UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Electrotécnica

Sistemas de Telecomunicações

2011/2012

Trabalho 0:

Demonstração do ambiente Delphi 4
Aprendizagem do desenvolvimento de aplicações

Mestrado integrado em Engenharia Electrotécnica e de Computadores

Luís Bernardo
Paulo da Fonseca Pinto

1. Objectivo

Familiarização com o ambiente DELPHI 4 e com o desenvolvimento de aplicações. O trabalho consiste na introdução do código seguindo as instruções do enunciado, aprendendo a utilizar o ambiente e um conjunto de classes da biblioteca do Delphi 4.

2. Ambiente DELPHI 4

O Delphi 4 suporta a programação em PASCAL orientado a objectos. Para facilitar o desenvolvimento de aplicações, contém uma biblioteca de componentes vasto, que simplifica toda a parte de interacção com o utilizador, e mesmo, o desenvolvimento de aplicações distribuídas. Adicionalmente, o ambiente de programação gera automaticamente uma parte importante do código, deixando para o programador apenas a programação das rotinas específicas da aplicação. Por fim, fornece um manual em linha e um ambiente de "Debug" poderoso.

O Delphi usa a mesma sintaxe que o PASCAL para a especificação de algoritmos (ex. PROCEDURES, FUNCTIONS, BEGIN, END, WHILE, DO WHILE, FOR, IF, :=, etc). No entanto, usa uma abstracção de "**programação por eventos**" em vez da programação sequencial tradicional do PASCAL. O programador escreve apenas as rotinas de tratamento dos "clicks do rato", "movimentos de rato", "premir de teclas", "recepção de pacotes da rede", "expirar de temporizadores", e todo o tipo de eventos relevantes para cada aplicação.

A rotina '*main*' é gerada automaticamente pelo Delphi (num ficheiro com a extensão **dpr**), que se limita a arrancar as janelas (*forms*) associadas à aplicação. Todo o desenvolvimento de aplicações é feito em módulos (*units*) tipicamente associados a janelas (*forms*), guardados em ficheiros com a extensão **pas**).

A primeira parte do desenvolvimento consiste no abrir de uma aplicação (opção "New Application" dentro do menu "Main"), que abre duas janelas:

- uma *form* onde se irão colocar os componentes de biblioteca usados na aplicação, e no caso dos componentes gráficos, o seu aspecto
- uma janela de edição de texto, onde se declararão tipos, variáveis, procedimentos e funções auxiliares, e se editará o código de tratamento dos eventos.

Depois, deve-se puxar para dentro da *form* todos componentes que se pretende usar e editar as propriedades de cada um, seleccionando o componente e utilizando a janela "**Object Inspector**".

Existem algumas propriedades que são comuns à maior parte dos componentes:

- '**Name**' é o nome da variável que é gerada com o componente
- '**Caption**' é usado nos componentes gráficos (ex. botão, *form*) como o nome visível no écran

Estas propriedades definem o valor inicial dos parâmetros, quando a aplicação arranca. Posteriormente, durante o funcionamento das aplicações, estes valores podem ser modificados. Por exemplo, o texto de um botão pode mudar com a instrução {botao.Caption:= 'Abrir';}.

A janela "**Object Inspector**" também é utilizada para criar o protótipo das funções de tratamento de eventos. Ao se seleccionar a pasta "Events", pode-se consultar os eventos que o componente sabe tratar. Qualquer dúvida pode ser esclarecida seleccionando um evento e premindo a tecla "**F1**" (*help*). Ao se premir sobre um evento com duplo *click*, o Delphi cria o protótipo da rotina de processamento do evento, e coloca-a na janela de edição. O programador deve então escrever o código PASCAL referente ao processamento da rotina. Para auxiliar esta

tarefa, sempre que é introduzida uma referência para uma variável seguida de um ponto, o Delphi abre uma janela de ajuda com todas a propriedades e métodos (funções ou procedimentos associados ao objecto), e sempre que se está a preencher parâmetros de uma função, o Delphi mostra numa janela de ajuda o tipo pretendido.

Por fim, falta compilar e testar a aplicação. Para isso carregue na seta verde.

O desenvolvimento de uma aplicação consiste assim, na selecção do conjunto de tipos e variáveis necessárias para guardar o estado da aplicação entre o processamento dos vários eventos, dos componentes, e na escrita do código de processamento dos eventos.

3. Primeira Aplicação - Somadora

Pretende-se programar uma calculadora que aceite a introdução de números tanto por um conjunto de botões como através de um vector de caracteres directamente no écran da calculadora, que suporte um conjunto mínimo de operações ("+" e "="), que crie um registo de todas as operações numa janela de texto, e opcionalmente, num ficheiro.

Para desenvolver esta aplicação irão ser necessários vários tipos de componentes (seleccionados na barra de classes):

- 16 objectos **Button**  (da pasta '*Standard*') { para fazer os botões 0-9, +, =, C, CE, etc }
- um objecto **Label**  (da pasta '*Standard*') { para escrever o nome da aplicação }
- um objecto **Memo**  (da pasta '*Standard*') { Janela com o registo das operações }
- um objecto **Edit**  (da pasta '*Standard*') { Mostrador da calculadora }
- um objecto **SaveDialog**  (da pasta 'Dialogs') { Para pedir o nome do ficheiro a guardar }

Esta aplicação vai ser realizada em duas fases:

1. é realizada apenas uma funcionalidade mínima de entrada de números e soma.
2. é realizada a somadora completa com o registo de operações numa janela e num ficheiro.

3.1 Somadora simplificada

Nesta primeira fase vai ser realizada uma máquina apenas com as teclas '1', '2', 'C'(apagar último caracter), 'CE' (reiniciar), '+' e '=' (**6 objectos Button**), e um mostrador (**um objecto Edit**).

Comece por copiar estes seis objectos para a '*form*'. Em seguida modifique as seguintes propriedades:

- objecto 'Edit'
  - ✓ Name:= 'Total'; // Nome da variável Delphi
  - ✓ Text:= '0'; // String inicial apresentada quando o programa arranca
- Seis objectos 'Button'
  - ✓ Height:= 33;
  - ✓ Width:= 33;
  - ✓ Name:= 'Button1', 'Button2', 'ButtonC', 'ButtonCE', 'ButtonMais', 'ButtonIgual';
  - ✓ Caption:= '1', '2', 'C', 'CE', '+' e '=' respectivamente;

Os objectos do tipo 'Edit' têm a propriedade '**Text**' (uma 'String'), que vai ser utilizada para guardar o número que se está a compor. Para permitir a alteração directa no visor da

calculadora, não se tratam os eventos da classe ´Edit´, mas sempre que se toca numa tecla de operação, ou igual, é usado o valor que estiver no momento no visor.

Adicionalmente, são ainda necessárias: uma variável adicional para memorizar o operando da adição { operand : Integer }, e para preparar o programa para futuras operações; e uma variável para memorizar a operação pendente { oper : String }. Estas variáveis são declaradas precedidas pela palavra-chave **var** na secção '**interface**'[1] do ficheiro com a declaração da *unit* (associada à *form*).

Em seguida devem ser programadas as rotinas de tratamento dos eventos dos botões. O esqueleto da rotina é criado automaticamente no Delphi, com um duplo click no evento correspondente, na janela "Object Inspector". Por exemplo, para o botão 1 seria qualquer coisa deste género:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Total.Text = '0' then { se tem um 0, apaga-o }
    Total.Text:= '1'
  else
    Total.Text:= Total.Text + '1';
end;
```

As rotinas dos outros botões com números são semelhantes.

```
procedure TForm1.ButtonCClick(Sender: TObject);
var
  aux : string;
begin
  if Length(Total.Text) > 0 then
  begin
    aux:= Total.Text;
    SetLength(aux, Length(aux)-1);  { Trunca último caracter }
    Total.Text:= aux;
  end;
end;
```

```
procedure TForm1.ButtonCEClick(Sender: TObject);
begin
  oper:= '';
  operand:= 0;
  Total.text:= '0';
end;
```

[1] Outra alternativa seria declarar as variáveis na secção '**implementation**' do mesmo ficheiro. A diferença é a visibilidade das variáveis: as que são declaradas na secção 'interface' são visíveis para outras unidades. Neste caso é indiferente, pois só se vai utilizar uma unidade.

```
procedure TForm1.ButtonIgualClick(Sender: TObject);
var
  valor: Integer;
begin
  { Converte 'String' para 'Integer' }
  valor:= StrToInt(Total.Text);
  if oper = '+' then
    valor:= valor+operand;
  operand:= 0;
  oper:= '=';
  { Escreve total parcial }
  Total.text:= IntToStr(valor);
end;
```

```
procedure TForm1.ButtonMaisClick(Sender: TObject);
var
  valor: Integer;
begin
  { Converte 'String' para 'Integer' }
  valor:= StrToInt(Total.Text);
  if oper = '+' then  { Se já havia uma operação pendente }
    valor:= valor+operand;
  { memoriza operando }
  operand:= valor;
  oper:= '+';
  { Limpa mostrador para novo número }
  Total.Text:= '0';
end;
```

Este exemplo recorre a algumas funções úteis da biblioteca do Delphi:

- **IntToStr** permite converter inteiros para 'String'. Também poderia ser usada a função 'Format', que faz a conversão de qualquer formato numérico.
- **StrToInt** faz a conversão de 'String' para 'Integer'
- **Length** retorna a dimensão de uma 'String' ou array de tamanho variável
- **SetLength** permite modificar o tamanho de uma 'String' ou array variável.

Pode ainda verificar que pode editar directamente o número no mostrador, permitindo assim fazer somas com números inteiros arbitrários.

3.1.1 Exercício

Complete a código anterior com as teclas numéricas '3' a '9' e '0'.

3.2 Somadora Completa

Nesta segunda fase vão ser adicionadas uma janela para manter um registo de todas as operações e a escrita opcional deste registo num ficheiro.

Vai ser necessário adicionar um objecto 'Memo', um objecto 'SaveDialog', um objecto 'Label' e dois objectos 'Button', que vão ser utilizados para controlar a escrita do ficheiro e a limpeza da janela de registo. Em seguida modifique as seguintes propriedades:

- objecto 'Memo1'
  - ✓ ScrollBars:= ssVertical;
  - ✓ Editar a propriedade 'Lines' e apagar a primeira linha.

- objecto 'Label'
  - ✓ Caption:= 'Somadora – Grupo **??**'; // preencher com o número do grupo
- Dois objectos 'Button'
  - ✓ Height:= 25;
  - ✓ Width:= 75;
  - ✓ Name:= 'Limpar' e 'Gravar';
  - ✓ Caption:= 'LimparMemo' e 'GravarFicheiro' respectivamente;

As rotinas programadas anteriormente vão ser modificadas para se guardar um registo das operações. Vão ser necessárias mais algumas variáveis que devem ser declaradas juntamente com as anteriores:

{ Aberto : Boolean; } TRUE se o ficheiro está aberto

{ Logf : Text; } Para escrever no ficheiro de texto.

Para facilitar a programação vai ser declarada uma função auxiliar '*Log*' na classe da aplicação ('Tform1'). Esta declaração tem de ser feita em dois locais. Na secção '**interface**' deve ser declarada a assinatura da função dentro da zona '**private**' (privada) (também podia ser na zona 'public' (pública)):

```
procedure Log(txt : String);
```

Na secção '**implementation**' é declarado o código:

```
procedure TForm1.Log(txt : String);
begin
  // Escreve uma nova linha na janela 'Memo'
  Memo1.Lines.Add(txt);
  // Escreve no ficheiro se este estiver aberto
  if Aberto then
    Writeln(Logf, txt);
End;
```

O novo código dos botões '+', '=' e 'CE' será:

```
procedure TForm1.ButtonMaisClick(Sender: TObject);
var
  valor: Integer;
begin
  { Converte 'String' para 'Integer' }
  valor:= StrToInt(Total.Text);
  if oper <> '=' then
    Log(IntToStr(valor));
  if oper = '+' then  { Se já havia uma operação pendente }
    valor:= valor+operand;
  { memoriza operando }
  operand:= valor;
  oper:= '+';
  Log('+');
  { Limpa mostrador para novo número }
  Total.Text:= '0';
end;
```

```
procedure TForm1.ButtonIgualClick(Sender: TObject);
var
  valor: Integer;
begin
  { Converte 'String' para 'Integer' }
  valor:= StrToInt(Total.Text);
  if oper <> '=' then
    Log(IntToStr(valor));
  if oper = '+' then
    valor:= valor+operand;
  operand:= 0;
  oper:= '=';
  { Escreve total parcial }
  Log('=');
  Log(IntToStr(valor));
Total.text:= IntToStr(valor);
end;
```

```
procedure TForm1.ButtonCEClick(Sender: TObject);
begin
  oper:= '';
  operand:= 0;
  Log('0');
  Total.text:= '0';
end;
```

O botão 'Gravar' vai controlar se se começa ou pára de gravar. O texto do botão vai ser modificado, e o ficheiro vai ser aberto ou fechado, conforme o estado do programa. O objecto 'SaveDialog', da pasta 'Dialogs', é utilizado para escolher o nome do ficheiro a gravar. Seleccione-o e coloque-o algures na *form* da somadora:

```
procedure TForm1.GravarClick(Sender: TObject);
begin
  if Aberto then
  begin
  { Fechar ficheiro }
    CloseFile(Logf);
    Aberto:= FALSE;
    Gravar.Caption:= 'Gravar Ficheiro'; // novo nome botão
  end else
  begin
  { Gravar o conteúdo do memo para ficheiro }
    if SaveDialog1.Execute{ abre a janela para obter o nome }then
    begin
      AssignFile(Logf, SaveDialog1.FileName);
      Rewrite(Logf); { Ficheiro aberto para escrita }
      Aberto:= TRUE;
      Gravar.Caption:= 'Fechar Ficheiro'; // novo nome botão
    end;
  end;
end;
```

O botão 'LimparMemo' limpa o conteúdo da janela. Este botão é necessário pois as janelas 'Memo' têm um número de linhas limitado. Se ele for excedido deixam de aceitar novas linhas.

```
procedure TForm1.LimparClick(Sender: TObject);
begin
  Memo1.Clear;
end;
```

Falta ainda inicializar o valor da variável Aberto a FALSE numa secção **Initialization**, que deve ser criada imediatamente antes do 'END.' final do ficheiro de definição do módulo (.pas):

```
Initialization
  Aberto:= FALSE;
```

3.2.1 Exercícios

1) Acrescente uma janela 'Edit' ou 'Label' onde escreva o nome do ficheiro aberto.

2) Acrescente um temporizador (objecto 'Timer' da para 'System') de maneira a que apareça automaticamente o resultado, caso o utilizador não introduza nenhuma operação durante três segundos. Sugestão: Programe o evento associado ao 'Timer' de forma semelhante à operação '='.

3) Complete a código anterior com a operação ''. Sugestão: Adicione um 'operando e operação' adicional de modo a suportar precedências (3+3*3= 3+(3*3)= 12).*

4. Segunda Aplicação – Conversa em Rede

Esta secção ilustra o desenvolvimento de uma aplicação utilizando sockets datagrama e orientados à ligação. A aplicação suporta uma conversa em rede, onde cada participante tem uma janela onde recebe mensagens de outros elementos (*Remote*) e uma janela onde escreve as suas mensagens (*Local*). Num menu pode seleccionar a máquina para onde envia as mensagens e pode desligar a aplicação. Por fim, numa barra de estado, a aplicação indica a máquina para onde está a enviar as mensagens.

4.1 Realização utilizando sockets datagrama

Os sockets datagrama estão permanentemente ligados. Nesta realização é desenvolvida uma aplicação que está permanentemente disponível para enviar e receber mensagens de qualquer computador da rede.

O desenvolvimento desta aplicação no ambiente Delphi 4 é simples graças à biblioteca de classes disponível. Assim, a aplicação pode ser realizada utilizando:

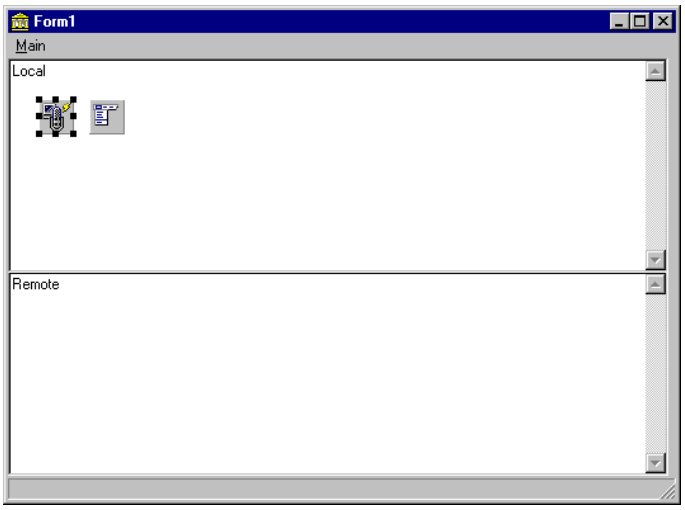
- um objecto **NMUDP** (da pasta '*Internet*') { socket UDP }
- um objecto **MainMenu** (da pasta '*Standard*') { Menu principal }
- dois objectos **Memo** (da pasta '*Standard*') { Janelas de entrada/saída de mensagens }
- um objecto **StatusBar** (da pasta '*Win32*') { Indicador de estado }

Utilizando a janela 'Object Inspector' deve ser modificado:

- objecto 'MainMenu'
  - ✓ acrescentar coluna '*&Main*' com as opções '*&Ligar*' e '*&Sair*'.

- objecto 'Memo1'
  - ✓ Name:= '**Remote**'; { Nome da variável }
  - ✓ ReadOnly:= TRUE;
  - ✓ ScrollBars:= ssVertical.
- objecto 'Memo2'
  - ✓ Name:= '**Local**'; { Nome da variável }
  - ✓ ScrollBars:= ssVertical.
- objecto 'NMUDP1'
  - ✓ LocalPort:= 1024;
  - ✓ RemotePort:= 1024;
  - ✓ RemoteHost:= '127.1.1.1'. { Por defeito envia para ele próprio }
- objecto 'StatusBar1'
  - ✓ acrescentar 'Panel 0' (na propriedade 'Panels');

A disposição dos objectos na janela "Form1" deve ser a representada na figura seguinte:



A aplicação vai ser realizada programando quatro rotinas de tratamento de eventos (NOTA 1):

- as duas rotinas de tratamento das opções do menu;
- uma rotina de processamento do evento '*OnKeyDown*' no objecto 'Local' (para processar a entrada de teclas);
- uma rotina de processamento do evento '*OnDataReceived*' no socket UDP (para escrever no écran as mensagens recebidas de outros utilizadores).

NOTA 1: **Crie os esqueletos das rotinas com um duplo *click* sobre o evento na janela "Object Inspector", e preencha o código em seguida. Se não fizer a associação das rotinas aos eventos, a aplicação não funciona correctamente.**

NOTA 2: **Não se esqueça de guardar a aplicação numa directoria sua (/user/…) no disco rígido, antes de testar o programa. Não grave os seus ficheiros na directoria raiz do Delphi, usada por defeito quando se manda guardar um projecto.**

O código é programado no ficheiro ‘unit1.pas’:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
     Dialogs, Menus, ComCtrls, StdCtrls, NMUDP;

type
  TForm1 = class(TForm)
    NMUDP1: TNMUDP;
    MainMenu1: TMainMenu;
    Remote: TMemo;
    Local: TMemo;
    StatusBar1: TStatusBar;
    Man1: TMenuItem;
    Bind1: TMenuItem;
    N1: TMenuItem;
    Exit1: TMenuItem;
    procedure Bind1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure LocalKeyDown(Sender: TObject; var Key: Word;
       Shift: TShiftState);
    procedure NMUDP1DataReceived(Sender: TComponent;
       NumberBytes: Integer; FromIP: String);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  server : String; { Destino para onde são enviadas as mensagens }

implementation

{$R *.DFM}

procedure TForm1.Ligar1Click(Sender: TObject);
begin
  If InputQuery('Máquina a ligar', 'Endereço:', server) then
  begin
    NMUDP1.RemoteHost:= server;
    StatusBar1.Panels[0].Text := 'Sending to ' + server;
  end;
end;

procedure TForm1.Sair1Click(Sender: TObject);
begin
  close; { Fecha a janela e logo a aplicação }
end;
```

```
procedure TForm1.LocalKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  buf: TMemoryStream;
  data: TDateTime;
  aux : String;
begin
  if Key = VK_Return then
  begin  { Fim de Linha }
    if Length(server)>0 then
    begin                                    { server está definido }
      aux:= Local.Lines[Local.Lines.Count-1]; { última linha écran }
      data:= Now;                                  { data actual }
      buf := TMemoryStream.Create;    { inicializa buffer de envio }
          { preenche o início da mensagem com a data }
      buf.Write(data, sizeof(data));
          { preenche resto da mensagem com string }
      buf.Write(aux[1], Length(aux));
      NMUDP1.SendStream(buf);       { envia buffer para destino }
      buf.Free;          { liberta memória utilizada pelo buffer }
    end else
      StatusBar1.Panels[0].Text := 'server not initialized';
  end;
end;

procedure TForm1.NMUDP1DataReceived(Sender: TComponent;
  NumberBytes: Integer; FromIP: String);
var
  buf: TMemoryStream;
  aux: String;
  data: TDateTime;
begin
  buf := TMemoryStream.Create;    { Inicializa buffer ler mensagem }
  NMUDP1.ReadStream(buf);           { Lê mensagem para buffer }
          { Lê primeiro campo da mensagem para variável data }
  buf.Read(data, sizeof(data));
  SetLength(aux, NumberBytes-sizeof(data)); {Aloca espaço string }
  buf.Read(aux[1], NumberBytes-sizeof(data));{Lê string p/buffer }
        { Escreve mensagem recebida indicando o endereço IP e a hora }
  Remote.Lines.Add(FromIP + ' ; ' + FormatDateTime('h:n:s',data)
                                  + ' : ' + aux);
  buf.Free;
end;

initialization
  { Inicialização da variável server }
  server:= '127.1.1.1';
end.
```

O programa principal (com a função '*main*') é gerado automaticamente (Project1.dpr).

4.1.1 Exercícios

1) *Acrescente uma janela 'Edit' ou 'Label' onde escreva o endereço IP para onde está a enviar as strings.*

2) *Acrecente a funcionalidade de enviar as mensagens para todos os endereços 172.16.54.1 a 172.16.54.10 (quando um objecto do tipo 'CheckBox' (na janela Standard) estiver seleccionado).* ***Sugestão:*** *estes endereços podem ser obtidos como a soma da string '172.16.54.' com a conversão para string de um inteiro.*

3) *Se se utilizar sempre o mesmo porto, só pode correr um executável por cada máquina. Modifique o programa anterior de maneira a poder modificar o porto local e o porto remoto.* ***Sugestão:*** *acrescente mais duas janelas com os portos local e de destino e um botão para modificar o endereço local.*

4.2 Realização utilizando sockets orientados à ligação

Nesta segunda realização vão ser estabelecidas ligações temporárias entre pares de servidores. Vão ser necessários dois objectos, um para iniciar novas ligações (cliente) e outro para receber ligações (servidor).

O desenvolvimento da aplicação pode ser feito modificando a aplicação com sockets UDP, seguindo os vários passos indicados.

1) Comece por guardar o programa 'UDPchat' com outro nome (TCPchat.dpr e chatTCP.pas, por exemplo) na vossa directoria.

2) Retire o objecto NMUDP e substitua-o por dois objectos novos:

- um objecto **ClientSocket** (da pasta '*Internet*')
- um objecto **ServerSocket** (da pasta '*Internet*')

3) Utilizando a janela 'Object Inspector' deve modificar as propriedades:

- objecto 'ClientSocket'
  - ✓ Port:= 1024; { Porto a que se vai tentar ligar }
  - ✓ Address:= '172.16.54.1'. { Por defeito envia para o computador 1 }
- objecto 'ServerSocket'
  - ✓ Port:= 1024; { Porto com que se vai registar }
  - ✓ Active:= True; { Arranca preparado para receber ligações }

4) Para além da variável '**server**' vai ser necessário declarar variáveis para saber se se está ligado a outro programa (*sending*) e se recebeu a ligação ou estabeleceu a ligação (*is_server*):

```
var
  Form1: TForm1;
  server: String;
  sending: Boolean;
  is_server: Boolean;
```

5) A secção de inicialização deve ter um valor inicial para '*sending*':

```
sending:= FALSE;
server:= '172.16.54.1';
```

A aplicação vai ser realizada modificando e acrescentando opções ao menu, modificando a rotina de processamento das teclas, e programando um conjunto de eventos associados à criação, recepção de dados e destruição das ligações dos sockets.

6) O menu deve ter duas opções adicionais: '**&Receber**' e '**&Desligar**'.

7) Na opção '*Receber*' (mapeada no objecto '*Receber1*') deve modificar o valor da propriedade '***Checked***' para **True**, aparecendo uma marca que simboliza que o servidor está preparado para receber ligações. O código associado a esta opção será:

```
procedure TForm1.Receber1Click(Sender: TObject);
begin
  if not Receber1.Checked then
  begin
    ServerSocket1.active:= TRUE; // socket aceita ligações
    Receber1.Checked:= TRUE;
  end else
  begin
    serversocket1.active:= FALSE; // socket não aceita ligações
    receber1.checked:= FALSE;
    if sending and is_server then
    begin   { se estava ligado, a ligação foi desligada }
      sending:= FALSE;
    end
  end;
end;
```

8) A opção '*Desligar*' deverá terminar qualquer ligação que esteja activa:

```
procedure TForm1.Desligar1Click(Sender: TObject);
begin
  if sending then
  begin
    if is_server then { Desliga a primeira (e única) ligação }
      ServerSocket1.Socket.Connections[0].Close
    else   { Desliga ligação do cliente }
      ClientSocket1.Close;
    sending:= FALSE;
    StatusBar1.Panels[0].Text := 'Desligado';
  end
end;
```

9) A opção '*Ligar*' deve ser modificada para:

```
procedure TForm1.Ligar1Click(Sender: TObject);
begin
  if not sending then
  begin
    If InputQuery('Máquina a ligar', 'Endereço:', server) then
    begin
      ClientSocket1.Address:= server;  { Define máquina a ligar }
      ClientSocket1.active:= TRUE;  { Tenta estabelecer Ligação }
    end;
  end;
end;
```

10) A última opção ('*Sair*') também deve modificada para fechar os dois sockets quando se fecha a aplicação:

```
procedure TForm1.Sair1Click(Sender: TObject);
begin
  ClientSocket1.close;
  ServerSocket1.close;
  close;
end;
```

11) A rotina de processamento do evento '*OnKeyDown*' no objecto 'Local' (para processar a entrada de teclas) deve ser modificada para enviar a última linha escrita pelo 'ClientSocket' ou 'ServerSocket', conforme o valor de '*is_server*':

```
procedure TForm1.LocalKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = VK_Return then
  begin
    if Sending then
    begin
      if is_server then
        ServerSocket1.Socket.Connections[0].SendText(
             Local.Lines[Local.Lines.Count-1])
      else
        ClientSocket1.Socket.SendText(
             Local.Lines[Local.Lines.Count-1]);
    end else
      StatusBar1.Panels[0].Text := 'server not connected';
  end;
end;
```

Note-se que neste caso, o socket orientando à ligação suporta o envio de sequência de caracteres, não sendo necessária a utilização de um tampão de memória auxiliar.

O objecto 'ServerSocket' vai suportar a recepção de ligações a partir de objectos 'ClientSocket'. Vai ser necessário programar rotinas para tratar três eventos:

12) Rotina de processamento do evento '*OnClientDisconnect*' do objecto '*ServerSocket*', activado quando termina uma ligação:

```
procedure TForm1.ServerSocket1ClientDisconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  if sending AND is_server AND (Socket.SocketHandle =
      ServerSocket1.Socket.Connections[0].SocketHandle) then
  begin   { Terminou a ligação 0 }
    sending:= FALSE;
    Local.Lines.Add('******** Fim de Ligação ********');
    Remote.Lines.Add('******** Fim de Ligação ********');
    StatusBar1.Panels[0].Text := 'Desligado';
  end;
end;
```

13) Rotina de processamento do evento '*OnClientRead*' do objecto '*ServerSocket*', activado quando há dados disponíveis para leitura na ligação definida pelo argumento'*Socket*':

```
procedure TForm1.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
var
  buf: String;
Begin
  buf:= Socket.ReceiveText;
  Remote.Lines.Add(buf);
end;
```

14) Rotina de processamento do evento '*OnAccept*' do objecto '*ServerSocket*', activado quando por cada 'ClientSocket' que se tente ligar:

```
procedure TForm1.ServerSocket1Accept(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  if not sending then
  begin
    StatusBar1.Panels[0].Text := 'Recebeu ligação do endereço ' +
        socket.RemoteAddress;
    Remote.Lines.Add('******** Ligação a ' + socket.RemoteAddress
        + ' ********');
    Local.Lines.Add('******** Ligação a ' + socket.RemoteAddress
        + ' ********');
    is_server:= TRUE;
    sending:= TRUE;
  end else begin
    Local.Lines.Add('******** Rejeitou ligação de ' +
       socket.RemoteAddress + ' ********');
    socket.close; // se já está ligado desliga nova ligação
  end
end;
```

Embora este objecto suporte várias ligações activas, só se vai deixar haver uma activa em cada instante.

O objecto 'ClientSocket' vai suportar a criação de uma ligação para um destino. A programação do objecto é semelhante ao anterior. Vai ser necessário programar rotinas para tratar três eventos, mais um adicional, para evitar o aparecimento de uma janela de erro quando falha um estabelecimento de ligação.

15) Rotina de processamento do evento '*OnConnect*' do objecto '*ClientSocket*', activado quando uma ligação é estabelecida:

```
procedure TForm1.ClientSocket1Connect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  sending:= TRUE;
  is_server:= FALSE;
  StatusBar1.Panels[0].Text := 'Ligou-se ao endereço ' + server;
  Remote.Lines.Add('******** Ligação a ' + server + ' ********');
  Local.Lines.Add('******** Ligação a ' + server + ' ********');
end;
```

16) Rotina de processamento do evento '*OnRead*' do objecto '*ClientSocket*', activado quando há dados disponíveis para leitura na ligação definida pelo argumento'*Socket*':

```
procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
begin
    EXERCÍCIO:  COMPLETE ESTE CÓDIGO BASEADO NO PASSO 13 …
end;
```

17) Rotina de processamento do evento '*OnDisconnect*' do objecto '*ClientSocket*', activado quando termina uma ligação:

```
procedure TForm1.ClientSocket1Disconnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  if sending and not is_server then
  begin
    sending:= FALSE;
    StatusBar1.Panels[0].Text := 'Desligado';
    Remote.Lines.Add('******** Fim de Ligação ********');
  end;
end;
```

18) Rotina de processamento do evento '*OnError*' do objecto '*ClientSocket*', activado quando uma tentativa de ligação falha, ou se tentar ler ou escrever por uma ligação inválida:

```
procedure TForm1.ClientSocket1Error(Sender: TObject;
  Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
  var ErrorCode: Integer);
begin
  if Socket.Connected then
    ClientSocket1.close  // Falhou ligação, desliga-a
  else begin
    Local.Lines.Add('Error connecting to : ' + Server);
    ErrorCode := 0; // Evita o aparecimento de uma janela de erro
  end;
end;
```

19) Guarde a aplicação no disco, compile e corra-a comunicando com outra aplicação a correr noutro computador (o programa não corre na mesma máquina, porque isso obrigaria a ter os objectos 'ClientSocket' e 'ServerSocket' activos simultaneamente).

4.2.1 Exercícios

1) *Modifique a mensagem trocada entre aplicações, acrescentando o tempo de envio.* ***Sugestão:*** *Envie a uma sequência de três campos, o tempo, o tamanho da mensagem e o conteúdo da mensagem, e na recepção, leia os três campos na mesma sequência.*

   Explique o porquê de ser necessário o campo 'comprimento da mensagem'.

2) *Modifique o programam anterior de modo a suportar várias ligações activas em paralelo.* ***Sugestão:*** *Tem de passar a enviar a mensagem para o cliente e para todas a ligações de servidor activo, e retirar o código no servidor que rejeita novas ligações quando já está ligado.*

3) *Se se utilizar sempre o mesmo porto, só pode correr um executável por cada máquina. Modifique o programa anterior de maneira a poder modificar o porto local e o porto remoto.* ***Sugestão:*** *acrescente janelas com os portos local e de destino, e um botão para modificar o endereço local.*