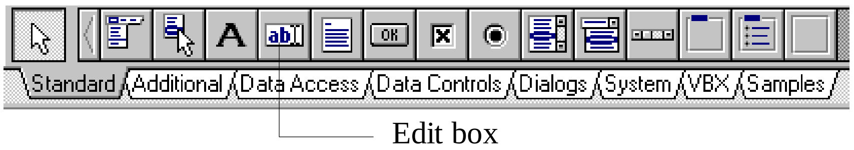
FOUR

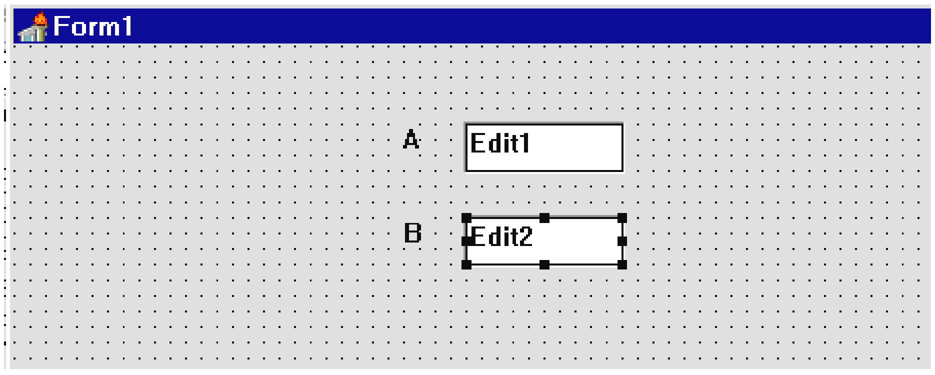
Calculations

Many computer programs require calculations to be carried out. In this section we will examine the way Delphi handles the input and processing of numbers.

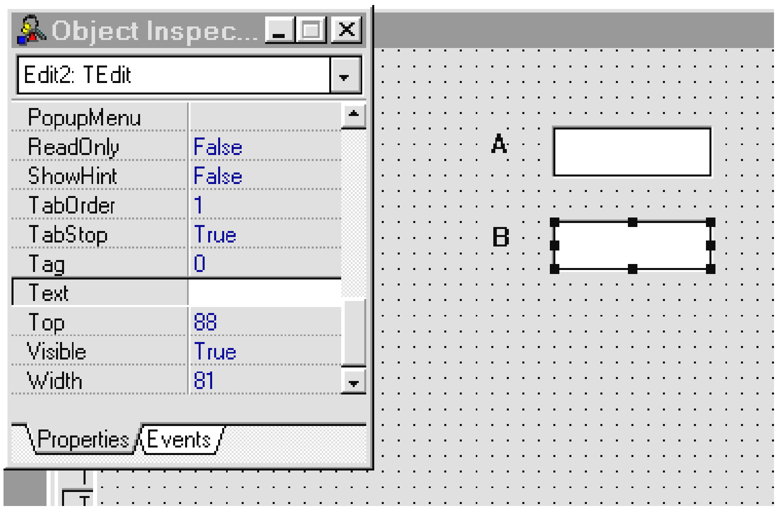
Let us start with a very simple program to add two numbers. Set up a new sub-directory called ADD, open a Delphi project and save it into the sub-directory. Set **Form1** to be **Maximized**, and drag the grid to nearly fill the screen. From the STANDARD component menu select the Edit box:



Place two **edit boxes** on the form. Select the **Label** component and place the labels 'A' and 'B' alonside the edit boxes:



Click on **edit1** and press ENTER to bring up the Object Inspector window. Find the Text property and delete the word 'Edit1', leaving the right hand column blank.
Repeat for **edit2** so that no text appears in the edit boxes on the form.



PopupMenu	
ReadOnly	False
ShowHint	False
TabOrder	1
TabStop	True
Tag	0
Text	
Top	88
Visible	True
Width	81

Compile and run the program. You will find that you can click the mouse in either of the edit boxes to make a text cursor appear. Text or numbers can then be typed into the box, edited or deleted. Exit from the program and return to the Delphi editing screen.

Variables

While a program is carrying out calculations it is usually necessary to store numbers in the computer memory. This is done using memory locations known as *variables*. Each variable is given a name to identify it. Lines of program can then be written to store numbers into the variables.
For example:

```
A:=24;
```

means 'store the value 24 into a memory location called A'.

```
C:=A+B;
```

means 'add the contents of memory locations A and B, then store the answer in memory location C'.

When our program runs, the user will type two numbers into the edit boxes. We will need to store these as variables so that a formula can be used to add them. The answer can then be displayed on the screen. Let's store the input numbers in variables called A and B. To do this, first open the **Unit1** program window and scroll to the point marked 'public declarations'. Add the line shown below:

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
    A, B: integer;
  end;
```

This is telling the computer to reserve two memory locations for our use, and to give these the names **A** and **B**. The word 'integer' tells the computer that the numbers to be stored in A and B will be whole numbers - with no decimal fraction.

When a number is typed into edit box 1, the computer should store this in memory location A. To do this we will need to set up an event handling procedure. Go to the Form1 grid and double-click the mouse on the first edit box. An event handling procedure called **Edit1Change** will appear.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if edit1.text='' then
     A:=0
  else
     A:=strtoint(edit1.text);
end;
```

Add the lines of program shown above. This needs a bit of explanation!

Whatever is typed into the edit box is refered to as **edit1.text.** Two quote marks with nothing in between means that the box is empty, and in this case it is best to store a zero value in variable A.

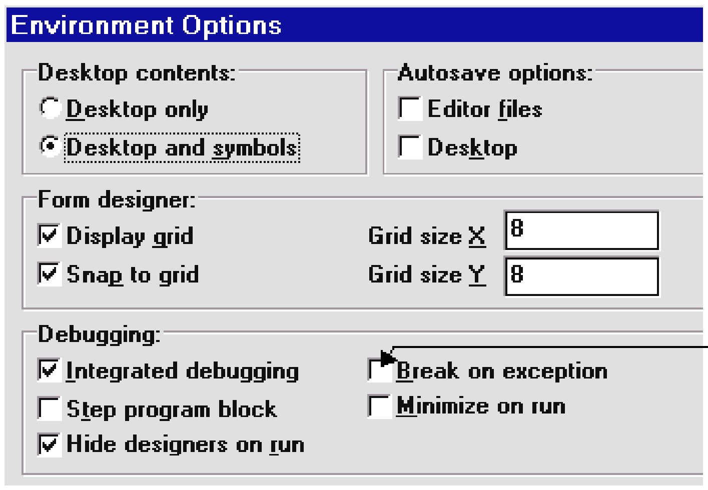
If text really has been entered in the box, this must be converted into a number before it can be stored in variable A. The conversion is carried out by the instruction **strtoint -** this is short for **'string to integer'** (A piece of text is referred to as a 'string').

Could anything go wrong?

Yes, the user might make a mistake and type letters in the box instead of a number. Fortunately the computer will give an automatic warning if a wrong entry has been made - this is known as **error trapping**.

Check that the error trapping is set up correctly by going to the OPTIONS drop down menu and selecting **Environment**. A choice window will appear.

Make sure that there is no tick alongside the option 'Break on exception' then click OK to exit.



Environment Options

Desktop contents:
Desktop only [✓ Desktop and symbols]

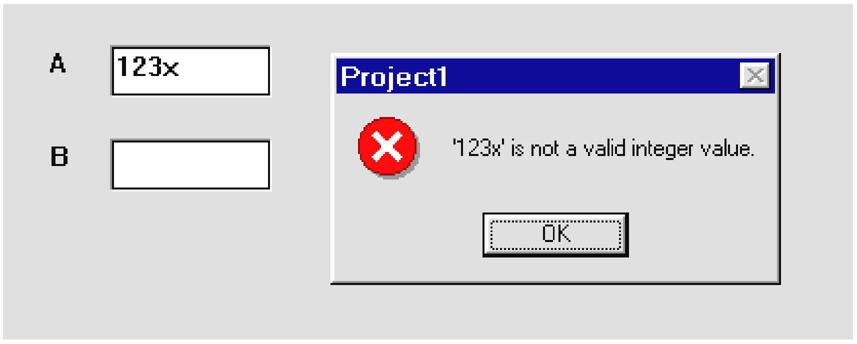
Autosave options:
☐ Editor files
☐ Desktop

Form designer:
☑ Display grid
☑ Snap to grid
Grid size X: 8
Grid size Y: 8

Debugging:
☑ Integrated debugging
☐ Step program block
☑ Hide designers on run
☐ Break on exception
☐ Minimize on run

Make sure there is not a tick in this box

We can try out the error trapping. Compile and run the program. If anything other than a whole number is typed into the first edit box, an error message should appear:



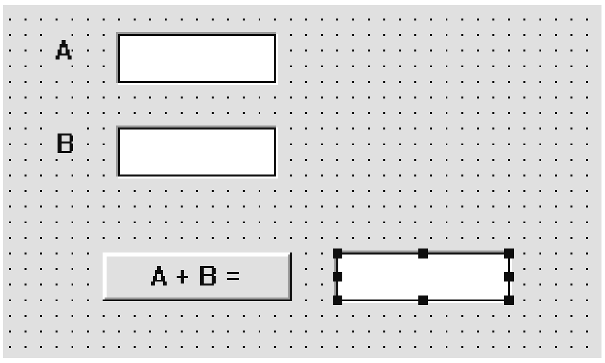
'123x' is not a valid integer value.

Return to the Delphi editing screen and set up a similar event handing procedure for edit box 2 ; double-click on the edit box and add lines of program to the event handler procedure which appears:

```
procedure TForm1.Edit2Change(Sender: TObject);
begin
  if edit2.text='' then
     B:=0
  else
     B:=strtoint(edit2.text);
end;
```

This time we want the number to be stored in the variable B.

The next step is to get the program to add the numbers and display the result. Go to the Form1 grid and add a button. Label this 'A + B ='. Place another edit box alongside the button and use the Object Inspector to blank out the text in the box.



Double-click the button to create an event handler, then add the lines shown below.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  C:=A+B;
  edit3.text:=inttostr(C);
end;
```

Here we are asking the computer to add the numbers stored in variables A and B, and to store the result in a variable called C. A memory location for C has not yet been set up, so go back to the public declarations section near the top of the program and add this variable name to the list:

```
public
   { Public declarations }
   A, B, C: integer;
```

The **inttostr** command (short for '**integer to string**') converts the answer C back into text which can be displayed in the edit box.

We can now finally compile the program and try it out.

When your program is working correctly, try adding another button to multiply A and B and display the result. Set up a new variable D for this. The line of program needed to carry out the multipliction is:

```
D:= A * B;
```

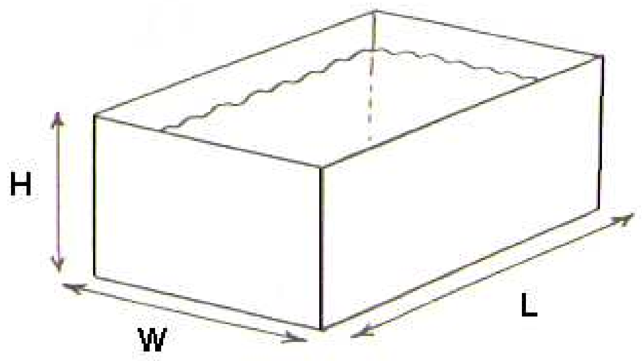
Notice that the computer uses a *star* symbol for multiplication rather than 'x'. This avoids confusion with the letter of the alphabet.

Note: When testing the multiplication please use small numbers. Integer variables are only designed to handle numbers up to a few thousand - larger values will give an incorrect negative result. We will be seeing how to handle large numbers shortly in the course.

Using formulae

The next program will involve slightly more complicated calculations:

A company manufactures water tanks for use in domestic plumbing systems. A customer can ask for a tank to be made with any length (L), width (W) and height (H).



The bottom and sides of the the tank are made from sheet steel which is welded together with no overlap. The tank is open at the top.

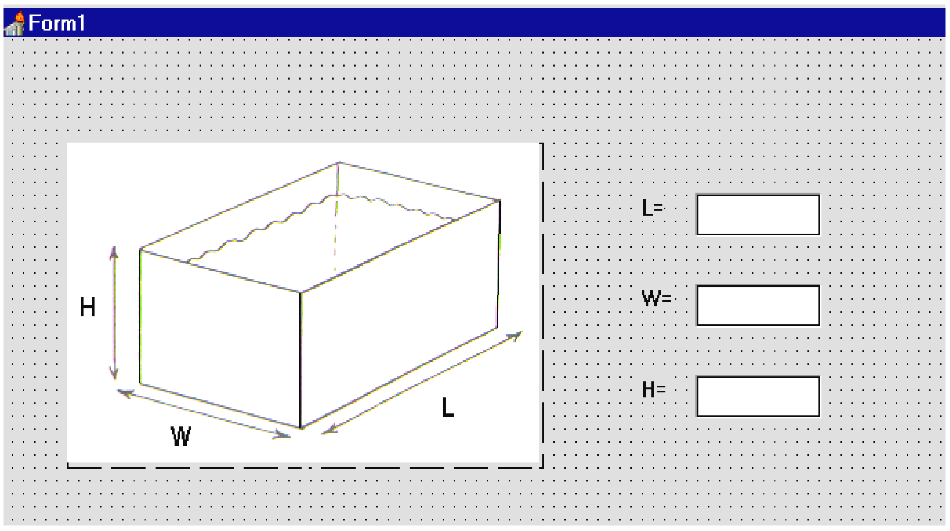
A program is required which will:

1. Input the length, width and height of the tank required (in metres).
2. Calculate the maximum volume of water (in cubic metres) which the tank could hold.
3. Calculate the area of sheet steel (in square metres) needed to make the tank.

Begin by setting up a new sub-directory called TANK, open a Delphi project and save it into the sub-directory.

Use the Object Inspector to maximize the form, and drag the grid larger to nearly fill the screen.

A diagram of the water tank is stored on disc as TANK.BMP. Place an **image** box on the grid and load the diagram. Put three **edit** boxes alongside and add **labels**:



The next task is to set up variables to store the measurements entered by the user. Select the program Unit window and go to the Public declarations area. Add the line shown below:

```
private
  { Private declarations }
public
  { Public declarations }
  L,W,H:real;
end;
```

It is likely that the length, width and height measurements for the tank will require decimal fractions, e.g. 2.75 metres, so we cannot use variables which are **integers.** Instead we choose to make the variables **real** numbers which can have a decimal fraction part.

Double-click on the edit box alongside the caption 'L=' to produce an event handler. Add lines of program to store the input value in the variable **L**:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if edit1.text='' then
     L:=0
  else
     L:=strtofloat(edit1.text);
end;
```

The command **strtofloat** (short for '**string to floating point number**') is used to convert the edit box text into a number which can have a decimal fraction.

Compile and run the program. Check that decimal numbers (e.g. 2.75) can be entered successfuly, but incorrect entries (such as 2.7x) cause an error message window to appear.

Exit from the program and add similar event handlers for the other two edit boxes:

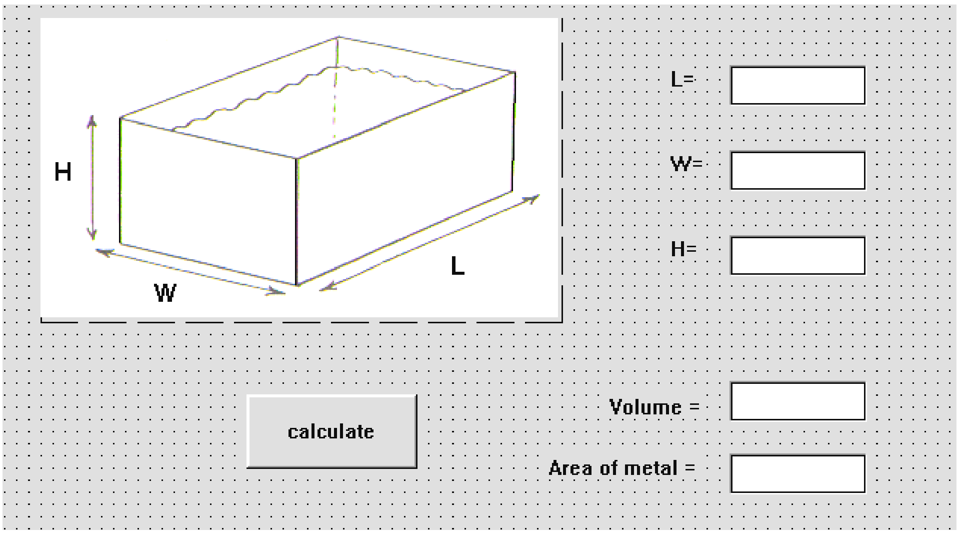
```
procedure TForm1.Edit2Change(Sender: TObject);
begin
  if edit2.text='' then
     W:=0
  else
     W:=strtofloat(edit2.text);
end;

procedure TForm1.Edit3Change(Sender: TObject);
begin
  if edit3.text='' then
     H:=0
  else
     H:=strtofloat(edit3.text);
end;
```

We must now think about displaying the results required. Go to the form grid and add two more edit boxes and the captions:

' **Volume=** ' and ' **Area of metal** ='

Also add a button as shown below, and give this the caption ' **calculate** ' :



Double-click the '**calculate**' button to create an event handler procedure to carry out the calculations.

Finding the volume is simply a case of multiplying the length, width and height of the tank. Add lines of program to do this and display the result:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  volume:=L*W*H;
  edit4.text:=floattostr(volume);
end;
```

We are going to store the result in a variable called **volume.** This name must be added to the list under **public delarations**:

```
  public
    { Public declarations }
    L,W,H:real;
    volume:real;
  end;
```

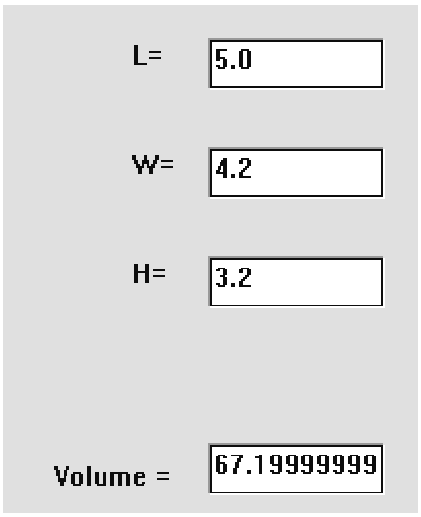
Compile and run the program. Enter example figures for length, width and height, then press the 'calculate' button - the volume of the tank should be displayed. You might check that the answers are correct using a calculator.

In many cases a satisfactory answer is displayed, but sometimes the program seems to be slightly inaccurate. Try, for example:

L = 5.0 **W = 4.2** **H = 3.2**

The correct answer should be 67.2 but the computer displays 67.19999....999

This is due to a slight inaccuracy which can occur when the computer carries out calculations with decimal numbers. The inaccuracy is usually too small to matter, but it makes the result appear strange.

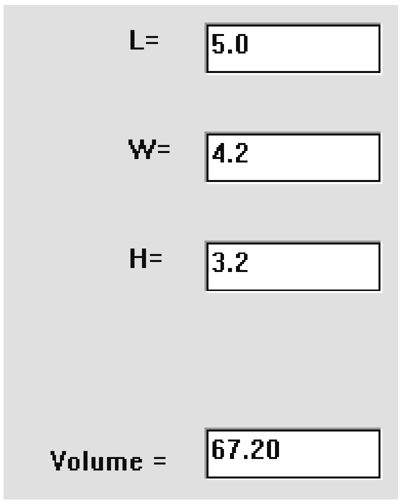


The solution to the difficulty is to make the program display the result with a fixed number of decimal places - two decimal places would be suitable in this case. We need to modify the ButtonClick event handler:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  volume:=L*W*H;
  edit4.text:=floattostrf(volume,ffFixed,8,2);
end;
```

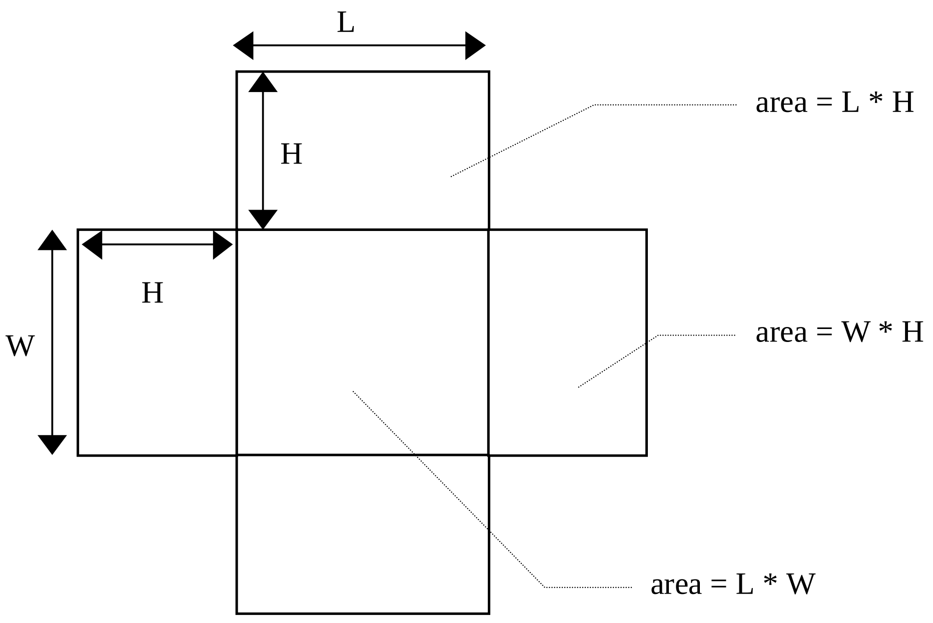
The instruction **floattostrf** means '**float to string formatted**'. The final number 2 specifies the number of decimal places required. The number 8 specifies that numbers of up to 8 digits are to be accepted.

Run the program using the test data as before. A correct screen display is now produced.



Return to the Delphi editing screen.

The next stage is to calculate and display the area of sheet metal needed to make the tank.
Imagine that the tank has been unfolded into a flat shape:



The base of the tank has an area given by L * W
Each of the sides has an area of L * H
Each of the ends has an area of W * H

The total area of metal required to make the tank is therefore:

(L * W) + 2 * (L * H) + 2 * (W * H)

This formula can be incorporated into the calculation procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  volume:=L*W*H;
  edit4.text:=floattostrf(volume,ffFixed,8,2);
  area:= (L*W)+2*(L*H)+2*(W*H);
  edit5.text:=floattostrf(area,ffFixed,8,2);
end;
```

Don't forget to add the variable name '**area**' into the public declarations list:

```
 public
    { Public declarations }
    L,W,H:real;
    volume,area:real;
```

Application

You can now test the completed program and save it on disc.

Fast food restaurant

The next project which we will work on makes use of variables and formulae to calculate the bill for a customer in a fast food restaurant. The restaurant menu offers:

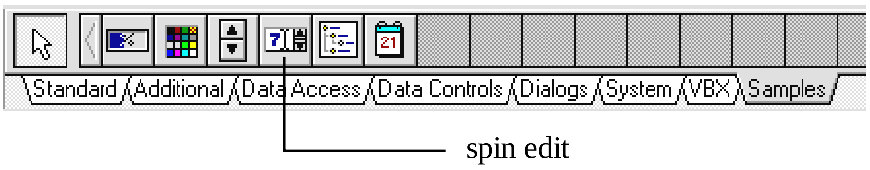
Megaburger	£1.20
Megaburger with cheese	£1.80
Fries	£0.60
Coke	£0.40
Cheesecake	£0.90

Set up a new sub-directory and start a Delphi project. Maximise the form, and drag the form grid to nearly fill the screen.

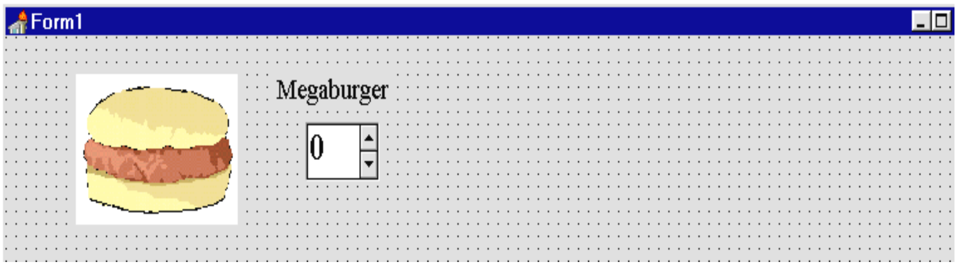
We are going to create a graphical display of the menu, with entry boxes for the user to show how many of each item have been ordered by the customer.

Begin by putting an image box near the top left hand corner of the Form grid. Load a picture of a Megaburger, stored on disc as the file FOOD1.BMP.

Go to the SAMPLES component menu and select **spin edit**:



This component provides a number window which can be directly edited from the keyboard or can be changed by clicking the mouse on small up/down arrows. Place a spin edit box on the grid next to the image, and add a label 'Megaburger'.



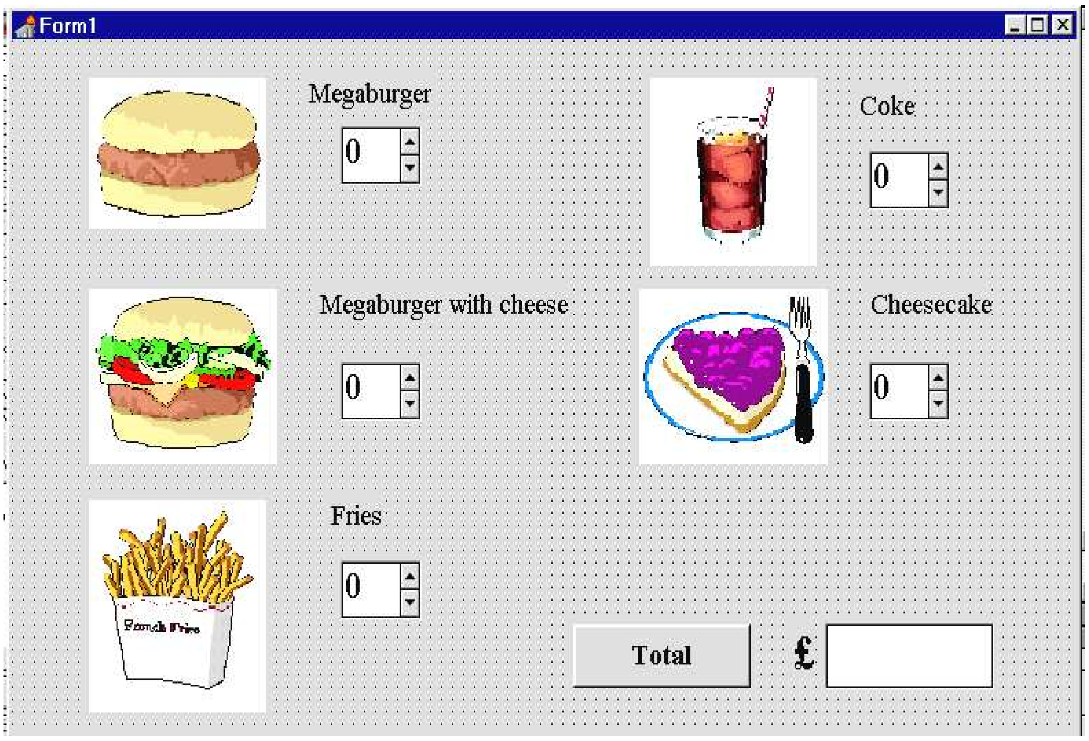
Select the spin edit box and bring up the Object Inspector window. Change the **font** property to

Times New Roman Bold 18 point

or a suitable equivalent large type face. Set the **MaxValue** property to 20 to limit the range of numbers which the spin edit box will accept.

Compile and run the program to test out the spin edit component, then return to the Delphi editing screen.

Complete the menu screen by adding four more image boxes to the grid and loading the bitmap files FOOD2.BMP - FOOD5.BMP. Provide a spin edit box for each menu item. You might find it easiest to use **Edit/Copy** and **Edit/Paste** to replicate the first spin edit box; this will avoid the need to reset the **Font** and **MaxValue** properties each time. Complete the menu by adding labels for each food item.



Place a button component at the bottom of the grid and give this the caption 'Total'. Alongside add an edit box and a '£' label as shown above.

Compile and run the program to check that all the spin edit boxes operate correctly, then return to the Delphi editing screen.
We now need to produce the sections of program which will calculate the amount that the customer has to pay.

Begin by adding a goup of lines just above the **implementation** section:

```
var
  Form1: TForm1;

const
  mega=1.20;
  megacheese=1.80;
  fries=0.60;
  coke=0.40;
  cake=0.90;

implementation
```

These lines specify the price of each of the food items. The heading **const** (short for **constant**) means that the values are fixed. This is in contrast to **variables** which can take on different values each time the program runs - for example we used variables for the length, width and height of the water tank which may be different for each tank manufactured.

Double-click the mouse on the spin edit box for the megaburger to produce an event handler procedure. Add a line of program:

```
 procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  megaCost:=mega*SpinEdit1.Value;
end;
```

We are telling the program to find the total cost of the megaburgers by multiplying the burger price by the number shown in the SpinEdit box. The cost will be stored as a variable called **megaCost.**

Now double-click the spin edit box alongside the megaburger with cheese, and add a similar line to the event handler:

```
procedure TForm1.SpinEdit2Change(Sender: TObject);
begin
  megacheeseCost:=megacheese*SpinEdit2.Value;
end;
```

The two variables need to be added to the public declarations list:

```
  public
    { Public declarations }
```

```
        megaCost, megacheeseCost:real;
    end;
```

Finally, double-click the 'Total' button to produce an event handler and add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  total:=megaCost + megacheeseCost;
  edit1.text:=floattostrf(total,ffFixed,8,2);
end;
```

The variable '**total**' represents the sum of the items ordered, and will be displayed with two decimal places to represent poinds and pence. Add '**total**' to the public declarations:

```
public
   { Public declarations }
   megaCost, megacheeseCost:real;
   total:real;
end;
```

Compile and run the program. You should be able to order megaburgers and megaburgers with cheese, then display the total price by clicking the 'Total' button.

The figures for the total may look too small, so you might like to use the Object Inspector to set the **font** property for the **edit box** to:

Times New Roman   Bold   18 point

or something similar.

See if you can complete the project by adding event handlers for the other spin edit boxes. Extra variables:

friesCost   **cokeCost**   **cakeCost**

will need to be added to the public declarations list.

SUMMARY

In this chapter you have:

- Used edit boxes to input integer and decimal numbers
- Used the error trapping system to detect incorrect entries
- Stored input numbers as **variables**
- Carried out calculations in a program using formulae
- Output the results of calculations, either as integers or with a fixed number of decimal places
- Used the **spin edit** component for inputting integers
- Extered fixed numbers in the program as **constants**