The COMP16121 Laboratories

John Latham

September 2018

1 10 exercises, in 14 laboratory sessions

COMP16121 is assessed by laboratory and by examination, each counting 50%. There are 14 laboratory sessions, forming 10 exercises, one per week, each of either 1 or 2 sessions duration, depending on the A/B week structure. The marks associated with each exercise are weighted equally per session. Thus, a 1 session laboratory exercise counts as \( \frac{1}{14} \) (7.14%) of the whole COMP16121 laboratory mark, and a 2 session exercise counts twice as much.

However, we must reserve the right to adjust this weighting for good reason if, for example, a particular exercise goes badly. We may even adjust the number of sessions for good reason.

In addition, there are two exercises – 8o and 10o – which are optional. This means the following.

- If you do not do an optional exercise, it does not count.
- If you do an optional exercise, but get a lower average than your other marks, it is not counted.
- If the marks obtained for an optional exercise would increase your overall average, then they are counted. Each optional exercise counts as one more session, and so if both are counted, then each single session exercise is worth \( \frac{1}{16} \) of the whole COMP16121 laboratory mark.
- The optional exercises are done in parallel with the non-optional ones, and have a deadline which is the same as the last non-optional exercise.

2 Deadlines

The deadline scheme is described in the general section of the laboratory manual. However, please note that the deadline for the first two exercises cannot be extended. This is because they are largely introductory, involve a little less work than the exercises that follow, and to get you off to a good start with pacing your work.

For the other exercises, you can request for an extension for finishing off, and this will be granted only if you have nearly completed the work.
3 Task structured exercises

Each exercise contains a number of tasks, or ‘tablets’, where (in general) each task is associated with an example section from the book. Typically, the number of tasks per laboratory session will be around 6, but this will vary depending on the topics being learned in the tasks.

Each task is marked out of 20, so a typical exercise will be marked out of 120 or 240! These numbers may seem rather large, but the priority is to make it possible to mark each task reasonably quickly, using a one-size-fits-all marking scheme. Also, the inevitable variations in marking between different markers have less impact on the overall total if the range of possible marks is large. (E.g. a subjective difference of 6 marks out of 120 has little impact on the final sum).

4 Task marking scheme

Each task will be marked using the following scheme.

<table>
<thead>
<tr>
<th>Category</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness and correctness</td>
<td>8</td>
</tr>
<tr>
<td>Code layout</td>
<td>2</td>
</tr>
<tr>
<td>Code comments</td>
<td>2</td>
</tr>
<tr>
<td>Code identifiers</td>
<td>2</td>
</tr>
<tr>
<td>General code quality</td>
<td>2</td>
</tr>
<tr>
<td>Logbook entries</td>
<td>2</td>
</tr>
<tr>
<td>Student understanding</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>20</td>
</tr>
</tbody>
</table>

For ‘Completeness and correctness’, the mark will be chosen from the following matrix.

<table>
<thead>
<tr>
<th></th>
<th>Fully correct</th>
<th>Almost correct</th>
<th>Mostly correct</th>
<th>Half correct</th>
<th>Mostly wrong</th>
<th>All wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully completed</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Almost completed</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>About half completed</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Much evidence of progress</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Slight evidence of progress</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Noise or all irrelevant</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that correctness is not just about the program producing the correct results, but also about it working in the correct way as required by the task.

In the early tasks (before code commenting has been covered), the ‘Code comments’ criteria does not apply, and so you get free marks! ;-)
have received too much help from friends then your understanding may well be poor. This process of spot check assessment may give rise to the marker being suspicious that the work is not really your own, in which case you will be referred for investigation by the laboratory supervisor and/or laboratory manager.

If you have programmed before, you might wish to use some Java concepts that have not been covered in the course by the point at which a task occurs. Please check this is okay before doing so, because it may undermine the aims of the task. (E.g. using the API sort method when you were asked to create your own, or using nested loops when you were asked to write code ‘the long way round’.)

## 5 Tasks per exercise

The tasks in the table below refer to the coursework associated with a section number in the course handouts/book.

The table might be subject to change as the course proceeds. Please check for the latest version on the COMP16121 web pages.

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.6, 1.7, 2.2, 2.3, 2.4, 2.5</td>
</tr>
<tr>
<td>2</td>
<td>2.6, 2.7, 2.8, 2.9, 3.2, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9</td>
</tr>
<tr>
<td>3</td>
<td>4.2, 4.3, 4.4, 5.2, 5.3, 5.5</td>
</tr>
<tr>
<td>4</td>
<td>5.6, 5.7, 5.8, 5.9, 5.10, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7</td>
</tr>
<tr>
<td>5</td>
<td>8.2, 8.3, 8.4, 8.5</td>
</tr>
<tr>
<td>6</td>
<td>8.6, 8.7, 8.8, 10.2, 10.3, 10.4</td>
</tr>
<tr>
<td>7</td>
<td>10.5, 11.2</td>
</tr>
<tr>
<td>8a</td>
<td>11.3, 12.2</td>
</tr>
<tr>
<td>8b</td>
<td>13.2, 13.3, 13.4, 13.5</td>
</tr>
<tr>
<td>8o</td>
<td>12.5</td>
</tr>
<tr>
<td>9</td>
<td>13.7, 13.8, 13.9, 13.12</td>
</tr>
<tr>
<td>10</td>
<td>14.2, 14.3, 14.4, 14.5</td>
</tr>
<tr>
<td>10o</td>
<td>14.6, 14.7</td>
</tr>
</tbody>
</table>

## 6 Organising your work

For each *each task*, you are *required* to proceed in the following steps.

1. Logbook preparation, including the design of test data as appropriate. (See more information about this in section 7.)
2. Creating the task directory as described below.
3. Implementing the task test script as described below.
4. Designing the solution to the task, as appropriate, in your logbook.
5. Implementing the solution.
6. Testing it with your pre-prepared script.
7. Recording conclusions in your logbook.

If you undertake these steps in the wrong order, for example ‘faking’ your logbook preparation after you have written the code, or writing the test script after the code, we shall consider that you are cheating!

Please try to have the handout/book closed when performing steps 4 and 5. For the early tasks, if the associated example from the book is in front of you, you are reducing the challenge to little more than a typing exercise. Sure, this will make the tasks easier, but you will learn much less and then when we get to the tasks that are not so similar to the book example, you will crash because you have learned nothing. Please do not let that happen.

And when the whole set of tasks for an exercise is completed:

1. Implementing the exercise test script as described below.
2. Submitting your work with the submit command. (See the explanation of submit in the first section of this laboratory manual.)
3. Obtaining hard copy using the labprint command.
4. Getting feedback and having it marked, face-to-face with a teaching assistant, at your earliest opportunity.

### 6.1 Exercise and task directory structure

In order for submit (etc.) and labprint to function properly, it is imperative that you store your work in the correct place.

Create a directory called COMP16121 in your home directory. (Note the use of capital letters in this name.) Then, inside this create a directory for each exercise, called ex1, ex2, etc.. There are 10 exercises plus 2 optional exercises. Within each exercise directory, create a directory for each task associated with that exercise – you should probably do this as you start each task. The task directories must be called task (lower case) followed by the number of the task, e.g. task1.6.

```
/home/a11111aa/COMP16121
   |-- ex1
   |    |-- task1.6/...
   |    |-- task1.7/...
   |    |-- task2.2/...
   |    |-- task2.3/...
   |    |-- task2.4/...
   |    |-- task2.5/...
   |    |-- ex2
   |         |-- task2.6/...
```

Develop the files needed for each task within that task’s directory. Be careful to ensure that these files have the correct name – otherwise submit and labprint will not be able to find them, and so they won’t get marked!

### 6.2 Running submit and labprint

The programs submit (etc.) and labprint should be run when the appropriate exercise directory is the current working directory. For example

```bash
cd ~/COMP16121/ex1
submit
labprint
```

will submit your work and produce a hard copy of the tasks for exercise 1. You should run submit as soon as you have finished an exercise (i.e. not upon completion of each task). submit records the time of completion and archives your work for plagiarism/collaboration detection. Make sure the command has confirmed that your work has been successfully submitted to the archive.

*Note:* if submit tells you that an ‘optional’ file is missing, think hard before confirming you wish to continue. The file may be ‘optional’ to submit (e.g. you might not have done that task), but perhaps not ‘optional’ to you – maybe you got the name wrong?

You should run labprint when you are next in the school after submitting (e.g. straight away if you are already in the school).

At your next opportunity during laboratory time, you should write your first name and the last two characters (digits) of the name of the machine you are logged into on the marking queue on the white board. A teaching assistant will thus find you and together you will proceed with the following feedback and marking process.

1. You will run submit-diff to show there are no differences in your work since you submitted it.
   - If there are significant differences, you will run submit-again. This will make your submission late, unless it is still before the deadline (or extended deadline as appropriate).

2. You and the teaching assistant undertake analysis of, and discussion about, your work. This kind of feedback is the highest quality you could have, compared with comments written on your work while you are not present. Ask your marker to explain anything you do not understand. Please take heed of the feedback, rather than only want to know your marks.
3. You will run `submit-mark` to submit your mark. This will require the mark and a **marking token**, given to you by the teaching assistant.

4. You will ask the teaching assistant to check you have entered these correctly and he or she will confirm to the command.

5. **Important:** make sure you wait for confirmation on the screen that your mark has been submitted – read the output carefully in case you have made an error. (A common stupidity is to close the terminal straight after confirming the details, thus killing the program before it has had chance to report your mark to the server!) If, for whatever reason (i.e. user or system error), you do not get confirmation of the mark being recorded, then you will need to run `submit-mark` and try again.

Remember that if you take an unreasonable period of time after submitting, to get your work marked, we will remove your submission from the archive.

### 6.3 Organising your tests ready for feedback and marking

As you **start** the development of each task, you should create, in the **task** directory, a **shell script** called `run-tests`. This should start with the following two lines.

```
#!/bin/bash
set -v
```

These are to be followed by `java` commands to run your program with the test data you **have already designed in your logbook**, or just run it once if there is no test data. E.g.:

```
java ProgramName test data one
java ProgramName test data two
java ProgramName test data three
```

**where** `ProgramName` is the name of the program for this task, and `test data one`, etc., are the command line argument tests you have designed. You will need to make the file executable.

For the first task, there is no test data, so your `run-tests` script will contain the following.

```
#!/bin/bash
set -v
java HelloWorld
```

**Your run-tests scripts serve two purposes.**

1. You only need to type your test data once, into the script **before** you write any Java code. This means as soon as your Java code compiles, you can test it by merely running `./run-tests` which saves a lot of typing when, for later tasks, you may be wanting to run your program many times with different test data.

2. It makes the feedback and marking process run smoothly – together with the script described below it removes the need for you to be changing directory into the various task sub-directories as you demonstrate your programs and typing lots of test data. We will refuse to mark your work if you have not set up these scripts.
After writing your run-tests script for a task, you can design (see section 7) and develop your solution to the task.

When you have completed the exercise, but before submission, you should create another shell script called run-all-tests, in the exercise directory. This should contain the following text. (Type this in carefully.)

```bash
#!/bin/bash
for task in task* 
  do
    echo "========================================"
    echo "Task $task"
    echo "----------------------------------------"
    (cd $task; ./run-tests) 2>&1
  done | less
```

Make this file executable as well. This script can then be used to see your programs being run with your test data, and you should not need to spend time typing test data during the feedback and marking. You should check your script works before you submit your exercise.

**Tip:** as this is the same script for each exercise, you don’t need to retype it each time.

7 Logbook

Every task requires you to write entries in your logbook before, during and after any programming work for the task. Each entry should have the date and time recorded and what task it is for. Your logbook is assessed, and will not be accepted if you have not recorded dates, times and task numbers.

For every task, you should have the following entries.

- Record the date and time when you started the task.
- Some tasks ask you to record specific analysis.
- Test data (as appropriate), along with expected results, must be designed and written in your logbook in advance of designing the solution.
- Your design of the solution must be written in your logbook in advance of implementing it. How much you write will depend on how hard the task is and how much previous programming experience you have. Thus, if you think it will be trivial for you then you may write just that.
- Record any general comments or observations you have about the task as they occur. Perhaps you thought the task would be trivial for you, but it turned out not to be. Perhaps there was something else that surprised you or didn’t happen as you expected it to. Perhaps your test data expected results were wrong. Record anything at all that might be interesting to note!
- Record the date and time you finished the task.
The logbook entries are not there just as hoops for you to jump through. If you write your logbook entries in the wrong order (e.g. faking a design after you have completed the corresponding implementation, or not designing test data in advance of design) then you will be deemed to have attempted to cheat, by only pretending to follow the process in order to get the marks. In this case you will lose all the logbook marks for the entire exercise, as a minimum!

Finally, your logbook doesn’t need to be neat! Some people write their design work on paper, cross bits out, etc., and only copy it into the logbook when they have figured it out. Don’t do that! Part of the point of the logbook is that you get to keep all that scribble and crossed out stuff, forever – so you, and perhaps your tutor, can monitor your progress and your thought processes.

If you have programmed before then please do not resent our requirement for you to use a logbook. We are not asking you to write anything that is not useful to you, so in the early tasks please do not write designs if these are unnecessary for you (just write that fact instead, but of course do include any surprises you get). However, designing your test data in advance of implementation is an important discipline, as is recording the start and end of tasks.

Whether or not you have programmed before please note the teaching assistants have been instructed not to accept any logbook entries that do not have the date and time on them, and to report ones that have apparently been faked to the laboratory supervisor, for consideration that cheating is occurring (faking logbook entries is cheating).

You must bring your logbook to the lab at all times, otherwise you would not be able to make progress without cheating!

8 Teaching tool – Java animator

You are encouraged to make appropriate use of an experimental teaching tool which is being developed by John Latham, to animate your Java programs. This may well help you get a better grasp of Java concepts, by allowing you to see what your programs are doing as they run. It may also be an aid to helping you figure out why they do not work – i.e. to help you debug your code.

Using the Java animator is simple. Instead of compiling your source file(s) using the javac command, you use the command janc – short for Java ANimator Compiler. Then, instead of running your programs using the java command, you use the command jan. The rest should be obvious – a window pops up displaying your program source code along with a history of the program’s effect. You can single step each instruction, or allow it to run at various speeds, or even make it step backward! You can also set break points by clicking on a line of code.

This is a new tool, under development, and experimental. John Latham welcomes your (constructive) feedback and bug reports.

9 Working alone

Programming is an instance of design, and as such is a mostly creative activity. You can only learn creative skills by practising them. You can only be taught concepts about the raw material
building blocks (e.g. a programming language), and perhaps some tips about good approach to
organising your thoughts. The rest you must learn yourself.

The COMP16121 exercises are designed to be undertaken individually. It is most important that
you work on your own, obtaining only very basic help from your fellow students. If you work
together, then you will seriously undermine your own learning. As you know, such behaviour is
regarded by us as collaboration or plagiarism, and you and your co-workers will lose all marks
when we catch you.

If you see someone else’s code, or detailed design, before you have completed your own, then
you have cheated. Worse still, they have put your learning in jeopardy by showing it to you: that
is not really the act of a friend.

If you do want to work together on some exercises, then please do, but not with the laboratory
exercises!

However if you treat the laboratory exercises in this way, any benefits of working together are far
outweighed by the disadvantages to you. There is a good chance that you cannot see this fact
yet, particularly if you are used to an environment where you have always worked with others.
You should accept that the kind of learning you are now being asked to achieve is different in
many ways to what you may be used to, and so the way you learn must also be different.

9.1 The bottom line

Our rules on collaboration and plagiarism are there to deter you from trying to learn in the wrong
way, in case you don’t yet see why you shouldn’t work together, and thus protect you by brute
force if necessary! Be warned: we mean it. Unfortunately, the collaboration/plagiarism detection
process is an ongoing one, and it is not always possible to detect collaboration/plagiarism
immediately after work is handed in. Sometimes it is only detected much later, perhaps near to
the end of the year, when it is too late for those caught to do extra work in order to pass. Sadly,
there are sometimes a few students who do not make it to the second year, apparently because
they thought they could get away with collaboration/plagiarism, and ended up at the last minute
with failed laboratories. I personally wish I could be better at convincing people how serious this
issue is.

Worried? Please don’t be. If you never ever see (or hear...) the code, or detailed pseudo code
or design of another student, before you complete your own work; nor help another student at
such level of detail, then you have nothing to worry about.

10 Meet the deadlines:
don’t be obsessed about getting full marks

We intend that none of you should need to miss any of the deadlines, unless you have some
special circumstances like illness. It is true that you may need to work hard on the exercises to
meet the deadlines, and you should be prepared to do this: no pain, no gain!
However, due to the extremely wide range of current abilities and previous experiences, we do
not expect every student to do every task in every exercise. So, you should work hard, get as
far as you can, and stop when you run out of time or when you complete the whole exercise.
That way, your learning will be timely – to tie in with the lectures, and at a depth that reflects
your growing ability. Moreover, you will not get depressed about falling behind.

On the other hand, you might get depressed about not getting the highest marks available, if
you let yourself. You must accept this fact: half of our students have suddenly moved from being
"near the top of the class" to being "below average", simply by coming to University. You should
bear in mind that a good student is not somebody who gets high marks, but somebody who
works hard and knows at any time what their ability is. That way, the good student can improve.

Even if you don't buy this argument yet, there is a simpler one which you will. Given that the
penalty for missing a single deadline is probably in effect to lose all the marks you get for it, you
are nuts if you decide to miss a deadline in order to get more marks for it! ;-)