FOURTEEN

Developing a database (2)

In this second chapter on the Estate Agent's Database project, we will complete the '*house records*' section, then work on the '*customer records*' options:

Begin by loading your project from the ESTATE sub-directory.

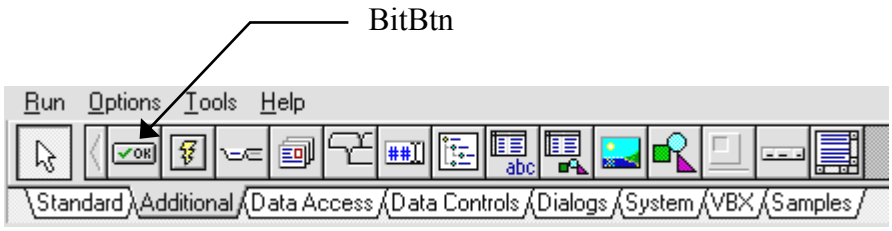
The existing program is not very well error trapped. For example, it is possible to accidentally click on the '**Set up new file**' option and lose all the house records previously entered. Let's do something about this...

Use the 'New form' short-cut button, and select 'Blank form' Form5 will be created. Press ENTER to bring up the Object Inspector and set the properties:

BorderStyle:	**Dialog**
Caption:	**Set up new file**
FormStyle:	**StayOnTop**

Place a small *Image Box* on the left of the Form and load the file QUEST.BMP. This will display the 'question mark' icon. Add two *Labels* with the captions: '**Confirm to set up new file**' and '**WARNING: Existing data will be lost**'.

Select the Bit Button component from the ADDITIONAL menu:



Bit Buttons differ from the standard *Button* component by allowing bitmaps to be displayed - in this case the *tick* and *cross* icons.

Place two *Bit Buttons* on the Form. Select the left button and press ENTER to bring up the Object Inspector. Set the **Caption** to '**OK**'. Double-click the right hand column for the **Glyph** property, then load the file YES.BMP.

Give the right hand *Bit Button* the caption '**cancel**', and load the graphics file NO.BMP.

Double-click the '**cancel**' button and add a line to the event handler to close the window:

```
procedure TForm5.BitBtn2Click(Sender: TObject);
begin
  form5.visible:=false;
end;
```

Double-click the '**OK**' button and add lines to create the new file on disc:

```
procedure TForm5.BitBtn1Click(Sender: TObject);
begin
  with form1 do
  begin
    assignfile(housefile,'houses.dat');
    rewrite(housefile);
    closefile(housefile);
  end;
  form5.visible:=false;
end;
```

This requires variables belonging to Form1, so add a '**uses**' instruction under the **implementation** heading:

```
implementation
{$R *.DFM}
uses
  unit1;
```

It is now necessary to link Form5 into the program so that it can be opened from the Main Menu.

Go to the *Unit1* program and add **Unit5** to the '**uses**' list:

```
uses
  SysUtils,.... Menus, Unit2, Unit3, Unit5;
```

Go to the *Form1* window and click '**House records/Set up new file**' to open the event handler procedure. Delete the lines which set up a new file and replace these with the *Form5* command:

```
procedure TForm1.Setupnewfile1Click
                              (Sender: TObject);
begin
  form5.visible:=true;
end;
```

Build and run the program. Begin by selecting the '**Set up new file**' option. Click the '**OK**' button. From the Main Menu, now select the '**Display house index**' option; there should be no houses listed, showing that a new file has been created.

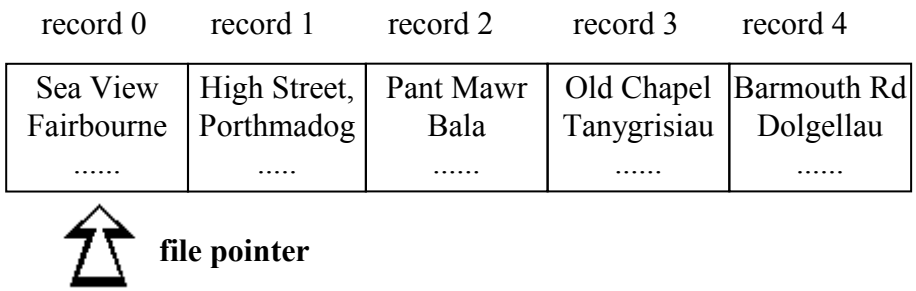
Enter the first of the properties from the test data: *Sea View, Fairbourne*. Use the '**Display house index**' option to check that this has been saved on disc correctly. Now select '**Set up new file**', but this time click the '**cancel**' button. The house file should not have been affected - check that the record for *Sea View* is still showing in the list of houses, then return to the Delphi editing screen.

We still have to complete the EDIT and DELETE functions for customer records. Both will require the '**seek**' command, so we need to be sure we can use this correctly.

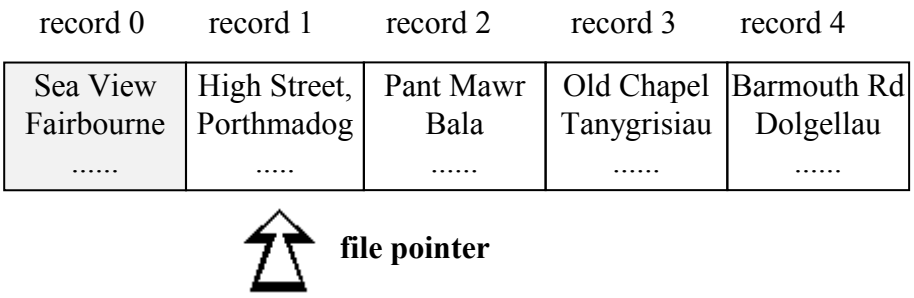
The SEEK command

SEEK is a very important command which allows us to pick out a particular record from a file.

The records we are storing are all the same size, and are arranged in a file using a numbering system which begins at 0. Let us assume that the first five houses have been entered so far:



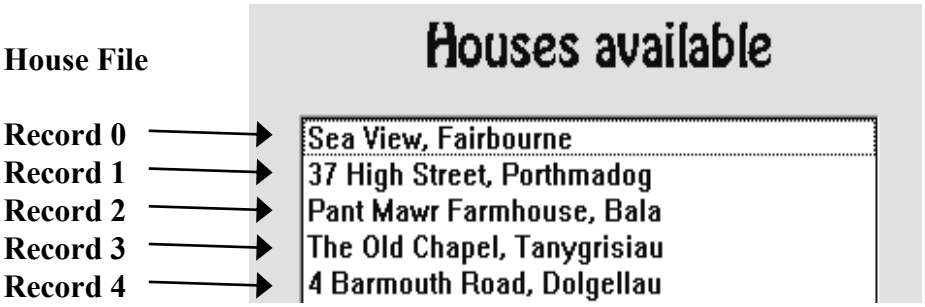
When the file is *opened* ready to read records, the computer sets up a '**file pointer**'. Initially this points to the first record in **position 0**. When the first record has been read, the file pointer moves to the record in **position 1**:



Each time a record is read into the computer, the file pointer moves forwards by one position until it reaches the end of the file.

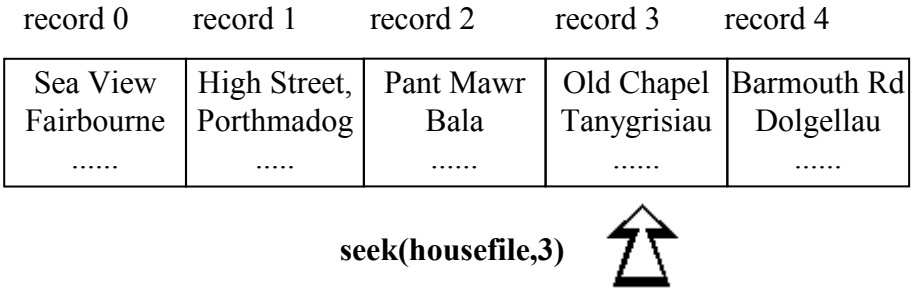
Suppose that we didn't want to read every record, but instead wanted to pick out just one particular record - for example, *The Old Chapel, Tanygrisiau*.

The '***Houses available***' list shows the order in which records are stored:



The *Old Chapel* is record number 3 in the file.

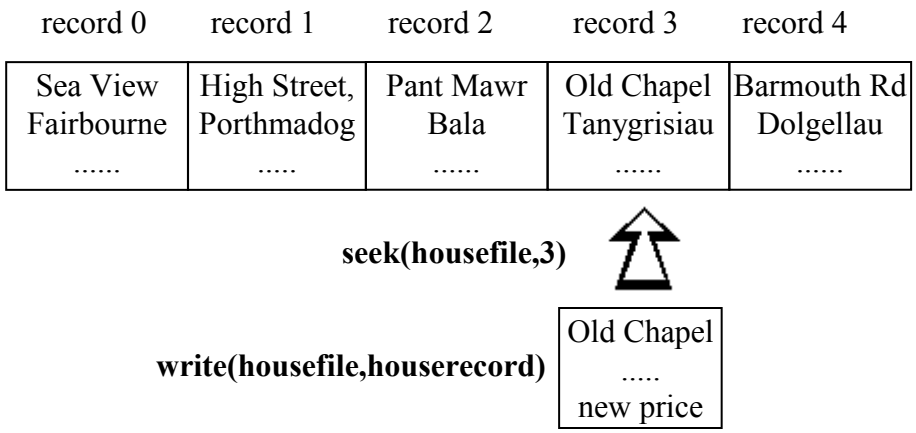
Immediately the file is opened we can use the SEEK command to move the file pointer directly to **position 3:**



The record which will now be read into the computer is *The Old Chapel.*

The same method can be used if we wish to save a record back into a particular position inside the file, rather than just adding it to the end of the current records.

For example, suppose we have read in the record for the *Old Chapel* and have made an alteration to the price. We now need to save the amended record back into the same position in the file. To do this, use the SEEK command to move the file pointer to **position 3**:



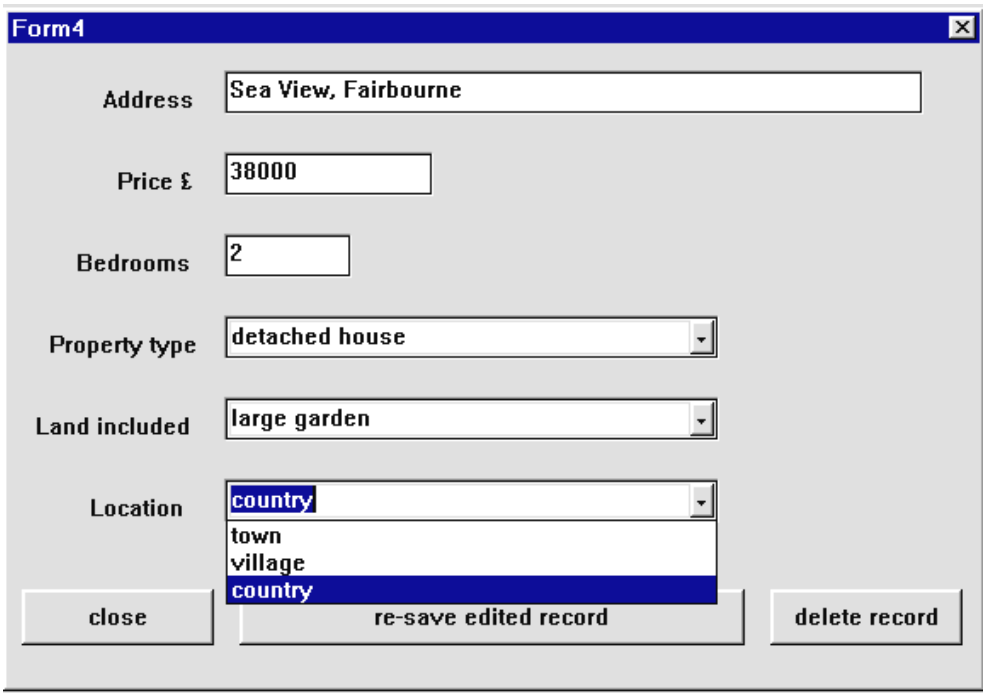
The amended record can then be saved back into the file using the '**write**' command.

Let's do this now in the procedure for editing house records. Go to the **Form4** window and double-click the '**re-save edited record**' button to create an event handler. Add the lines:

```
procedure TForm4.Button2Click(Sender: TObject);
begin
  form1.houserecord.address:=edit1.text;
  form1.houserecord.price:=strtofloat(edit2.text);
  form1.houserecord.bedrooms:=strtoint(edit3.text);
  form1.houserecord.housetype:=combobox1.itemindex;
  form1.houserecord.land:=combobox2.itemindex;
  form1.houserecord.location:=combobox3.itemindex;
  with form1 do
  begin
    assignfile(housefile,'houses.dat');
    reset(housefile);
    seek(housefile,form3.filepointer);
    write(housefile,houserecord);
    closefile(housefile);
  end;
  form4.visible:=false;
  form3.visible:=true;
```

```
end;
```

Build and run the program. Select '**House records/Display house index**', then click '*Sea View, Fairbourne*' to display the record:



By clicking the small arrows at the right hand end of each *ComboBox*, drop-down menus appear. Click on '**detached house**', '**large garden**' and '**country**'. Also change the price and number of bedrooms.

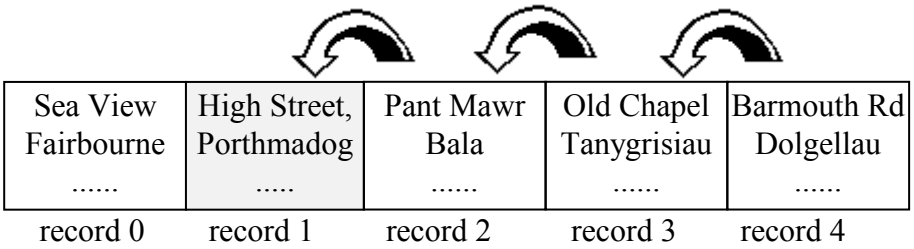
Click the '**re-save edited record**' button. The program returns to the '**Houses available**' list. If you again select '*Sea View, Fairbourne*', the amended record should appear.

The final section we need to complete is: '**delete record**'. Before doing that, please enter some more test data. Use the '**Add house record**' menu option to input the next four houses from the list on page 244. Check that these are saved correctly using the '**Display house index**' option.

Suppose that the house in *High Street, Porthmadog* has now been sold and we wish to delete the record. This would leave a gap in the file:

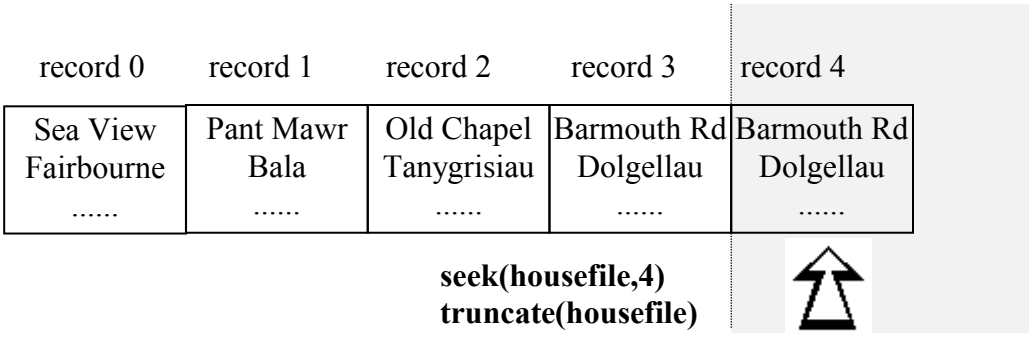
record 0	record 1	record 2	record 3	record 4
Sea View Fairbourne	High Street, Porthmadog	Pant Mawr Bala	Old Chapel Tanygrisiau	Barmouth Rd Dolgellau .....

To avoid a gap, we then copy each of the following records back one place:



The only problem is that this leaves two copies of the last record:

We move the file pointer to the last record. The TRUNCATE command is then used to cut off the file just before the file pointer - at position 3:



Go to the **Form4** screen and double-click the '**delete record**' button to create an event handler. Add the lines:

```
procedure TForm4.Button3Click(Sender: TObject);
var
  i:integer;
begin
  with form1 do
  begin
    assignfile(housefile,'houses.dat');
```

```
    reset(housefile);


    for i:=form3.filepointer+1 to
                          filesize(housefile)-1 do
    begin
      seek(housefile,i);
      read(housefile,houserecord);
      seek(housefile,i-1);
      write(housefile,houserecord);
     end;
     seek(housefile,filesize(housefile)-1);
     truncate(housefile);
     closefile(housefile);
  end;
  form4.visible:=false;
  form3.visible:=true;
end;
```

The procedure begins by opening the house file:

```
assignfile(housefile,'houses.dat');
reset(housefile);
```

We now use a loop. Each record following the one to be deleted will be copied backwards by one position:

```
for i:= form3.filepointer+1 to  filesize(housefile)-1 do
begin
    {copy each record back one position}
 end;
```

Remember that the variable '**filepointer**' specifies the position of the record currently being displayed, and this is the one we have decided to delete.

The actual copying makes use of the SEEK command. Each record is read into the computer, then the file pointer is moved back so that the record can be re-saved one position earlier in the file:

```
seek(housefile, i );
read(housefile,houserecord);
seek(housefile, i-1 );
write(housefile,houserecord);
```

The final step is to chop off the duplicated record at the end of the file:

```
seek(housefile,filesize(housefile)-1);
truncate(housefile);
```

This has been quite complicated, so Build and Run the program to convince yourself that it works!

Select '**House records/Display house index**', then select '*37, High Street, Porthmadog*'. When details are displayed, press the '**delete record**' button. Check that the record has disappeared from the list of houses without affecting any of the other entries.

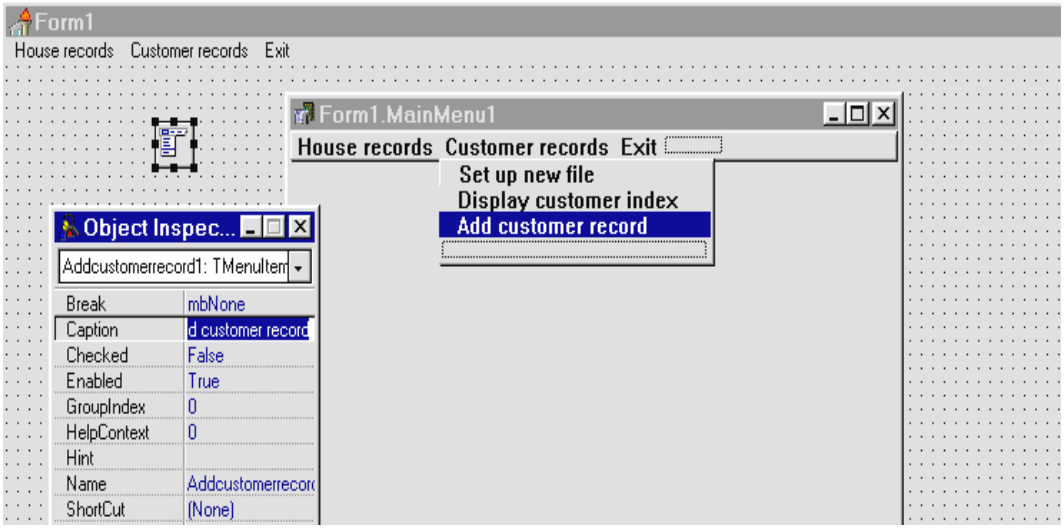
Try adding and deleting houses until you are sure that the program works correctly, then return to the Delphi editing screen.

{Note: It would be a good idea to add an error trapping window, similar to the one for 'Set up new file', to check that the user really did want to delete the record. Do this later if you have time.}

Customer records

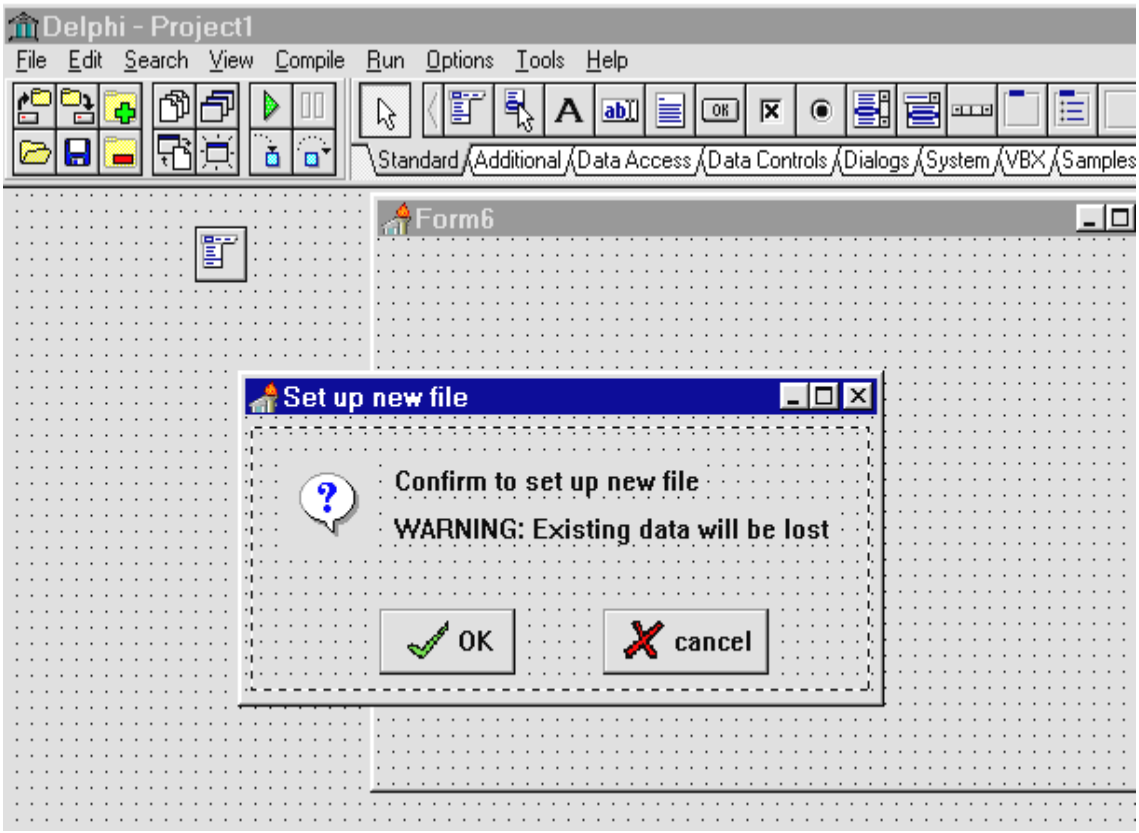
We can now move on to the *Customer records* section of the program. This will operate in a very similar way to the *House records* section, so we will follow the same sequence in setting up the program:

Begin by adding the *Customer records* menu options. Double-click the Main Menu icon on Form1 to open the menu editor window, then produce a drop-down menu for *Customer records* by adding the three options: '**Set up new file**', '**Display customer index**', and '**Add customer record**':



Build and run the program to check that the drop-down menu appears correctly, then return to the Delphi editing screen.
We next need to set up an error trapping window for the '**New file**' option. Click the 'New form' short-cut button to create **Form6**.

A quick way to set up **Form6** is to copy the components from **Form5**. To do this, open the *Form5* window:



Take the mouse pointer to *Form5*, near the top left-hand corner of the dotted grid. With the mouse button held down, move downwards and to the right until a dotted rectangle encloses the *image box*, *labels* and *bit buttons*. Release the mouse button, then select **Edit/Copy** from the top menu line of the Delphi screen.

Click on the **Form6** grid to bring this to the front, then select **Edit/Paste**. A copy of components will be transferred to *Form6*. Adjust the edges of the Form to a suitable size, and use the Object Inspector to set the properties:

BorderStyle:	**Dialog**
Caption:	**Set up new file**
FormStyle:	**StayOnTop**

Create an event handler for the '**cancel**' button and add the line:

```
procedure TForm6.BitBtn2Click(Sender: TObject);
begin
  form6.visible:=false;
end;
```

Double-click the '**OK**' button and add lines to the procedure to create a file '**custom.dat**':

```
procedure TForm6.BitBtn1Click(Sender: TObject);
begin
  with form1 do
  begin
    assignfile(customfile,'custom.dat');
    rewrite(customfile);
    closefile(customfile);
  end;
  form6.visible:=false;
end;
```

Add a '**uses**' instruction below the implementation heading:

```
implementation
{$R *.DFM}
uses
  unit1;
```

We now need to set up the customer record and file structures. Bring the Unit1 program window to the front and add lines below the '**type**' heading:

```
type
  customer=record
    name:string[30];
    maxprice:real;
    minbeds:integer;
    typewanted,landwanted,locwanted:integer;
  end;
```

Also add customer variables to the Public declarations:

```
public
   { Public declarations }
   houserecord:house;
   housefile:file of house;
   customrecord:customer;
```

```
    customfile:file of customer;
  end;
```

Go to the Form1 screen and click '**Customer records/Set up new file**' to create an event handler. Add the line:

```
procedure TForm1.Setupnewfile2Click(Sender:TObject);
begin
  form6.visible:=true;
end;
```

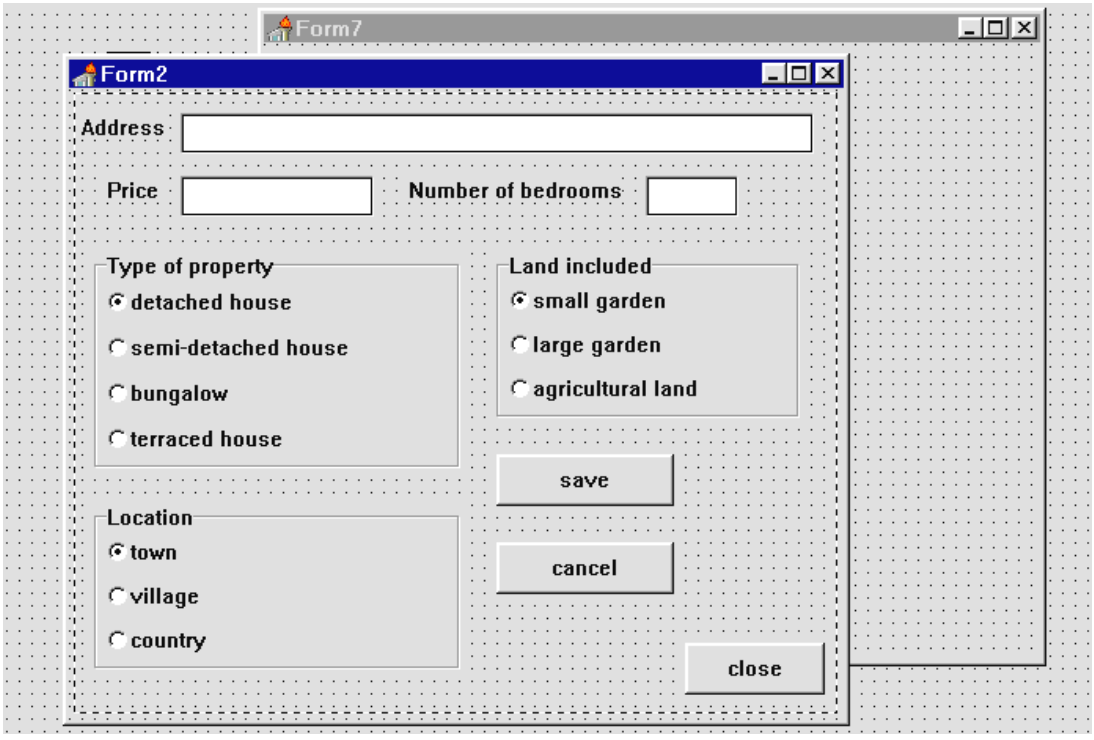
Also add **unit6** to the 'uses' list:

```
uses
  SysUtils,....Unit3, Unit5, Unit6;
```

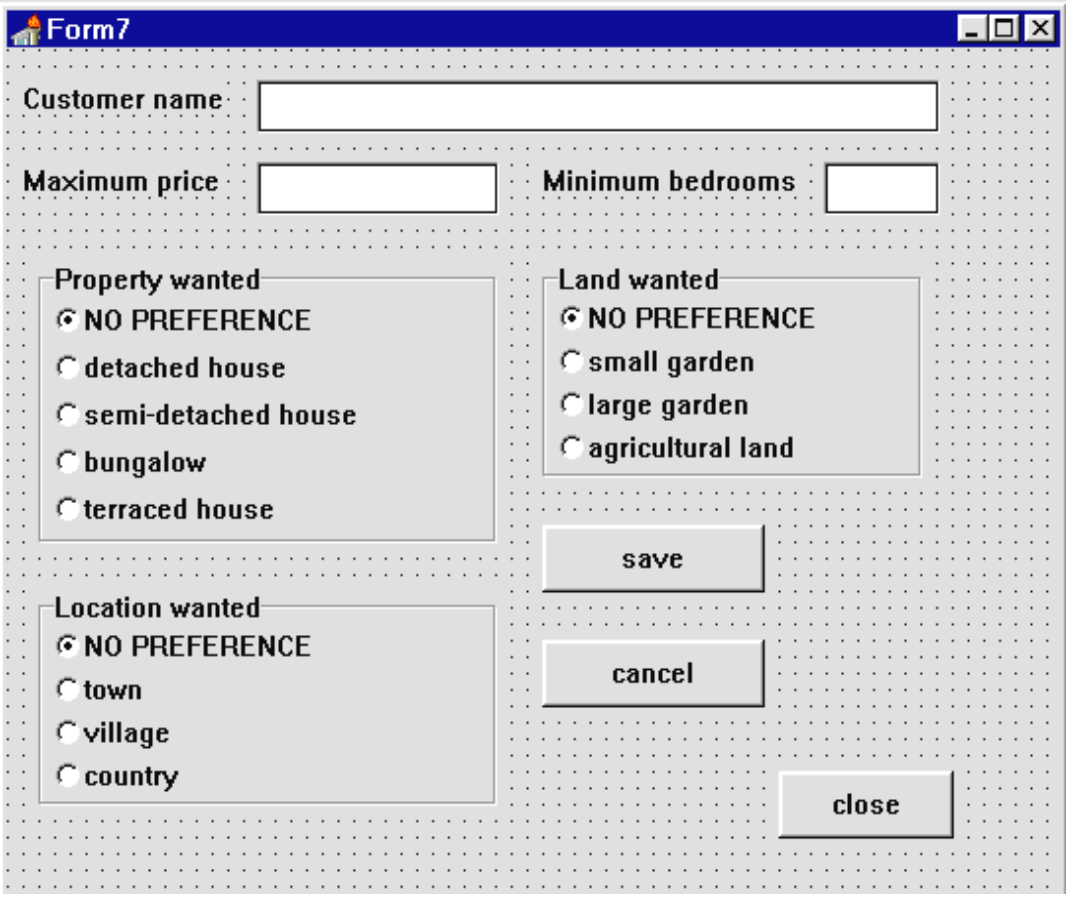
Build and run the program. Select '**Customer records/Set up new file**', then click the '**OK**' button in the confirmation window. Exit from the program and use *Windows Explorer* to check that a new file '**custom.dat**' has been created in your ESTATE sub-directory.

The next stage is the '**Add customer record**' option. Use the short-cut button to create a new blank form.

The easiest way to set up the customer input screen is to copy all the components from Form2:



As with the '*New file*' window, drag the mouse across the Form until a dotted rectangle covers all the components, then click **Edit/Copy**.
Bring **Form7** to the front. Make sure that it is at least as large as *Form2*, then click **Edit/Paste** to transfer a copy of the components.



Change the *Edit Box* captions to read: '**Customer name**', '**Maximum price**', and '**Minimum bedrooms**'. Add the word '**wanted**' to the caption headings of the *Radio Groups*, and insert a '**NO PREFERENCE**' option in each group.

Double-click the 'Customer name' edit box and add a line of program to the event handler:

```
procedure TForm7.Edit1Change(Sender: TObject);
begin
  form1.customrecord.name:=edit1.text;
end;
```

Create similar event handlers for the '**Maximum price**' and '**Minimum bedrooms**' Edit Boxes:

```
procedure TForm7.Edit2Change(Sender: TObject);
begin
  if edit2.text='' then
    form1.customrecord.maxprice:=0
  else
    form1.customrecord.maxprice:=strtofloat(edit2.text);
end;

procedure TForm7.Edit3Change(Sender: TObject);
begin
  if edit3.text='' then
    form1.customrecord.minbeds:=0
  else
    form1.customrecord.minbeds:=strtoint(edit3.text);
end;
```

Set up a '**clear**' procedure:

```
procedure TForm7.clear;
begin
  edit1.text:='';
  edit2.text:='';
  edit3.text:='';
  radiogroup1.itemindex:=0;
  radiogroup2.itemindex:=0;
  radiogroup3.itemindex:=0;
end;
```

and add:

```
procedure clear;
```

to the procedure list near the top of the program. Create an event handler for the '**cancel**' button which uses the '**clear**' procedure:

```
procedure TForm7.Button2Click(Sender: TObject);
begin
  clear;
end;
```

Create the event handler for the '**close**' button which also uses '**clear**':

```
procedure TForm7.Button3Click(Sender: TObject);
begin
```

```
  clear;
  form7.visible:=false;
end;
```

Include a '**uses**' instruction under the **implementation** heading:

```
implementation
{$R *.DFM}
uses
  unit1;
```

It just remains to set up the event handler for the '**save**' button. Add the lines:

```
procedure TForm7.Button1Click(Sender: TObject);
begin
  with form1 do
  begin
    customrecord.typewanted:=radiogroup1.itemindex;
    customrecord.landwanted:=radiogroup2.itemindex;
    customrecord.locwanted:=radiogroup3.itemindex;
    assignfile(customfile,'custom.dat');
    reset(customfile);
    seek(customfile,filesize(customfile));
    write(customfile,customrecord);
    closefile(customfile);
  end;
  clear;
end;
```

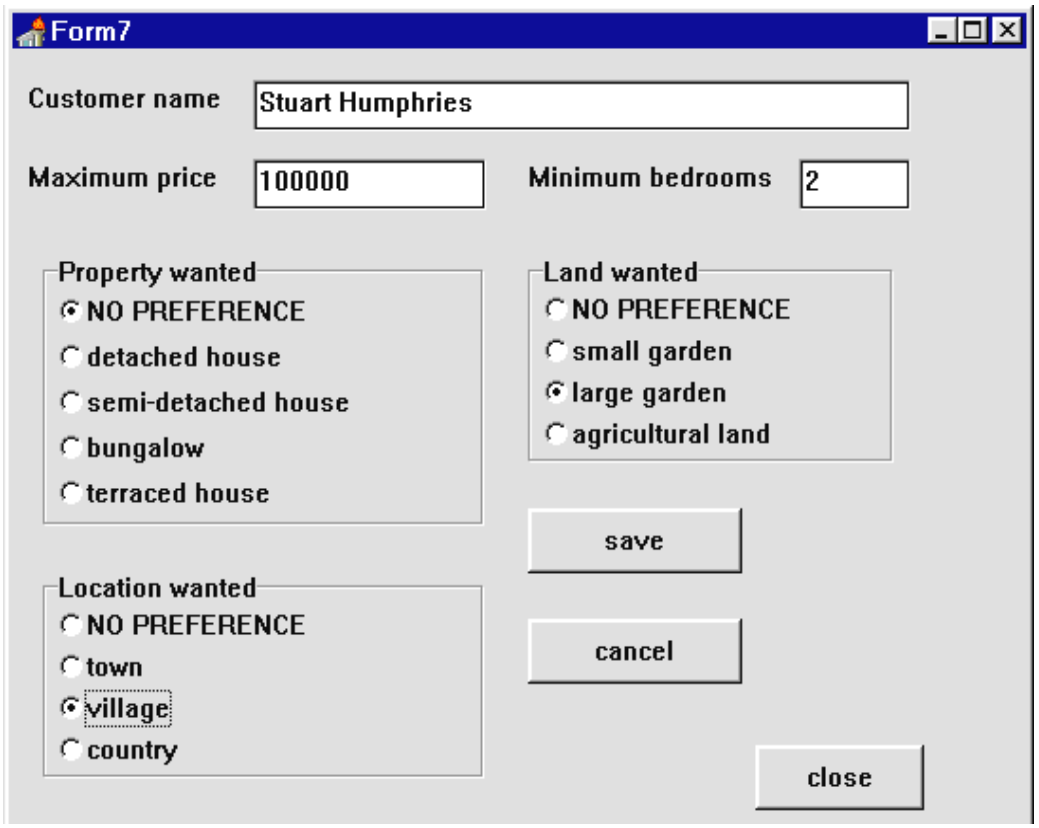
Go to Form1 and click '**Customer records/Add customer record**' to produce an event handler. Insert the line:

```
procedure TForm1.Addcustomerrecord1Click
                                  (Sender: TObject);
begin
  form7.visible:=true;
end;
```

Also add Unit7 to the '**uses**' list near the top of the program.

Build and run the program, then select '**Customer records/Add customer record**'. Enter details for the first two customers from the list on page 245. Where a particular type of house, land or location is not mentioned, select a 'NO PREFERENCE' button as shown in the example on the next page.

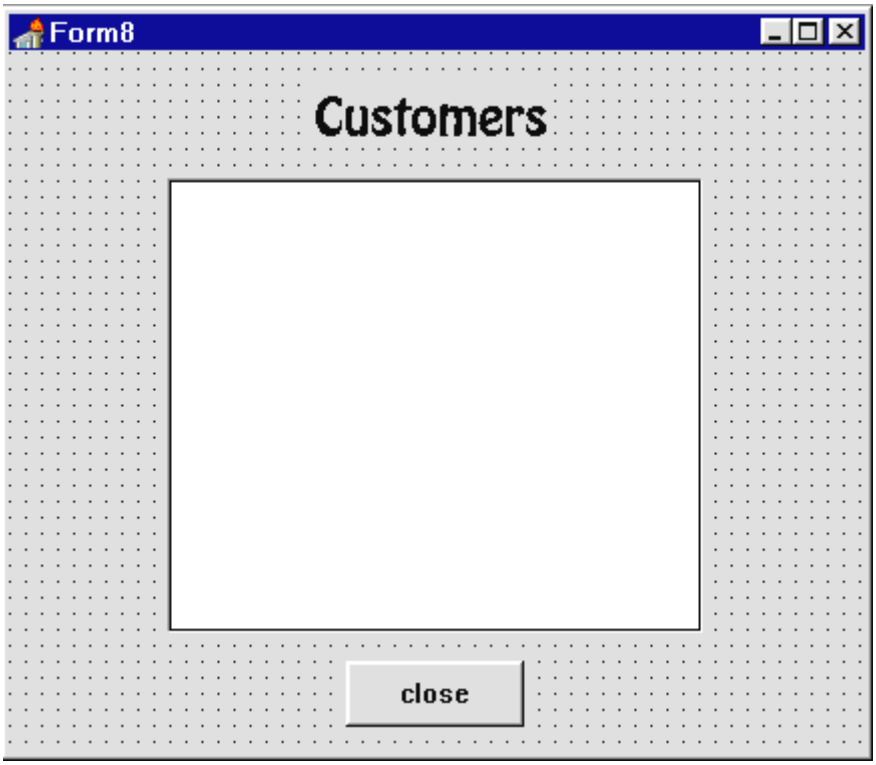
Exit from the program. Use the NOTEPAD utility to check that the customer names now appear in the '**custom.dat**' file, then return to the



Delphi editing screen.

We can now move on to the '**Display customer index**' option. Create a new blank form for this. Add **unit8** to the **'uses'** list near the top of *unit1***.**

Place a *List Box* on *Form8*, a *Label* with the caption '**Customers**', and a *Button* with the caption '**close**':



Set up an event handler for the '**close**' button:

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  form8.visible:=false;
end;
```

Go to the **Form1** screen and click '**Customer records/Display customer index**' to create an event handler. Add a line of program to open the *Form8* customer index window:

```
procedure TForm1.Displaycustomerindex1Click
                                  (Sender: TObject);
begin
  form8.visible:=true;
end;
```

Build and run the program. Select '**Customer records/Display customer index**'. The Customer Index window should open, although no names are displayed yet. Press the '**close**' button and the window should disappear. Return to the Delphi editing screen.

Go to Form8. Click on the dotted grid and press ENTER to bring up the Object Inspector. Click the 'Events' tab, then double-click alongside 'OnActivate to produce an event handler. Add the lines of program to load the customer names from disc:

```
procedure TForm8.FormActivate(Sender: TObject);
begin
  listbox1.clear;
  assignfile(form1.customfile,'custom.dat');
  reset(form1.customfile);
  while not eof(form1.customfile) do
  begin
    read(form1.customfile,form1.customrecord);
    listbox1.items.add(form1.customrecord.name);
  end;
end;
```

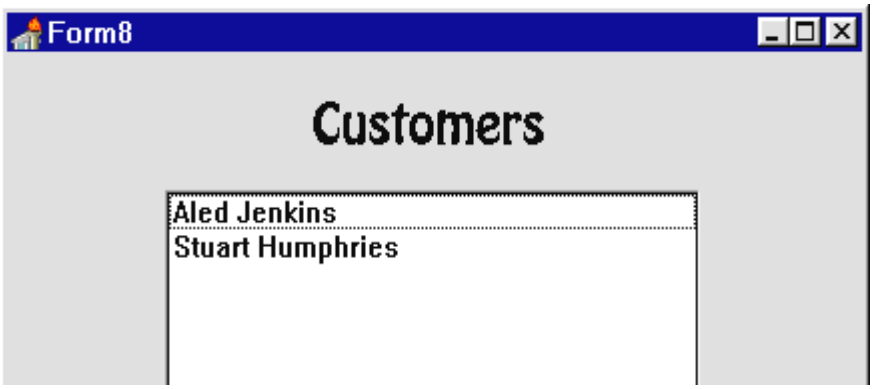
Go back to the Object Inspector for Form8 and set up an event handler for the '**OnDeactivate**' event:

```
procedure TForm8.FormDeactivate(Sender: TObject);
begin
  form8.visible:=false;
end;
```

Add a '**uses**' instruction below the '**implementation**' heading:

```
implementation
{$R *.DFM}
uses
  unit1;
```

Build and run the program. When '**Customer records/Display customer index**' is selected this time, the two customers entered earlier should be listed:



Exit from the program. As in the *House records* section, we would like the **Customer index** to lead to a display screen. Create a new blank form for this. Use the Object Inspector to set **FormStyle** to **StayOnTop**, and **BorderStyle** to **Dialog**. Add '**unit9**' to the '**uses**' line near the top of *unit8*.

Double-click the *List Box* on *Form8* to create an event handler, then add the lines:

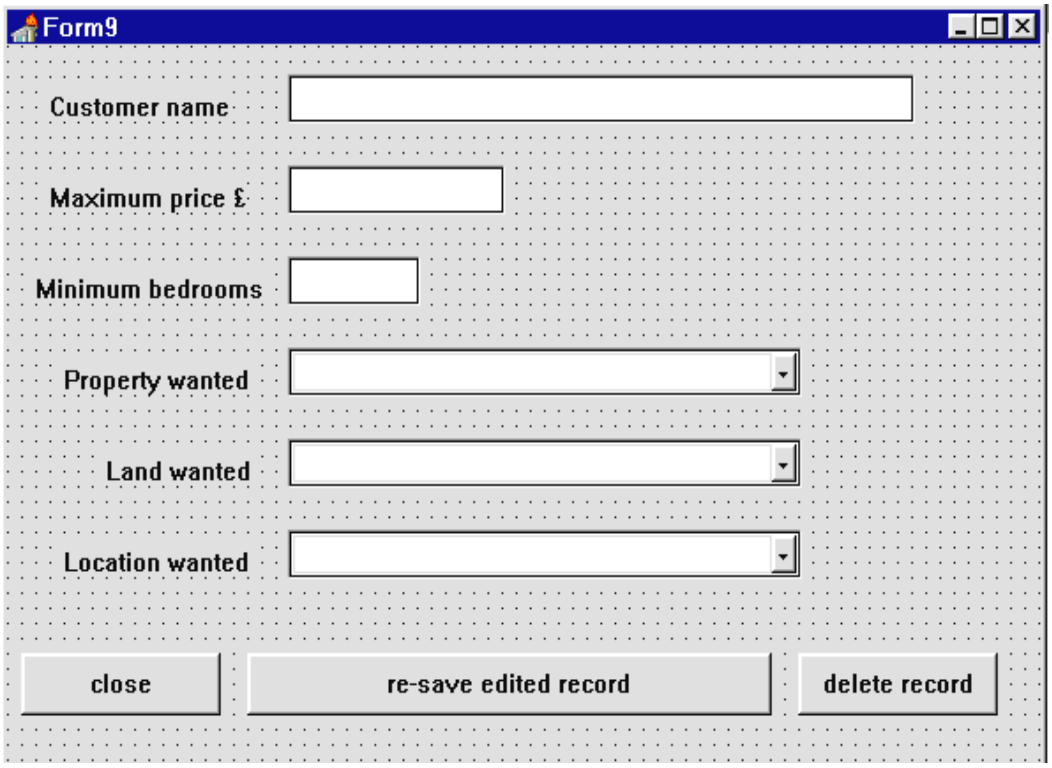
```
procedure TForm8.ListBox1Click(Sender: TObject);
begin
  filepointer:=listbox1.itemindex;
  form8.visible:=false;
  form9.visible:=true;
end;
```

Include '**filepointer**' in the *Public declarations* section of *Unit8*:

```
public
  { Public declarations }
  filepointer:integer;
```

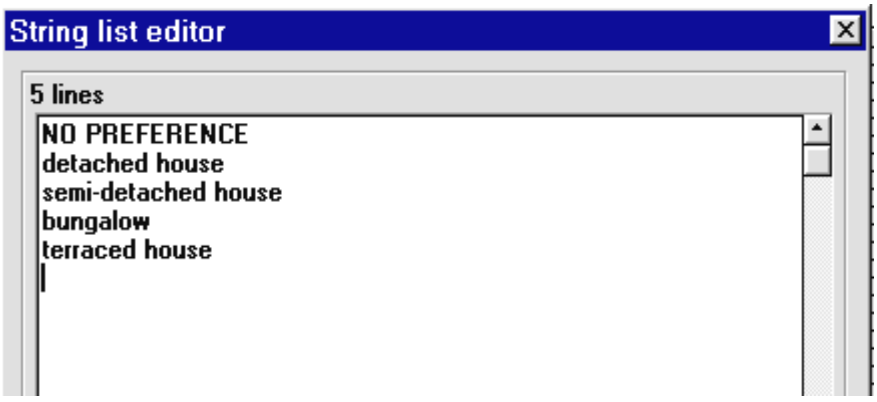
Build and run the program. Select '**Customer records/Display customer index**', then click on one of the customer names. A blank window for **Form9** should open. Exit and return to the Delphi editing screen.

The easiest way to set up **Form9** is to copy the components from *Form4*. Bring the **Form4** window to the front, then drag the mouse over the grid to put a dotted rectangle around all components. Select '**Edit/Copy**'. Go to **Form9** and ensure that the window is at least as large as *Form4*, then select '**Edit/Paste**'.



Change the *Edit Box* and *ComboBox* captions to: '**Customer name**', '**Maximum price £**', '**Minimum Bedrooms**', '**Property wanted**', '**Land wanted**', and '**Location wanted**'.

Select the first *ComboBox*, bring up the Object Inspector, and double-click the **Items** property to display the *String List Editor*. Insert a '**NO PREFERENCE**' entry:



Add '**NO PREFERENCE**' entries for the other two *ComboBoxes*.

Go back to Form9 and click on the dotted grid. Press ENTER to bring up the Object Inspector. Select the **Events** tab and set up an '**OnActivate**' procedure. Add the lines to load and display the selected customer record:

```
procedure TForm9.FormActivate(Sender: TObject);
begin
  with form1 do
  begin
    assignfile(customfile,'custom.dat');
    reset(customfile);
    seek(customfile,form8.filepointer);
    read(customfile,customrecord);
    closefile(customfile);
  end;
  edit1.text:=form1.customrecord.name;
  edit2.text:=floattostrf
           (form1.customrecord.maxprice,ffFixed,8,0);
  edit3.text:=inttostr(form1.customrecord.minbeds);
  combobox1.itemindex:=form1.customrecord.typewanted;
  combobox2.itemindex:=form1.customrecord.landwanted;
  combobox3.itemindex:=form1.customrecord.locwanted;
end;
```

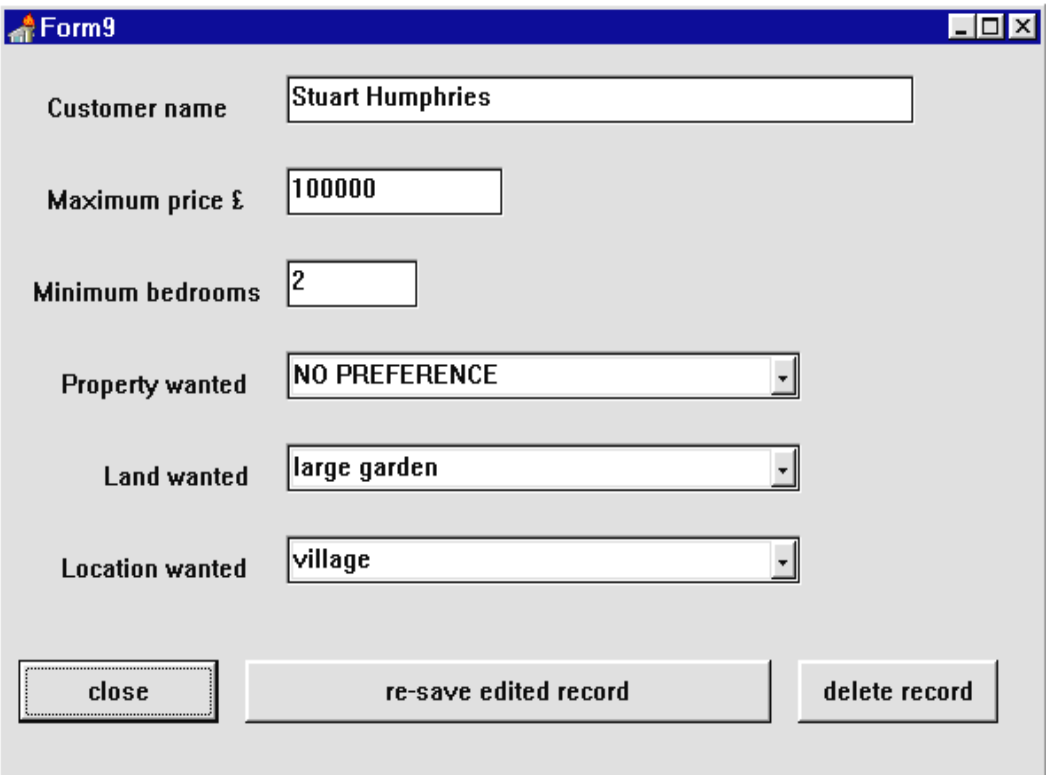
Go back to the Form9 screen and double-click the 'close' button to create an event handler. Add the lines:

```
procedure TForm9.Button1Click(Sender: TObject);
begin
  form9.visible:=false;
  form8.visible:=true;
end;
```

Include **unit1** and **unit8** in a '**uses**' instruction below the *implementation* heading:

```
implementation
{$R *.DFM}
uses
   unit1,unit8;
```

Build and run the program. Go to the **Customer Index** screen and select each of the customer records in turn. Check that the details are displayed correctly as shown below.



Add the remaining customers listed on page 245 and check that these have been added to the customer index, then return to the Delphi editing screen. Go to From9 and double-click the '**re-save edited record**' button to create an event handler. Add the lines:

```
procedure TForm9.Button2Click(Sender: TObject);
begin
  form1.customrecord.name:=edit1.text;
  form1.customrecord.maxprice:=strtofloat(edit2.text);
  form1.customrecord.minbeds:=strtoint(edit3.text);
  form1.customrecord.typewanted:=combobox1.itemindex;
  form1.customrecord.landwanted:=combobox2.itemindex;
  form1.customrecord.locwanted:=combobox3.itemindex;
  with form1 do
  begin
    assignfile(customfile,'custom.dat');
    reset(customfile);
    seek(customfile,form8.filepointer);
    write(customfile,customrecord);
    closefile(customfile);
  end;
  form9.visible:=false;
  form8.visible:=true;
end;
```

Build and run the program to check that customer records can be edited successfully, then return to the Delphi editing screen.

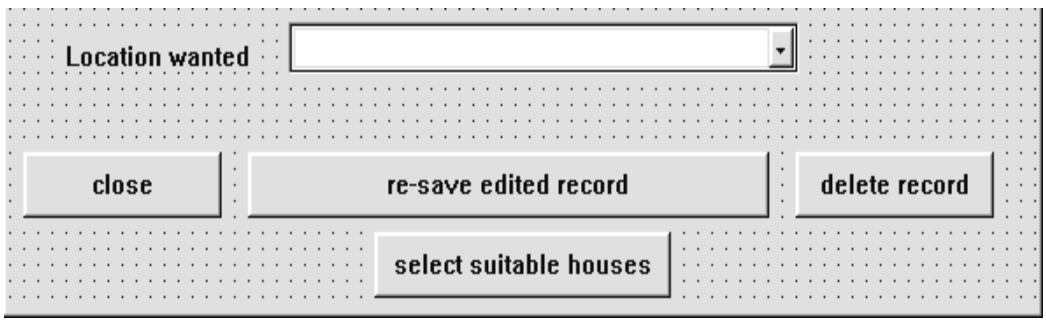
The final step in setting up the *Customer records* system is the *delete record* option. Go to Form9 and double-click the '**delete record**' button. Add lines to the event handler:

```
procedure TForm9.Button3Click(Sender: TObject);
var
  i:integer;
begin
  with form1 do
  begin
    assignfile(customfile,'custom.dat');
    reset(customfile);
    for i:=form8.filepointer+1
                to filesize(customfile)-1 do
    begin
      seek(customfile,i);
      read(customfile,customrecord);
      seek(customfile,i-1);
      write(customfile,customrecord);
     end;
     seek(customfile,filesize(customfile)-1);
     truncate(customfile);
     closefile(customfile);
  end;
  form9.visible:=false;
  form8.visible:=true;
end;
```

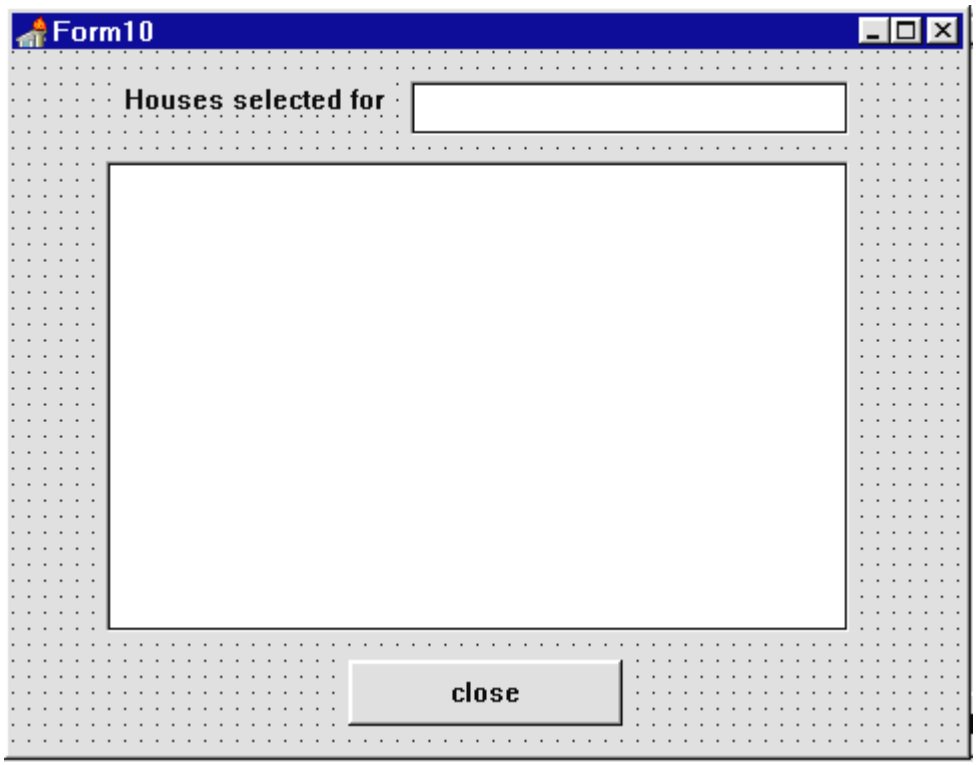
Build and run the program to check that records can be deleted successfuly, then return to the Delphi editing screen.

Selecting suitable houses

The final stage of the project is to select suitable houses for each of the customers. Add another button at the bottom of **Form9** to do this:



Use the short-cut button to create a **new blank form.** Set the **FormStyle** to **StayOnTop,** and the **BorderStyle** to **Dialog**.



Add a *Label* '**Houses selected for**', and put an *Edit Box* alongside it. Place a *List Box* in the centre of the Form, and a *Button* at the bottom with the caption '**close**'.

Double-click the '**close**' button to produce an event handler and add the line:

```
procedure TForm10.Button1Click(Sender: TObject);
begin
  form10.visible:=false;
end;
```

Return to **Form9** and create an event handler for the '**select suitable houses**' button:

```
procedure TForm9.Button4Click(Sender: TObject);
begin
  form10.visible:=true;
end;
```

Add **Unit10** to the '**uses**' list at the top of *Unit9*, then build and run the program. Select the '**Display customer index**' option and click on one of the customer names. When the Customer record is displayed, click the

'**select suitable houses**' button. The blank List Box on Form10 should appear. Click the 'close' button and this should disappear. Return to the Delphi editing screen.

The next step is to display the customer's name in the *Edit Box* at the top of *Form10*. Bring *Form10* to the front. Click on the dotted grid then press ENTER to bring up the Object Inspector. Click the **Events** tab, then double-click alongside '**OnActivate**' to produce an event handler. Add the line:

```
procedure TForm10.FormActivate(Sender: TObject);
begin
  edit1.text:=form1.customrecord.name;
end;
```

Also produce an '**OnDeactivate**' event handler. This is to ensure that *Form10* is closed neatly if the user selects some other option from the *Main Menu*:

```
procedure TForm10.FormDeactivate(Sender: TObject);
begin
  form10.visible:=false;
end;
```

Add a '**uses**' instruction below the implementation heading:

```
implementation
{$R *.DFM}
uses
  unit1;
```

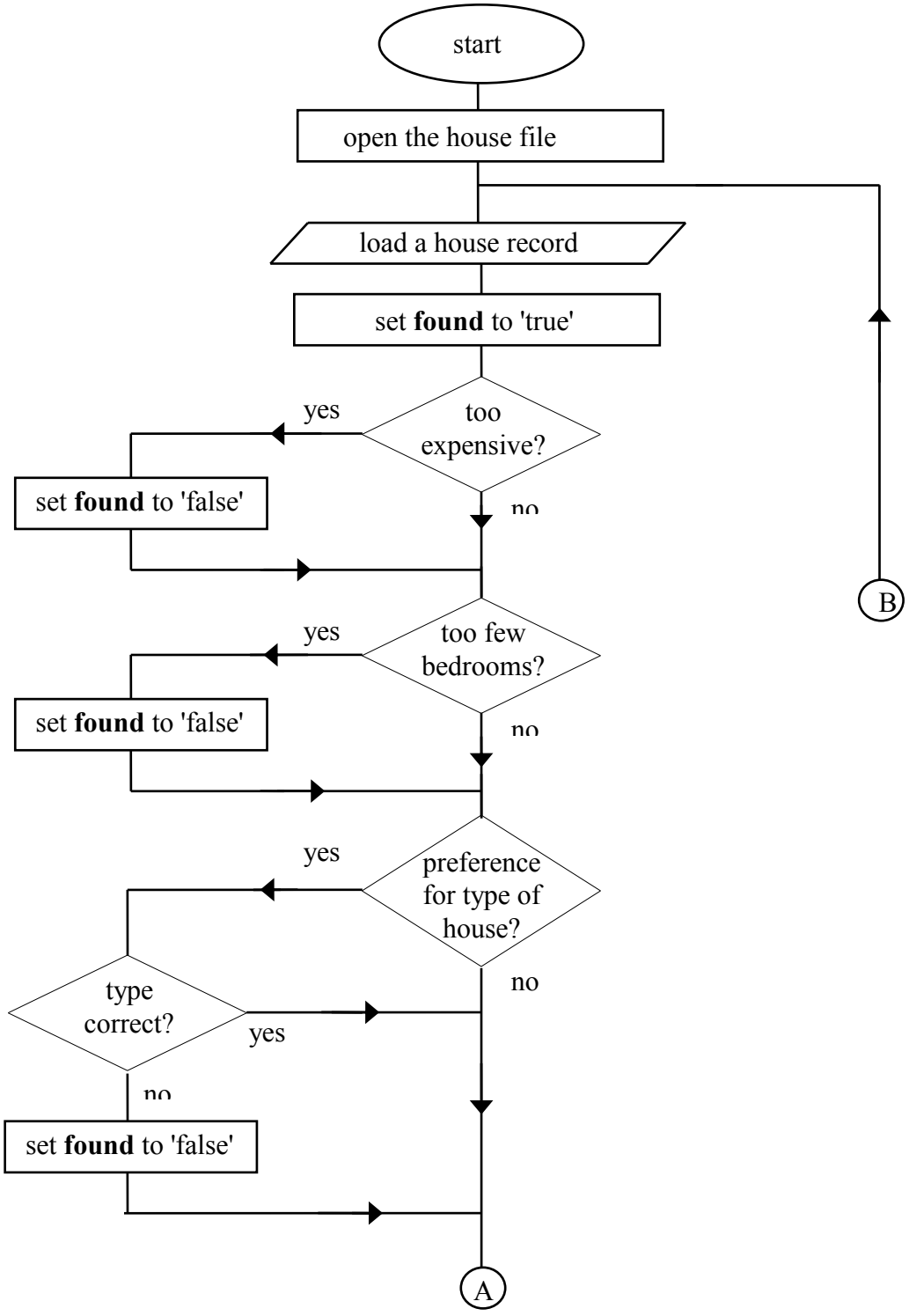
Build and run the program. Select a customer from the Index list and go to the Customer record display screen. Click the 'select suitable houses' button and Form10 should appear - this time showing the name of the customer. Exit and return to the Delphi editing screen.

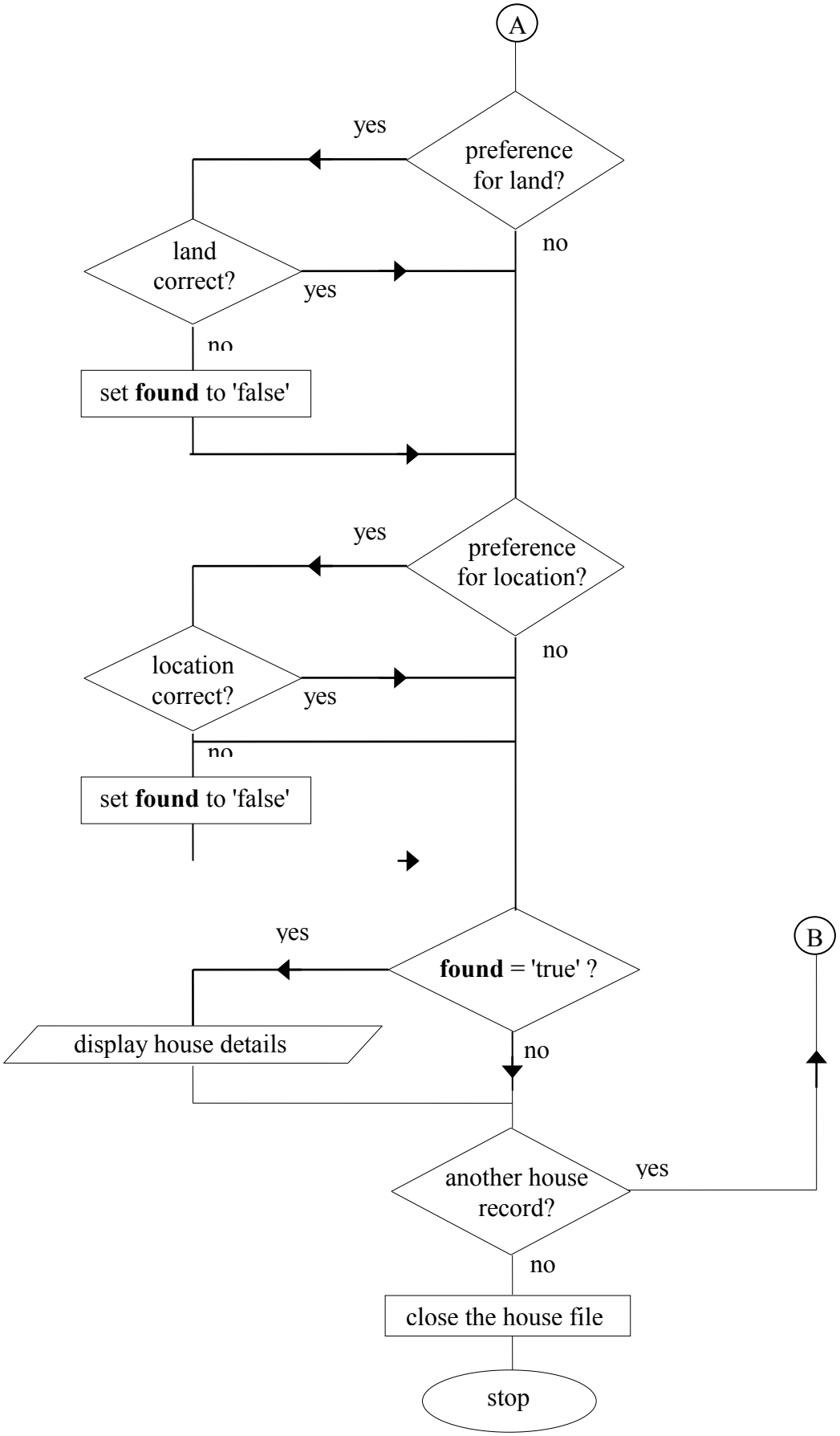
We can now begin work on the actual selection of suitable houses. A flowchart for the procedure is given on the next pages.

We begin by opening the 'houses.dat' file so that each of the house records can be examined. The first record is read into the computer.

We begin by assuming that the current house is suitable for the customer. A series of checks are then carried out to see if the house is <u>not</u> suitable. If at the end of all the checks we have found no reason to exclude it, we will display details of the house in the Form10 *List Box*.

Flow chart to select suitable houses for a customer:




A
preference for land?
yes
no
land correct?
yes
no
set **found** to 'false'
preference for location?
yes
no
location correct?
yes
no
set **found** to 'false'
found = 'true' ?
yes
no
B
display house details
another house record?
yes
no
close the house file
stop

The program uses a Boolean variable '**found**' which has the following meaning:

found = true	A suitable house has been found.
found = false	There is some reason why the current house is not suitable for the customer.

The first check carried out is for **price**; if the house is too expensive then ***'found'*** will be set to **false**.

A similar check is carried out for **number of bedrooms;** '*found*' will be set to **false** if there are not enough bedrooms.

We then come to the check for **type of property:**

- If the customer has indicated NO PREFERENCE, then there will be no need to carry out this check. They would be willing to consider any type: detached house, semi-detached, bungalow or terraced house.
- If the customer has specified a definite type of property, then the check must be carried out. For example, they may only be interested in buying a bungalow. If the current house is not the correct type then '*found*' will be set to **false**.

Similarly, the checks for '**land**' and '**location**' are only carried out if the customer specified a definite preference.

Bring the Unit10 program window to the front and find the 'OnActivate' procedure. Add lines of program to load and check the house records:

```
procedure TForm10.FormActivate(Sender: TObject);
var
  found,success:boolean;
begin
  edit1.text:=form1.customrecord.name;
  listbox1.clear;
  with form1 do
  begin
    assignfile(housefile,'houses.dat');
    reset(housefile);
    success:=false;
    while not eof(housefile) do
    begin
      read(housefile,houserecord);
      found:=true;
      if houserecord.price>customrecord.maxprice then
        found:=false;
      if houserecord.bedrooms<customrecord.minbeds then
        found:=false;
```

```
      if customrecord.typewanted>0 then
      begin
        if houserecord.housetype+1 <>
                  customrecord.typewanted then
        found:=false;
      end;
      if customrecord.landwanted>0 then
      begin
        if houserecord.land+1 <>
                  customrecord.landwanted then
        found:=false;
      end;
      if customrecord.locwanted>0 then
      begin
        if houserecord.location+1 <>
                   customrecord.locwanted then
        found:=false;
      end;
      if found=true then
      begin
        success:=true;
        listbox1.items.add(houserecord.address);
        listbox1.items.add('');
      end;
    end;
    closefile(housefile);
    if success=false then
       listbox1.items.add
                 ('No suitable houses found - sorry');
  end;
end;
```

The procedure begins by opening the house file:

```
assignfile(housefile,'houses.dat');
reset(housefile);
```

A loop then operates to read each of the house records in turn until the end of the file is reached:

```
while not eof(housefile) do
begin
  read(housefile,houserecord);
```

We begin by assuming that the current house is suitable for the customer:

```
found:=true;
```

A check is carried out on the house price. If the price is greater than the maximum price the customer can pay, **found** is set to '**false**' to indicate that the house is unsuitable:

```
if houserecord.price > customrecord.maxprice then
   found:=false;
```

The number of bedrooms is checked in a similar way, then we examine the type of property. The first line looks for a **typewanted** code of **0,** which is the NO PREFERENCE option. The check only goes ahead if the code is greater than zero.

```
if customrecord.typewanted>0 then
begin
```

The next line checks whether the type of property is the same as the type wanted by the customer. Remember that the code numbers are one greater for the customer input screen because the NO PREFERENCE option was inserted as code 0:

```
if houserecord.housetype+1 < > customrecord.typewanted then
   found:=false;
```

The symbols < > taken together mean '**not the same as...**'.

The **landwanted** and **locationwanted** codes are treated in a similar way.

At the end of all the checks, the Boolean variable '*found*' will still be **true** if current house is suitable for the customer, so we can display the address:

```
if found=true then
begin
   success:=true;
   listbox1.items.add(houserecord.address);
end;
```

There is a possibility that no suitable houses will be found, so a message should be displayed to indicate this. We have used a Boolean variable '**success**' which is set to **false** at the start of the procedure. It will be reset to **true** if any suitable house is found. If '**success**' has not been reset and is still **false** after checking all the houses, the message is displayed:

```
if success=false then
  listbox1.items.add('No suitable houses found - sorry');
```

We can now test the **house selection** procedure. Build and run the program. Begin by checking that all the house and customer records listed on pages 244-5 are present and the details are correct.

Take each of the customers in turn. Compare the customer requirements given on page 245 with the houses listed on page 244, and work out which houses meet the customer's requirements. Click the '**select suitable houses**' button and compare the computer output with your list. Convince yourself that the correct properties are selected each time, then return to the Delphi editing screen.

It would be useful to display full details of the properties, not just the addresses. Include '**textline**' as a **string** variable at the start of the *Form10* '**OnActivate**' procedure, then add lines to the program:

```
    .....
if found=true then
begin
  success:=true;
  listbox1.items.add(houserecord.address);
  textline:='Price: £'+
       floattostrf(houserecord.price,ffFixed,8,0);
  listbox1.items.add(textline);
  textline:=inttostr(houserecord.bedrooms)+
                                    ' bedrooms';
  listbox1.items.add(textline);
  case houserecord.housetype of
    0:textline:='Detached house';
    1:textline:='Semi-detached house';
    2:textline:='Bungalow';
    3:textline:='Terraced house';
  end;
```

```
case houserecord.location of
   0:textline:=textline + ' in the town';
   1:textline:=textline + ' in the village';
   2:textline:=textline + ' in the country';
 end;
 listbox1.items.add(textline);
 case houserecord.land of
   0:textline:='with a small garden';
   1:textline:='with a large garden';
   2:textline:='with agricultural land';
 end;
 listbox1.items.add(textline);
 listbox1.items.add('');
         .....
```

Build and run the completed program to check that house details are displayed correctly for each customer.