

## SIXTEEN

### Selecting files and printing data

So far during the course we have produced several programs which saved data records on disc, but in each case we specified the file name to be used. For example, the Estate Agent's database program always saved house records in a file called '*houses.dat*'. Generally, however, it is better if the user can choose their own file name when saving data. Consider the following situation:

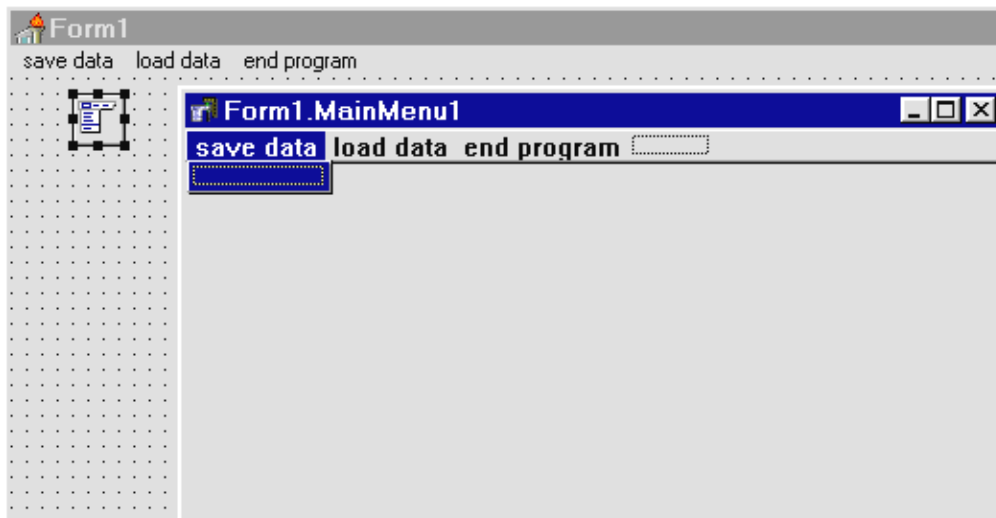
A supermarket chain has branches in a number of towns in North Wales. Each day the total sales are recorded in a computer file, and at the end of the week each branch sends the file to head office. The company devises a naming system for the files which combines code letters for the branch and a number for the week. For example, the file:

PORT24.DAT

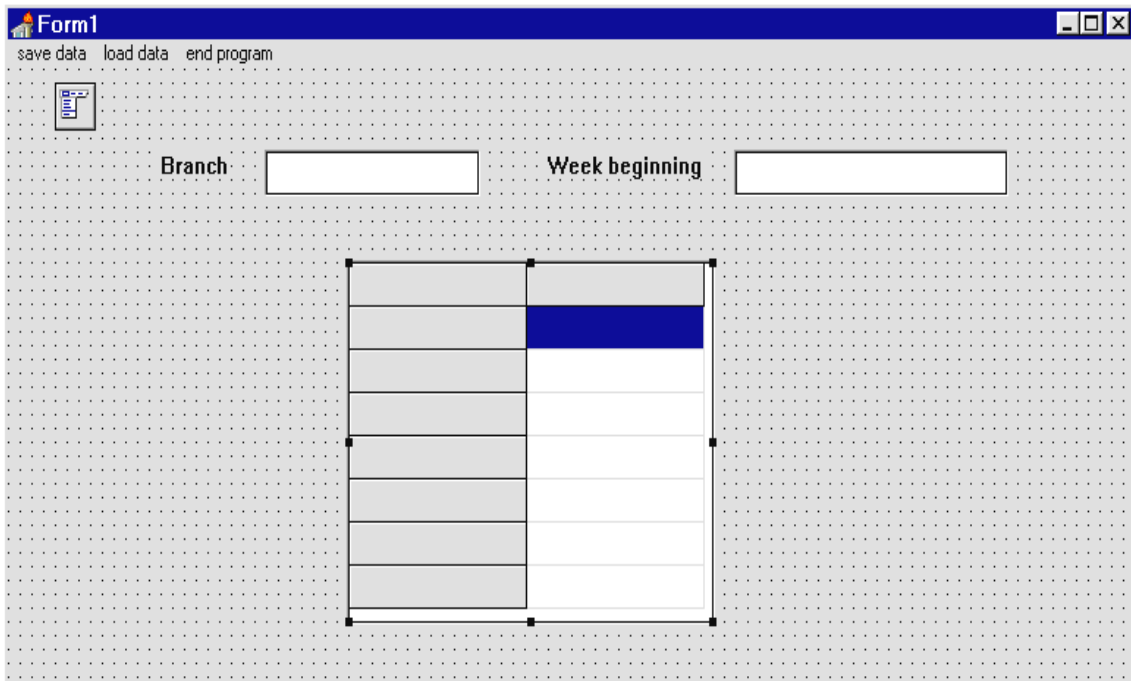
contains the sales figures for the Porthmadog branch in week 24 of the year.

To implement this system we would need a way for the store employees to enter the appropriate file name when storing their data. We will write the program to see how this is done:

Set up a new directory SALES and save a Delphi project into it. Use the Object Inspector to Maximize the Form, and drag the grid to nearly fill the screen. Place a *MainMenu* component on the Form. Double-click the icon to bring up the Menu Editor, and put the captions 'save data', 'load data' and 'end program' on the top line of the screen:



Add two *Edit Boxes* to the Form, and place the *Labels* 'Branch' and 'Week ending' alongside:



Put a *String Grid* in the centre of the form, and use the Object Inspector to set the properties:

<b>DefaultColWidth</b>	<b>120</b>
<b>ColCount</b>	<b>2</b>
<b>RowCount</b>	<b>8</b>
<b>Options:</b>	
<b>goEditing</b>	<b>True</b>
<b>ScrollBars</b>	<b>none</b>

Go to the menu line at the top of the *Form*, and double-click the '**end program**' option to produce an event handler. Add a **halt** command:

```
procedure TForm1.endprogram1Click(Sender: TObject);
begin
    halt;
end;
```

Compile and run the program to check that the components are displayed correctly, then click '**end program**' to return to the Delphi editing screen.

Double-click the dotted grid of the *Form* to produce an '**OnCreate**' procedure. Add lines of program to write captions for the string grid:

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  with stringgrid1 do
  begin
    cells[1,0]:='sales £';
    cells[0,1]:='Monday';
    cells[0,2]:='Tuesday';
    cells[0,3]:='Wednesday';
    cells[0,4]:='Thursday';
    cells[0,5]:='Friday';
    cells[0,6]:='Saturday';
    cells[0,7]:='Sunday';
  end;
end;

```

We need to set up a record structure for storing the sales data on disc. Insert this near the top of the program below the **'type'** heading:

```

type
  sales=record
    branch,week:string[24];
    daytotal:array[1..7] of real;
  end;

TForm1 = class(TForm)
  .....

```

The name of the branch and the date will be stored as strings of text; the figure **24** in square brackets allows each entry to be up to 24 characters in length. The daily sales figures will be stored as an array of seven decimal numbers to allow for pounds and pence.

Go to the *Public declarations* section and add the variable names for the file and records:

```

public
  { Public declarations }
  salesrecord:sales;
  salesfile:file of sales;
end;

```

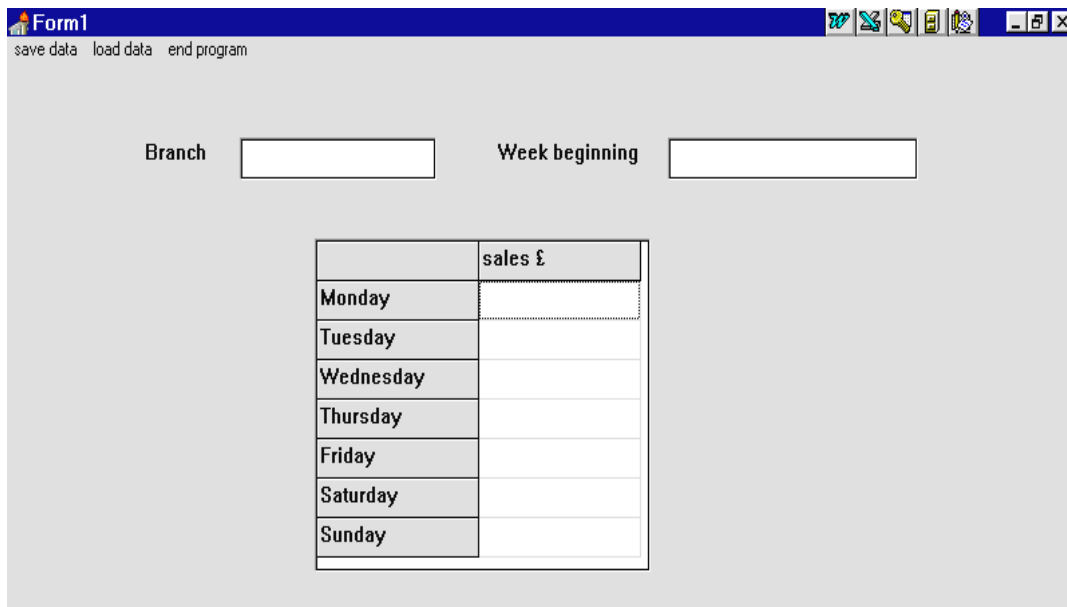
We now need a procedure to transfer the sales figures into the array. Click on the *String Grid* and press ENTER to bring up the Object Inspector. Click the **Events** tab, then double-click alongside **'OnKeyUp'** to produce an event handler. Add the lines:

```

procedure TForm1.StringGrid1KeyUp(Sender: TObject;
                               var Key: Word;Shift:TShiftState);
var
  y:integer;
begin
  y:=stringgrid1.row;
  if stringgrid1.cells[1,y]='' then
    salesrecord.daytotal[y]:=0
  else
    salesrecord.daytotal[y]:=
      strtofloat(stringgrid1.cells[1,y]);
end;

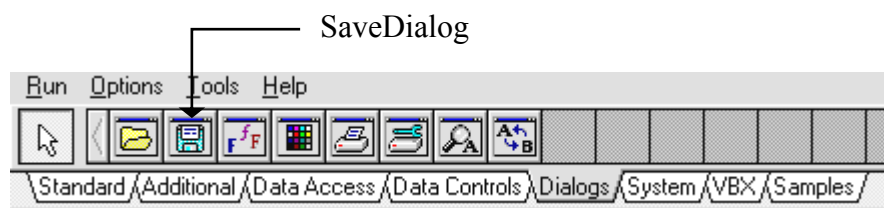
```

Compile and run the program. Headings should now be displayed in the *String Grid*:

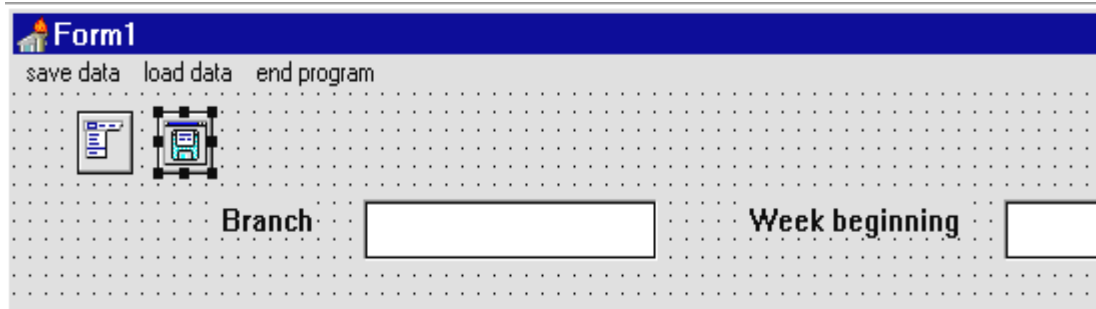


Check that the String Grid is error trapped to accept decimal numbers, then return to the Delphi editing screen.

We can now begin work on the '**save data**' option. Go to the DIALOGS menu and select the *Save Dialog* component:



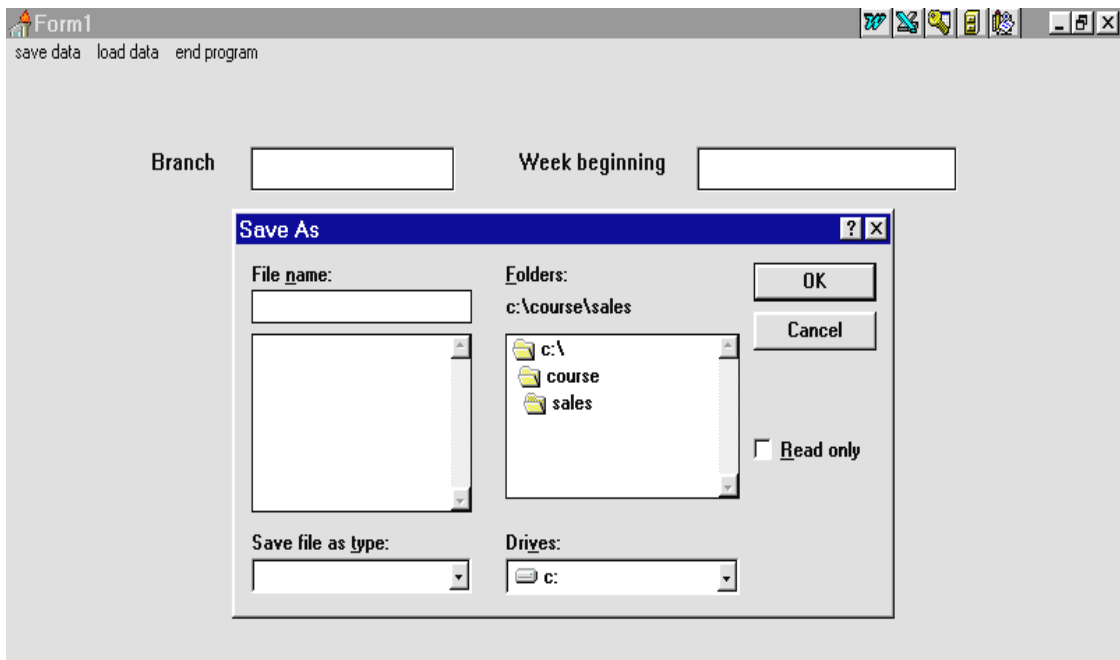
Place a *SaveDialog* component on the Form. As in the case of the *Main Menu*, this appears as a fixed sized icon which can be positioned anywhere convenient on the grid.



Go to the menu line at the top of the Form and double-click the 'save data' option to produce an event handler. Add the instruction:

```
procedure TForm1.savedata1Click(Sender: TObject);  
begin  
    savedialog1.execute;  
end;
```

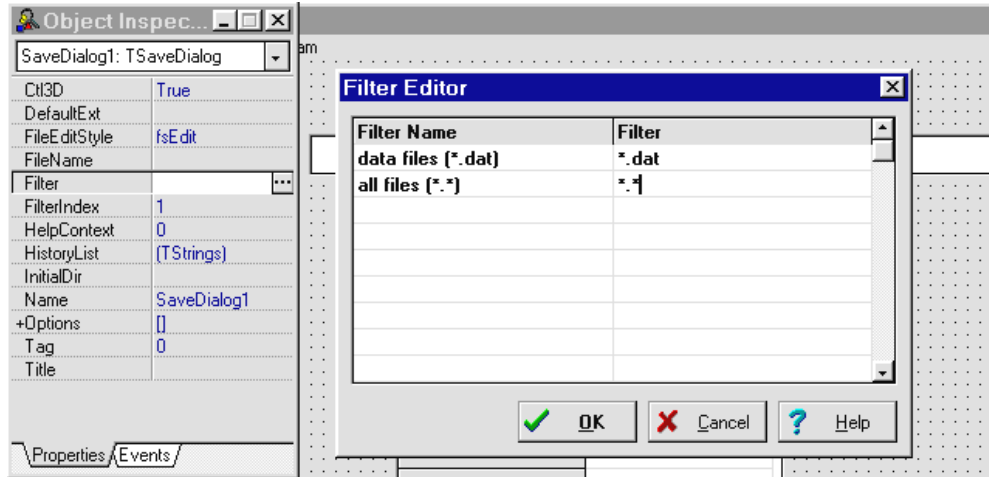
Compile and run the program. Click the 'save data' menu option and a file selection window will open - this is probably familiar to you from other *Windows* applications:



It is possible to change the drive or directory, but no file names are being displayed yet. It is necessary to set up a *filter* to show files in the display window - we will do that next...

Click the 'Cancel' button, then select 'end program' to return to the Delphi editing screen.

Go to the *Form* grid, click on the **SaveDialog** icon and press ENTER to bring up the Object Inspector. Double-click alongside the **Filter** property and the 'Filter Editor' window will open:



Add the entries:

Filter name	Filter
data files (*.dat)	*.dat
all files (*.*)	*.*

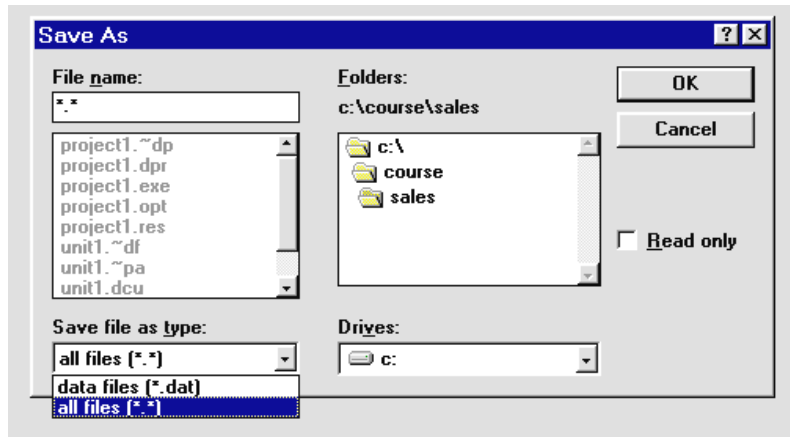
then click the 'OK' button.

Set the property:

DefaultExt	DAT
------------	-----

This ensures that if the user enters a file name without a three-letter extension, the computer will automatically add the letters 'DAT'. For example: 'PORT24' would become 'PORT24.DAT'.

Compile and run the program. Click the 'Save data' option and check that it is now possible to list just the .DAT files or all the files in a particular directory:



Click 'cancel' to close the file window, then 'end program' to return to the Delphi editing screen.

We can now set up the program to save the sales data onto disc. Go to the *Form* grid and click the 'save data' menu option to open the event handler. Add the lines:

```

procedure TForm1.savedata1Click(Sender: TObject);
begin
    if savedialog1.execute then
    begin
        salesrecord.branch:=edit1.text;
        salesrecord.week:=edit2.text;
        assignfile(salesfile,savedialog1.filename);
        rewrite(salesfile);
        write(salesfile,salesrecord);
        closefile(salesfile);
    end;
end;

```

The conditional block:

```

    if savedialog1.execute then
    begin
        .....
    end;

```

will only operate if the user enters a valid filename.

The lines:

```

        salesrecord.branch:=edit1.text;
        salesrecord.week:=edit2.text;

```

transfer the 'Branch' and 'Week beginning' entries from the edit boxes into the fields of the data record.

We then open a new file using the filename entered by the user:

```
assignfile(salesfile,savedialog1.filename);  
rewrite(salesfile);
```

The sales record is written into the file, then the file is closed:

```
write(salesfile,salesrecord);  
closefile(salesfile);
```

Compile and run the program. Enter a set of sales data, as shown on the next page, then click the 'save data' option. When the 'Save As' window appears, select a suitable directory and give the file name:

**HAR30**

without a .DAT extension. Click 'OK', then exit to the Delphi editing screen.

The screenshot shows a Delphi application window with a light gray background. At the top, there are two input fields: 'Branch' with the text 'Harlech' and 'Week beginning' with the text '10 August'. Below these fields is a table with the following data:

	sales £
Monday	685.20
Tuesday	750.40
Wednesday	425.67
Thursday	728.92
Friday	642.10
Saturday	1136.90
Sunday	348.27

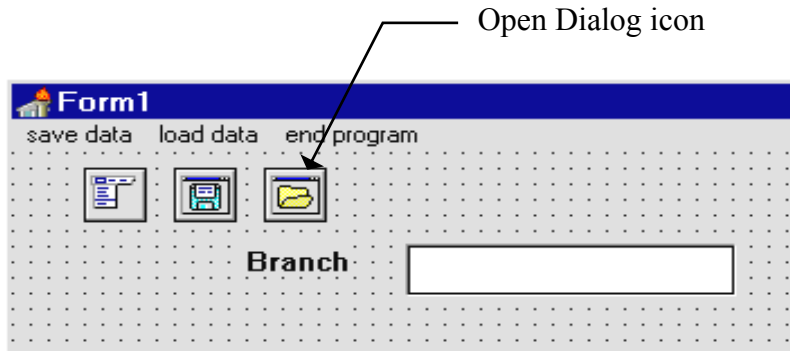
Use the NOTEPAD utility to check that a file has been saved with the full name  
HAR30.DAT

and that this shows the name of the branch and date. There will also be some random characters present which represent the numerical data for the sales.





We can now work on the menu option to reload the sales data. Begin by adding an '**Open Dialog**' component to the form:



Click the *OpenDialog* icon and press ENTER to bring up the Object Inspector. Double-click alongside the '**Filter**' property, and make the same entries as you did earlier for the *SaveDialog* component:

:

Filter name	Filter
data files (*.dat)	*.dat
all files (*.*)	*.*

Go back to the Form grid and click the '**load data**' menu option to produce an event handler. Add the lines:

```

procedure TForm1.loaddata1Click(Sender: TObject);
var
  i:integer;
begin
  if opendialog1.execute then
  begin
    assignfile(salesfile,opendialog1.filename);
    reset(salesfile);
    read(salesfile,salesrecord);
    closefile(salesfile);
    edit1.text:=salesrecord.branch;
    edit2.text:=salesrecord.week;
    for i:=1 to 7 do
      stringgrid1.cells[1,i]:=floattostrf
        (salesrecord.daytotal[i],ffFixed,8,2);
  end;
end;

```

This procedure is very similar to the one for saving data. We begin by loading the file selected by the user. Data is then transferred from the fields

of the record into the *Edit Boxes* and *String Grid*. The sales values need to be converted from decimal numbers into text strings so that they can be displayed.

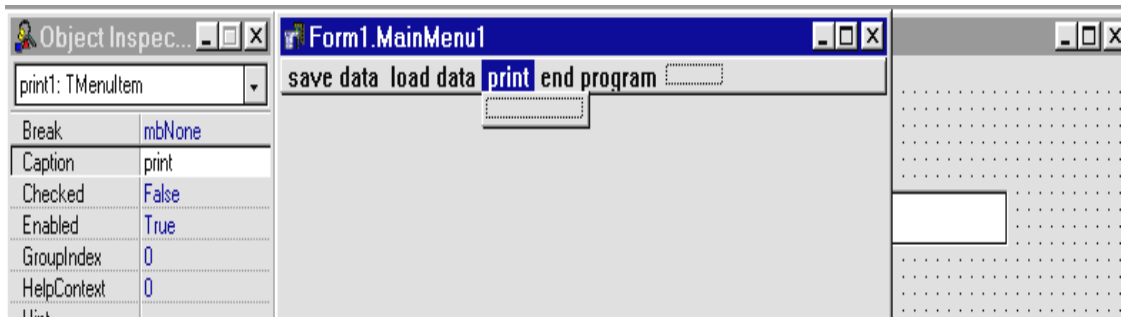
Compile and run the program. Click the **'load data'** menu option and a file window should appear. Select the file **HAR30** and click **'OK'**. The data saved earlier should now appear on the Form.

Check that it is possible to alter and resave the data using a different file name, then reload either the original or the updated file from disc. Exit to the Delphi editing screen.

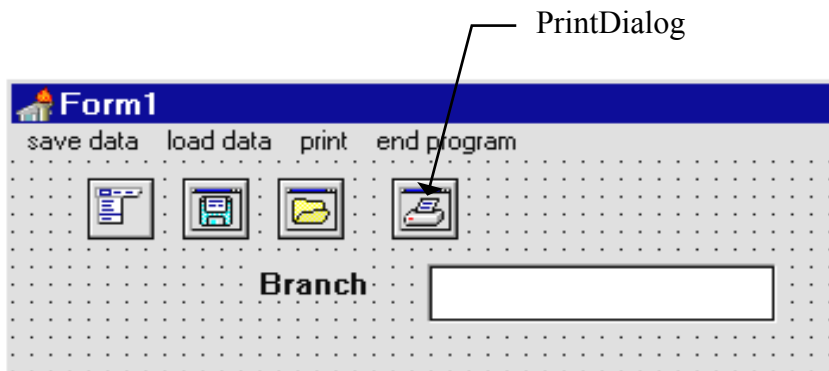
A further feature we can add to the program is to print out the sales data on paper.

Double-click the *MainMenu* icon on the Form grid to bring up the Menu Editor window:

Click **'end program'** so that it is highlighted, then press the INSERT key. An empty box should appear between the 'load data' and 'end program' options. Enter the caption **'print'** as shown below:



Return to the *Form1* grid. Go to the DIALOGS component menu and select **Print Dialog**. Place the Print Dialog icon on the Form grid:

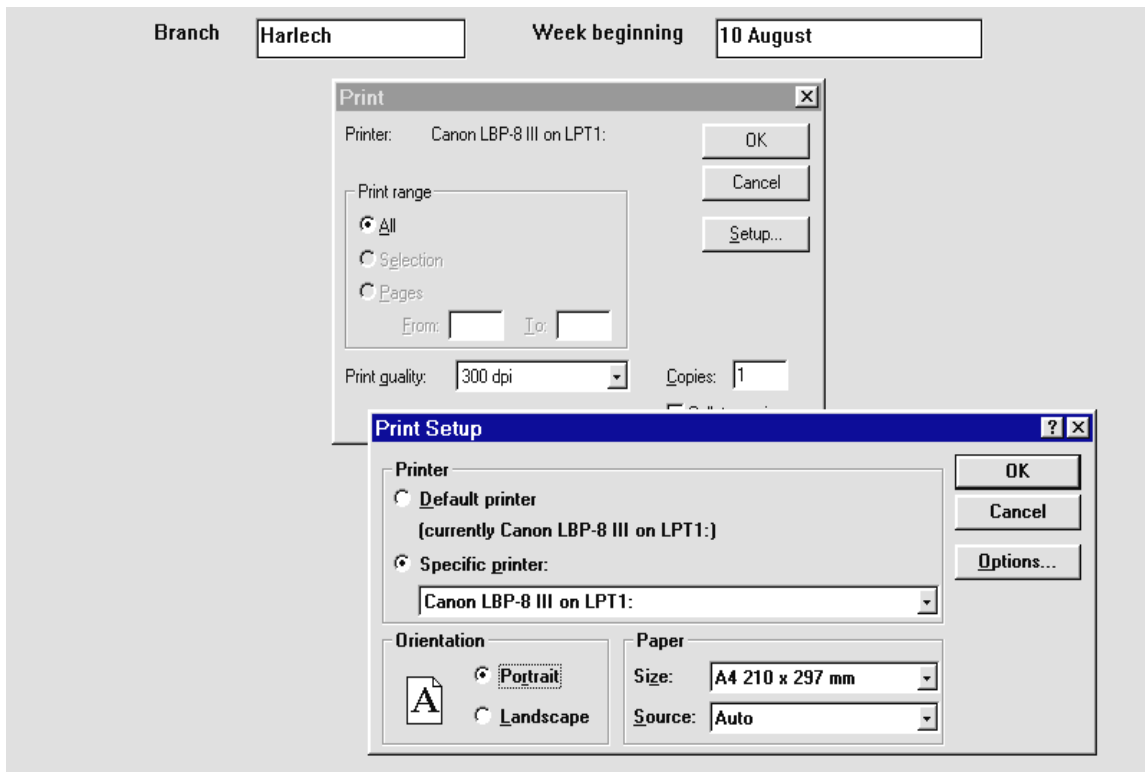


Click on the **'print'** menu option at the top of the Form to produce an event handler. Add the line:

```
procedure TForm1.print1Click(Sender: TObject);  
begin  
    printdialog1.execute;  
end;
```

Compile and run the program. Check that the saved data can still be reloaded, then click the **'print'** option. A print dialog window will open, as shown on the next page. This allows you to set the number of copies required. There is also a button to open a **setup** window in which the orientation of the paper (portrait or landscape format) can be chosen.

We have not yet written the program lines to produce a printout. Choose



**'cancel'** and **'end program'** to return to the Delphi editing screen.

Click the **'print'** option to display the event handler, then add the lines below.

NOTE: In places it is necessary to include blank spaces so that the text will be neatly layed out on the paper. Important spaces will be shown by shading and a number, e.g.:

□ □ □ □

indicates a point where four blank spaces are typed with the space bar.

```
procedure TForm1.print1Click(Sender: TObject);
var
  printfile: textfile;
begin
  if printdialog1.execute then
  begin
    assignprn(printfile);
    rewrite(printfile);
    printer.canvas.font.name:='Courier New';
    printer.canvas.font.size:=12;
    writeln(printfile, '    Branch: '
              +edit1.text);
    writeln(printfile, '    Week beginning: '
              + edit2.text);

    closefile(printfile);
  end;
end;
```

Printing data from a Delphi program is similar to saving data on disc - you send a *file* to the printer in a similar way to sending a file to the disc drive.

The first lines:

```
var
  printfile: textfile;
```

set up a file variable. We have called this **'printfile'**, but any name could be used.

The conditional block:

```
if printdialog1.execute then
begin
  ....
end;
```

will not operate if the user clicks the **'cancel'** button in the print dialog window.

We open the file to write data to the printer:

```
assignprn(printfile);
rewrite(printfile);
```

The type face and size for the printout can be chosen. 'Courier New' is a non-proportional spaced font in which every letter is the same width - this is good for keeping tables of data correctly aligned in columns:

```
printer.canvas.font.name:='Courier New';
printer.canvas.font.size:=12;
```

We then output the name of the branch and the date to the printer:

```
writeln(printfile,'    Branch: '+edit1.text);
writeln(printfile,'    Week beginning: '+ edit2.text);
```

Four blank spaces have been included at the start of each line so that the text is not printed right at the left hand edge of the paper.

The print file is closed when the printing is completed:

```
closefile(printfile);
```

Go to the **'uses'** line at the top of *Unit1* and add **'printers'** to the list:

```
uses
  SysUtils, WinTypes, .... Menus, printers;
```

Compile and run the program. Load the test data from disc then select the print option. Click 'OK' to print. A printout should be produced showing the branch and date:

```
Branch:   Harlech
Week beginning: 10 August
```

Click **'end program'** to return to the Delphi editing screen, then click the **'print'** option to open the event handler. Add lines of program to print the table of sales figures:

```
procedure TForm1.print1Click(Sender: TObject);
var
  printfile: textfile;
  textline,item:string;
  i:integer;
begin
  if printdialog1.execute then
  begin
    assignprn(printfile);
    rewrite(printfile);
    printer.canvas.font.name:='Courier New';
    printer.canvas.font.size:=12;
    writeln(printfile,'    Branch: '+edit1.text);
    writeln(printfile,
```

```

        '    Week beginning:' + edit2.text);
writeln(printfile);
writeln(printfile,
        '    Day            sales (£)');
for i:=1 to 7 do
begin
    textline:='    '+stringgrid1.cells[0,i];
    item:='            '+stringgrid1.cells[1,i];
    textline:=textline+item;
    writeln(printfile,textline);
end;
closefile(printfile);
end;
end;

```



The command:

```
writeln(printfile);
```

simply misses a blank line on the printout.

Column headings are printed with:

```
writeln(printfile,'    Day            sales (£)');
```

A loop then begins for printing the seven days' sales data:

```
for i:=1 to 7 do . . . .
```

We begin to build up a line of text by adding the name of each day from column 0 of the string grid:

```
textline:='    '+stringgrid1.cells[0,i];
```

Spaces are included to separate the columns, then the sales figure from column 1 of the string grid is added:

```
item:='            '+stringgrid1.cells[1,i];
```

```
textline:=textline+item;
```

The completed line of text is sent to the printer:

```
writeln(printfile,textline);
```

Compile and run the program. Load the test data from disc and select the **print** option. Click **OK** and the table of data should be printed:

Branch: Harlech	
Week beginning: 10 August	
Day	sales (£)
Monday	685.20
Tuesday	750.40
Wednesday	425.67
Thursday	728.92
Friday	642.10
Saturday	1136.90
Sunday	348.27

The data is all present, but unfortunately it is not aligned neatly in columns yet. There is a problem because the day names have different lengths, and the sales figures contain different numbers of digits. Return to the Delphi editing screen and click **print** to bring back the event handler procedure. Make the following alterations to the program:

```

.....
writeln(printfile, '    Day                sales (£) ');
for i:=1 to 7 do
begin
  item:='    '+stringgrid1.cells[0,i]+
                                             '          ';
  textline:=copy(item,1,16);
  item:='          '+stringgrid1.cells[1,i];
  item:=copy(item,length(item)-10,11);
  textline:=textline+item;
  writeln(printfile, textline);
end;

```



This new section of program makes use of the COPY command. This takes a piece of text and copies a specified number of characters into a string variable. COPY has three parameters:

**copy** (string, starting position, number of characters to be copied)

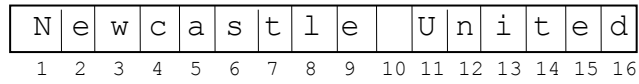
For example:

```

item:= copy ('Newcastle United', 4, 6) ;

```

would begin at the fourth character of 'Newcastle United' and copy six characters into the string variable called 'item':



**item =**

c	a	s	t	l	e
---	---	---	---	---	---

Let's see how the COPY command is used to format the columns of the printout:

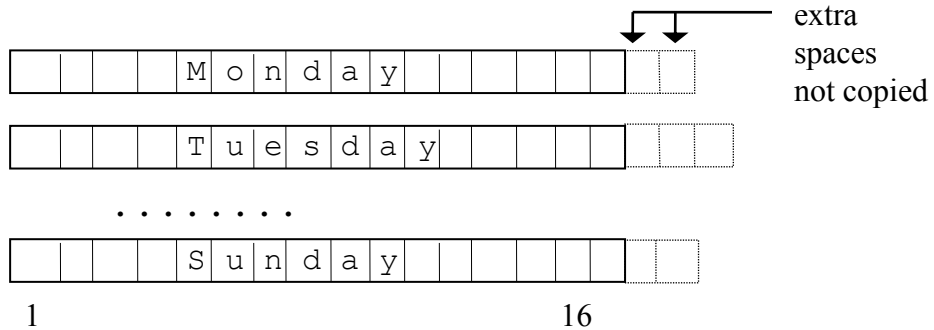
We begin by adding some blank spaces to the end of each day name:

**item:='□□□□'+stringgrid1.cells[0,i]+'□□□□□□□□';**

We then use the 'copy' command to begin at character 1 and copy just the first 16 characters into the variable 'textline':

**textline:=copy(item,1,16);**

In this way, each day name will be followed by sufficient spaces to make the length exactly 16 characters:



We use a similar technique to make the sales figures into equal length strings. This time, however, we need to 'right align' the figures. Space is added before the numbers, then we copy just the last 11 characters:

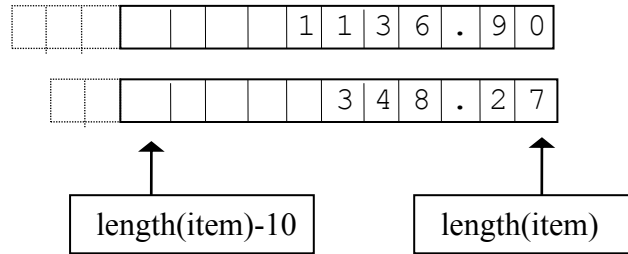
**item:='□□□□□□□□'+stringgrid1.cells[1,i];**

**item:=copy(item,length(item)-10,11);**

count back 10 characters from the end of the string

copy the 11 characters from here to the end of the string





These equal length strings can now be neatly aligned when the column of data is printed.

Compile and run the program. Load the test data from disc and select the **print** option. Click 'OK' and the full table of data should be printed with correct alignment:

```

Branch:   Harlech
Week beginning: 10 August

Day           sales (£)
Monday        685.20
Tuesday       750.40
Wednesday     425.67
Thursday      728.92
Friday        642.10
Saturday      1136.90
Sunday        348.27

```

For our next program we will make use of the techniques for saving files and printing data, but in a situation requiring more complex processing of information:



## Coach tour bookings

A tour operator in Barmouth is organising three coach trips each day (Monday - Sunday) during particular weeks in the summer. The trips advertised are:

- Circular tour of Snowdonia
- Visit to the Ffestiniog Railway
- Day out in Aberystwyth

The tour operator has three drivers available each day, along with three vehicles:

- a 15 seat minibus
- a small 30 seat coach
- a large 50 seat coach

Your task is to design a computer program to handle bookings and enquiries for the tours. A program is required which can:

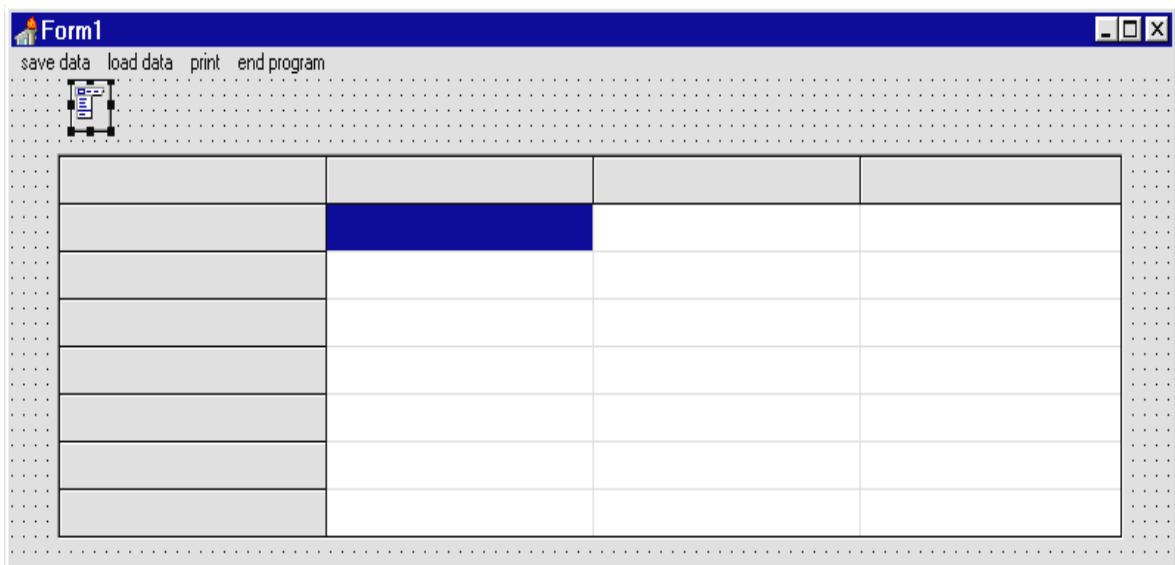
- (i) Record the current number of passengers for tours to Snowdonia, the Ffestiniog Railway, and Aberystwyth on each of the days Monday to Sunday.
- (ii) Allow new bookings to be entered for any day. If the required number of seats are still available, the computer will make the booking, otherwise a suitable message is displayed.

The program should be able to save data on disc and print out the current numbers of passengers booked.

**IMPORTANT!**

The program must take into account the limited number of seats on the smaller vehicles. If 31 passengers are going to Aberystwyth, the large coach will be needed, so no more than 30 passengers can go to either of the other destinations.

To begin the program, set up a new directory COACHES and save a Delphi project into it. Use the Object Inspector to Maximize the Form, and drag the grid to nearly fill the screen.



Place a string grid on the form.

Use the Object Inspector to set the properties for the string grid:

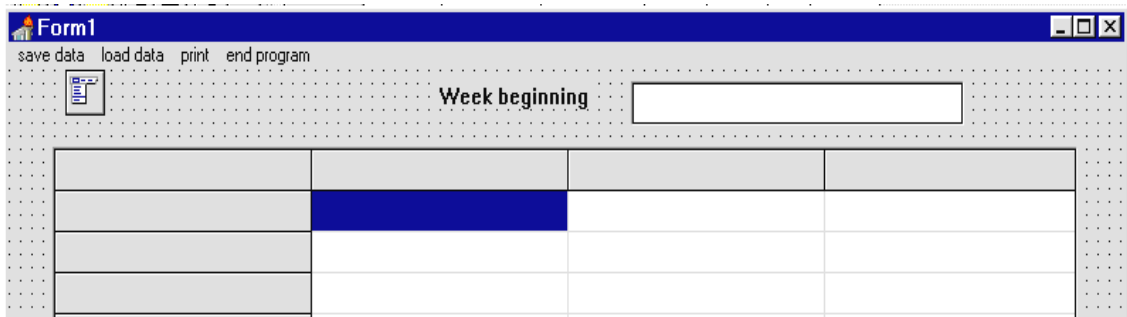
<b>DefaultColWidth</b>	<b>180</b>
<b>ColCount</b>	<b>4</b>
<b>RowCount</b>	<b>8</b>
<b>ScrollBars</b>	<b>none</b>

Add a **Main Menu** component to the grid, and double-click the icon to bring up the **Menu Editor** window. As for the supermarket sales program we have just completed, add options along the top line of the Form: 'save data', 'load data', 'print', and 'end program'.

Close the Menu Editor window and return to the *Form1* grid. Click the 'end program' option to produce an event handler and add the line:

```
procedure TForm1.endprogram1Click(Sender: TObject);
begin
    halt;
end;
```

Put an *Edit Box* on the Form1 grid above the String grid. Place a *Label* alongside with the caption 'Week beginning':



Compile and run the program to check that the components are displayed correctly, then click the 'end program' option to return to the Delphi editing screen.

The program will need a record structure in which to store the date and the numbers of passengers booked on each coach trip. Insert this below the 'type' heading:

```
type
    coach=record
        date:string[24];
        booking:array[1..7,1..3] of integer;
    end;
```

Go to the Public declarations section and add variable names for the record and file:

```
public
  { Public declarations }
  coachrecord:coach;
  coachfile:file of coach;
end;
```

Bring the *Form1* window to the front and double-click on the grid to produce an 'On create' event handler. Add the lines of program:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i,j:integer;
begin
  with stringgrid1 do
  begin
    cells[0,1]:='Monday';
    cells[0,2]:='Tuesday';
    cells[0,3]:='Wednesday';
    cells[0,4]:='Thursday';
    cells[0,5]:='Friday';
    cells[0,6]:='Saturday';
    cells[0,7]:='Sunday';
    cells[1,0]:='Snowdonia';
    cells[2,0]:='Ffestiniog Railway';
    cells[3,0]:='Aberystwyth';
  end;
  for i:=1 to 7 do
  begin
    for j:=1 to 3 do
    begin
      coachrecord.booking[i,j]:=0;
      stringgrid1.cells[j,i]:=inttostr(0);
    end;
  end;
end;
```

The first group of lines:

```
with stringgrid1 do
begin
  cells[0,1]:='Monday';
  .....
  cells[3,0]:='Aberystwyth';
```

will write captions on the String Grid for the days of the week and the three destinations of the coach trips.

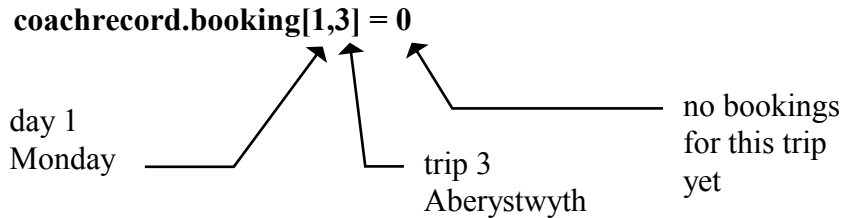
The next set of lines use two loops to initialise the bookings for each trip to zero on each of the days. We also initialise the String Grid with zero values in each cell:

```

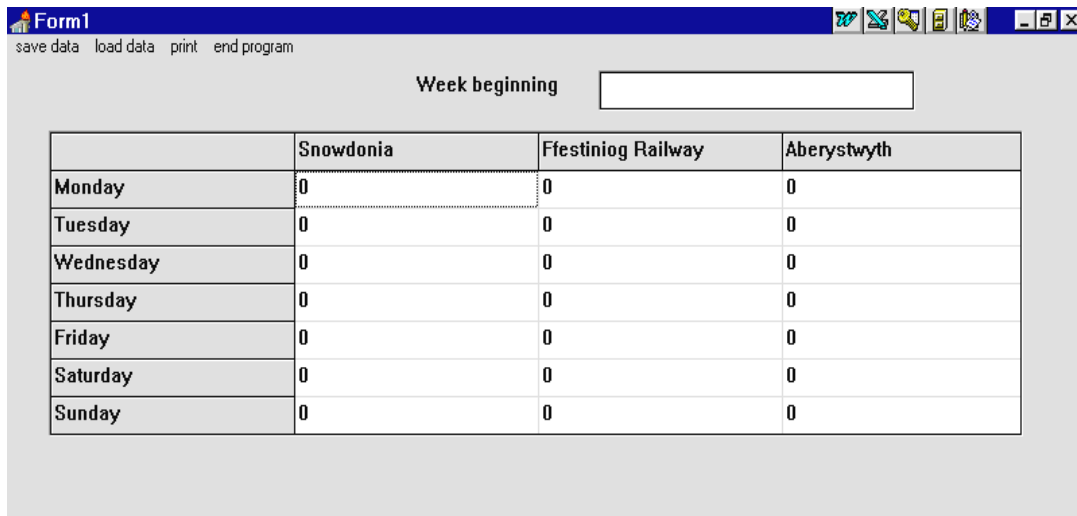
for i:=1 to 7 do                                repeat for each day
begin
  for j:=1 to 3 do                                repeat for each trip
  begin
    coachrecord.booking[i,j]:=0;
    stringgrid1.cells[j,i]:=inttostr(0);
  end;
end;

```

The *booking* field of *coachrecord* is a two dimensional array. The first index value represents the number of the day, whilst the second index represents the number of the trip, e.g.:



Compile and run the program. Check that captions are displayed on the *String Grid* correctly and that all the coach bookings are shown as zero, then return to the Delphi editing screen.

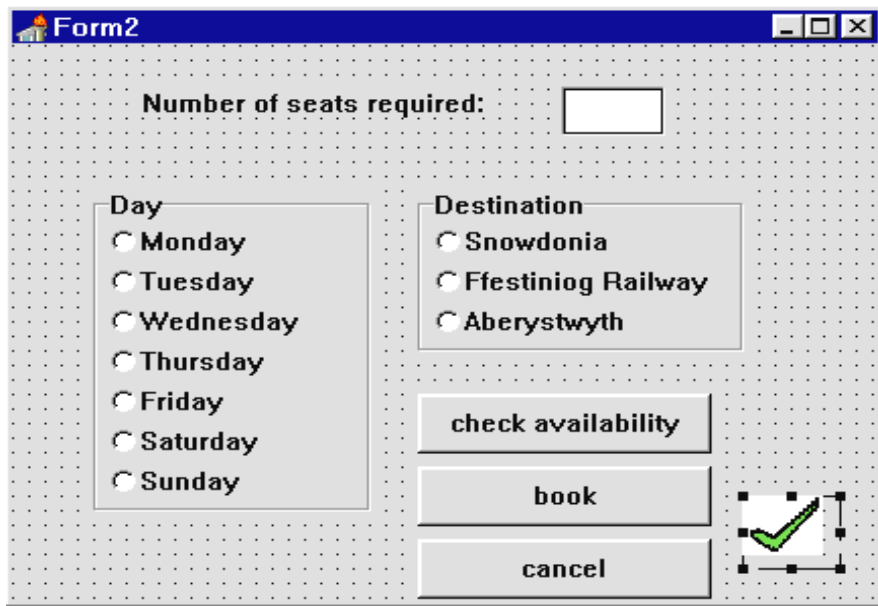


A separate window can be used to enter the bookings, in a similar way to the railway ticket program in chapter 5. Use the **New form** short-cut button to create a blank form.

Use the Object Inspector to set the properties for *Form2*:

<b>BorderStyle</b>	<b>Dialog</b>
<b>FormStyle</b>	<b>StayOnTop</b>
<b>Visible</b>	<b>True</b>

Place an *Edit Box* on *Form2*, and a *Label* alongside with the caption 'Number of seats required:'. Add three *Buttons* with the captions 'check availability', 'book' and 'cancel'.



Set up two *Radio Groups* to display buttons for the **days of the week**, and the trip destinations 'Snowdonia', 'Ffestiniog Railway' and 'Aberystwyth'.

Complete the Form with a small *Image Box* in the bottom right hand corner. Load the picture file YES2.BMP which is provided. Set the **Visible** property the **False** for the Image Box. Also set **Visible** to **False** for the 'book' and 'cancel' Buttons.

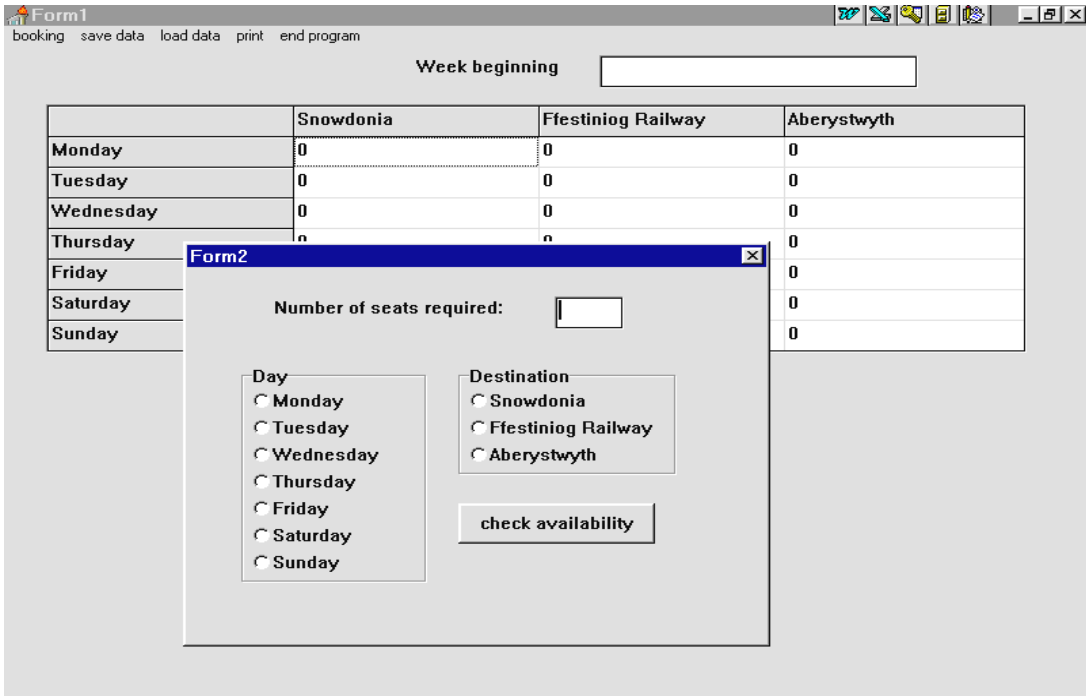
Before testing the program we need to link the two forms. Do this by adding a 'uses' instruction below the 'implementation' heading in *Unit2*:

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

Also add **Unit2** to the 'uses' list at the top of *Unit1*:

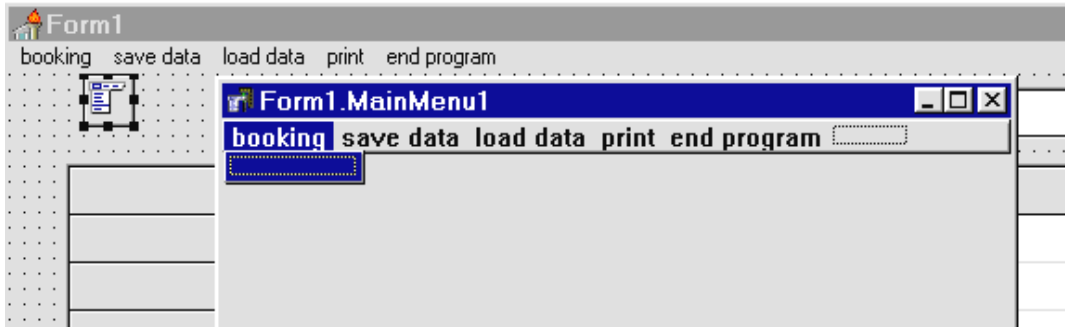
```
uses
  SysUtils, WinTypes, ..., Menu, unit2;
```

Build and run the program. *Form2* should appear 'floating' on top of the table of bookings, with only the 'check availability' button visible. Check that a day and destination can be selected with the Radio Groups, then return to the Delphi editing screen.



When the program is running, the user may wish to close the *Form2* data entry window in to examine the table of bookings underneath. We therefore need a way of opening *Form2* again when needed for the next booking. We can add a menu option to do this.

Double-click the '**Main Menu**' icon on *Form1* to bring up the **Menu Editor** window. Click on '**save data**' then press INSERT to produce an empty box to the left. Add the caption '**booking**':



Close the Menu Editor window, then click the booking option to produce an event handler. Add the line:

```
procedure TForm1.booking1Click(Sender: TObject);
begin
    form2.visible:=true;
end;
```

Build and run the program. Close the *Form2* window by clicking the cross in the top corner. It should then be possible to reopen the window by clicking the **'booking'** option on the menu line. Check this, then return to the Delphi editing screen.

We are able to input booking details from the customer. The next step is to make the computer check whether enough seats are still available for the required trip:

First we will set up an event handler to store the number of seats required as an integer variable **'seats'**. Bring *Form2* to the front and double-click the **edit box** to produce an event handler. Add the lines:

```
procedure TForm2.Edit1Change(Sender: TObject);
begin
    if edit1.text='' then
        seats:=0
    else
        seats:=strtoint(edit1.text);
end;
```

Include the variable **'seats'** in the Public declarations section:

```
public
    { Public declarations }
    seats:integer;
end;
```

Compile and run the program to check that the **'seats required'** edit box is error trapped to only accept integers, then return to the Delphi editing screen.

We can now begin the event handler procedure for the **'check availability'** button. Double-click the button to create this.

When the button is pressed, we first want the computer to record the number of the day (1..7) and the number of the trip (1..3) which the customer has chosen. This data can be obtained from the *Radio Groups*. The program



should be error trapped to check that a *day* and *trip* have been selected, and that a valid number of passengers (at least 1) has been entered. We can build this error trapping into the event handler:

Add lines to the '**check availability**' button click procedure:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
    day:=radiogroup1.itemindex+1;
    trip:=radiogroup2.itemindex+1;
    if (day>0) and (trip>0) and (seats>0) then
    begin
        halt;
    end;
end;
```

This begins by transferring the *day* and *trip* numbers into variables:

```
day:=radiogroup1.itemindex+1;
trip:=radiogroup2.itemindex+1;
```

The conditional block will only operate if entries have been selected in both Radio Groups, and the number of passengers is greater than zero:

```
if (day>0) and (trip>0) and (seats>0) then
begin
    halt;
end;
```

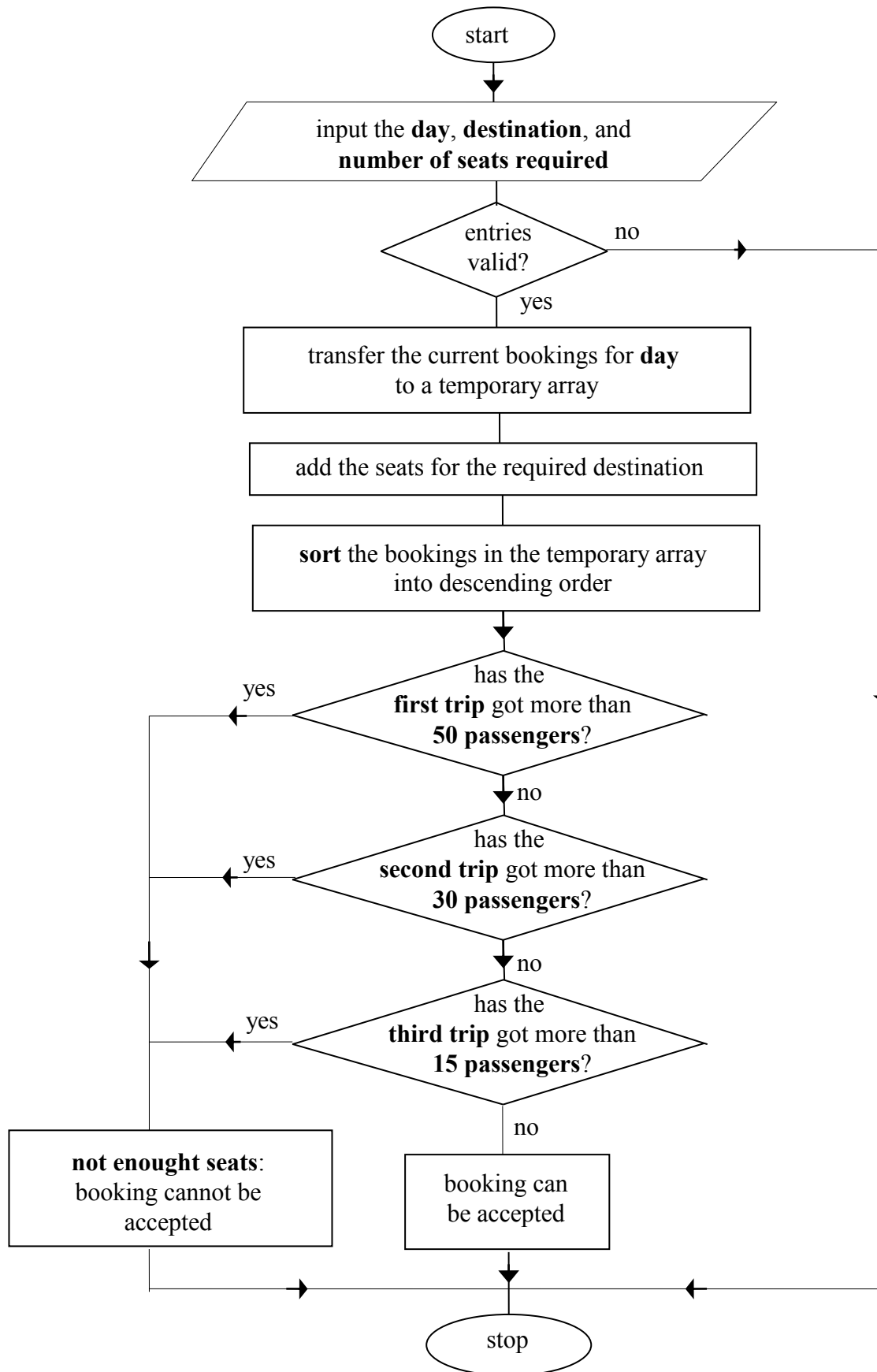
A '**halt**' command has been included at this point to allow the procedure to be tested. Once we know that it works correctly, the '**halt**' can be replaced by program lines to carry out the booking enquiry.

Add the variables *day* and *trip* to the *Public declarations* section:

```
public
    { Public declarations }
    day, trip, seats:integer;
end;
```

Build and run the program. If the '**check availability**' button is clicked before a *day* or *destination* has been selected, there should be no response. Select a *day* and *destination*, but set the *number of seats required* to zero - there should still be no response to clicking the button. Only when a valid number of passengers is entered will the conditional block operate and the program halts.

We now need to design the algorithm for handling the bookings. This is quite tricky, so let's consider an example:



Suppose that on Monday the current booking situation is:

**Snowdonia 22**  
**Ffestiniog 16**  
**Aberystwyth 12**

Another group of 10 people now wish to book for the trip to the Ffestiniog Railway; can we accept their booking?

An approach you might try is to transfer the current bookings into a temporary array, then add the extra bookings, giving:

```
temp[1]= 22
temp[2]= 16 + 10    = 26
temp[3]= 12
```

Now sort the temporary array into descending order to give:

```
temp[1]= 26
temp[2] = 22
temp[3] = 12
```

This arrangement of passengers will be possible provided:

**temp[1] is not larger than 50, so they fit on the large coach**  
**temp[2] is not larger than 30, so they fit on the small coach**  
**temp[3] is not larger than 15, so they fit on the minibus**

In this case, the booking can be accepted, and the booking array updated as appropriate. A flowchart for this sequence is given on the next page.

Delete the **'halt'** command from the **'check availability'** button click procedure and add the following lines of program:

```
procedure TForm2.Button1Click(Sender: TObject);
var
  swap,i,j:integer;
  temp:array[1..3]of integer;
  acceptbooking:boolean;
begin
  day:=radiogroup1.itemindex+1;
  trip:=radiogroup2.itemindex+1;
  if (day>0) and (trip>0) and (seats>0) then
  begin
    for i:=1 to 3 do
      temp[i]:=
        strtoint(form1.stringgrid1.cells[i,day]);
    temp[trip]:=temp[trip]+seats;
```

```

    for i:=1 to 2 do
      for j:=i+1 to 3 do
        if temp[i]<temp[j] then
          begin
            swap:=temp[i];
            temp[i]:=temp[j];
            temp[j]:=swap;
          end;
        acceptbooking:=true;
        if (temp[1]>50) or (temp[2]>30)
           or (temp[3]>15) then
          acceptbooking:=false;
        case acceptbooking of
          true: imagel.picture.loadfromfile('yes2.bmp');
          false: imagel.picture.loadfromfile('no2.bmp');
        end;
        imagel.visible:=true;
      end;
    end;
end;

```

The loop:

```

    for i:=1 to 3 do
      temp[i]:=strtoint(form1.stringgrid1.cells[i,day]);

```

copies the current bookings for the required day into a temporary array.

We add the extra seats for the destination selected by the customer:

```

    temp[trip]:=temp[trip]+seats;

```

A *bubble sort* puts booking totals for the three trips into descending order:

```

    for i:=1 to 2 do
      for j:=i+1 to 3 do
        if temp[i]<temp[j] then
          begin
            swap:=temp[i];
            temp[i]:=temp[j];
            temp[j]:=swap;
          end;

```

Begin by assuming that the booking can be accepted:

```

    acceptbooking:=true;

```

The 50 seat coach will be allocated to the trip with the most passengers, the 30 seat coach to the second-busiest trip, and the 15 seat minibus to the trip with the least passengers. If there would be too many passengers for any of the buses, the extra booking cannot be accepted:

```

if (temp[1]>50) or (temp[2]>30) or (temp[3]>15) then
  acceptbooking:=false;

```

Depending on the outcome of the enquiry, we display either a green tick ('YES2.BMP') or a red cross ('NO2.BMP') in the image box:

```

case acceptbooking of
  true: image1.picture.loadfromfile('yes2.bmp');
  false: image1.picture.loadfromfile('no2.bmp');
end;

```

Make sure that the picture files YES2.BMP and NO2.BMP are copied into your COACHES directory, then build and run the program. Initially none of the trips have any bookings, so up to 50 people can be accepted for any destination:

If more than 50 seats are requested, a **cross** should be displayed. Check this, then return to the Delphi editing screen.

Once we know whether a booking can be accepted or not, the 'check availability' button can be replaced by the 'book' and 'cancel' buttons. Add the lines to do this:

```

.....
case acceptbooking of
  true: image1.picture.loadfromfile('yes2.bmp');
  false: image1.picture.loadfromfile('no2.bmp');
end;
image1.visible:=true;
button1.visible:=false;      {turn off 'check availability'}

```

```

        if acceptbooking=true then
            button2.visible:=true;      {turn on 'book'}
            button3.visible:=true;      {turn on 'cancel'}
        end;
    end;
end;

```

It will be convenient to set up a 'clear' procedure to blank out entries on Form2 ready for the next booking. Go to the bottom of the program and add the procedure:

```

procedure TForm2.clear;
begin
    radiogroup1.itemindex:=-1;
    radiogroup2.itemindex:=-1;
    seats:=0;
    edit1.text:='';
end;

```

Add 'clear' to the list of procedures near the top of the program:

```

.....
procedure Button1Click(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure clear;

```

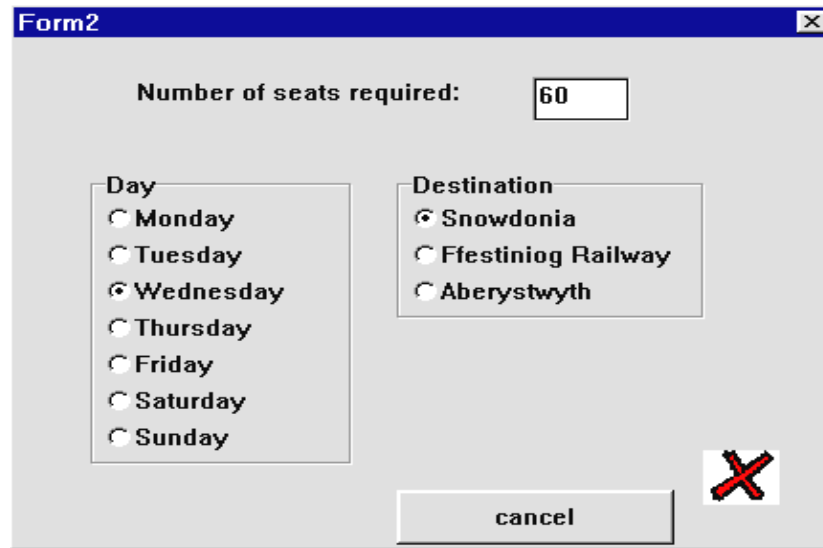
Bring *Form2* to the front and double-click the 'cancel' button to produce an event handler. Add the lines:

```

procedure TForm2.Button3Click(Sender: TObject);
begin
    image1.visible:=false;
    button2.visible:=false;
    button3.visible:=false;
    button1.visible:=true;
    clear;
end;

```

This simply resets the buttons and image box without altering any of the booking data.



Build and run the program. Enter a booking for more than 50 seats. The cross symbol should be displayed, plus the **'cancel'** button. Click *'cancel'* and check that the entries are blanked out.

Now enter a booking for less than 50 seats. This time the program should display the tick symbol and both the **'cancel'** and **'book'** buttons. Click *'cancel'*, then return to the Delphi editing screen.

Go to Form2 and double-click the **'book'** button to create an event handler. Add the lines:

```
procedure TForm2.Button2Click(Sender: TObject);
var
  previous,newtotal:integer;
begin
  image1.visible:=false;
  button2.visible:=false;
  button3.visible:=false;
  button1.visible:=true;
  previous:=
    strtoint(form1.stringgrid1.cells[trip,day]);
  newtotal:=previous+seats;
  form1.stringgrid1.cells[trip,day]:=
    inttostr(newtotal);
  form1.coachrecord.booking[day,trip]:=newtotal;
  clear;
end;
```

We begin by resetting the buttons. The next set of lines then add the new booking to the previous total for the day and destination. We obtain the total from the String Grid:

```
previous:=strtoint(form1.stringgrid1.cells[trip,day]);
```

The required number of seats are added:

```
newtotal:=previous+seats;
```

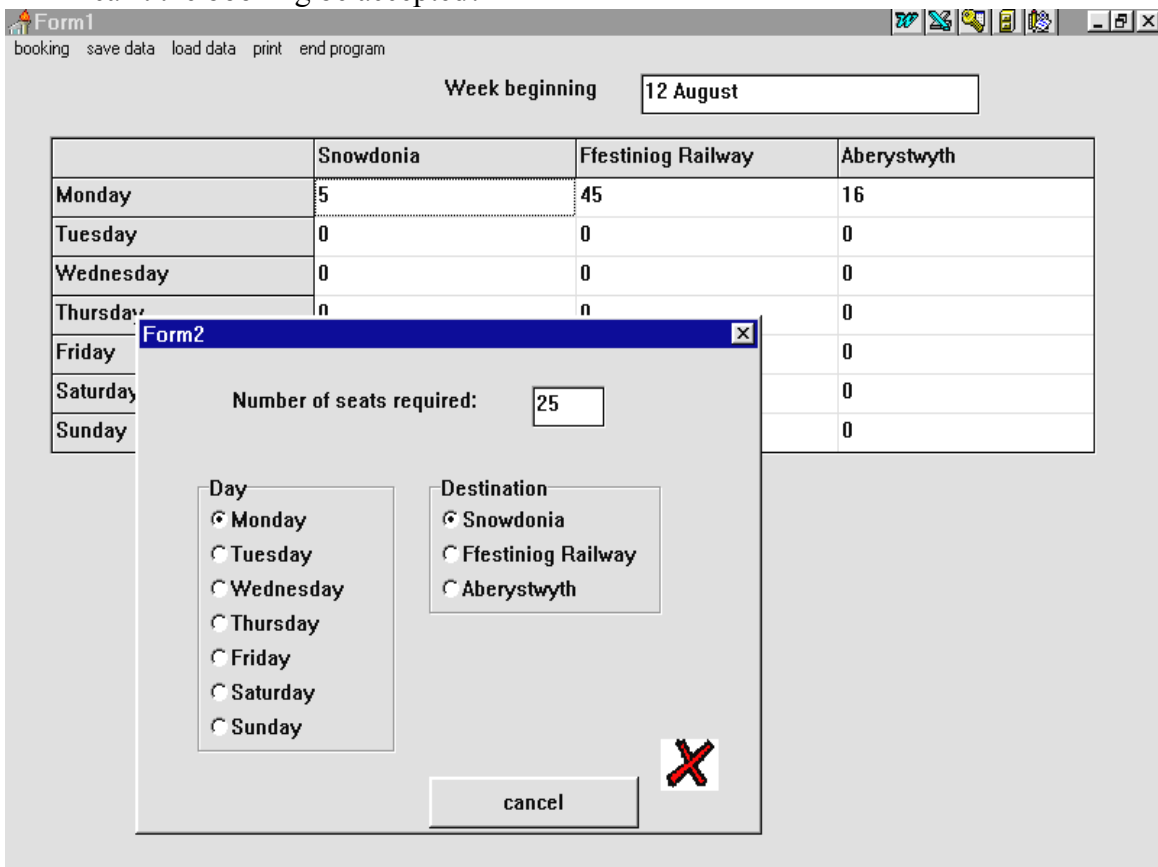
The new total is saved into the String Grid and the booking array:

```
form1.stringgrid1.cells[trip,day] := inttostr (newtotal);  
form1.coachrecord.booking[day,trip] := newtotal;
```

Build and run the program. Enter several bookings and check that the figures are transferred correctly to the string grid.

Try to devise test data to thoroughly check the program. You will need to include a variety of bookings which can and cannot be accepted. When you are convinced that the program works correctly, return to the Delphi editing screen.

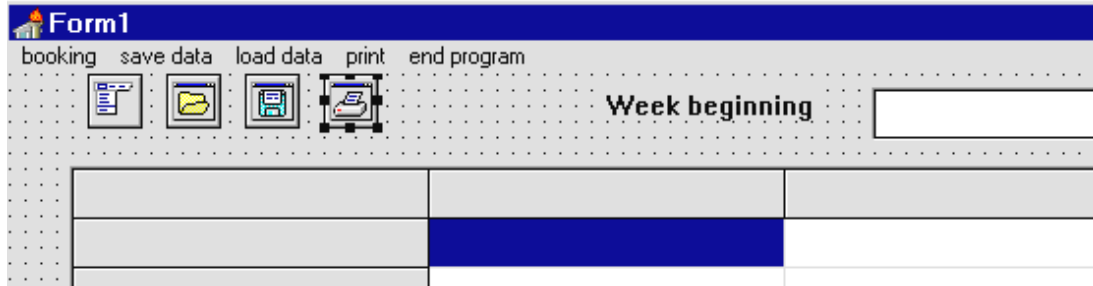
Question: A booking is shown in the illustration on the next page. Why can't the booking be accepted?





To finish the program, we need to set up the 'save data', 'load data' and 'print' options. These will be very similar to the supermarket sales program at the start of this chapter. Carry out the following steps:

Add '*OpenDialog*', '*SaveDialog*' and '*PrintDialog*' components to *Form1*:



We will begin with the 'save data' option. Click the '**SaveDialog**' icon and press ENTER to bring up the Object Inspector. Set the **DefaultExt** property to **DAT**. Double-click alongside the Filter property to open the Filter Editor window and add the entries:

Data files	*.dat
All files	*.*

Return to the *Form1* screen and click the 'save data' menu option to create an event handler. Add the program lines:

```
procedure TForm1.savedata1Click(Sender: TObject);
begin
  if savedialog1.execute then
  begin
    coachrecord.date:=edit1.text;
    assignfile(coachfile,savedialog1.filename);
    rewrite(coachfile);
    write(coachfile,coachrecord);
    closefile(coachfile);
  end;
end;
```

We can now move on to the 'load data' option. Click the '**OpenDialog**' icon on the *Form1* grid, then press ENTER to bring up the Object inspector. Enter the **Filter** items:

Data files	*.dat
All files	*.*

Click the '**load data**' menu option on *Form1* to create an event handler, then add the lines:

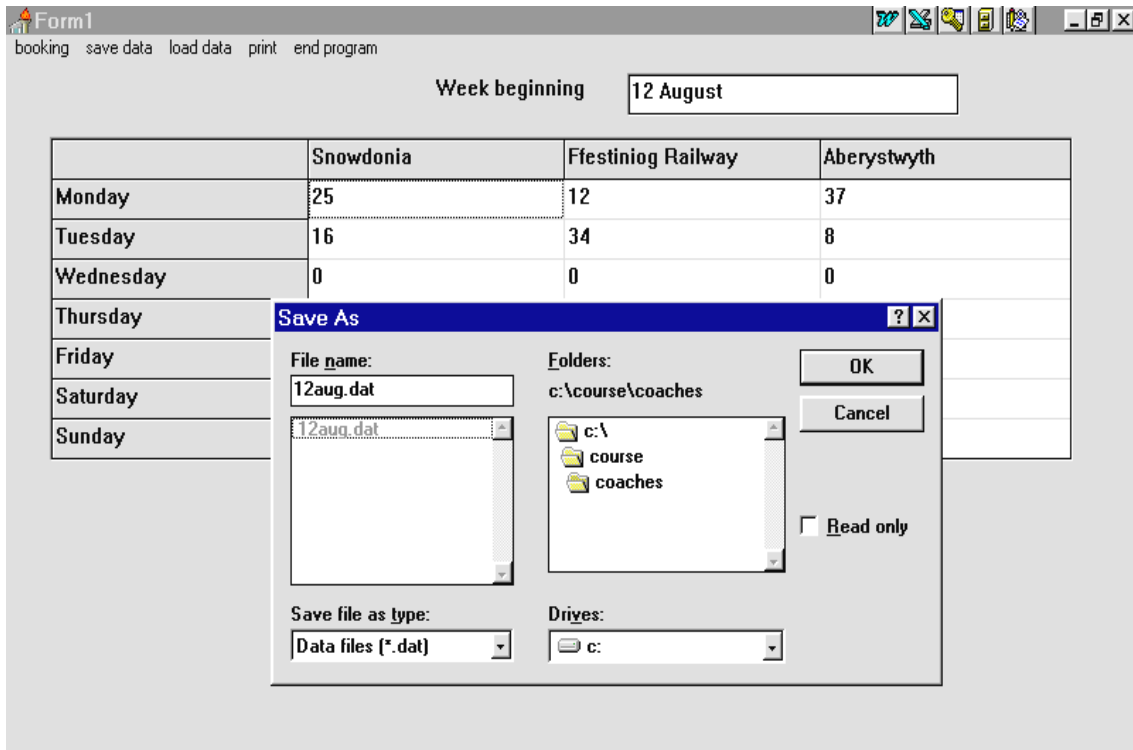
```

procedure TForm1.loaddata1Click(Sender: TObject);
var
  i,j:integer;
begin
  if opendialog1.execute then
  begin
    assignfile(coachfile,opendialog1.filename);
    reset(coachfile);
    read(coachfile,coachrecord);
    closefile(coachfile);
    edit1.text:=coachrecord.date;
    for i:=1 to 3 do
      for j:=1 to 7 do
        stringgrid1.cells[i,j]:=
          inttostr(coachrecord.booking[j,i]);
    end;
  end;
end;

```

Build and run the program. Enter test data as shown on the next page, then select the 'save data' option. Give a file name based on the 'week beginning' date, e.g.

12AUG.DAT



End the program and return to the Delphi editing screen. Re-run the program and select the 'load data' option. It should be possible to reload the saved data. Try adding extra bookings, resaving and reloading the data a

few times with different file names. When you are convinced that the program is working correctly, return to the Delphi editing screen.

It just remains to set up a procedure to print the table of bookings. Begin by adding **'printers'** to the **'uses'** list at the top of *Unit1*:

```
uses
  SysUtils, WinTypes, ...Menus, unit2, printers;
```

Click the **'print'** option on Form1 to produce an event handler, then add the program lines below. As before, the symbol □ represents a blank space to be typed with the space bar:

```
procedure TForm1.print1Click(Sender: TObject);
var
  printfile: textfile;
  textline, item: string;
  i, j: integer;
begin
  if printdialog1.execute then
  begin
    assignprn(printfile);
    rewrite(printfile);
    printer.canvas.font.name:='Courier New';
    printer.canvas.font.size:=12;
    writeln(printfile, '□□□□Week beginning:□'+
      edit1.text);

    writeln(printfile);
    writeln(printfile, '□□□□Day□□□□□□□□□□□□□□
      Snowdonia□□□□Ffestiniog□□Aberystwyth');
    for i:=1 to 7 do
    begin
      item:='□□□□'+stringgrid1.cells[0,i]+
        '□□□□□□□□□□□□□□';

      textline:=copy(item,1,14);
      for j:=1 to 3 do
      begin
        item:=
          '□□□□□□□□□□□□□□'+stringgrid1.cells[j,i];
        item:=copy(item,length(item)-12,13);
        textline:=textline+item;
      end;
      writeln(printfile, textline);
    end;
    closefile(printfile);
  end;
end;
end;
```

Build and run the program. Load a data file and select the **'print'** option.  
Check that a correctly aligned table is printed:

```
Week beginning: 12 August

Day           Snowdonia   Ffestiniog  Aberystwyth
Monday        25          12           37
Tuesday       16          34           8
Wednesday     0           0            0
Thursday      0           0            0
Friday        0           0            0
Saturday      0           0            0
Sunday        0           0            0
```