SEVENTEEN

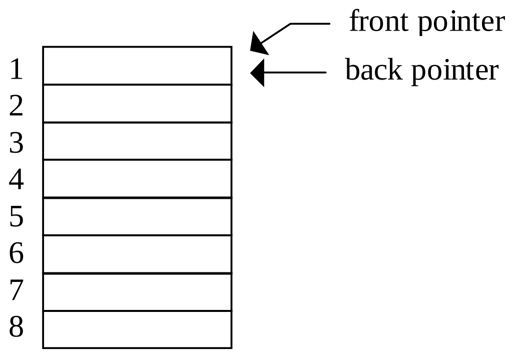
Queues

We are all familiar with the concept of a queue:

Queues play an
important role in computer systems, for example:

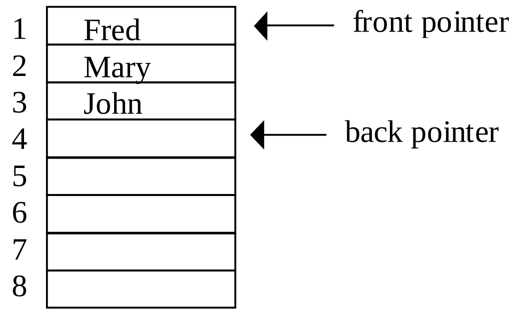
- If several computers are sharing the same printer, a queue can be used to store documents arriving while the printer is busy.
- A queue is used to store key presses received from the keyboard, so that they are processed in the correct order.

Queues are also often used in computer programs to ensure that data is dealt with in the correct sequence. The simplest way to set up a queue structure in a program is to use an array:

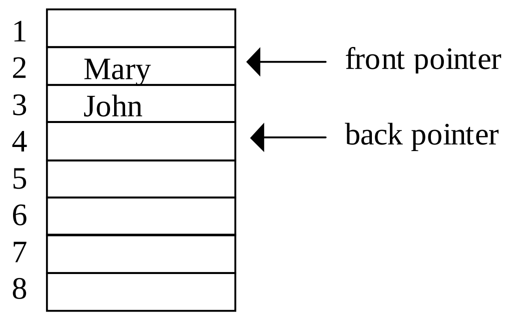


To operate a queue structure, two pointers are required:

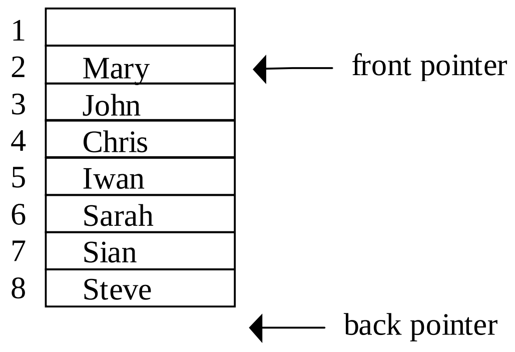
- the **back pointer** shows where the *next data item will be added* to the array,
- the **front pointer** shows where the *next data item will leave* the array.



Initially the **back pointer** points to **array box 1**, but it moves downwards as items are added. The back pointer always shows where the *next item will be added.*

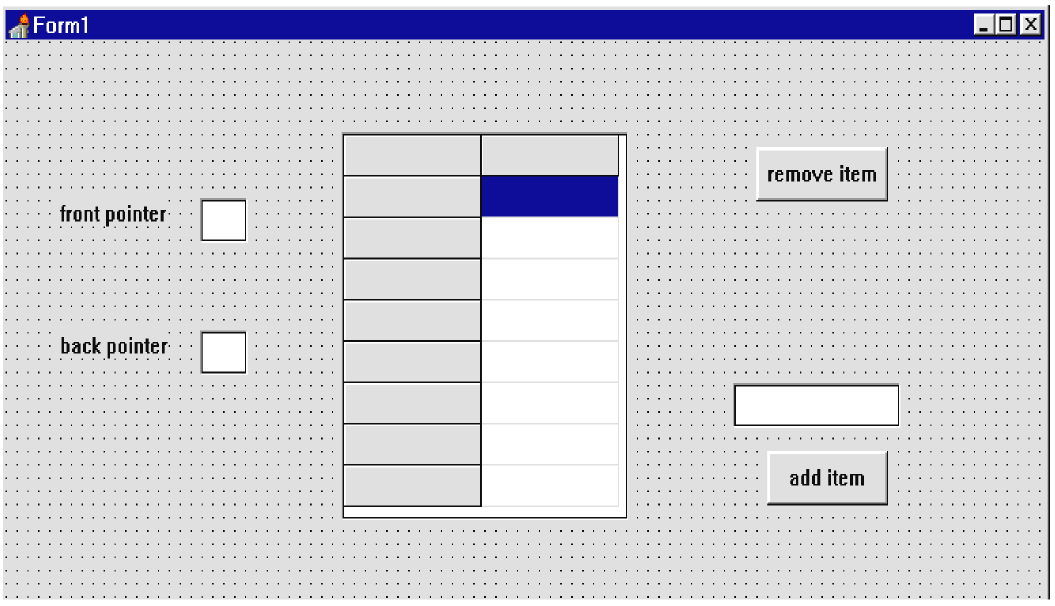


If an item is removed from the queue, the **front pointer** moves downwards. The front pointer always shows the position where the *next item will leave* the array.



If we continue to add items to the queue, eventually the the **back pointer** moves beyond the last array box. Once this happens, no further items can be added to the queue.

Let's see how this queue structure can operate in a computer program. Set up a new directory QUEUE and save a Delphi project into it. Use the Object Inspector to **maximize** the Form, and drag the form grid to nearly fill the screen.



Add components to the *Form* as shown above:

- Towards the left of the *Form* put two ***Edit boxes,*** with ***Labels*** alongside captioned '**front pointer**' and '**back pointer**'.
- In the centre of the *Form* add a ***String Grid***. Set the properties:

ColCount	**2**
RowCount	**9**
DefaultColWidth	**100**
ScrollBars	**None**

- Towards the upper right of the *Form* put a ***Button*** with the caption '**remove item**'
- Towards the lower right of the *Form* put an ***Edit Box***, and below it a ***Button*** with the caption '**add item**'.

Compile and run the program to check that the components are displayed correctly, then return to the Delphi editing screen.

Double-click the Form grid to produce an 'OnCreate' event handler, then add the lines of program:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i:integer;
begin
  with stringgrid1 do
  begin
    cells[0,0]:='array index';
    cells[1,0]:='data';
    for i:=1 to 8 do
    begin
      cells[0,i]:=inttostr(i);
      data[i]:='****';
      cells[1,i]:=data[i];
    end;
  end;
  front:=1;
  back:=1;
  edit1.text:=inttostr(front);
  edit2.text:=inttostr(back);
end;
```

The procedure begins by writing captions for the columns of the string grid:

```
with stringgrid1 do
begin
    cells[0,0]:='array index';
    cells[1,0]:='data';
```

The queue is going to be set up using an array of eight memory locations as shown in the diagrams on the previous page. We use a loop to put the entry '****' into each of the array boxes to indicate that no data has been entered yet:

```
for i:=1 to 8 do
    data[i]:='****';
```

The loop displays these entries in the '**data**' column of the *string grid*:

```
    cells[1,i]:=data[i];
```

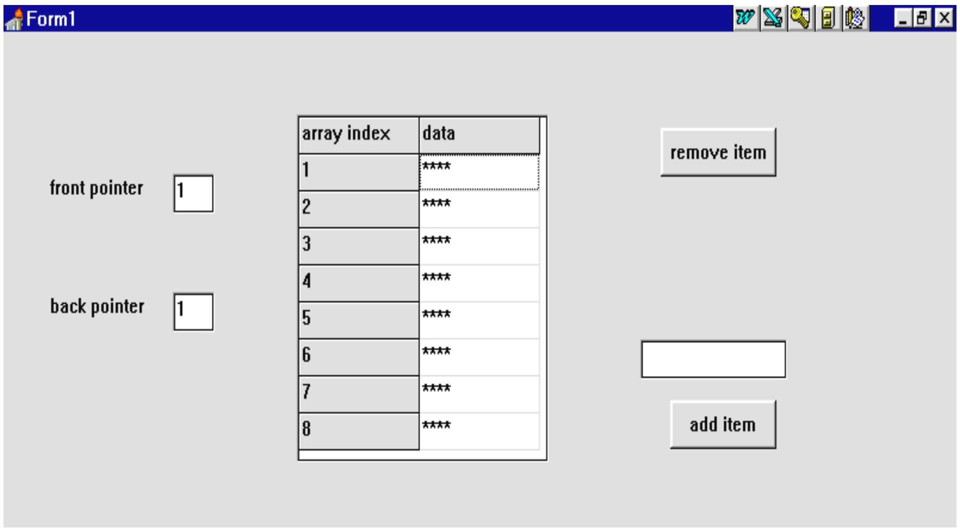
The front and back pointers for the queue are both intialised to 1. This represents the starting position for an empty queue. The pointer values are then displayed in the *edit boxes* on the form:

```
front:=1;
back:=1;
edit1.text:=inttostr(front);
edit2.text:=inttostr(back);
```

Go to the public declarations section near the top of Unit1 and add the variables:

```
{ Public declarations }
  front,back:integer;
  data:array[1..8] of string;
```

Compile and run the program. Check that the *string grid* is correctly displaying the eight '****' values from the empty data array, and that the **front** and **back** pointer positions are both shown as **1**:



array index	data
1	****
2	****
3	****
4	****
5	****
6	****
7	****
8	****

front pointer: 1
back pointer: 1

Return to the Delphi editing screen. Double-click the '**add item**' button to create an event handler, then insert the lines:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if edit3.text>'' then
  begin
    data[back]:=edit3.text;
    back:=back+1;
    display;
    edit3.text:='';
    edit3.setfocus;
  end;
end;
```

The purpose of this procedure is to add an entry to the data array. We begin with an error trapping line to check that an entry has been typed into the *edit box:*

if edit3.text >' ' **then**
begin

If valid data is available, this is transferred from the *edit box* to the data array at the position indicated by the **back** pointer. (Remember that new data joins the back of the queue):

data[back]:=edit3.text;

The **back** pointer is moved down by one position, ready for the next data item to be added:

back:=back+1;

A procedure '**display**' will then update the *string grid* entries to show the new contents of the data array, and the *edit box* entries to show the new pointer values:

display;

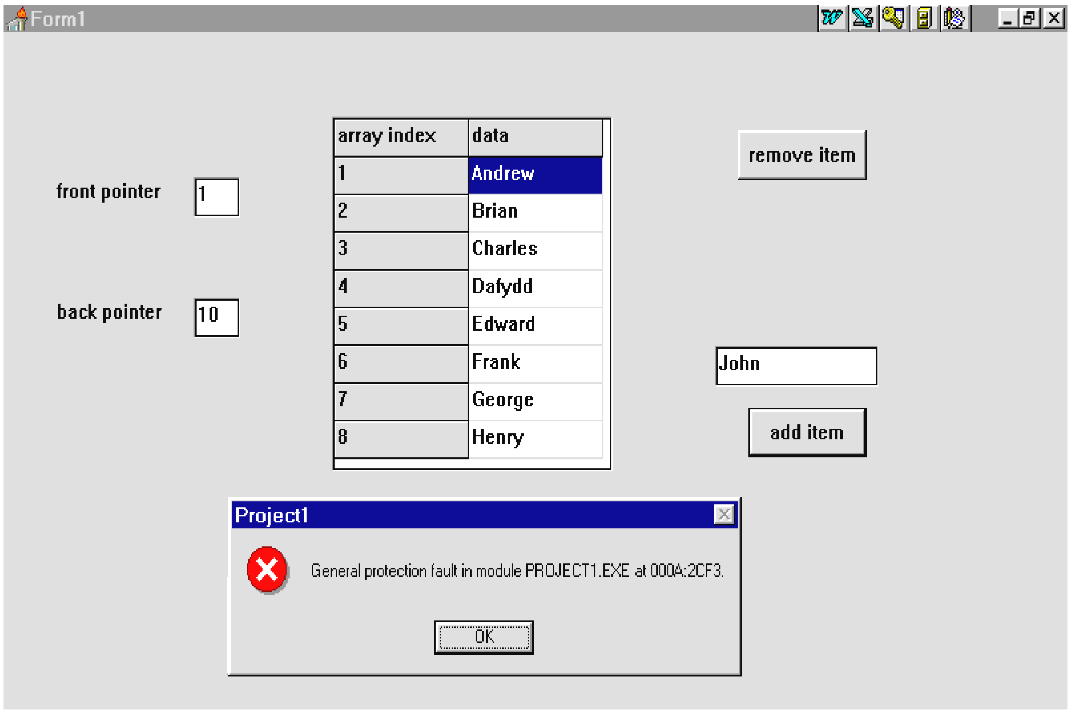
The next step is to write the '**display**' procedure. Go to the bottom of the program and insert the lines:

```
procedure TForm1.display;
var
  i:integer;
begin
  for i:=1 to 8 do
    stringgrid1.cells[1,i]:=data[i];
  edit1.text:=inttostr(front);
  edit2.text:=inttostr(back);
end;
```

Add '**display**' to the list of procedures near the top of the program:

```
    ......
procedure FormCreate(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure display;
```

Save your program so far, then compile and run it. Enter a series of names by typing in the edit box and clicking the '**add item**' button each time. Each new name should appear in the string grid, and the position of the back pointer should change. Notice that the back pointer always gives the number of the array element where the next name will be added:



array index	data
1	Andrew
2	Brian
3	Charles
4	Dafydd
5	Edward
6	Frank
7	George
8	Henry

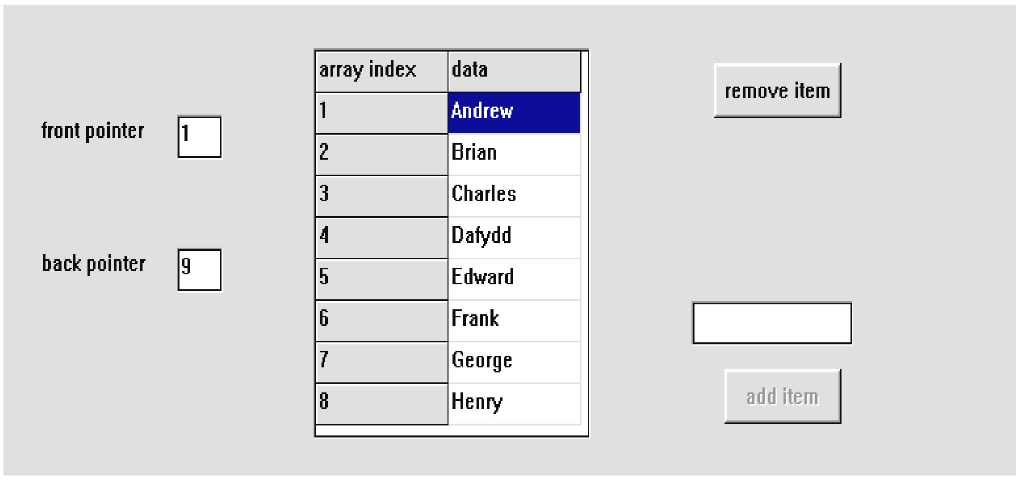
```
General protection fault in module PROJECT1.EXE at 000A:2CF3.
```

Eight names can be added successfuly, but problems then occur. There are no memory locations available in the '**data**' array for a nineth or tenth name, and this leads to an error which stops the program from running.

The simplest way of avoiding difficulties is to disable the '**add item**' *button* and *edit box* as soon as the eighth name has been entered. Go to the button click procedure and add the lines:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
      .....
    edit3.text:='';
    edit3.setfocus;
    if back>8 then
    begin
      button2.enabled:=false;
      edit3.enabled:=false;
    end;
  end;
end;
```

Compile and run the program, then re-enter the test data. This time, the '**add item**' button should be *greyed out* as soon as eight names have been entered:



front pointer 1

back pointer 9

array index	data
1	Andrew
2	Brian
3	Charles
4	Dafydd
5	Edward
6	Frank
7	George
8	Henry

Return to the Delphi editing screen. We can now turn our attention to removing data from the queue. Double-click the '**remove item**' button to produce an event handler, then add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  data[front]:='****';
  front:=front+1;
  display;
end;
```

This empties the data array box at the position of the **front** pointer. (Remember that data items leave from the front of the queue.) The front pointer is then moved down one place, ready to mark the position for the next data item to leave the queue.

Compile and run the program. Enter several names, then check that they can be removed by clicking the button. This should work, but it is possible to keep selecting the '**remove item**' option even if the queue is empty. The front pointer keeps increasing, and eventually the an error will occur and the program will stop.

Return to the '**remove item**' button click procedure and add an instruction to disable the button as soon as the queue becomes empty - this occurs when the front pointer has moved to the position of the back pointer:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  data[front]:='****';
  front:=front+1;
  display;
  if front=back then
    button1.enabled:=false;
end;
```

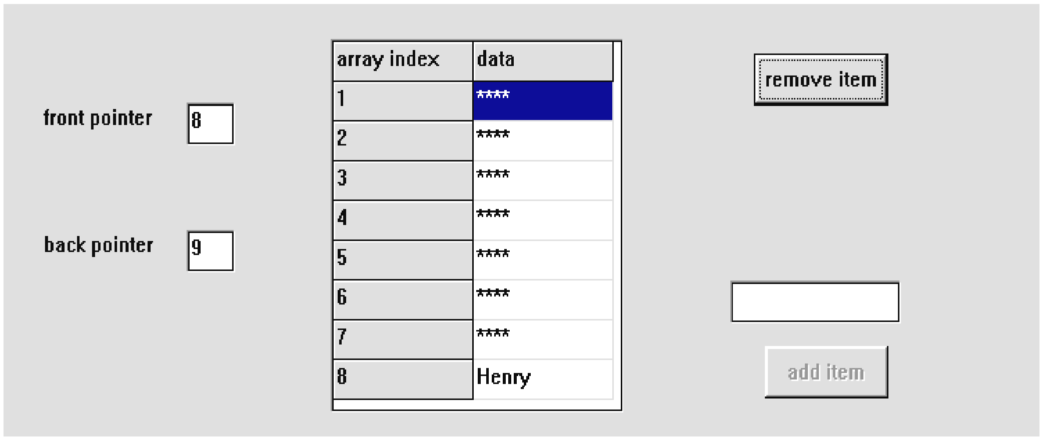
We must, however, remember to enable the button again if more items are entered. Go to the '**add item**' button click procedure and add a line:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if edit3.text>'' then
  begin
    data[back]:=edit3.text;
    back:=back+1;
    button1.enabled:=true;
    display;
    .......
```

Compile and run the program to check that items can be added and removed correctly, then return to the Delphi editing screen.

A circular queue

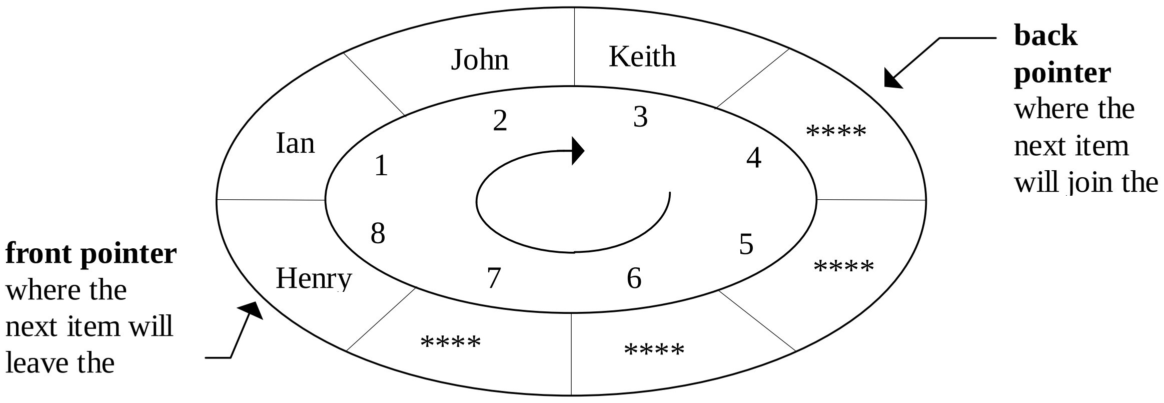
The program works, but we cannot operate the queue for very long before before running out of memory space in the array. As soon as box 8 is reached, no further entries are possible:



front pointer: 8
back pointer: 9

array index	data
1	****
2	****
3	****
4	****
5	****
6	****
7	****
8	Henry

A simple way to solve this problem is to reuse the empty memory locations at the start of the array where data items have been deleted:



back pointer
where the next item will join the
front pointer
where the next item will leave the
John
Keith
Ian
Henry
****
****
****
****
1
2
3
4
5
6
7
8

The queue can be treated as a circular data structure. A pointer passing position 8 returns to the start of the array at position 1. This allows the queue to continue indefinitely, provided that the total number of names in the queue at any one time never exceeds the eight array boxes available.

To implement the circular queue, it is necessay to make a few modifications to the program. Go first to the 'add item' button click procedure and add the lines indicated:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if edit3.text>'' then
  begin
    data[back]:=edit3.text;
    back:=back+1;
    button1.enabled:=true;
    if back>8 then
      back:=1;
    display;
    edit3.text:='';
    edit3.setfocus;
    if back=front then
    begin
      button2.enabled:=false;
      edit3.enabled:=false;
    end;
  end;
end;
```

The command:

if back>8 then
back:=1;

resets the **back** pointer to position 1 when it passes the end of the array.

The section of program:

```
if back=front then
begin
    button2.enabled:=false;
    edit3.enabled:=false;
end;
```

disables the '**add item**' button and edit box whenever the queue becomes full.

Note: The **front** and **back** pointers are at the same position in the array whenever the queue is either ***full*** or ***empty***. In this case we know the queue must be ***full*** because the pointers have become equal as a result of adding an item.

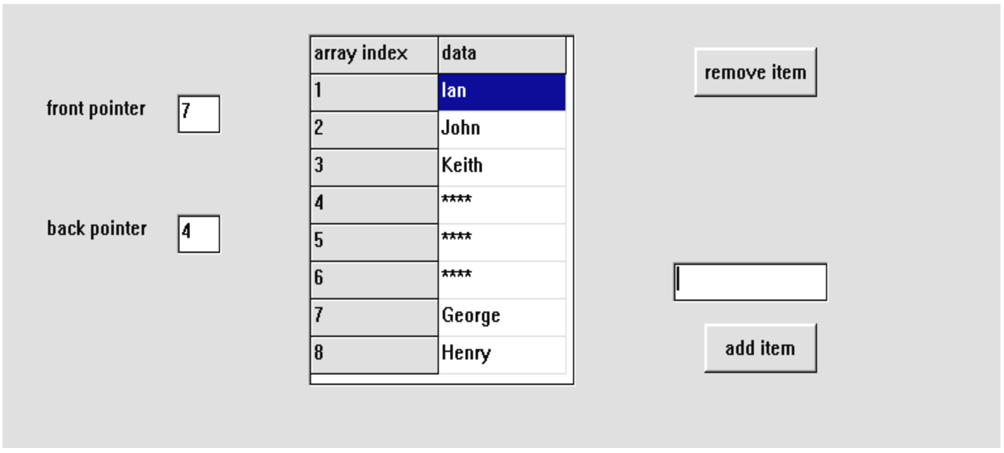
Go now to the '**remove item**' button click procedure and add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  data[front]:='****';
  front:=front+1;
  button2.enabled:=true;
  edit3.enabled:=true;
  if front>8 then
    front:=1;
  display;
  if front=back then
    button1.enabled:=false;
end;
```

The lines:

```
button2.enabled:=true;
edit3.enabled:=true;
```

ensure that the '**add item**' *button* and *edit box* are enabled whenever there is an empty space available in the array to accept a data item.



array index	data
1	Ian
2	John
3	Keith
4	****
5	****
6	****
7	George
8	Henry

Compile and run the program. It should now be possible to add and delete names from the queue, with the pointers operating in a circular structure as in the diagram above.

Our next project illustrates how a queue structure can be incorporated into a data processing program:

Application

Airport landings

A busy airport is being expanded by the addition of extra facilities. The plan is to be able to accept one aircraft landing per minute at peak capacity. The air traffic controllers are concerned to ensure that the increased traffic in the airspace around the airport will not jepordise the safety of aircraft waiting to land, and they have asked for a simulation of the aircraft arrivals to be carried out.

The simulation will be based on two input values, **arrivals frequency** and **landing probability**:

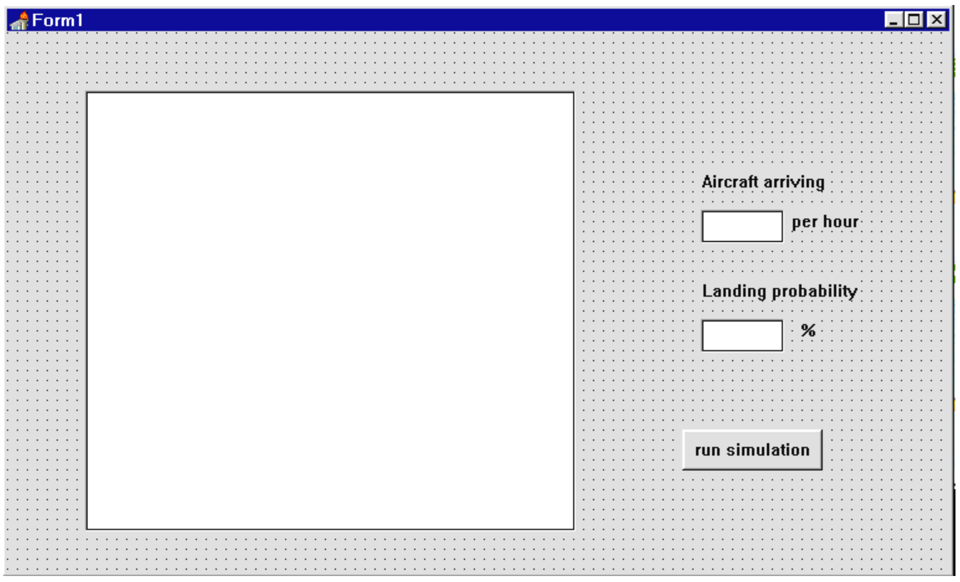
- The **arrivals frequency** is the number of aircraft arriving in the airspace around the airport each hour.
- In ideal conditions, one aircraft can land each minute. However, there are times when a runway cannot be used because a previous aircraft is not yet clear or the runway surface is being checked by mainteneance crews. The **landing probability** is the percentage chance that a runway is clear and available for an aircraft to land during a particular minute.

At present the airport has a single runway and the landing probability is **50%**. If a second runway is built, it is expected that the landing probability will increase to **80%**.

A program is required which will input the *arrivals frequency* and *landing probaility*, then simulate three hours of operation of the airport.

The air traffic control authority has specified that the time any aircraft has to wait to land must not exceed 30 minutes. Use the program to find the maximum number of aircraft which can use the airport per hour with the current single runway, and how many per hour could be handled if the second runway is built.

To begin the simulation program, set up a new directory LANDING and save a Delphi project into it. Use the Object Inspector to **maximize** the Form, and drag the form grid to nearly fill the screen.



Add components to the *Form* as shown above:

- On the left of the *Form* put a ***List Box***.
- To the right put an ***Edit Box*** and the ***Labels:*** '**Aircraft arriving**' and '**per hour**'.
- Below this put another ***Edit Box*** and the ***Labels:*** '**Landing probability**' and '**%**'.
- Complete the Form by adding a ***Button*** with the caption '**run simulation**'.

Double-click the 'Aircraft arriving' *Edit Box* to produce an event handler, then add the lines:

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

Double-click the 'landing probability' Edit Box and set up a similar event handler:

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

Add the variables '**freq**' and '**prob**' to the Public declarations section:

```
public
  { Public declarations }
  freq,prob:integer;
end;
```

Compile and run the program. Check that the 'Aircraft arriving' and 'Landing probability' *Edit Boxes* are error trapped to accept integer numbers, then return to the Delphi editing screen.

Go to the Public declarations section of *Form1* and add an array:

```
{ Public declarations }
freq,prob:integer;
arrive:array[1..180] of integer;
```

This array has 180 memory locations which will be used to represent the 180 minutes of the simulation period. We can arrange to put numbers in each array box to show the number of aircraft arriving above the airport in any particular minute, for example:

ARRIVE array

1	2	0	0	1	2	1	0	1	0	0	0	1	0	2	0	...															

1 2 3 .... minute 180

In this case:

- 1 aircraft arrives in minute 1
- 2 aircraft arrive in minute 2
- no aircraft arrive in minute 3, etc....

Before the simulation runs we will know the **arrival frequency** for the aircraft. Supposing that we choose a value of ***50 aircraft per hour***. This will mean that:

- 50 aircraft arrive between minute 1 and minute 60

- 50 more aircraft arrive between minute 61 and 120,   etc...

Within each hour, however, we will assume that the arrivals which add up to these totals are randomly distributed - arrival times cannot be planned precisely due to wind speeds and other factors affecting the flights. An algorithm for setting up the ARRIVE array is therefore:

```
1.  LOOP for minute from 1 to 180
2.      Set the arrivals for that minute to zero
3.  END LOOP
4.  Initialise the random number generator
5.  LOOP for each of the 3 hours
6.      LOOP for each of the aircraft arriving that hour
7.          Get a random number to choose a minute during
            the hour when this aircraft will arrive
8.          Add 1 aircraft to the number arriving in the
            chosen minute
9.      END LOOP
10. END LOOP
```

Go to the end of the program and add a procedure '**arrivals**' to implement this algorithm:

```
procedure TForm1.arrivals;
var
  i,n,hour,minute:integer;
begin
  for i:=1 to 180 do
    arrive[i]:=0;
  randomize;
  for hour:=0 to 2 do
  begin
    for i:=1 to freq do
    begin
      n:=random(60)+1;
      minute:=(hour*60)+n;
      arrive[minute]:=arrive[minute]+1;
    end;
  end;
end;
```

Add '**arrivals**' to the list of procedures at the top of the Unit 1:

```
type
  TForm1 = class(TForm)
         ....
    procedure Edit2Change(Sender: TObject);
    procedure arrivals;
```

Double-click the '**run simulation**' button to create an event handler and add the lines of program:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  minute:integer;
begin
  arrivals;
  listbox1.clear;
  for minute:=1 to 180 do
  begin
    display(minute);
  end;
end;
```

This begins by calling the '**arrivals**' procedure to set up the arrival times for aircraft during the simulation. We then begin a loop which will repeat for each of the 180 minutes of the simulation period. Each time around the loop we will call another procedure '**display**' to give current information about aircraft arriving, landing or waiting to land. This will allow us to monitor what is happening at the airport.

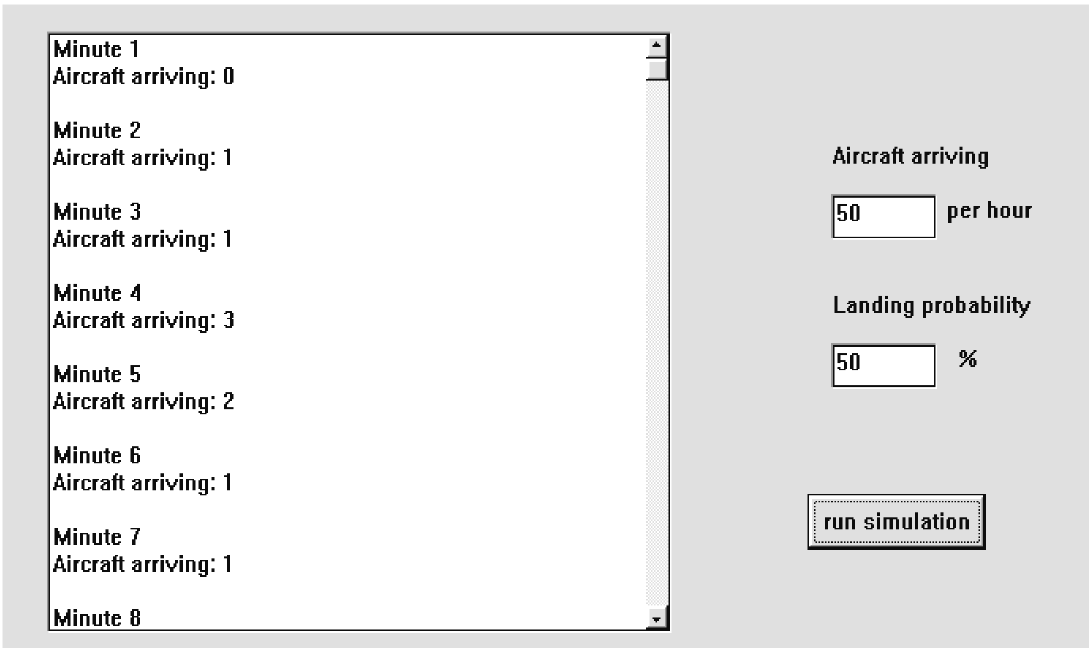
Go to the end of the program and add the '**display**' procedure:

```
procedure TForm1.display(minute:integer);
var
  textline:string;
begin
  textline:='Minute '+inttostr(minute);
  listbox1.items.add(textline);
  textline:='Aircraft arriving: '+
                   inttostr(arrive[minute]);
  listbox1.items.add(textline);
  listbox1.items.add('');
end;
```

Also add '**display**' to the procedure list at the top of Unit 1:

```
  TForm1 = class(TForm)
          .......
     procedure arrivals;
```

```
        procedure display(minute:integer);
```
Compile and run the program. Enter an **arrival frequency** of **50** aircraft per hour, and a **landing probability** of **50%**, then press the '**run simulation**' button. The numbers of the minutes from 1 to 180 should be displayed in the *List Box*, along with the numbers of aircraft arriving during the minute:


```
Minute 1
Aircraft arriving: 0

Minute 2
Aircraft arriving: 1

Minute 3
Aircraft arriving: 1

Minute 4
Aircraft arriving: 3

Minute 5
Aircraft arriving: 2

Minute 6
Aircraft arriving: 1

Minute 7
Aircraft arriving: 1

Minute 8
```
Aircraft arriving: 50 per hour

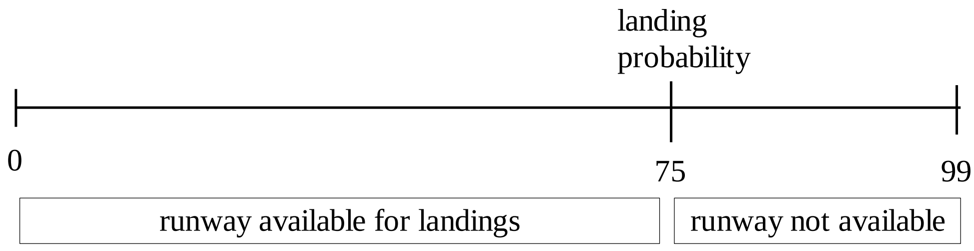
Landing probability: 50 %

You might check that the total numbers of aircraft arriving during the first hour (minutes 1 - 60) add up to the arrival frequency of 50 which we specified. Try different arrival frequencies, then return to the Delphi editing screen.

The next step is to use a random number to decide whether the runway is clear for landings. Go to the '**run simulation**' button click procedure and add the lines:

```
   procedure TForm1.Button1Click(Sender: TObject);
  var
    n,minute:integer;
  begin
    arrivals;
    listbox1.clear;
    for minute:=1 to 180 do
    begin
      n:=random(100);
      if n<=prob then
        runwayclear:=true
      else
        runwayclear:=false;
    display(minute);
    .....
```

This makes use of a number line technique. We obtain a random number in the range 0-99, then compare this to the **percentage probability** for the runway being clear. Suppose the probability is 75%, this would mean that a landing is possible if the random number is below 75, but the runway is not available if the number is 75 or more:



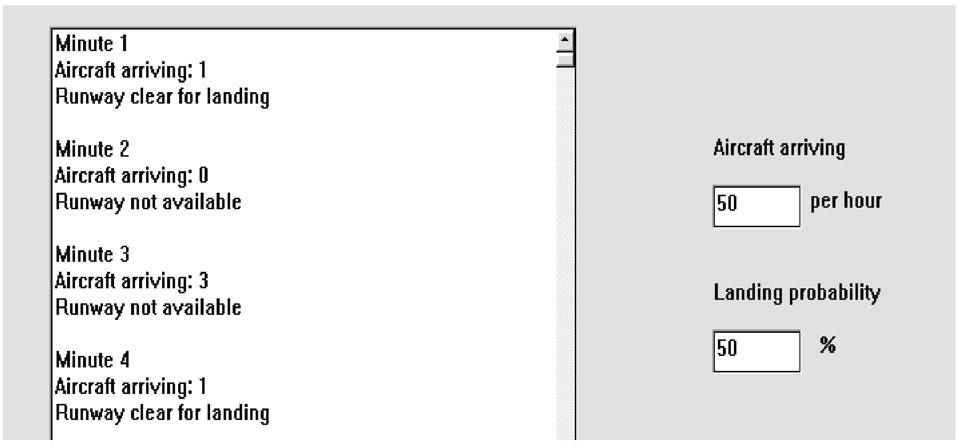
Add '**runwayclear**' to the list of variables under the Public declarations heading:

```
public
  { Public declarations }
  freq,prob:integer;
  runwayclear:boolean;
```

We also need to include a message about the runway in the '**display**' procedure. Insert the lines:

```
procedure TForm1.display(minute:integer);
var
  textline:string;
begin
  textline:='Minute '+inttostr(minute);
  listbox1.items.add(textline);
  textline:='Aircraft arriving: '+
                   inttostr(arrive[minute]);
  listbox1.items.add(textline);
  if runwayclear then
     listbox1.items.add
                  ('Runway clear for landing')
  else
     listbox1.items.add('Runway not available');
  listbox1.items.add('');
end;
```

Compile and run the program. A message about the state of the runway should be shown for each minute. Return to the Delphi editing screen when you have tested this.


```
Minute 1
Aircraft arriving: 1
Runway clear for landing

Minute 2
Aircraft arriving: 0
Runway not available

Minute 3
Aircraft arriving: 3
Runway not available

Minute 4
Aircraft arriving: 1
Runway clear for landing
```

Aircraft arriving: 50 per hour

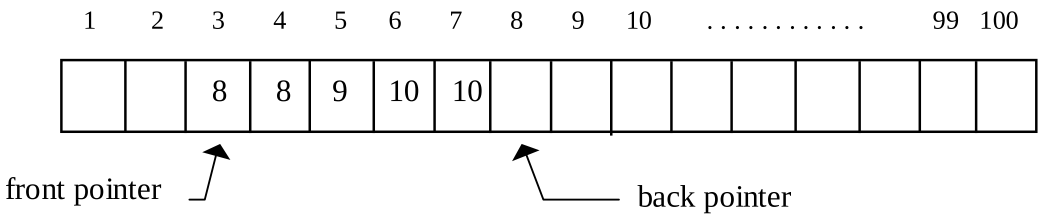
Landing probability: 50 %

We are now ready to tackle the main objective of the simulation - to find the amount of time that aircraft have to wait above the airport before being able to land. This will require a queue structure to record the arrival time of each aircraft.

To see how the queue will work, consider the following sequence of events:

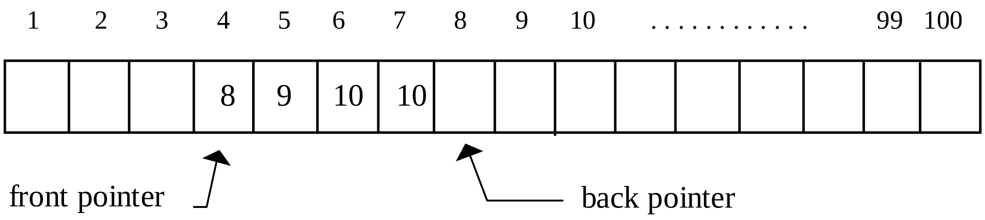
minute 8	2 aircraft arrive
minute 9	1 aircraft arrives
minute 10	2 aircraft arrive

We record each arrival by entering the **minute number** in the queue:



New arrivals are entered at the position of the **back pointer**, then the pointer moves forward one space.

Suppose that it is now minute 11. The runway is clear, so one of the aircraft which arrived in minute 8 can land. This is removed from the queue and the **front pointer** is moved forward on place:



We know that the aircraft arrived in **minute 8** and landed in **minute 11**, so the waiting time must have been **3 minutes**.

We can go on adding new aircraft to the queue as they arrive above the airport, and removing them from the queue when they land, until the simulation period is completed. Each time an aircraft lands, we calculate the waiting time with the formula:

waiting time = (minute of landing) - (minute of arrival)

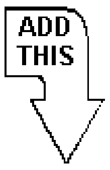
The queue can be represented as an array using the techniques we learned earlier in this chapter. We must allow for the maximum number of aircraft which could be in the queue at any time - 100 should be more than enough array boxes.

Begin by adding extra variables to the *Public declarations* section:

```
{ Public declarations }
  freq,prob:integer;
  runwayclear:boolean;
  arrive:array[1..180] of integer;
  queue:array[1..100] of integer;
  front,back,waittime:integer;
```

The next step is to add lines of program to the '**run simulation**' button click procedure. These will initialise the empty queue, then record aircraft arrivals:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 n,i,minute:integer;
begin
  arrivals;
  listbox1.clear;
  front:=1;
  back:=1;
  for minute:=1 to 180 do
  begin
    n:=random(100);
    if n<=prob then
      runwayclear:=true
    else
      runwayclear:=false;
    if arrive[minute]>0 then
    begin
      for i:=1 to arrive[minute] do
      begin
        queue[back]:=minute;
        back:=back+1;
```



```
          if back>100 then
            back:=1;
        end;
      end;
      display(minute);
    end;
  end;
```

The lines:

```
front:=1;
back:=1;
```

set the front and back pointers to the starting positions for an empty queue.

The section of program:

```
if arrive[minute]>0 then
  begin
   ....
  end;
end;
```

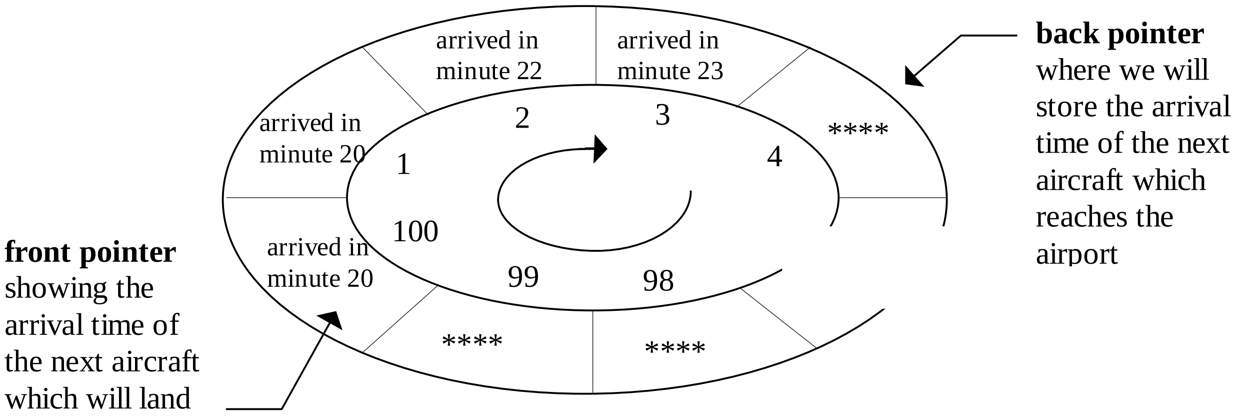
only operates if there is at least one aircraft arriving during the current minute. We then begin a loop:

for i:=1 to arrive[minute] do . . . .

which repeats for each of the aircraft arriving.

The **minute number** is stored in the array at the position of the **back pointer**, then the pointer is moved forward one place:

```
queue[back]:=minute;
back:=back+1;
```


arrived in minute 22
arrived in minute 23
back pointer where we will store the arrival time of the next aircraft which reaches the airport
arrived in minute 20
2
3
****
1
4
100
arrived in minute 20
99
98
front pointer showing the arrival time of the next aircraft which will land
****
****

We operate the queue as a circular structure, so that the pointer returns to position 1 after passing the end of the array:

if back>100 then

back:=1;

We must now arrange for aircraft to leave the queue when they land. A further set of lines must be inserted in the '**run simulation**' button click procedure to do this:

```
        .........
     if arrive[minute]>0 then
     begin
       for i:=1 to arrive[minute] do
       begin
         queue[back]:=minute;
         back:=back+1;
         if back>100 then
           back:=1;
       end;
     end;
     planelanded:=false;
     if runwayclear then
     begin
       if front<>back then
       begin
         planelanded:=true;
         waittime:=minute-queue[front];
         front:=front+1;
         if front>100 then
           front:=1;
       end;
     end;
    display(minute);
  end;
end;
```

The section of program:

```
        if runwayclear then
        begin
         ......
        end;
```

only operates if the runway is clear for an aircraft to land.

The next section only operates if the queue is not empty:

```
        if front<>back then
        begin
         .....
        end;
```

(Remember that the front and back pointers are in the same position for an empty queue.)

A boolean variable is used to record that the aircraft lands:

planelanded:=true;

We calculate the waiting time for the plane which has just landed:

```
waittime:=minute-queue[front];
```

then remove the aircraft from the queue by moving the front pointer forward by one place. The pointer is reset to position 1 if it passes the end of the array:

```
front:=front+1;
if front>100 then
    front:=1;
```

Add the booleann variable '**planelanded**' to the *Public declarations* section:

```
{ Public declarations }
 .........
runwayclear,planelanded:boolean;
```

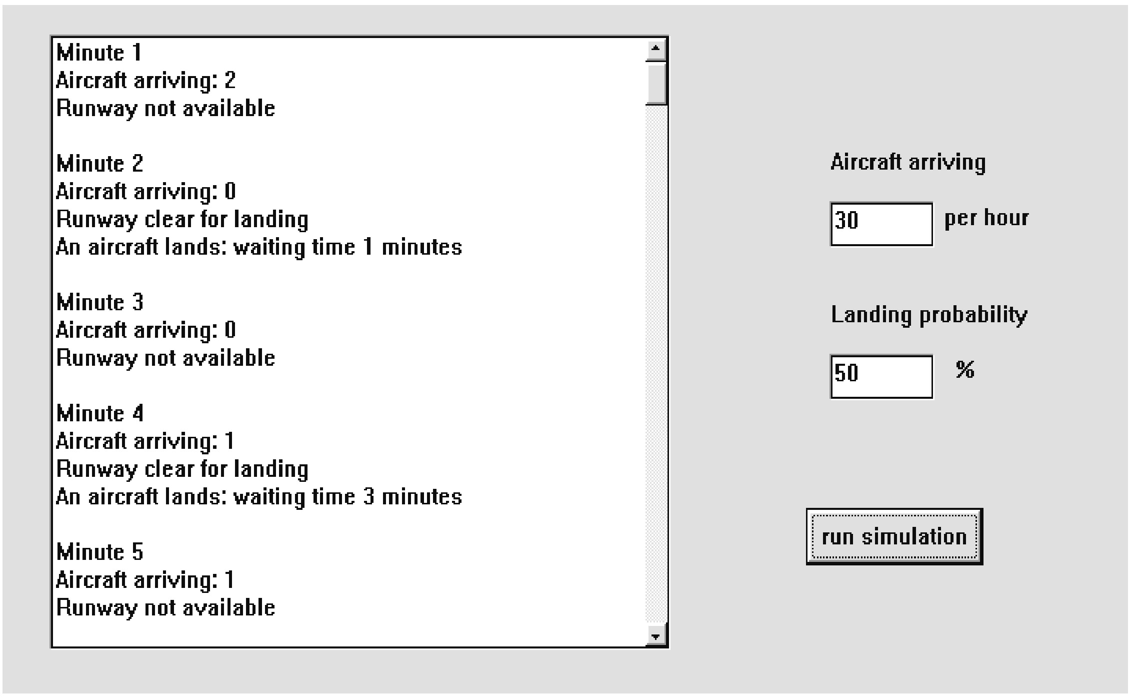
The work on the queue is now completed, but we need to add some more lines of program to the '**display**' procedure to show what is happening during the simulation. Insert the section shown below:

```
procedure TForm1.display(minute:integer);
var
  textline:string;
  i,last:integer;
begin
  textline:='Minute '+inttostr(minute);
  listbox1.items.add(textline);
  textline:='Aircraft arriving: '+
                        inttostr(arrive[minute]);
  listbox1.items.add(textline);
  if runwayclear then
  begin
    listbox1.items.add('Runway clear for landing');
    if planelanded=false then
      listbox1.items.add
                    ('No aircraft waiting to land')
    else
    begin
      textline:='An aircraft lands: waiting time ';
      textline:=textline+
                     inttostr(waittime)+' minutes';
      listbox1.items.add(textline);
    end;
  end
  else
     listbox1.items.add('Runway not available');
  listbox1.items.add('');
end;
```

Compile and run the program.

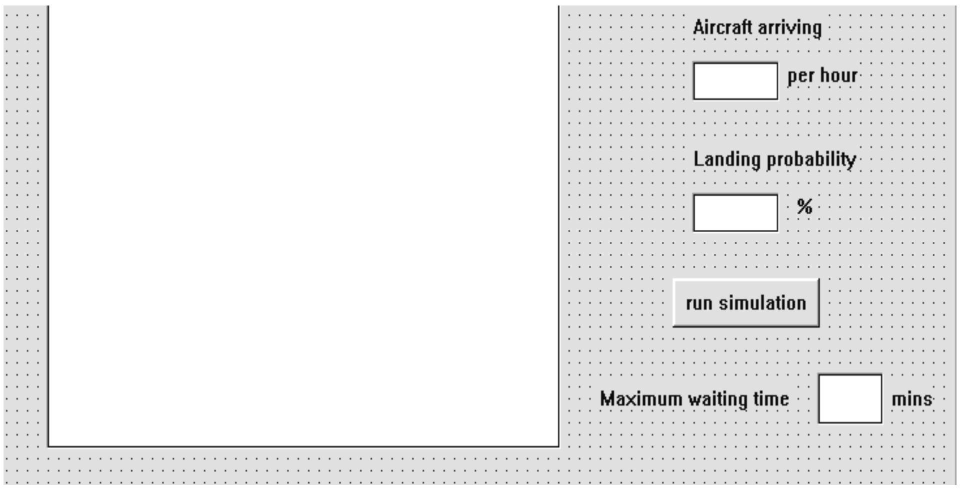
For each minute of the simulation the program should display one of the three messages:

- runway not available
- runway available but no aircraft waiting to land
- runway available and an aircraft lands - the waiting time will be given



```
Minute 1
Aircraft arriving: 2
Runway not available

Minute 2
Aircraft arriving: 0
Runway clear for landing
An aircraft lands: waiting time 1 minutes

Minute 3
Aircraft arriving: 0
Runway not available

Minute 4
Aircraft arriving: 1
Runway clear for landing
An aircraft lands: waiting time 3 minutes

Minute 5
Aircraft arriving: 1
Runway not available
```

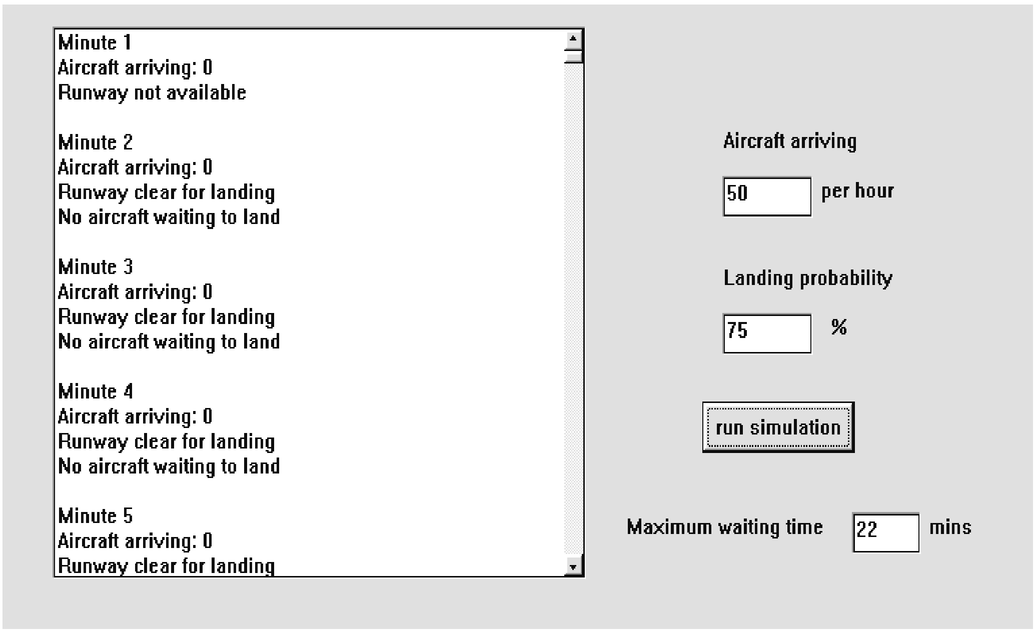
The final stage of the program is to record and display the maximum waiting time of any aircraft during the simulation period. Return to the Delphi editing screen and bring the *Form1* window to the front. Add an ***Edit Box*** and ***Labels*** with the captions '**Maximum waiting time**' and '**mins**':



Add the following lines to the '**run simulation**' button click procedure:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 n,i,minute:integer;
 maxwait:integer;
begin
  arrivals;
  listbox1.clear;
  maxwait:=0;
  front:=1;
  back:=1;
  for minute:=1 to 180 do
  begin
    ..........
    if runwayclear then
    begin
      if front<>back then
      begin
        waittime:=minute-queue[front];
        if waittime>maxwait then
          maxwait:=waittime;
       ........
    display(minute);
  end;
  edit3.text:=inttostr(maxwait);
end;
```

Compile and run the simulation. The maximum waiting time should be displayed:



```
Minute 1
Aircraft arriving: 0
Runway not available

Minute 2
Aircraft arriving: 0
Runway clear for landing
No aircraft waiting to land

Minute 3
Aircraft arriving: 0
Runway clear for landing
No aircraft waiting to land

Minute 4
Aircraft arriving: 0
Runway clear for landing
No aircraft waiting to land

Minute 5
Aircraft arriving: 0
Runway clear for landing
```

Aircraft arriving: 50 per hour

Landing probability: 75 %

Maximum waiting time: 22 mins

Use your program to answer the questions we were asked by the airport planners:

> At present the airport has a single runway and the landing probability is **50%**. If a second runway is built, it is expected that the landing probability will increase to **80%**. The air traffic control authority has specified that the time any aircraft has to wait to land must not exceed 30 minutes.
>
> - What is the maximum number of aircraft which can use the airport per hour with the current single runway?
> - How many aircraft per hour could be handled if the second runway is built?