Aim:

To complete the implemented class & understand class member uses and interactions.

Preliminary:

Simulation of real-world activities is one of the most common uses of computers. Whether it is simulating a system of supermarket checkouts to determine whether there is a need for extra checkouts or better equip-ment, or a set of traffic lights to test different timings of the stop/go interval, simulation utilises random num-bers to emulate the random occurrences of nature. Often such simulations lead to the automation programs used in many day-to-day computerised systems. This assignment is an introduction to this area of computing – to control an automatic elevator.

Procedure:

You have been asked to write the code which controls an automated elevator. As part of the process, you are to create the class `Elevator` to represent the physical entity. Let's see whether we can specify what an elevator does, what it knows of the outside world, and what it indicates to the outside world.

Suppose the elevator is in a building with floors numbered `0, 1, 2, ... maxfloor`. (Usually in Aus-tralia, floor 0 is called the ground floor.) Without requests, the elevator remains stationary. There are two sets of buttons the elevator responds to. If a request for service comes from the floor buttons (two per floor - up and down - except for the top and bottom floors which have only one), the elevator is activated to move to-wards that floor. When it reaches the floor, its doors open to accept the rider, who then presses the required floor on the internal button pad. Provided no further requests are made, the elevator then proceeds to the re-quested floor. In general the elevator will continue to travel in one specific direction if there are requests, either from floor buttons or internal buttons, for that direction. When it reaches its last requested floor, it will then attempt to satisfy requests in the opposite direction.
So here is an incomplete class description of the elevator:

```
const int UP = 1, DOWN = -1, STOPPED = 0;
class Elevator
{
     public:
     // constructor for number of floors - no default constructor
          Elevator(int);
          ~Elevator();         // destructor
          void Move();         // move one floor in current direction
     // report current status: floor, direction, door status
          void Status(int&, int&, bool&);

private:
          int maxfloor;        // highest numbered floor in the building
          int currentfloor;
          int currentdirection;
          bool dooropen;
          bool* floorup;       // array for up buttons on floor
          bool* floordown;     // array for down buttons
          bool* button;        // array of internal buttons
```

```
};
```

With all this information, the code for manipulating the elevator can be written.
However, we are still missing the communication events between the outside world and the elevator. We have to provide mechanisms for the buttons to be pressed. Normally these would be provided by other objects - the riders. But they would still need access to the buttons. So we will add some public member functions to access the buttons. (We could make the buttons directly accessible by placing them in the public section.) So we add

```
void DirectionSelect(int, int);
```

for a rider at the specified floor (first argument) to select up/down request (second argument using one of the named constants). And

```
void ChooseFloor(int);
```

for a rider in the elevator to select the required floor.
Of course the elevator will operate by going to the specified floor and open its doors even if the rider does not actually get on the elevator. The elevator has to handle turning off the buttons when a floor has been reached where the door will open, either to let riders get off or get on.

The header file `Elevator.h`, and a driver program `Assignment1.cpp` which provides the simulation of requests to the elevator so that you can test your class, are provided on the website.

Work:

The only C++ file you need to write code in is: `Elevator.cpp`