1001ICT Programming 1 2011-1
Project

School of Information and Communication Technology
Griffith University

May 22, 2011

Due	Friday, teaching week 14, 5.00 pm
Goals	In the project you get to work on some programming problems that are larger than can be dealt with in a 2 hour laboratory class, and with more attention paid to the presentation of the source code.
Marks	36 (making it possible to get 101 marks for the course!)
Robot	Elevator-NXT

1 Background

The theme for this project is provided by the Redding Elevator Company, which has been designing, manufacturing and installing elevators in buildings worldwide for many decades.

2 Exercises

All of the following programming exercises are to be implemented in MaSH, and should all have at least a `main()` method. That means that programs that don't use methods at all can not get full marks.

The exercises are not presented in any particular order. All exercises are independent. So if you have difficulty with one, there is no reason why you can't complete the rest.

Start work early. That way, if you get stuck, there are opportunities to ask questions, and you can make the best use of the time in which you have access to robots.

2.1 Program 1 (nxt, 6 marks)

Despite elevators being fully automatic for many decades, maintenance technicians routinely require manual control of them.

Write a program that enables the Elevator-NXT robot to be controlled manually using the NXT's front panel buttons (LEFT for door open, RIGHT for door close, ENTER for up, and ESCAPE for down), as demonstrated in this video.

Hints:

- Use button handlers to respond to both pressing and releasing the NXT's front panel buttons.

2.2 Program 2 (nxt, 6 marks)

The installation and maintenance technicians need a program to calibrate and test the mechanical operation of an elevator.

Write the program that takes the Elevator-NXT through a calibration and test sequence, as shown in this video.

To calibrate the elevator:

1. Raise the car as far as it goes to the top (the top floor).
2. Reset the rotation sensor.
3. Lower the car to the bottom (the bottom floor).

4. Record the rotation value. This value can be used to calculate the amount of rotation required to get to the middle two floors.

To test the elevator, make it go to floors 2, 3, 1, and 0 in that order. At each floor open the doors, wait for 1.5 seconds, and close the doors.

Hints:

- Use time to control how long the motor runs when opening and closing the doors.
- This problem is much easier to write if you make good use of procedures. Write a procedure for every basic action the elevator performs.
- The robot does not have to do more than one thing at a time. Threads are not necessary.

2.3 Program 3 (console, 6 marks)

The qualities that are most important to the Redding Elevator Company are safety and reliability. Both are achieved by high standards of design, manufacture and service. All elevators are serviced regularly, but even so, there *will* be unexpected failures. The long term goal is to minimise their frequency to enhance the quality reputation of the products and the company.

Each elevator's computerised controller maintains a log of the times it is put into service t_{up} and the times at which it fails t_{down}. This log is transmitted to the company to build a statistical profile of the reliability of each elevator and, collectively, of each elevator model.

The main metric used to describe reliability is the *mean time between failures* (MTBF), the sum of all of the failure times t_{down} minus the time it last came into service t_{up} divided by the number or failures.

$$\mathrm{MTBF} = \frac{\sum t_{\mathrm{down}} - t_{\mathrm{up}}}{\textit{number of failures}}$$

Write a program that can be used to read an extract from a log for one elevator and print the following statistics:

1. the total "uptime", that is the total time the elevator was in service, $\sum t_{\mathrm{down}} - t_{\mathrm{up}}$ (in years);
2. the number of failures; and
3. the MTBF (in years).

The log extract excludes scheduled times the elevator was taken out of service for maintenance. A typical log looks like this:

- Each line consists of a `U` to label a time as a t_{up} or a `D` to label a time as a t_{down}, followed by the time in seconds since the elevator was first put into service.
- The first line will always indicate t_{up} at 0 seconds.
- The last line is usually also a t_{up}, though it is possible that the log has been uploaded while the elevator is out of service, and therefore the last line is a t_{down}.
- A fairly new elevator that has not failed yet may have a log with only one line in it.

file:
`S00000001.txt`

```
U          0
D   15664610
U   15670184
D   30700326
U   30724536
D   37622324
U   37648485
D   60455362
U   60480874
D   79166234
U   79181602
D   95881395
U   95892167
```

Some more example log files: `S00000002.txt` (only one line); `S00000003.txt` (finishes with a t_{down}); `S00000004.txt` (longer).

For the purposes of the calculations, assume there are 365.25 days per year. Your program should read a log file from standard input, using input redirection.

Example output for these four data files:

```
$ java AnalyseOne < S00000001.txt
Total uptime = 3.035229865389003 (years)
Number of failures = 6
Mean time between failures = 0.5058716442315004 (years)
```

```
$ java AnalyseOne < S00000002.txt
No failures.
$ java AnalyseOne < S00000003.txt
Total uptime = 4.63133147007377 (years)
Number of failures = 3
Mean time between failures = 1.5437771566912568 (years)
$ java AnalyseOne < S00000004.txt
Total uptime = 13.717806709001954 (years)
Number of failures = 9
Mean time between failures = 1.5242007454446616 (years)
$
```

Note that the output is different for the file with no failures.

2.4 Program 4 (console, 6 marks)

The company is very interested in making a comparison of the reliability of the different models it has installed.

Its models are simply named model A, model B, etc. For each model there are many separate installations, identified by sequence number of 8 digits. Each elevator is uniquely identified by a serial number that starts with the model letter and ends with its sequence number.

Files have been created that aggregate the logs of many elevators:

- Each line starts with the serial number of an elevator.
- The rest of line is as per a log for one elevator:
  - A `U` labels a time as a t_{up}, or a `D` labels a time as a t_{down}, followed by the time in seconds since the elevator was first put into service.
  - The first line for each elavator will always indicate t_{up} at 0 seconds.
  - The last line for each elavator is usually also a t_{up}, though it is possible that the log has been uploaded while the elevator is out of service, and therefore the last line is a t_{down}.
  - A new elevator that has not failed yet may have one line for it.
- The file has been sorted by serial number and then time.
- Not all models might be represented in the file.
- A new model might not have suffered any failures yet.

The first 25 lines of file: `small.txt`

```
A00000001 U         0
A00000001 D  37902420
A00000001 U  37926167
A00000001 D  81482848
A00000001 U  81508838
A00000001 D 129373255
A00000001 U 129393392
A00000001 D 172752889
A00000001 U 172793133
A00000001 D 208904424
A00000001 U 208933554
A00000001 D 243425572
A00000001 U 243454010
A00000002 U         0
A00000003 U         0
A00000003 D  67943195
A00000003 U  67966385
A00000003 D 136952512
A00000003 U 136979618
A00000003 D 213738461
A00000003 U 213766673
A00000004 U         0
A00000004 D  23733729
A00000004 U  23772528
A00000004 D  51036894
```

Another data file exists with much more data: `big.txt`.

Write a program that reports the comparison for all models represented in the data file in a table, reporting: the number of each model installed; the total uptime, the total number of failures, and the overall MTBF.

Example output for these two data files:

```
$ java AnalyseMany < small.txt
                 total
                uptime     total      MTBF
model      #   (years)     fails   (years)
```

```
    A      7      23.7        17       1.39
    B      5      10.4        12       0.87
    D      4      17.5        14       1.25
    E      2       ---         0        ---
$ java AnalyseMany < big.txt
                  total
                 uptime      total      MTBF
model      #    (years)      fails    (years)
    A   2150     8370.6      12601      0.66
    B   3113    48216.5      96360      0.50
    C   2842    16980.9      23590      0.72
    D   2075    26007.3      40786      0.64
    E   3078    12670.8      13341      0.95
    F    734    15772.3      36384      0.43
    G   2371     3907.1       2524      1.55
    H   3720    48103.1      23993      2.00
    I   2938    28301.9      42149      0.67
    J   1355     7641.2      14022      0.54
$
```

Note how the results are reported when there have been no failures, and that to make the table neat, the format of the numbers has been controlled. This requires the latest version of the `console` environment. Instructions: Download the latest `console.mashenv` and place it in the same folder as your program.

2.5 Program 5 (graphics, 6 marks)

Write a program that plots a bar chart for comparing the MTBFs of different elevator models.

The data to be displayed is prepared in a file:

- Each line consists of the model's letter and the MTBF that has been measured for that file.
- There will always be at least one line of data in the file.
- There will always be at least one MTBF that is greater than zero.
- Don't assume the letters denoting the models are consecutive.

file: `bar.txt`

```
A       0.66
B       0.50
C       0.72
F       0.43
G       1.55
H       2.00
I       0.67
J       0.54
```

The data is to be read into the program using the input redirection method used in the previous two problems. For example:

```
$ java Bar < bar.txt
```

Sample output is shown in figure 1. Your output may be more elaborate or attractive, but it must convey the same data, including the numerical values and the units (years), as a bar chart.

Note tha the format of the numbers has been controlled. This requires the latest version of the `graphics` environment. Instructions: Download the latest `graphics.mashenv` and place it in the same folder as your program.

2.6 Program 6 (graphics, 6 marks)

Write a program that presents an animated graphical model of an elevator, that has at least four floors, with doors that open and close, and some indication that shows the car moving. It must be interactive, responding to mouse clicks that call it to a particular floor. The solution displayed in this movie is more elaborate than is required. It has a representation of all of the buttons available to users in the car (on the left) and on each floor (on the right). It also chooses to stop only at floors requested by passengers in the car or on floors, who wish to travel in the direction the elevator is already heading. A much simpler solution is all that is required. Stop at any floor clicked on. This should be considered more an exercise in animation, than logic.

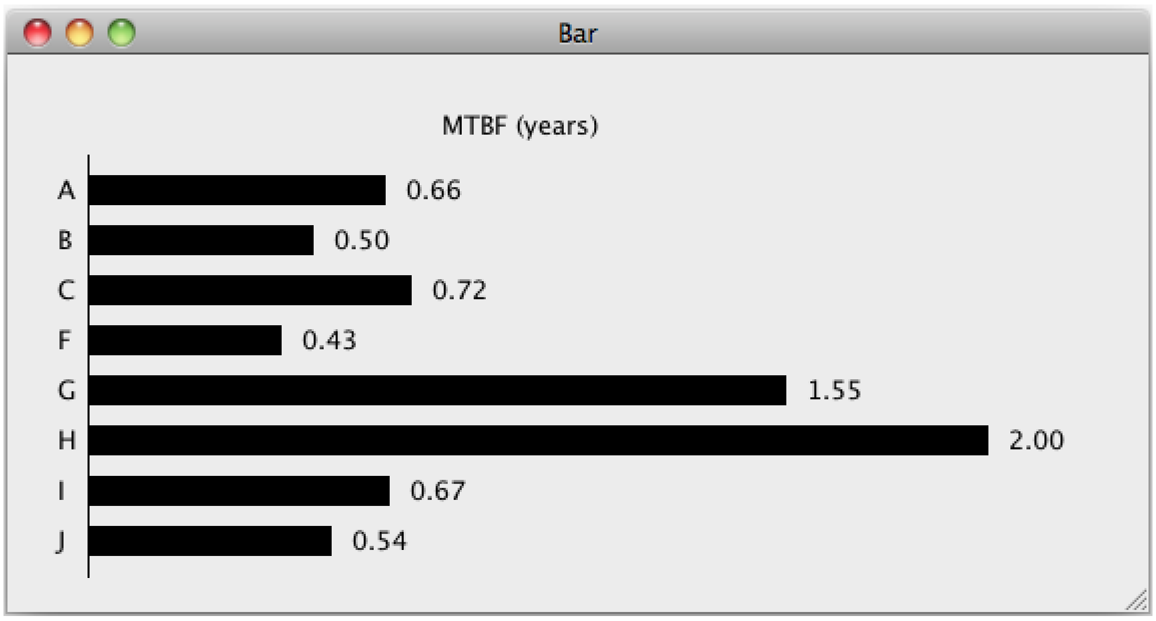

Figure 1: Bar chart comparing MTBFs for different models of elevator.

3 Submission instructions

Use the links on the project page of the course website to submit your solutions. You only need to supply your MaSH program code. We will supply all the data files for testing.

4 Marking criteria

General advice as to how the programs will be assessed:

- Programs that are complete and function according to the requirements will get more marks than programs that don't.
- Programs that don't work and/or are incomplete will still attract part marks. Make sure you submit anything you have done.
- Commenting is an important part of quality programming. In particular, we will look for these kinds of comments:
  - a header comment that identifies the programmer, the program and its purpose;
  - comments that describe the purpose of any global variables and constants; and
  - comments that describe the purpose of any methods (other than the standard ones like `main` or `paintWindow`).

  On the other hand it is possible to write too many comments. It is usually not necessary to write comments that just describe what individual statements do. Always assume that the reader of your program knows the programming language at least as well as you do.
- Programs that make good use of methods to organize and simplify programs will be rewarded. As will programs that have good choices of variable names.
- Points will be awarded for appropriate use of constants.