EIGHT

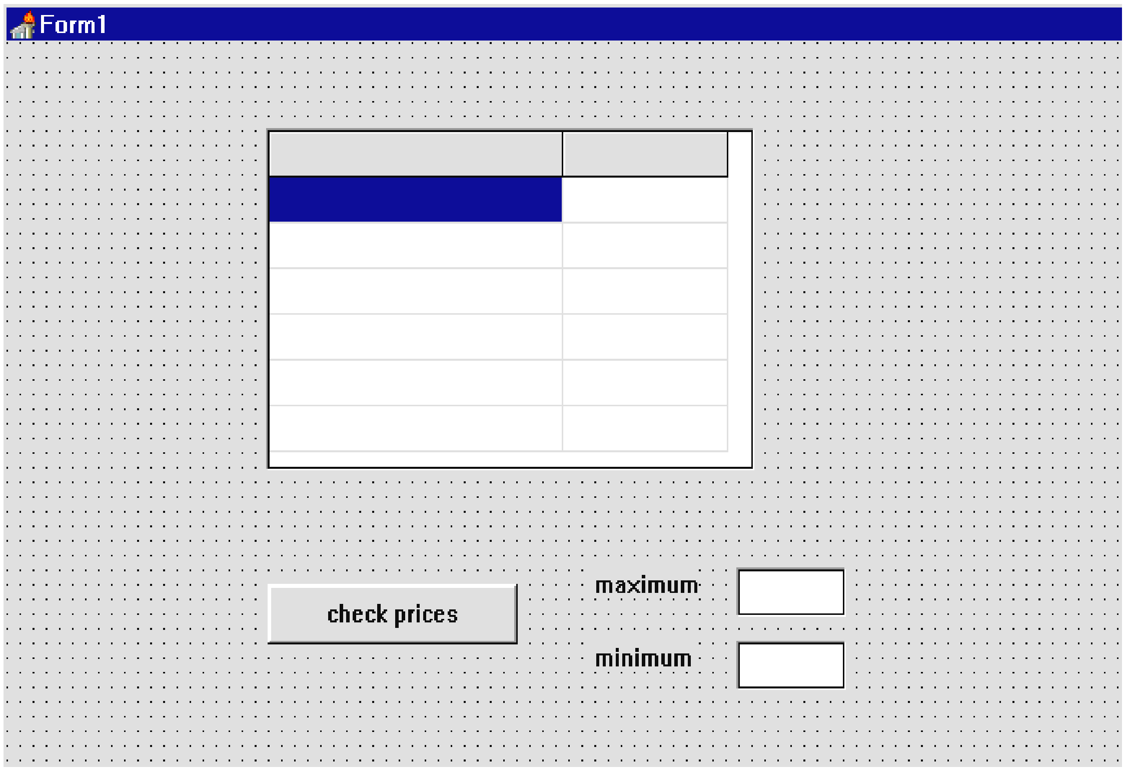
Maximums and Minimums

In computer programs it is often necessary to find the largest or smallest of a set of numbers stored in an array. The next program demonstrates how this is done:

We are going to write a program which will record the prices of a number of different makes of printer, then search for the most and least expensive. Begin by setting up a new subdirectory PRINTER and save a Delphi project into it. Use the Object Inspector to maximize the form, then drag the grid to nearly fill the screen.

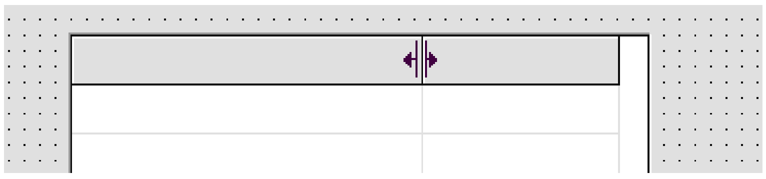
Place a string grid component on the form and set its properties:

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



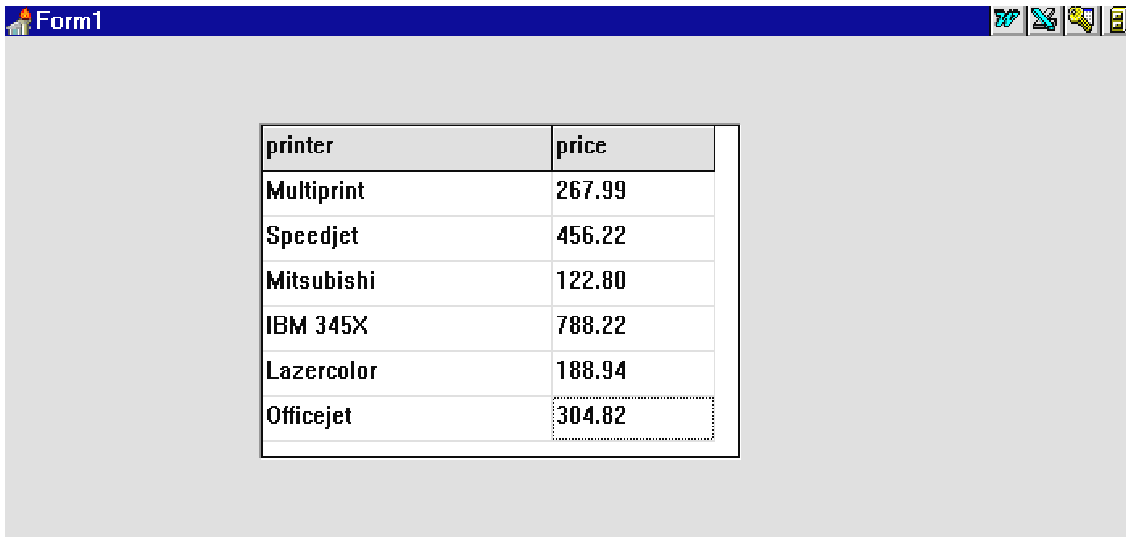
Add a *button* with the caption '**check prices**' below the string grid, and two *edit* boxes and the labels '**maximum**' and '**minimum**', as shown above.

Take the mouse pointer to the top grey row of the string grid and move to the right edge of the first column. A column sizing symbol will appear:



Hold down the mouse button and drag the first column to be about twice as wide as the second column.

The columns of the string grid are going to hold makes of printer and their prices, as in the example below:



Form1

printer	price
Multiprint	267.99
Speedjet	456.22
Mitsubishi	122.80
IBM 345X	788.22
Lazercolor	188.94
Officejet	304.82

The column headings can be set up by a 'FormCreate' event handler. Double-click the dotted form grid to produce the procedure, then add the lines:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  stringgrid1.cells[0,0]:='printer';
  stringgrid1.cells[1,0]:='price';
end;
```

Compile and run the program. Check that you are able to type information into the string grid, then return to the Delphi editing screen.

When figures are entered in the '**price**' column, these need to be stored in an array in a similar way to the exam marks program in the previous chapter. Begin by setting up the array in the 'public declarations' section near the top of the program:

```
public
   { Public declarations }
     price:array[1..6] of real;
 end;
```

Click on the *string grid* component and go to the Object Inspector. From the **Events** list double-click alongside '**OnKeyUp**'. Add lines to the procedure to transfer values from column 1 into the **price** array:

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

This procedure begins by checking whether an entry has just been made in column 1:

if stringgrid1.col=1 then . . .

If so, the variable y is used to record the row number where the entry has taken place:

y:=stringgrid1.row;

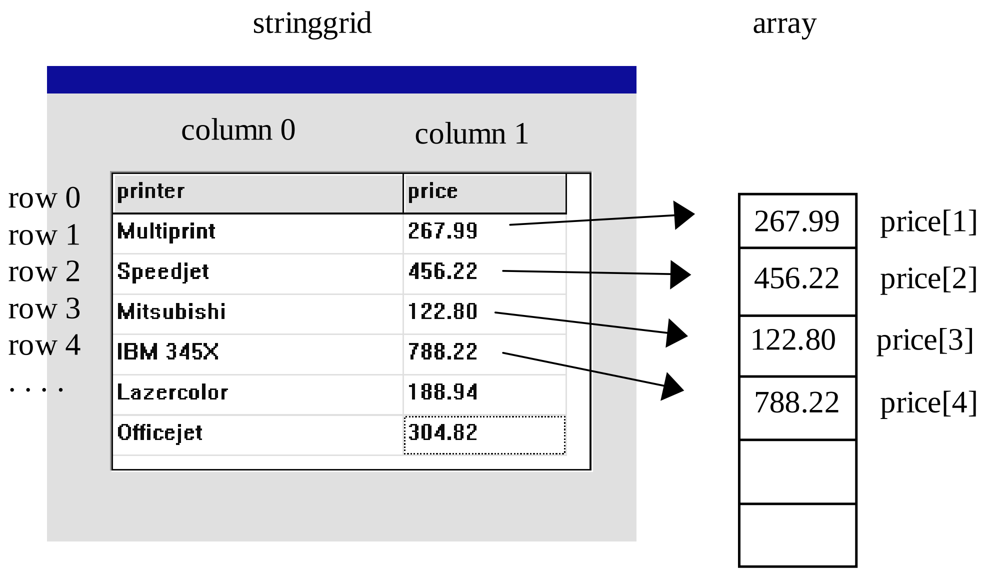
If that cell of the string grid is blank, the corresponding array element is set to zero:

```
if stringgrid1.cells[1,y]='' then
    price[y]:=0
```

Otherwise, the text in the cell is converted to a decimal number and stored in the corresponding array box:

```
else
    price[y]:=strtofloat(stringgrid1.cells[1,y]);
```

This is illustrated in the diagram below.



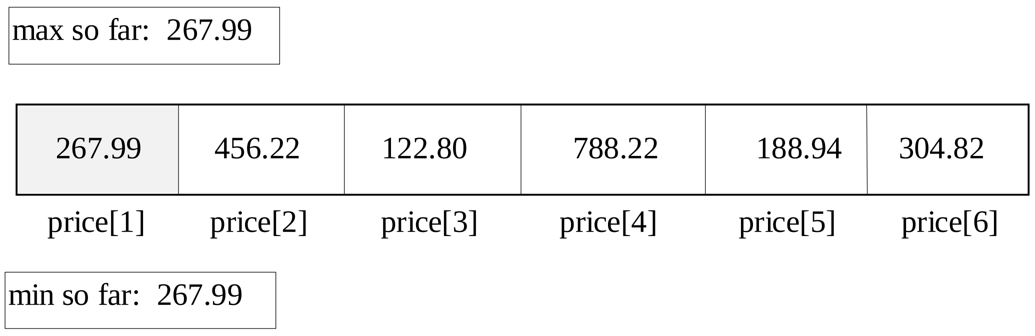
Compile, and run the program. Enter correct and incorrect data in the **price** column to check the error trapping, then return to the Delphi editing screen.

The next stage is to write a procedure to check the prices, select the maximum and minimum, and display these in the edit boxes below the string grid. Set up an event handler procedure for the '**check prices**' button and add the lines:

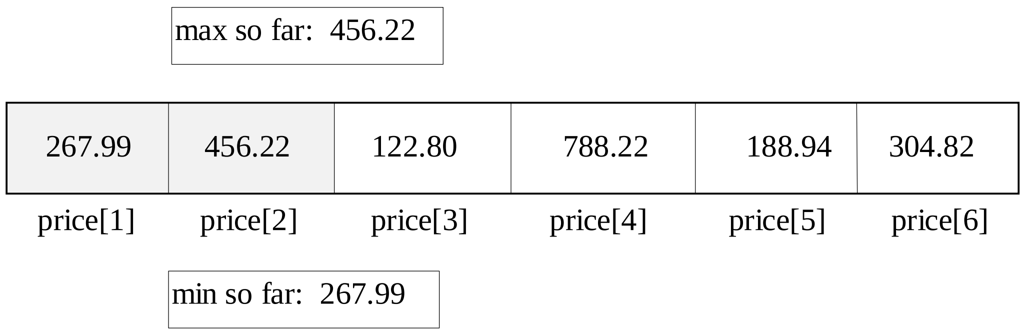
```
procedure TForm1.Button1Click(Sender: TObject);
var
  maxprice,minprice:real;
  i:integer;
begin
  maxprice:=price[1];
  minprice:=price[1];
  for i:=2 to 6 do
  begin
    if price[i]>maxprice then
      maxprice:=price[i];
    if price[i]<minprice then
      minprice:=price[i];
  end;
  edit1.text:=floattostrf(maxprice,ffFixed,8,2);
  edit2.text:=floattostrf(minprice,ffFixed,8,2);
end;
```

The way this works is illustrated by the diagrams on the next page.

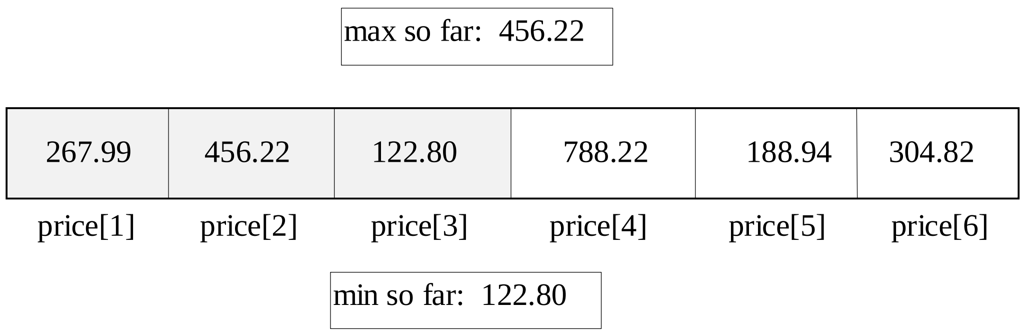
We begin by assuming that box 1 of the **price** array contains the both the **maximum** and the **minimum** price so far...



The procedure then uses a loop to move along each of the remaining array boxes in turn, seeing whether a new **maximum** or **minimum** occurs. Each time a new winner is found, the results are updated:



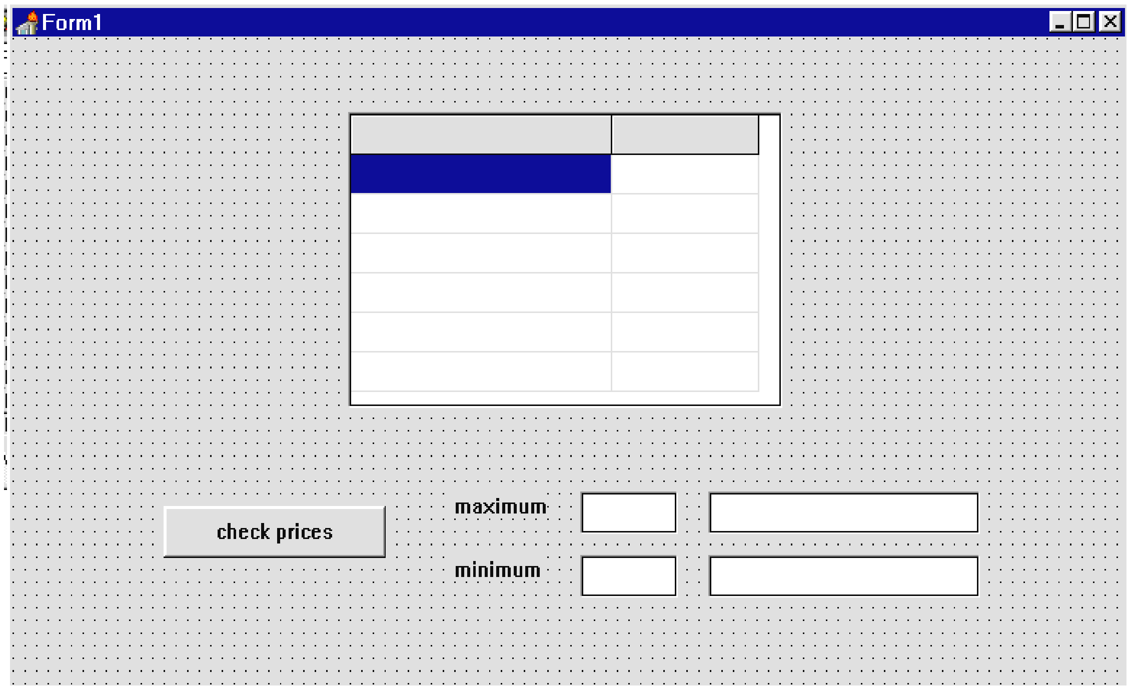
The **maximum** is updated at array box 2.



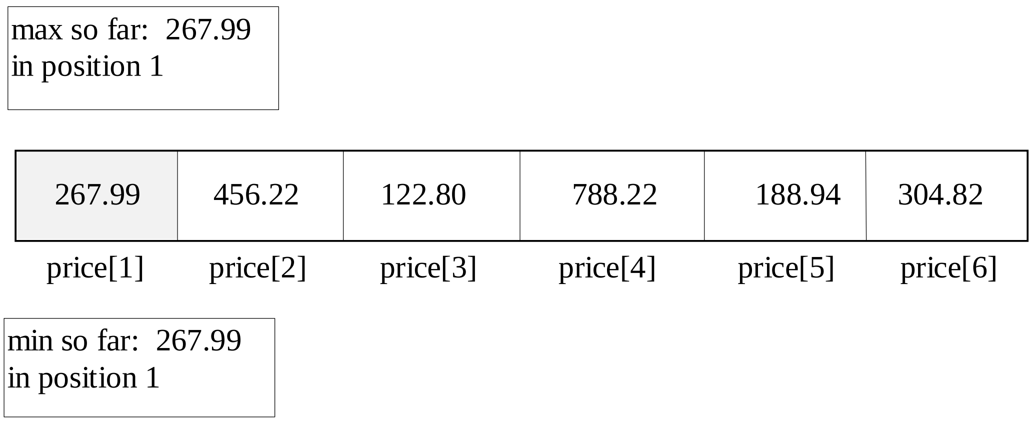
The **minimum** is updated at box 3, and so on...
By the end of the search we know the **maximum** and **minimum** prices for the whole array.

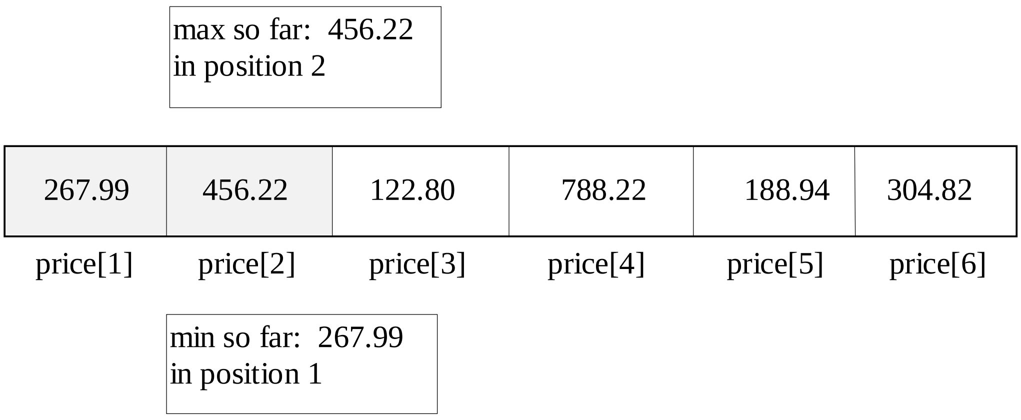
Compile and run the program. Enter the test data and check that the maximum and minimum prices are shown correctly when the button is pressed. Try varying the prices and make sure the program always gives the correct result. Return to the Delphi editing screen.

We are now going to tackle a slightly more difficult problem. As well as showing the prices of the most expensive and cheapest printer, we also want to display the make. Add two extra edit boxes to the form:

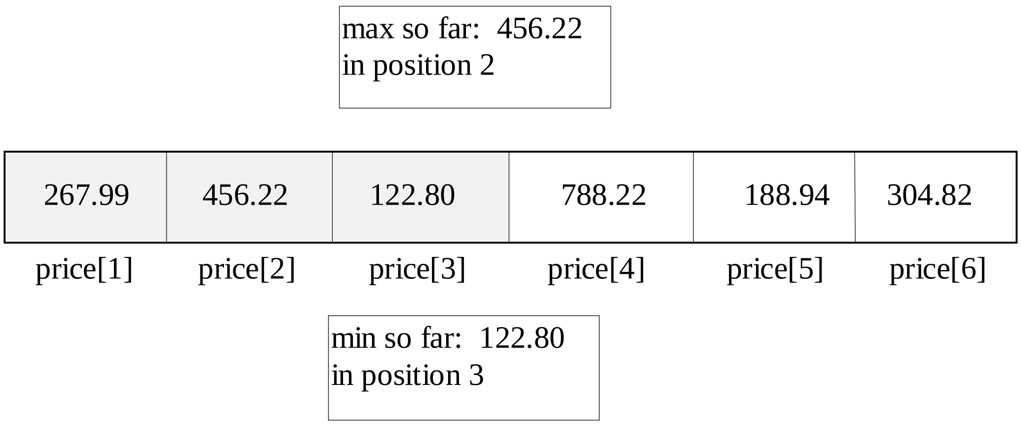


The event handler procedure for the button needs to be modified so that as well as the maximum and minimum prices, we keep track of the *positions in the array where these occur*. Refering back to the diagrams, we update the **maximum position** and **minimum position** each time a new winner is found during the search:



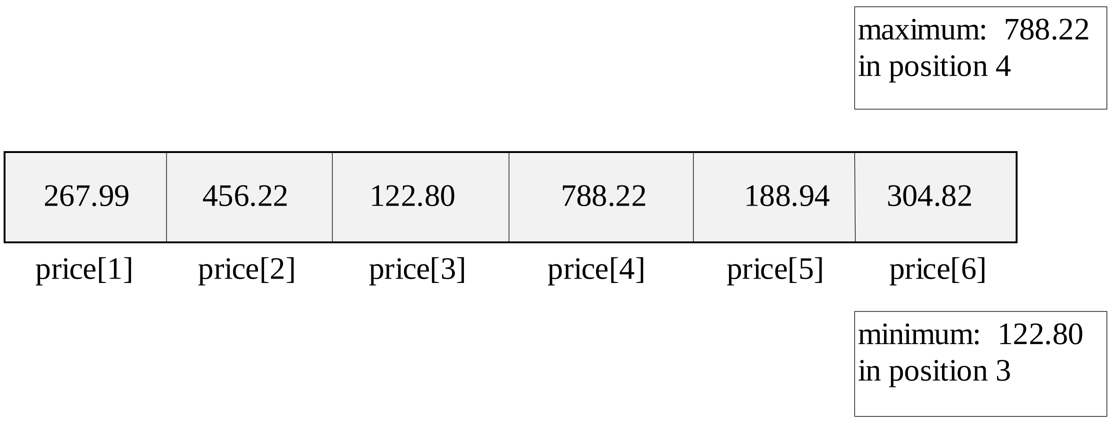


The **maximum position** is updated at array box 2 as well as the maximum price itself.

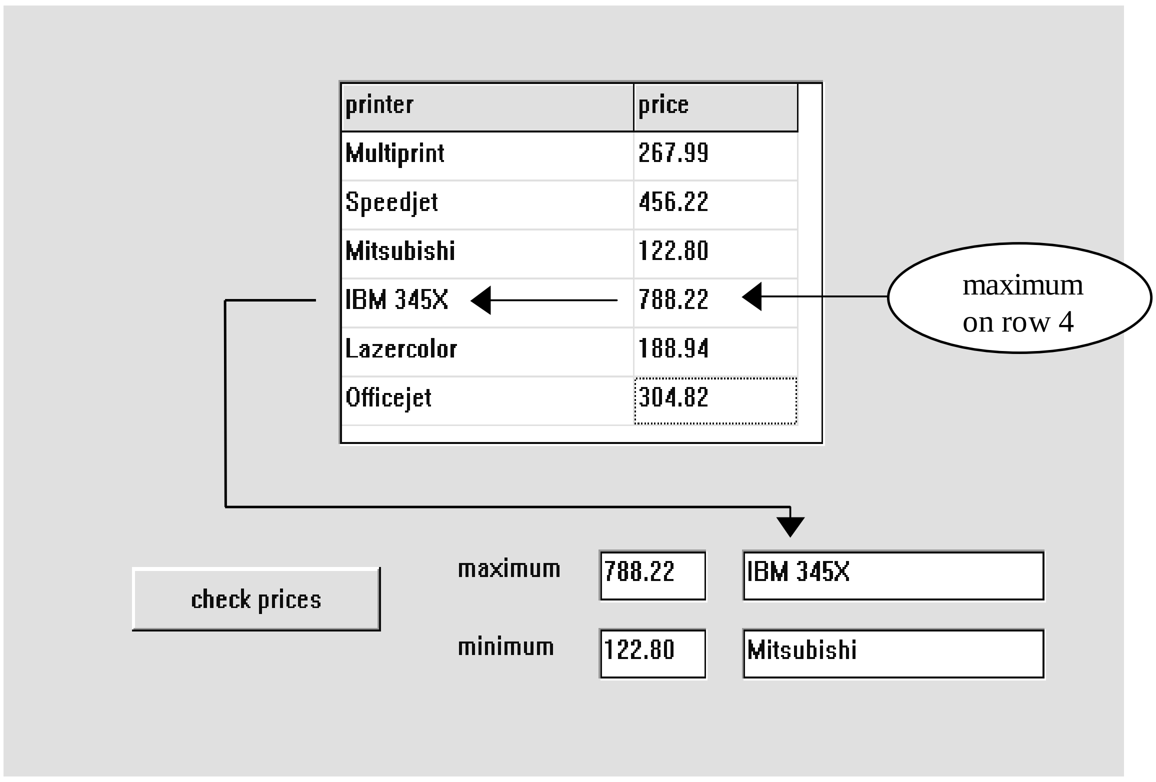


The **minimum position** is updated at box 3, and so on...

By the end of the search we know the positions of the **maximum** and **minimum** prices in the array, as well as the actual values.



We can then use this information to print the corresponding entries from column 0 of the string grid on the screen - this will show the actual names of the printers.



printer	price
Multiprint	267.99
Speedjet	456.22
Mitsubishi	122.80
IBM 345X	788.22
Lazercolor	188.94
Officejet	304.82

maximum	788.22	IBM 345X
minimum	122.80	Mitsubishi

Make changes to the button click event handler procedure as shown below:

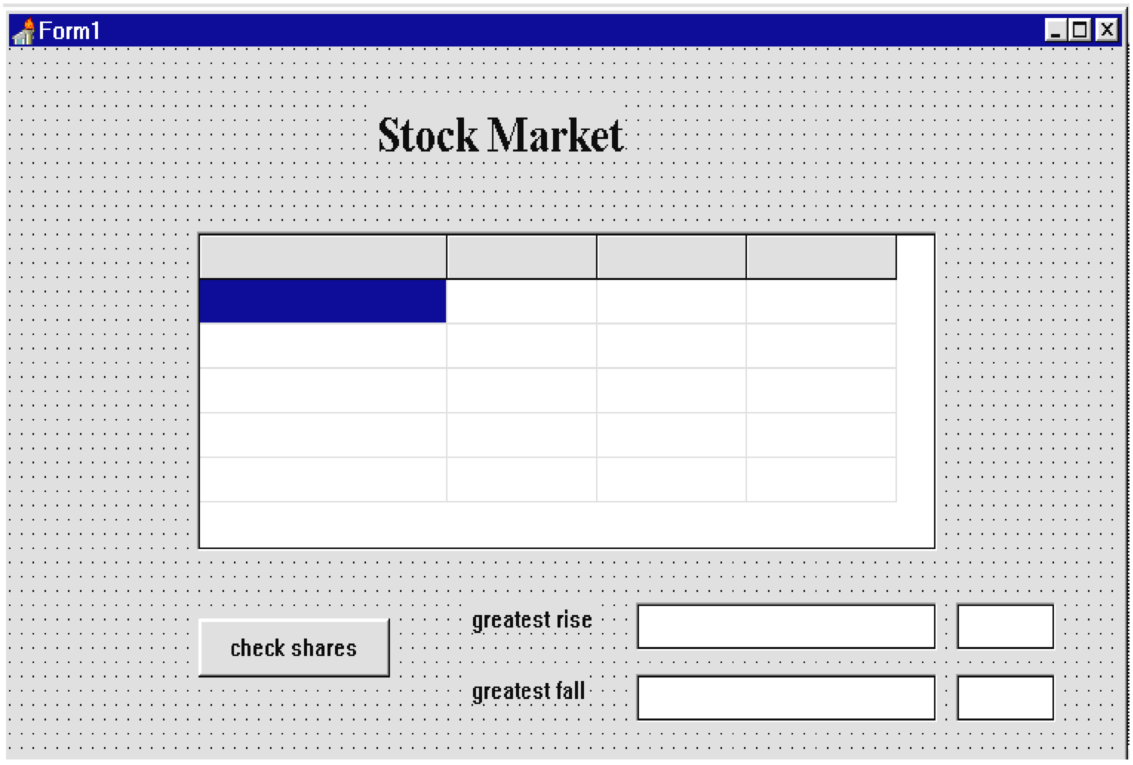
```
procedure TForm1.Button1Click(Sender: TObject);
var
  maxprice,minprice:real;
  i,maxpos,minpos:integer;
begin
  maxprice:=price[1];
  minprice:=price[1];
  maxpos:=1;
  minpos:=1;
  for i:=2 to 6 do
  begin
    if price[i]>maxprice then
    begin
      maxprice:=price[i];
      maxpos:=i;
    end;
    if price[i]<minprice then
    begin
      minprice:=price[i];
      minpos:=i;
    end;
  end;
  edit1.text:=floattostrf(maxprice,ffFixed,8,2);
  edit2.text:=floattostrf(minprice,ffFixed,8,2);
  edit3.text:=stringgrid1.cells[0,maxpos];
  edit4.text:=stringgrid1.cells[0,minpos];
end;
```

Compile and run the completed program. Enter the test data and check that the names of the most expensive and cheapest printers are found correctly.

Application

Stock market prices

The next program is to show changes in the price of company shares on the stock market, and to identify the companies with the largest rise or fall in share value.



Set up a new sub-directory SHARES 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 component on the form and set its properties:

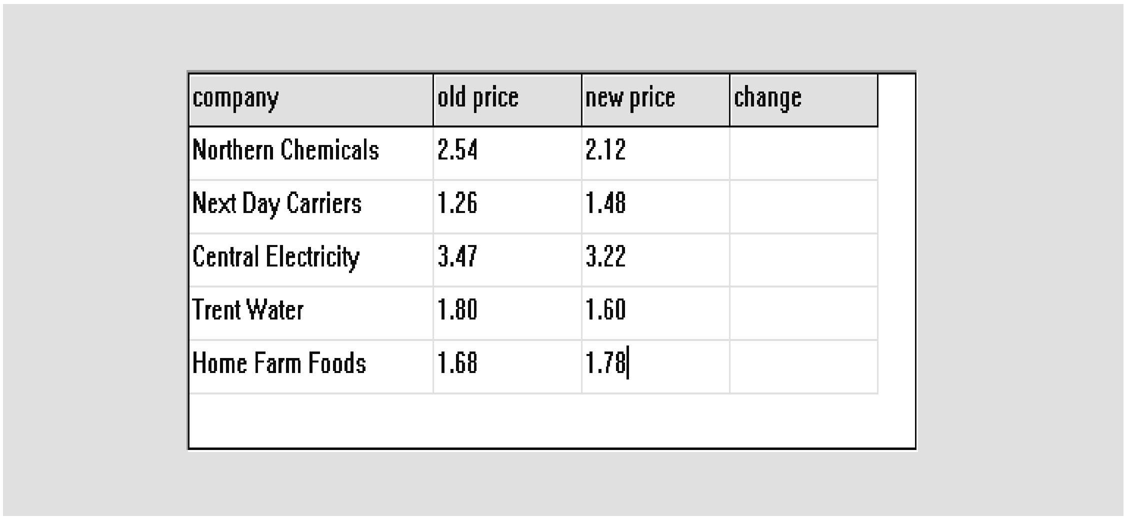
FixedCols	**0**
ColCount	**4**
RowCount	**6**
DefaultColWidth	**100**
Options:	
goEditing	**True**
ScrollBars	**None**

Add a *label* with the caption '**Stock Market**' above the string grid, and a *button* below with the caption '**check shares**'. Also add two pairs of *edit* boxes and the labels '**greatest rise**' and '**greatest fall**', as shown above.

Take the mouse pointer to the top grey row of the string grid and move to the right edge of the first column. A column sizing symbol will appear:

Hold down the mouse button and drag the first column to be about twice as wide as the others.

The columns of the string grid are going to hold company names, the old price of their shares, the new share price, and the change in share value, as in the example below:



company	old price	new price	change
Northern Chemicals	2.54	2.12	
Next Day Carriers	1.26	1.48	
Central Electricity	3.47	3.22	
Trent Water	1.80	1.60	
Home Farm Foods	1.68	1.78	

The column headings can be set up by a 'FormCreate' event handler. Double-click the dotted form grid to produce the procedure.

Add the lines:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  stringgrid1.cells[0,0]:='company';
  stringgrid1.cells[1,0]:='old price';
  stringgrid1.cells[2,0]:='new price';
  stringgrid1.cells[3,0]:='change';
end;
```

Compile and run the program. Check that you are able to type information into the string grid, then return to the Delphi editing screen.

We will need three arrays of real (decimal) numbers to store the **old price**, **new price**, and **change in price** for each of the shares. Set up the arrays in the *public declarations* section:

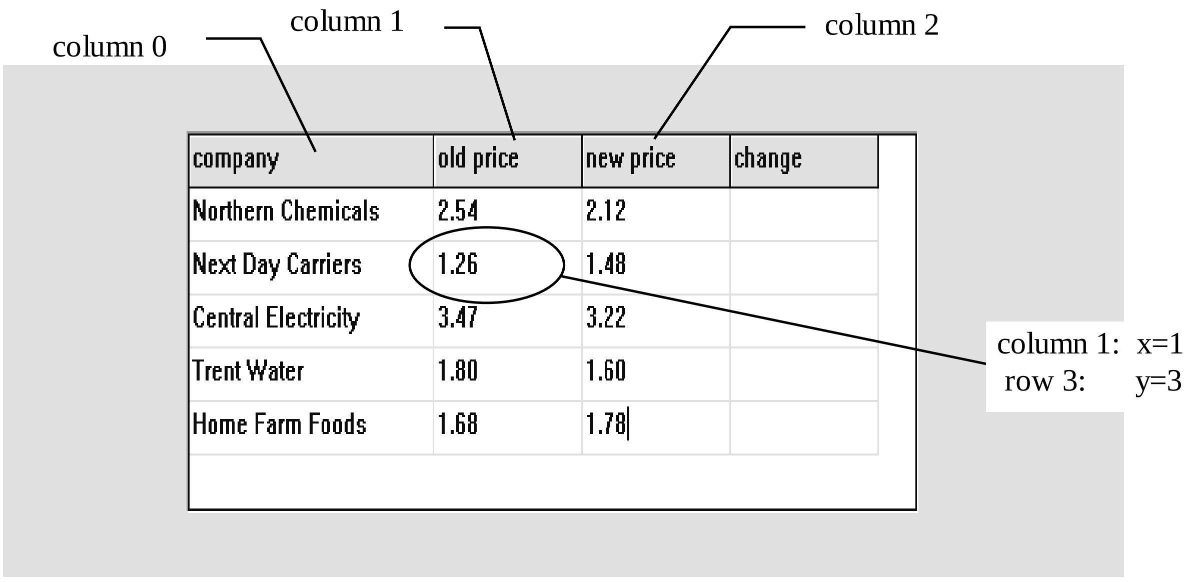
```
 public
    { Public declarations }
    old,new,change:array[1..5] of real;
  end;
```

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
  x,y:integer;
begin
  x:=stringgrid1.col;
  y:=stringgrid1.row;
  if x=1 then
  begin
    if stringgrid1.cells[1,y]='' then
       old[y]:=0
    else
       old[y]:=strtofloat(stringgrid1.cells[1,y]);
  end;
  if x=2 then
  begin
    if stringgrid1.cells[2,y]='' then
       new[y]:=0
    else
       new[y]:=strtofloat(stringgrid1.cells[2,y]);
  end;
end;
```

This procedure begins by setting x and y to be the column and row numbers where an entry has been made in the string grid:

```
x:=stringgrid1.col;
y:=stringgrid1.row;
```



company	old price	new price	change
Northern Chemicals	2.54	2.12	
Next Day Carriers	1.26	1.48	
Central Electricity	3.47	3.22	
Trent Water	1.80	1.60	
Home Farm Foods	1.68	1.78	

If the entry is in column 1 then the text is converted to a decimal number and put into the '**old price**' array:

```
if x=1 then
begin
  if stringgrid1.cells[1,y]='' then
     old[y]:=0
  else
     old[y]:=strtofloat(stringgrid1.cells[1,y]);
end;
```

If the entry is in column 2 then the text is converted to a decimal number and put into the '**new price**' array:

```
if x=2 then
begin
  if stringgrid1.cells[2,y]='' then
     new[y]:=0
  else
     new[y]:=strtofloat(stringgrid1.cells[2,y]);
end;
```

Compile and run the program. Enter correct and incorrect data in the **old price and new price** columns to check the error trapping, then return to the Delphi editing screen.

The next stage is to write a procedure to calculate the changes in price, select the maximum and minimum change, and display the results in the edit boxes below the string grid. Set up an event handler procedure for the '**check shares**' button and add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  y,maxpos,minpos:integer;
  max,min:real;
begin
  for y:=1 to 5 do
  begin
    change[y]:=new[y]-old[y];
    stringgrid1.cells[3,y]:=
                   floattostrf(change[y],ffFixed,8,2);
  end;
  max:=change[1];
  min:=change[1];
  maxpos:=1;
  minpos:=1;
  for y:=2 to 5 do
  begin
    if change[y]>max then
    begin
      max:=change[y];
      maxpos:=y;
    end;
    if change[y]<min then
    begin
      min:=change[y];
      minpos:=y;
    end;
  end;
  edit1.text:=stringgrid1.cells[0,maxpos];
  edit2.text:=floattostrf(change[maxpos],ffFixed,8,2);
  edit3.text:=stringgrid1.cells[0,minpos];
  edit4.text:=floattostrf(change[minpos],ffFixed,8,2);
end;
```

This procedure begins with a loop which takes each of the shares in turn and calculates the change in price - this is stored in the **change** array**:**

```
for y:=1 to 5 do
begin
  change[y]:=new[y]-old[y];
  stringgrid1.cells[3,y]:=floattostrf(change[y],ffFixed,8,2);
end;
```

change in price = new price - old price

The loop then displays the change in share price on the appropriate line of the string grid.

The next part of the procedure is very similar to the printer program we completed earlier in this chapter. We record that the first array box contains the maximum and minimum **change** in share values so far:

```
max:=change[1];
min:=change[1];
maxpos:=1;
minpos:=1;
```

A loop then checks the other entries in the **price change** array. If a new maximum or minimum is found, the variables are updated:

```
for y:=2 to 5 do
begin

  {a new maximum has been found}
  if change[y]>max then
  begin
    max:=change[y];
    maxpos:=y;
  end;

  {a new minimum has been found}
  if change[y]<min then
  begin
    min:=change[y];
    minpos:=y;
  end;
end;
```

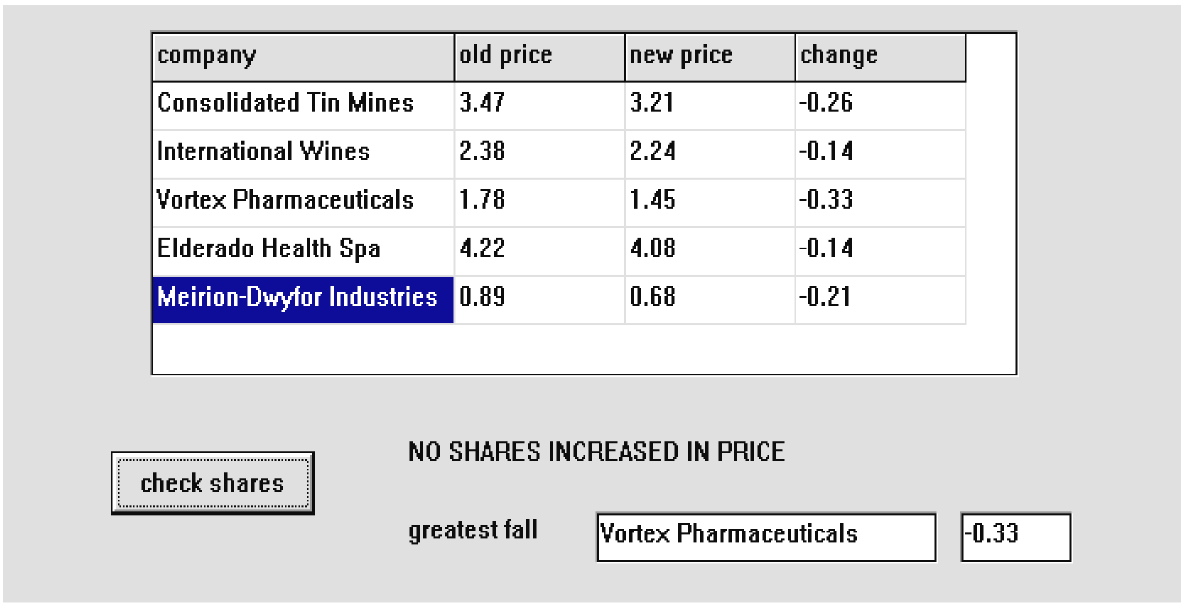
The last step is to display the company name and change in price for the maximum and minimum price variations:

```
edit1.text:=stringgrid1.cells[0,maxpos];
edit2.text:=floattostrf(change[maxpos],ffFixed,8,2);
edit3.text:=stringgrid1.cells[0,minpos];
edit4.text:=floattostrf(change[minpos],ffFixed,8,2);
```

Notice that all prices are being displayed with two decimal places to represent pounds and pence.

Compile and run the program. Check this with suitable test data.

There is one final problem we should deal with - there may be no shares which have increased in price, or no shares which have decreased in price. These situations require special screen messages:



company	old price	new price	change
Consolidated Tin Mines	3.47	3.21	-0.26
International Wines	2.38	2.24	-0.14
Vortex Pharmaceuticals	1.78	1.45	-0.33
Elderado Health Spa	4.22	4.08	-0.14
Meirion-Dwyfor Industries	0.89	0.68	-0.21

NO SHARES INCREASED IN PRICE

greatest fall Vortex Pharmaceuticals -0.33

Add lines near the end of the '**check shares**' button click procedure to do this:

```
         ......
if change[y]<min then
  begin
    min:=change[y];
    minpos:=y;
  end;
end;
if change[maxpos]>0 then
begin
  label2.caption:='greatest rise';
  edit1.visible:=true;
  edit2.visible:=true;
  edit1.text:=stringgrid1.cells[0,maxpos];
  edit2.text:=floattostrf
                  (change[maxpos],ffFixed,8,2);
end
else
begin
  label2.caption:='NO SHARES INCREASED IN PRICE';
  edit1.visible:=false;
  edit2.visible:=false;
end;
```

ADD THIS

```
    if change[minpos]<0 then
    begin
      label3.caption:='greatest fall';
      edit3.visible:=true;
      edit4.visible:=true;
      edit3.text:=stringgrid1.cells[0,minpos];
      edit4.text:=floattostrf
                          (change[minpos],ffFixed,8,2);
    end
    else
    begin
      label3.caption:='NO SHARES FELL IN PRICE';
      edit3.visible:=false;
      edit4.visible:=false;
    end;
  end;
```

The sections we have added contain IF..THEN..ELSE.. conditional blocks. The first of these checks whether any change in share price is greater than zero:

```
if change[maxpos]>0 then  { IF there has been an increase in share price}
begin
   { display details of the company shares with maximum price increase}
end
else
begin
  {  give a message that no shares have increased in price}
end;
```

To display the message about no increase in share price we use the line:

label3.caption:='NO SHARES FELL IN PRICE';

to change the label text at the bottom of the screen, and the lines:

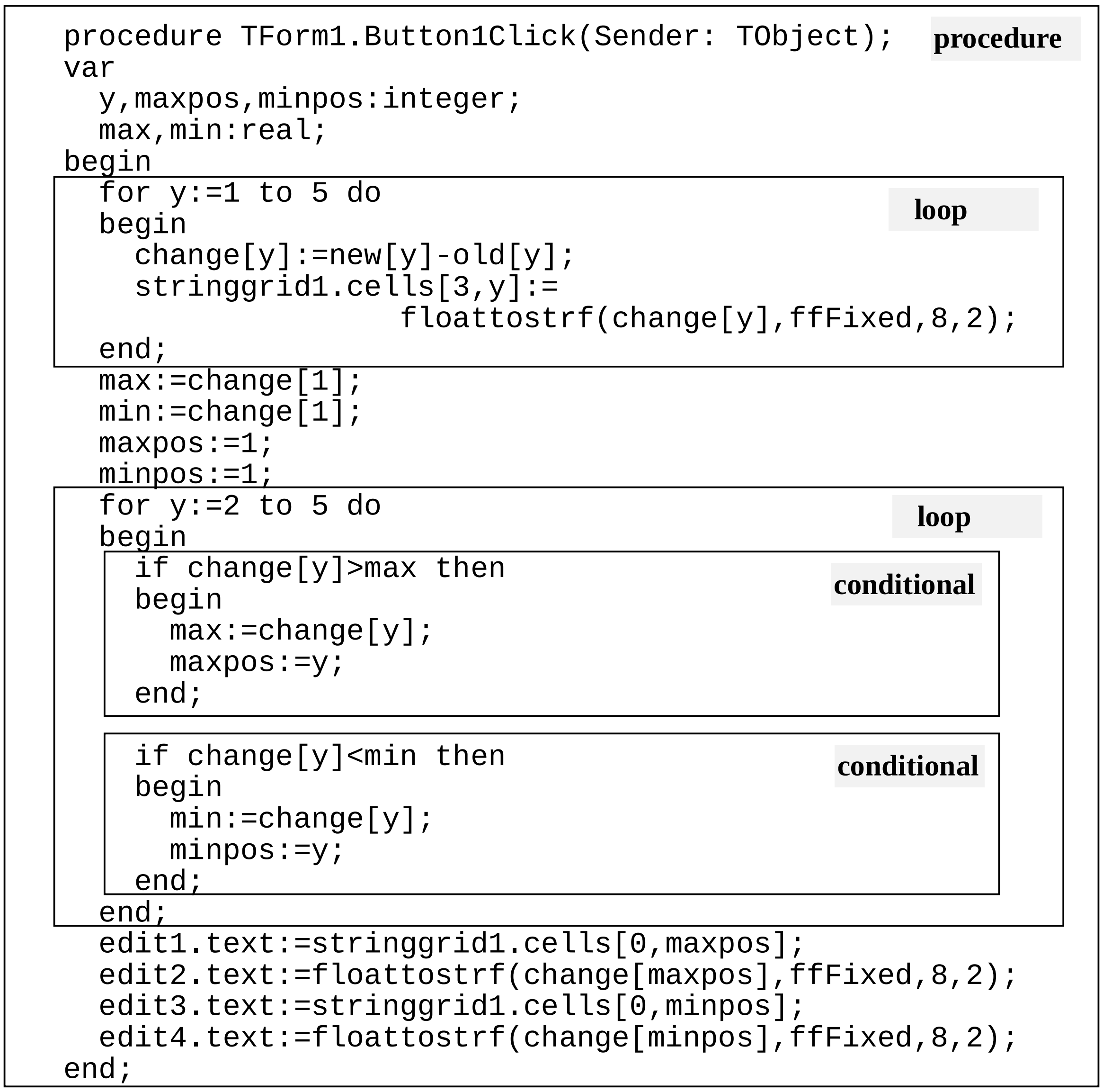
edit3.visible:=false;
edit4.visible:=false;

to turn off the edit boxes where the company name and share price would have been displayed.

Compile and test the completed program with a variety of share prices. Check the cases where all shares rise in price, and where all shares fall in price, as well as a combination of prices rising and falling.

Block analysis

A very useful technique to show the structure of a complicated piece of program is **block analysis**. Boxes are drawn around each *procedure*, *loop* and *conditional* (if..then..else..) block in the program. As an example, this is done for the Button Click event handler:



```
procedure TForm1.Button1Click(Sender: TObject);
var
  y,maxpos,minpos:integer;
  max,min:real;
begin
  for y:=1 to 5 do
  begin
    change[y]:=new[y]-old[y];
    stringgrid1.cells[3,y]:=
                   floattostrf(change[y],ffFixed,8,2);
  end;
  max:=change[1];
  min:=change[1];
  maxpos:=1;
  minpos:=1;
  for y:=2 to 5 do
  begin
    if change[y]>max then
    begin
      max:=change[y];
      maxpos:=y;
    end;

    if change[y]<min then
    begin
      min:=change[y];
      minpos:=y;
    end;
  end;
  edit1.text:=stringgrid1.cells[0,maxpos];
  edit2.text:=floattostrf(change[maxpos],ffFixed,8,2);
  edit3.text:=stringgrid1.cells[0,minpos];
  edit4.text:=floattostrf(change[minpos],ffFixed,8,2);
end;
```

Notes can then be added to explain the purpose of each of the procedures, loops and conditionals which you have identified. This type of documentation would be very useful for a programmer who needed to modify the program at a later date.

SUMMARY

In this chapter you have:

- set up string grids with multiple columns, and altered column widths
- transferred decimal numbers from a string grid to a **real** array
- used an algorithm to search an array for the maximum and minimum value
- modified the algorithm to record the positions in an array of the maximum and minimum values
- seen how **block analysis** can show the structure of a program more clearly, by identifying the procedures, loops and conditional blocks which are present.