TWELVE

Procedural programming

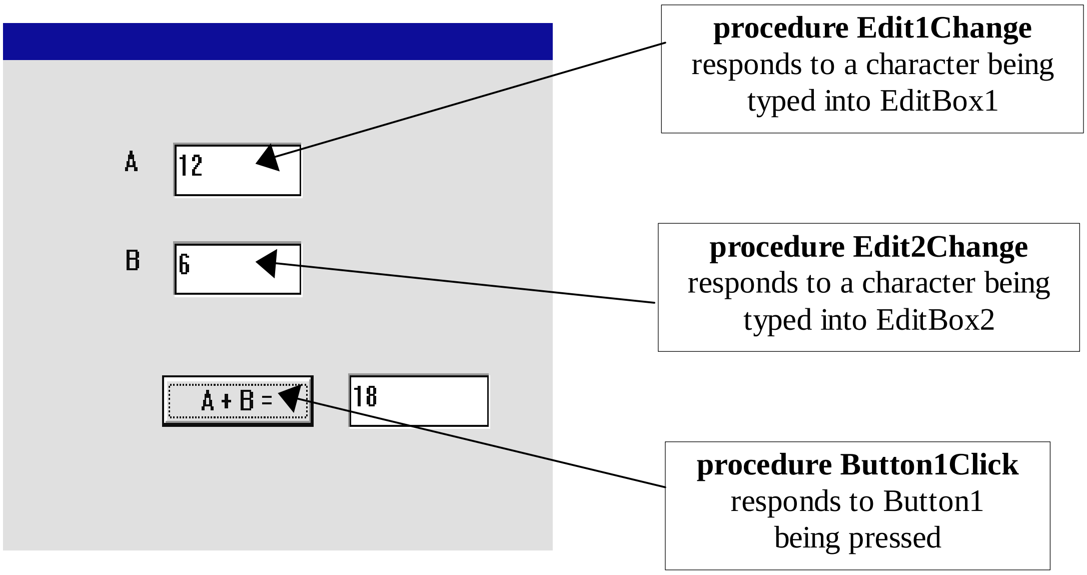
Delphi programs are divided into **procedures**. These are sections of the overall program with specific tasks, such as responding to a button being pressed or a character being typed into an edit box. If you look back for a moment at the ADD program which we produced in chapter 4, this is made up of three separate procedures which are listed under the **implementation** heading:

```
implementation
{$R *.DFM}

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

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

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

end.
```

Notice that each procedure has a **begin...end** pair around the lines of program which do the processing. There is a separate **end** command, followed by a full stop, for the overall program.

When you carry out a block analysis you should draw an outer box around each procedure, as well as boxes around any **loops** or **conditional blocks** inside the procedures.

The procedures in the ADD program are all '**event handlers**' which respond to mouse or keyboard input:



procedure Edit1Change responds to a character being typed into EditBox1

procedure Edit2Change responds to a character being typed into EditBox2

procedure Button1Click responds to Button1 being pressed

The event handler procedures are set up automatically by the Delphi system if we double-click components or select from the **Events** list in the Object Inspector.

Sometimes, however, it is useful to set up our own procedures in a program. The following project demonstates how this is done:

House heating costs

A two storey house is rectangular in plan, with a width of 16 metres and a length of 18 metres. The ceilings are 2.5 metres high.

The householders wish to keep the ground floor rooms at a constant temperature of 20°C and the first floor rooms at a constant temperature of 15°C. The average outside temperature during the year is 8°C.

It is known that for this particular type of house, the heating cost is 4 pence per cubic metre of room space per year for every Centigrade degree by which the room temperature exceeds the outside temperature average.

The householder has the option of installing double glazing and loft insulation at a cost of £680.00. This will lead to a 20% reduction in heating costs. The householder will go ahead with the insulation if there will be sufficient saving to pay for its cost within 8 years.

Calculate whether the householder should insulate the house or not.

A **computer program** is now needed to carry out similar calculations for a variety of different houses. You may assume that all houses will be two stories high and rectangular in plan. Also assume that insulating the house will reduce heating costs by 20%, and that the average outside temperature will be 8°C.

Your program should input the following data:

- width of the house
- length of the house
- height of ceilings
- required downstairs temperature
- required upstairs temperature
- estimated cost of insulation

As before, the heating cost will be 4 pence per cubic metre of room space per year for every Centigrade degree by which the room temperature exceeds the outside temperature average.

The program should output:

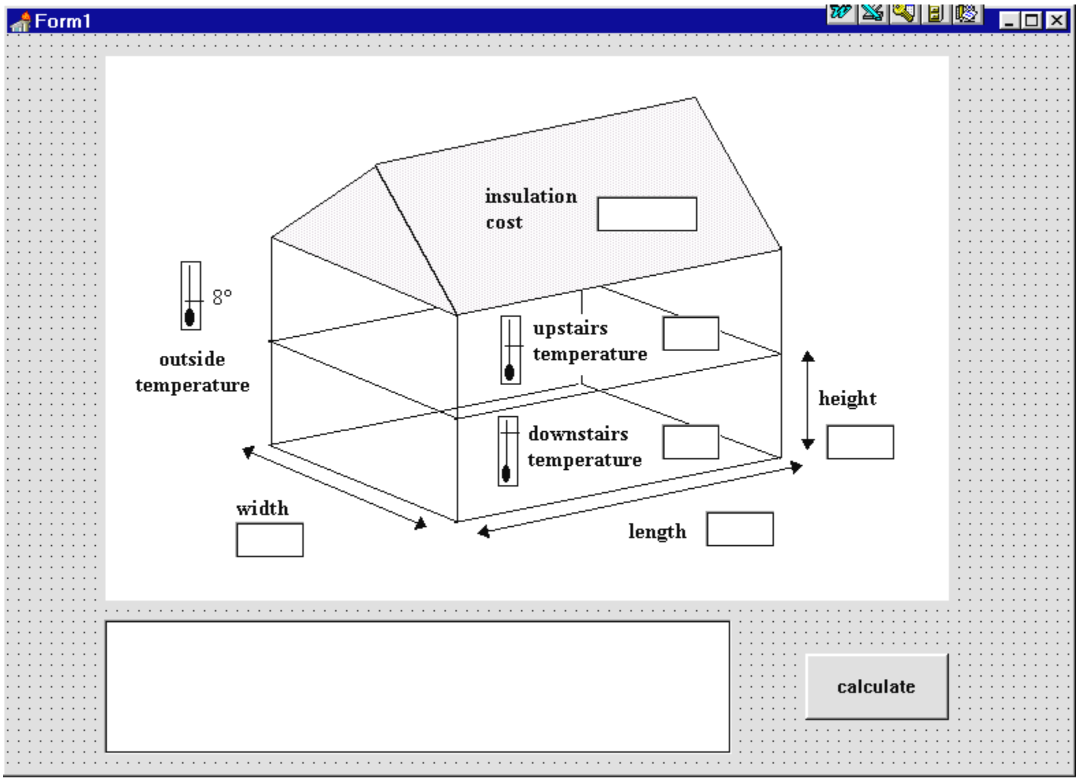
- the annual heating costs
- the saving on heating bills over 8 years with insulation
- a message to indicate whether or not insulation should be installed.

This is going to be quite a complex calculation program, so let's try to break the problem down into stages:

1. Input the house size, required temperatures, and insulation cost.
2. Calculate the heating costs per year for the uninsulated house.
3. Calculate the saving over 8 years if insulation is installed.
4. Decide whether or not to install insulation.

We will begin by setting up a program to carry out the inputs for step 1:

Set up a directory HEATING and save a Delphi project into it. **Maximize** the *Form*, and drag the dotted grid to nearly fill the screen.



Place an *Image Box* on the form and load the bitmap file HEATCOST.BMP. Put six *Edit Boxes* alongside or below the labels on the diagram; these will be used for entering the **length (m)**, **width (m)**, **ceiling height (m),** **downstairs temperature (C)**, **upstairs temperature (C)**, and **insulation cost (£)**.

Add a *List Box* at the bottom of the Form which will be used later for displaying the results of the calculations. Complete the form by adding a button with the caption '**calculate**'.

Go to the *Public declarations* section and list the variables which will be needed to store the input values:

```
public
   { Public declarations }
   length,width,height:real;
   downtemp,uptemp:integer;
   insulation:real;
 end;
```

Double-click the 'length' Edit Box to create an event handler and add the lines:

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

Produce similar event handlers for the '**width**' and '**height**' Edit Boxes which also input decimal numbers.

Double-click the '**Downstairs temperature**' edit box and add the lines to input an integer value:

```
procedure TForm1.Edit4Change(Sender: TObject);
begin
  if edit4.text='' then
    downtemp:=0
  else
    downtemp:=strtoint(edit4.text);
end;
```

The 'Upstairs temperature' Edit Box will require a similar integer input procedure.

Complete the input event handlers with the procedure to accept a decimal number for 'Insulation cost':

```
procedure TForm1.Edit6Change(Sender: TObject);
begin
  if edit6.text='' then
    insulation:=0
  else
    insulation:=strtofloat(edit6.text);
end;
```

Compile and run the program. Check that the temperature input boxes are correctly error trapped to accept whole numbers, and that the other four input boxes are error trapped for decimal numbers. Return to the Delphi editing screen.

We can now consider how to carry out steps 2, 3 and 4 of the program which we listed on page 197. These should occur when the '**calculate**' button is pressed.

It would be possible to put all the lines of program for steps 2, 3 and 4 into the event handler for the '**calculate**' button, but this would become long and complicated - increasing the chances of making a mistake. Instead, we will create three new procedures: YEARCOST, SAVING and DECISION. These will need to be called from inside the event handler for the button.

Double-click the 'calculate' button and add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  yearcost;
  saving;
  decision;
end;
```

Go now to the '**type**' section near the top of the program and find the point where the other procedures are listed. Add our three new ones to the list:

```
          ......
  procedure Edit5Change(Sender: TObject);
  procedure Edit6Change(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure yearcost;
  procedure saving;
  procedure decision;
private
  { Private declarations }
          ......
```

Now move down to the bottom of the program and add the procedure blocks. At the moment they do not contain lines of program to carry out any processing - just comments to indicate their purpose:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  yearcost;
  saving;
  decision;
end;

procedure TForm1.yearcost;
begin
  {calculate annual heating cost}
end;
```

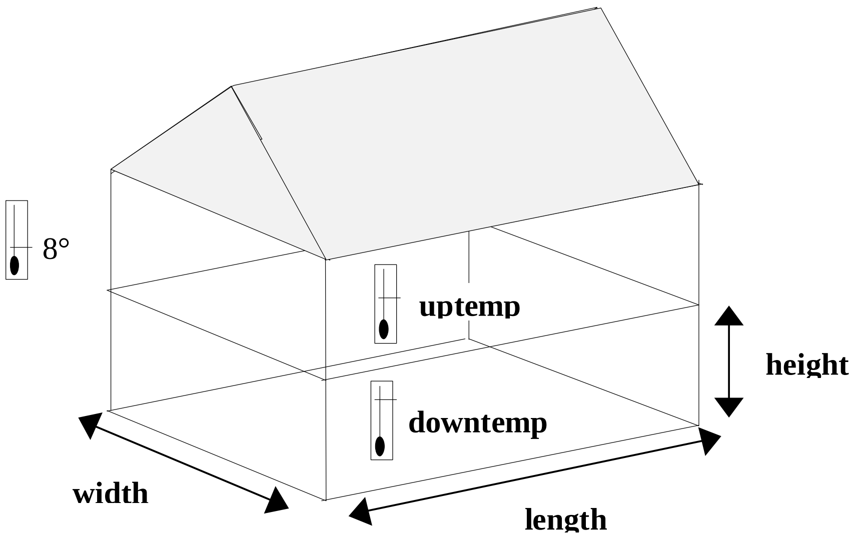
ADD THIS

ADD THIS

```
procedure TForm1.saving;
begin
  {calculate saving in 8 years with insulation}
end;

procedure TForm1.decision;
begin
  {decide if insulation should be installed}
end;

end.
```

As in the case of the event handler procedures provided by Delphi, we must write '**TForm1.**' before the names of our own procedures to show that they belong to Form1 of the project.

We can now begin work on the first of the procedures 'yearcost' which will calculate and display the annual heating costs for the house.



We need to calculate the volume of air heated for each storey of the house. This can be found with the formula:

volume = length * width * height

The heating cost for the ground floor of the house will depend on the difference between the required temperature '**downtemp'** and the outside temperature which averages 8°C.

The annual downstairs heating cost is given by the formula:

downstairs cost = volume * (downtemp - 8) * 0.04

The factor 0.04 is the heating cost of 4 pence (i.e. £0.04) for each cubic metre of air space, for each °C hotter than the outside average temperature.

A similar formula gives the annual upstairs heating cost:

upstairs cost = volume * (uptemp - 8) * 0.04

and the total annual heating cost for the house will be:

annual cost = upstairs cost + downstairs cost

Let's now put these formulae into the **'yearcost'** procedure. Add the lines:

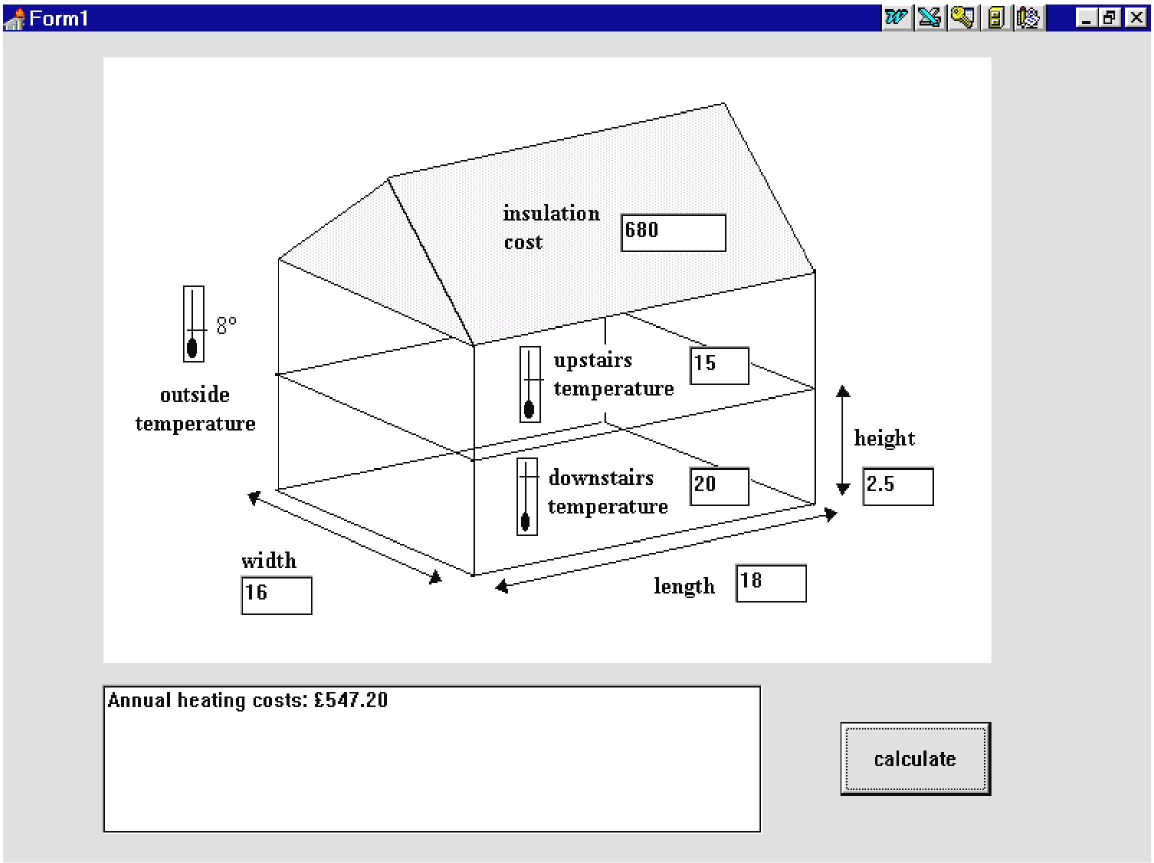
```
procedure TForm1.yearcost;
var
  volume,upcost,downcost:real;
begin
  {calculate annual heating cost}
  volume:=length*width*height;
  downcost:=volume*(downtemp-8)*0.04;
  upcost:=volume*(uptemp-8)*0.04;
  yeartotal:=downcost+upcost;
end;
```

Add the variable '**yeartotal**' to the list under the *Public declarations* heading:

```
 public
   { Public declarations }
   length,width,height:real;
   downtemp,uptemp:integer;
   insulation,yeartotal:real;
 end;
```

Other procedures will need to use the '**yeartotal**' result, so it must be *publicly* available to them. The variables '**volume**','**upcost**' and '**downcost**' are used only in the yearcost procedure, so there is no need to make them *public*.

Now that we have calculated the annual heating cost we must display it on the screen:



Annual heating costs: £547.20

Add lines to the **yearcost** procedure to clear the List Box then display the annual heating cost:

```
procedure TForm1.yearcost;
var
  volume,upcost,downcost:real;
  textline:string;
begin
  {calculate annual heating cost}
  volume:=length*width*height;
  downcost:=volume*(downtemp-8)*0.04;
  upcost:=volume*(uptemp-8)*0.04;
  yeartotal:=downcost+upcost;
  listbox1.clear;
  textline:='Annual heating costs: £' +
             floattostrf(yeartotal,ffFixed,8,2);
  listbox1.items.add(textline);
end;
```

Compile and run the program using the example figures given in the question. Press the '**calculate**' button to check that the yearly heating cost is calculated correctly, then return to the Delphi editing screen.

The next procedure to work on is '**saving**', which should calculate the amount saved over an eight year period if insulation is installed. This will be given by the formula:

8 year saving = annual heating cost * 8 * 20%

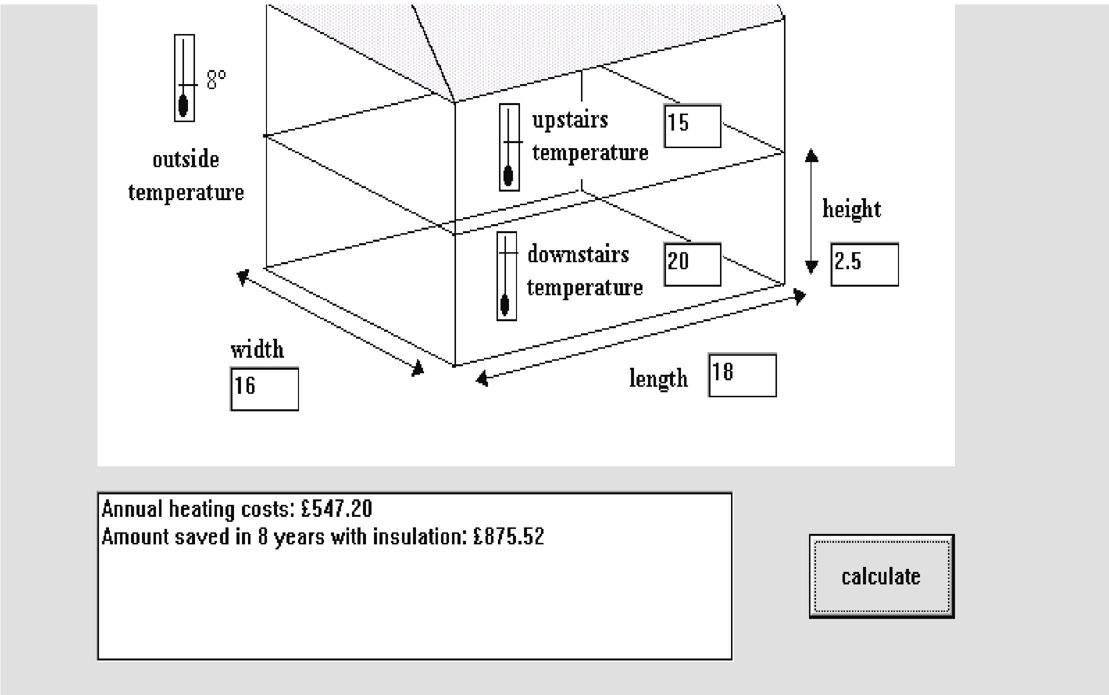
Add lines to the 'saving' procedure to calculate and display this result:

```
procedure TForm1.saving;
var
  textline:string;
begin
  {calculate saving in 8 years with insulation}
  saved:=yeartotal*8*0.2;
  textline:='Amount saved in 8 years with insulation: £'
                     + floattostrf(saved,ffFixed,8,2);
  listbox1.items.add(textline);
end;
```

The variable '**saved**' should be added to the list of *public declarations*:

```
         { Public declarations }
               .....
         insulation,yeartotal,saved:real;
```

Compile and run the program. Use the test data to check that the amount saved is displayed correctly, then return to the Delphi editing screen.



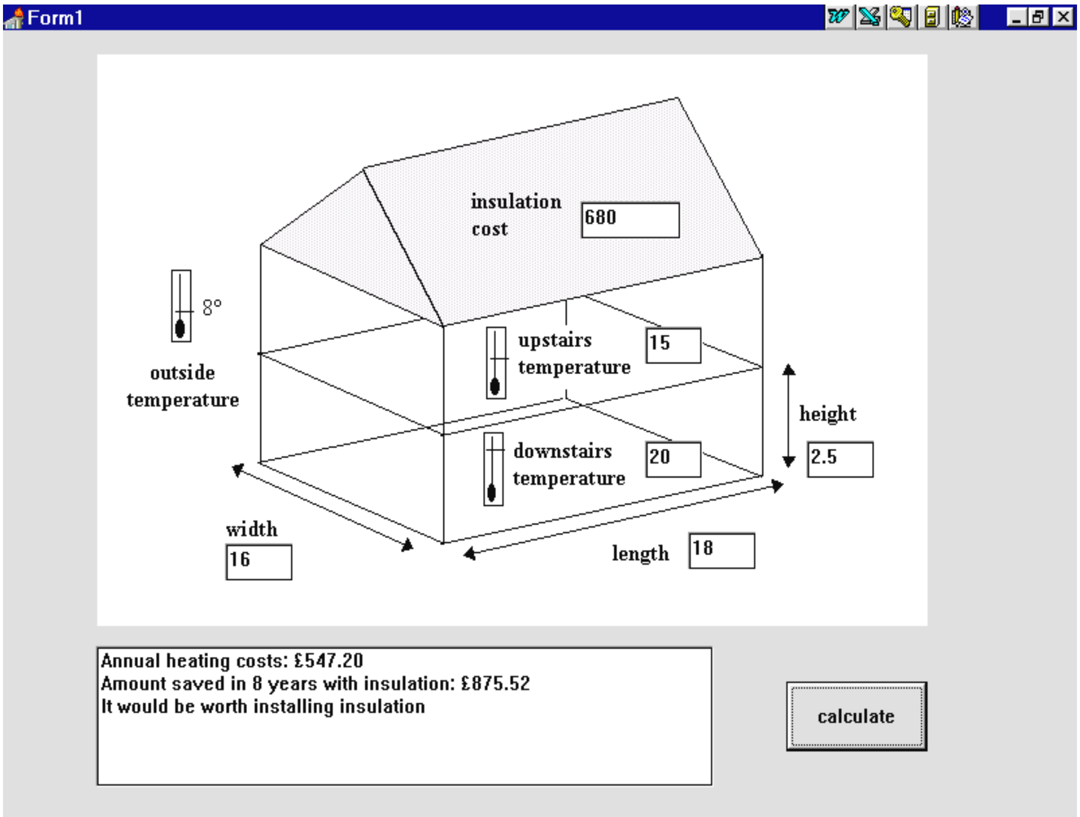
```
Annual heating costs: £547.20
Amount saved in 8 years with insulation: £875.52
```

The final stage in the program is to decide whether or not it would be economic to instal insulation. This is done in the procedure '**decide**'. Add the lines of program:

```
procedure TForm1.decision;
begin
  {decide if insulation should be installed}
  if saved>insulation then
    listbox1.items.add
        ('It would be worth installing insulation')
  else
    listbox1.items.add
        ('It would NOT be worth installing insulation');
end;
```

The procedure compares the amount which would be saved over the 8 year period with the cost of the insulation, and displays advice according to which is the larger figure.

Compile and run the completed program. Enter the test data and check that correct advice is given, then try the program for other sizes of house, temperatures, and insulation costs. It should be most worthwhile insulating the house if the occupants want to keep the rooms very warm, or if the insulation is cheap to install:



```
Annual heating costs: £547.20
Amount saved in 8 years with insulation: £875.52
It would be worth installing insulation
```

For our next project we will develop another simulation program. This will require some complicated programming, and is best broken down into a series of procedures as we did with the house heating costs program:

Application

Rabbit population model

A population of rabbits lives in burrows in a warren. The characteristics of this population are as follows:

- The rabbits form pairs and breed, each pair producing a litter each month throughout the year.
- Newborn rabbits are equally likely to be of either sex.
- A rabbit is able to start breeding so that it produces its first litter at the age of three months.

Some foxes live in a wood nearby and eat the rabbits. The rabbit population is also affected by seasonal factors.

A zoologist wishes to simulate the fluctuations in the rabbit population using a computer. The simulation is to be based on the following assumptions:

- The size of litters is in accordance with these probabilities

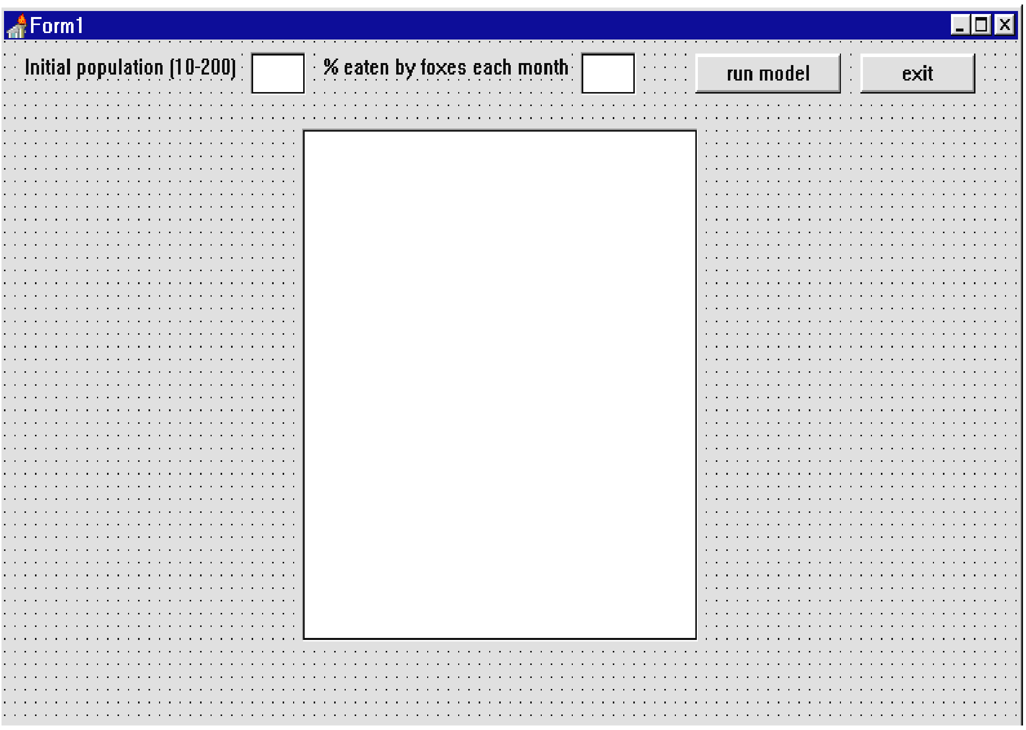
size	probability
7	0.1
6	0.2
5	0.4
4	0.2
3	0.1

- By the end of each month half of the rabbits are eaten by the foxes and a further one eighth of the rabbits also die from other causes.
- In each of the four winter months, a further eighth of the rabbits die of starvation.
- Any rabbit in the population is equally likely to be one of the casualties.

The zoologist wishes to investigate the effects of varying the size of the initial rabbit population, and varying the percentage of rabbits eaten by foxes if the amount of fox hunting in the area changes.

Begin the program by setting up a directory RABBITS and saving a Delphi project into it. Use the Object Inspector to Maximize the form, and drag the dotted grid to nearly fill the screen.

Before running the simulation it will be necessary to specify the initial population of rabbits (a suitable range is between 10 and 200), and the percentage of rabbits eaten each month by foxes.



Set up the screen display as shown. Place two small *Edit Boxes* at the top of the form with *labels* alongside - give these the captions '**Initial population (10-200)**' and '**% eaten by foxes each month**'.

Add two buttons with the captions '**run model**' and '**exit**', and a *List Box* in the middle of the form for text output.

Double-click the 'exit' button to create an event handler:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  halt;
end;
```

The initial population of adult rabbits can be stored as the variable '**adult**', and the percentage eaten by foxes as the variable '**eaten**'. Add these to the *Public declarations* section:

```
public
  { Public declarations }
  adult,eaten:integer;
 end;
```

The next step is to transfer the input value from the *Edit Box* to the variable '**adult**'. Double-click the Edit Box to produce an event handler and add the lines:

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

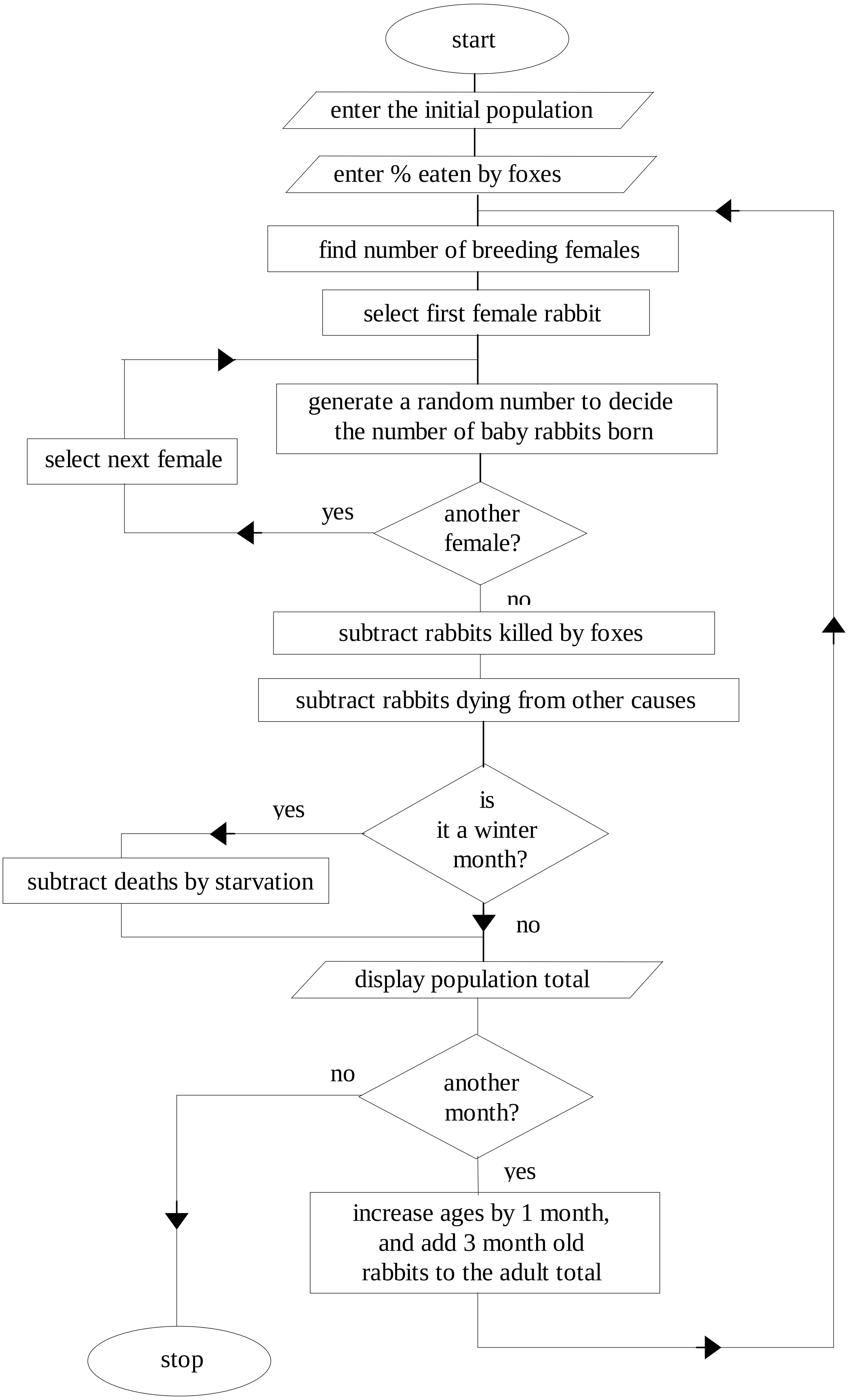
Produce a similar event handler for the '**% eaten**' edit box:

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

We can now turn our attention to the calculations to show changes in rabbit population over a period of time - five years would be suitable. A flowchart for the model is given on the next page. During each month of the simulation the stages in the calculation are:

1. Calculate the number of newborn rabbits this month
2. Subtract the rabbits killed by foxes
3. Subtract the rabbits dying from other causes
4. IF it is a winter month THEN subtract the rabbits dying from starvation
5. Increase the ages of rabbits by 1 month

Each of these steps can be written as a separate procedure, called from the event handler for the '**run model**' button. We will discuss the processing carried out by each of the procedures in detail later, but first we must set up the event handler.


start
enter the initial population
enter % eaten by foxes
find number of breeding females
select first female rabbit
generate a random number to decide the number of baby rabbits born
select next female
yes
another female?
no
subtract rabbits killed by foxes
subtract rabbits dying from other causes
yes
is it a winter month?
subtract deaths by starvation
no
display population total
no
another month?
yes
increase ages by 1 month, and add 3 month old rabbits to the adult total
stop

Double-click the '**run model**' button and add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  month:integer;
begin
  initialise;
  for month:=1 to 60 do
  begin
    births;
    foxdeaths;
    otherdeaths;
    if (month mod 12) <=3 then
       winterdeaths;
    older;
  end;
end;
```

This event handler will respond to the '**run model**' button by:

- carrying out a procedure '**initialise**' to set all variables to the correct values before the simulation starts
- operating a loop which repeats 60 times for each month of a five year simulation period
- carrying out a procedure '**births**' to calculate the number of newborn rabbits in the current month
- carrying out a procedure '**foxdeaths**' to subtract the number of rabbits eaten by foxes in the current month
- carrying out a procedure '**otherdeaths**' to subtract the number of rabbits dying from other causes in the current month
- checking whether it is a winter month - if so, it will carry out a procedure '**winterdeaths**' to subtract the number of rabbits dying from starvation.
- carrying out a procedure '**older**' to increase the age of all baby rabbits by one month - it is necessary to know when rabbits reach an age of three months because they then join the breeding population.

Let us examine in detail the line which checks whether it is a winter month:

if (month mod 12) <=3 then
winterdeaths;

The MOD function gives the remainder when division takes place, for example:

24 mod 5 = 4

because there is a remainder of 4 when 24 is divided by 5. Similarly:

30 mod 5 = 0

since there is no remainder in this case. 5 divides exactly into 30.

We will take the four winter months when the rabbits are most at risk from starvation to be: **January, February, March and December**.

- In the first year of the simulation, these will be months 1, 2, 3 and 12
- In the second year they will be months 13, 14, 15 and 24
- In the third year they will be months 25, 26, 27 and 36

and so on...

Notice that all these numbers give a remainder of 3 or less when divided by 12. For all other months, the remainder is greater than 3. This useful observation allows us to use the formula:

(month mod 12) <=3

to detect if a particular month number represents a winter month.

Go to the **type** section at the top of the program and add our procedures to the list:

```
          ...........
    procedure Button2Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure initialise;
    procedure births;
    procedure foxdeaths;
    procedure otherdeaths;
    procedure winterdeaths;
    procedure older;
  private
    { Private declarations }
         ............
```

We can now set up the procedure blocks. It is helpful to include comment lines to indicate the purpose of each procedure. Put the procedure blocks at the end of the program, just above the final '**end.**' line:

```
        ............
  procedure TForm1.initialise;
  begin
    {set the variables before simulation starts}
  end;

  procedure TForm1.births;
  begin
    {calculate newborn rabbits in the current
     month}
  end;
```

```
procedure TForm1.foxdeaths;
begin
  {subtract rabbits eaten by foxes in the
   current month}
end;
procedure TForm1.otherdeaths;
begin
  {subtract rabbits dying from other causes in
   current month}
end;

procedure TForm1.winterdeaths;
begin
  {subtract rabbits dying from starvation in
   a winter month}
end;

procedure TForm1.older;
begin
  {increase the age of baby rabbits by one
   month}
end;

end.
```

We can now start work on the individual procedures. The first to consider is '**initialise**'.

At the start of the program the user will input the initial number of adult rabbits. It will also be necessary to record the numbers of **newborn** rabbits, **1 month old** and **2 month old** rabbits which are not yet part of the breeding population. Add lines to the '**initialise**' procedure to set these to zero before the simulation begins:

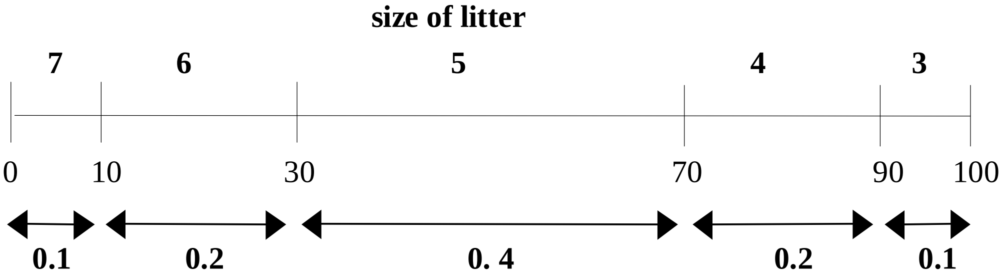
```
procedure TForm1.initialise;
begin
  {set the variables before simulation starts}
  newborn:=0;
  month1:=0;
  month2:=0;
end;
```

Add these variables to the *Public declarations*:

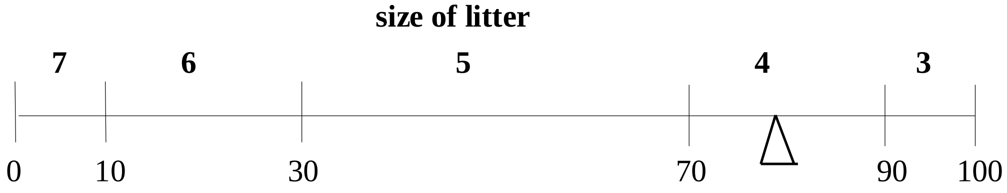
```
public
    { Public declarations }
    adult,newborn,month1,month2:integer;
    eaten:integer;
```

The next procedure is '**births**'. This should begin by finding the number of breeding females - we can take this to be half the number of adults, rounded to the nearest whole rabbit!

For each female, we need to find the number of baby rabbits born that month according to the probabilities given in the question. A *random number* and *number line* technique similar to the electricity simulation can again be used:



A number line from 0 to 100 is divided up in proportion to the probabilities of the different size of litter. A random number in the range 0-100 is then generated by the computer, and the position on the number line determines the litter size. For example, a random number of 78 gives a litter of 4 baby rabbits:



Add lines to the '**births**' procedure to calculate the number of newborn rabbits:

```
procedure TForm1.births;
var
  breeding,i,n:integer;
begin
  {calculate newborn rabbits in the current
   month}
  breeding:=round(adult/2);
  randomize;
  for i:=1 to breeding do
  begin
     n:=random(100);
     if n<10 then
       newborn:=newborn+7;
```

```
          if (n>=10) and (n<30) then
            newborn:=newborn+6;
          if (n>=30) and (n<70) then
            newborn:=newborn+5;
          if (n>=70) and (n<90) then
            newborn:=newborn+4;
          if n>=90 then
            newborn:=newborn+3;
       end;
     end;
```

Let's look at how this works...

We calculate the number of breeding females as half the adult population.
breeding:=round(adult/2);

We then begin a loop which will repeat for each of the breeding females:
for i:=1 to breeding do

To find the litter size for the first female, a random number is produced:
n:=random(100);

If the random number is less than 10, this represents a litter of 7 baby rabbits as shown on the number line above. This number is added to the total of baby rabbits born so far this month:

```
if n<10 then
  newborn:=newborn+7;
```

Similar lines of program handle the other number ranges. For example, a random number between 10 and 30 would indicate a litter of 6 baby rabbits:

```
if (n>=10) and (n<30) then
  newborn:=newborn+6;
```

The next procedure is '**foxdeaths**'. This will reduce the numbers of rabbits in each age category according to the percentage eaten by foxes. Add the lines:

```
procedure TForm1.foxdeaths;
begin
  {subtract rabbits eaten by foxes in the current
   month}
  adult:=adult-round(adult*eaten/100);
  month2:=month2-round(month2*eaten/100);
  month1:=month1-round(month1*eaten/100);
  newborn:=newborn-round(newborn*eaten/100);
end;
```

The '**otherdeaths**' procedure will be similar, but in this case the number of rabbits in each age category is reduced by one eighth. Add the lines:

```
procedure TForm1.otherdeaths;
begin
  {subtract rabbits dying from other causes in
   current month}
  adult:=adult-round(adult/8);
  month2:=month2-round(month2/8);
  month1:=month1-round(month1/8);
  newborn:=newborn-round(newborn/8);
end;
```

The '**winterdeaths**' procedure again reduces the number of rabbits in each age category by one eighth. **Edit/Copy/Paste** could be used to copy the lines from 'otherdeaths' into the 'winterdeaths' procedure:

```
procedure TForm1.winterdeaths;
begin
  {subtract rabbits dying from starvation in
   a winter month}
  adult:=adult-round(adult/8);
  month2:=month2-round(month2/8);
  month1:=month1-round(month1/8);
  newborn:=newborn-round(newborn/8);
end;
```

The fial stage is to write the 'older' procedure which promotes each group of rabbits to the next age category:

- 2 month old rabbits join the adult breeding population
- 1 month old rabbits become 2 months old
- newborn rabbits become 1 month old.

Add the lines of program to do this:

```
procedure TForm1.older;
begin
  {increase the age of baby rabbits by one month}
   adult:=adult+month2;
   month2:=month1;
   month1:=newborn;
end;
```

The processing procedures for the simulation are now completed, but there will not yet be any screen output when the program runs. We can add lines to the 'run model' button click procedure to write comments in the List Box in a similar way to the House heating costs program.

Add lines to the 'run model' button click procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  month,total:integer;
  textline:string;
begin
  initialise;
  listbox1.clear;
  adult:=strtoint(edit1.text);
  for month:=1 to 60 do
  begin
    births;
    foxdeaths;
    otherdeaths;
    if (month mod 12) <=3 then
       winterdeaths;
    textline:='Month '+inttostr(month);
    listbox1.items.add(textline);
    total:=adult+month2+month1+newborn;
    if total<0 then
      month:=60
    else
    begin
      textline:='Total population: ' +
                              inttostr(total);
      listbox1.items.add(textline);
    end;
    listbox1.items.add(' ');
    older;
  end;
end;
```

The purpose of the lines:

listbox1.clear;
adult:=strtoint(edit1.text);

is to blank out the List Box and ensure that a correct starting value for 'adult' is obtained from the Edit Box. These lines are needed if the simulation is run more than once.

The number of the current month is displayed by the instructions:

textline:='Month '+inttostr(month);
listbox1.items.add(textline);

The total number of rabbits is found by adding the different age groups:

```
total:=adult+month2+month1+newborn;
```

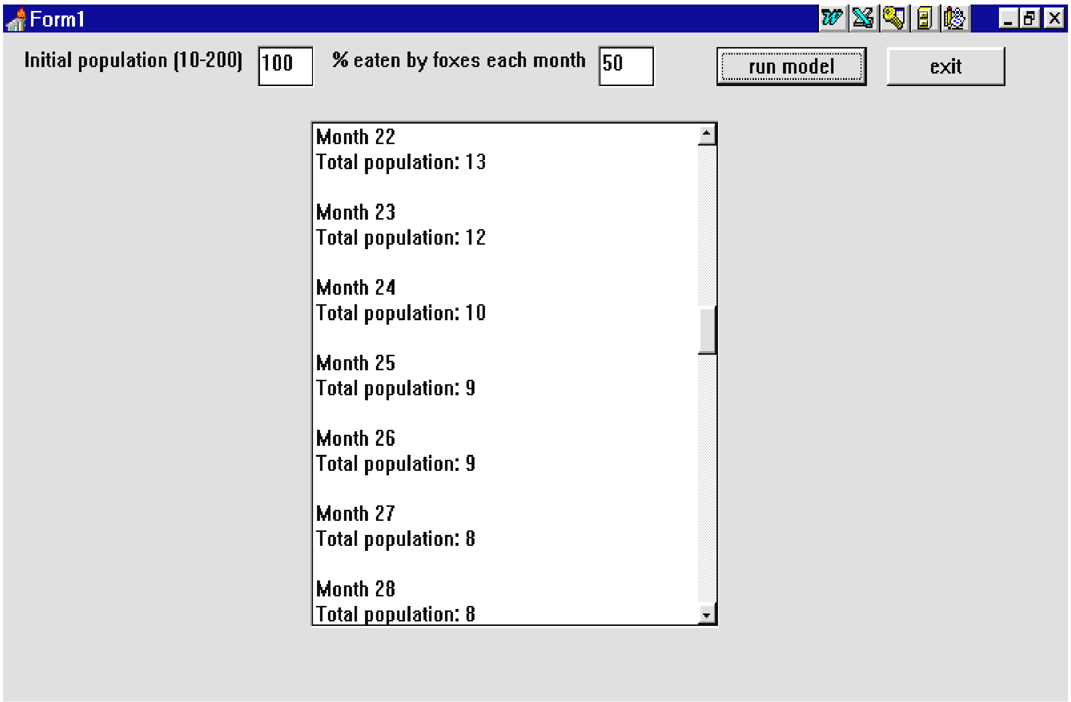
It is possible that the population will grow so large that the total cannot be displayed correctly as an integer value - the integer goes out of range and takes on an incorrect negative value. The purpose of the next IF.. condition is to stop the simulation if this happens:

```
if total<0 then
   month:=60
else
begin
    textline:='Total population: ' +  inttostr(total);
    listbox1.items.add(textline);
end;
```

By setting the loop counter **'month'** to 60, the computer will think that all 60 months of the simulation have been completed and the loop will end.

If the population total has stayed in range, the value will be displayed in the List Box.

The time has now come to compile and try out the simulation. Run the program with an initial population of **100** adult rabbits, and set the percentage eaten by foxes to **50%**. Press the 'run model' button:



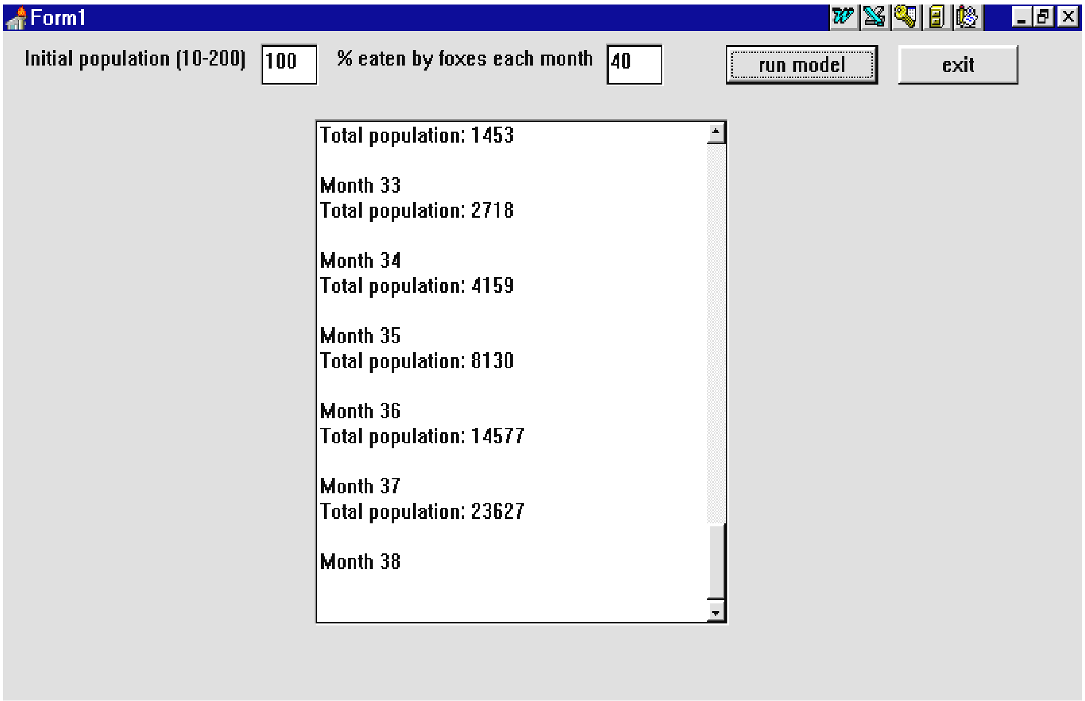
100

50

```
Month 22
Total population: 13

Month 23
Total population: 12

Month 24
Total population: 10

Month 25
Total population: 9

Month 26
Total population: 9

Month 27
Total population: 8

Month 28
Total population: 8
```

In this run of the model the population decreased, becoming stable in month 27 with only 8 rabbits surviving.

Run the model again with the same starting population of 100 adults, but this time set the percentage eaten to **40%**:



```
Total population: 1453

Month 33
Total population: 2718

Month 34
Total population: 4159

Month 35
Total population: 8130

Month 36
Total population: 14577

Month 37
Total population: 23627

Month 38
```

This time the population increased, reaching over 23,000 by month 37 when the total went out of range and the loop stopped.

Keep the starting population at 100 and vary the percentage eaten by foxes until you find a % value which keeps the population as stable and close to 100 as possible.

Although the model is giving correct results, these are difficult to interpret in text form - as in the case of the electricity simulation, a graph would be better.

Exit from the program and go back to the Delphi editing screen. Delete the List Box from the Form and replace it with an *Image Box*. Use the Object Inspector to set the **Width** to 640 and the **Height** to 480.

Return to the program listing and delete all lines from the '**run model**' button click procedure which begin: **listbox1. . . -** the listbox has been deleted, so these would cause an error.
NOTE: An easy way to do this is to use the **Find** option on the **Search** menu at the top of the Delphi editing screen.

Add two new procedures: '**axes**' which will plot the axes for a linegraph, and '**plotpoint**' which will draw the graph line. After deleting the instructions which are no longer needed, the '**run model**' button click procedure becomes:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  month,total:integer;
begin
  initialise;
  axes;
  adult:=strtoint(edit1.text);
  for month:=1 to 60 do
  begin
    births;
    foxdeaths;
    otherdeaths;
    if (month mod 12) <=3 then
       winterdeaths;
    total:=adult+month2+month1+newborn;
    if total<0 then
      month:=60
    else
      plotpoint(month,total);
    older;
  end;
end;
```

Add the new procedures to the list under the type heading at the top of the program:

```
        ....
    procedure winterdeaths;
    procedure older;
    procedure axes;
    procedure plotpoint(month,total:integer);
```

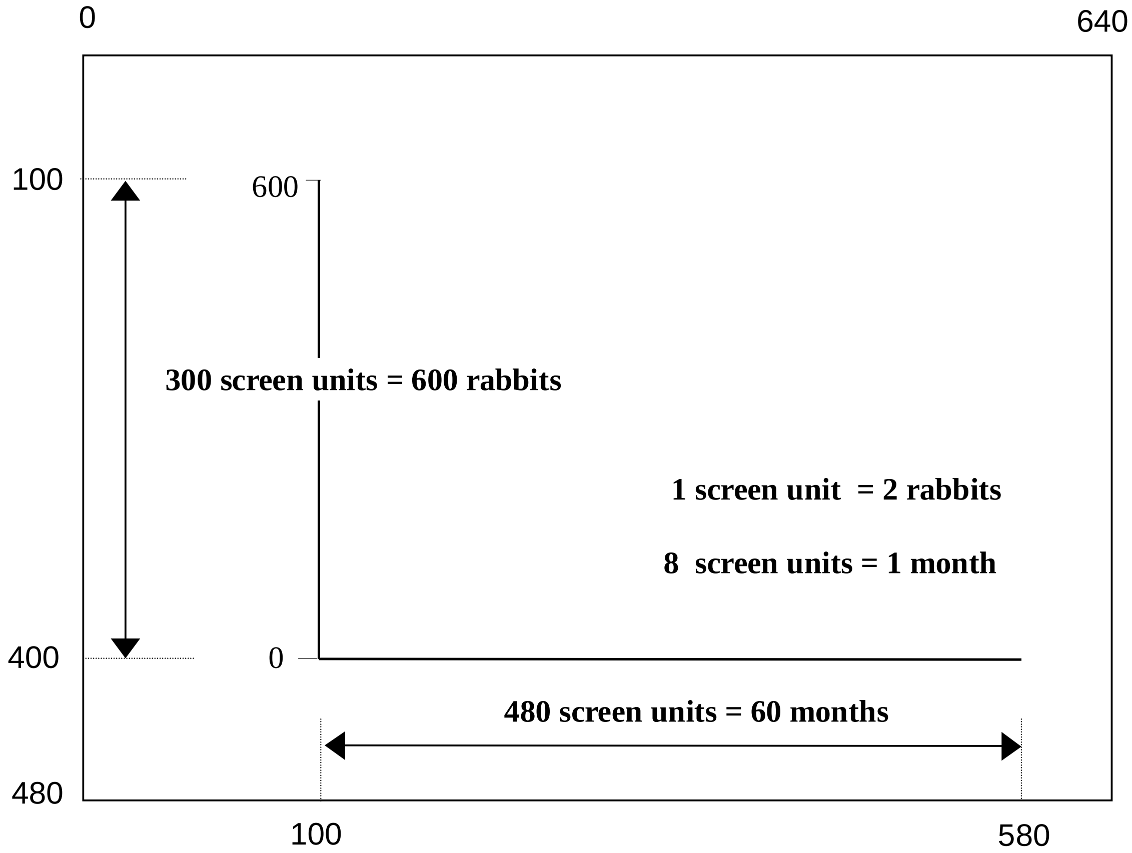
Notice that the heading for the procedure has the two variables 'month' and 'total' shown in brackets:

procedure TForm1.plotpoint(month,total:integer);

This is done whenever a procedure needs to make use of variables from elsewhere in the program which have **not** been listed under the *Public declarations* heading. The variables shown in a procedure heading are known as '*input parameters*'. This topic will be discussed in detail in a later chapter of the course.

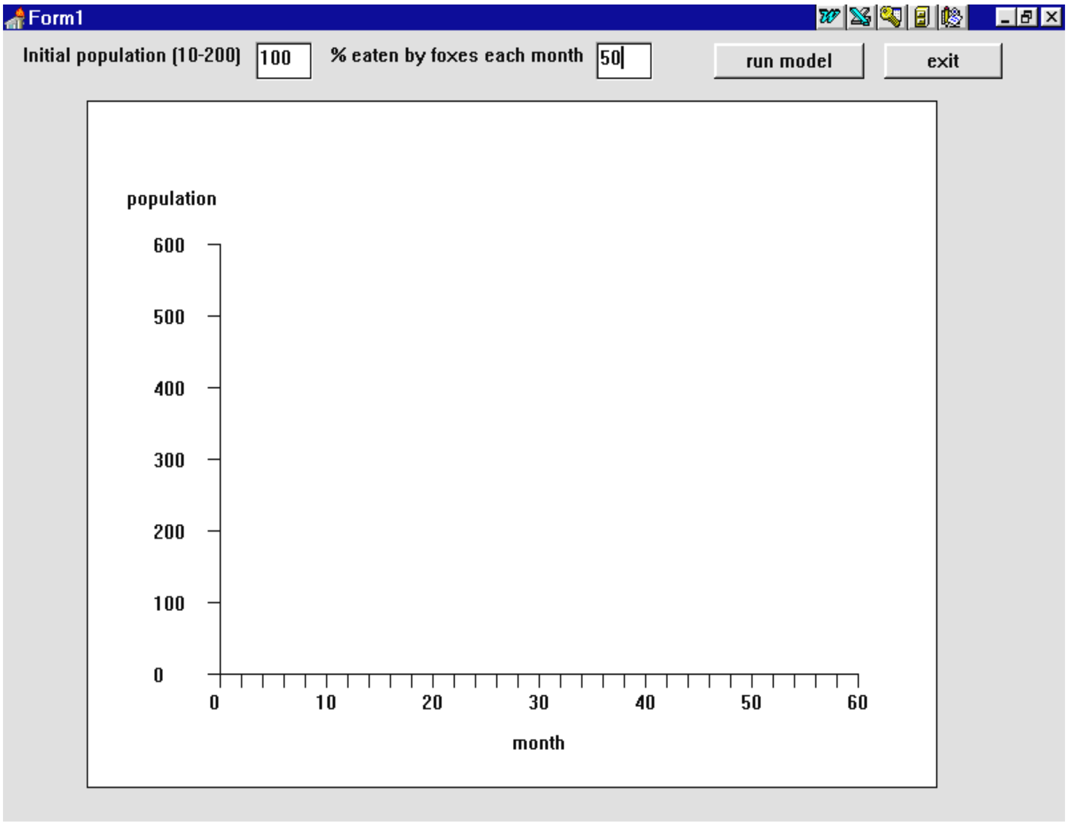
Put in the procedure blocks:

```
procedure TForm1.axes;
begin
  {draw the graph axes}
end;

procedure TForm1.plotpoint(month,total:integer);
begin
  {plot a point for the graph line}
end;

end.
```


0
640
100
600
300 screen units = 600 rabbits
1 screen unit = 2 rabbits
8 screen units = 1 month
400
0
480 screen units = 60 months
480
100
580

We will scale the graph so that the 60 months of the simulation are shown along the horizontal axis, and the vertical axis gives a range from 0 - 600 rabbits. The **scale factors** will be: 1 screen unit vertically = 2 rabbits, and 8 screen units horizontally = 1 month.

Add the lines of program to draw the axes and add the graduations:



Initial population (10-200) 100

% eaten by foxes each month 50

```
procedure TForm1.axes;
var
  x,y:integer;
  textline:string;
begin
  {draw the graph axes}
  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 y:=0 to 6 do
  begin
    image1.canvas.moveto(100,400-y*50);
    image1.canvas.lineto(90,400-y*50);
    textline:=inttostr(y*100);
    image1.canvas.textout(50,393-y*50,textline);
  end;
  image1.canvas.textout(30,60,'population');
  for x:=0 to 30 do
  begin
    image1.canvas.moveto(100+x*16,400);
    image1.canvas.lineto(100+x*16,410);
    if x mod 5 = 0 then
    begin
      textline:=inttostr(x*2);
      image1.canvas.textout(92+x*16,412,textline);
    end;
  end;
  image1.canvas.textout(320,440,'month');
end;
```

The loop beginning:

for y:=0 to 6 do

draws graduation lines on the vertical axis and numbers these 0, 100, 200 ...

The loop counter variable **y** which runs from 0 to 6 is multiplied by 100 to produce the figures 0 - 600 for the graduations:

```
textline:=inttostr(y*100);
image1.canvas.textout(50,393-y*50,textline);
```

The second loop beginning:

for x:=0 to 30 do

puts graduation lines along the horizontal axis at 2 month intervals. We use a MOD function:

```
if x mod 5 = 0 then
begin
   textline:=inttostr(x*2);
   image1.canvas.textout(92+x*16,412,textline);
end;
```

to control the numbers which are displayed. The IF... condition only operates if the loop counter **x** divides exactly by 5. That is in the cases when:

x = 0, 5, 10, 15, etc.

These figures are multiplied by 2 then written on the screen as the month numbers 0, 10, 20, 30 ....

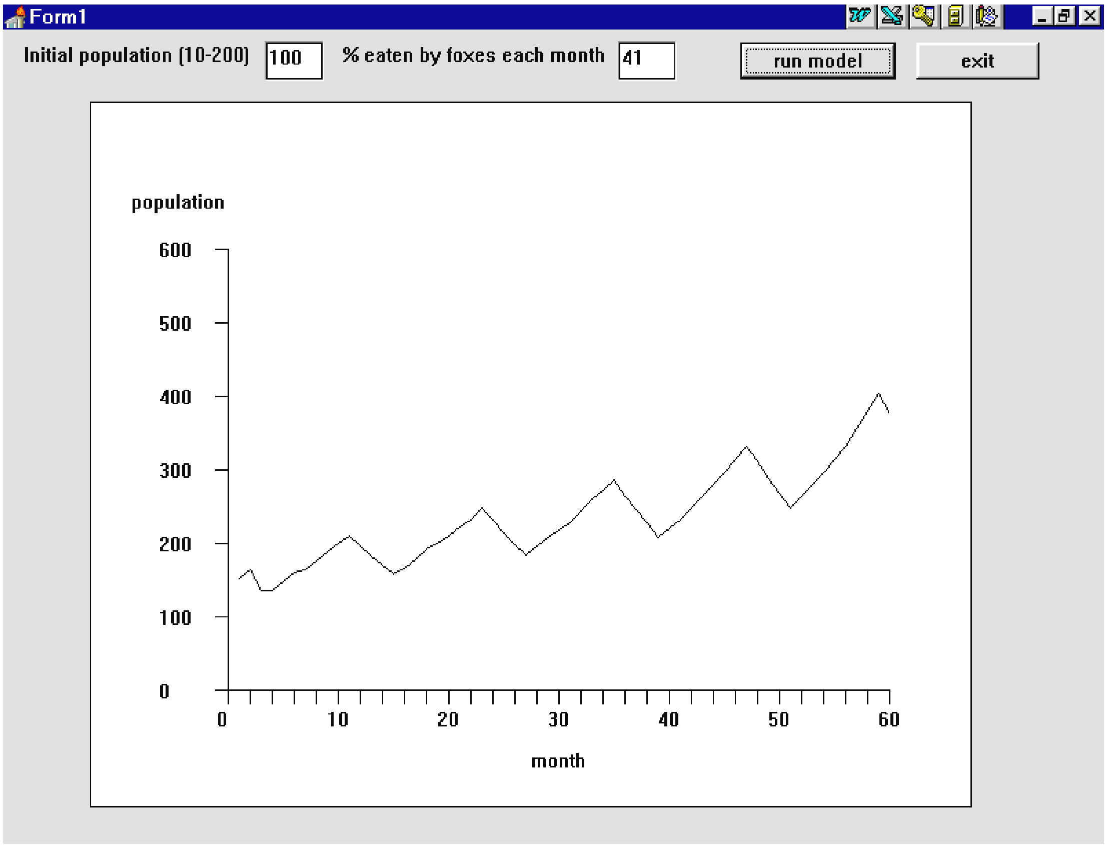
We now only need to finish the '**plotpoint**' procedure. Add the lines of program:

```
procedure TForm1.plotpoint(month,total:integer);
begin
  {plot a point for the graph line}
  if month=1 then
    image1.canvas.moveto(100+month*8,
                            400-round(total/2))
  else
    image1.canvas.lineto(100+month*8,
                            400-round(total/2));
end;
```

The procedure contains an IF.. ELSE.. conditional block:

```
if month=1 then
  image1.canvas.moveto(100+month*8,
                 400-round(total/2))
else
  image1.canvas.lineto(100+month*8,
                 400-round(total/2));
```

For **month 1**, the computer needs to move to the starting point of the graph without drawing a line. For each of the other months a line is drawn to the current graph point.



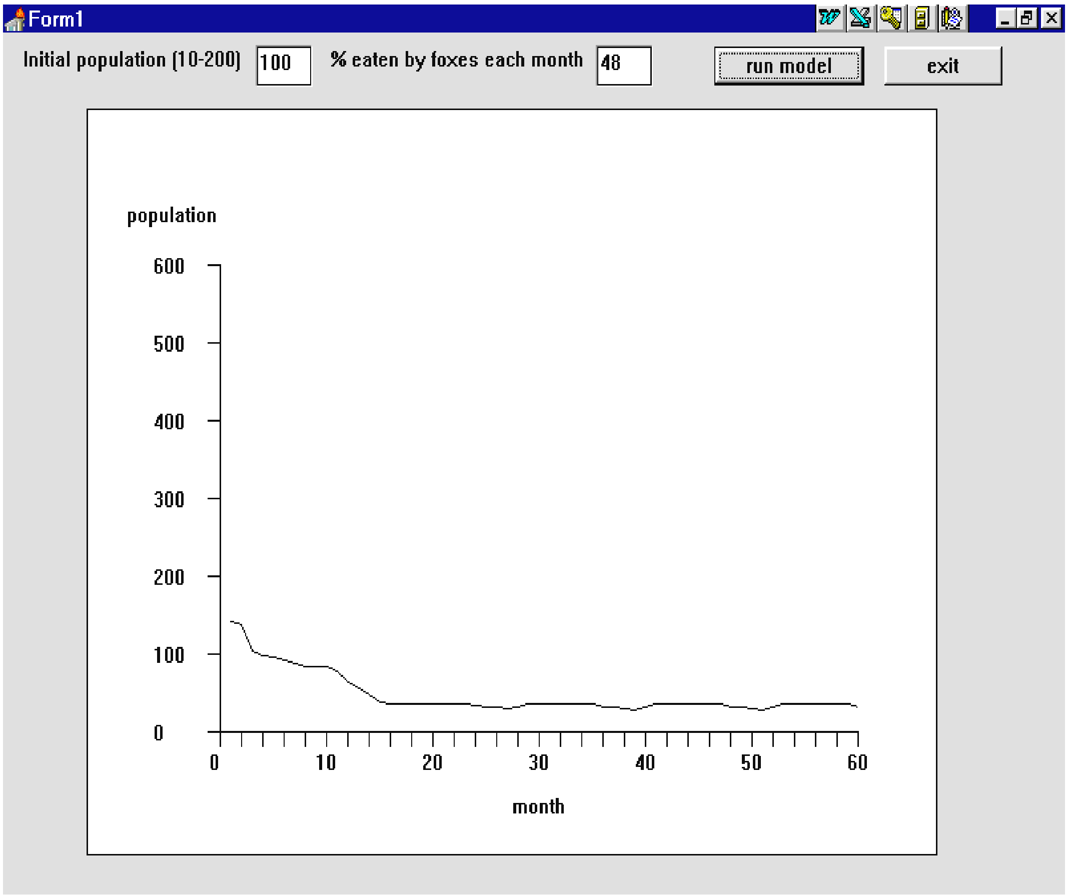
Initial population (10-200) 100

% eaten by foxes each month 41

Compile and run the program. Enter **100** as the initial population, and **41%** eaten by foxes. Press the 'run model' button and a graph similar to the one below should be drawn:

The graph line is generally rising, indicating a long term increase in the rabbit population. The 'saw tooth' pattern is due to the increased death rate in the winter months which temporarily reduces the population.

Now try **48%** eaten by foxes:



Initial population (10-200) 100

% eaten by foxes each month 48

The pattern changes. The population now shows a decline to a low number of rabbits surviving.

Experiment with different percentages eaten to find a value which keeps the population as stable and close to 100 as possible.

Now try varying the initial population in the range 10 - 200 rabbits. Does this affect the percentage eaten by foxes for a stable population?