ELEVEN

Simulation programs

An increasingly important use of computers is to carry out simulations. These might be: to predict changes in population which could affect the economy of a country, to predict whether a new factory is likely to be profitable, or to predict the effect on traffic if a new road is built. Use of computer simulations can help avoid costly mistakes being made.

In this chapter we will carry out several computer simulations. This will allow us to practice many of the programming techniques from earlier in the course, and will also introduce some new ideas about **probability**.

Supermarket checkout simulation

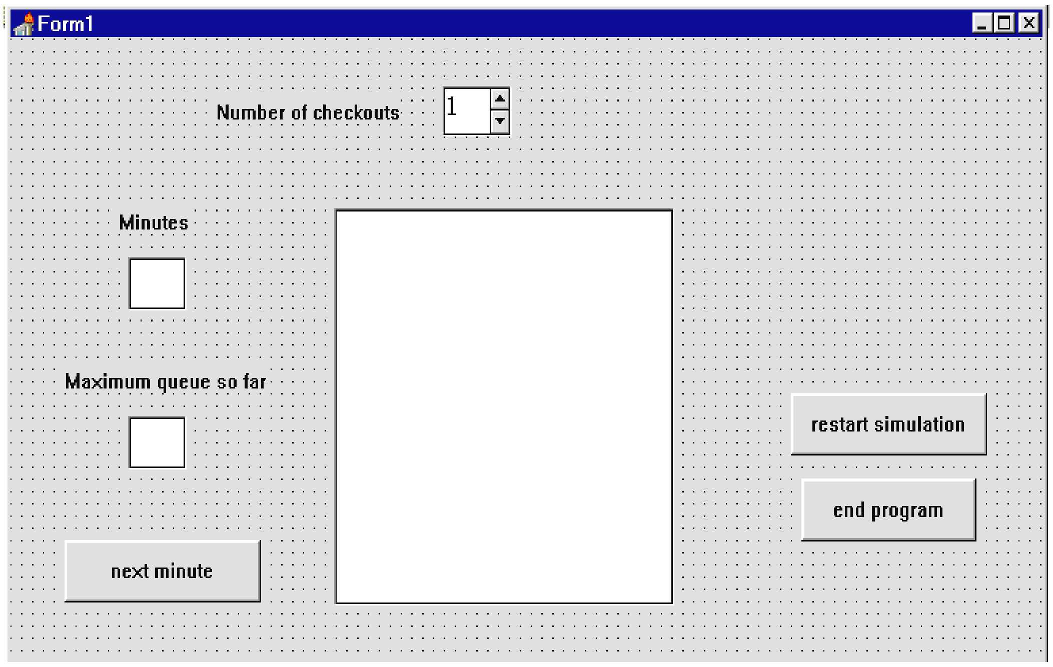
A large supermarket in a city is planning to provide a number of fast checkouts for customers who are only purchasing a couple of items.

It is expected that a customer can be served at a fast checkout in one minute. Customers arriving will wait in a single queue, and move to a checkout as soon as one becomes available.

A survey is carried out to find the current number of customers with only a couple of items to purchase; these customers would benefit from the introduction of the new fast checkout system. It is found that the number varies randomly between 0 and 4 customers per minute.

When the new system is introduced, the managers do not want the fast service queue to exceed 3 customers at any time. You are asked to write a simulation program which could determine the least number of checkouts which would be needed to meet this objective.

Start the program by setting up a new directory CHECKOUT 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.



Place components on the form as shown above. At the top is a *spin edit*, with a *label* alongside captioned '**Number of checkouts**'. Set the **MaxValue** of the *spin edit* to 6, and the **MinValue** to 1.

Put a *List Box* component onto the grid below the *spin edit*. The *List Box* is selected from the **Standard** component menu.

Place two *Edit Boxes* to the left of the *List Box*. Above these add *labels* with the captions '**Minutes**' and '**Maximum queue so far**'.

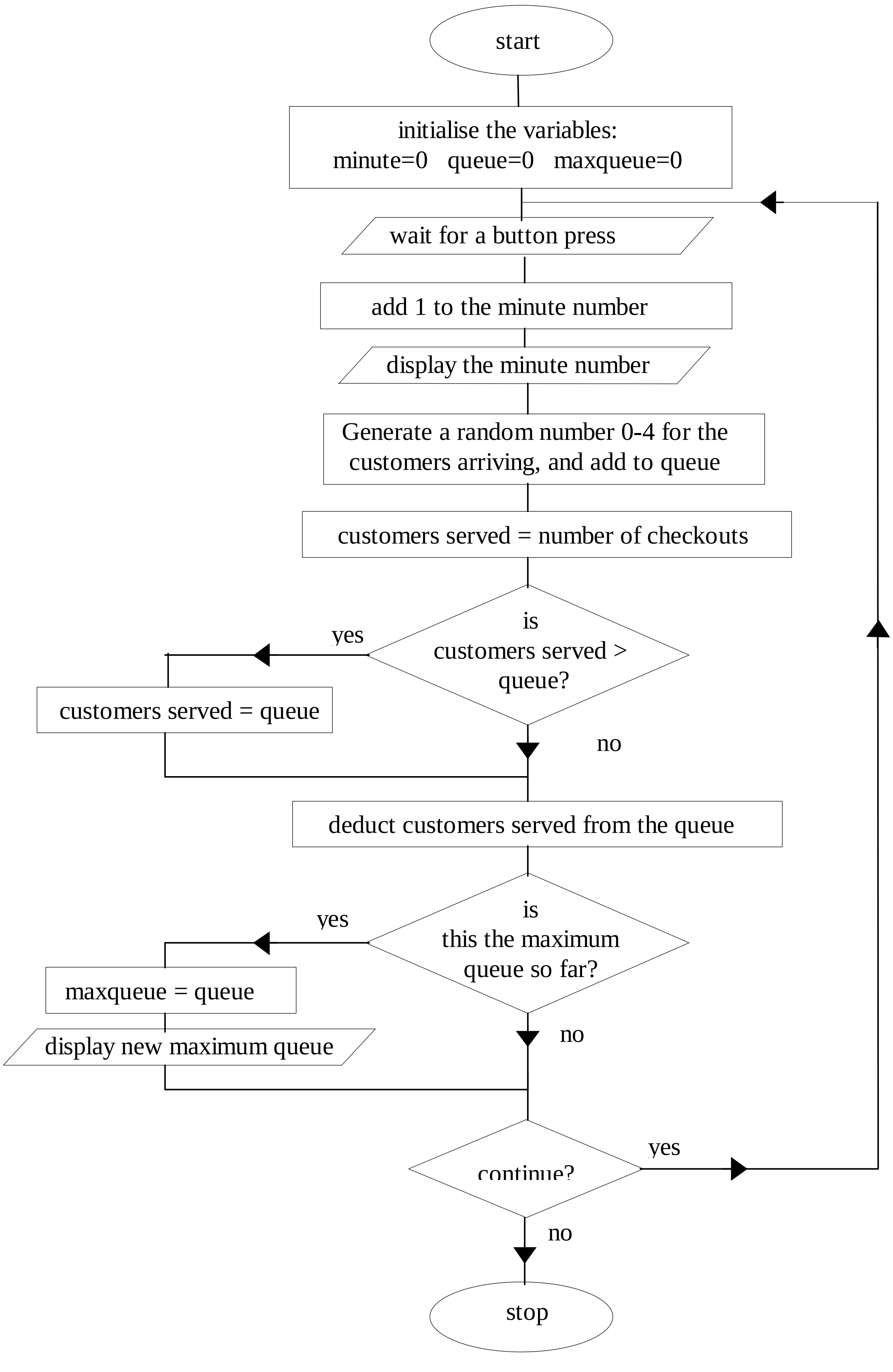
Complete the form by adding three *buttons* with the captions '**next minute**', '**restart simulation**', and '**end program**'.

Double-click the 'end program' button and add the '**halt**' command to the event handler:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  halt;
end;
```

Compile and run the program to check that the *spin edit* operates correctly in the range 1 - 6, and that the program returns to the Delphi editing screen when the '**end program**' button is pressed.

A flowchart will help illustrate the processing to be carried out:


start
initialise the variables:
minute=0 queue=0 maxqueue=0
wait for a button press
add 1 to the minute number
display the minute number
Generate a random number 0-4 for the customers arriving, and add to queue
customers served = number of checkouts
is customers served > queue?
yes
customers served = queue
no
deduct customers served from the queue
is this the maximum queue so far?
yes
maxqueue = queue
display new maximum queue
no
continue?
yes
no
stop

When the simulation is running, we will need variables to record the number of the current **minute**, the length of the current **queue**, the **maximum** length of queue so far, and the number of **checkouts** simulated. Add these to the *Public declarations* section:

```
public
  { Public declarations }
  queue,maxqueue:integer;
  checkouts:integer;
  minute:integer;
end;
```

At the start of the simulation we will initialise **minute**, **queue** and **maxqueue** to zero. Double-click the dotted grid of the form to produce an **OnCreate** procedure, then add lines of program to carry out the initialisation:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  queue:=0;
  maxqueue:=0;
  minute:=0;
  randomize;
end;
```

The simulation will involve the use of random numbers, so it is convenient to set the random number generator using the '**randomize**' command at this point.

Most of the processing occurs when the 'next minute' button is pressed. Double click this button to create an event handler, then add the lines of program below:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  arriving,served:integer;
  textline:string;
begin
  checkouts:=spinedit1.value;
  minute:=minute+1;
  textline:='Minute '+inttostr(minute);
  listbox1.items.add(textline);
  edit1.text:=inttostr(minute);
  arriving:=random(5);
  textline:='Customers arriving: '
                        +inttostr(arriving);
  listbox1.items.add(textline);
  queue:=queue+arriving;
```

```
    served:=checkouts;
    if (served>queue) then
      served:=queue;
    textline:='Customers served: '
                                +inttostr(served);
    listbox1.items.add(textline);
    queue:=queue-served;
    textline:='Current queue: '+inttostr(queue);
    listbox1.items.add(textline);
    if queue>maxqueue then
    begin
      maxqueue:=queue;
      edit2.text:=inttostr(maxqueue);
    end;
    listbox1.items.add('');
  end;
```

This procedure will carry out the main loop shown on the flow chart. This involves finding the number of customers arriving during the current minute, the number who are served at the checkouts, and the new length of the queue. To help us follow what is happening in the simulation, each piece of information will be displayed in the *List Box* as soon as it is calculated.

The procedure begins by reading the value in the spin edit box to find the number of checkouts for the simulation:

checkouts:=spinedit1.value;

We add 1 to the minute number. This is displayed as a line of text in the *List Box*:

minute:=minute+1;
textline:='Minute '+inttostr(minute);
listbox1.items.add(textline);

The minute number is also displayed in the *Edit Box* on the left of the screen:

edit1.text:=inttostr(minute);

A random number is generated to represent the number of customers arriving, and this is displayed in the *List Box*:

arriving:=random(5);
textline:='Customers arriving: ' + inttostr(arriving);
listbox1.items.add(textline);

The customers who have arrived are added to the queue:

queue:=queue+arriving;

The maximum number of customers who could be served during the current minute is equal to the number of checkouts:

```
served:=checkouts;
```

If the maximum number of customers who could be served is greater than the current queue length, then only the number queueing would actually be served:

```
if (served>queue) then
   served:=queue;
```

The number of customers served is displayed in the List Box:

```
textline:='Customers served: ' + inttostr(served);
listbox1.items.add(textline);
```

The number of customers served is deducted from the queue, and the new length of the queue is shown in the *List Box*:

```
queue:=queue-served;
textline:='Current queue: '+inttostr(queue);
listbox1.items.add(textline);
```

If the new length of queue is the largest so far, this is recorded as the new maximum and displayed in the *Edit Box* at the left of the screen:

```
if queue>maxqueue then
begin
   maxqueue:=queue;
   edit2.text:=inttostr(maxqueue);
end;
```

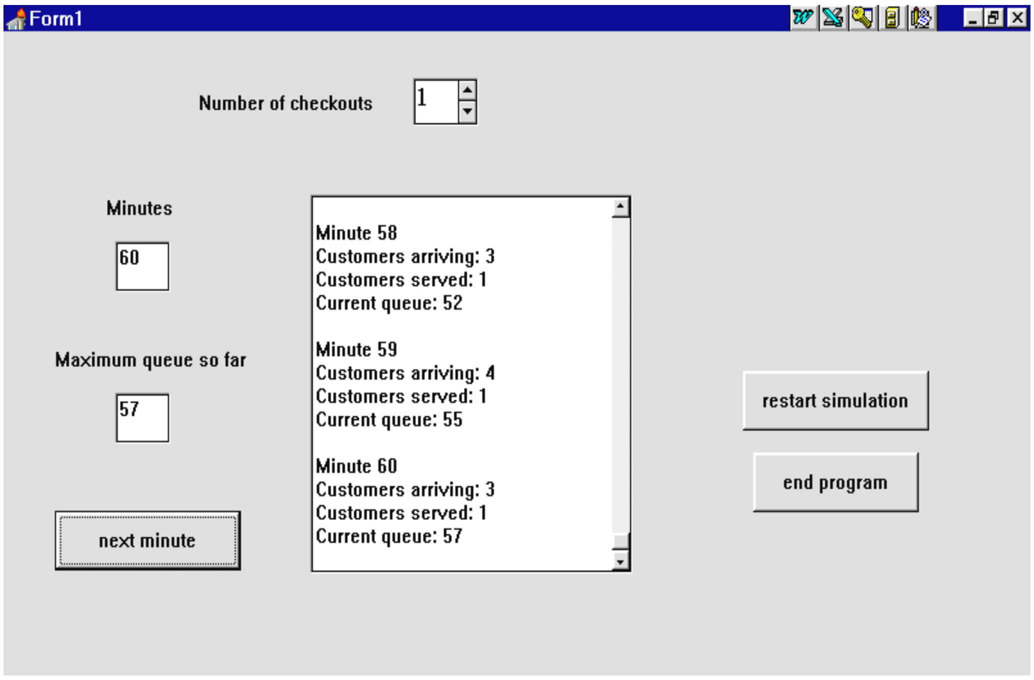
Compile and run the simulation. Set the number of checkouts to be 1, then press the '**next minute**' button a couple of times. Examine the information in the *List Box* and convince yourself that the program is calculating the current queue and maximum queue correctly. Press the '**end program**' button to return to the Delphi editing screen.

The only task remaining is to create an event handler for the '**restart simulation**' button. This needs to set the variables back to zero and blank out the *List Box* and *Edit Boxes*, ready for the next run of the simulation. Double-click the '**restart simulation**' button and add program lines to the procedure as shown below:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  listbox1.clear;
  edit1.clear;
  edit2.clear;
  minute:=0;
  queue:=0;
  maxqueue:=0;
end;
```

We are now ready to use the simulation program to solve the problem - how many checkouts will be needed to be sure the queue never exceeds 3 people?

Run the program first with 1 checkout selected. Keep clicking the '**next minute**' button until 60 minutes have been simulated, then make a note of the maximum queue length ...



```
Minute 58
Customers arriving: 3
Customers served: 1
Current queue: 52

Minute 59
Customers arriving: 4
Customers served: 1
Current queue: 55

Minute 60
Customers arriving: 3
Customers served: 1
Current queue: 57
```

- this result would not be acceptable to the supermarket manager!

Press the '**restart simulation**' button and try again with 2 checkouts. Continue if necessary until you find the minimum number of checkouts needed to ensure the queue is no longer than 3 customers at any time.

Application

Electricity usage simulation

A new estate of twenty houses is to be built, and the Electricity Board wish to estimate the amount of electricity which would be required by the houses at different times of the day and night. Imagine that you are a computer software designer given the task of writing a program to simulate the electricity usage.

From a study of similar houses on another estate, you obtain the following data:

- The average number of electric lights in each house is 8. The power of each light may be taken as 60 watts.
- The average number of domestic appliances in each house is 12. The average power rating of the domestic appliances is 500 watts.

You are asked to show the total number of watts of electricity required by the whole housing estate hourly for a 24-hour period. You decide to divide the period into three time zones:

day	- 6 a.m. to 5 p.m.
evening	- 5 p.m. to 12 p.m.
night	- 12 p.m. - 6 a.m.

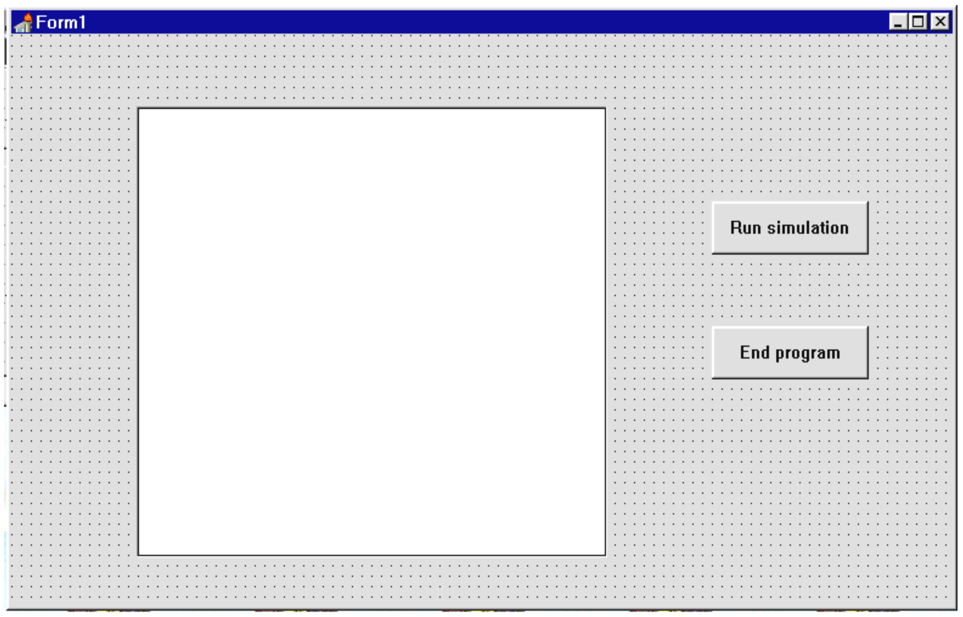
During the **day**, you find that there is a 20% change that each light is switched on, and a 20% chance that each domestic appliance is switched on.

During the **evening**, there is a 60% chance that each light is switched on, and a 40% chance that each domestic appliance is switched on.

During the **night**, there is a 10% chance that each light is switched on, and a 5% chance that each domestic appliance is switched on.

Start the program by setting up a new directory ELECTRIC 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.

Place a *List Box* on the dotted grid as shown below. Add two *buttons* with the captions '**Run simulation**' and '**End program**'.



Now we can turn our attention to an algorithm for the simulation. In the case of complex problems like this, an **algorithm progressive refinement sequence** is the best approach. We can begin by summarising the objectives of the program:

```
1.  LOOP for hour = 0 to 23
2.       calculate the amount of electricity used by the housing estate
3.  END LOOP
```

We are just stating that an electricity usage figure is to be calculated for each hour of the day.

Step 2 can be expanded to show how the usage will be found:

```
1.  LOOP for hour = 0 to 23
2.1      set the electricity total to zero
2.2      LOOP for house = 1 to 20
2.3             calculate the amount of electricity used by the house and add
                this to the total
2.4      END LOOP
2.5      display the total electricity used this hour
3.  END LOOP
```

A loop can be used to add the electricity usage for each of the twenty houses to find the total for the hour.

The next step is to show in more detail how the usage for a particular house will be calculated:

```
1.  LOOP for hour = 0 to 23
2.1     set the electricity total to zero
2.2     LOOP for house = 1 to 20
2.3.1       LOOP for lightbulb = 1 to 8
2.3.2           IF this lightbulb is switched on THEN
2.3.3               add 60 watts to the electricity total
2.3.4       END LOOP
2.3.5       LOOP for appliance = 1 to 12
2.3.6           IF this appliance is switched on THEN
2.3.7               add 500 watts to the electricity total
2.3.8       END LOOP
2.4     END LOOP
2.5     display the total electricity used this hour
3.  END LOOP
```

Two more loops have been introduced to check the light bulbs and electrical appliances in the house - if they are switched on, the appropriate number of watts are added to the electricity total.

Let's begin constructing an event handler procedure for the '**Run simulation**' button to implement this algorithm. Double-click the button and add the lines of program:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  hour,house,light,appliance:integer;
  total:real;
begin
  for hour:=0 to 23 do
  begin
    total:=0;
    for house:=1 to 20 do
    begin
      for light:=1 to 8 do
      begin
         {check if this light is on}
      end;
      for appliance:=1 to 12 do
      begin
         {check if this appliance is on}
      end;
    end;
  end;
end;
```

Notice the way the procedure has the same loop structures shown in the algorithm design. Each loop in the procedure starts with a 'BEGIN' command and finishes with an 'END' command.

It is easy to make an error over the number of BEGIN's or END's when writing a complicated program. To reduce the chances of a mistake, programmers normally indent all the lines following a BEGIN command by about three characters, then unindent by three characters when the corresponding END is reached. The structure is shown clearly in a block analysis of the procedure:

```
for hour:=0 to 23 do
begin
  total:=0;
  for house:=1 to 20 do
  begin
    for light:=1 to 8 do
    begin
       {check if this light is on}
    end;

    for appliance:=1 to 12 do
    begin
       {check if this appliance is on}
    end;
  end;
end;
```

Notice the use of 'comment lines' in the program:

```
{check if this light is on}
{check if this appliance is on}
```

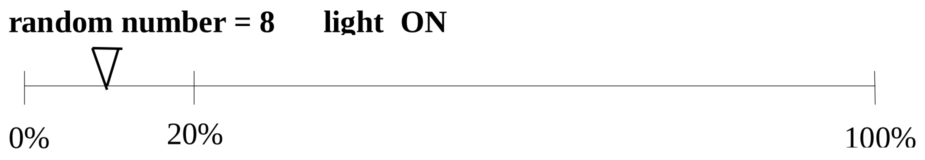
You are allowed to add any notes you wish to a program listing - provided they are enclosed in curly brackets, they will be ignored by the computer when the program is run.

Let's now look in more detail at the loop where each light bulb is checked. The probability that a light bulb is switched on will depend on the time zone:

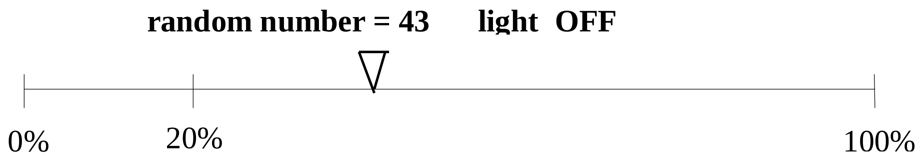
day - 20% **evening - 60%** **night - 10%**

The program can generate a random number in the range 0-100 to represent a position on a number line.

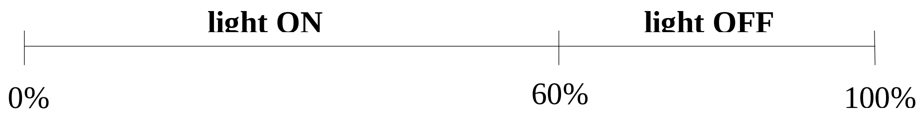
In the case of the daytime period, there is a 20% chance of any particular light bulb being on. If the random number is found to be in the range 0-20, we could take this to mean that the light is on, e.g.:



If , however, the random number turns out to be greater than 20, we take this to mean the light is off, e.g.:



The position of the cut-off point on the number line will depend on the percentage probability during the different time zones. In the evening this is 60%:



A similar method can be used to determine whether each electrical appliance is switched on or off.

Return to the 'Run simulation' button click procedure and add lines of program:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  hour,house,light,appliance,n:integer;
  total:real;
  textline:string;
begin
  randomize;
  listbox1.clear;
  for hour:=0 to 23 do
  begin
    total:=0;
```

ADD THIS

```
for house:=1 to 20 do
begin
  for light:=1 to 8 do
  begin
     {check if this light is on}
     n:=random(100);
     if hour<6 then
     begin
       if n<10 then
          total:=total+60;
     end;
     if (hour>=6) and (hour<17) then
     begin
       if n<20 then
          total:=total+60;
     end;
     if hour>=17 then
     begin
       if n<60 then
          total:=total+60;
     end;
  end;
  for appliance:=1 to 12 do
  begin
     {check if this appliance is on}
     n:=random(100);
     if hour<6 then
     begin
       if n<5 then
          total:=total+500;
     end;
     if (hour>=6) and (hour<17) then
     begin
       if n<20 then
          total:=total+500;
     end;
     if hour>=17 then
     begin
       if n<40 then
          total:=total+500;
     end;
  end;
end;
total:=total/1000;
textline:='Hour: '+inttostr(hour);
listbox1.items.add(textline);
textline:= 'Power usage: ' +
         floattostrf(total,ffFixed,8,2)+' kW';
```

```
    listbox1.items.add(textline);
    listbox1.items.add('');
  end;
end;
```

The procedure begins by setting the random number genrator and blanking out the list box:

```
    randomize;
    listbox1.clear;
```

The loops which repeat for each **hour** and each **house** begin, followed by the loop which repeats for each **light bulb**:

```
for hour:=0 to 23 do
begin
   total:=0;
   for house:=1 to 20 do
   begin
      for light:=1 to 8 do
      begin
```

To check if a light is on, the computer first generates a random number:

```
      n:=random(100);
```

We check if the current hour is during the night period (12 p.m. - 6 a.m.):

```
      if hour<6 then
      begin
```

If so, we see if the random number was less than the 10% cut-off point on the number line:

```
        if n<10 then...
```

If so, then we add 60W to the electricity usage total:

```
          total:=total+60;
```

Similar groups of program lines check the percentages if the current hour is during the day or evening period, then the whole loop is repeated for the electrical appliances using 500W of power.

At the end of the **hour** loop, the total power consumption needs to be displayed. We first divide the total by 1000 to convert watts to kilowatts:

```
          total:=total/1000;
```

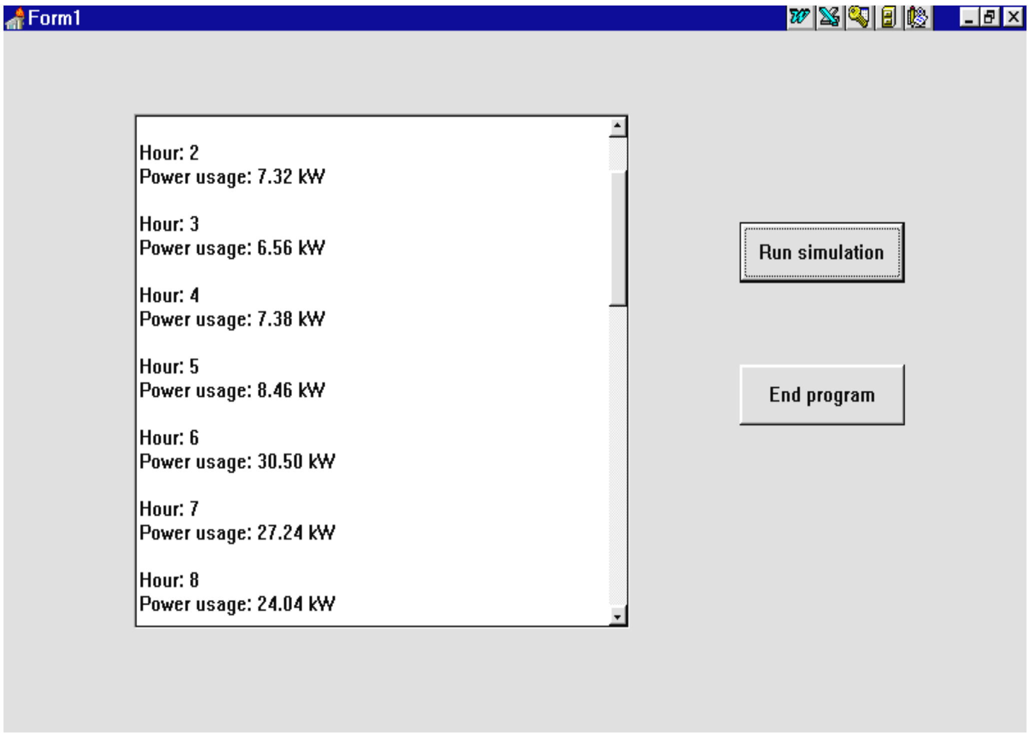
Text lines are then built up to display the time and power usage:

```
textline:='Hour: '+inttostr(hour);
textline:= 'Power usage: ' + floattostrf(total,ffFixed,8,2)+' kW';
```

Complete the program by producing an event handler for the '**End program**' button:

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

Compile and run the program. Press the '**Run simulation**' button several times, displaying a new set of results each time:

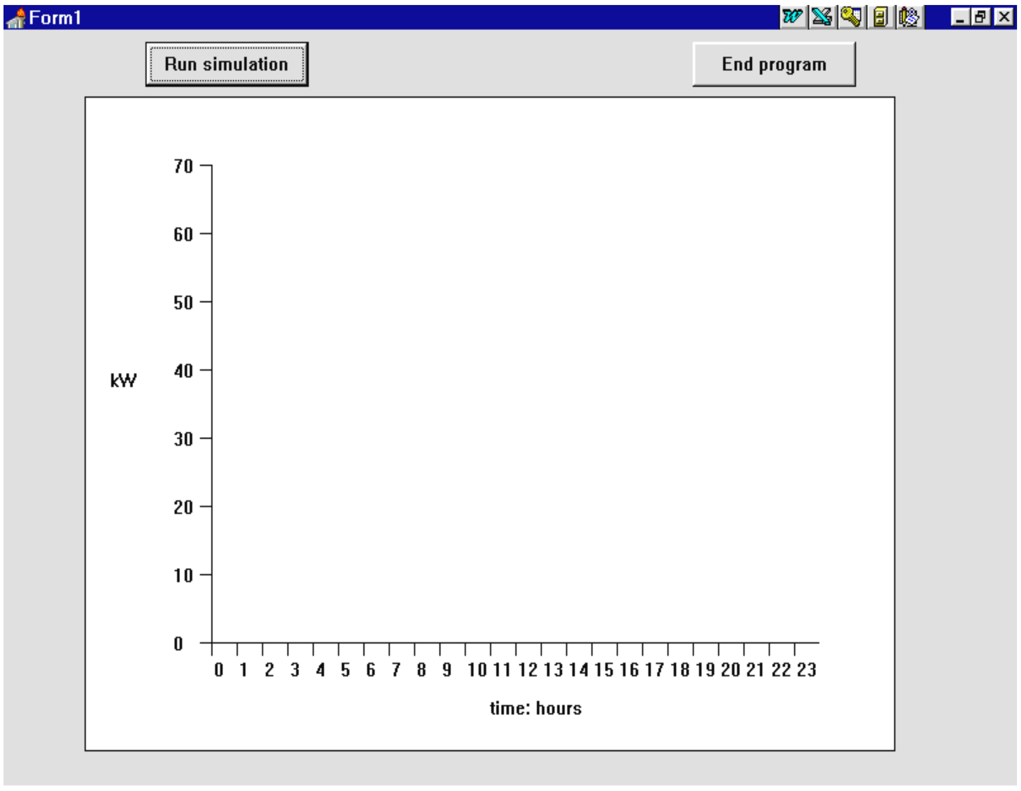


```
Hour: 2
Power usage: 7.32 kW

Hour: 3
Power usage: 6.56 kW

Hour: 4
Power usage: 7.38 kW

Hour: 5
Power usage: 8.46 kW

Hour: 6
Power usage: 30.50 kW

Hour: 7
Power usage: 27.24 kW

Hour: 8
Power usage: 24.04 kW
```

Each run of the simulation will give slightly different figures due to the random numbers, but notice that there is always a large increase in power consumption at 6 o'clock in the morning, and again at 5 o'clock in the evening.

When results are output in text form it can be difficult to appreciate the pattern of the data, so a graph of results is often better. We can do this now using the techniques for drawing histograms which we learned in chapter 9:

Return to the Delphi editing screen. Click on the List Box and press the DELETE key to remove it from the form. Place an *image box* on the form as shown below, and drag the buttons to a position above the image box. Use the Object Inspector to set the **Width** of the *image box* to 640, and the **Height** to 480.

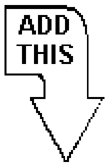


Modify the first part of the '**Run simulation**' procedure so that it draws and labels the graph axes:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  hour,house,light,appliance,n,x,y:integer;
  total:real;
  textline:string;
begin
  randomize;
  image1.canvas.brush.color:=clWhite;
  image1.canvas.rectangle(0,0,640,480);
  image1.canvas.moveto(100,50);
  image1.canvas.lineto(100,400);
  image1.canvas.lineto(580,400);
  for y:=0 to 7 do
  begin
    image1.canvas.moveto(100,400-y*50);
    image1.canvas.lineto(90,400-y*50);
    textline:=inttostr(y*10);
    image1.canvas.textout(70,393-y*50,textline);
  end;
  image1.canvas.textout(20,200,'kW');
```

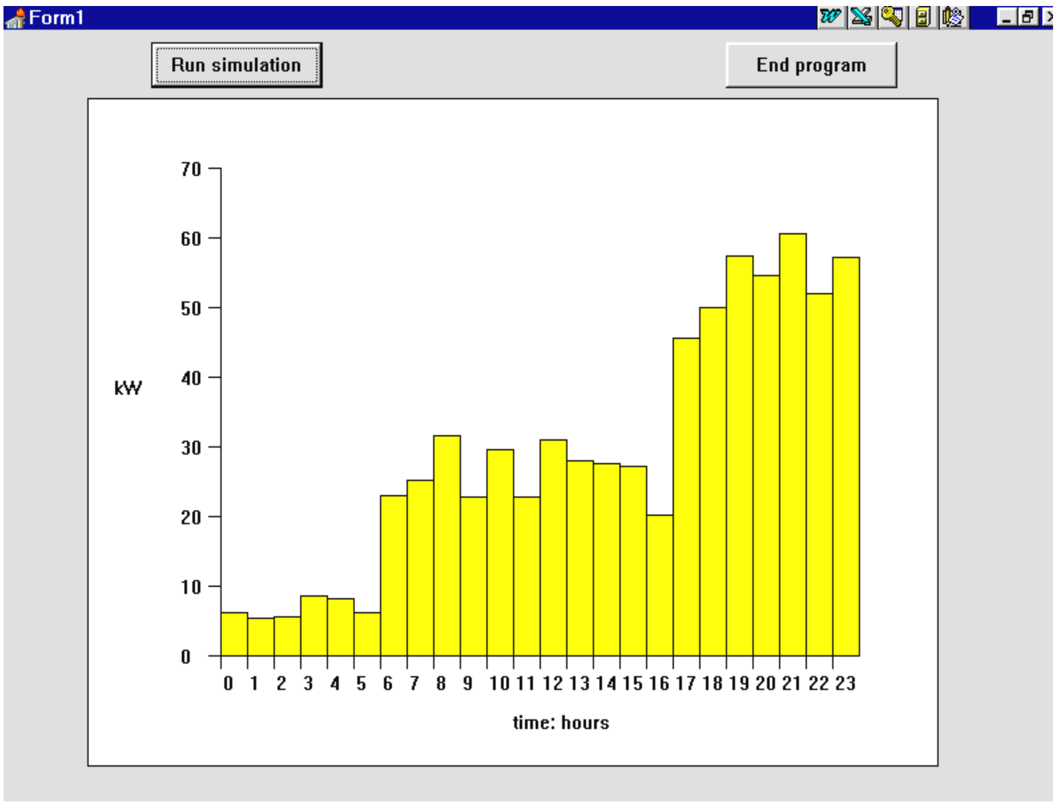
ADD THIS

```
for x:=0 to 23 do
begin
  image1.canvas.moveto(100+x*20,400);
  image1.canvas.lineto(100+x*20,410);
  textline:=inttostr(x);
  image1.canvas.textout(102+x*20,412,textline);
end;
image1.canvas.textout(320,440,'time: hours');
for hour:=0 to 23 do
begin
  total:=0;
  for house:=1 to 20 do
  begin
         .....
```



Remove all lines from the start and end of the procedure which contain **listbox1** commands - the list box has now been removed from the program so these would cause errors when the program runs.

Compile and run the program to check that the axes are drawn and labelled correctly when the '**Run simulation**' button is pressed, then return to the Delphi editing screen.

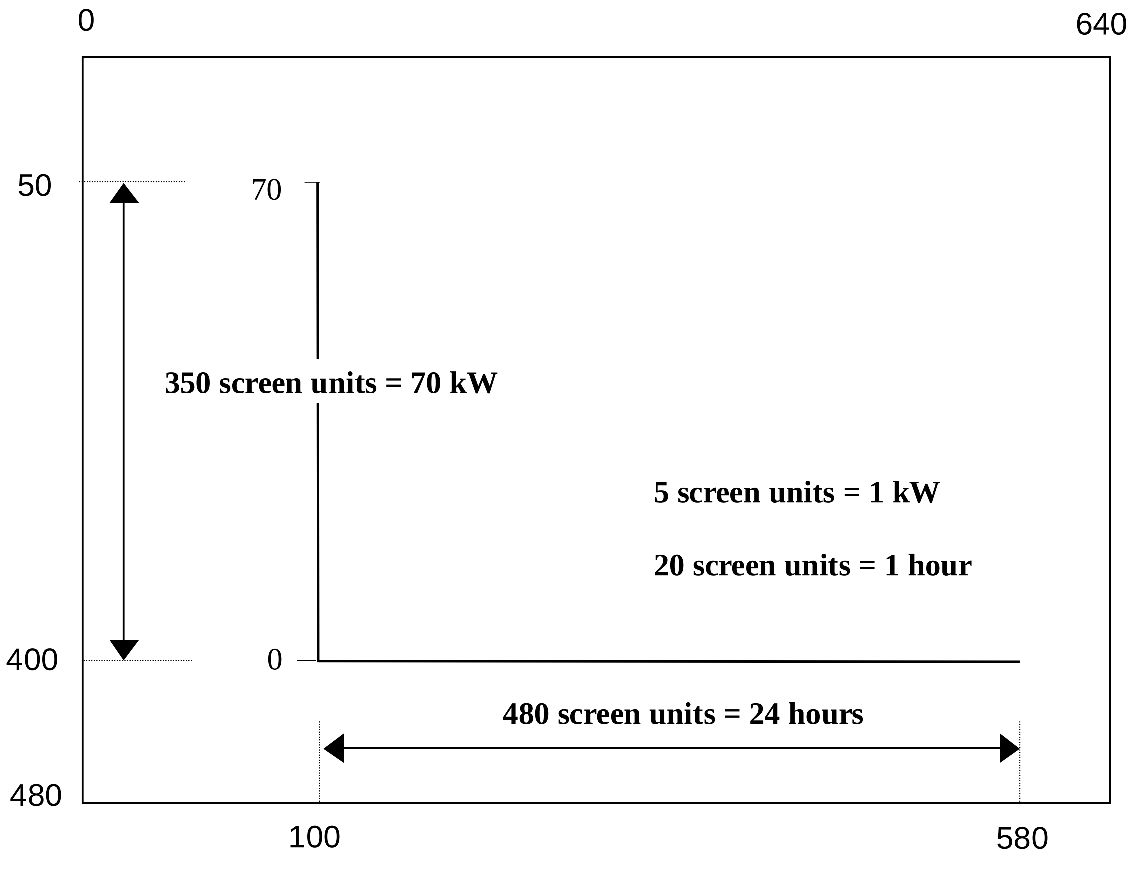


We now need to plot the columns on the histogram as the simulation runs.

Add lines at the end of the procedure which will use the '**total**' value to draw the columns of the histogram:

```
          ......
      end;
    end;
    total:=total/1000;
    image1.canvas.brush.color:=clRed;
    image1.canvas.rectangle(100+hour*20,
             400-round(total*5),121+hour*20,401);
  end;
end;
```

The graph has been scaled so that each hour is represented by 20 screen units horizontally, and each kilowatt is represented by 5 screen units vertically:



Compile and run the completed program. Each time the 'Run simulation' button is pressed, a new set of results is displayed. Notice the underlying pattern of electricity usage through the 24 hour period, despite the random fluctuations.