

TEN

Sorting

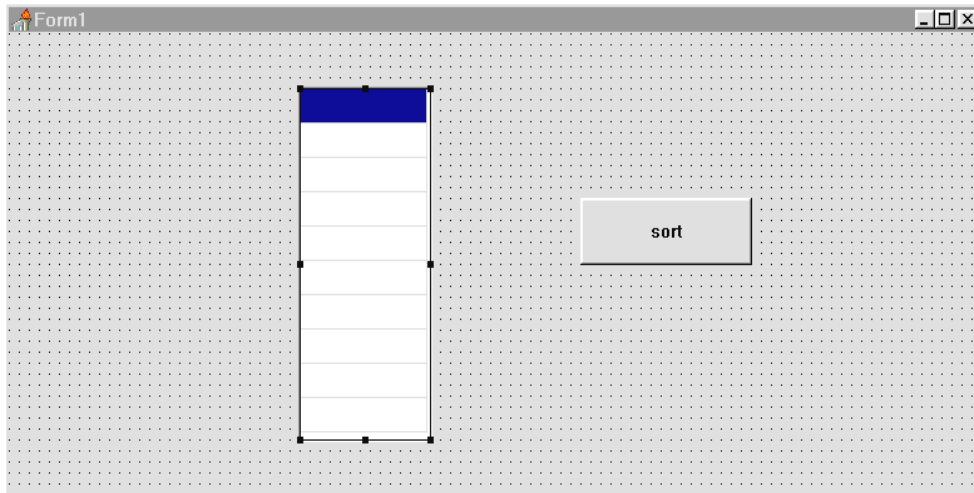
In this chapter we will look at various ways in which sorting might be used in programs. To begin with, let's see how a series of words could be arranged into alphabetical order by the computer...

Alphabetical sort

Set up a new directory SORT 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 a string grid and set the properties:

FixedCols	0
ColCount	1
FixedRows	0
RowCount	10
DefaultColWidth	100
Options:	
goEditing	True
ScrollBars	None



Place a button alongside, with the caption '**sort**'. When words are typed into the string grid, these will be stored in an array '**word**' ready for sorting. Set up the array in the *public declarations* section:

```

public
  { Public declarations }
  word:array[0..9] of string;
end;

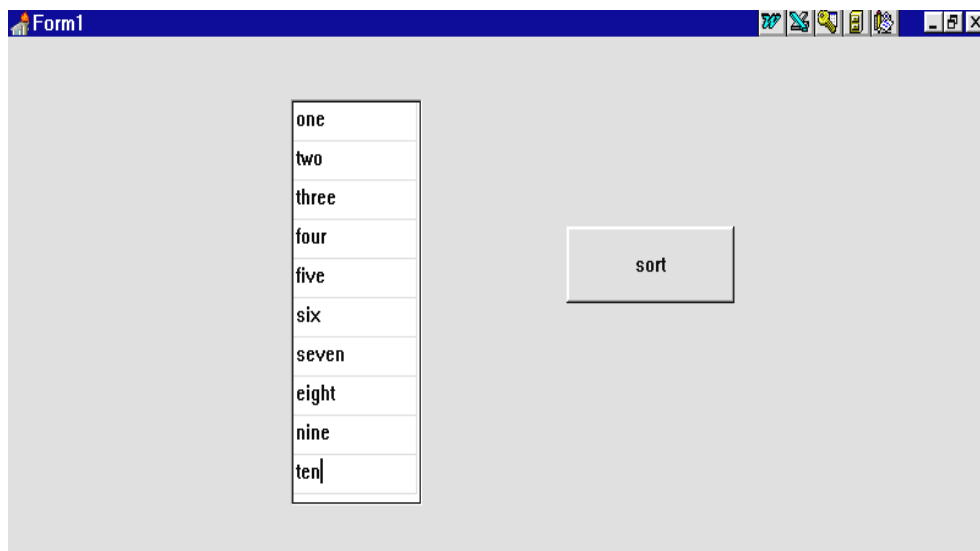
```

We now need a procedure to transfer entries from the string grid into the **word** array. Click on the string grid then press ENTER to bring up the Object Inspector. Click the **Events** tab, then double-click alongside '**OnKeyUp**'. Add lines to the event handler procedure:

```

procedure TForm1.StringGrid1KeyUp(Sender:
  TObject; var Key: Word; Shift: TShiftState);
var
  n:integer;
begin
  n:=stringgrid1.row;
  word[n]:=stringgrid1.cells[0,n];
end;

```



Compile and run the program to check that text can be entered in the string grid, then return to the Delphi editing screen.

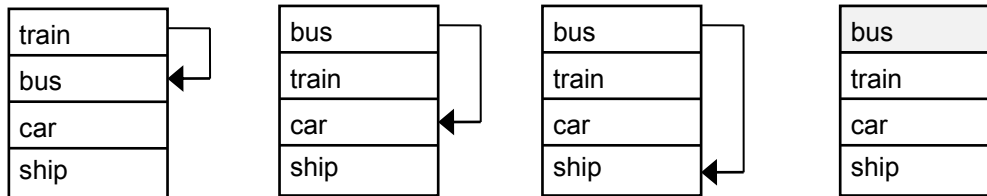
The objective of the program is to input a series of words, such as: *one, two, ... nine, ten*, then press the '**sort**' button. The computer should then redisplay the words in alphabetical order.

The technique we will use is called a '***bubble sort***'. This is illustrated by the diagrams on the next page:

Bubble sort algorithm

As an example, we will sort the words: *train*, *bus*, *car*, *ship* into alphabetical order.

First pass through the data

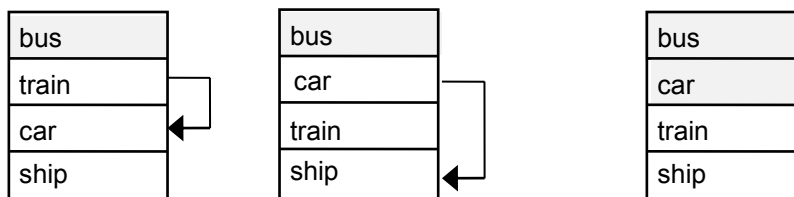


The algorithm begins by comparing the top word with the word below it. If the words are already in correct order then no action is taken, otherwise their positions are swapped. In this case, '*bus*' should come before '*train*' alphabetically, so a swap is necessary.

The process is repeated, comparing the top word with words in each of the other positions in turn. No exchange is necessary between the words '*bus*' and '*car*', or between '*bus*' and '*ship*'.

At the end of the first pass through the data we have brought the word '*bus*' up to the top of the list where it should be in the alphabetical sequence. Notice, however, that the other words are not yet in correct order.

Second pass through the data

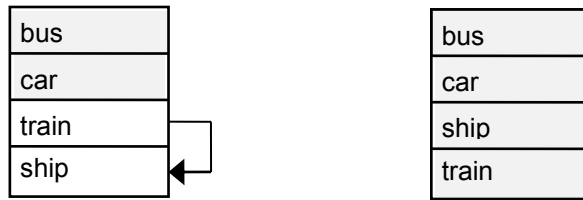


The objective of the second pass through the data is to bring the correct word into the second position in the alphabetical sequence. The word '*bus*' is already in the right place so this takes no further part in the sorting.

The second pass begins by comparing '*train*' and '*car*'; these need to be exchanged. The final comparison is between '*car*' and '*ship*', but no exchange is needed this time.

At the end of the second pass we have the words '*bus*' and '*car*' in the correct two positions at the top of the list.

Third pass through the data



One final pass through the data is needed to compare the bottom two words 'train' and 'ship'. These are exchanged to produce the final sorted sequence.

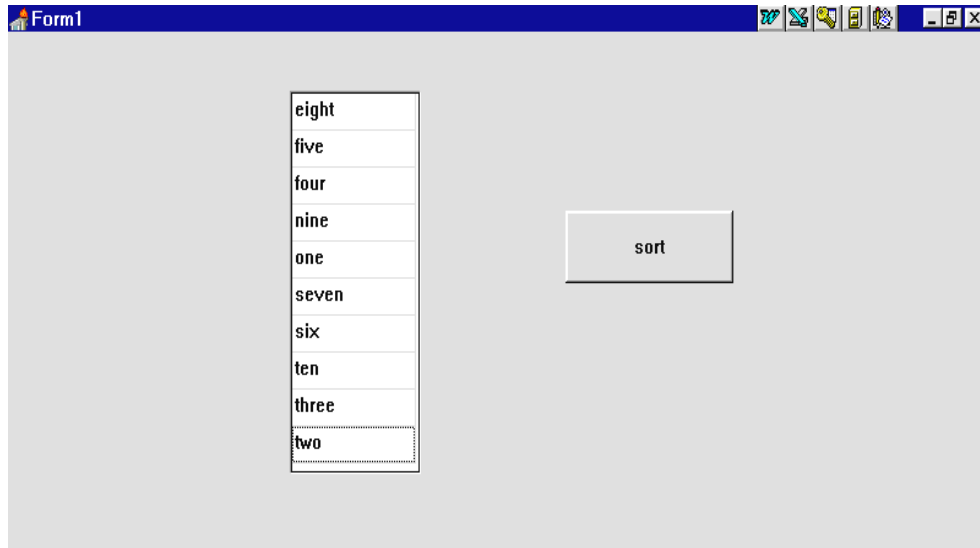
The '**bubble sort**' gets its name from the way each item of data '*bubbles*' its way up to the top of the list in turn.

We must now think about how the bubble sort algorithm is written as a section of computer program. Notice that there will be **two** loop structures involved - an outer loop to carry out the correct number of passes through the data, and an inner loop which operates each time to make the set of comparisons between words in the list.

There is always *one less* pass through the data than the number of data items to be sorted. In our example above, once three of the words are in their correct positions then the fourth word must also be in the right place, so no fourth pass through the data is needed.

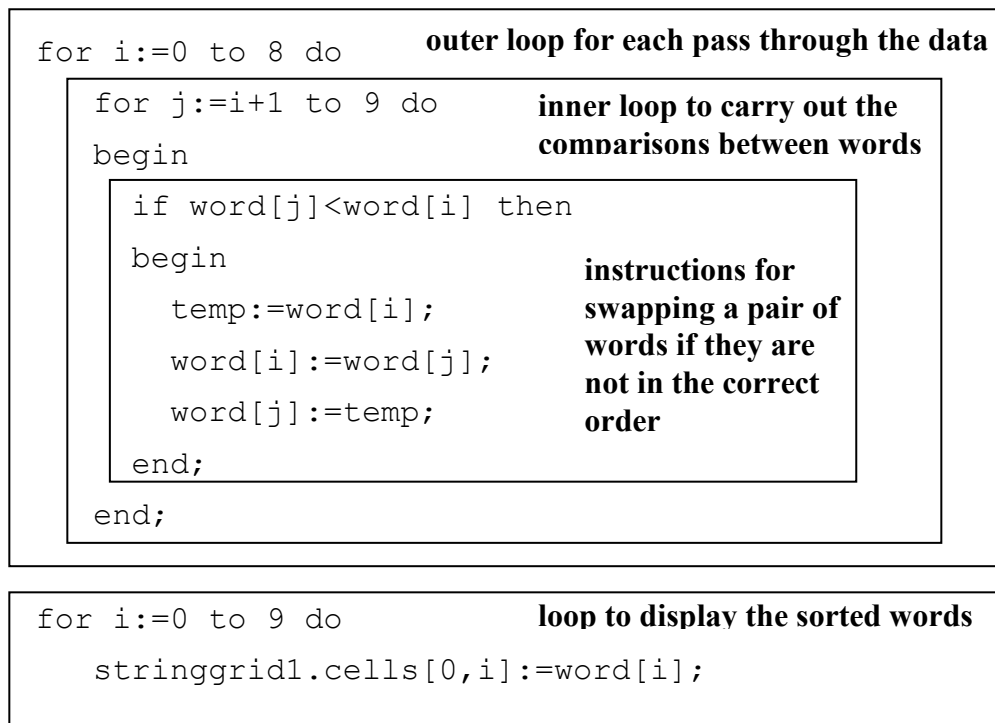
Go to the Delphi editing screen and double-click the 'sort' button to create an event handler. Add the lines of program to carry out the bubble sort:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,j:integer;
  temp:string;
begin
  for i:=0 to 8 do
    for j:=i+1 to 9 do
      begin
        if word[j]<word[i] then
          begin
            temp:=word[i];
            word[i]:=word[j];
            word[j]:=temp;
          end;
        end;
      for i:=0 to 9 do
        stringgrid1.cells[0,i]:=word[i];
      end;
```



Compile and run the program. Enter the words 'one'..'ten' then press the 'sort' button. Check that the words are sorted into correct alphabetical order. Try entering other sequences of words in the string grid and sorting them.

Before leaving this program, let's look in more detail at how the program works. The structure is shown more clearly if we carry out a block analysis:



Let's examine the section of program which carries out the swapping:

```
if word[j]<word[i] then
begin
  temp:=word[i];
  word[i]:=word[j];
  word[j]:=temp;
end;
```

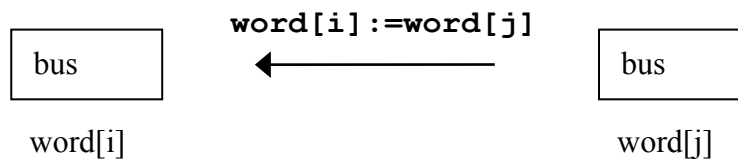
The loop counter variables *i* and *j* specify which words in the array are currently being compared, for example:



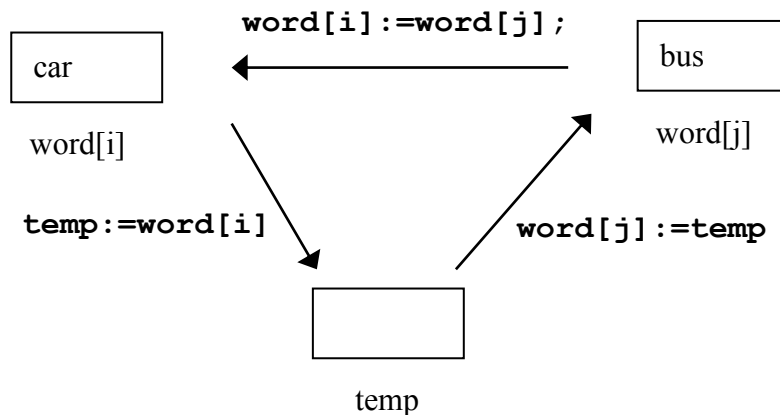
If they are not in the correct positions then they will need to be swapped, but it would not be sufficient to do this with the lines of program:

```
word[i]:=word[j];
word[j]:=word[i];
```

The first line would copy the word 'bus' from box *j* into box *i*, then the second line would copy 'bus' back from box *i* into box *j*.



Instead of exchanging the words, we would simply end up with the word 'bus' in both array boxes! To get around this problem we use a **triangular exchange** method. This involves storing a word in a temporary memory location, so that it can't be overwritten before reaching its final destination:



Using parallel arrays

Often in programs we need to keep more than one piece of data for each item entered, for example: the description *and* price of products in a warehouse. This may be done using **parallel arrays**:

	product		price
1	television	1	234.50
2	stereo	2	388.00
3	video recorder	3	294.00
4	microwave oven	4	178.99

Two separate arrays are used to hold the descriptions of the products and the prices, but array boxes with the same numbers refer to the same item - for example: **product 2** (the stereo) has **price 2** (£388.00).

If data in parallel arrays is sorted, it is very important that entries in both arrays are moved simultaneously. Supposing we wished to sort the product information into alphabetical order, the corresponding prices would also need to move:

	product		price
1	microwave oven	1	178.99
2	stereo	2	388.00
3	television	3	234.50
4	video recorder	4	294.00

We will explore how to sort parallel arrays in the next program:



Video shop

A shop hires out videos of popular films. For each video, details are kept of :

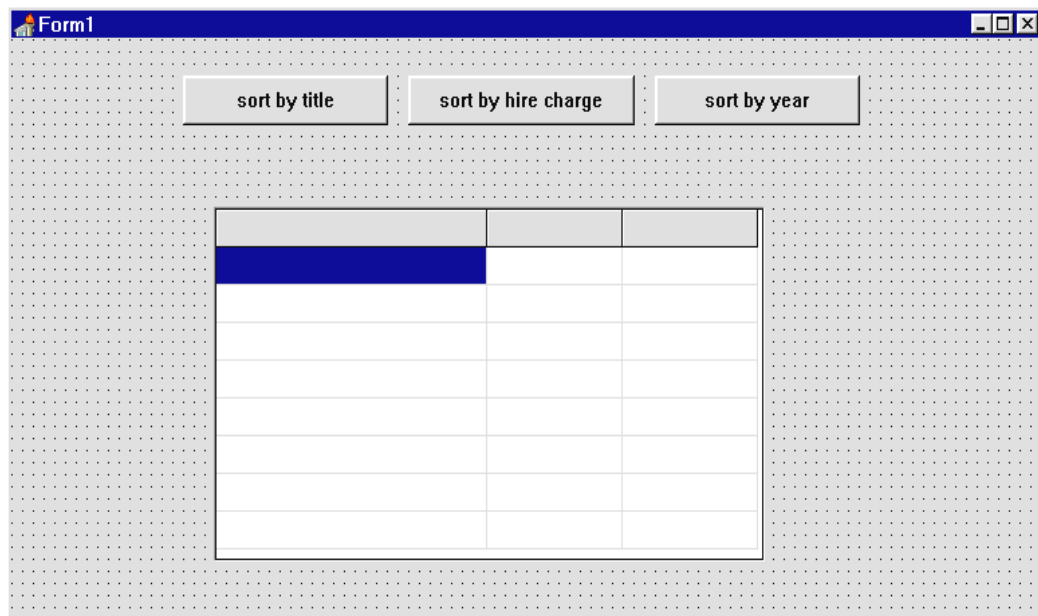
- title
- hire charge
- year produced

A program is required which can sort the information according to any of these categories of data.

Set up a new directory VIDEO 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 a string grid to the form and set the properties:

DefaultColWidth	100
ColCount	3
FixedCols	0
RowCount	9
Options:	
goEditing	True
ScrollBars	None



Drag the first column so that it is twice as wide as the other columns.

Put three buttons above the string grid with the captions '*sort by title*', '*sort by hire charge*', and '*sort by year*'.

Double-click on the dotted grid of the form to produce an event handler, then add lines to write captions for the string grid columns:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    stringgrid1.cells[0,0]:='Title';
    stringgrid1.cells[1,0]:='Hire charge';
    stringgrid1.cells[2,0]:='Year';
end;
```


When data is entered, this will be stored in three parallel arrays. Set these up in the 'public declarations' section:

```
public
  { Public declarations }
  title: array[1..8] of string;
  charge: array[1..8] of real;
  year: array[1..8] of integer;
end;
```

Notice how the type of variable depends on the data which will be entered:

- '*title*' will be text, so is stored as a **string**
- '*charge*' will be a decimal number representing pounds and pence, so is stored as **real**
- '*year*' will be a whole number, so is stored as an **integer**.

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 procedure. Add the lines of program to transfer data from the string grid to the arrays:

```
procedure TForm1.StringGrid1KeyUp(Sender:
  TObject;var Key: Word; Shift: TShiftState);
var
  x,y:integer;
begin
  x:=stringgrid1.col;
  y:=stringgrid1.row;
  case x of
  0: title[y]:=stringgrid1.cells[0,y];
  1: begin
      if stringgrid1.cells[1,y]='' then
        charge[y]:=0
      else
        charge[y]:=
          strtofloat(stringgrid1.cells[1,y]);
      end;
  2: begin
      if stringgrid1.cells[2,y]='' then
        year[y]:=0
      else
        year[y]:=
          strtoint(stringgrid1.cells[2,y]);
      end;
  end;
end;
end;
```

The procedure begins by setting the variables **x** and **y** to be the **column** and **row** where data has just been entered:

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

A CASE command is then used to transfer the data into one of the three arrays, depending on the column where the entry occurred:

```
case x of  
  0: {store data in the title array}  
  1: {store data in the charge array}  
  2: {store data in the year array}  
end;
```

Entries in the string grid need to be converted to **real** numbers for the **charge** array, or **integers** for the **year** array.

Title	Hire charge	Year
Local Hero	2.40	1983
Wayne's World	2.00	1992
Terminator 2	2.50	1991
Evita	1.50	1987
Starship troopers	3.99	1998
Spiceworld: the Movie	3.99	1998
Apollo 13	2.00	1993
Grease	1.99	1978

Compile and run the program. Check that the column headings are displayed correctly and that data can be typed into the string grid. Test the error trapping for decimal numbers in the *'hire charge'* column and whole numbers in the *'year'* column, then return to the Delphi editing screen.

The next step is to write procedures to carry out the sorting. We will deal with the alphabetical sort of the titles first:

Double-click the 'sort by title' button to create an event handler, then add the lines of program:

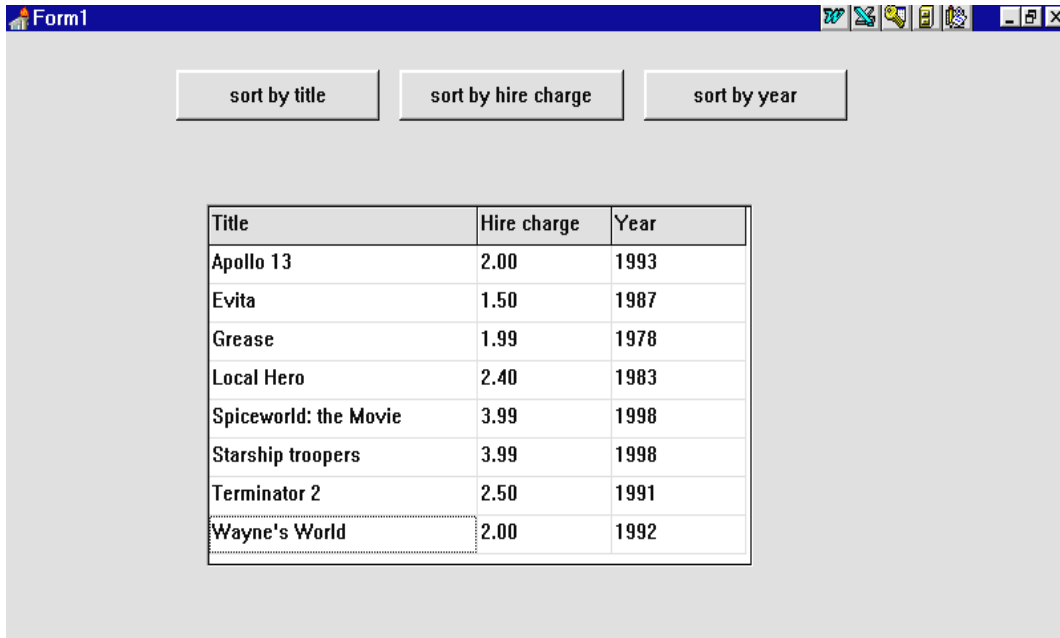
```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,j:integer;
  temptitle:string;
  tempcharge:real;
  tempyear:integer;
begin
  for i:=1 to 7 do
    for j:=i+1 to 8 do
      begin
        if title[j]<title[i] then
          begin
            temptitle:=title[i];
            tempcharge:=charge[i];
            tempyear:=year[i];
            title[i]:=title[j];
            charge[i]:=charge[j];
            year[i]:=year[j];
            title[j]:=temptitle;
            charge[j]:=tempcharge;
            year[j]:=tempyear;
          end;
        end;
      for i:=1 to 8 do
        begin
          stringgrid1.cells[0,i]:=title[i];
          stringgrid1.cells[1,i]:=
            floattostrf(charge[i],ffFixed,8,2);
          stringgrid1.cells[2,i]:=inttostr(year[i]);
        end;
      end;
end;
```

This is very similar to the sort procedure we used earlier in the chapter, but notice that three temporary variables are needed: *temptitle*, *tempcharge* and *tempyear*.

If the two **titles** being compared are not in correct alphabetical order, a triangular exchange is carried out with the **titles** plus the corresponding **hire charge** and **year** entries.

After the sorting is completed, the final loop copies the data back into the string grid in its new positions. The real numbers in the **charge** array and the integers in the **year** array need to be converted to strings before being displayed.

Compile and run the program. Enter test data and check that the alphabetical sorting by title works correctly. When a position change occurs, the complete set of *title*, *hire charge* and *year* entries should move as a group:



Return to the Delphi editing screen. Double-click the '*sort by hire charge*' button to create an event handler.

Use the **Edit/Copy/Paste** facility to copy the entire contents of the '*sort by title*' ButtonClick procedure into the one just created for the '*sort by hire charge*' button. The only change that needs to be made is the comparison line:

```

.....
for j:=i+1 to 8 do
begin
  if charge[j]<charge[i] then change only this line
  begin
    temptitle:=title[i];
    tempcharge:=charge[i];
  end
end
.....

```

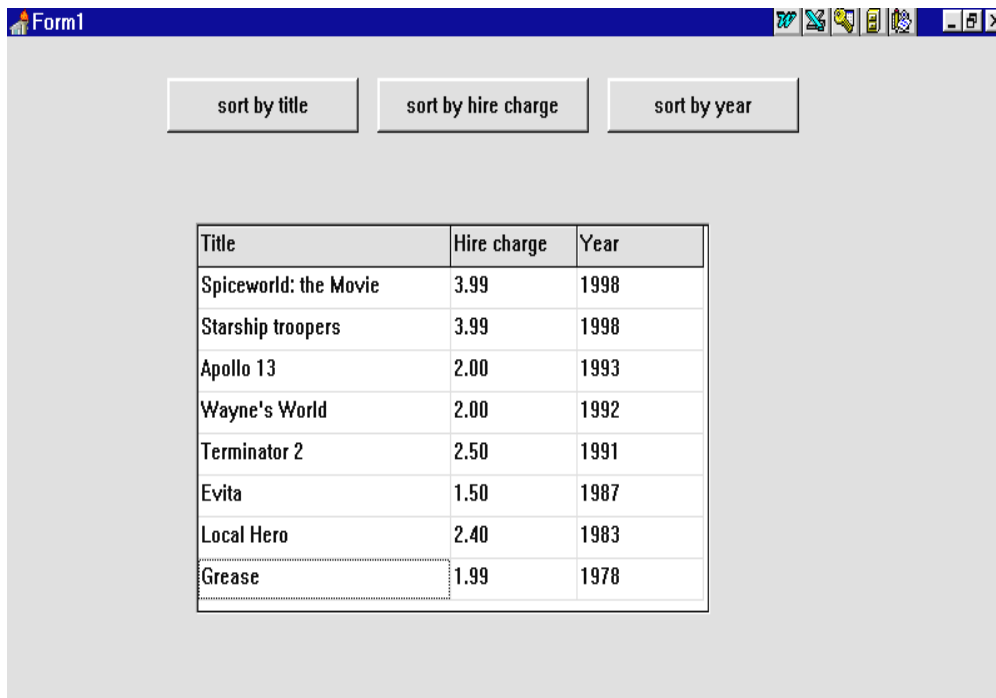
Now double-click the '*sort by year*' button to create an event handler.

As before, use the **Edit/Copy/Paste** facility to copy the entire contents of the '*sort by title*' ButtonClick procedure into the one just created for the '*sort by year*' button. Only change the comparison line:

```
.....  
for j:=i+1 to 8 do  
begin  
  if year[j]>year[i] then change only this line  
  begin  
    temptitle:=title[i];  
    tempcharge:=charge[i];  
    .....
```

Compile and test the completed program. It should be possible to sort the entries into:

- alphabetical order of title
- ascending order of hire charge with the cheapest video at the top
- descending order of year, with the most recent film at the top



example of sorting by year

For the final project in this chapter we will tackle a more complicated program which again involves sorting of data:



Washing-up liquid testing

A consumers' organisation carries out tests on different brands of washing-up liquid, to find which is the best value for money. The tests involve preparing a large supply of identical greasy plates, then seeing how many of the plates can be washed with a single container of each brand of liquid. You are asked to produce a computer program to analyse the results of the tests and list the brands in order from *best value* to *worst value*.

You may assume that the number of brands to be tested will be between 2 and 8.

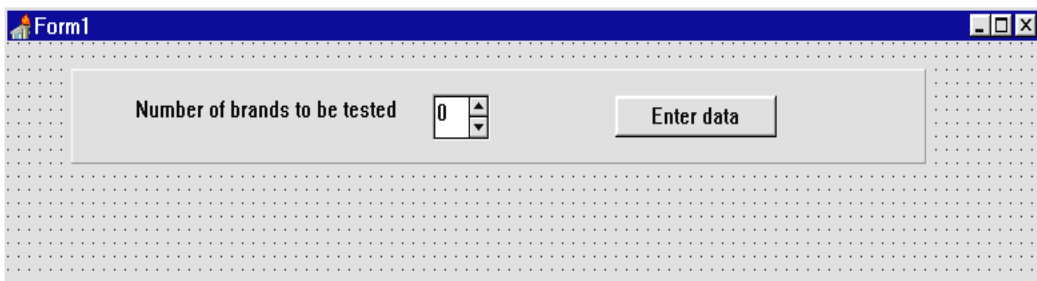
To start this program, set up a new directory WASHUP and save a Delphi project into it. Use the Object Inspector to **maximize** the form, then drag the grid to nearly fill the screen.

The strategy we can use is:

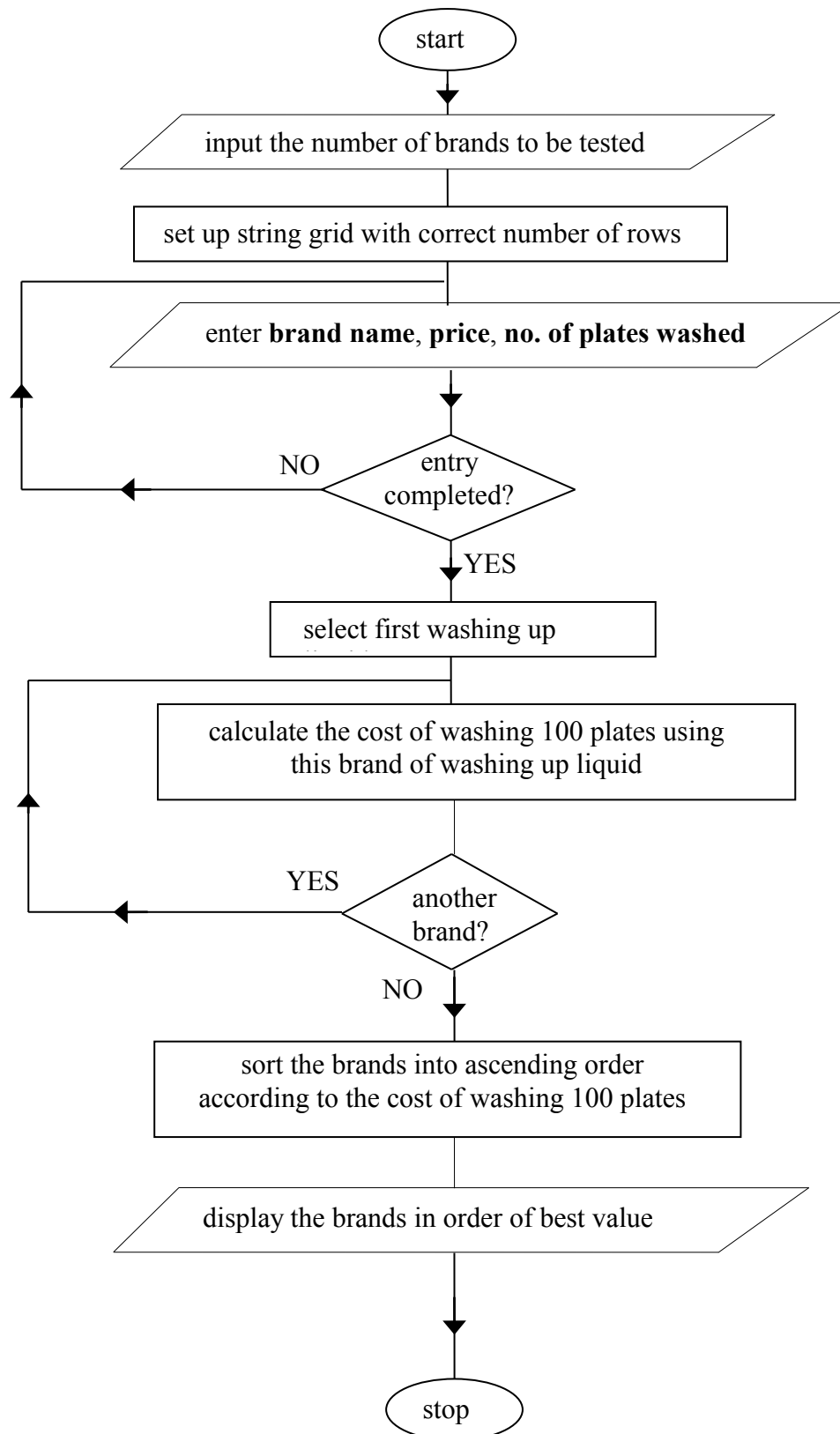
- ask **how many brands** of liquid are being tested
- set up a string grid with the appropriate number of rows
- input the **brand name**, **price** for a container of liquid, and the **number of plates** which can be washed
- calculate for each brand the **cost of washing 100 plates** and display this information in the string grid
- sort the brands according to the cost of washing 100 plates, with the lowest cost brand at the top of the list as the best value.

A flowchart for this sequence is given on the next page.

At the top of the form put a *panel*. Use the Object Inspector to delete the caption from the panel, then add a *spin edit* component:



Flowchart for the washing up liquid program:



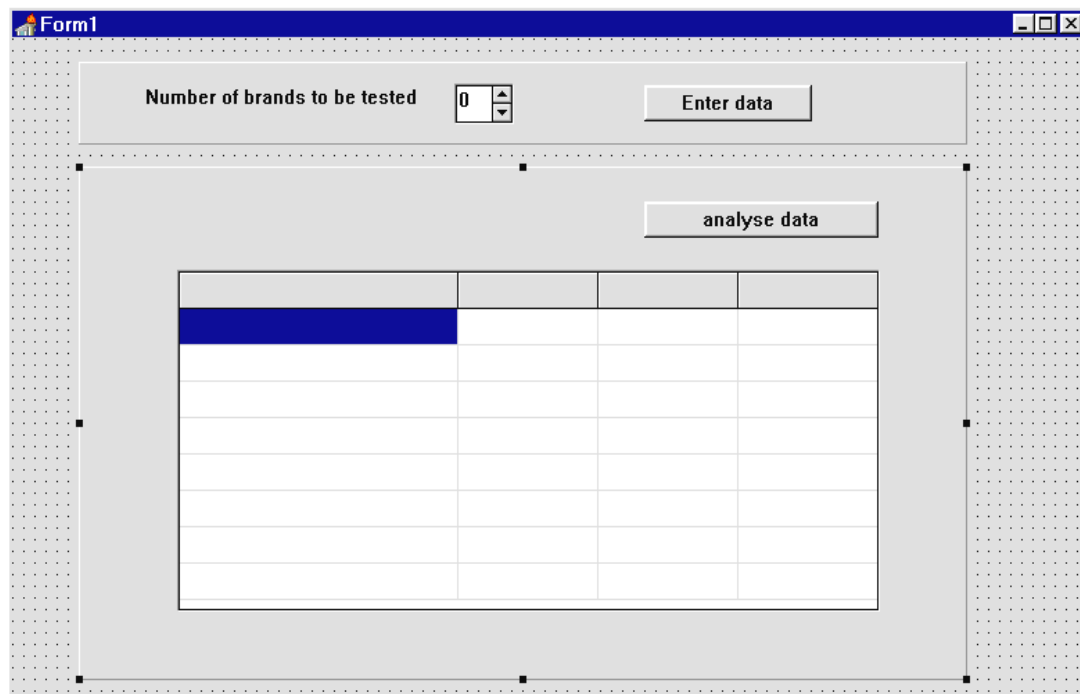
Click on the *spin edit* and press ENTER to bring up the Object Inspector. Set the **MaxValue** property to 8 and **MinValue** to 2.

Add a *label* with the caption '**Number of brands to be tested**', and a *button* with the caption '**Enter data**' as shown above.

Compile and run the program. Check that the *spin edit* can display values between 2 and 8, then return to the Delphi editing screen.

Add another panel to the form and delete its caption. Place a *string grid* on the panel as shown below, and use the Object Inspector to set its properties:

ColCount	4
DefaultColWidth	100
FixedCols	0
RowCount	9
Options:	
goEditing	True
ScrollBars	None



Drag the first column to be twice as wide as the other columns.

Click on the panel outside the string grid and press ENTER to bring up the Object Inspector. Set the **Visible** property to **False**.

Add a button to the panel. Give this the caption '**analyse data**'.

Double-click the **'Enter data'** button on the top panel to create an event handler, then add the lines:

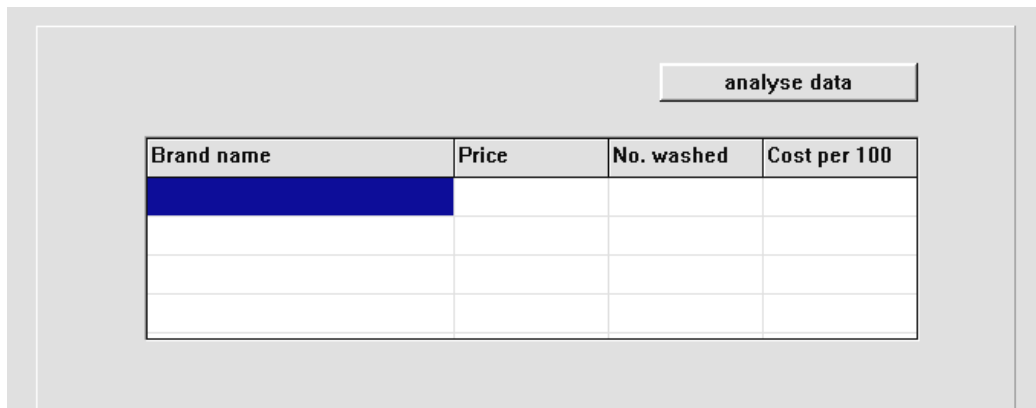
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  panel1.visible:=false;
  panel2.visible:=true;
end;
```

This should hide the *spin edit* panel and make the *string grid* panel visible when the **'Enter data'** button is pressed. Compile and run the program to check that this works correctly, then return to the Delphi editing screen.

Double-click the dotted grid outside the panels to produce an OnCreate event handler for the form. Add lines of program to write captions for the columns of the string grid:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  stringgrid1.cells[0,0]:='Brand name';
  stringgrid1.cells[1,0]:='Price';
  stringgrid1.cells[2,0]:='No. washed';
  stringgrid1.cells[3,0]:='Cost per 100';
end;
```

Run the program to check this, then return to the Delphi editing screen.



We are going to record the number of brands to be tested as the variable **'count'**. Enter this under the Public declarations heading:

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

This variable can be used to set the size of the string grid to have the correct number of input lines. For example, when testing **four** brands of liquid, the input grid should look like the picture above. To set the size of the grid, add the following lines to the **'Enter data'** button click procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  panel1.visible:=false;
  panel2.visible:=true;
  count:=spinedit1.value;
  stringgrid1.height:=25*count+30;
end;
```

Each normal input line on the string grid is 25 units in height, with the top row of headings being 30 units. The calculation

$$25*\text{count}+30$$

therefore gives the total height that the grid should be drawn. Run the program several times with different numbers of brands selected to check that the grid is correctly displayed each time, then return to the Delphi editing screen.

The next step is to set up arrays to hold the brand **names** of the washing-up liquids, their **prices**, the number of **plates** washed, and the calculated costs of washing **100 plates**. Add these to the Public declarations section:

```
public
  { Public declarations }
  count:integer;
  name:array[1..8] of string;
  price:array[1..8] of real;
  plates:array[1..8] of integer;
  hundredcost:array[1..8] of real;
end;
```

We now need an event handler procedure to transfer the *names*, *prices*, and *numbers of plates washed* from the string grid into the three arrays. Click on the string grid and press ENTER to bring up the Object Inspector. Click the **Events** tab and double click alongside **OnKeyUp** to create the procedure, then add the lines:

```

procedure TForm1.StringGrid1KeyUp(Sender: TObject;
                                var Key: Word; Shift: TShiftState);
var
  x,y:integer;
begin
  x:=stringgrid1.col;
  y:=stringgrid1.row;
  case x of
    0:name[y]:=stringgrid1.cells[0,y];
    1:begin
      if stringgrid1.cells[1,y]='' then
        price[y]:=0
      else
        price[y]:=
          strtofloat(stringgrid1.cells[1,y]);
      end;
    2:begin
      if stringgrid1.cells[2,y]='' then
        plates[y]:=0
      else
        plates[y]:=
          strtoint(stringgrid1.cells[2,y]);
      end;
    end;
  end;
end;
end;

```

Compile and run the program. Choose a suitable number of washing-up liquids for testing, then check that test data can be entered in the string grid.

Brand name	Price	No. washed	Cost per 100
Fairy	1.20	346	
Asda Delicate	1.80	622	
Sparkle economy	1.08	531	
Squeezy	1.30	448	
Dish Fresh	0.95	398	
Ultraclean	1.05	204	

The *price* column should be error trapped to accept decimal numbers, and the *number washed* column error trapped for whole numbers.

Return to the Delphi editing screen. We can now think about processing the data to find the best value brand. Double-click the '**analyse data**' button to produce an event handler procedure, then add the lines shown below:

```

procedure TForm1.Button2Click(Sender: TObject);
var
  i:integer;
begin
  for i:=1 to count do
  begin
    hundredcost[i]:=price[i]/plates[i]*100;
    stringgrid1.cells[3,i]:=
      floattostrf(hundredcost[i],ffFixed,8,2);
  end;
end;

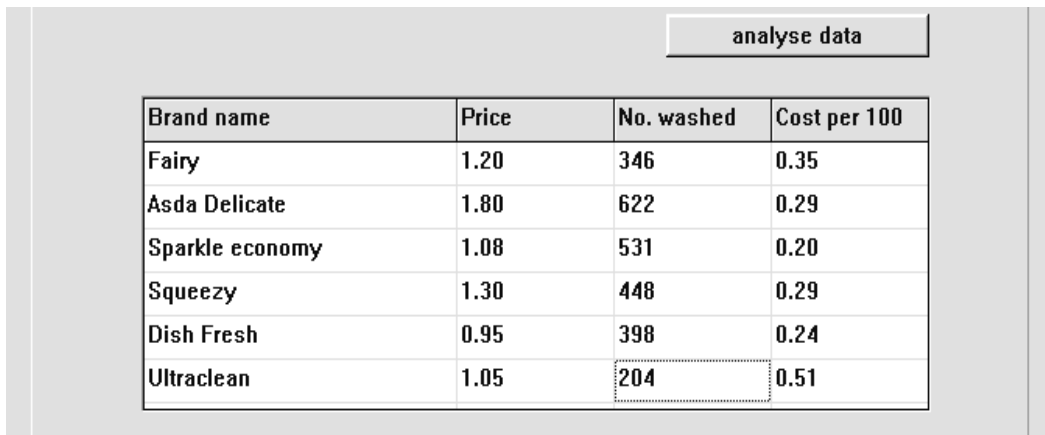
```

This procedure uses a loop to process each of the washing-up liquids in turn. The cost of washing 100 plates is calculated with the formula:

$$\text{hundredcost} = \frac{\text{price of washing up liquid}}{\text{number of plates washed}} * 100$$

The results are then displayed in column 3 of the string grid:

Compile and run the program. Enter test data and check that correct results are produced, then return to the Delphi editing screen.



The screenshot shows a Delphi application window with a button labeled 'analyse data' and a table with the following data:

Brand name	Price	No. washed	Cost per 100
Fairy	1.20	346	0.35
Asda Delicate	1.80	622	0.29
Sparkle economy	1.08	531	0.20
Squeezy	1.30	448	0.29
Dish Fresh	0.95	398	0.24
Ultraclean	1.05	204	0.51

The final step is to sort the parallel arrays so that the best value liquid is at the top of the list. The lines of program to do this should be added to the event handler procedure for the 'analyse data' button. The technique is

similar to the Video Shop program, except that *four* temporary variables will be needed during the triangular exchange.

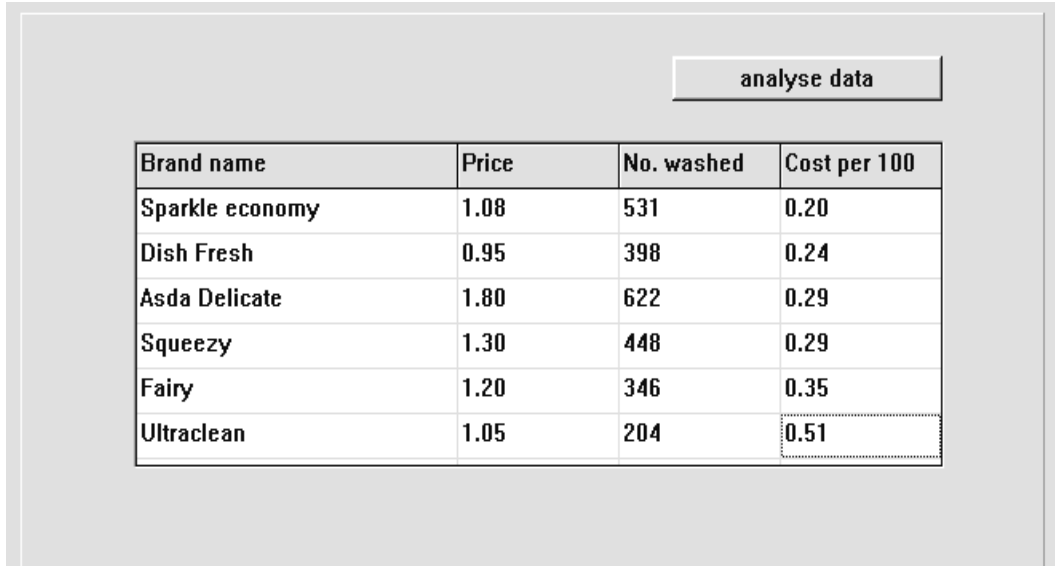
Add the lines shown below:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i,j:integer;
  tempname:string;
  tempprice,tempundredcost:real;
  templates:integer;
begin
  for i:=1 to count do
  begin
    hundredcost[i]:=price[i]/plates[i]*100;
    stringgrid1.cells[3,i]:=
      floattostrf(hundredcost[i],ffFixed,8,2);
  end;
  for i:=1 to count-1 do
    for j:=i+1 to count do
      begin
        if hundredcost[j]<hundredcost[i] then
          begin
            tempname:=name[i];
            tempprice:=price[i];
            templates:=plates[i];
            tempundredcost:=hundredcost[i];
            name[i]:=name[j];
            price[i]:=price[j];
            plates[i]:=plates[j];
            hundredcost[i]:=hundredcost[j];
            name[j]:=tempname;
            price[j]:=tempprice;
            plates[j]:=templates;
            hundredcost[j]:=tempundredcost;
          end;
        end;
    for i:=1 to count do
    begin
      stringgrid1.cells[0,i]:=name[i];
      stringgrid1.cells[1,i]:=
        floattostrf(price[i],ffFixed,8,2);
      stringgrid1.cells[2,i]:=inttostr(plates[i]);
      stringgrid1.cells[3,i]:=
        floattostrf(hundredcost[i],ffFixed,8,2);
    end;
  end;
end;
```



Remember that we do not know in advance how many washing-up liquids will be tested. The variable '**count**' which records the number of liquids is used to control the number of times the sorting loops operate.

Compile and run the program. Enter the test data and check that the results are sorted and displayed correctly:



The screenshot shows a graphical user interface with a button labeled "analyse data" in the top right corner. Below the button is a table with four columns: "Brand name", "Price", "No. washed", and "Cost per 100". The table contains six rows of data, sorted by "Cost per 100" in ascending order.

Brand name	Price	No. washed	Cost per 100
Sparkle economy	1.08	531	0.20
Dish Fresh	0.95	398	0.24
Asda Delicate	1.80	622	0.29
Squeezy	1.30	448	0.29
Fairy	1.20	346	0.35
Ultraclean	1.05	204	0.51

SUMMARY

In this chapter you have:

- Seen how the *bubble sort* algorithm works
- Seen the need for a *triangular exchange* of array items
- Sorted an array of text strings into alphabetical order
- Sorted an array of numbers into *ascending* order
- Sorted an array of numbers into *descending* order
- Sorted parallel arrays, keeping corresponding data items correctly together during exchanges
- Incorporated a bubble sort into a calculation procedure