

# THE AUSTRALIAN NATIONAL UNIVERSITY

*First Semester 2005*

## *Final Lab Exam*

### **COMP2100/2500 Software Construction**

*Writing Period: 4 hours*

*Study Period: 15 minutes*

*Permitted Materials: One A4 page with notes on both sides*

#### **Instructions**

1. Answer all five questions.
2. The questions have equal value.
3. Marks for each part are indicated on the exam paper.
4. Answer all questions by creating or modifying files on the computer system as instructed. Do not move or rename files unless explicitly told to do so in the instructions.
5. During the exam you may not communicate with anyone, in any way. Any attempt to do so will be dealt with under the rules for Misconduct in Examinations.
6. You will have no internet access, however the following web resources have been copied onto the local file system:
  - The class web site is at  
`/dept/dcs/comp2100/public/www/index.html`.
  - The full Java JDK documentation is at  
`/dept/dcs/comp2100/public/www/jdk/index.html`.  
The API docs are at  
`/dept/dcs/comp2100/public/www/jdk/api/index.html`.
  - The full Java Tutorial is at  
`/dept/dcs/comp2100/public/www/tutorial/index.html`.  
The Swing Tutorial is at  
`/dept/dcs/comp2100/public/www/tutorial/uising/index.html`.

## QUESTION 1 [20 marks]

This is a repeat of Homework 10 (Making change).

Write a Java program called 'Change' that works out how to make change in coins for a given number of cents.

The program shall take exactly one command line argument, which should be an integer in the range 3–497. If the argument is out of range, the program should print an error message and stop.

If the argument is not a multiple of 5, the program shall first round it to the nearest multiple of 5.

The different types of coins you may use for making change are 5c, 10c, 20c, 50c, \$1 and \$2.

The program shall use a 'greedy' algorithm to choose between the many possible ways to make change. That is, it shall first use as many \$2 coins as it can, then as many \$1 coins as it can, then as many 50c pieces and so on.

For example:

```
[comp2100@partch]$ java Change 20
20c = 1 x 20c
[comp2100@partch]$ java Change 85
85c = 1 x 50c + 1 x 20c + 1 x 10c + 1 x 5c
[comp2100@partch]$ java Change 90
90c = 1 x 50c + 2 x 20c
[comp2100@partch]$ java Change 385
$3.85 = 1 x $2 + 1 x $1 + 1 x 50c + 1 x 20c + 1 x 10c + 1 x 5c
[comp2100@partch]$ java Change 497
$4.95 = 2 x $2 + 1 x 50c + 2 x 20c + 1 x 5c
[comp2100@partch]$ java Change 498
Error: argument 498 is too large.
```

Output should be formatted exactly as in the examples above.

*Hints:* Looks to me like you'll either need nested loops or recursion, or perhaps just one loop and some clever use of % and integer division. If I were you I'd probably store the values of the coins in order in an array or `Vector`, same with their string representations.

Make sure you test your program carefully. You will be penalised if your program crashes when we test it. You will be penalised severely if it fails to compile.

**Important:** Your solution *must* be in a file called `Change.java` in the `q1` directory.

[20 marks]

## QUESTION 2 [20 marks]

Bash scripts.

- (a) A lecturer wants to send email to all students who have not submitted an assignment. Your task is to write a Bash script that will prepare a list of the names and email addresses of all students who have not yet submitted the assignment.

You have two data files to work with.

The first is a plain text database file that lists the name, lab group and student ID of every student in the class. The file has one line for each student. The format of the lines is:

```
student-id      lab-group      Last-name, Other names
```

There is a single tab character between the student-ID and the lab group, and another tab character between the lab group and the family name. There is a comma followed by a space between the family name and the other names. An example file `database.txt` satisfying these conditions is in the `q2` directory.

The second file is a log file that has a line for every time a student submits the assignment. The format of the lines is:

```
student-id      time & date of submission      filename
```

There is a single TAB character between the student-ID and the submission time & date, and another TAB character between the time & date and the filename.

Write a Bash script that will print a comma-separated list of the names and email addresses of students who have *not* submitted any files, in a format suitable for cutting and pasting into an email program.

The script must be called 'rogues'. It must take the names of the database file and the log file as its two command line arguments.

Here is an example interaction with the script:

```
[comp2100@partch]$ cat database.txt
u4567890      wed09-11      BARNES, Ian
u2345678      tue13-15      WALKER, Richard
u1234567      wed11-13      JARSO, Tamiru
u3456789      tue17-19      KHOREV, Alexei
[comp2100@partch]$ cat log.txt
u1234567      13 June 2005 19:54:38  a2.jar
u4567890      13 June 2005 22:37:04  a2.jar
u1234567      14 June 2005 08:11:53  a2.jar
[comp2100@partch]$ ./rogues database.txt log.txt
Richard WALKER <u2345678@anu.edu.au>, Alexei KHOREV <u3456789@anu.edu.au>
```

Note that there must be no comma after the last name in the list.

*Hint:* You may find it useful to look up the manual pages for the `cut` and `grep` commands.

**Important:** Your solution *must* be in an executable file called `rogues` in the `q2` directory. I recommend that you test your script on the sample files *and* on other test data.

[15 marks]

(b) A little harder...

For loading marks into Streams, the lecturer needs a text file with comma-separated values (a CSV file) with one line for each student who has submitted the assignment.

Write another Bash script called `csv` that prepares a template file for the lecturer to enter marks into. Using the same two data files as command-line arguments, this script must write one line to the standard output for each student who *has* submitted the assignment.

The format of each output line must be as follows: first the student-ID, then two commas (the lecturer will enter the mark between them), then the student's name, another comma, and finally the date and time of the student's last submission, in the format year-month-day hour:minutes:seconds timezone-offset as in the example below.

The lines of output must be sorted into ascending order of student-ID.

With the same data in `database.txt` and `log.txt` as in part (a), here is an example interaction with the script:

```
[comp2100@partch]$ ./csv database.txt log.txt
u1234567,,Tamiru JARSO,2005-06-14 08:11:53 +1000
u4567890,,Ian BARNES,2005-06-13 22:37:04 +1000
```

Note that the spacing must be *exactly* as in the sample output above. In particular, there must be no space character between the comma and the start of the name.

You may rely on the lines in the log file being sorted in chronological order (earlier submissions will appear before later submissions in the file). You cannot rely on entries in the database file being in order of student-ID.

*Hint:* You may find it useful to look up the manual pages for the `sort` and `date` commands.

**Important:** Your solution *must* be in an executable file called `csv` in the `q2` directory. I recommend that you test your script on the sample files *and* on other test data.

[5 marks]

### QUESTION 3 [20 marks]

Graphical user interfaces with Java Swing.

Go to the `q3` directory, where you will find the source code for the final version of the Tally Counter application from lectures. In this question you will add some new features to it.

(a) Straightforward.

Add a new item in the File menu called 'About'. When selected, this should pop up a message dialog box with title 'Credits' and message "Counter Application by" followed by your name and student ID number, just like the screenshot below (except that you should put *your* name and number, not mine.)



*Hint:* You will probably want to look up how to create message dialogs in the API documentation and/or the Swing tutorial.

[6 marks]

(b) A little harder.

Add a new item in the Action menu called 'Set'. When selected, this should pop up a dialog box that asks for a number and presents a text field for the user to type a response in. When the user enters a non-negative integer and clicks OK, this should set the value of the counter to that number. If the user clicks Cancel or enters a negative integer or anything that is not an integer, there should be no action. Make sure the dialog box displays with the standard question-mark icon, as in the following screenshot.



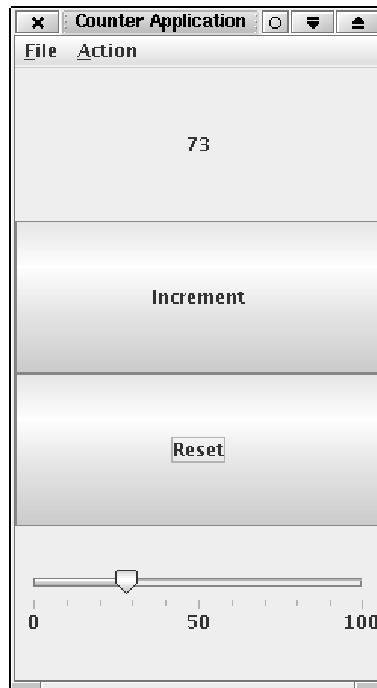
*Hints:* You will probably need to look this up in the Swing tutorial.

You will need to either modify class *Counter* to add a *set()* method, or use a combination of a call to *reset()* and a loop with repeated calls to *increment()* to actually set the value.

[6 marks]

(c) Harder.

Add a *slider* to the GUI so that the user can set the value of the counter using the mouse. The values available on the slider should run from 0 to 100. The slider should be horizontal and located below the buttons and label. Have the slider displayed with tick marks every 10 and numeric labels at 0, 50 and 100 as in the screenshot below.



*Hint:* You will need to look up class *JSlider* in the API docs and/or the Swing tutorial.

[5 marks]

(d) GUI design reflection

Discuss any user interface design issues raised by adding the slider in part (c). Write your answer in a plain text file called `Slider.txt` in the `q3` directory.

*Note:* You do not have to have completed part (c) to attempt this. Just looking at the screenshot above should be enough.

*Hint:* Think about the role of the slider in the Model-View-Controller architecture.

[3 marks]

**Important:** Your solutions to the first three parts must be in the Java source files in the `q3` directory. Since all three parts involve modifying the same code, take care that what you do for the later parts doesn't mess up what you have done for the earlier parts. I recommend that you test all three parts once you have finished.

Your solution for part (d) must be in a plain text file `Slider.txt` in the `q3` directory.

**QUESTION 4 [20 marks]**

Richard's question about stacks and trees and traversals etc.

**[20 marks]**

## QUESTION 5 [20 marks]

Ethics, Makefiles and the project

(a) Professional ethics.

Consider the following scenario:

Jane is a member of the IEEE Computer Society and the Association for Computing Machinery. She works as a software developer for an internet company that creates peer-to-peer file sharing software. Although it has legal uses, this software is mostly used to distribute illegal copies of music, feature films and other proprietary content. The Hollywood studios and major record labels have been trying to stop this by tracking network file transfers and using this evidence to take individual users of the software to court for copyright violation.

To avoid these attacks, developers are working on an innovative cyclic encryption of the data streams that will prevent people listening on the network from proving any illegal activity. Jane is asked to help with this work.

Unconfirmed reports suggest that the software is being used for secure, untraceable distribution of child pornography. Another report suggests that terrorist groups are using the software. Yet another report says that organised crime groups are using it to co-ordinate drug deals.

Two officers of the Federal Police approach Jane and ask for her assistance in providing a back-door to the software that will enable them to catch and convict users engaged in criminal activity. They say that they are not interested in copyright violations, only serious crimes. They threaten Jane with criminal charges if she refuses to co-operate or tells the other developers about this.

Discuss the issues raised by this scenario, in the light of the software professional's two main areas of professional responsibility:

- To your employer; and
- To the broader public interest.

Note that you are not asked to say what Jane should do, but rather to discuss the various factors she should take into account in making her decision.

**Important:** Write your answer in plain text in a file called `Ethics.txt` in the `q5` directory.

[6 marks]

(b) Makefiles.

This question is very similar to exercises 1–4 of Lab 5, but not identical.

Go to the `q5` directory. In there you will find the following source files:

- `sort.c` A simple insertion sort routine (unchanged from Lab 5).
- `sort.h` A header file for the sort module (again unchanged from Lab 5).
- `harness.c` A simple test harness for the sort routine (simpler than the one in the lab).
- `tests.txt` Test cases for the harness, one per line, each ending with `-1` (unchanged from Lab 5).
- `tester` A simple generic test script that runs its first command-line argument on each line of its second command-line argument in turn, and writes the results to the standard output. (This was not in the lab.)
- `expected.txt` The list of expected output corresponding to `tests.txt`. This is the same as the one in the lab except that the numbering starts at 1 rather than 0.
- `reporter` A test script that takes the expected and actual output files as its command-line arguments and writes a test report to the standard output.

Write a Makefile that automates the process of compiling and testing this program. The Makefile is responsible for keeping the following files up to date:

- The object files `sort.o` and `harness.o` that result from compiling the corresponding C files.
- The executable file `harness` that results from linking the two object files.
- A plain text file `actual.txt` that is the redirected output of the `tester` script when run on `harness` and `tests.txt`.
- A plain text file `report.txt` that is the redirected output of the `reporter` script when run on `expected.txt` and `actual.txt`.

When the user types ‘make’ at the command line, this Makefile must bring `report.txt` up to date with the minimum amount of work possible. (In other words, it should not rebuild any files that are still up to date.)

Add a phony target ‘clean’ that removes all the generated files, leaving only the original source files. Be careful that this doesn’t delete your solutions to parts (a) and (c).

**Important:** Your answer must be in a file called `Makefile` in the `q5` directory.

[8 marks]

(c) Reflection on the project

I mentioned a few times in lectures that the project software is only a prototype designed for educational purposes, and that there are some aspects of its design that could be improved. Suppose you were given the task of redesigning and rebuilding the project software (the endpoint of Assignment 2: the graphical interface document browser and tree-oriented editor) with a view to releasing it as commercial software.

What changes would you make to the design? Why?

**Important:** Write your answer in plain text in a file called `Project.txt` in the `q5` directory.

**[6 marks]**

**Checklist:**

Before you leave the exam, make sure that all your answers are in the right files in the right locations.

- Your answers to Question 1 part (a) must be in a Java source code file `q1/Change.java`.
- Your answer to Question 2 part (a) must be in the Bash script `q2/rogues`.
- Your answer to Question 2 part (b) must be in the Bash script `q2/csv`.
- Your answer to Question 3 parts (a), (b) and (c) must be in the various Java source files in the `q3` directory.
- Your answer to Question 3 part (d) must be in a plain text file `q3/Slider.txt`.
- Your answer to Question 4 must be in...?
- Your answer to Question 5 part (a) must be in a plain text file `q5/Ethics.txt`.
- Your answer to Question 5 part (b) must be in `q5/Makefile`.
- Your answer to Question 5 part (c) must be in `q5/Project.txt`.

5 questions, total marks 100