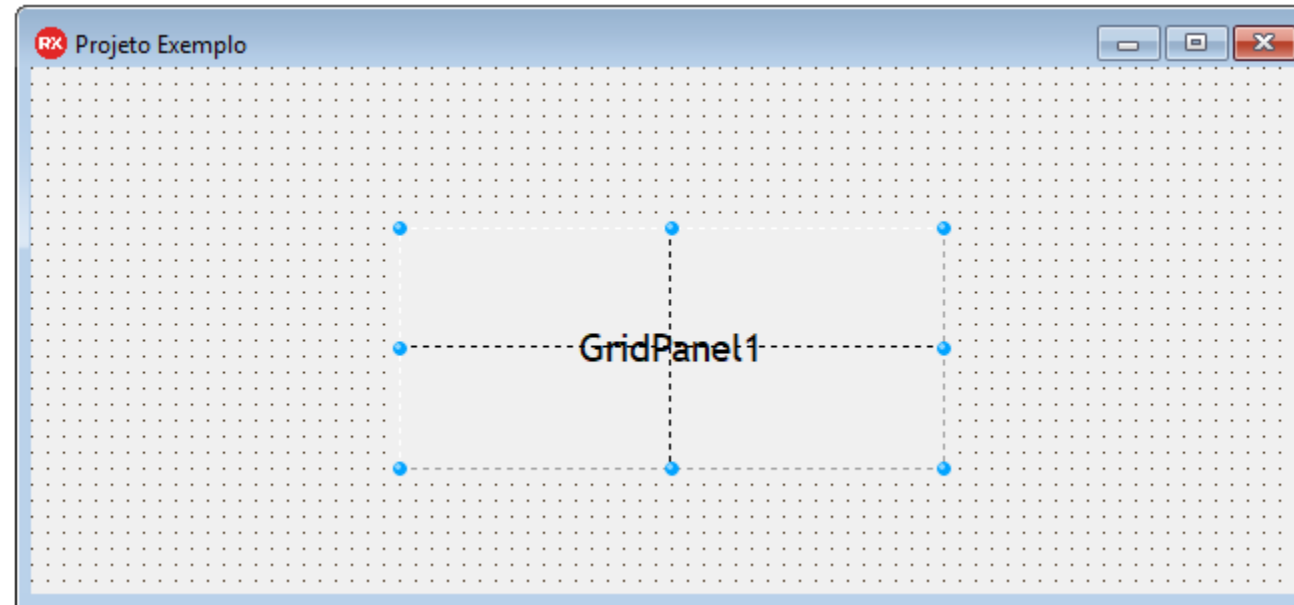


[9] TPanel (Container) – Exemplo

```
procedure TForm1.ChamaFormClick(Sender: TObject);  
begin  
  
    Panel.Width := 250;  
    Panel.Height := 250;  
    Panel.Color := clBlue;  
    Panel.TabOrder := 2;  
    Panel.Enabled := TRUE;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do panel mudando as dimensões, a cor e a ordem do TAB

[10] TGridPanel (Container): Definição



Componente TForm com um TGridPanel

[10] TGridPanel (Container): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Permite obter ou modificar o nome do TGridPanel
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Color**: Define a cor interna do componente.
- **Visible**: Torna o componente ocultável
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.
- **Bevel**: (Inner/Kind/Outer): Permite estilizar as bordas do TGridPanel.
- **Enabled**: Habilita o acesso ao componente.
- **ColumnCollection.Value**: Define o valor absoluto de espaço de cada coluna.
- **RowCollection.Value**: Define o valor absoluto de espaço de cada linha.

[10] TGridPanel (Container): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TGridPanel.
- **onExit**: Executa toda vez que o TAB sair dentro do TGridPanel.
- **onClick**: Executa toda vez que um clique for feito no TGridPanel.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TGridPanel.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TGridPanel.
- **onMouseLeave**: Executa toda vez que o mouse sair no TGridPanel.

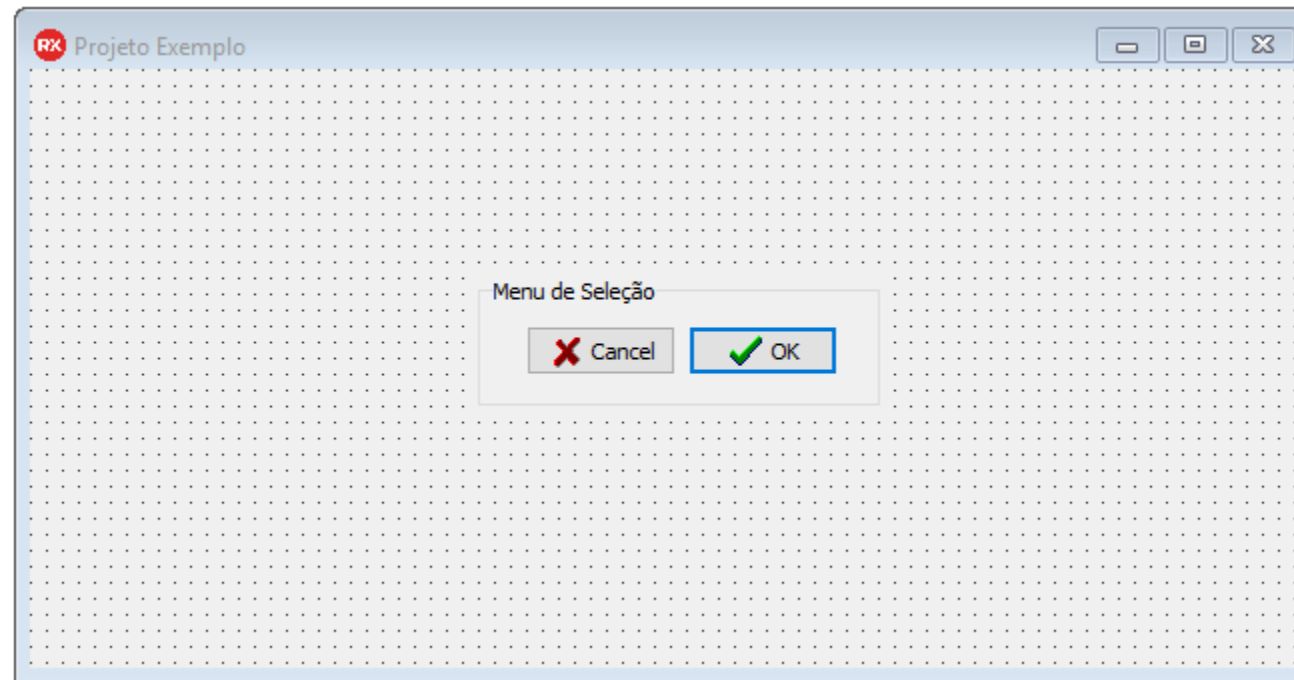


[10] TGridPanel (Container): Exemplo

```
procedure TForm2.BtnClick(Sender: TObject);  
begin  
  
    GrdPnl.Width := 100;  
    GrdPnl.Height := 100;  
    GrdPnl.Visible := TRUE;  
    GrdPnl.BevelKind := bkTile;  
    GrdPnl.ColumnCollection[0].SizeStyle := ssAbsolute;  
    GrdPnl.ColumnCollection[0].Value := 50;  
    GrdPnl.ColumnCollection[1].SizeStyle := ssAbsolute;  
    GrdPnl.ColumnCollection[1].Value := 50;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do TGridPanel em 100 x 100 com duas colunas de 50px cada.

[11] TGroupBox (Container): Definição



Componente TForm com um TGroupBox

[11] TGroupBox (Container): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o rótulo do TGroupBox
- **Font**: Define a fonte e o tamanho do texto do TGroupBox
- **Enabled**: Define se o TGroupBox será habilitado
- **TabOrder**: Define a ordem de chamada do TAB no TGroupBox
- **Hint**: Define uma dica para o TGroupBox
- **ShowHint**: Define se a dica será mostrada ou não
- **Visible**: Define se o botão será visível ou não
- **Height**: Altura do TGroupBox em pixel.
- **Width**: Largura do TGroupBox em pixel.



[11] TGroupBox (Container): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TGroupBox.
- **onExit**: Executa toda vez que o TAB sair dentro do TGroupBox.
- **onClick**: Executa toda vez que um clique for feito no TGroupBox.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TGroupBox.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TGroupBox.
- **onMouseLeave**: Executa toda vez que o mouse sair no TGroupBox.



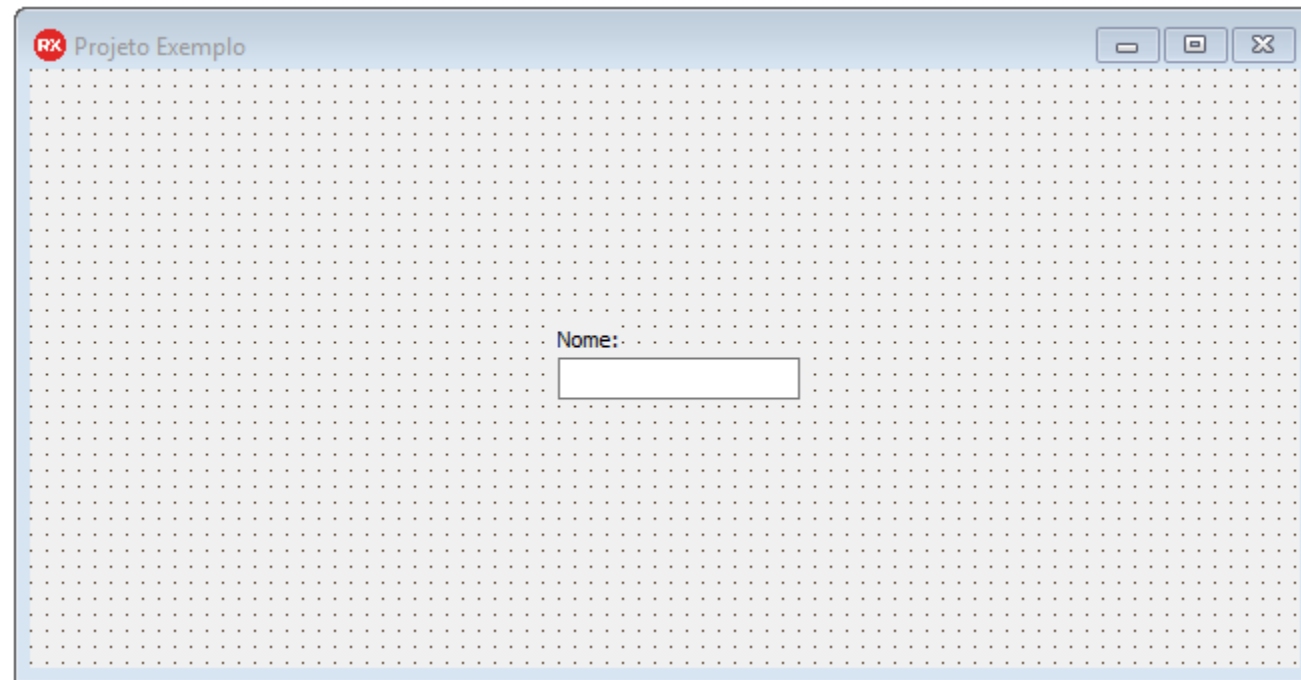
[11] TGroupBox (Container): Exemplo

```
procedure TForm18.BtnClick(Sender: TObject);  
begin  
  
    GrpBox.Caption := 'Grupo das Frutas';  
    GrpBox.Font.Size := 10;  
    GrpBox.Font.Color := clBlue;  
    GrpBox.TabOrder := 2;  
    GrpBox.Visible := TRUE;  
  
end;
```

Tratamento do evento do onClick do botão

Define o conteúdo do TGroupBox como o Caption e a cor e o tamanho da fonte

[12] TLabelEdit (Diálogo): Definição



Componente TForm com um TLabelEdit

[12] TLabeledEdit (Diálogo): Propriedades I

- **Align**: Define a posição do componente no TForm.
- **Text**: Permite manipular o conteúdo do TLabeledEdit.
- **TextHint**: Define um texto de dica para o componente (caso Text não existir).
- **Height**: Altura do TLabeledEdit em pixel.
- **Width**: Largura do TLabeledEdit em pixel.
- **ReadOnly**: Permite que o conteúdo do TLabeledEdit seja somente lido.
- **MaxLength**: Comprimento máximo do TLabeledEdit em caracteres.
- **PasswordChar**: Permite ocultar o texto com (*).
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.



[12] TLabelEdit (Diálogo): Propriedades II

- **Visible**: Torna o componente ocultável.
- **Name**: Troca o nome do TLabelEdit (Dentro do Código Delphi).
- **Color**: Define a cor de fundo de um TLabelEdit.
- **Hint**: Define uma caixa de dica para o componente.
- **ShowHint**: Habilita a dica.
- **Bevel**: (Inner/Kind/Outer): Permite estilizar as bordas do TLabelEdit.
- **CharCase**: Permite definir se as letras serão em CAPS ou não.
- **LabelSpacing**: Define o espaço do label com relação ao TLabelEdit.
- **NumbersOnly**: Permite aceitar apenas números.
- **Editlabel.Caption**: Define o rótulo do Label do Caption.
- **Editlabel.Font**: Define a fonte e a cor do Caption.

[12] TLabelEdit (Diálogo): Eventos

- **onChange**: Executa toda vez que o TLabelEdit for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TLabelEdit.
- **onExit**: Executa toda vez que o TAB sair dentro do TLabelEdit.
- **onClick**: Executa toda vez que um clique for feito no TLabelEdit.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TLabelEdit.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TLabelEdit.
- **onMouseLeave**: Executa toda vez que o mouse sair no TLabelEdit.

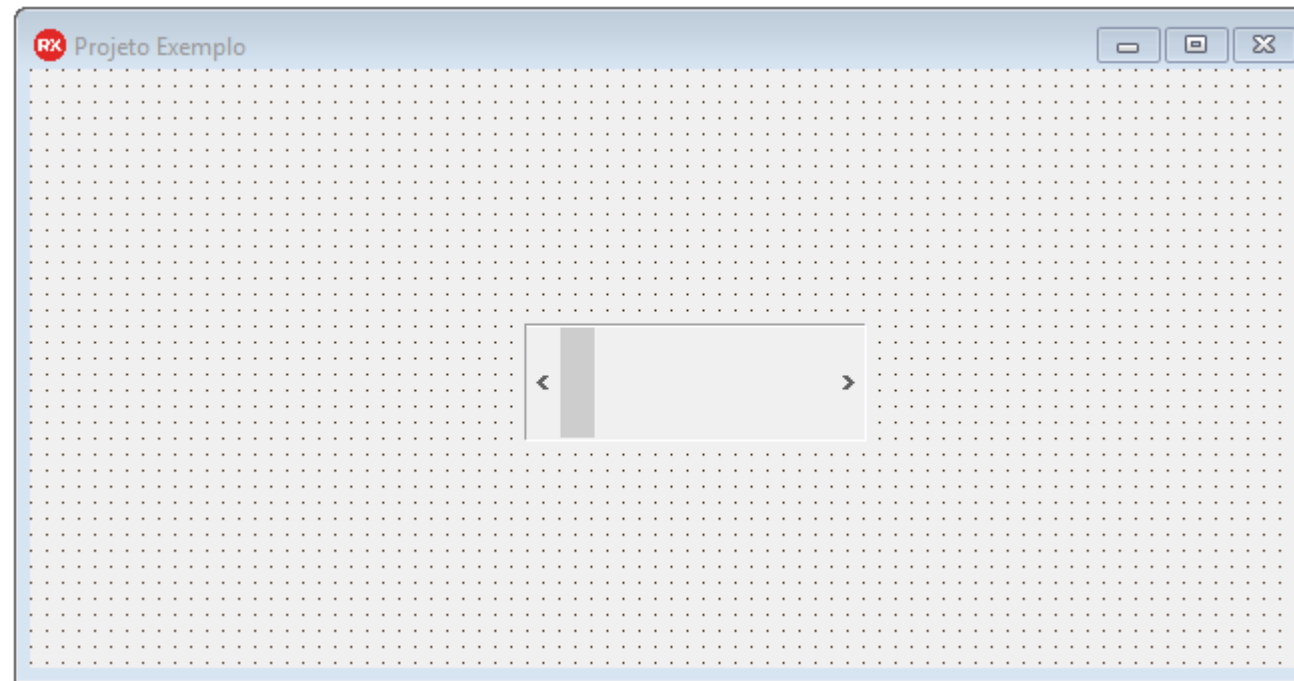


[12] TLabelEdit (Diálogo): Exemplo

```
procedure TForm18.BtnClick(Sender: TObject);  
begin  
  
    LblEdt.TextHint := 'Entre com um nome: ';  
    LblEdt.ShowHint := TRUE;  
    LblEdt.EditLabel.Caption := 'Nome';  
    LblEdt.EditLabel.Font.Color := clBlue;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do EditLabel mudando o TextHint o Caption e o Color

[13] TScrollBar (Rolagem): Definição



Componente TForm com um TScrollBar

[13] TScrollBar (Rolagem): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Visible**: Torna o componente ocultável
- **TabOrder**: Define a ordem que o componente será selecionado pelo TAB.
- **Enabled**: Habilita o acesso ao componente.
- **Hint**: Define uma dica para o ScrollBar.
- **Kind**: Define se a barra de rolagem será horizontal ou vertical.
- **ShowHint**: Habilita a dica.
- **Min**: Intervalo mínimo do ScrollBar.
- **Max**: Intervalo máximo do ScrollBar.
- **PageSize**: Define o tamanho do passo do ScrollBar.
- **Position**: Posição atual do ScrollBar.

[13] TScrollBar (Rolagem): Eventos

- **onChange**: Executa toda vez que o TScrollBar for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TScrollBar.
- **onExit**: Executa toda vez que o TAB sair dentro do TScrollBar.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TScrollBar.
- **onMouseLeave**: Executa toda vez que o mouse sair no TScrollBar.

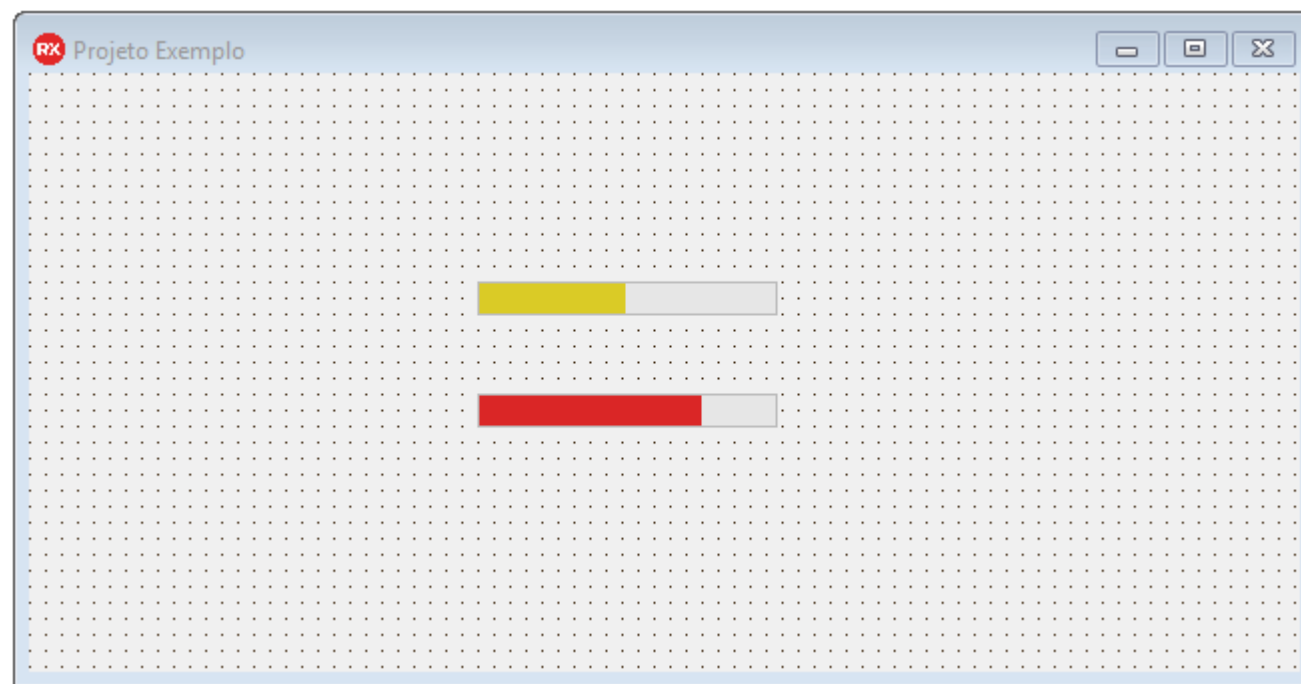


[13] TScrollBar (Rolagem): Exemplo

```
procedure TForm2.SclBarChange(Sender: TObject);  
begin  
  
    SclBar.position := 50;  
    SclBar.min := 1;  
    SclBar.max := 100;  
    SclBar.enabled := TRUE;  
  
end;
```

Tratamento do evento do onChange da barra de rolagem
Define o conteúdo da barra com min em 1 e max em 100 e com posição em 50

[14] TProgressBar (Status): Definição



Componente TForm com um TProgressBar

[14] TProgressBar (Status): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Visible**: Torna o componente ocultável
- **Enabled**: Habilita o acesso ao componente.
- **Hint**: Define uma dica para o TProgressBar.
- **Orientation**: Define se a barra de progresso será horizontal ou vertical.
- **ShowHint**: Habilita a dica.
- **Min**: Intervalo mínimo do TProgressBar.
- **Max**: Intervalo máximo do TProgressBar.
- **Position**: Posição atual do TProgressBar.
- **State**: Define se o estado é normal (verde) ou de erro (vermelho).



[14] TProgressBar (Status): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TProgressBar.
- **onExit**: Executa toda vez que o TAB sair dentro do TProgressBar.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TProgressBar.
- **onMouseLeave**: Executa toda vez que o mouse sair no TProgressBar.

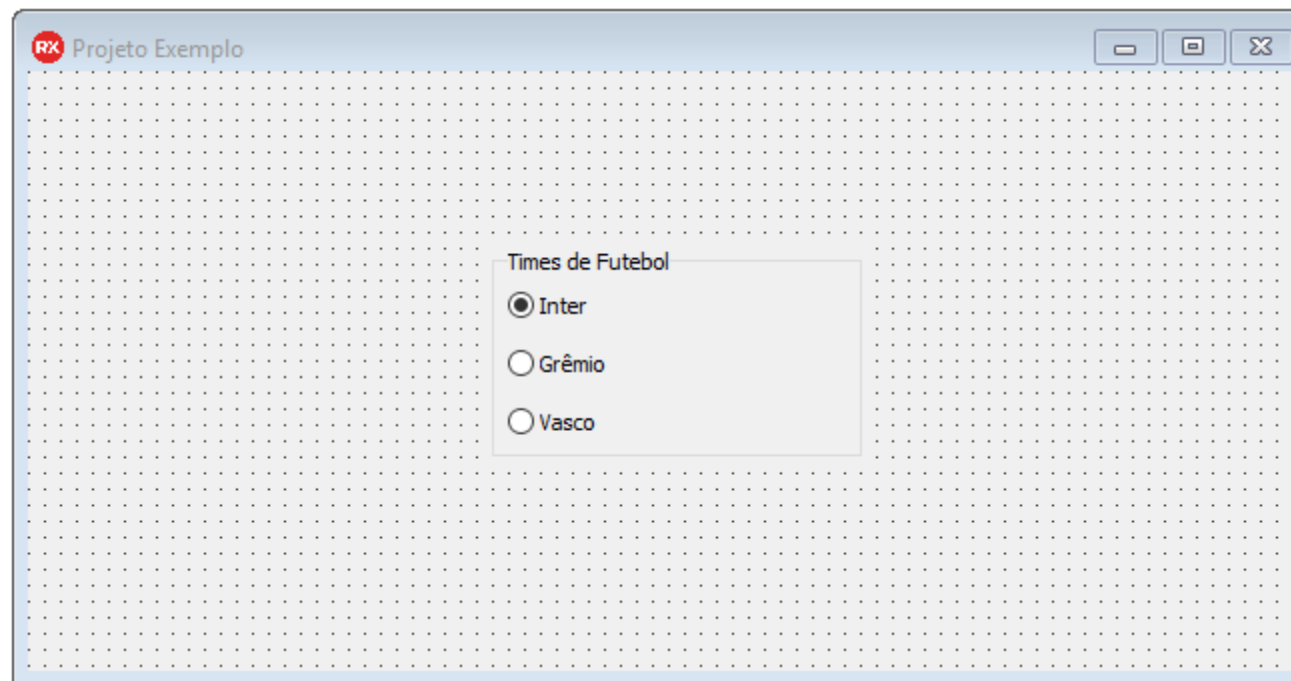


[14] TProgressBar (Status): Exemplo

```
procedure TForm2.BitBtn2Click(Sender: TObject);  
begin  
  
    ProgBar.position := 50;  
    ProgBar.min := 1;  
    ProgBar.max := 100;  
    ProgBar.enabled := TRUE;  
    ProgBar.State := pbsNormal;  
    ProgBar.Orientation := pbVertical;  
  
end;
```

Tratamento do evento do onClick do botão
Define o intervalo da barra de progresso, seu estado e a orientação

[15] TRadioGroup (Seleção): Definição



Componente TForm com um TRadioGroup

[15] TRadioGroup (Seleção): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o rótulo do RadioGroup.
- **Color**: Define a cor do RadioGroup.
- **Font**: Define a fonte e o tamanho do texto do RadioGroup.
- **Enabled**: Define se o botão será habilitado.
- **TabOrder**: Define a ordem de chamada do TAB no RadioGroup.
- **ItemIndex**: Define qual dos botões será marcado (zero é o primeiro).
- **Items**: Define os botões no RadioGroup.
- **Columns**: Define quantas colunas o deve ter RadioGroup.
- **Hint**: Define uma dica para o RadioGroup.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.

[15] TRadioGroup (Seleção): Eventos



- **onClick**: Executa toda vez que um click for feito na TRadioGroup.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TRadioGroup
- **onExit**: Executa toda vez que o TAB sair dentro do TRadioGroup.

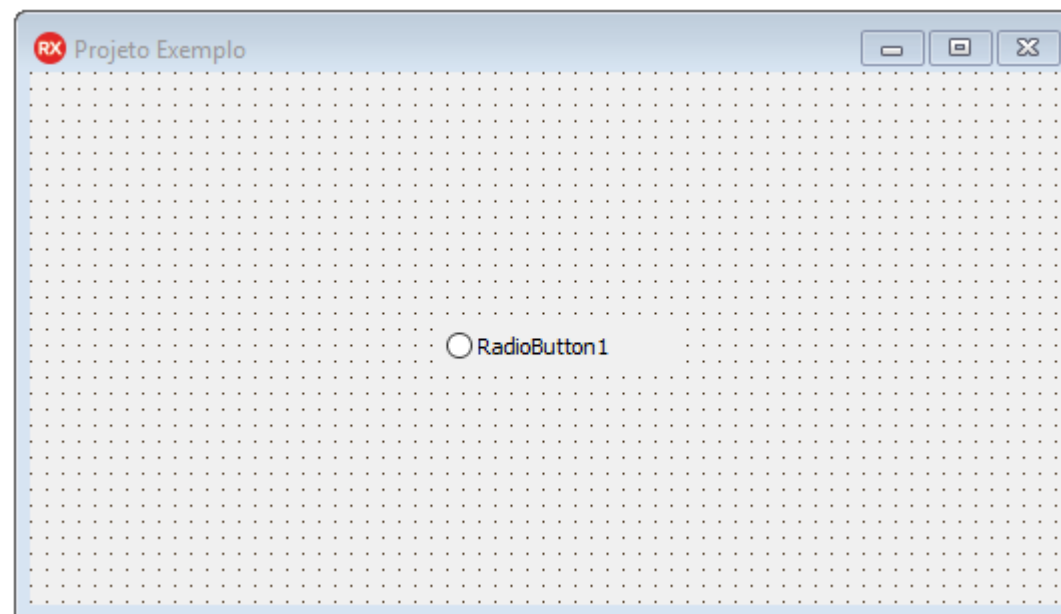


[15] TRadioGroup (Seleção): Exemplo

```
procedure TForm2.RdGrpClick(Sender: TObject);  
begin  
  
    RdGrp.Caption := 'Times de Futebol!';  
  
    if RdGrp.ItemIndex = 0 then  
    begin  
        showmessage('Time Escolhido: ', RdGrp.Items[RdGrp.ItemIndex]);  
    end  
    else if RdGrp.ItemIndex = 1 then  
    begin  
        showmessage('Time Escolhido: ', RdGrp.Items[RdGrp.ItemIndex]);  
    end;  
  
end;
```

Tratamento do evento do onClick do botão
Seleciona uma das opções do RadioGroup utilizando ItemIndex
Captura o conteúdo da posição ItemIndex utilizando Items

[16] TRadioButton (Seleção): Definição



Componente TForm com um TRadioButton

[16] TRadioButton (Seleção): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o rótulo do TRadioButton.
- **Font**: Define a fonte e o tamanho do texto do TRadioButton
- **Enabled**: Define se o TRadioButton será habilitado
- **TabOrder**: Define a ordem de chamada do TAB no TRadioButton.
- **Hint**: Define uma dica para o TRadioButton.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o TRadioButton será visível ou não.
- **Checked**: Permite marcar o TRadioButton.



[16] TRadioButton (Seleção): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TRadioButton.
- **onExit**: Executa toda vez que o TAB sair dentro do TRadioButton.
- **onClick**: Executa toda vez que um clique for feito no TRadioButton.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TRadioButton.
- **onMouseLeave**: Executa toda vez que o mouse sair no TRadioButton.

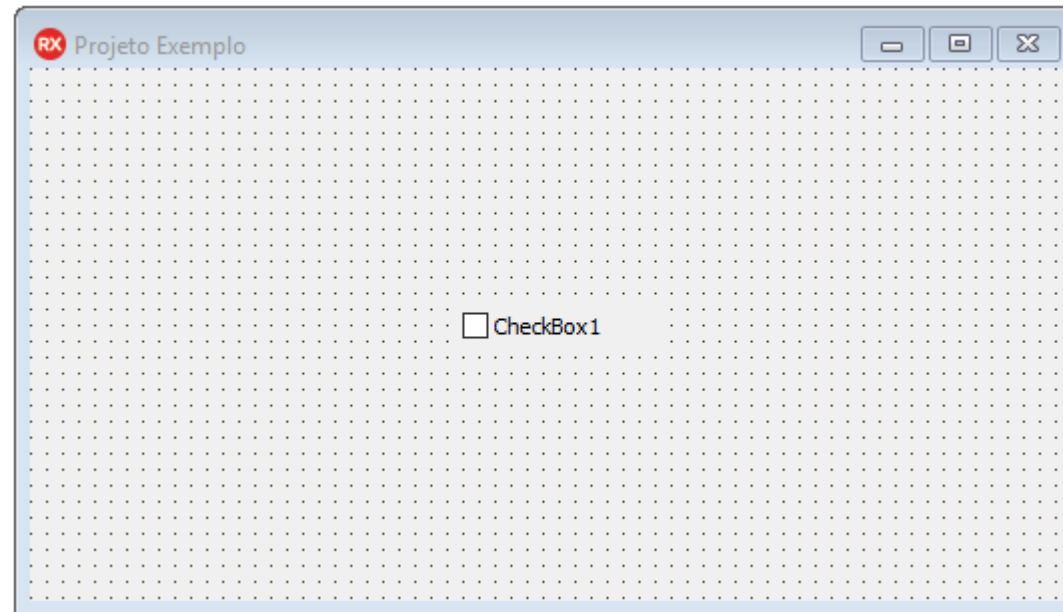


[16] TRadioButton (Seleção): Exemplo

```
procedure TForm2.RdGrpClick(Sender: TObject);  
begin  
  
    RadioBut.Caption := 'Devo Comparar?';  
  
    if RadioBut.Checked = TRUE then  
    begin  
        // Bloco que implementa o cálculo  
    end;  
  
end;
```

Tratamento do evento do onClick do TRadioGroup
Define o conteúdo do rótulo do TRadioButton e verifica se o mesmo foi marcado

[17] TCheckBox (Seleção): Definição



Componente TForm com um TCheckBox

[17] TCheckBox (Seleção): Propriedades

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o rótulo do TCheckBox.
- **Font**: Define a fonte e o tamanho do texto do TCheckBox
- **Enabled**: Define se o TCheckBox será habilitado
- **TabOrder**: Define a ordem de chamada do TAB no TCheckBox.
- **Hint**: Define uma dica para o TCheckBox.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o TCheckBox será visível ou não.
- **State**: Permite modificar o tipo de checagem.
- **Checked**: Permite marcar o TCheckBox.



[17] TCheckBox (Seleção): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TCheckBox.
- **onExit**: Executa toda vez que o TAB sair dentro do TCheckBox.
- **onClick**: Executa toda vez que um clique for feito no TCheckBox.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TCheckBox.
- **onMouseLeave**: Executa toda vez que o mouse sair no TCheckBox.

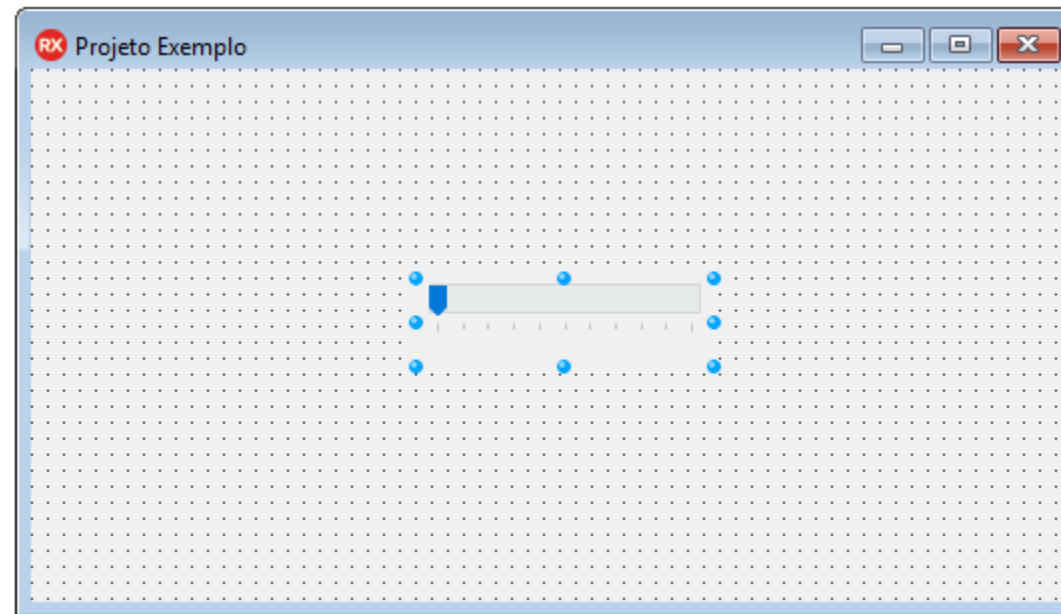


[17] TCheckBox (Seleção): Exemplo

```
procedure TForm2.RdGrpClick(Sender: TObject);  
begin  
  
    ChkBox.Caption := 'Devo Comparar?';  
  
    if ChkBox.Checked = TRUE then  
    begin  
        // Bloco que implementa o cálculo  
    end;  
  
end;
```

Tratamento do evento do onClick do botão
Define o conteúdo do rótulo do TCheckBox e verifica se o mesmo foi marcado

[18] TTrackBar (Rolagem): Definição



Componente TForm com um TTrackBar

[18] TTrackBar (Rolagem): Propriedades I

- **Align**: Define a posição do componente no TForm.
- **Name**: Define o nome do componente na Unit.
- **Font**: Define a fonte e o tamanho do texto do botão.
- **Enabled**: Define se o botão será habilitado.
- **Frequency**: Define a frequência da barra de rolagem.
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Min**: Intervalo mínimo do TTrackBar.
- **Max**: Intervalo máximo do TTrackBar.
- **Position**: Posição atual do TTrackBar.



[18] TTrackBar (Rolagem): Propriedades II

- **BorderWidth**: Define uma borda ao redor do componente.
- **SelStart**: Define a posição inicial do seletor.
- **ShowSelRange**: Muda o tipo de seletor.
- **SliderVisible**: Permite ocultar a chave de seleção.
- **TickMarks**: Permite inverter a barra de rolagem.
- **TabOrder**: Define a ordem de chamada do TAB no botão.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.
- **Orientation**: Define a orientação do TTrackBar (Barra de rolagem).



[18] TTrackBar (Rolagem): Eventos

- **onChange**: Executa toda vez que o TTrackBar for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TTrackBar.
- **onExit**: Executa toda vez que o TAB sair dentro do TTrackBar.
- **onClick**: Executa toda vez que um clique for feito no TTrackBar.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TTrackBar.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TTrackBar.
- **onMouseLeave**: Executa toda vez que o mouse sair no TTrackBar.



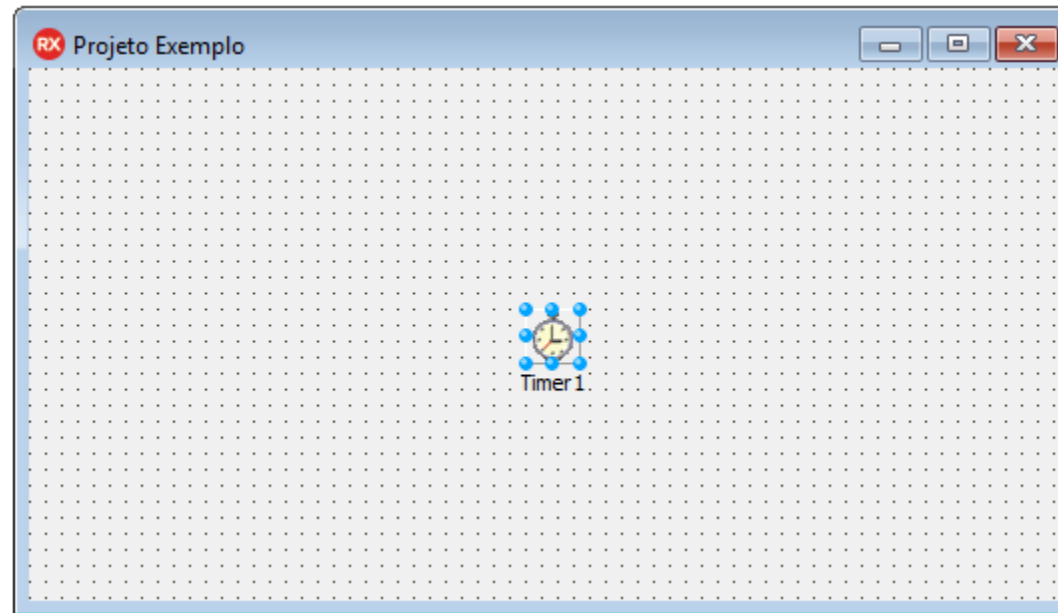
[18] TTrackBar (Rolagem): Exemplo

```
procedure TForm18.BtnClick(Sender: TObject);  
begin  
  
    TrkBar.Min := 0;  
    TrkBar.Max := 100;  
    TrkBar.Frequency := 10;  
    TrkBar.SelStart := 0;  
    TrkBar.SliderVisible := TRUE;  
  
end;
```

Tratamento do evento do onClick do botão

Define o conteúdo do TTrackBar com min em 0 e max em 100 com frequência de 10

[19] TTimer (Temporizador): Definição



Componente TForm com um TTimer

[19] TTimer (Temporizador): Propriedades



- **Name:** Define o nome do componente na Unit.
- **Interval:** Define o intervalo de tempo do Timer.
- **Enabled:** Define se o Timer será habilitado



[19] TTimer (Temporizador): Eventos



- **onTimer**: Executa toda vez que o interval zerar.



[19] TTimer (Temporizador): Exemplo

```
procedure TForm1.RelogioTimer(Sender: TObject);  
begin  
    FrmMenu.Color := clBlue;  
end;
```

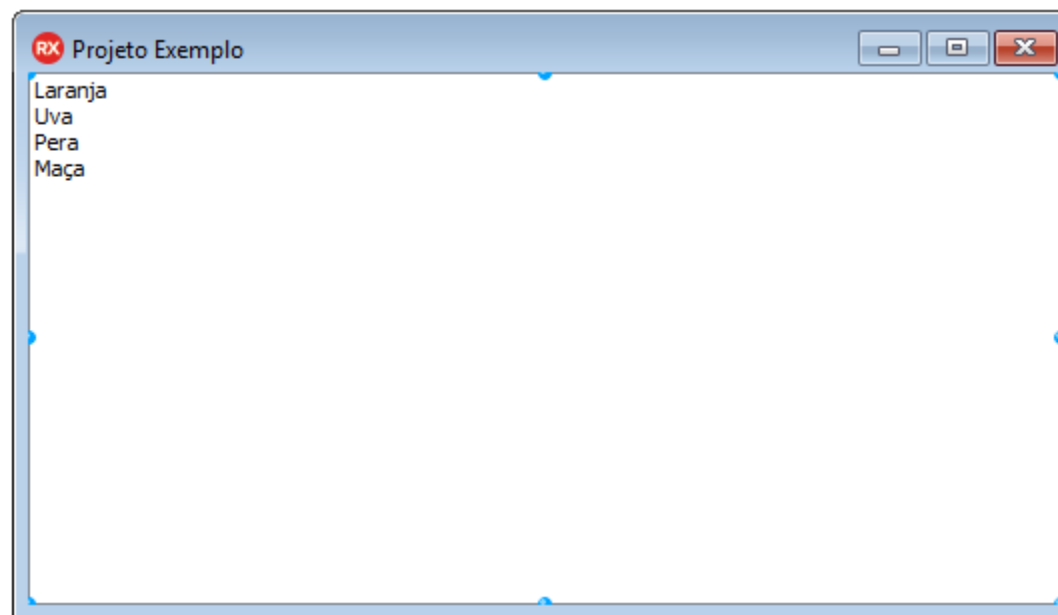
Tratamento do evento do onTimer do Relógio
Atualiza a cor do formulário.

Componentes VCL (III Trimestre)

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Componentes VCL (III Trimestre)



[20] TListBox (Lista): Definição



Componente TForm com um TListBox

[20] TListBox (Lista): Propriedades I

- **Align**: Define a posição da lista no TForm.
- **Name**: Define o nome do componente na Unit.
- **Color**: Define a cor interna do componente.
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Font**: Define a fonte e o tamanho da lista.
- **Enabled**: Define se a lista será habilitada.
- **TabOrder**: Define a ordem de chamada do TAB no botão.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.



[20] TListBox (Lista): Propriedades II

- **Items**: Permite definir valores para a lista.
- **ItemIndex**: Informa qual item foi selecionado (lista começa em zero).
- **lista.Items.add(item:string)**: Permite adicionar um valor na lista (sempre no final).
- **lista.Items.Insert(indice:integer, item:string)**: Permite adicionar um valor em um índice específico.
- **lista.Items.Delete(indice:integer)**: Permite deletar um valor em um índice específico.
- **lista.Items.Move(PosCorrente:integer, NovaPos:integer)**: Move os itens.
- **lista.Items.Count**: Retorna a quantidade de itens presentes dentro da lista.
- **lista.Clear**: Limpa toda a lista.
- **lista.Items[I]**: Captura o conteúdo do índice I da lista.



[20] TListBox (Lista): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TListBox.
- **onExit**: Executa toda vez que o TAB sair dentro do TListBox.
- **onClick**: Executa toda vez que um clique for feito no TListBox.
- **onData**: Executa toda vez que um dado for selecionado no TListBox.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TListBox.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TListBox.
- **onMouseLeave**: Executa toda vez que o mouse sair no TListBox.



[20] TListBox (Lista): Exemplo

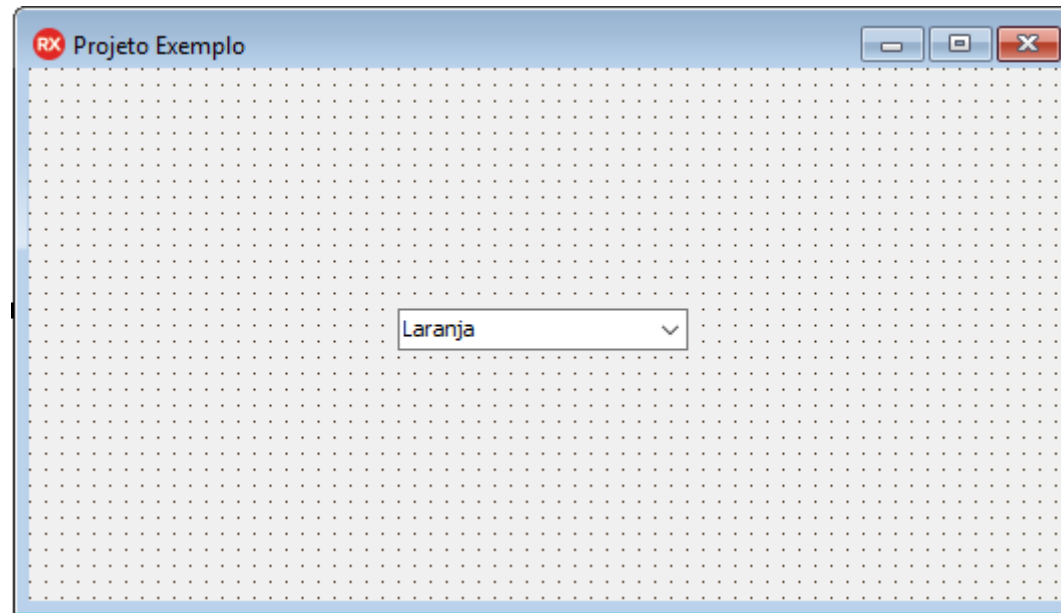
```
var
  I: Integer;
begin
  Lista.Clear;

  // Adiciona 10 números a lista
  FOR I := 1 TO 10 DO
  BEGIN
    Lista.Items.Add(inttostr(I));
  END;

  // Mostra o conteúdo da lista
  I := 0;
  WHILE (I < Lista.Items.Count) DO
  BEGIN
    showmessage(Lista.Items[I]);
    I := I + 1;
  END;
end;
```

Adiciona 10 valores na lista
Consulta os elementos da lista e mostra na tela

[21] TComboBox (Lista): Definição



Componente TForm com um TComboBox

[21] TComboBox (Lista): Propriedades

- **Align**: Define a posição da lista no TForm.
- **Name**: Define o nome do componente na Unit.
- **Color**: Define a cor interna do componente.
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Font**: Define a fonte e o tamanho da lista.
- **Enabled**: Define se a lista será habilitada.
- **TabOrder**: Define a ordem de chamada do TAB no botão.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.
- **Style**: Define o estilo do ComboBox.
- **DropDownCount**: Define o tamanho da lista do ComboBox.
- **Sorted**: Ordena em ordem alfabética os itens.

[21] TComboBox (Lista): Propriedades

- **Items**: Permite definir valores para a lista.
- **ItemIndex**: Informa qual item foi selecionado (lista começa em zero).
- **cbox.Items.add(item:string)**: Permite adicionar um valor na lista (sempre no final).
- **cbox.Items.Insert(indice:integer, item:string)**: Permite adicionar um valor em um índice específico.
- **cbox.Items.Delete(indice:integer)**: Permite deletar um valor em um índice específico.
- **cbox.Items.Move(PosCorrente:integer, NovaPos:integer)**: Move os itens.
- **cbox.Items.Count**: Retorna a quantidade de itens presentes dentro da lista.
- **cbox.Clear**: Limpa toda a lista.
- **cbox.Items[I]**: Captura o conteúdo do índice I da lista.



[21] TComboBox (Lista): Eventos

- **onEnter**: Executa toda vez que o TAB entrar dentro do TComboBox.
- **onExit**: Executa toda vez que o TAB sair dentro do TComboBox.
- **onClick**: Executa toda vez que um clique for feito no TComboBox.
- **onSelect**: Executa toda vez que um valor for selecionado no TComboBox.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TComboBox.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TComboBox.
- **onMouseLeave**: Executa toda vez que o mouse sair no TComboBox.

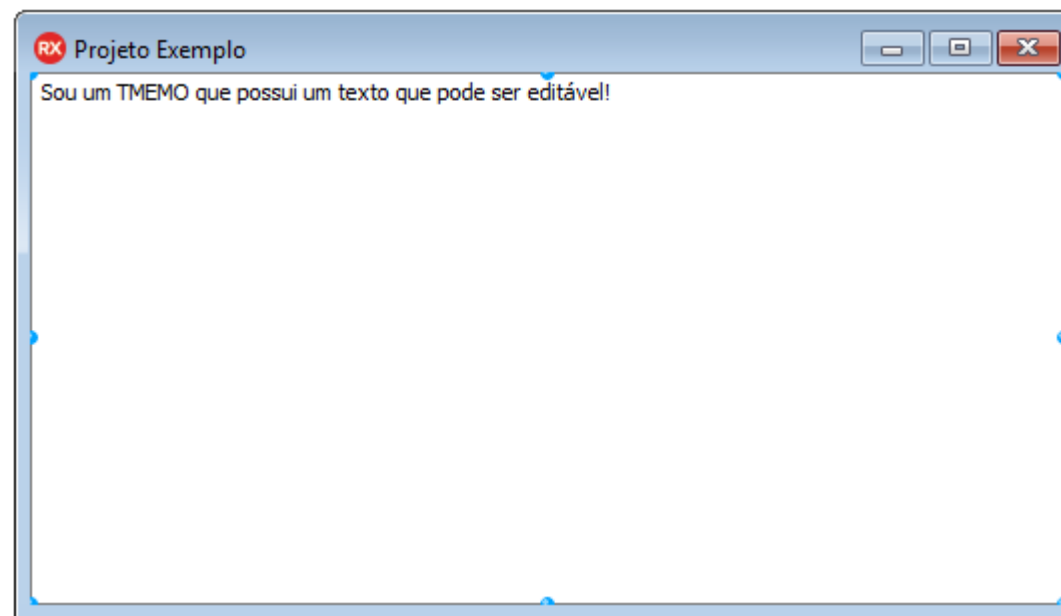


[21] TComboBox (Lista): Exemplo

```
procedure TFrmPrincipal.ComboSelect(Sender: TObject);  
begin  
    showmessage('Selecionou : ' + combo.Items[combo.ItemIndex]);  
end;
```

Tratamento do evento do onSelect do ComboBox
Captura o conteúdo de Items utilizando o ItemIndex selecionado

[22] TMemo (Lista): Definição



Componente TForm com um TMemo

[22] TMemo (Lista): Propriedades I

- **Align**: Define a posição do Memo no TForm.
- **Name**: Define o nome do componente na Unit.
- **CharCase**: Define se o conteúdo será maiúsculo ou minúsculo.
- **Font**: Define a fonte e o tamanho do texto do botão
- **Enabled**: Define se o botão será habilitado
- **TabOrder**: Define a ordem de chamada do TAB no botão.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **Visible**: Define se o botão será visível ou não.
- **Lines**: Define as linhas do Memo.
- **ReadOnly**: Define se as linhas do Memo serão de somente leitura.
- **ScrollBar**: Define barras de rolagem verticais e horizontais para o Memo.
- **WordWrap**: Define se as linhas terão quebra de linha.

[22] TMemo (Lista): Propriedades II

- **Items**: Permite definir valores para a lista.
- **ItemIndex**: Informa qual item foi selecionado (memo começa em zero).
- **memo.Items.add(item:string)**: Permite adicionar um valor na lista (sempre no final).
- **memo.Items.Insert(indice:integer, item:string)**: Permite adicionar um valor em um índice específico.
- **memo.Items.Delete(indice:integer)**: Permite deletar um valor em um índice específico.
- **memo.Items.Move(PosCorrente:integer, NovaPos:integer)**: Move os itens.
- **memo.Items.Count**: Retorna a quantidade de itens presentes dentro da lista.
- **memo.Clear**: Limpa toda a lista.
- **memo.Items[I]**: Captura o conteúdo do índice I da lista.



[22] TMemo (Lista): Eventos

- **onChange**: Executa toda vez que o TMemo for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TMemo.
- **onExit**: Executa toda vez que o TAB sair dentro do TMemo.
- **onClick**: Executa toda vez que um clique for feito no TMemo.
- **onDbClick**: Executa toda vez que dois clicks forem feitos no TMemo.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TMemo.
- **onMouseLeave**: Executa toda vez que o mouse sair no TMemo.

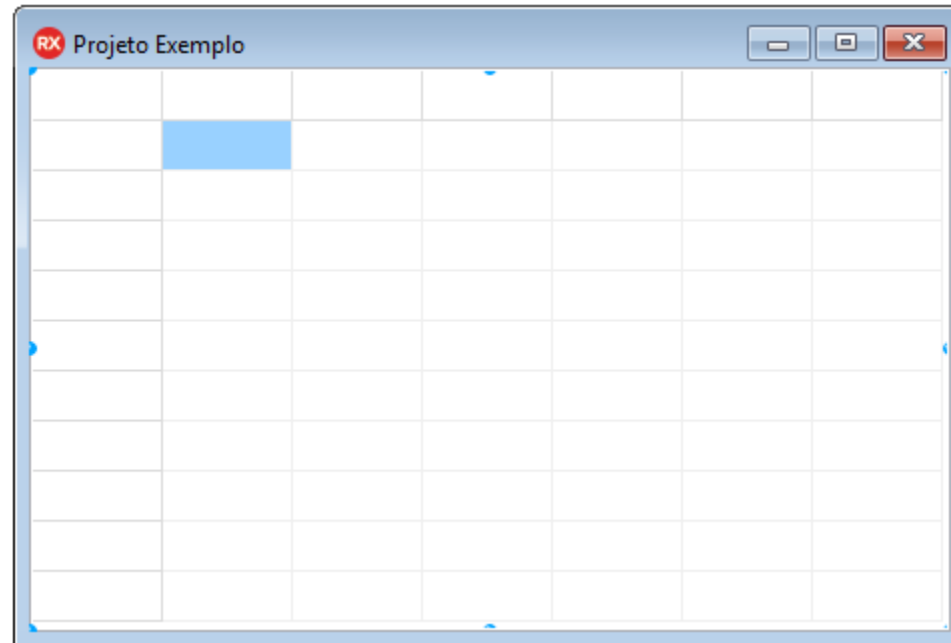


[22] TMemo (Lista): Exemplo

```
var  
    I: Integer;  
begin  
  
    Memo.Clear;  
    FOR I := 0 TO 10 DO  
    BEGIN  
        Memo.Lines.Insert(I, IntToStr(I));  
    END;
```

Inserir valores no Memo

[23] TStringGrid (Grade): Definição



Componente TStringGrid

[23] TStringGrid (Grade): Propriedades I

- **Align**: Define a posição da grade no TForm.
- **RowCount**: Define a quantidade de linhas da grade.
- **ColCount**: Define a quantidade de colunas da grade.
- **FixedCols**: Define quantas colunas serão fixas (Começa na zero).
- **FixedRows**: Define quantas linhas serão fixas (Começa na zero).
- **ScrollBars**: Define as barras de rolagem da grade.
- **Visible**: Define se o componente estará visível ou não.
- **Name**: Define o nome do componente na Unit.
- **TabOrder**: Define a ordem de chamada do TAB da grade.



[23] TStringGrid (Grade): Propriedades II

- **Cells**: Define uma posição [Col,Row] a ser manipulada.
- **Row**: Define a linha selecionada.
- **Col**: Define a coluna selecionada.
- **DefaultRowHeight**: Define a altura da linha.
- **DefaultColWidth**: Define a largura da coluna.
- **Height**: Define a altura da grade.
- **Width**: Define a largura da grade.
- **GridLineWidth**: Espessura das linhas da grade.



[23] TStringGrid (Grade): Eventos

- **onClick**: Executa toda vez que um click for feito na grade.
- **onDblClick**: Executa toda vez que um click duplo for feito na grade.
- **onMouseEnter**: Executa toda vez que o mouse entrar na grade.
- **onMouseLeave**: Executa toda vez que o mouse sair da grade.

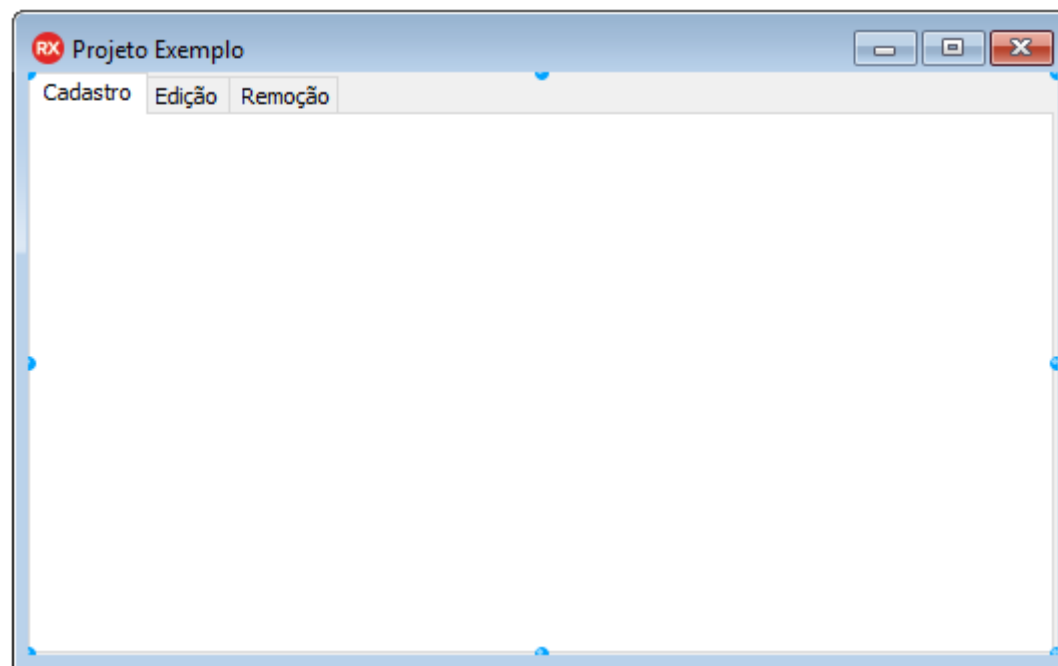


[23] TStringGrid (Grade): Exemplo

```
Grid.ColCount := 5;  
Grid.RowCount := 5;  
for I := 1 to 5 do  
  for J := 1 to 5 do  
    if I = J then  
      begin  
        Grid.Cells[J,I] := 'X';  
      end;  
    end;  
  end;  
end;
```

Cria uma matriz com 5 linhas e 5 colunas
Popula a diagonal principal com X

[24] TPageControl (Abas): Definição



Componente TForm com um TPageControl

[24] TPageControl (Abas): Propriedades

- **Align**: Define a posição do Page no TForm.
- **Name**: Define o nome do componente na Unit.
- **Caption**: Define o conteúdo da TAB.
- **Font**: Define a fonte e o tamanho do texto do botão
- **Enabled**: Define se o botão será habilitado
- **Width**: Define a largura do componente.
- **Height**: Define a altura do componente.
- **Hint**: Define uma dica para o botão.
- **ShowHint**: Define se a dica será mostrada ou não.
- **TabVisible**: Define se a TAB será visível ou não.
- **Style**: Define o estilo visual das TABs.



[24] TPageControl (Abas): Eventos

- **onChange**: Executa toda vez que o TPageControl for modificado.
- **onEnter**: Executa toda vez que o TAB entrar dentro do TPageControl.
- **onExit**: Executa toda vez que o TAB sair dentro do TPageControl.
- **onShow**: Executa toda vez que show for selecionado.
- **onMouseEnter**: Executa toda vez que o mouse entrar no TPageControl.
- **onMouseLeave**: Executa toda vez que o mouse sair no TPageControl.



[24] TPageControl (Abas): Exemplo

```
procedure TFrmPrincipal.ComboChange(Sender: TObject);  
begin  
  
    Cadastro.Show;  
    Cadastro.Caption:= 'Cadastro';  
  
end;
```

Tratamento do evento do onChange do ComboBox
Mostra a TabSheet Cadastro e atualiza o Caption

Revisão da Lógica de Programação

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Revisão da Lógica de Programação



Acrônimos para um projeto

- Projeto de uma Agenda de Contatos:
 - Nome do diretório do projeto: **Agenda**
 - Nome do projeto: **PrjAgenda**
 - Nome do Formulário Principal: **FrmPrincipal**
 - Nome da Unit do Formulário Principal: **UntPrincipal**



Estrutura de programa DELPHI

- **unit**
 - Local que define o nome do Unt do Form.
- **interface**
 - Não utilizada pelos programadores na disciplina.
- **uses**
 - Local onde são chamadas as bibliotecas e os formulários do projeto.
- **type**
 - Local onde são declarados os componentes VCL e os procedimentos dos eventos.
- **var**
 - Local onde as variáveis globais são declaradas.
- **implementation**
 - Local onde são implementados os procedures dos eventos



Variáveis e Tipos de Dados

- O que é uma **variável**?
 - É um **espaço** na **memória** do programa (RAM/Volátil) que **guarda informação**.
- O que possui uma **variável**?
 - Variáveis possuem (**Identificador** | **Tipo** | **Valor**).
- O que faz (define) o identificador?
 - **Identifica** a variável, permitindo que a mesma seja **chamada** (utilizada).
- O que faz (define) o **tipo**?
 - Define o **tipo** do **conteúdo** que a **variável** é capaz de **armazenar**.



Identificadores

- O identificador "identifica" a variável com um nome.
- Nomes precisam ser únicos (nunca utilize duas variáveis com mesmo nome).
- Nunca utilize caracteres especiais (!@#\$%^&*) em nomes de variáveis.
- Variáveis devem possuir nomes mudos como (cao, pao).
- Variáveis devem começar com letras (nunca com números).
- Utilize o padrão CamelCase onde nomes compostos seguirão a sintaxe:
 - nomePessoa : STRING
 - salarioMinimo : REAL
 - idadePessoa : INTEGER

Tipos de Dados (Nativos)

- 1. Integer:
 - Permite representar números inteiros como 2 ou 200.
 - Onde é utilizado: Em programas de física e matemática básicos.
- 2. Real
 - Permite representar números não inteiros como 2.2 ou 5.5.
 - Onde é utilizado: Em programas de física e matemática mais avançados.
- 3. String
 - Permite representar caracteres como 'c' ou 'A', palavras como 'Raffael' ou frases como 'Olá Mundo'
 - Onde é utilizado: Em programas que armazenam nomes e palavras.
- 4. Boolean
 - Permite armazenar dois estados lógicos, podendo ser eles TRUE ou FALSE.
 - Onde é utilizado: Por várias estruturas do DELPHI que realizam operações lógicas.



Operador de atribuição :=



- Observe como que o **operador** de **atribuição** (:=) é utilizado para **atribuir** um **valor** a uma **variável**.
- Lembre que o **valor** da **variável** pode ser **modificado** sempre que **necessário** pelo **programa**.
 - nomePessoa := 'Raffael Bottoli Schemmer';
 - salarioMinimo : '933,25';
 - idadePessoa : '32';



Constantes



- Utilizamos **constantes** para **guardar valores absolutos** como o raio de uma circunferência (PI).
- **Declarando e inicializando** constantes:
 - **const** PI = 3.14;
 - **const** JANEIRO = 31;
- Observe como **constantes** devem ser **declaradas e inicializadas**.
- Observe como o **operador** que **atribui** o **valor** a **constante** deve ser (=) e não (:=).
- Observe que o **valor** da **constante NUNCA** vai **mudar** OU **poderá** ser **mudado** durante a **execução**.



Escopo de Visualização (Global)

- O escopo **define** em quais **blocos** de **código** as **variáveis** são **acessíveis**.
- No escopo **global**, as **variáveis** podem ser **acessíveis** por **todos** os **blocos** do Form.
- O **programador** deve **declarar** as **variáveis globais** na clausula **var**:
 - Que fica entre **type** e **implementation**.

```
var  
  
    FrmMenu: TForm1;  
    Contador : Integer;  
  
implementation  
  
{$R *.dfm}
```

Declaração da variável global Contador do tipo Integer

Escopo de Visualização (Local)

- No escopo **local**, as variáveis podem ser **acessíveis** apenas pelo **procedure**.
- O **programador** deve **declarar** as **variáveis** utilizando **var**:
 - Entre **procedure** e **begin**.

```
procedure TForm1.ChamaFormClick(Sender: TObject);  
  
    var Contador : Integer;  
  
begin  
  
end;
```

Declaração da variável local Contador do tipo Integer

Aplicações das Variáveis

- 1. Acumuladoras:
 - Para acumular um valor, uma variável deve somar o seu valor atual, mais o novo valor.
 - A operação deve ser feita utilizando a variável + o valor a ser acumulado.
 - Exemplo: `salarioPessoa := salarioPessoa + 20;`
- 2. Contadoras:
 - Variáveis contadoras utilizam o mesmo princípio das acumuladoras.
 - Contam sempre um mesmo valor. São muito utilizadas em laços de repetição, vetores e matrizes.
 - Exemplo: `contadorLaco := contadorLaco + 1;`



Aplicações das Variáveis

- 3. Auxiliares:
 - Muitas vezes será necessário **trocar** o **valor** de **duas variáveis**.
 - Por exemplo, o problema pede que a variável A receba o valor de B e que B receba o valor de A.
 - O grande problema é como trocar o conteúdo, pois se $A := B$ o conteúdo de A automaticamente será substituído.
 - Nestes casos, o programador deve fazer uso de uma **terceira variável AUX** para **intermediar** a **troca**.
 - Veja que AUX é utilizada como uma **variável auxiliar** "apenas" para realizar a troca.
- O processo seria feito desta forma:
 - $AUX := A;$
 - $A := B;$
 - $B := AUX;$



Aplicações das Variáveis

- 4. Flags:
 - Em problemas (no futuro), vamos precisar percorrer longas **estruturas** de **dados** que armazenam valores.
 - Os problemas vão pedir para que determinados valores e padrões sejam **procurados** nos dados.
 - Nestes casos, as variáveis podem nos ajudar a guardar **estados** como (TRUE) ou (FALSE).
 - Antes de executar o código que busca por um padrão a **variável flag** (**sinalizador**) é marcada como FALSE.
 - Caso o código **encontre** o que o **programador** estiver programado, o flag será **modificado** para **TRUE**.
 - A verificação do **estado** do **flag** permite saber se um **valor** procurado ou não **está** na **estrutura**.



Atalhos do DELPHI



- Para declarar variáveis utilize o atalho:
 - 1. Digite o nome da variável.
 - 2. Com o cursor na frente do nome pressione **Ctrl + Shift + V**
- O atalho acima se aplica a variáveis locais (apenas).



Operadores Aritméticos

- A linguagem DELPHI suporta 6 operadores aritméticos sendo eles:
 - Adição : + (Exemplo: $A := A + B;$)
 - Subtração :: (Exemplo: $A := A - B;$)
 - Multiplicação : * (Exemplo: $A := A * B;$)
 - Divisão de ponto-flutuante : / (Exemplo: $A := A / B;$)
 - Divisão inteira : div (Exemplo: $A := A \text{ div } B;$)
 - Resto da divisão inteira : mod (Exemplo: $A := A \text{ mod } B;$)



Operadores Relacionais

- Operadores **relacionais** permitem realizar **comparações**.
- As **comparações** podem ser feitas com **constantes** e **variáveis**.
- Toda **comparação** retorna um valor **lógico** (TRUE ou FALSE).
- Toda **comparação relacional** precisa **SEMPRE** de **dois operandos**.
- O DELPHI suporta 6 operadores relacionais:
 - = Igualdade (Exemplo: **2 = 2** irá retornar TRUE)
 - <> Diferença (Exemplo: **2 <> 2** irá retornar FALSE)
 - < Menor que (Exemplo: **2 < 2** irá retornar FALSE)
 - > Maior que (Exemplo: **2 > 3** irá retornar FALSE)
 - <= Menor ou igual (Exemplo: **2 <= 3** irá retornar TRUE)
 - >= Maior ou igual (Exemplo: **3 >= 2** irá retornar TRUE)



Funções de Conversão de Tipos

- [IntToStr] Converte um número inteiro (X) em um string (STR).
- $STR := \text{IntToStr}(X);$
- [StrToInt] StrToInt Converte um string (X) em um número inteiro (INT).
- $INT := \text{StrToInt}(X);$
- [FloatToStr] FloatToStr Converte um valor de ponto flutuante (X) num string (STR).
- $STR := \text{FloatToStr}(X);$
- [StrToFloat] StrToFloat Converte um string (X) num valor de ponto flutuante (REAL).
- $FLOAT := \text{StrToFloat}(X);$



Operadores Lógicos

- 1. AND (Operador da Conjunção)
 - Pode ser utilizado em quantas variáveis (entradas) for necessário.
 - "E" Se os dois operandos são verdadeiros, o resultado será verdadeiro.
 - Se um dos dois operandos é falso, o resultado será falso.

Operador AND



```
var A,B : Integer;  
  
BEGIN  
    // Só irá executar se A e B forem > 0  
    IF ( (A > 0) AND (B > 0) ) THEN  
        BEGIN  
            // Lógica do programa!  
        END;  
    END;  
END;
```

A	AND	B	S
F		F	F
F		V	F
V		F	F
V		V	V

Operadores Lógicos

- 2. OR (Operador da Disjunção)
 - Pode ser utilizado em quantas variáveis (entradas) for necessário.
 - "OU" Se um dos operandos são verdadeiros, o resultado será verdadeiro.
 - Se os dois operandos são falsos, o resultado será falso.

Operador OR



```
var SENHA : Integer;  
  
BEGIN  
  
  // Só irá executar se SENHA for 123 ou 321  
  IF ( (SENHA = 123) OR (SENHA = 321) ) THEN  
  BEGIN  
  
    // Lógica do programa!  
  
  END;  
  
END;
```

A	OR	B	S
F		F	F
F		V	V
V		F	V
V		V	V

Operadores Lógicos

- 3. NOT (Operador da Negação)
 - Nega o estado (IF for FALSE retorna TRUE e/ou o contrário).

```
var cont : Integer;  
  
begin  
    cont := 1;  
    REPEAT  
        cont := cont + 1;  
    UNTIL NOT cont <= 5;  
end;
```

Operador NOT



NOT	A		S
	F		V
	V		F

Precedência dos Operadores

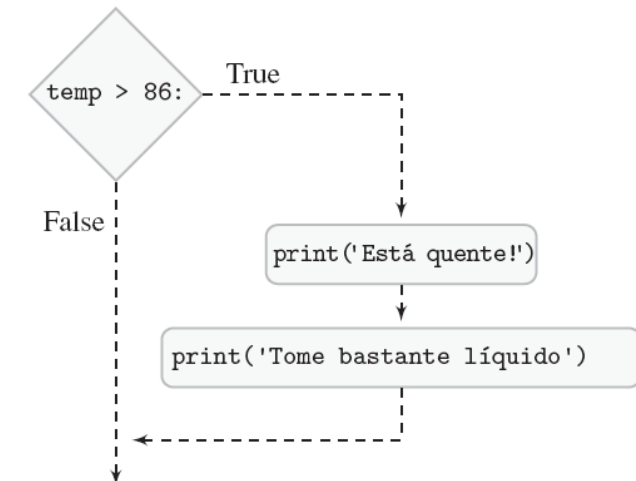
- 1. Parênteses mais **internos**
 - ()
- 2. Operadores **aritméticos**
 - * / div mod + -
- 3. Operadores **relacionais**
 - = < > > = < < =
- 4. Operadores **lógicos**
 - AND OR NOT

Estruturas Condicionais

- Permite realizar uma **verificação lógica** e **executa** o bloco de **código** (IF) se a **condição** for **TRUE**.

```
IF (mediaAluno >= 7.0) THEN  
BEGIN  
    SHOWMESSAGE ('Aluno Aprovado!');  
END;
```

Estrutura condicional simples IF

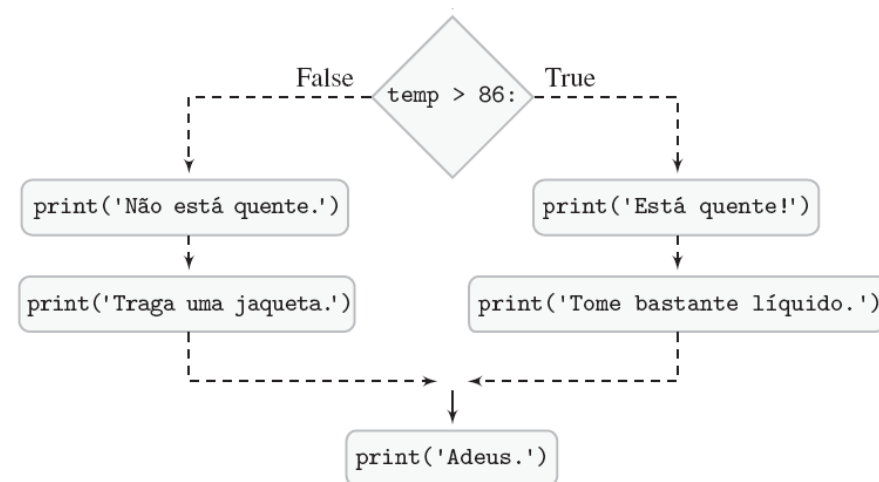


Estruturas Condicionais

- Permite realizar uma **verificação lógica** e executa o bloco de **código** (IF) se a **condição** for **TRUE**
 - Caso contrário irá **executar** o bloco (**ELSE**).
 - Estrutura **ELSE** é **opcional**.

```
IF (mediaAluno >= 7.0) THEN
BEGIN
    SHOWMESSAGE ('Aluno Aprovado!');
END
ELSE
BEGIN
    SHOWMESSAGE ('Aluno Não Aprovado!');
END;
```

Estrutura condicional composta IF/ELSE

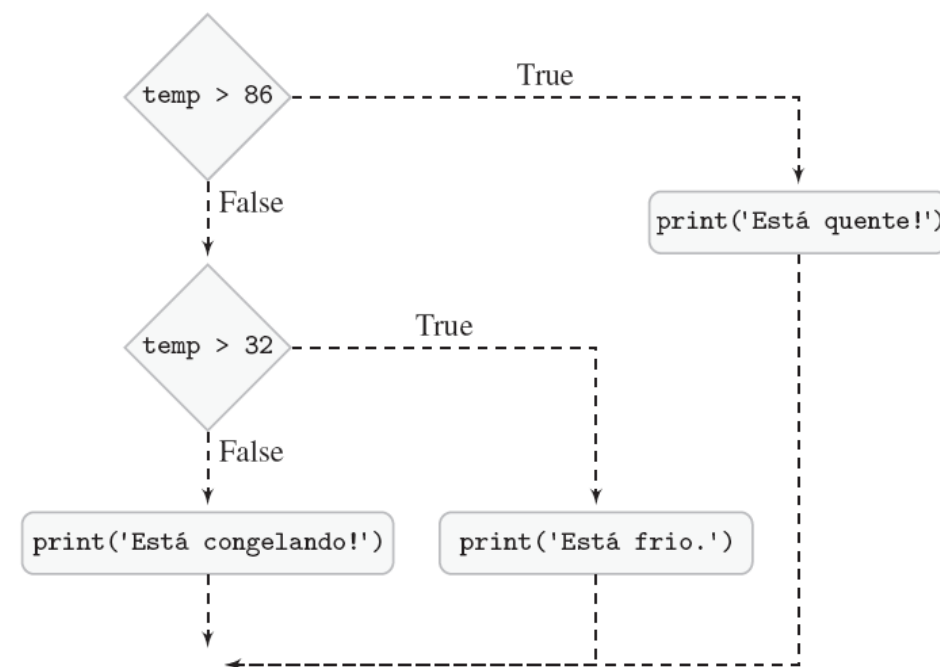


Estruturas Condicionais

- Permite realizar uma verificação lógica e executa o bloco de código (IF) se a condição for TRUE
 - Caso contrário, uma nova pergunta (ELSE IF) é realizada.
 - Estrutura **ELSE** é *opcional*.

```
IF (mediaAluno >= 7.0) THEN
BEGIN
    SHOWMESSAGE ('Aluno Aprovado!');
END
ELSE IF ((mediaAluno >= 5.0) AND (mediaAluno < 7.0)) THEN
BEGIN
    SHOWMESSAGE ('Aluno Em Exame!');
END
ELSE
BEGIN
    SHOWMESSAGE ('Aluno Reprovado!');
END;
END;
```

Estrutura condicional encadeada IF/ELSE IF/ELSE

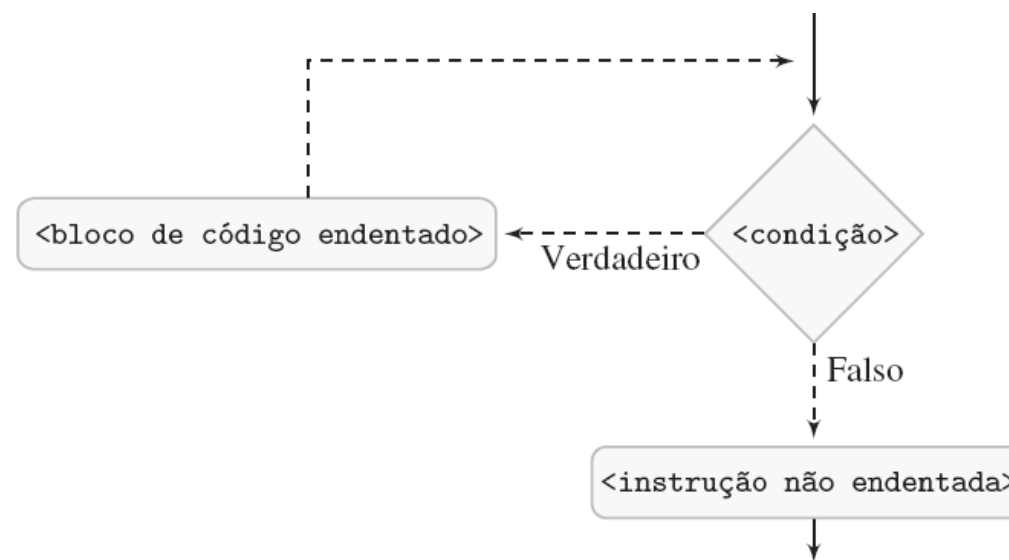


Estruturas Iterativas

- Laço while (Estrutura Pré-Testada):
 - Executa o bloco enquanto condição for TRUE.

```
var cont : Integer;  
begin  
    cont := 1;  
    WHILE (cont <= 10) DO  
    BEGIN  
        cont := cont + 1;  
    END;  
end;
```

Estrutura iterativa que executa 10 vezes

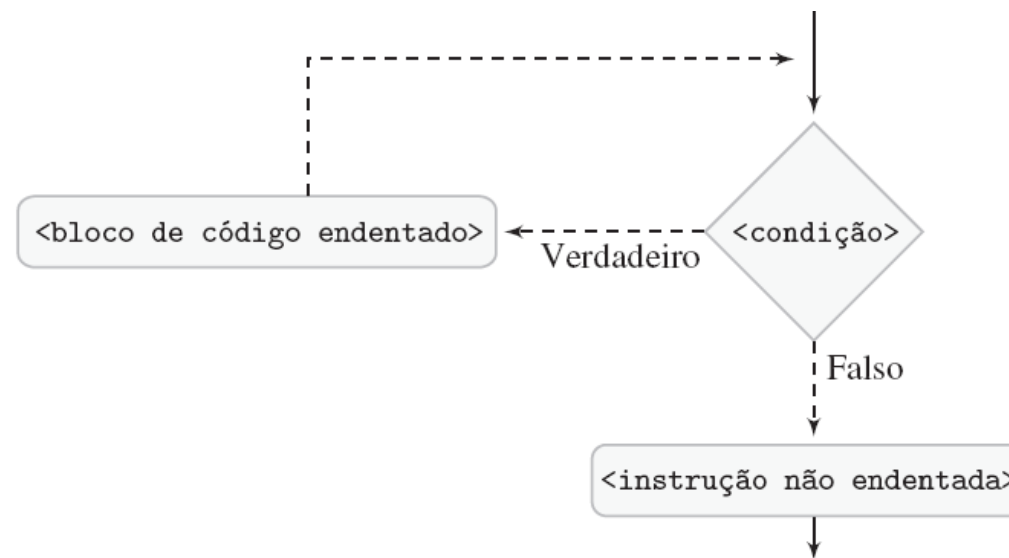


Estruturas Iterativas

- Laço for (Pré-Testada):
 - Executa o bloco dentro do intervalo de TO ou DOWNTO

```
var cont : Integer;  
  
begin  
    for cont := 1 TO 10 DO  
        BEGIN  
        END;  
    for cont := 10 DOWNTO 0 DO  
        BEGIN  
        END;  
end;
```

Estrutura iterativa que executa 10 vezes

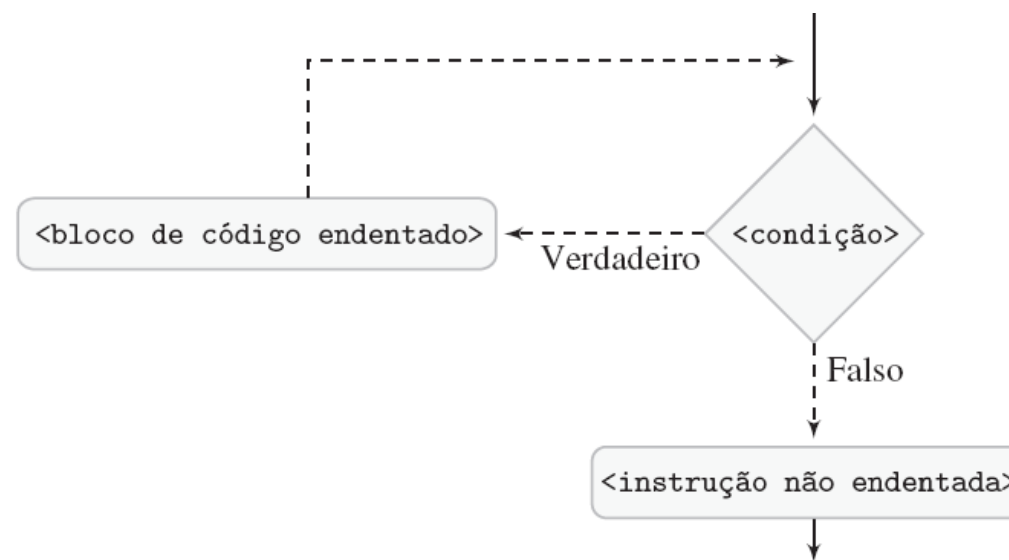


Estruturas Iterativas

- Laço repeat (Estrutura Pós-Testada):
 - Executa o bloco enquanto condição for FALSE.

```
var cont : Integer;  
  
begin  
    cont := 1;  
    REPEAT  
        cont := cont + 1;  
    UNTIL cont = 5;  
end;
```

Estrutura iterativa que executa 5 vezes



Atalhos do DELPHI

Ctrl + Space

O atalho acima deve ser utilizado SEMPRE
Ele permite completar os laços!



Vetores

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Vetores



O que é um Vetor?

- É uma **estrutura** de dados **Unidimensional**.
- É **utilizado** para **armazenar valores**.
- Possui **comportamento** similar as **variáveis**.
- Todo **vetor** possui **um único tipo de dado**.



Como Declarar um Vetor

- Utilize um dos 4 tipos de dados conhecidos.
- Vetores podem ser locais ou globais, igual as variáveis.
- Vetores possuem tamanhos estáticos (fixos).

```
VetorInteger: Array[1..100] of Integer;  
VetorReal: Array[1..100] of Real;  
VetorString: Array[1..100] of String;
```

Declaração de 3 vetores do tipo Integer, Real e String de 100 posições.

Funcionamento de um Vetor

Índice	1	2	3	4
Valor	55	43	12	24

Ilustração de um vetor com 4 posições (índices 1 até 4)

Como Utilizar um Vetor

Declarando o vetor



```
var  
  
    VI: Array[1..5] of Integer;  
    I: Integer;  
  
begin  
  
    for I := 1 to 5 do  
        VI[I] := RandomRange(20, 50);  
  
    for I := 1 to 5 do  
        GRID.Cells[I, 0] := IntToStr(VI[I]);
```

Acessando a posição
[I]



Inicializando valores aleatórios e escrevendo na GRID



Exercícios (Fácil)

Faça um programa que leia um **vetor** de **10 posições** de números **inteiros aleatórios (RandomRange)**. A seguir calcule o **quadrado** de cada elemento do **vetor lido**, armazenando o resultado num **segundo vetor**. Por fim, mostre o **segundo vetor** em um **TMEMO** e/ou em um **TSTRINGGRID**.



Exercícios (Difícil)



Faça um programa que leia **dois vetores** de **10 palavras** e **crie** um **terceiro vetor** que seja a **união** dos dois **vetores lidos**. Por fim, mostre a união em um **TMEMO** e/ou em um **TSTRINGGRID**.



O que fazer agora?

Lista de exercícios de Vetores



[GitHub.com/RaffaelSchemmer/A2](https://github.com/RaffaelSchemmer/A2)

Matrizes

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Matrizes



O que é uma Matriz?

- É uma **estrutura** de **dados Bidimensional**.
- É **utilizado** para **armazenar valores**.
- Possui **comportamento** similar as **variáveis**.
- Toda **matriz** possui **um único tipo de dado**.

Como Declarar uma Matriz

- Utilize um dos 4 tipos de dados conhecidos.
- Matrizes podem ser locais ou globais, igual as variáveis.
- Matrizes possuem tamanhos estáticos (fixos).

```
MatrizInteger: Array[1..5,1..5] of Integer;  
MatrizReal: Array[1..5,1..5] of Real;  
MatrizString: Array[1..5,1..5] of String;
```

Declaração de 3 matrizes do tipo Integer, Real e String de 25 posições.

Funcionamento de uma Matriz

Coluna

Linha

Linha/Coluna	1	2	3	4
1	55	43	12	24
2	23	12	28	32
3	54	18	45	63
4	65	44	13	54

Ilustração de uma matriz com 16 posições (índices 1,1 até 4,4)

Como Utilizar uma Matriz

```
var

  MI: Array [1 .. 5, 1 .. 5] of Integer;
  I, J : Integer;

begin

  // Popula matriz 5x5
  for I := 1 to 5 do
    for J := 1 to 5 do
      MI[I,J] := RandomRange(20,50);

  // Escreve matriz 5x5 no GRID
  for I := 1 to 5 do
    for J := 1 to 5 do
      GRID.Cells[J,I] := InttoStr(MI[I,J]);

end;
```

Inicializando e mostrando em um GRID uma matriz de 5 linhas por 5 colunas (25 posições)

Exercícios (Fácil)

Elabore um programa que leia duas matrizes 3X3 utilizando valores aleatórios. A seguir crie uma terceira matriz que seja a soma dos elementos das duas matrizes lidas. O resultado deve ser exibido em um TSTRINGGRID.

Exercícios (Difícil)

Faça um programa que leia uma matriz 3X4 utilizando números aleatórios e calcule e mostre a soma das linhas pares da matriz em um TMEMO.

O que fazer agora?

Lista de exercícios de Matrizes



[GitHub.com/RaffaelSchemmer/A2](https://github.com/RaffaelSchemmer/A2)

Procedimentos e Funções

Prof: Raffael Bottoli Schemmer
Disciplina: (LPI) Linguagem de Programação I
Ano: 2018.
Módulo: Procedimentos e Funções.



O que é um procedimento?

- Para o Delphi (Object Pascal):
 - Todos os eventos dos objetos visuais da VCL são procedimentos.
- O que é um procedimento?
 - É um subprograma (código) do programa principal (código).



Criando um procedimento

- Observe algumas particularidades de um procedure:
 - [1] Não retorna valor.
 - [2] O nome do procedure deve ser único no programa.
 - [3] Não utilize nomes de procedures iguais a palavras reservadas.
 - [4] A primeira linha do código deve (SEMPRE) terminar com ponto e vírgula.
 - [5] Variáveis locais devem ser utilizadas (APENAS) no procedimento.
 - [6] Todo código do procedimento deve estar entre BEGIN e END;
 - [7] Na última linha do procedimento o comando END deve possuir ;

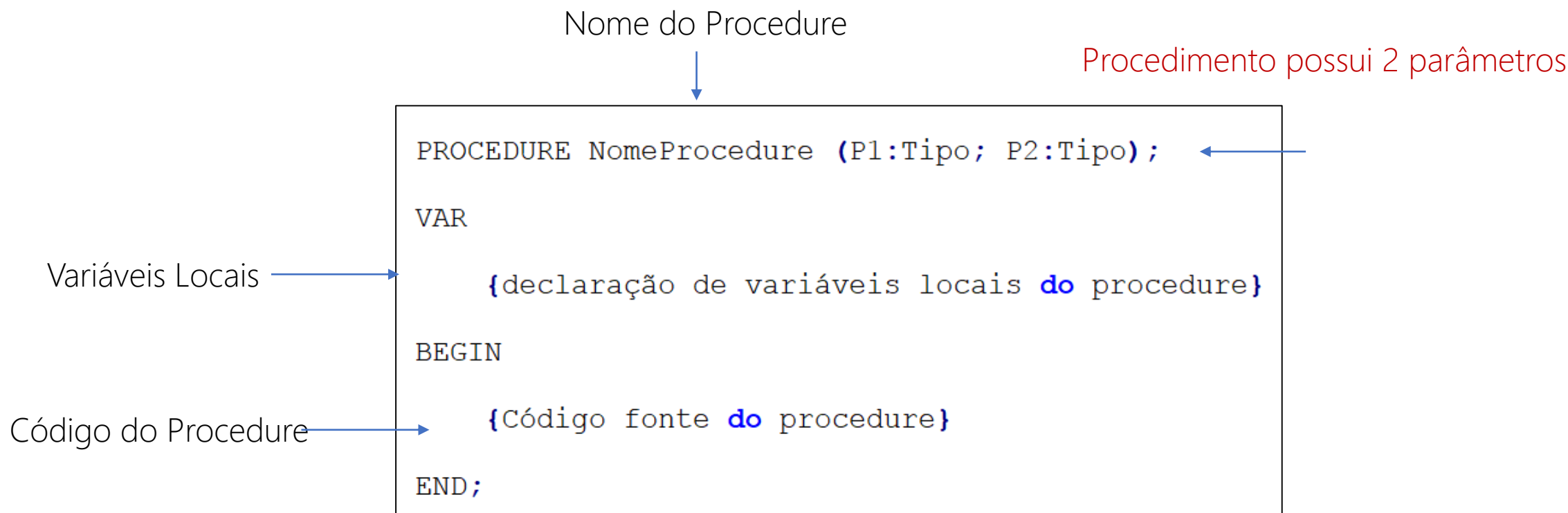


Criando um procedimento

- Qual o objetivo de um parâmetro?
 - Um parâmetro deve ser visto como uma variável local do procedimento.
- O que ele faz?
 - Ele recebe valores passados para o parâmetro em sua chamada.
- Quantos parâmetros eu preciso declarar ao criar um procedimento?
 - O procedimento não é obrigado a ter parâmetros.
 - Você verá que será inevitável (útil) criar procedimentos que recebam parâmetros.
 - O programador é livre para definir quantos parâmetros o procedimento deve ter.

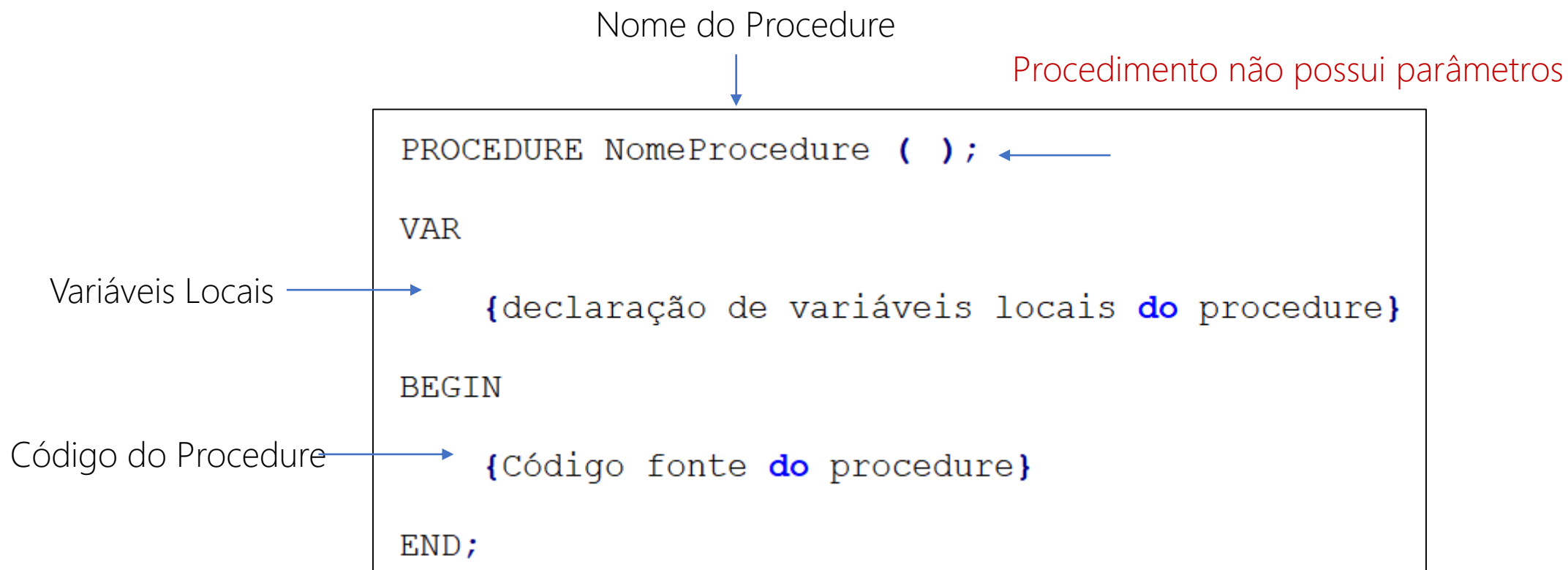


Estrutura de um procedimento



Estrutura mínima de um procedure em Objective Pascal (Delphi)

Estrutura de um procedimento



Estrutura mínima de um procedure em Objective Pascal (Delphi)

Criando um procedimento

- Dicas para nomes e tipos de parâmetros:
 - [0] Alguns programadores chamam **parâmetro** de **argumento** do procedure.
 - [1] Cada identificador (nome) do parâmetro deve possuir um nome único.
 - [2] Não utilize para nome de parâmetro os nomes das palavras da linguagem.
 - [3] Respeite as regras quanto aos nomes (não use caracteres especiais).
 - [4] Utilize todos os tipos de dados suportados pela linguagem:
 - idade: INTEGER
 - nota: REAL
 - salario: EXTENDED
 - nome: STRING
 - voto: BOOLEAN

Criando um procedimento

- Onde um procedimento deve ser criado?
 - Existem duas etapas a serem feitas.
 - [1] Declaração:
 - Indica (ao PAS) que estamos criando um novo procedimento.
 - Deve ser feito na seção interface dentro do public.
 - Nosso procedimento será público para todo programa (formulários).
 - Devemos escrever apenas a primeira linha do procedimento.
 - A apostila chama a primeira linha de header (cabeçalho) do procedimento.
 - Ela informa ao Delphi o nome da função e seus parâmetros.



Declaração de um procedimento

- Onde um procedimento deve ser criado?

Nome do Procedimento Argumentos (Parâmetros)

```
private  
    procedure media3Valores(n1: INTEGER; n2: INTEGER; n3: INTEGER);  
public
```

;

Local e formato para definição de um procedimento no PAS

Definição de um procedimento

- Onde um procedimento deve ser criado?
 - Existem duas etapas a serem feitas.
 - [2] Definição:
 - Deve ser feito após {\$R *.dfm} e antes de end.
 - O local não é importante mas o programador deve seguir uma estrutura lógica.
 - A organização ajuda outros programadores (professor) a entender o código.
 - A definição nada mais é do que o procedimento em si (código fonte).
 - Como nossos procedimentos irão utilizar componentes (VCL) do form:
 - Precisaremos utilizar uma cláusula "TForm." antes do nome do procedimento.



Definição de um procedimento

- Onde um procedimento deve ser criado?

Vinculação ao Form

Nome do Procedimento

Argumentos (Parâmetros)

```
PROCEDURE TForm.media3Valores(n1:INTEGER; n2:INTEGER; n3:INTEGER);  
VAR  
BEGIN  
    ..  
    ..  
    ..  
END;
```

Atalhos do DELPHI



- Para definir procedimentos e funções utilize o atalho:
 - 1. Declare o procedimento ou a função.
 - 2. Com o cursor na frente da declaração pressione **Crtl + Shift + C**
- O atalho acima se aplica a procedimentos e funções.



Definição de um procedimento

- Como chamar um procedimento?
 - Basta chamar o nome do procedimento e passar os parâmetros.
 - Se uma função possui 2 parâmetros você é obrigado a passar 2 valores.
 - Sempre que possível, passe os valores via variáveis.

Chamando procedure media3Valores
Que possui 3 parâmetros →

```
procedure TForm4.Button1Click(Sender: TObject);  
VAR  
  
N1:= 2;  
N2:= 4;  
N3:= 8;  
  
BEGIN  
  
    media3Valores (N1,N2,N3) ;  
  
END;
```

Exemplo Condicional (Fácil)

Faça um procedimento que **calcule a média de 3 notas e mostre na saída padrão do Delphi**. As notas são valores não inteiros (REAL) que devem ser passadas para o procedimento. O programa só deve calcular a média caso o somatório das notas seja maior do que zero (verifique e informe na **saída padrão** caso o cálculo não seja possível ser realizado). Faça um programa em Delphi para testar o procedimento. O programa deve receber (do usuário) valores via **InputBox** ou via **TEdit** e deve chamar o **procedimento acima**.

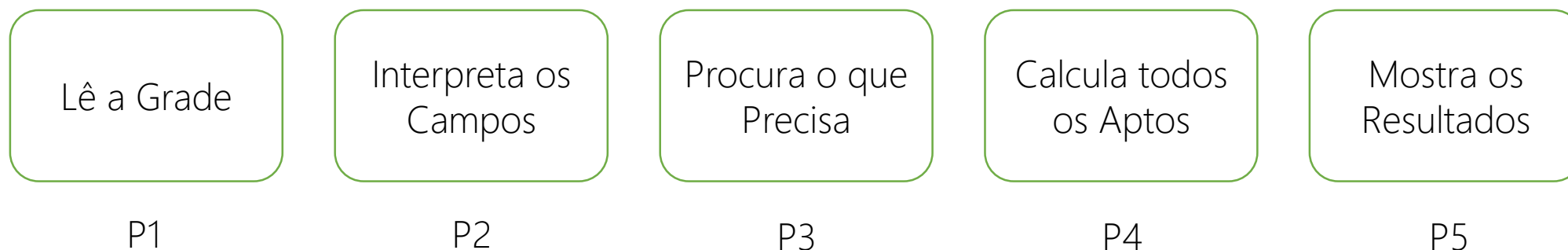
Exemplo Iterativo (Difícil)

Faça um procedimento que **calcule se um número é primo ou não e mostre na saída padrão do Delphi a mensagem (PRIMO) ou (NÃO PRIMO)**. O número passado ao procedimento deve ser inteiro (INTEGER) e maior do que zero (verifique e informe na saída padrão caso o cálculo não seja possível ser realizado). Números primos são números divisíveis por 1 e por ele mesmo. Por exemplo, 3 é primo pois só é capaz de ser dividido por 1 e por 3. Já 6 não é primo pois é capaz de ser dividido por 1, 2, 3 e por 6. Faça um programa em Delphi para testar o procedimento. O programa deve receber (do usuário) valores via InputBox ou via TEdit e deve chamar o procedimento acima.



Vantagens no uso de Procedimentos

- Desenvolvimento de programas modulares:
 - Parte da lógica do código é encapsulada em um procedimento.
 - Imagine um programa com 500 linhas de código.
 - Graças a análise e o projeto modular 5 funções de 100 linhas:
 - Organizam o que precisa ser programado e resolvido.



Vantagens no uso de Procedimentos

- Como saber a hora certa de criar um procedimento?
 - Antes de começar a programar o problema.
 - Realize um pouco de **engenharia** de **software**.
 - Leia com cuidado o **enunciado** do **problema** (requisitos do programa).
 - Faça uma breve **análise** do enunciado.
 - Se possível, divida o que precisa ser feito em pedaços.
 - Em seguida, realize o **projeto** dos pedaços de forma estruturada.
 - Caso seja possível, divida os pedaços em sub pedaços.
 - Só então, de início a programação.
 - Neste estágio você conseguirá enxergar o problema em sub problemas.



Passagem de Parâmetros

- Os valores passados aos procedimentos:
 - São copiados para os mesmos.
 - Chama-se esta técnica de passagem de parâmetros por valor (cópia).
 - Nossos procedimentos não retornam valores.
 - Quando a palavra END; é encontrada toda a estrutura do procedure acaba.
 - As variáveis locais ao procedimento são descartadas pelo Delphi.



Passagem de Parâmetros

- Os valores passados aos procedimentos:
 - Existe outra técnica para realizar a passagem de parâmetros.
 - Ela é conhecida como passagem de parâmetro por referência.
 - O uso da técnica é simples e o impacto no resultado é significativo.
 - Todo parâmetro por referência deve vir antecedido da palavra VAR.
- Isso significa que o parâmetro irá possuir o endereço da variável original:
 - Se este parâmetro for modificado pelo procedimento o valor será modificado.
 - Através desta técnica é possível retornar valores pelos parâmetros.
 - O usuário da função deverá conhecer se o parâmetro é por cópia ou por referência.
 - Pois alguns procedimentos poderão retornar valores (via parâmetro).



Passagem de Parâmetros

- Os valores passados aos procedimentos:

Parâmetros por cópia



Parâmetro referência



```
PROCEDURE Media (Nota1: REAL; Nota2: REAL; VAR Media:REAL) ;  
VAR  
BEGIN  
  
    Media:= (Nota1 + Nota2)/2;  
  
END;
```

Exemplo Condicional (Fácil)



Faça um procedimento que **calcule a média de 3 notas e mostre na saída padrão do Delphi**. As notas são valores não inteiros (REAL) que devem ser passadas para o procedimento. O procedimento deve possuir um quarto parâmetro que deve possuir o retorno do procedimento (por referência). O programa só deve calcular a média caso o somatório das notas seja maior do que zero (verifique e retorne 0 via parâmetro caso o cálculo não seja possível). Faça um programa em Delphi para testar o procedimento. O programa deve receber (do usuário) valores via InputBox ou via TEdit e deve chamar o procedimento acima. O retorno via referência deve ser capturado e mostrado na tela.



O que é uma função?

- Para o Delphi (Object Pascal):
 - Uma função é um *procedimento* que retorna um valor (apenas isso).
 - Sendo assim, todos os conhecimentos vistos anteriormente são os mesmos.
 - Exceto pelo retorno da função (aprenderemos como fazer a seguir).
 - Toda função pode retornar um valor (caso o programador queira retornar).
 - O retorno é feito com a palavra reservada RETORNE
 - Exceto pelo uso da palavra **function** e não **procedure**.



Como criar uma função?

- Tudo o que aprendemos se aplica em função:

FUNCTION



```
FUNCTION FAT(N: INTEGER;) : INTEGER;  
  
VAR F,I: INTEGER;  
  
BEGIN  
  
    F := 1;  
    FOR I:= 1 to N DO  
  
        BEGIN  
            F:= F * I;  
        END;  
  
    RESULT := F;  
  
END;
```



Tipo de Retorno



Função que calcula o fatorial de um número (N)

Retorno é feito com RESULT



RESULT := F;



O que é uma função?

- Para o Delphi (Object Pascal):
 - A palavra **RESULT** pode **aparecer várias vezes** na função.
 - Você é **livre** para **adicionar** a **palavra** dentro das **condições** e/ou **laços**.
 - **Tudo** o que foi visto para **procedure** se **aplica** para **function**:
 - Inclusive a **passagem** de **parâmetro** por **valor** e por **referencia**.
 - Você **não** é **obrigado** a utilizar **RESULT** podendo **retornar valor** por **referência**.



Exemplo Condicional (Fácil)

Escreva uma função no Delphi responsável por receber 3 lados (REAL) de um triângulo e por retornar o tipo de triângulo informado. Os tipos de retorno podem ser: (1) Triângulo isóceles; (2) Triângulo escaleno; (3) Triângulo equilátero. Triângulos escalenos possuem 3 lados com tamanhos diferentes. Triângulos equiláteros possuem 3 lados com o mesmo tamanho. Triângulos isóceles possuem 2 lados com medidas iguais e um com medida diferente. O cálculo só deve ser feito com números positivos e maiores que zero. Para entradas inválidas, a função deve retornar (-1). Faça ainda um programa VCL (mínimo) que receba da entrada padrão (InputBox) ou de um TEdit o número de cada um dos 3 lados e retorne na saída padrão (ShowMessage) ou TLabel as mensagens (TRIÂNGULO EQUILÁTERO) (TRIÂNGULO ISÓCELES) (TRIÂNGULO ESCALENO) (ENTRADAS NÃO SUPORTADAS).



Exemplo Iterativo (Difícil)

Escreva uma função no Delphi responsável por **receber 2 números (INTEGER) e por verificar se os mesmos são amigos ou não**. Números são amigos se a soma dos divisores de $N1$ (excluindo o próprio $N1$) é igual a $N2$ e se a **soma dos divisores** de $N2$ (excluindo o próprio $N2$) é **igual** a $N1$. Sua função deve **retornar** (1) se os 2 números **informados** para a **função** forem **amigos** (possuírem a propriedade anterior). Caso **contrário** deve retornar (0). Se uma das entradas forem **negativas** a função deve **retornar** (-1). Faça ainda um programa **VCL** (mínimo) que receba da **entrada padrão** (InputBox) ou de um TEdit os **dois números** e **retorne** na **saída padrão** (ShowMessage) ou TLabel as **mensagens** (NÚMEROS AMIGOS) (NÚMEROS NÃO AMIGOS) (ENTRADAS NÃO SUPORTADAS).



O Delphi possui funções?

- Sim, como você já deve imaginar:
 - Todos os elementos VCL do Delphi são procedimentos e funções.
 - Conforme aprendemos, funções são blocos de código reutilizáveis.
 - Quando você arrasta (chama) um InputBox você está reusando o código.
- Assuma sempre ao criar funções:
 - Que no futuro você poderá **reutilizar** as **funções**.
 - Que **outras pessoas** poderão **utilizar** estas **funções**.
 - Desde que você **disponibilize** o **código** (**publicamente**).
 - Funções também podem ser **vendidas** para **outros programadores**.
 - Encare um **programa** como **funções** que **chamam funções**.



O Delphi possui funções?

- Vamos estudar algumas funções matemáticas clássicas:

- $x := \text{sqr}(x)$ Quadrado de x (INTEGER) : (INTEGER)
- $x := \text{sqrt}(x)$ Raiz quadrada de x (INTEGER) : (INTEGER)
- $x := \text{sin}(x)$ Seno de x (INTEGER) : (INTEGER)
- $x := \text{cos}(x)$ Cosseno de x (INTEGER) : (INTEGER)
- $x := \text{tan}(x)$ Tangente de x (INTEGER) : (INTEGER)
- $x := \text{int}(x)$ Parte inteira de x (REAL) : (INTEGER)
- $x := \text{frac}(x)$ Parte fracionária de x (REAL) : (INTEGER)
- $x := \text{power}(x,y)$ x na potencia y (xy) (INTEGER,INTEGER) : (INTEGER)
- $x := \text{abs}(x)$ Valor absoluto de x (INTEGER) : (INTEGER)

O Delphi possui funções?

- Seguem mais funções mais clássicas:
 - // Entrada e saída
 - NomePais := **InputBox** ('Escolha de país', 'Digite o nome do país:', 'Brasil');
 - **ShowMessage** ('Olá mundo');
 - // Incremento e decremento
 - **Inc**(X);
 - **Dec**(X);

O Delphi possui funções?

- Seguem mais funções mais clássicas:
 - // Conversão entre Maiúsculo para Minúsculo
 - Nome := **LowerCase**('RAFAEL');
 - Nome := **UpperCase**('rafael');
 - // Números aleatórios e arredondamento/truncamento.
 - Aleatorio := RandomRange(10,20);
 - Num := **round**(23,4);
 - X:= **trunc**(23,4);

Boas práticas de programação

- Entre programadores:
 - É legal definir algumas regras comuns.
 - Regras baseadas no que diz o coletivo (consenso entre a comunidade).
 - Vamos aprender algumas convenções que tornam o código mais agradável.
 - Respeitando as regras do Object Pascal (Delphi) e do RAD (VCL).
- Regra [1] : Comentários
 - Lembre que o Delphi possui 3 tipos de comentários sendo eles.
 - // Sou um comentário de linha
 - (* Comentário de bloco *) ou { Comentário de bloco }



Boas práticas de programação

- Onde eu utilizo um comentário?
 - Sempre que estiver "definindo" um procedimento ou função.
 - Com o comentário deve ser escrito para um procedimento?
 - Faça um comentário de bloco como o descrito abaixo
- ```
{
 Autor: Raffael Bottoli Schemmer
 Objetivo: Procedimento que calcula e retorna no 2 parâmetro a média
 PAR: (real, VAR real) Notas a serem calculadas a média
}
```



# Boas práticas de programação

- Onde eu utilizo um comentário?
    - Sempre que estiver "definindo" um procedimento ou função.
    - Com o comentário deve ser escrito para uma função?
      - Faça um comentário de bloco como o descrito abaixo
- ```
{  
    Autor: Raffael Bottoli Schemmer  
    Objetivo: Função que calcula e retorna a média de 2 parâmetros  
    RET: (real) Retorna a média dos números  
    PAR: (real, real) Notas a serem calculadas a média  
}
```



Boas práticas de programação

- Onde eu utilizo um comentário?
 - Sempre que estiver "definindo" um procedimento ou função.
 - Com o comentário deve ser escrito para uma função?
 - Faça um comentário de bloco como o descrito abaixo
- ```
{
 Autor: Raffael Bottoli Schemmer
 Objetivo: Função que calcula e retorna no 2 parâmetro a média
 PAR: (real, VAR real) Notas a serem calculadas a média
}
```

# Boas práticas de programação

- Onde eu utilizo um comentário?
  - Sempre que implementar algo relativamente importante e complexo.
  - Comentários de linha são importantes para detalhar coisas simples.
  - Exemplo:
    - $\text{senha} = a*b + !a^3*2/1+2\backslash 3 + !a*b$
    - Observe como a linha acima necessita de um comentário para facilitar o entendimento.
- Afinal, porque comentar?
  - Pois outras pessoas (ou você mesmo) irá querer estudar o programa no futuro.
  - Os comentários auxiliam a entender um código complexo (ou feito no passado).



# Boas práticas de programação

- Regra [2]: **Nome** de **variável** ou **parâmetro**:
  - Nunca **utilize** nomes **similares** do **Delphi**.
  - Como uma variável **BEG** ou **IND**.
  - **Convencione** em utilizar as **palavras** da **linguagem** em letra **maiúscula**.
  - **Nome** das **variáveis** em letra minúscula.
  - **Nomes simples** (idade, data, salario, hora)
  - **Nomes composto** (diaSemana, salarioMinimo, valorMedio).
  - Este **padrão** é **conhecido** como **camelCase**.
  - Linguagens **modernas** como Java/C# possuem como **regra** este **padrão**.
  - Linguagens **antigas** não se dão bem com o **CapsLock** (CASE).
    - **FUNCTION** / **function** / **Function** são 3 coisas diferentes para o C e para o C++.
    - O case sensitive (**CAPS**) **muda** o **sentido** das **coisas**.



# Boas práticas de programação

- Regra [3]: Uso de **procedimento** e **função**:
  - Tente **SEMPRE** que puder **utilizar** os **parâmetros** e **variáveis locais**.
  - Imagine a **função** de forma **autocontida** (independente) do **programa**.
  - Imagine você **vendendo** a **função** no **futuro**.
  - O que você **explicaria** ao **usuário** da **função** (cliente/programador)?
- Pergunta: O cliente ficaria feliz se fosse necessário declarar 10 variáveis globais, sendo elas com nomes específicos (Ex: contadorLaco3) do tipo INTEGER para utilizar a função?
- Pela **pergunta** acima, você deve **procurar** criar uma **função** modular (**encapsulada**), que possua **tudo** o que **precisa** dentro de si mesma.
- Isso **facilita** que a mesma seja **copiada** e **vendida** a outros **códigos/programadores**.



# Boas práticas de programação

- Regra [4]: **Variáveis locais** e **globais**
  - Procure utilizar o **mínimo** possível as **variáveis globais**.
  - Projete as **funções** para que as mesmas **consigam trocar** (**copiar**) os **dados**.
  - Se a **informação** for **muito grande** (matriz 100x100) passe a **matriz** por **referência** a cada **chamada** de **função**.
  - Utilize **variáveis globais** apenas em **último caso**.
  - Considere que a **variável global** faz **parte** do **programa** e não das **funções**.



# O que fazer agora?

## Lista de exercícios de Funções



[GitHub.com/RaffaelSchemmer/A2](https://github.com/RaffaelSchemmer/A2)





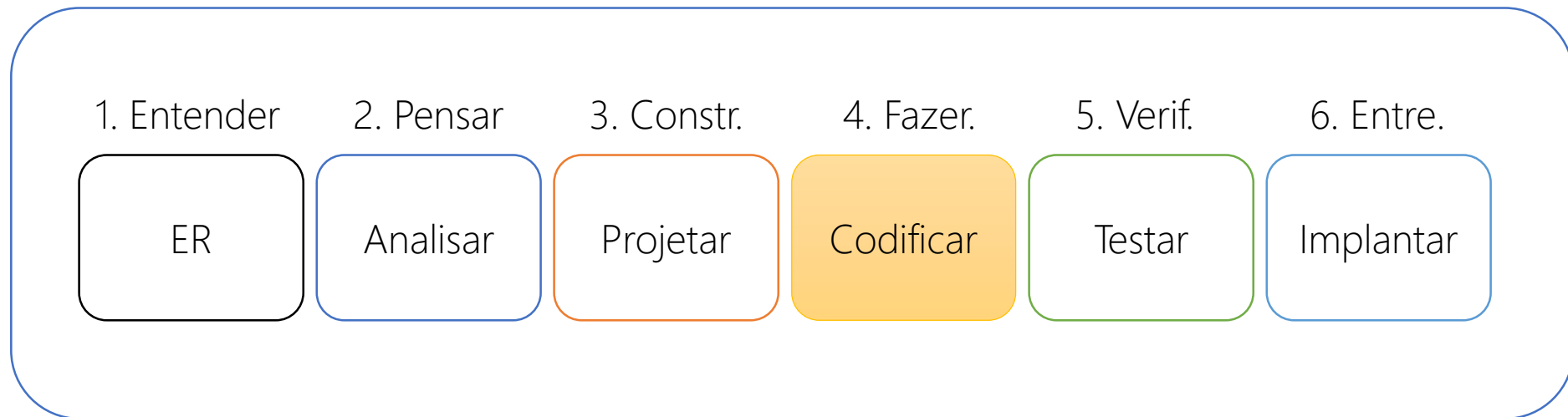


# Diretrizes do Projeto Final

Prof: Raffael Bottoli Schemmer  
Disciplina: (LPI) Linguagem de Programação I  
Ano: 2018.  
Módulo: Diretrizes do Projeto Final



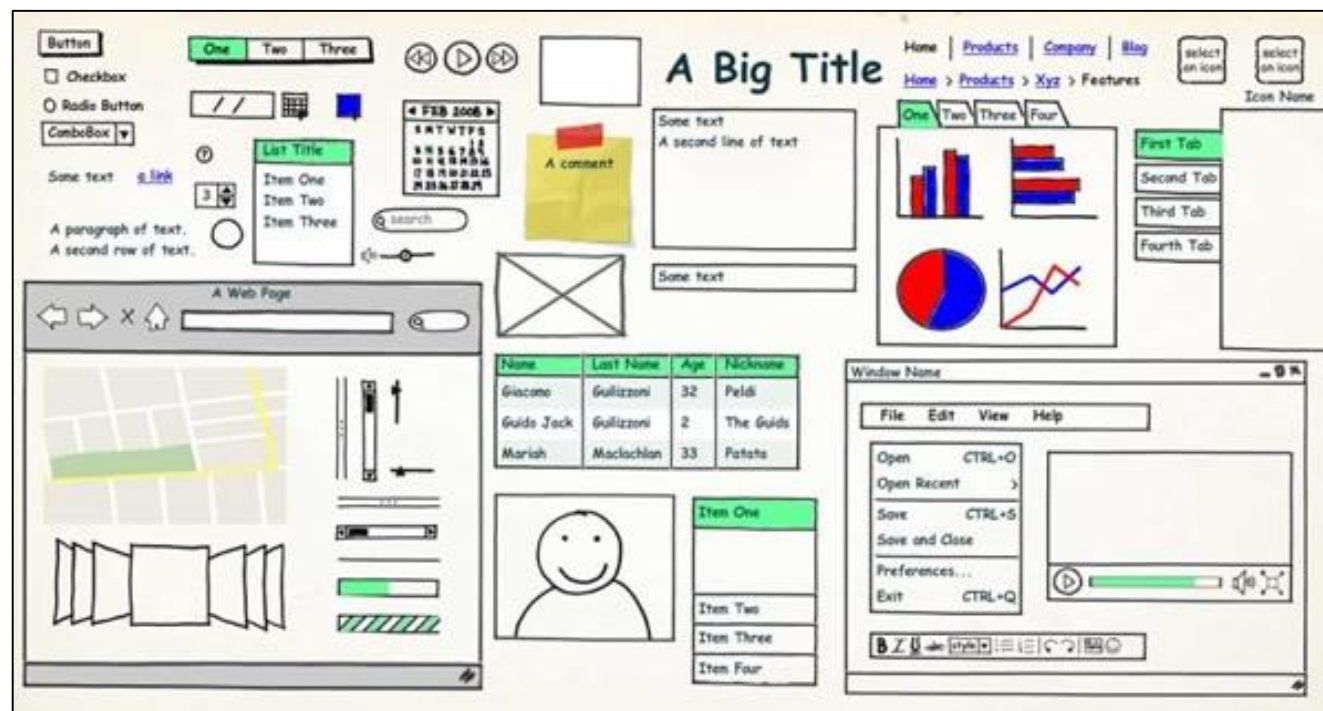
# Engenharia de Software



Fluxo de execução para o desenvolvimento modular do Trabalho Final



# Geração de Protótipos de Telas



# O que fazer agora?

## Realização do Trabalho Final (Projeto do Sistema de médio Porte)



[GitHub.com/RaffaelSchemmer/A2](https://github.com/RaffaelSchemmer/A2)





# InstallForge

Prof: Raffael Bottoli Schemmer  
Disciplina: (LPI) Linguagem de Programação I  
Ano: 2018.  
Módulo: InstallForge



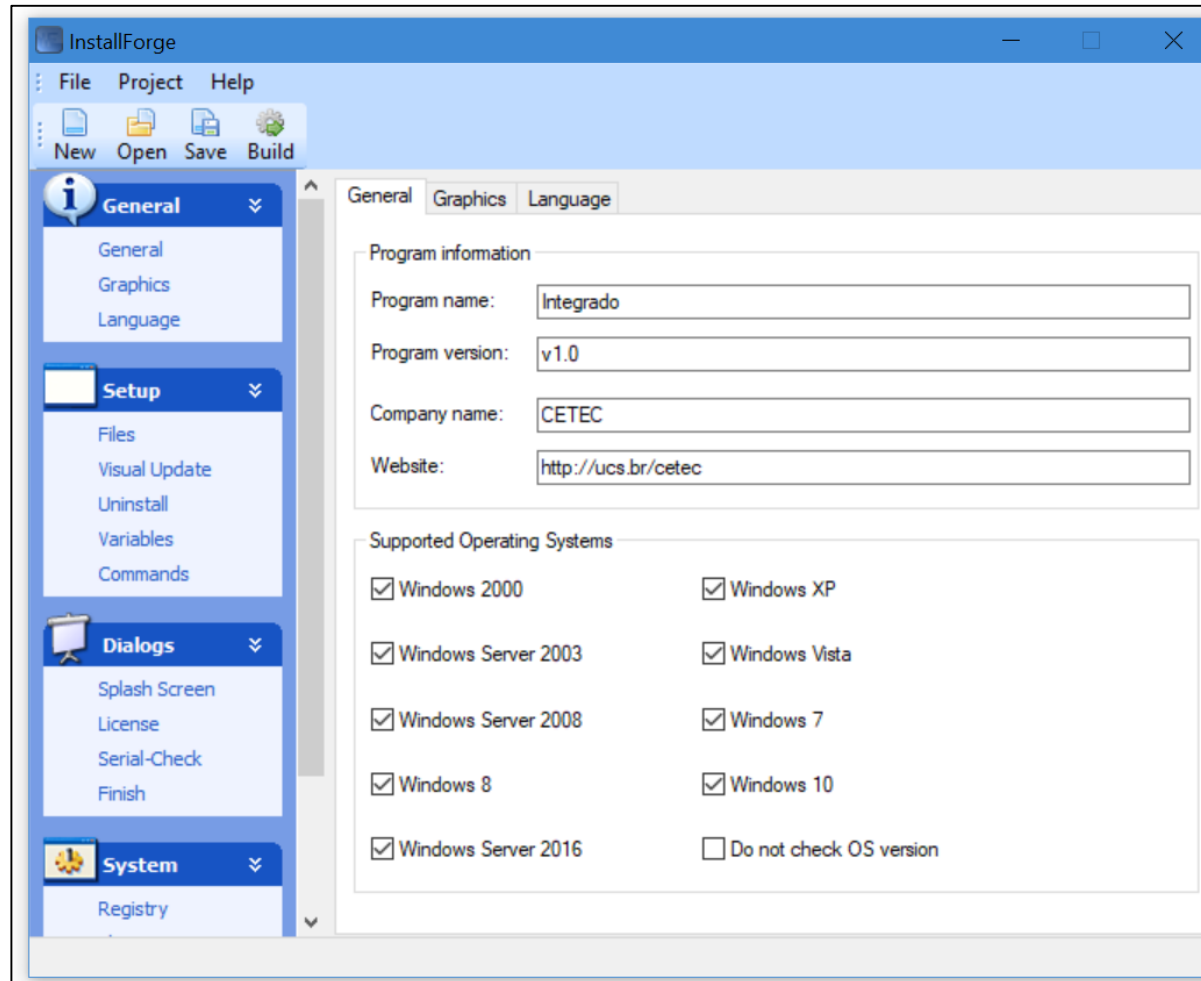
# Instalando no Computador



<https://youtu.be/mc7VfYelOml>

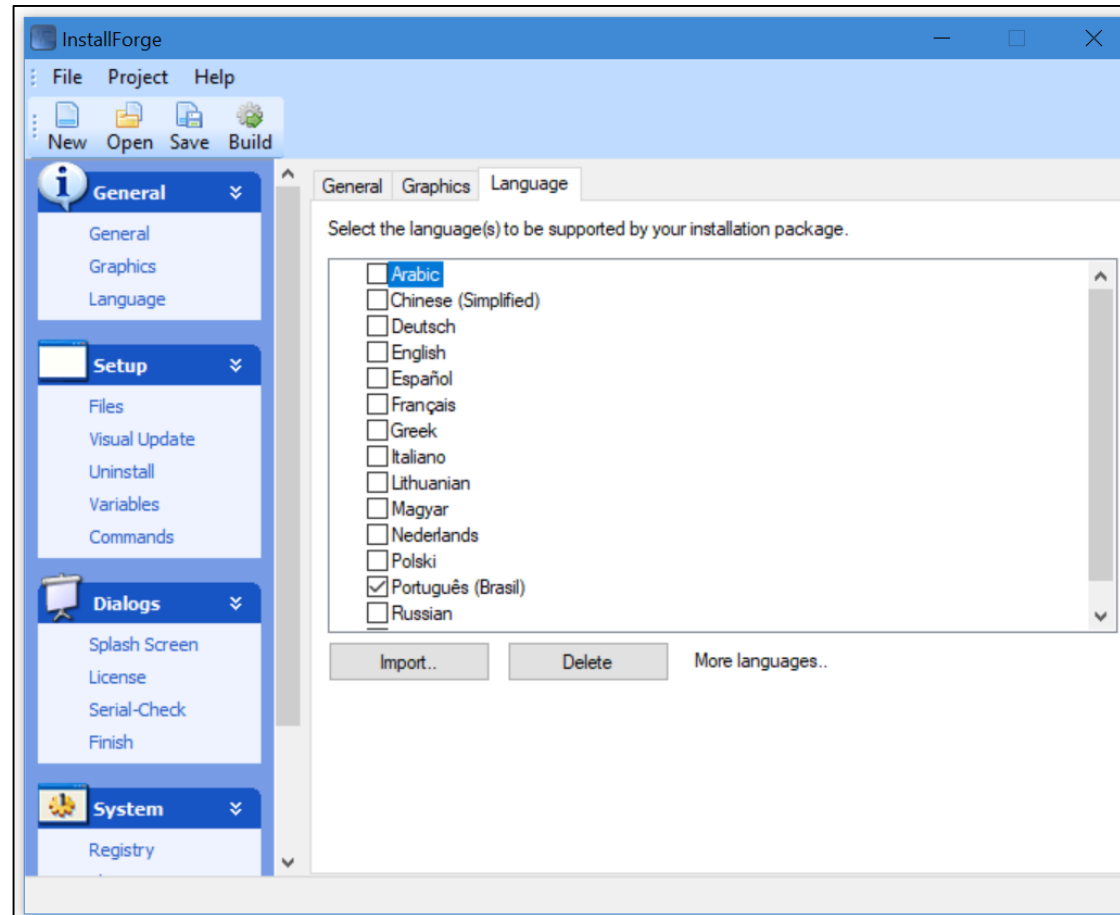


# Gerando Instalador



← Defina os dados do programa

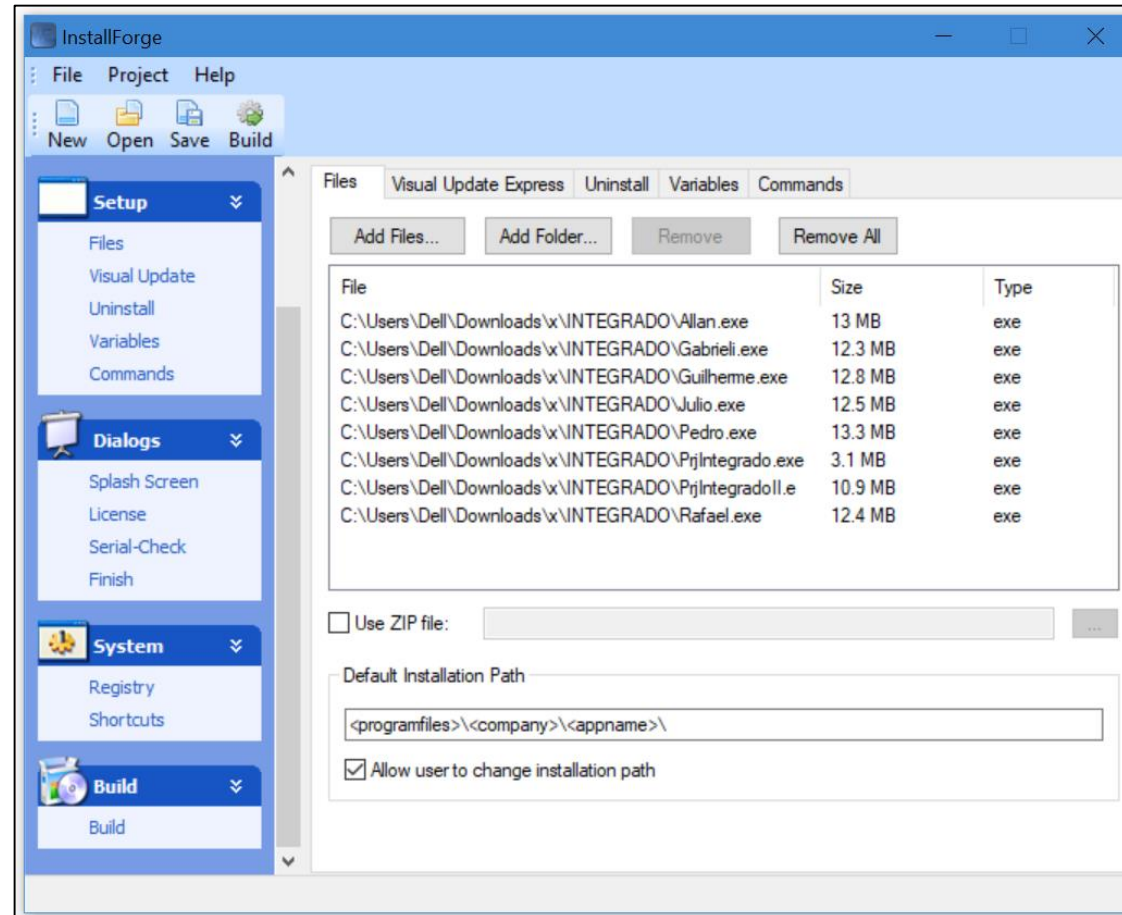
# Gerando Instalador



Defina o Idioma

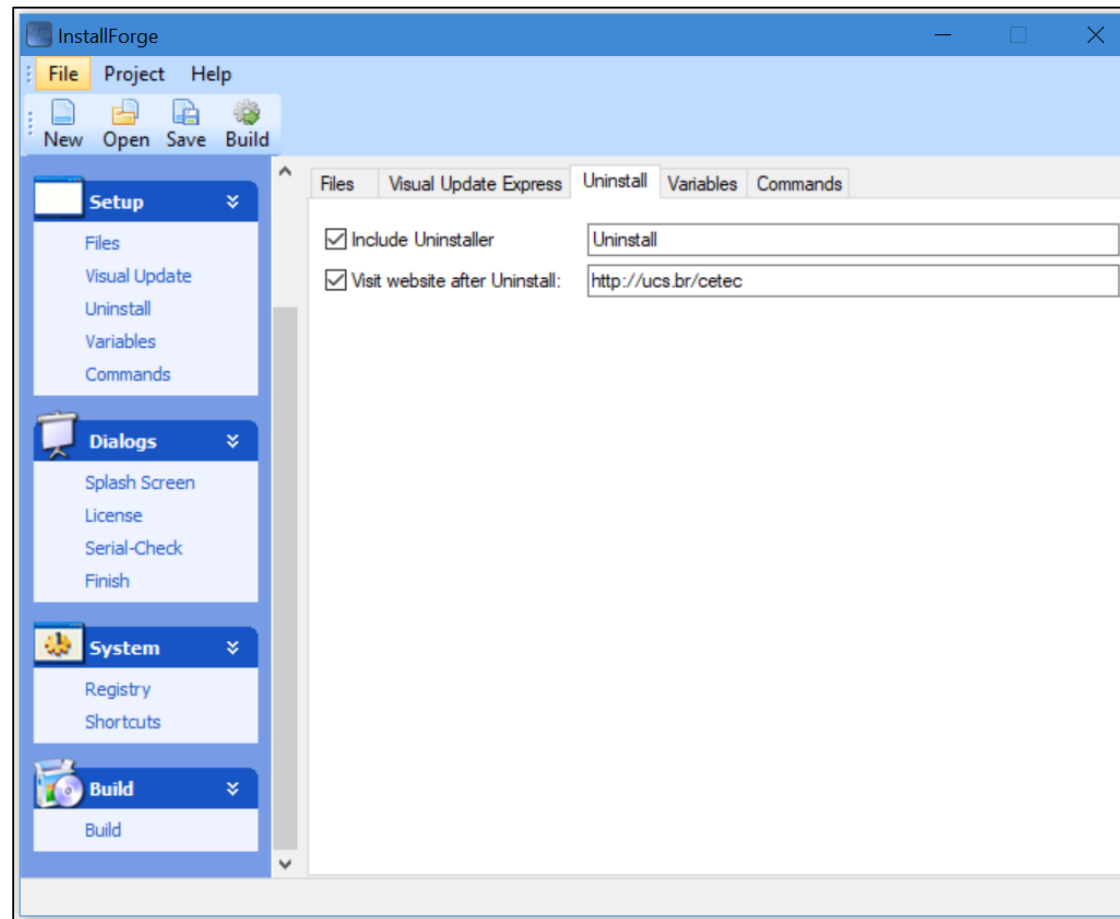


# Gerando Instalador



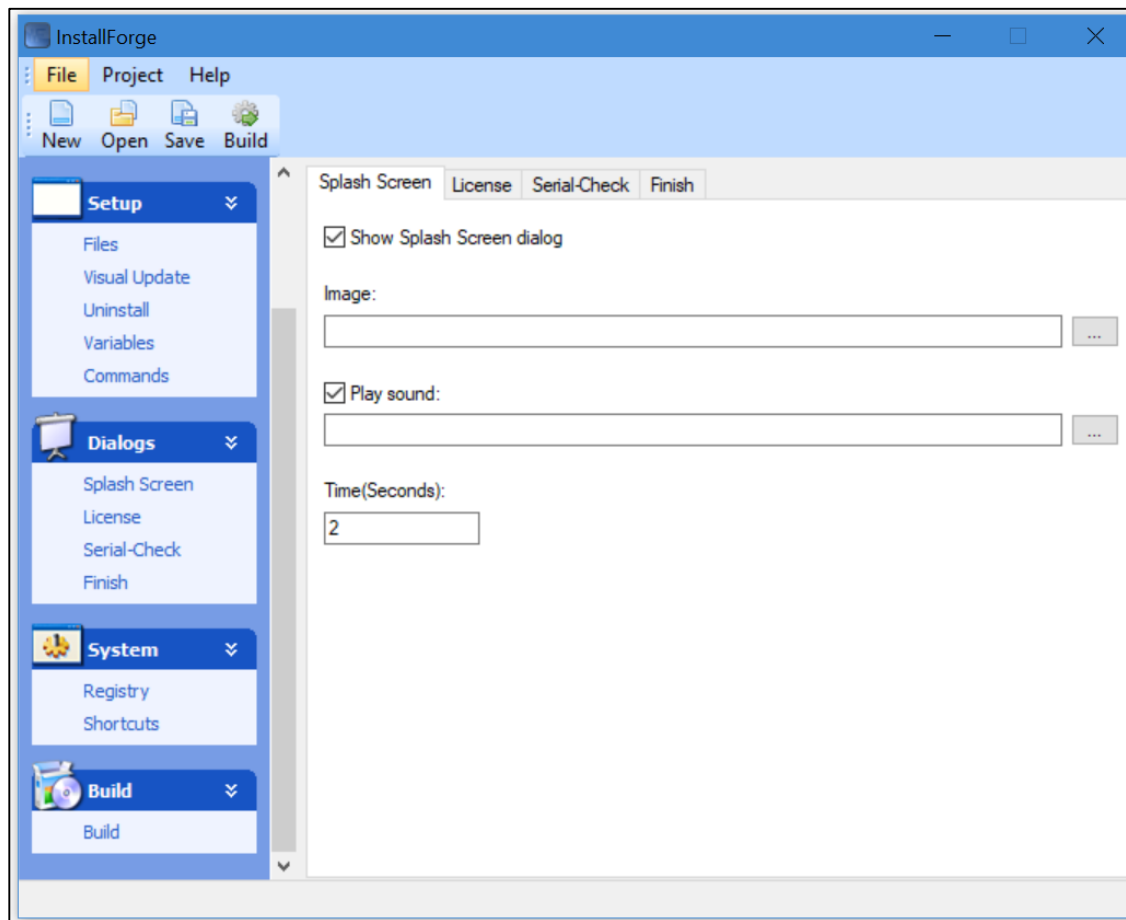
Adicione o programa

# Gerando Instalador



Adicione o Uninstaller

# Gerando Instalador



Adicione o logo do CETEC

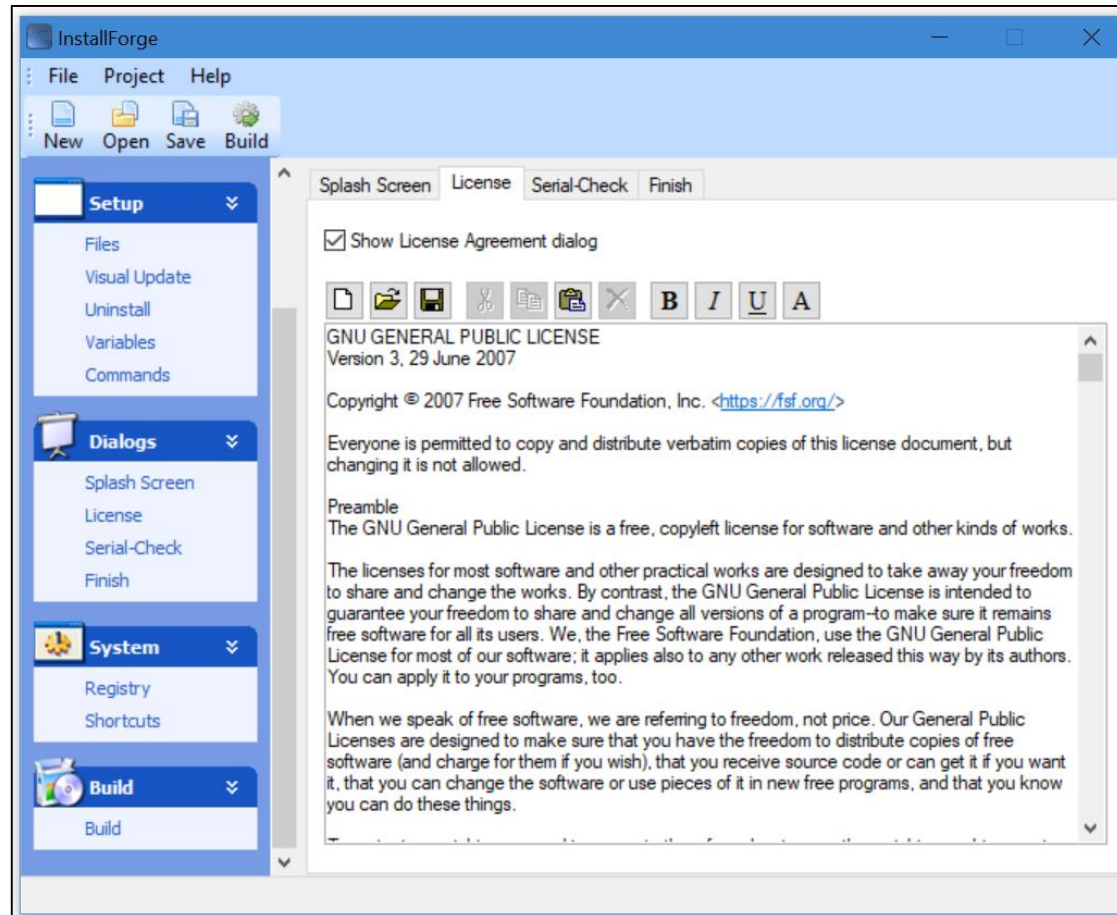


Adicione uma música



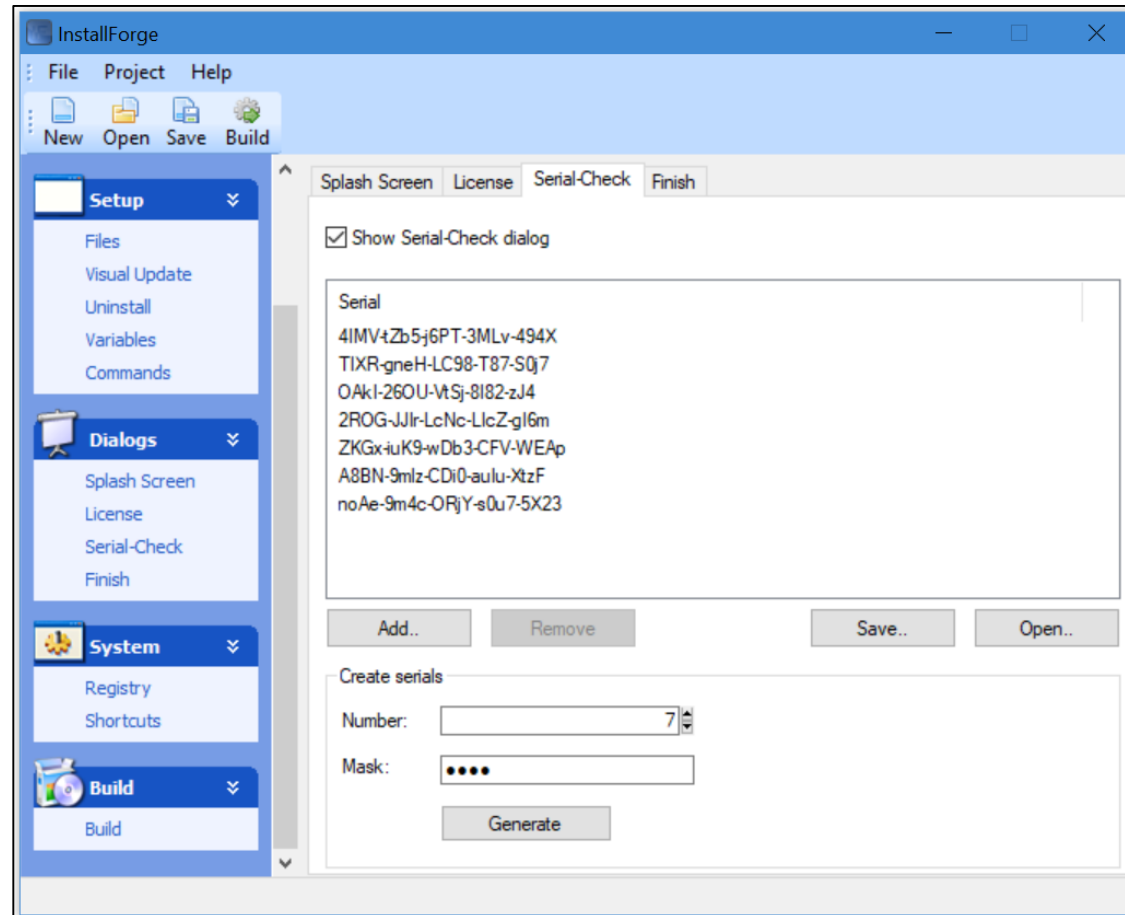
Defina um tempo inicial

# Gerando Instalador



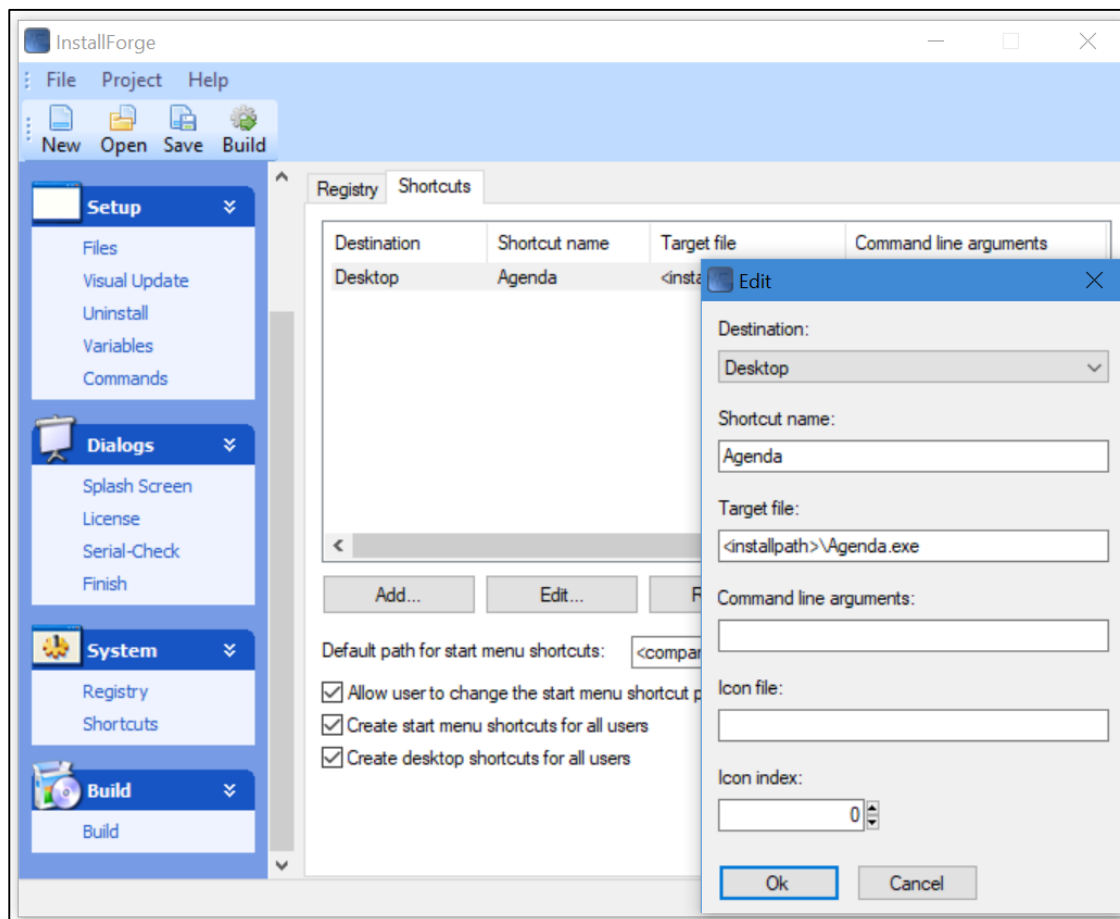
Defina uma Licença de uso

# Gerando Instalador



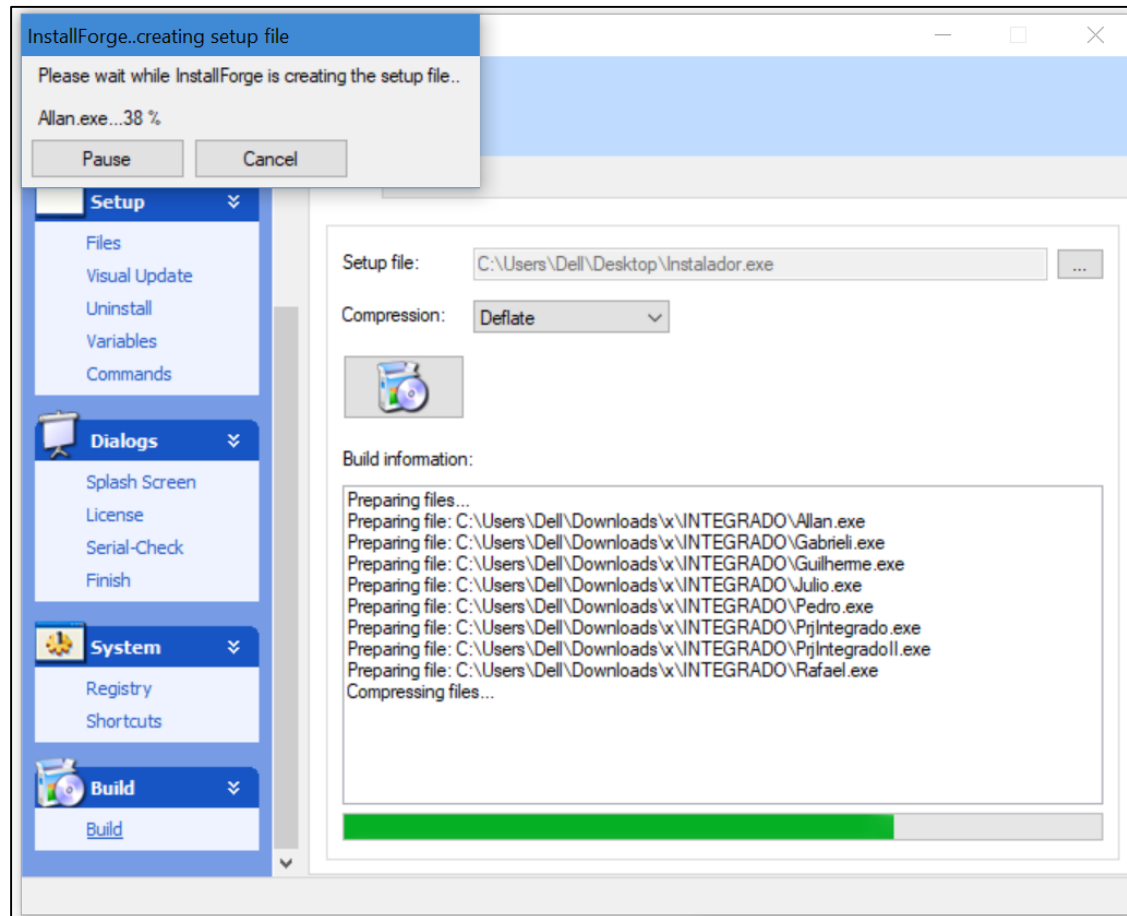
Defina um conjunto de seriais

# Gerando Instalador



Gerando atalho do programa

# Gerando Instalador



← Gerando instalador

# Realizando Instalação



Instale o executável que você gerou no computador







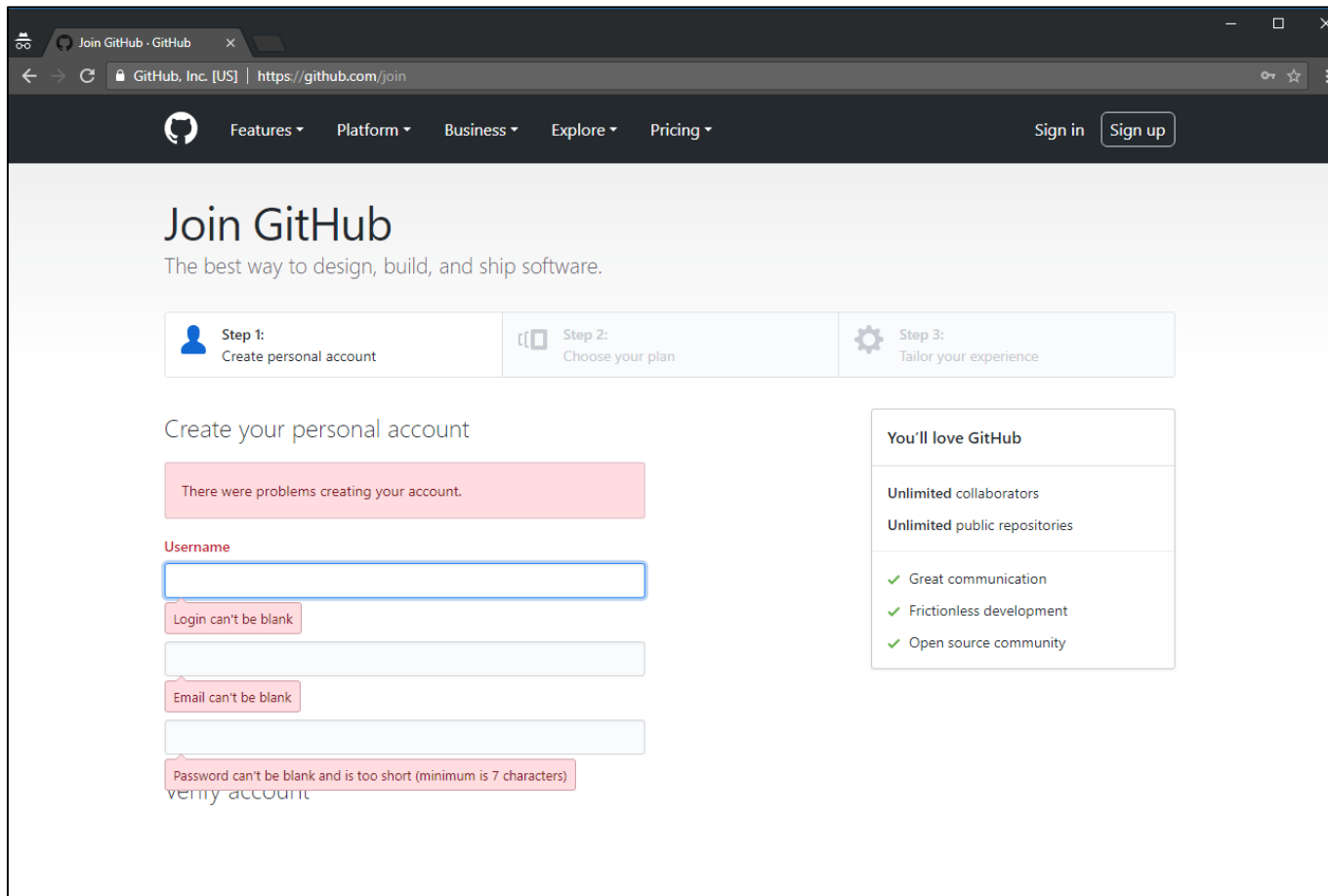
# GITHub



Prof: Raffael Bottoli Schemmer  
Disciplina: (LPI) Linguagem de Programação I  
Ano: 2018.  
Módulo: GITHub

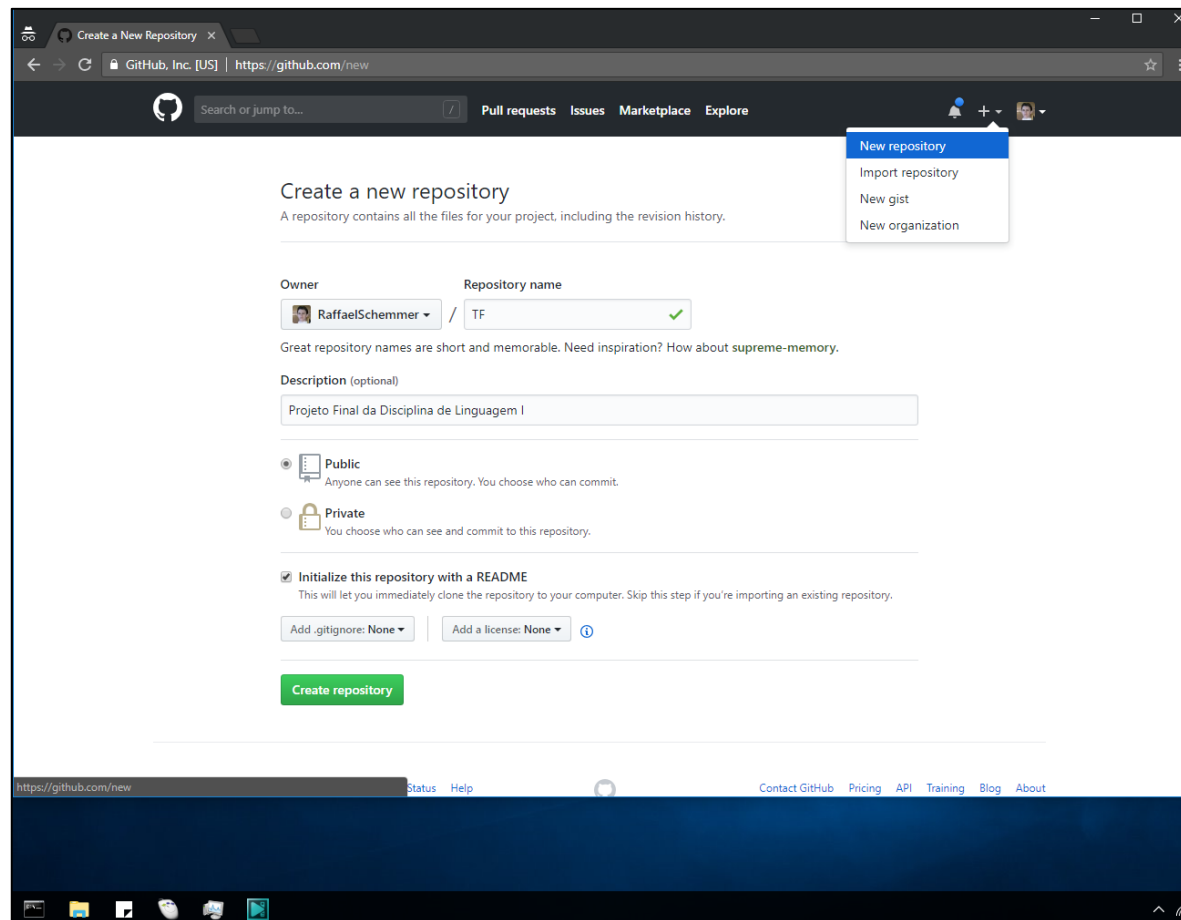


# Criando conta GitHub



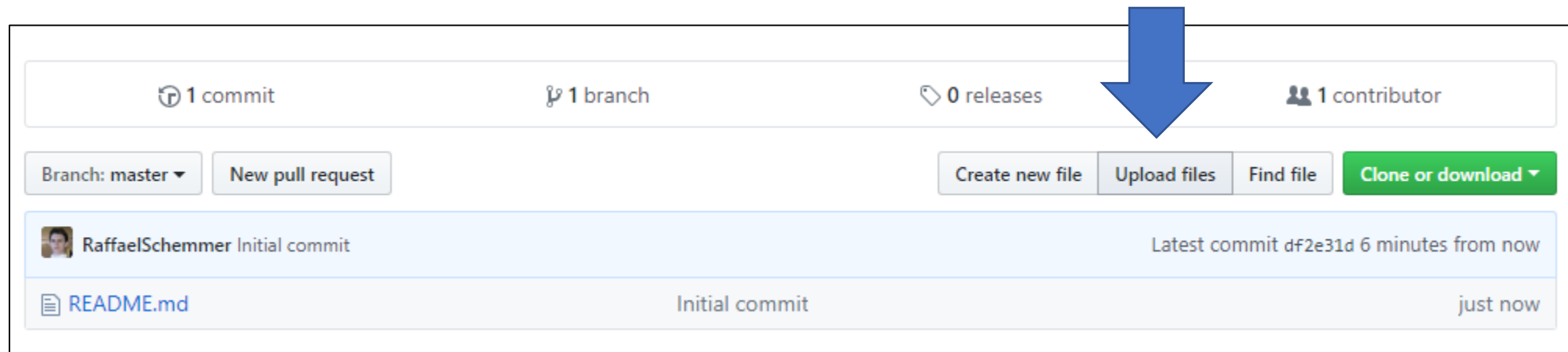
Crie uma conta com mail UCS e realize o login

# Criando novo Repositório



Crie um novo repositório com nome + descrição e com README

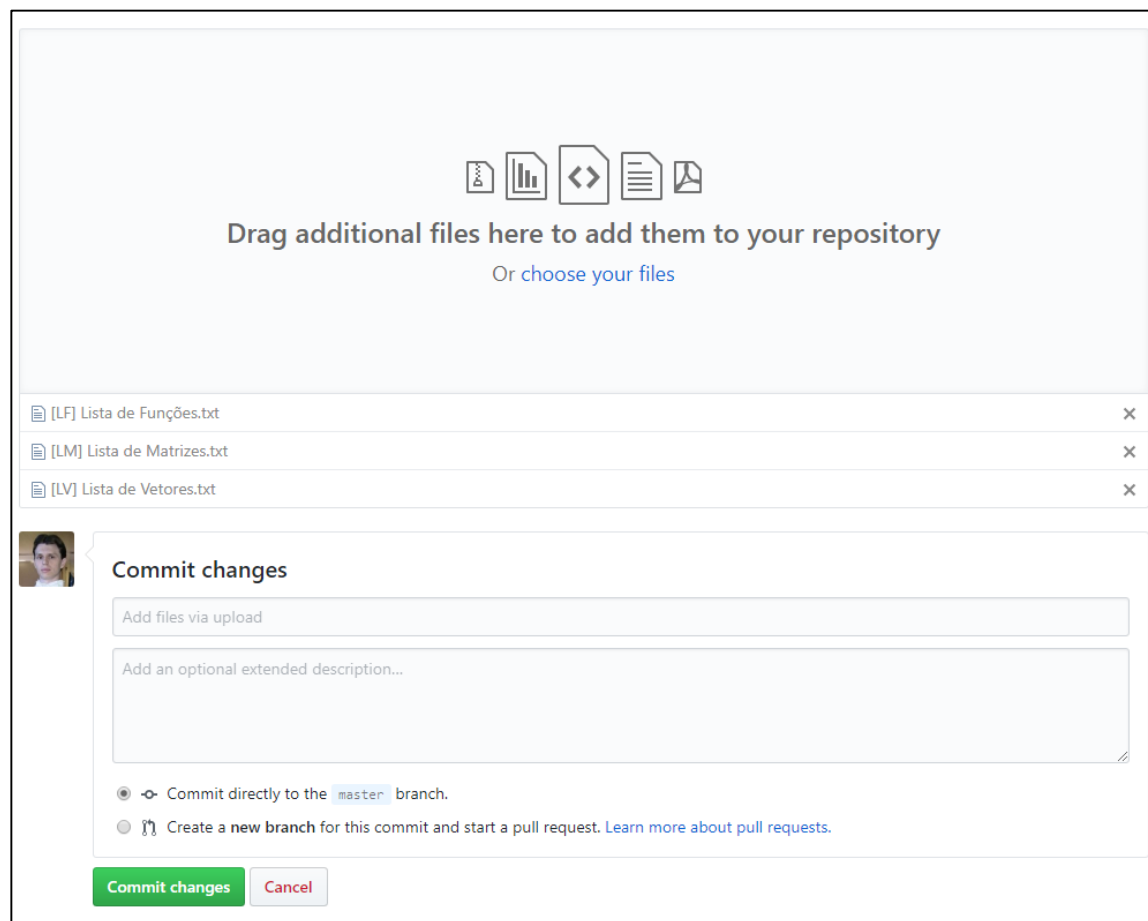
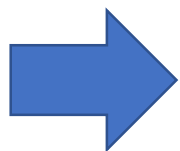
# Realizando Upload de Arquivos



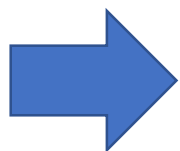
Local para Inserir Arquivos

# Realizando Upload de Arquivos

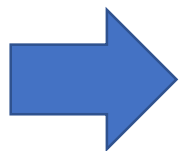
Arraste os arquivos pra cá



Espera o upload acontecer



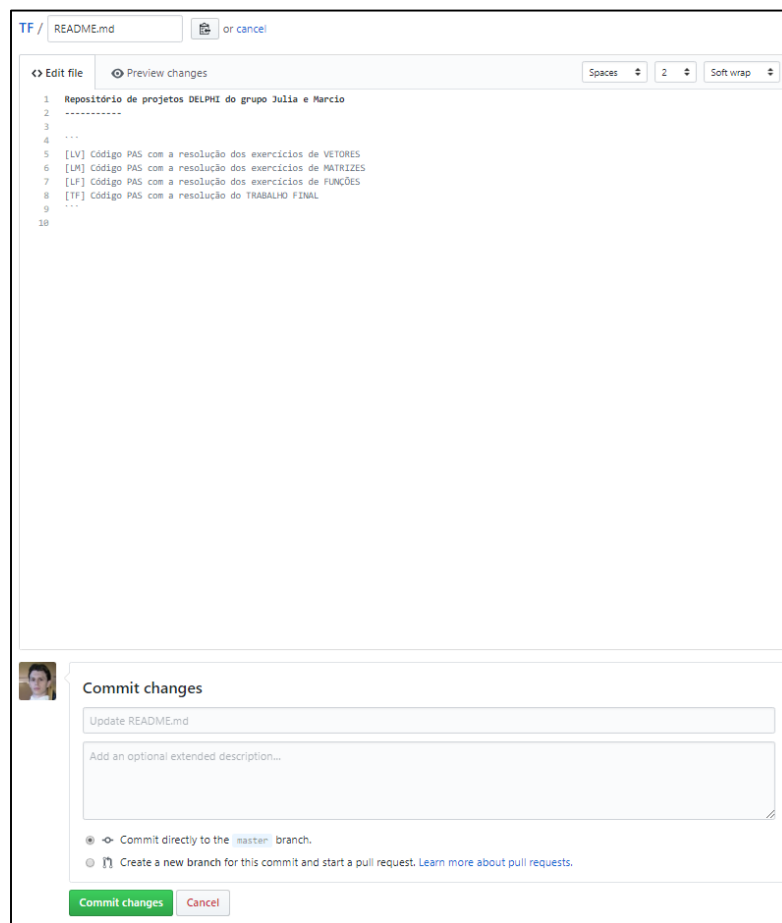
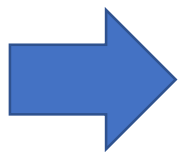
Confirme o upload



Inserindo Arquivos no GitHub

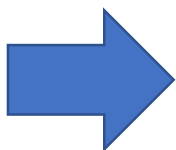
# Editando README

Descreva utilizando esta sintaxe



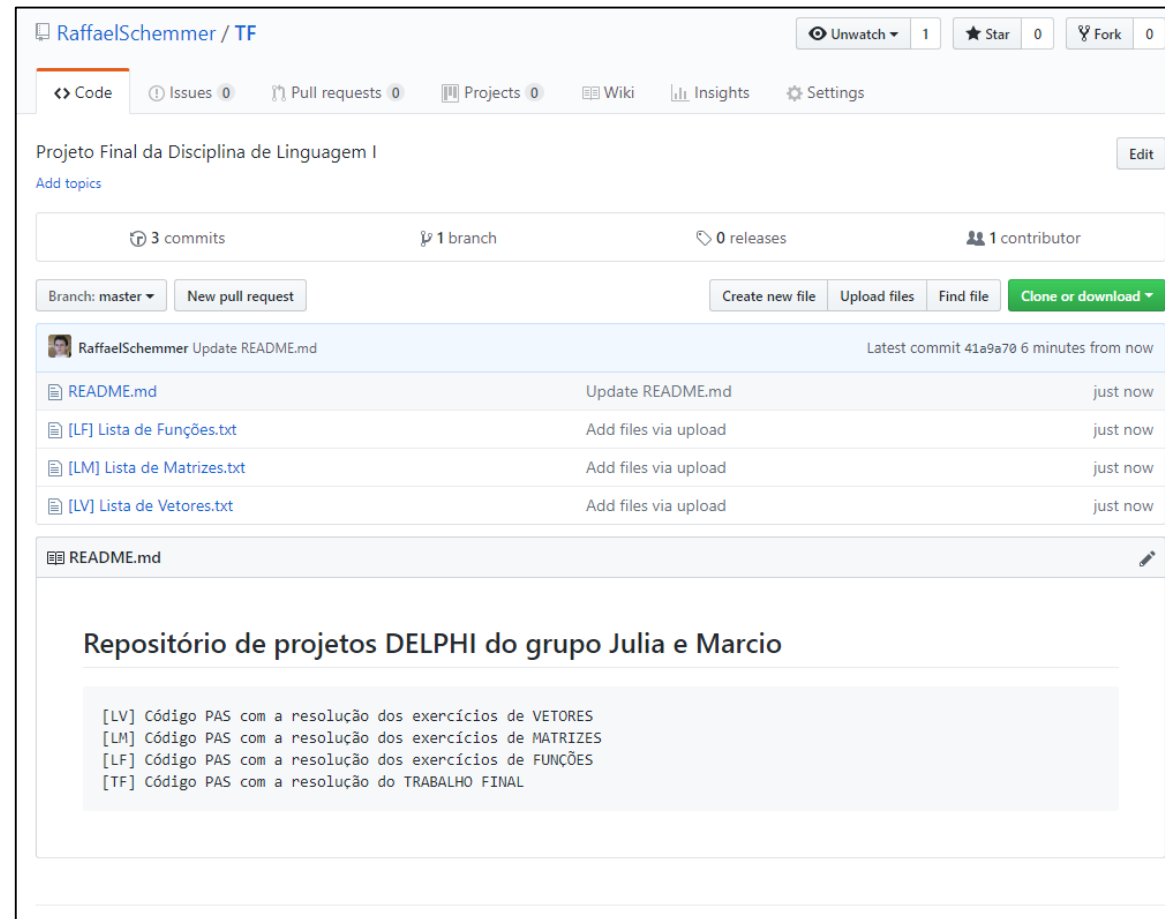
The screenshot shows the GitHub web interface for editing a README file. At the top, there's a tab for 'README.md' and a 'cancel' button. Below this, there are two tabs: 'Edit file' (selected) and 'Preview changes'. The 'Edit file' tab shows a code editor with a README template. The template includes a title 'Repositório de projetos DELPHI do grupo Julia e Marcio', a list of project types with their corresponding file extensions, and a list of project descriptions. The 'Commit changes' section at the bottom has a text input field for the commit message, a text area for an optional extended description, and two radio buttons for the commit target: 'Commit directly to the master branch' (selected) and 'Create a new branch for this commit and start a pull request'. At the bottom of the commit section are 'Commit changes' and 'Cancel' buttons.

Confirme o upload



Editando o arquivo README

# Realizando Upload de Arquivos



Repositório GITHub Atualizado