NINE

Graphs

In this chapter we will look at three ways in which data can be displayed graphically on the computer, using a histogram, a line graph and a pie graph.

Histogram

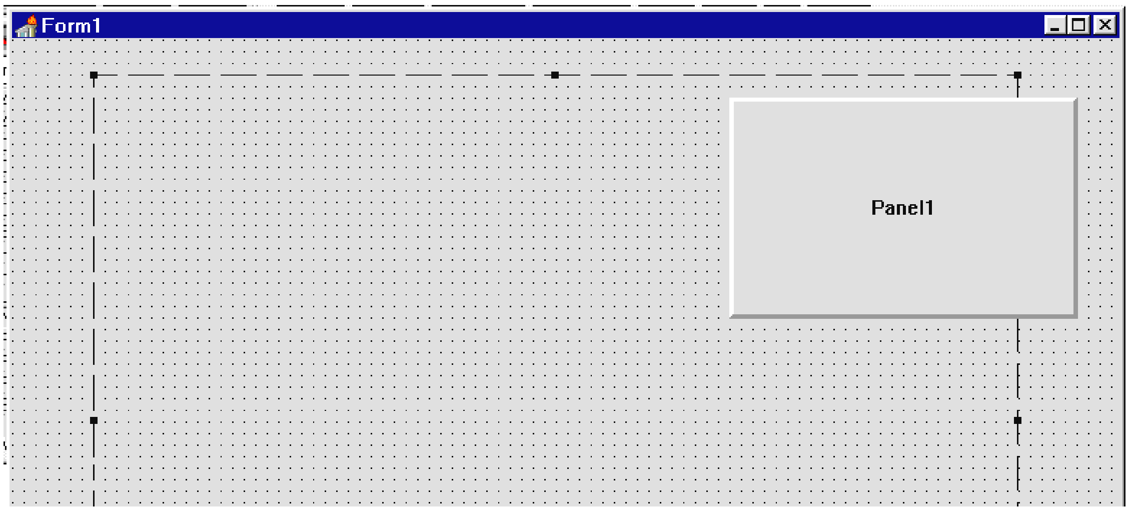
Suppose that the numbers of different types of vehicle passing a recording point in an hour were:

cars	368
heavy lorries	42
light vans	78
buses	12

A computer program is required to display the results of this traffic survey. A histogram (bar chart) would be a suitable method.

Set up a new sub-directory TRAFFIC and save a Delphi project into this. Use the Object Inspector to maximize the form, and drag the grid to nearly fill the screen.

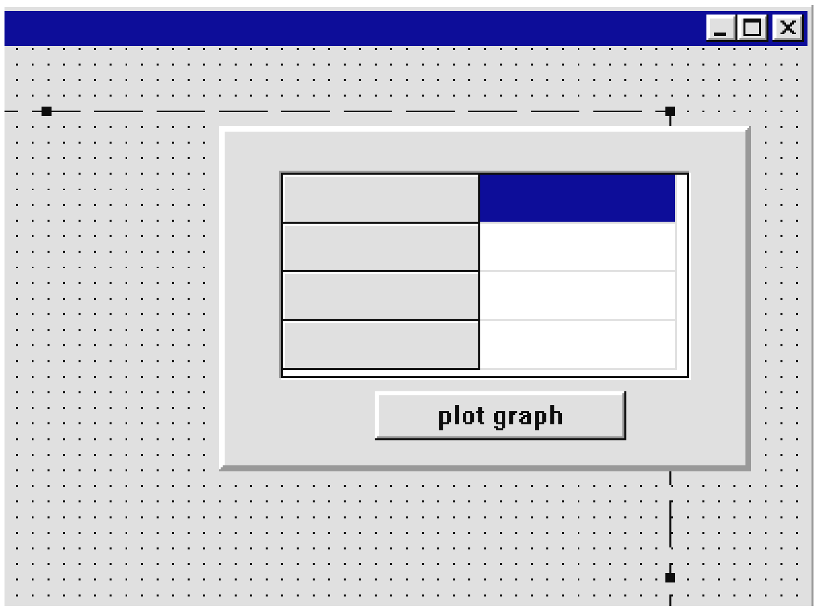
Add an **image** box. Click inside the image box and press ENTER to bring up the Object Inspector, then set the **Width** property to **640** and the **Height** property to **480**. Centre the image on the form. This will be the area in which the graph is drawn. Now add a panel so that it overlaps the top right corner of the image box:



The panel will be used for input of the traffic data. Place a string grid on the panel and set its properties:

Fixed Rows	**0**
ColCount	**2**
RowCount	**4**
DefaultColWidth	**100**
Options:	
goEditing	**True**
ScrollBars	**None**

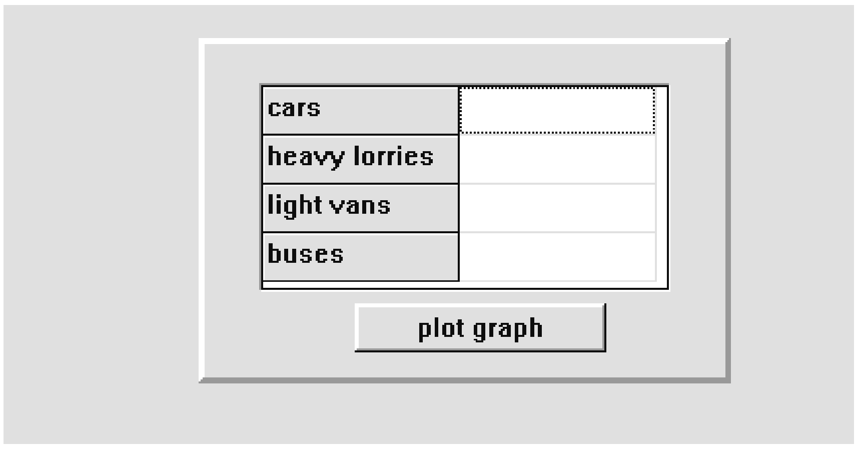
Also add a **button** component with the caption '**plot graph**'. Adjust the panel to a suitable size:



Double-click the dotted grid outside the image box to produce an 'OnCreate' event handler and add lines to display captions in the string grid:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  stringgrid1.cells[0,0]:='cars';
  stringgrid1.cells[0,1]:='heavy lorries';
  stringgrid1.cells[0,2]:='light vans';
  stringgrid1.cells[0,3]:='buses';
end;
```

Compile and run the program. Check that captions are displayed for the categories of traffic, then return to the Delphi editing screen.



We now need to set up an array to store the traffic figures when they are entered in the string grid. Make an entry in the '*public declarations*' section:

```
      public
         { Public declarations }
         vehicles:array[0..3]of integer;
      end;
```

Next we need a procedure to transfer data from the string grid into the **vehicles** 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 create an event handler. Add the lines:

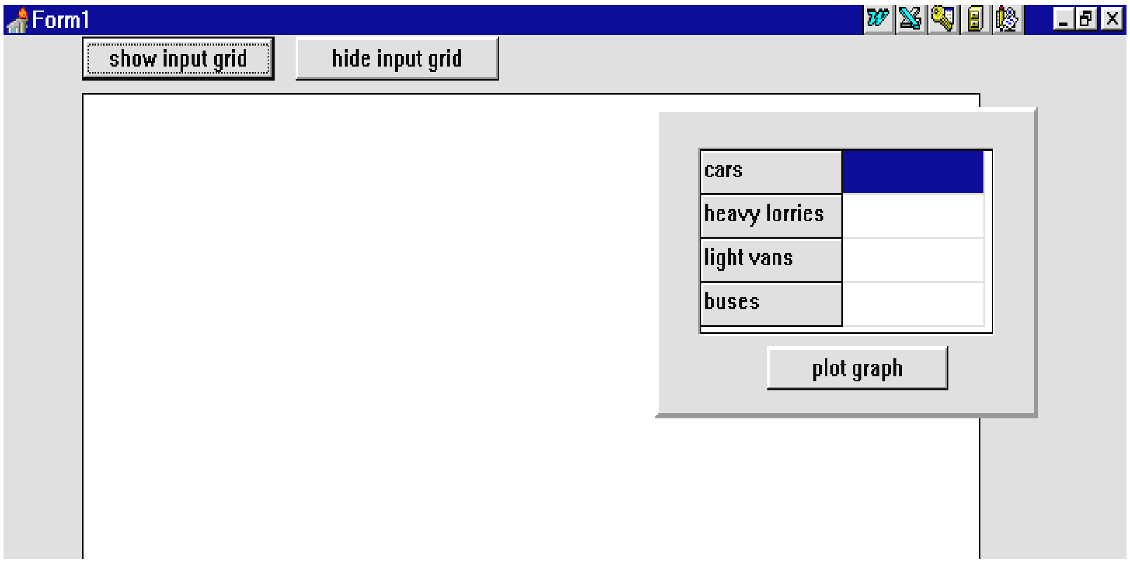
```
procedure TForm1.StringGrid1KeyUp(Sender:
TObject; var Key: Word; Shift: TShiftState);
var
  y:integer;
begin
  y:=stringgrid1.row;
  if stringgrid1.cells[1,y]='' then
    vehicles[y]:=0
  else
    vehicles[y]:=strtoint(stringgrid1.cells[1,y]);
end;
```

Compile and run the program to check that the error trapping works correctly for the string grid. Only integers (whole numbers) should be accepted. Return to the Delphi editing screen.

The graph drawing takes place when the button below the string grid is pressed. Double-click the button to create an event handler and add the lines shown below. The purpose of these are to set the fill colour to white, then draw a rectangle covering the image box:

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
end;
```

Compile and run the program, then press the '**plot graph**' button. A large white rectangle should appear with the panel 'floating' above it. Return to the Delphi editing screen.



When the program is running, it would be convenient if we could turn the panel on and off each time data is entered or a graph plotted. To do this, add two buttons at the top of the screen labelled '**show input grid**' and '**hide input grid**'. Create an event handler for the '*show input grid*' button and add the line:

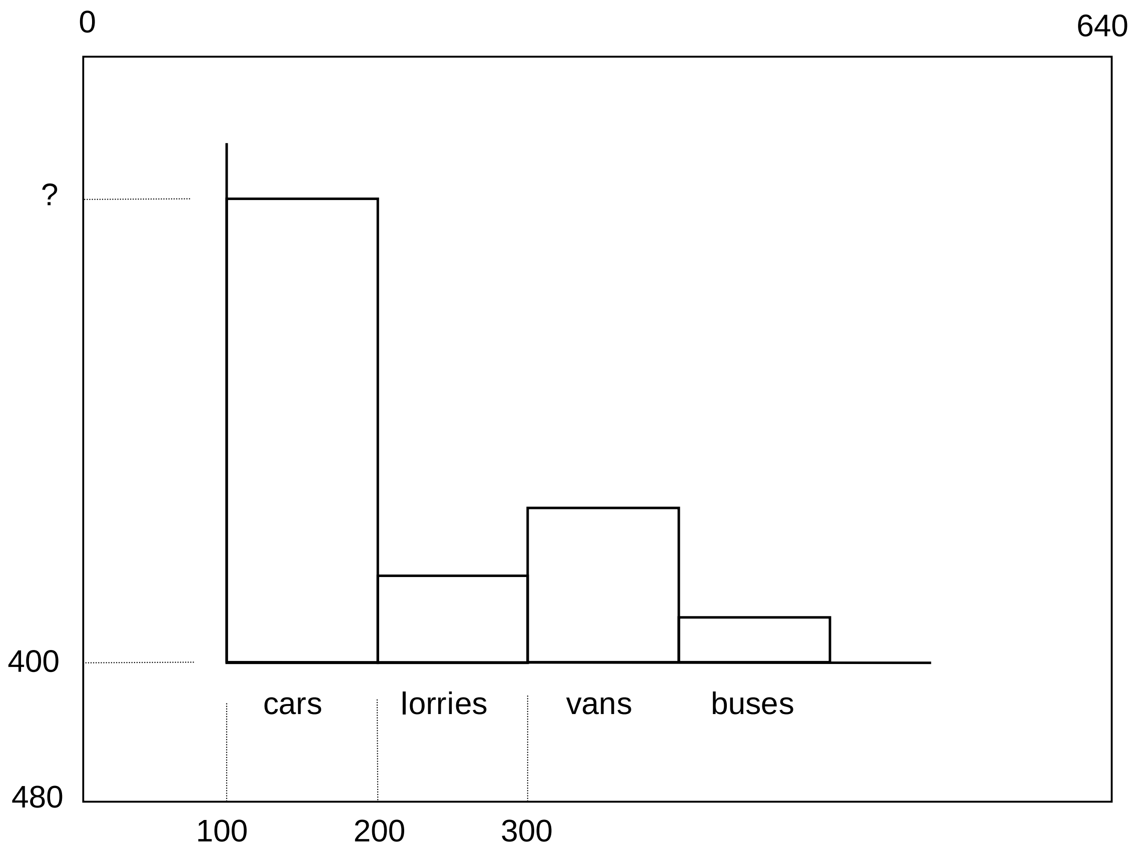
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  panel1.visible:=true;
end;
```

For the '*hide input grid*' button, create the event handler:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  panel1.visible:=false;
end;
```

Run the program to check that the panel can be turned on and off as required, then return to the Delphi editing screen.

The data entry part of the program is now completed and we can turn our attention to drawing the actual graph. From our earlier work on graphics, we know how to draw the lines and rectangles for the histogram. The main problem is deciding how to fit the histogram onto the screen within the 640 horizontal grid squares and the 480 vertical grid squares of the image area.



We will need to leave space below the graph for the names of the types of vehicle. It should be all right to start the columns from a *base line* 400 units down.

We need to fit four columns into the width of the image box. A suitable column *width* is 100 units, with the first column 100 units from the side of the box.

The next question is to decide how *high* to make each of the columns. One solution is to make the tallest column a fixed height, say 300 units, then scale all the other columns accordingly. To do this, the computer will have to check the data to find the vehicle total for the highest column - from the test data given above this would be 368 cars. We then calculate a **scaling factor**. If we multiply each of the results by this scaling factor, it will convert vehicle totals to screen units and give the bar heights for the histogram:

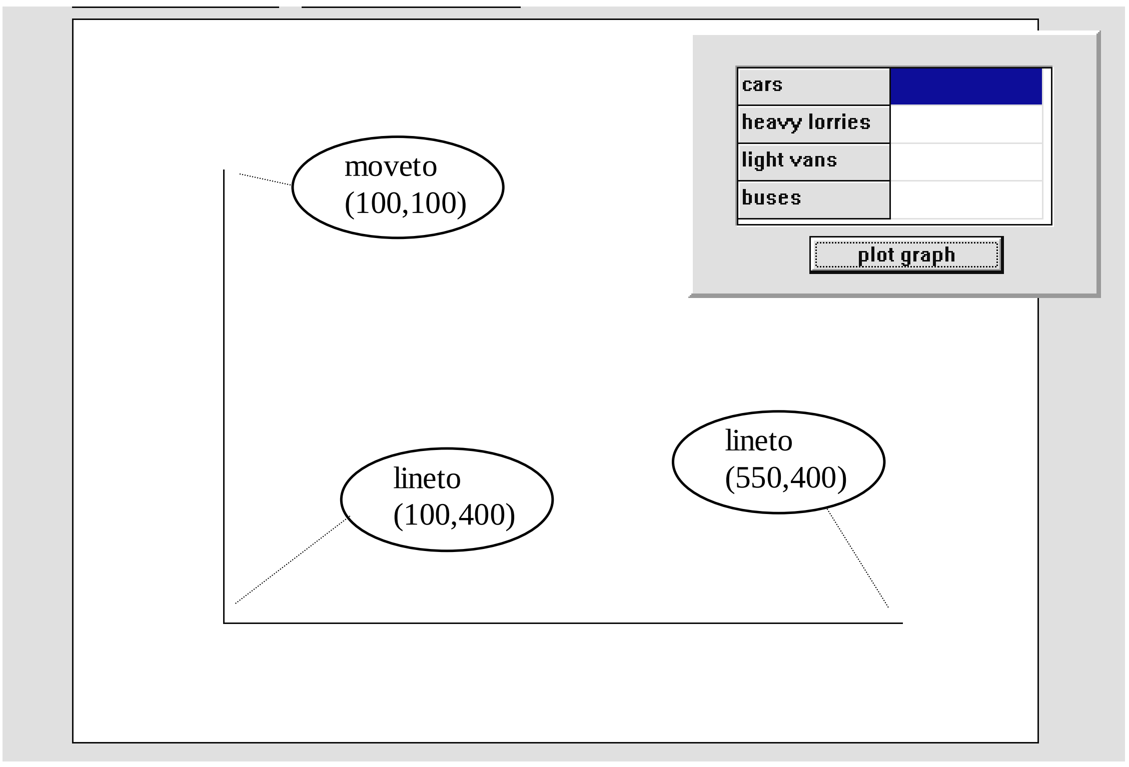
scaling factor = largest bar height / largest vehicle total
= 300 / 368

Begin the graph by drawing the axes. Add lines to the '**plot graph**' button procedure to draw the vertical and horizontal lines:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(100,100);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(550,400);
end;
```

ADD THIS

Compile and run the program. Press the 'plot graph' button and the axes should appear:



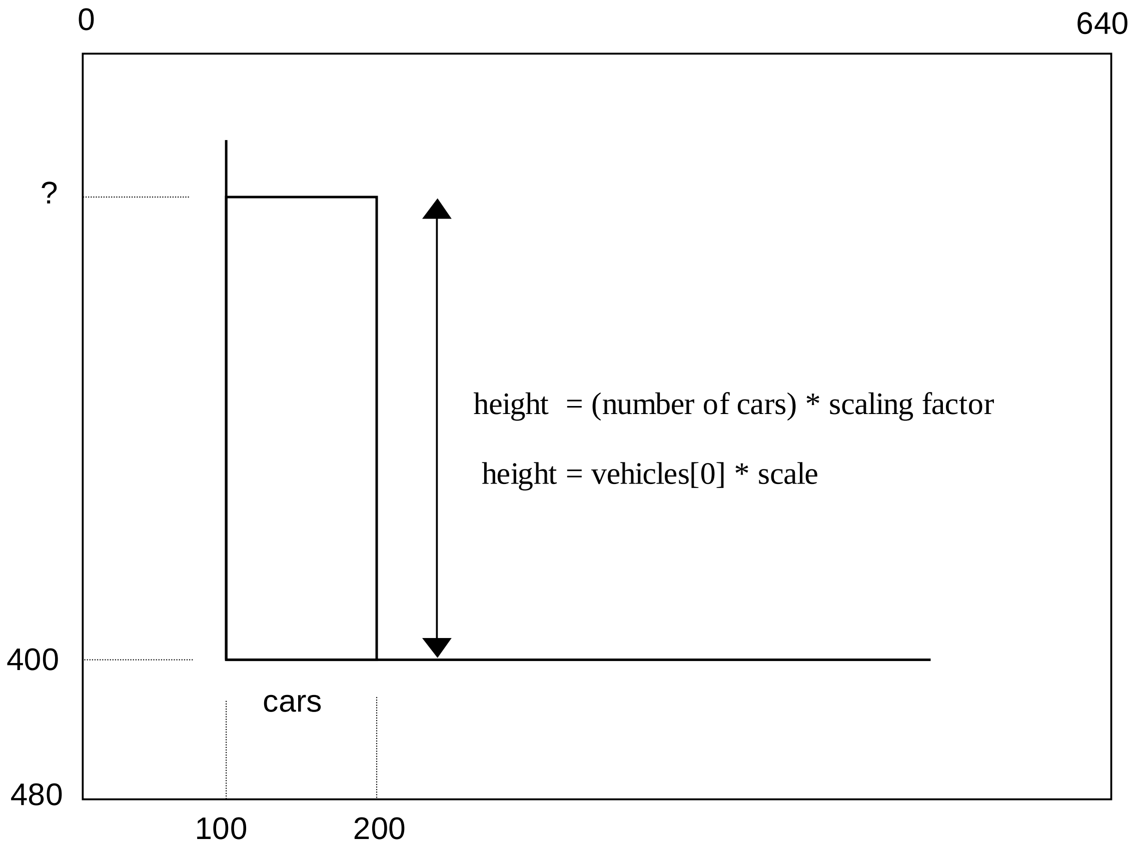
The next step is to write a section of program to calculate the **scaling factor** for converting vehicle totals into heights for the columns. This uses the technique introduced in the last chapter to find the maximum of the numbers in the vehicle array. We begin by making **vehicles[0]** the maximum so far, then a loop checks each of the remaining entries to see if a larger figure can be found. Add the following lines to the 'plot graph' event handler:

```
     . . . . . .
  image1.canvas.moveto(100,100);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(550,400);
  max:=vehicles[0];
  for count:=1 to 3 do
    if vehicles[count]> max then
       max:=vehicles[count];
  scale:=300/max;
end;
```

ADD THIS

In the final step, the chosen graph height of 300 screen units has been divided by the maximum vehicle total to give the **scaling factor**.

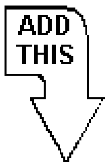
We are now ready to draw the first of the columns on the histogram. This will begin at 100 units across.


0
640
?
height = (number of cars) * scaling factor
height = vehicles[0] * scale
400
cars
480
100
200

The base line will be at 400 units, and the height of the column will depend on the number of cars recorded in the array as **vehicles[0]**.

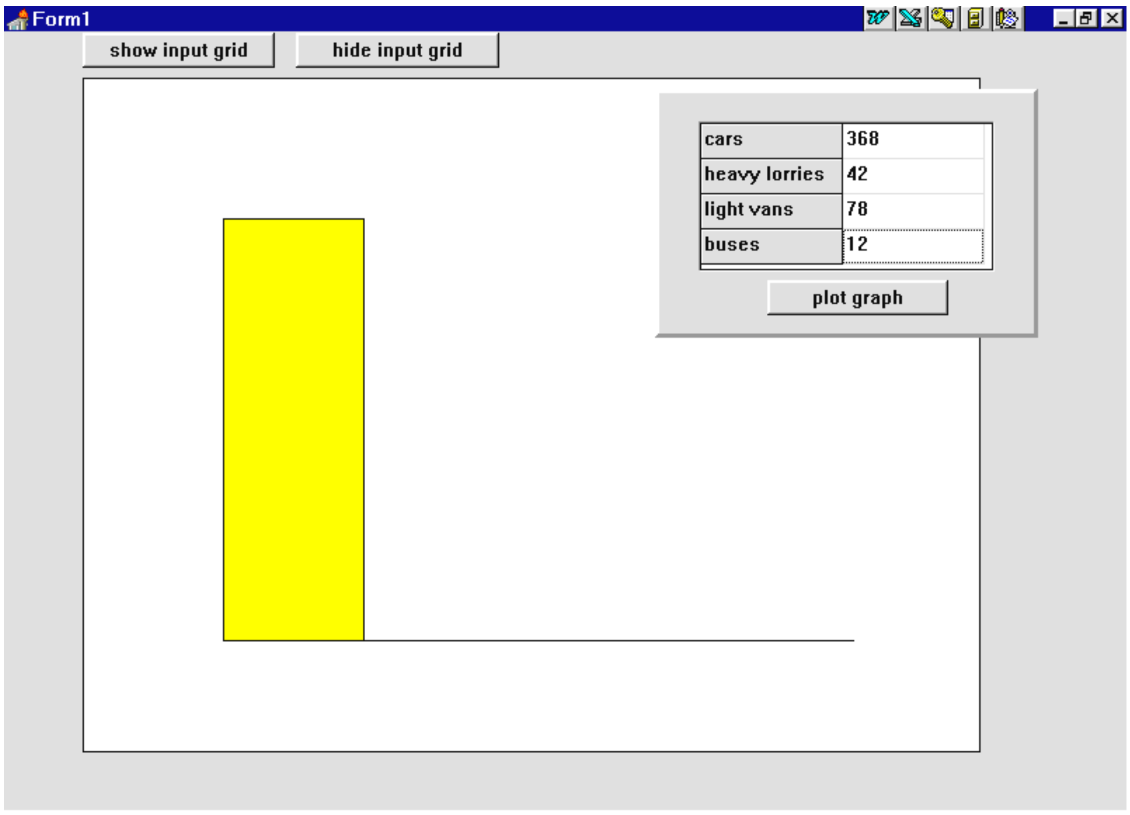
Add the lines below to draw the column. To simplify the calculation, we have calculated the height and set a value for the across position of the column before the rectangle is drawn:

```
  max:=vehicles[0];
  for count:=1 to 3 do
    if vehicles[count]> max then
       max:=vehicles[count];
  scale:=300/max;
  image1.canvas.brush.color:=clRed;
  height:=round(vehicles[0]*scale);
  startx:=100;
  image1.canvas.rectangle(startx,401,
                               startx+101,400-height);
end;
```



Variables will need to be listed at the start of the procedure:

```
  procedure TForm1.Button1Click(Sender: TObject);
  var
    max,count,height,startx:integer;
    scale:real;
```

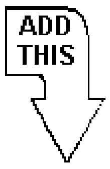


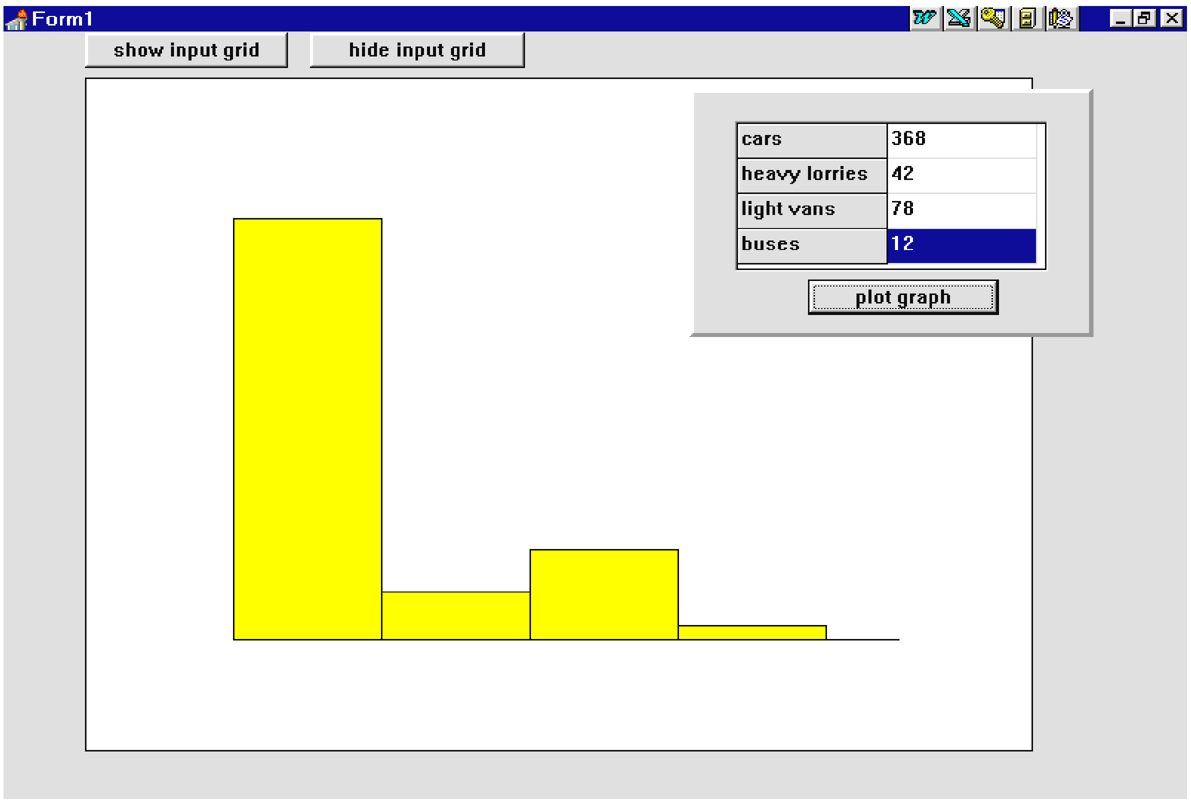
cars	368
heavy lorries	42
light vans	78
buses	12

Compile and run the program using the test data and press the '**plot graph**' button. The '**cars**' column should be plotted as shown above.

The other three columns can be plotted in a similar way. Each column starts 100 units further across the screen. Add lines of program to do this; you may find it easiest to use the Edit/Copy/Paste facility to do this:

```
   image1.canvas.brush.color:=clRed;
   height:=round(vehicles[0]*scale);
   startx:=100;
   image1.canvas.rectangle(startx,401,
                                 startx+101,400-height);
   height:=round(vehicles[1]*scale);
   startx:=200;
   image1.canvas.rectangle(startx,401,
                                 startx+101,400-height);
   height:=round(vehicles[2]*scale);
   startx:=300;
   image1.canvas.rectangle(startx,401,
                                 startx+101,400-height);
   height:=round(vehicles[3]*scale);
   startx:=400;
   image1.canvas.rectangle(startx,401,
                                 startx+101,400-height);
end;
```





cars	368
heavy lorries	42
light vans	78
buses	12

Compile and run the program with the test data. Return to the Delphi editing screen.

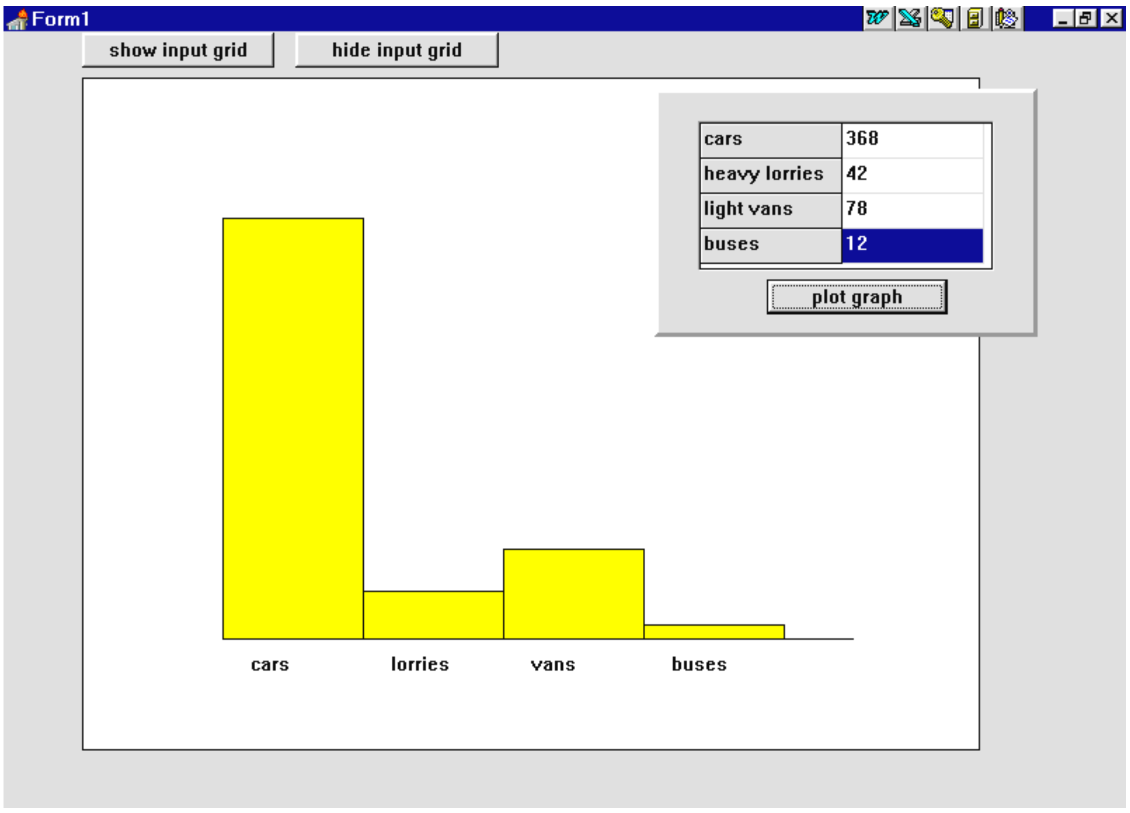
The graph should be drawn correctly, but the program is longer it needs to be. We could produce the same graph more neatly with a FOR..TO..DO loop - this would repeat four times to draw each of the columns in turn. The loop counter variable could be used to calculate the start position for each of the columns.

To demonstate how this would work, delete the lines of program which draw the columns and replace them by the loop shown below. The procedure becomes:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  max,count,height,startx:integer;
  scale:real;
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(100,100);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(550,400);
  max:=vehicles[0];
  for count:=1 to 3 do
    if vehicles[count]> max then
       max:=vehicles[count];
  scale:=300/max;
  for count:=0 to 3 do
  begin
    image1.canvas.brush.color:=clRed;
    height:=round(vehicles[count]*scale);
    startx:=100+100*count;
    image1.canvas.rectangle(startx,401,
                            startx+101,400-height);
  end;
end;
```

Check that the program gives the same result with the test data.

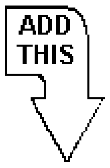
Our final task to complete the graph is to add captions along the horizontal axis for the types of vehicle, as shown below. This can be done by the same loop that draws each of the columns.



cars	368
heavy lorries	42
light vans	78
buses	12

Add the lines shown:

```
  for count:=0 to 3 do
  begin
    image1.canvas.brush.color:=clRed;
    height:=round(vehicles[count]*scale);
    startx:=100+100*count;
    image1.canvas.rectangle(startx,401,
                                 startx+101,400-height);
    case count of
      0: caption:='cars';
      1: caption:='lorries';
      2: caption:='vans';
      3: caption:='buses';
    end;
    image1.canvas.brush.color:=clWhite;
    image1.canvas.textout(startx+20,410,caption);
  end;
end;
```



The CASE command uses the loop counter to choose the correct caption, then the TEXTOUT command puts it onto the screen.

It will be necessary to add '**caption**' to the list of variables at the start of the procedure:

```
var
  max,count,height,startx:integer;
  scale:real;
  caption:string;
```

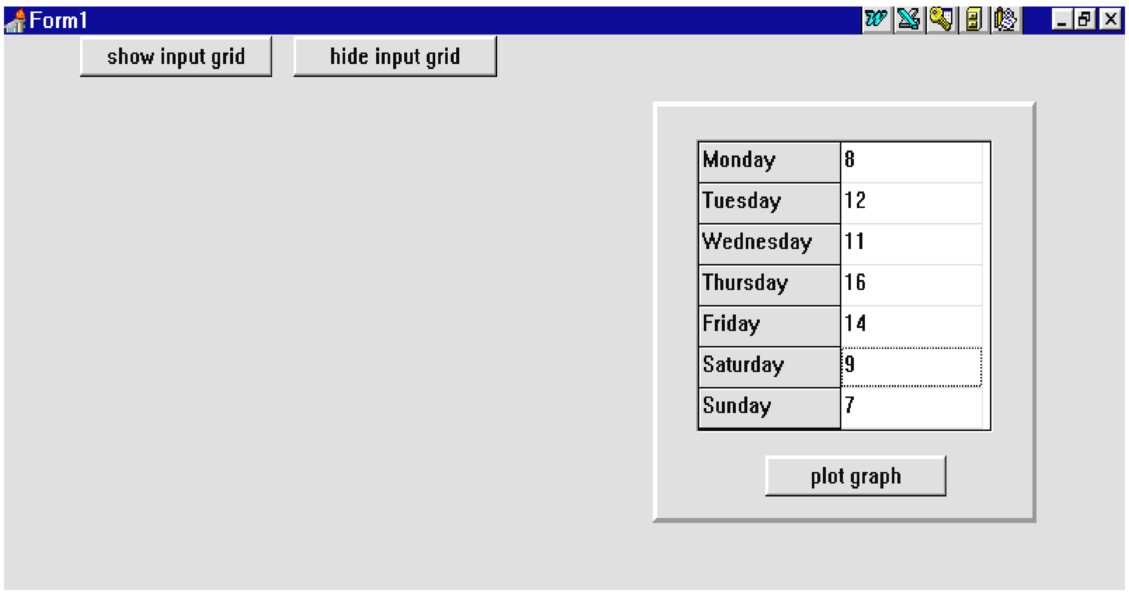
Compile and run the completed progam, and test this with a variety of vehicle totals.

Line graph

In our next program we will produce a line graph of temperature recordings by a weather station during a seven day period. The input section of the program will be very similar to the traffic survey.

Begin by creating a new directory TEMP and save a Delphi project into it. Use the Object Inspector to maximize the form, and drag the grid to nearly fill the screen.

Add an **image** box. Click inside the image box and press ENTER to bring up the Object Inspector, then set the **Width** property to **640** and the **Height** property to **480**. Centre the image on the form. This will be the area in which the graph is drawn. Now add a panel so that it overlaps the top right corner of the image box:



Monday	8
Tuesday	12
Wednesday	11
Thursday	16
Friday	14
Saturday	9
Sunday	7

The panel will be used for input of the daily temperature data. Place a string grid on the panel and set its properties:

Fixed Rows	**0**
ColCount	**2**
RowCount	**7**
DefaultColWidth	**100**
Options:	
goEditing	**True**
ScrollBars	**None**

Also add a **button** component with the caption '**plot graph**'. Adjust the panel to a suitable size.

Double-click the dotted grid outside the image box to produce an 'OnCreate' event handler and add lines to display captions in the string grid:

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

Now add two buttons above the image box with the captions '**show input grid**' and '**hide input grid**'. These will work in the same was as in the traffic survey program. Double click the '*show input grid*' button and add a line to the event handler:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  panel1.visible:=true;
end;
```

For the '*hide input grid*' button, create the event handler:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  panel1.visible:=false;
end;
```

We now need to set up an array to store the temperature figures when they are entered in the string grid. Make an entry in the '*public declarations*' section:

```
      public
         { Public declarations }
         temp:array[0..6]of integer;
      end;
```

Next we will set up a procedure to transfer data from the string grid into the **temp** 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 create 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]='') or
     (stringgrid1.cells[1,y]='-') then
    temp[y]:=0
  else
    temp[y]:=strtoint(stringgrid1.cells[1,y]);
end;
```

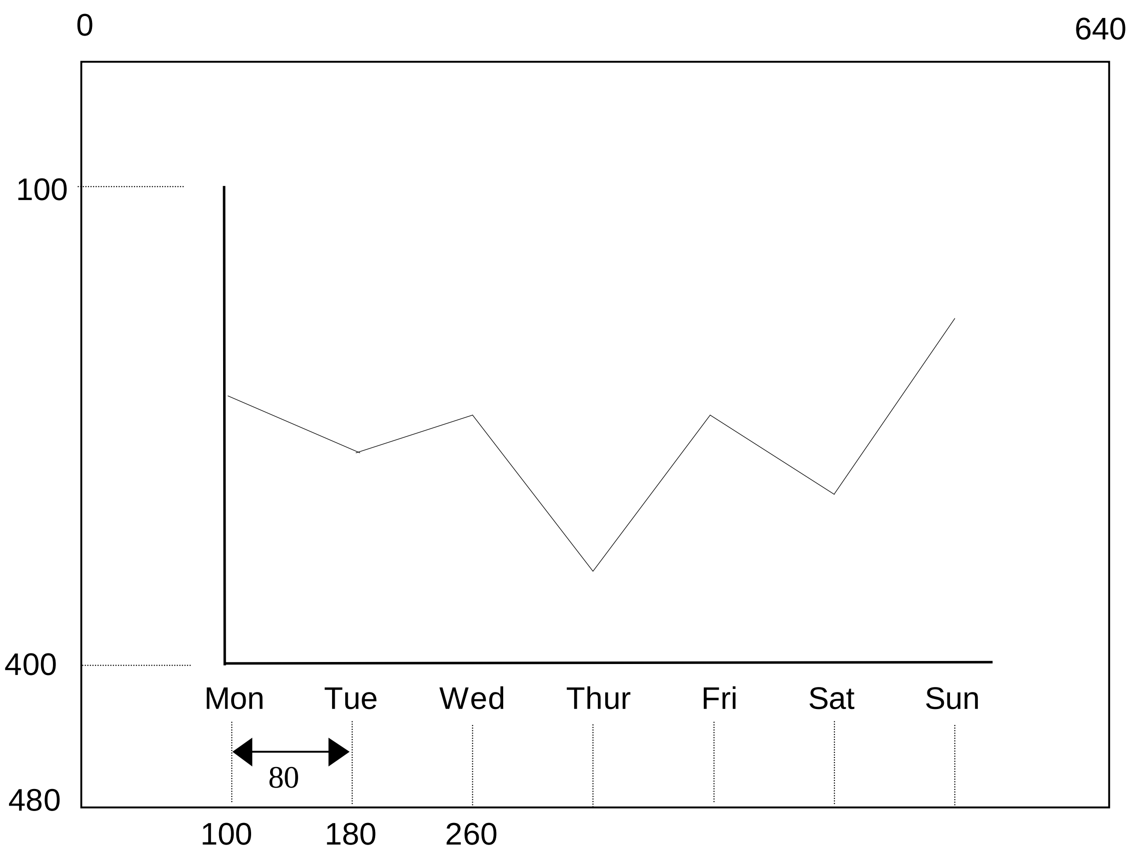
This is slightly more complicated than procedures we have written before to transfer numbers into an array because negative values have to be accepted. The extra part of the IF... line allows the user to type a minus sign in the string grid without an error message appearing.

Compile and run the program to check that the error trapping works correctly for the string grid. Only integers (whole numbers) should be accepted. Return to the Delphi editing screen.

We can turn our attention to the actual graph. The drawing area in the image box has again been set as 640 horizontal grid squares by 480 vertical grid squares.

We will need to leave space below the horizontal axis for the names of the days, so we can make the base line of the graph 400 units down.

A whole week's data will fit conveniently on the graph if the horizontal distance between days is 80 units, with the vertical axis 100 units from the edge of the image box, as in the diagram below:



Double-click the 'plot graph' button on the panel to create an event handler. Add the following lines:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x:integer;
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(100,100);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(580,400);
  for x:=0 to 6 do
  begin
    image1.canvas.moveto(100+80*x,400);
    image1.canvas.lineto(100+80*x,410);
  end;
end;
```

The line:

```
image1.canvas.brush.color:=clWhite;
```

sets the fill colour to white, then

```
image1.canvas.rectangle(0,0,640,480);
```

draws the white rectangle for the graph area.

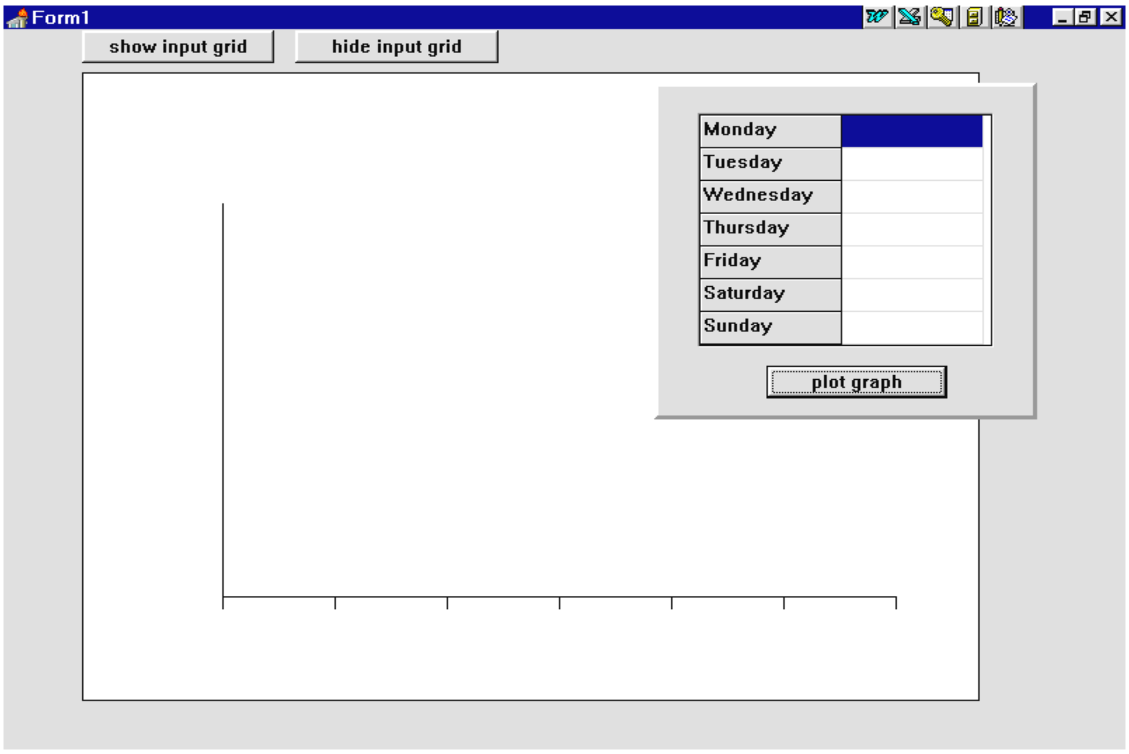
The next group of lines:

```
image1.canvas.moveto(100,100);
image1.canvas.lineto(100,400);
image1.canvas.lineto(580,400);
```

draw the vertical and horizontal axes.

The final part of the procedure is a loop to draw graduation lines on the horizontal axis for each day:

```
for x:=0 to 6 do
begin
  image1.canvas.moveto(100+80*x,400);
  image1.canvas.lineto(100+80*x,410);
end;
```



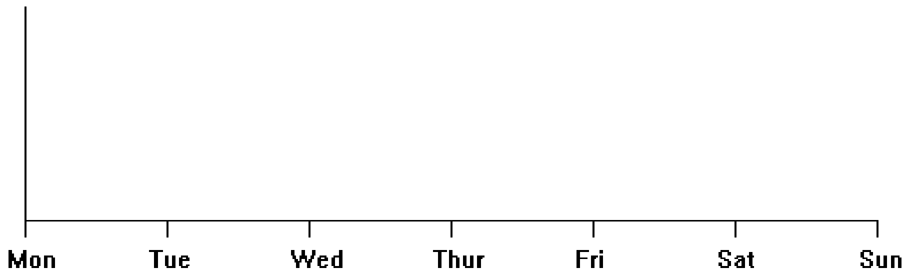
Monday	
Tuesday	
Wednesday	
Thursday	
Friday	
Saturday	
Sunday	

Compile and run the program. Press the 'plot graph' button to check that the axes and graduations are drawn correctly, then return to the Delphi editing screen.

Names for the days can be written below the graduation lines. We will use a CASE structure similar to the traffic survey program. Add lines to the loop so that the procedure becomes:

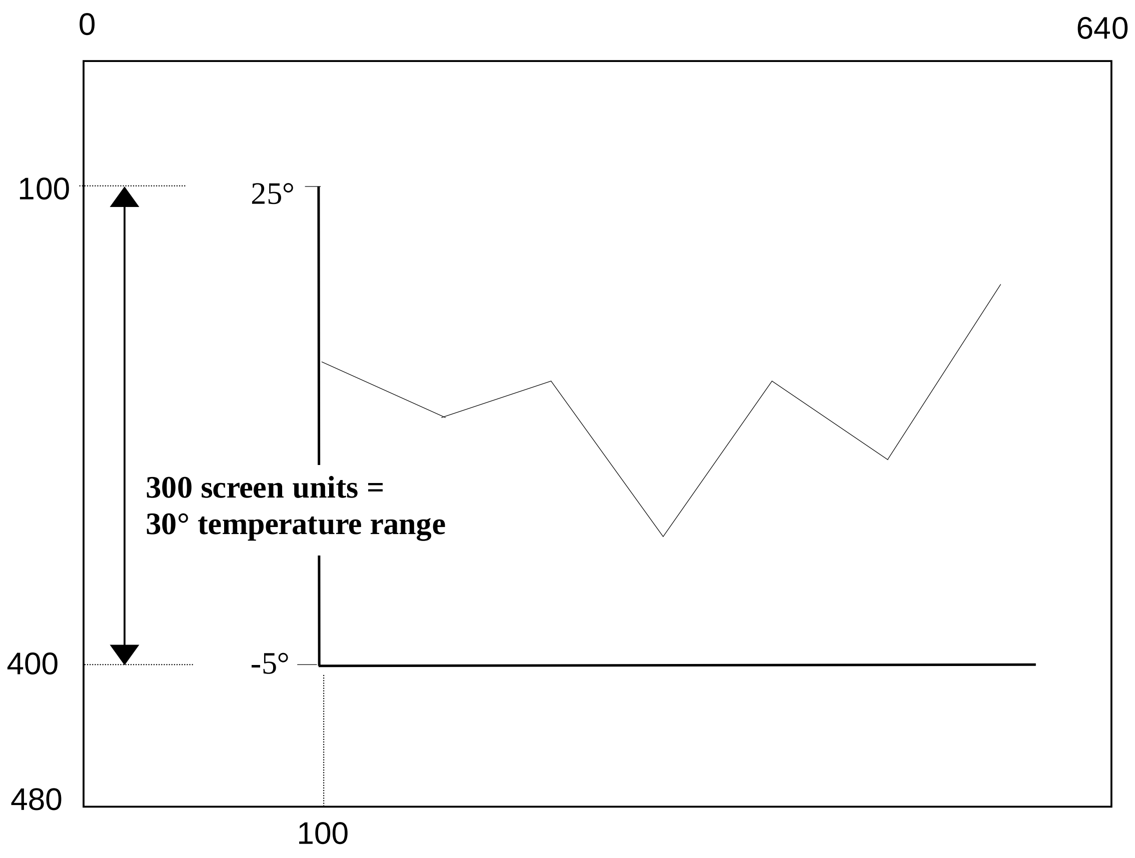
```
procedure TForm1.Button1Click(Sender: TObject);
var
  x:integer;
  day:string;
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(100,100);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(580,400);
  for x:=0 to 6 do
  begin
    image1.canvas.moveto(100+80*x,400);
    image1.canvas.lineto(100+80*x,410);
    case x of
      0:day:='Mon';
      1:day:='Tue';
      2:day:='Wed';
      3:day:='Thur';
      4:day:='Fri';
      5:day:='Sat';
      6:day:='Sun';
    end;
    image1.canvas.textout(90+80*x,414,day);
  end;
end;
```

Compile and run the program. Press the 'plot graph' button to check that the captions for the days are printed.



Return to the Delphi editing screen. We must now consider the vertical axis.

Let us assume that temperatures recorded at the weather station will be in the range -5° to +25° :



We can put graduation lines on the vertical axis in a similar way to the horizontal axis. Divisions at intervals of 5° would be suitable. Add another loop to the 'draw graph' event handler to do this:

```
      .........
     5:day:='Sat';
     6:day:='Sun';
     end;
     image1.canvas.textout(90+80*x,414,day);
   end;
   for y:=0 to 6 do
   begin
     image1.canvas.moveto(100,400-50*y);
     image1.canvas.lineto(80,400-50*y);
     caption:=inttostr(y*5-5);
     image1.canvas.textout(60,393-50*y,caption);
   end;
   image1.canvas.textout(10,220,'deg.C');
end;
```

ADD THIS

The variables **y** and **caption** will need to be added at the start of the procedure:

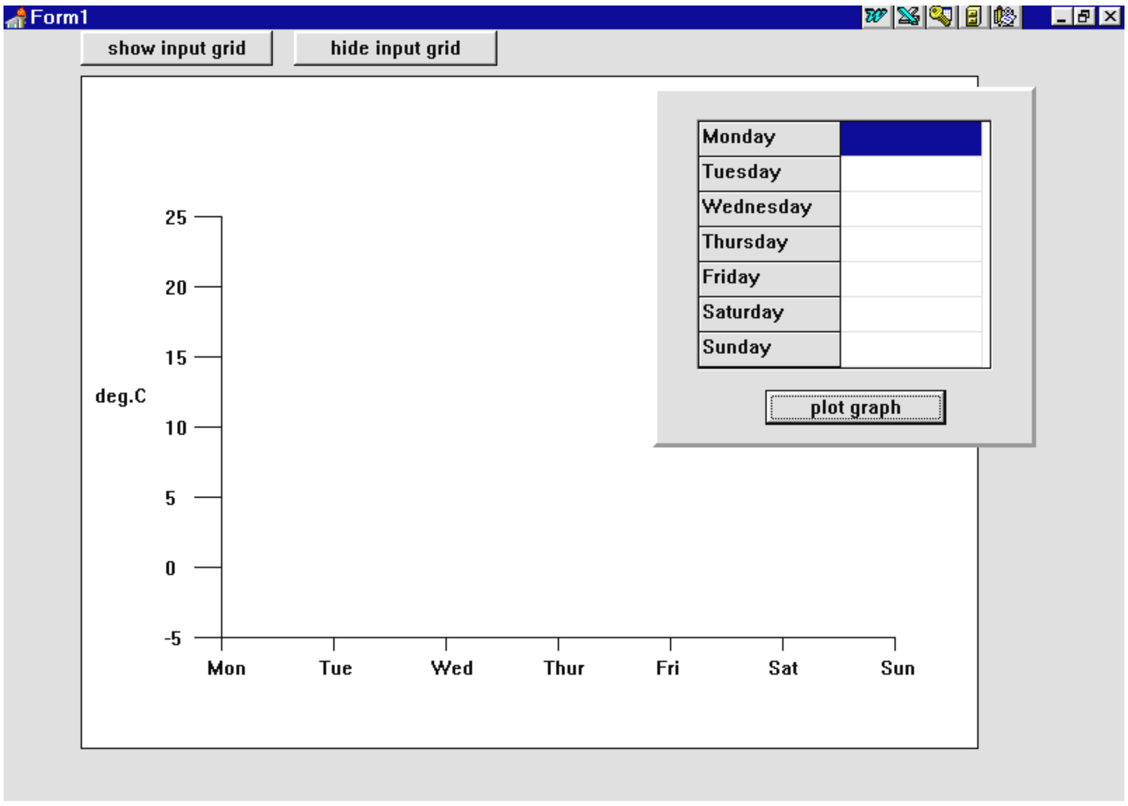
```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,y:integer;
  day,caption:string;
```

The loop repeats seven times to produce each of the graduation lines in turn. The loop counter, **y,** is used to calculate how far above the base line each graduation is drawn:

```
for y:=0 to 6 do
 begin
  image1.canvas.moveto(100,400-50*y);
  image1.canvas.lineto(80,400-50*y);
```

The loop counter value is used to generate the temperature numbers. The **y** value is multiplied by 5 because the graduations are at intervals of 5 degrees. We also subtract 5, so that the first graduation begins at -5 degrees:

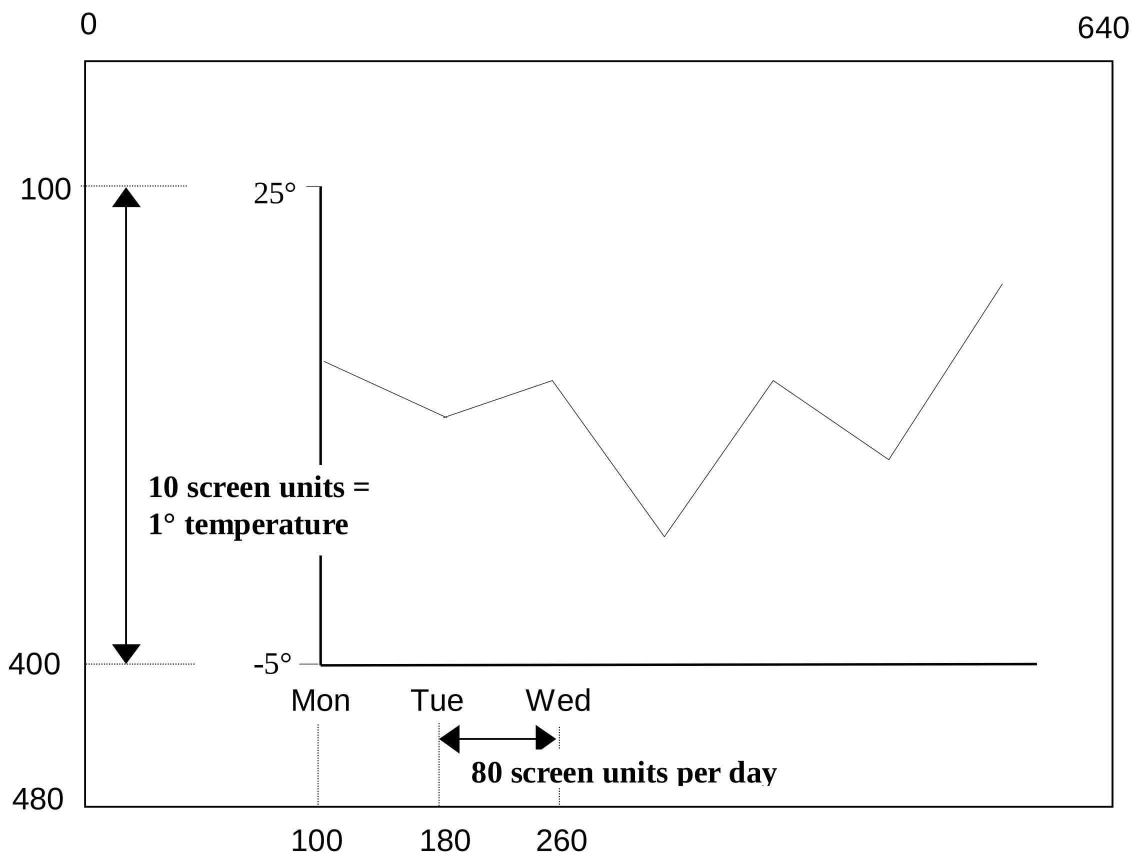
```
  caption:=inttostr(y*5-5);
  image1.canvas.textout(60,393-50*y,caption);
```



Monday	
Tuesday	
Wednesday	
Thursday	
Friday	
Saturday	
Sunday	

Compile and run the program to check that the graduations on the vertical axis are displayed correctly. Return to the Delphi editing screen.

The final stage in the program is to draw the line on the graph, using the input temperature values.



The scaling factor will be 10 screen units for each degree Centigrade. For each day on the graph, it is necessary to move 80 screen units to the right.

We can draw the line by moving to the starting point for Monday, then use a loop to continue for each of the remaining six days. Begin by adding a variable '**count**':

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x,y,count:integer;
  day,caption:string;
```

then insert the lines of program shown below:

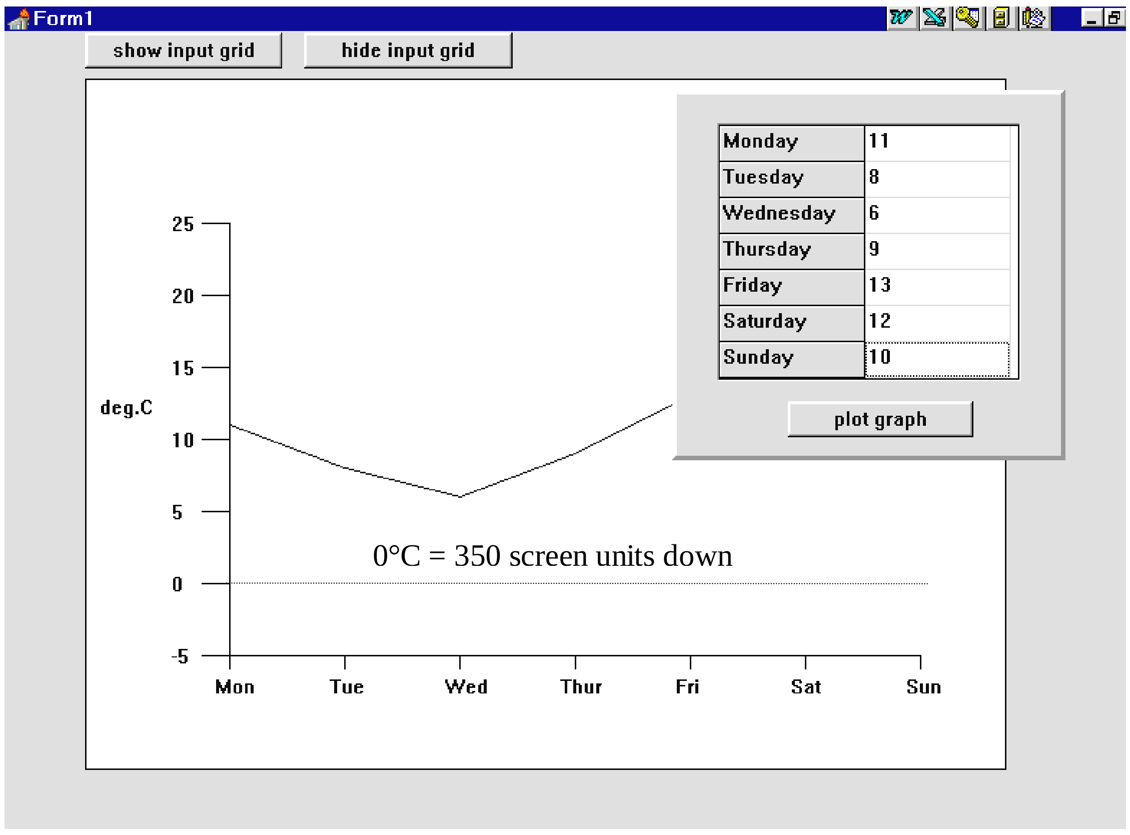
```
 for y:=0 to 6 do
  begin
    image1.canvas.moveto(100,400-50*y);
    image1.canvas.lineto(80,400-50*y);
    caption:=inttostr(y*5-5);
    image1.canvas.textout(60,393-50*y,caption);
  end;
  image1.canvas.textout(10,220,'deg.C');
  image1.canvas.moveto(100,350-10*temp[0]);
  for count:=1 to 6 do
    image1.canvas.lineto(100+80*count,
                              350-10*temp[count]);
end;
```

ADD THIS

Notice that the line drawing commands use a baseline of 350 units. This is the 0° C position on the graph:

image1.canvas.moveto(100,350-10*temp[0]);

baseline position



Monday	11
Tuesday	8
Wednesday	6
Thursday	9
Friday	13
Saturday	12
Sunday	10

0°C = 350 screen units down

The starting position of the line is calculated using the temperature reading for Monday, stored as **temp[0].**

The loop counter is used to select the correct value from the **temp** array for each of the other days:

```
for count:=1 to 6 do                                    loop counter
    image1.canvas.lineto(100+80*count,350-10*temp[count]);
```

Compile and run the completed program. Try this out with a variety of temperature values in the range -5° to 25°C.

Pie graph

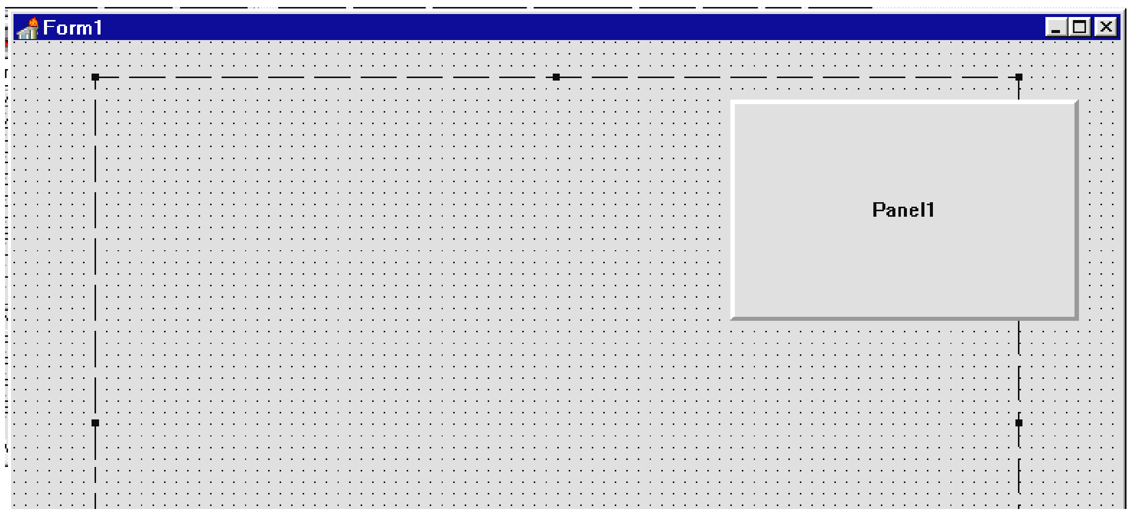
A third way of displaying data is by means of a pie graph which shows a circle divided into different categories. A typical use of a pie graph is to illustrate the numbers of votes for different candidates in an election. For example:

Conservative	368
Labour	1642
Liberal Democrat	784
Plaid Cymru	2126

A computer program is required to display these results.

Set up a new sub-directory ELECTION and save a Delphi project into this. Use the Object Inspector to maximize the form, and drag the grid to nearly fill the screen.

We will start to set up the program in a similar way to previous graphs:

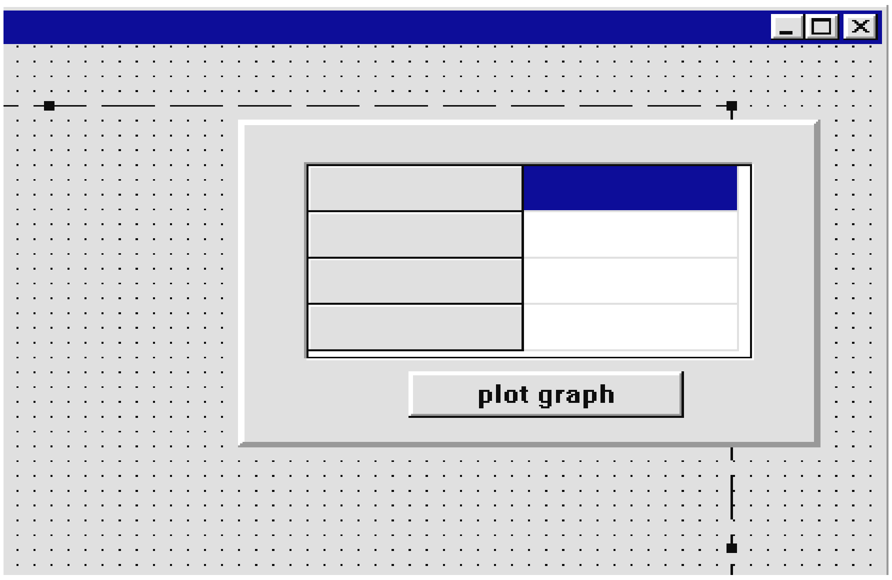


Add an **image** box to the form. Click inside the image box and press ENTER to bring up the Object Inspector, then set the **Width** property to **640** and the **Height** property to **480**. Centre the image on the form. This will be the area in which the graph is drawn. Now add a panel so that it overlaps the top right corner of the image box.

The panel will be used for input of the election results. Place a string grid on the panel and set its properties:

Fixed Rows	**0**
ColCount	**2**
RowCount	**4**
DefaultColWidth	**120**
Options:	
goEditing	**True**
ScrollBars	**None**

Also add a **button** component with the caption '**plot graph**'. Adjust the panel to a suitable size:



Double-click the dotted grid outside the image box to produce an 'OnCreate' event handler and add lines to display captions in the string grid:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  stringgrid1.cells[0,0]:='Conservative';
  stringgrid1.cells[0,1]:='Labour';
  stringgrid1.cells[0,2]:='Liberal Democrat';
  stringgrid1.cells[0,3]:='Plaid Cymru';
end;
```

Compile and run the program. Check that captions are displayed for the parties, then return to the Delphi editing screen.

We now need to set up an array to store the election results when they are entered in the string grid. Make an entry in the '*public declarations*' section:

```
public
   { Public declarations }
   votes:array[0..3]of real;
end;
```

The data type '**real**' has been used because large numbers of votes might have to be entered.

Next we need a procedure to transfer data from the string grid into the **votes** 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 create 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
    votes[y]:=0
  else
    votes[y]:=strtofloat(stringgrid1.cells[1,y]);
end;
```

Compile and run the program to check that the error trapping works correctly for the string grid. Only numbers should be accepted. Return to the Delphi editing screen.

Double-click the '**plot graph**' button on the panel to create an event handler. Add the lines:

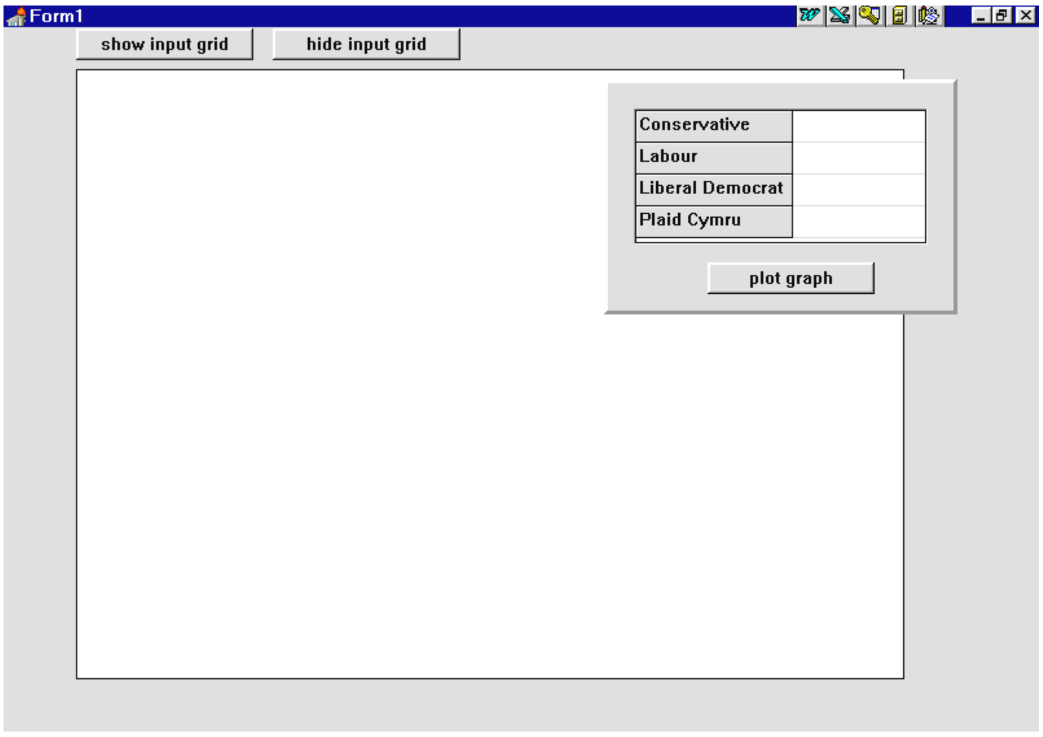
```
procedure TForm1.Button1Click(Sender:TObject);
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
end;
```

Add two buttons at the top of the screen labelled '**show input grid**' and '**hide input grid**'. Create an event handler for the '*show input grid*' button and add the line:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  panel1.visible:=true;
end;
```

For the '*hide input grid*' button, create the event handler:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  panel1.visible:=false;
end;
```



Run the program to check that the panel can be turned on and off as required, and that a white rectangle appears on the form when the '**plot graph**' button is pressed. Return to the Delphi editing screen.

The pie graph will be drawn with different coloured sectors representing the political parties. We will need a key to the colours. Add lines to the '**plot graph**' event handler to do this:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:integer;
  caption:string;
```

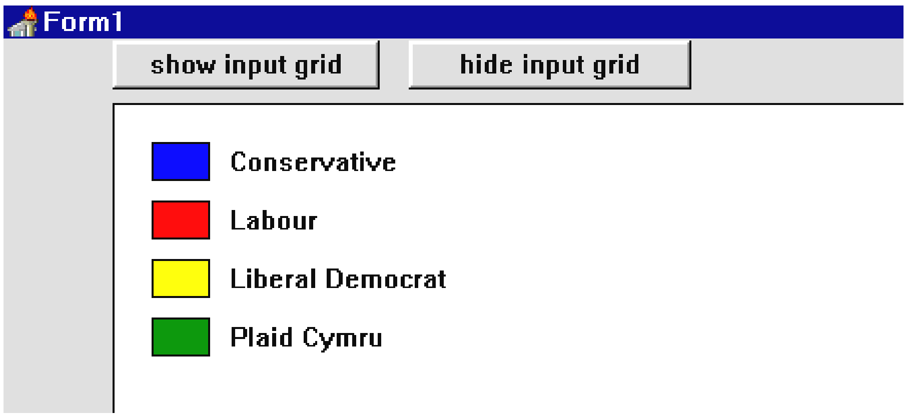
```
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  for i:=0 to 3 do
  begin
    case i of
    0:image1.canvas.brush.color:=clBlue;
    1:image1.canvas.brush.color:=clRed;
    2:image1.canvas.brush.color:=clYellow;
    3:image1.canvas.brush.color:=clLime;
    end;
    image1.canvas.rectangle(20,20+i*30,50,40+i*30);
    case i of
    0:caption:='Conservative';
    1:caption:='Labour';
    2:caption:='Liberal Democrat';
    3:caption:='Plaid Cymru';
    end;
    image1.canvas.brush.color:=clWhite;
    image1.canvas.textout(60,22+i*30,caption);
  end;
end;
```

This uses a loop to repeat four times for the different political parties. Each time around the loop:

- A CASE structure sets the fill colour to represent the party:

Conservative	blue
Labour	red
Liberal Democrat	yellow
Plaid Cymru	green

- A coloured rectangle is drawn. The distance down the screen depends on the loop counter ***i***.
- Another CASE structure selects the correct caption.
- The caption is written alongside the coloured rectangle. The distance down the screen again depends on the loop counter ***i***.

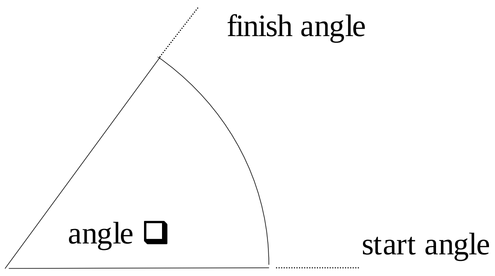


Compile and run the program. Click the '**plot graph**' button to check that the key is displayed, then return to the Delphi editing screen.

Plotting the pie graph will require some mathematical techniques. We begin by finding the total of the votes for all the four parties using a loop:

```
for i:=0 to 3 do
   total:=total+votes[i];
```

As each secor of the pie graph is drawn, we need to know the number of degrees for the angle ❑**.**



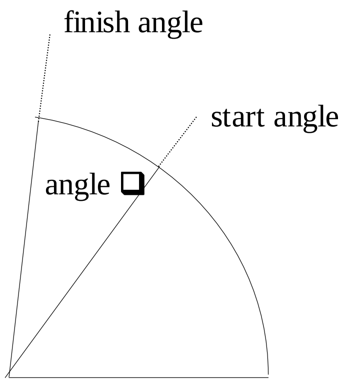
This will be the difference between the start angle and finish angle for the edges of the sector.

We can start the first sector at 0 degrees. The finish angle for the first sector will be given by the formula:

```
finish angle:= votes[1]/total * 360
```

We have just worked out what proportion of the whole 360 degrees should represent the votes for candidate 1.

The **start angle** for the **second sector** will be the *finish angle* of the *first sector*:



... and so on around the circle until all four sectors have been drawn.

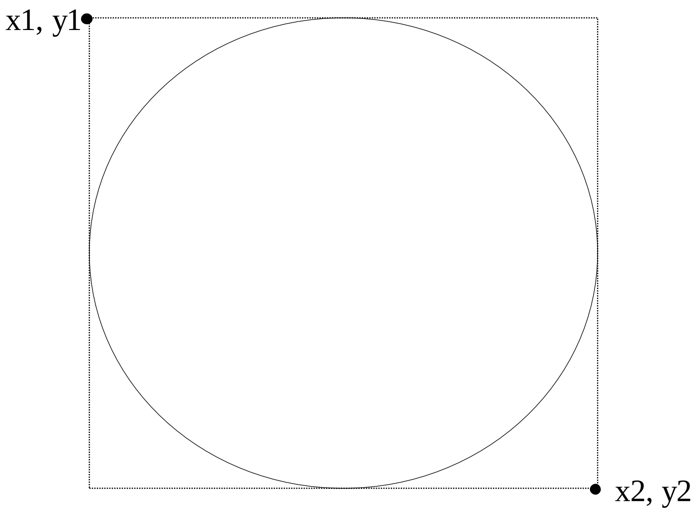
In general, the finish angle for sector ***i*** will be given by the formula:

finish angle = start angle + votes[i] / total * 360;

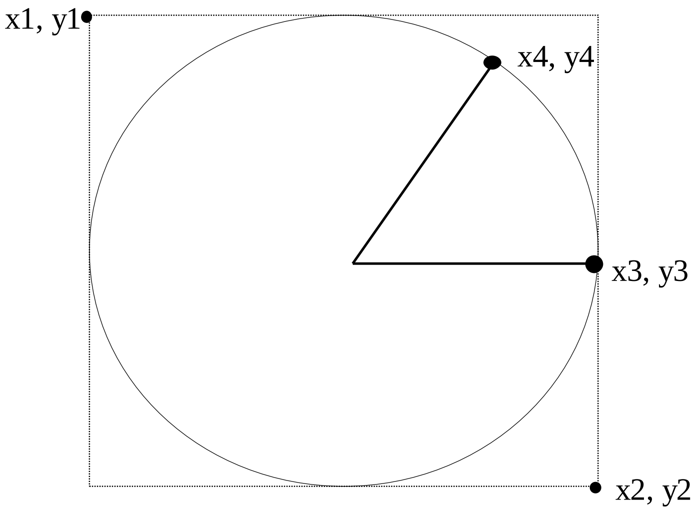
Once we know the start and finish angles of the sector, we can use the '**pie**' command to actually draw the sector on the screen. The '**pie**' command requires us to calculate eight numbers to make it work! It has the form:

pie (x1, y1, x2, y2, x3, y3, x4, y4);

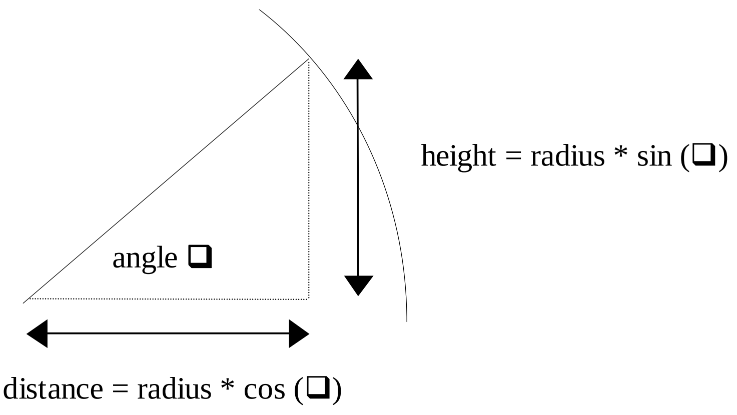
(**x1, y1**) and (**x2, y2**) are the coordinates of the opposite corners of a square containing the whole pie-graph circle:



The next pair of coordinates (x3, y3) give the position where the start angle line cuts the pie circle. (x4, y4) is the position where the finish angle line cuts the circle:



The method for finding the points where the sector lines cut the circle uses SINE and COSINE functions. For example:



These calculations can now be incorporated into the program. Add lines to the '**plot graph**' procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:integer;
  total,startangle,finishangle:real;
  caption:string;
begin
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  total:=0;
  startangle:=0;
  for i:=0 to 3 do
    total:=total+votes[i];
  for i:=0 to 3 do
  begin
    case i of
    0:image1.canvas.brush.color:=clBlue;
                .....
    3:image1.canvas.brush.color:=clLime;
    end;
    image1.canvas.rectangle(20,20+i*30,50,40+i*30);
    finishangle:=startangle+360*votes[i]/total;
    image1.canvas.pie(200,90,500,390,
        350+round(150*cos(startangle*pi/180)),
        240-round(150*sin(startangle*pi/180)),
        350+round(150*cos(finishangle*pi/180)),
        240-round(150*sin(finishangle*pi/180)));
    startangle:=finishangle;
    case i of
    0:caption:='Conservative';
          ......
```

Notice in the '**pie**' line:

image1.canvas.pie(200,90,500,390, ...

we are using a pie circle which fits inside a square with corners at the coordinates (**200,90**) and (**500,390**). This gives the pie circle a diameter of 300 units, or a **radius of 150 units**.

The centre of the pie circle is at the point (**350,240**) on the screen. To calculate the coordinates where the **start angle** and **finish angle lines** cut the circle, we multiply the **sin** and **cos** terms by the **radius**, then add these to the coordinate for the centre of the circle. For example:

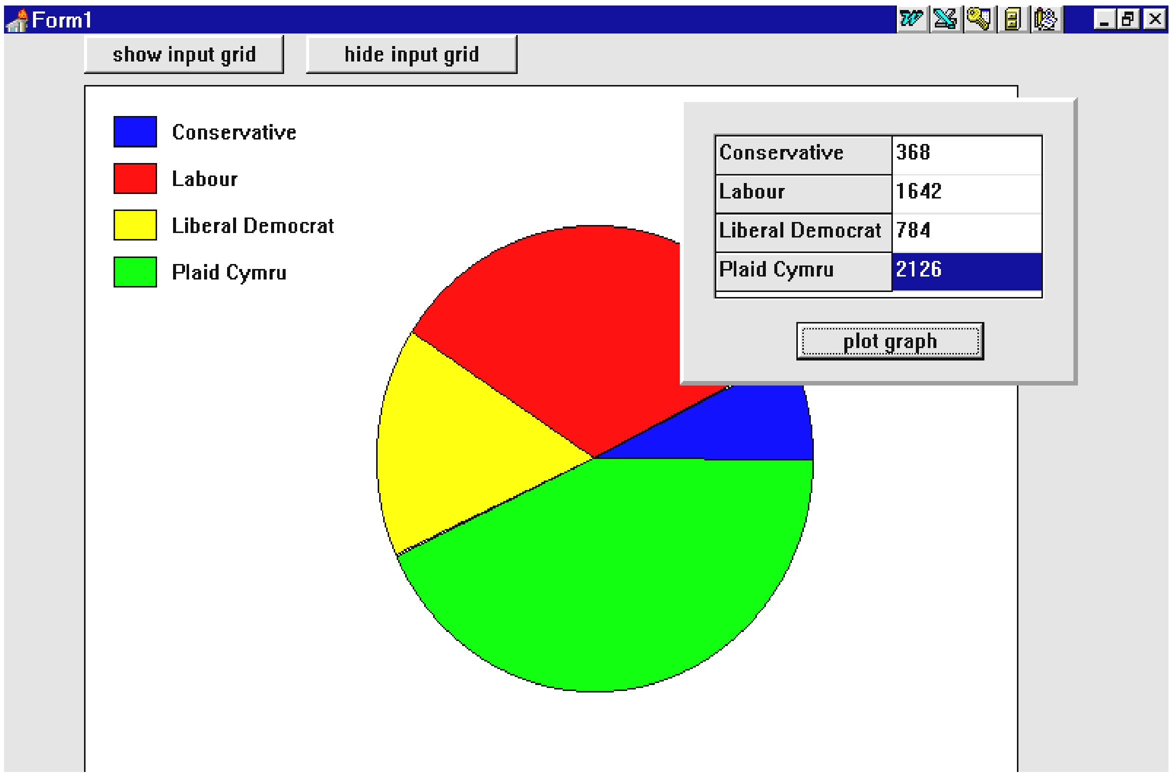
350+round(150*cos(startangle*pi/180)), ...

Add the across position for the centre of the circle

multiply by the radius

convert the start angle from degrees to radians

Unfortunately the graphics commands in Delphi and most other programming languages use **radians** instead of *degrees* for measuring angles. It is necessary to multiply the angle in degrees by **pi/180** to convert to radians.



Conservative	368
Labour	1642
Liberal Democrat	784
Plaid Cymru	2126

Compile and run the program. Enter the test data and check that the graph is drawn correctly:

SUMMARY

In this chapter you have:

- Set up an image box with a fixed size of 640 by 480 screen units. This is a convenient size for drawing graphs
- Used an input grid on a panel which can be turned on and off by buttons
- Seen how to calculate the height of columns on a histogram using a scaling factor
- Seen how to calcualate a scaling factor based on the maximum data value
- Used a loop structure to draw each column and add a caption
- Used a CASE command to select the correct caption for each column
- Seen how a line graph can cover a fixed range of values on the vertical axis, and calculated the scaling factor for the vertical axis
- Used a loop to plot a line graph from values stored in an array
- Used the 'pie' command to plot sectors of a pie graph
- Seen how the SIN and COS trigonometric functions can be used to find points on the circumference of the pie graph for the start and finish of each sector.