CSSE2310/7231 — "Computer Systems Principles and Programming"

"And so it begins."
— Kosh Naranek

"Computers are full of lies and disappointment."
–dlg

CSSE2310 — Me

I'm Joel Fenwick (joelfenwick@uq.edu.au)

- "Joel" for preference, "Dr Fenwick" or "Dr Joel" if you must
- PhD in computer science
- Some commercial programming experience
- UQ academic employed in a programming heavy role for ≈ 10 years

I'm not

- An engineer
- A professor

Misc

Some of the slides for this course have ancestors in slides by:

- E.N. Elnozahy, U Texas
- R. Chandra, Cornell University

Overview

- Rule 0
- Cheating — Don't
- Assessment
- Contact sessions
- Help
- What's it about?
- Challenging aspects & rationale

Rule 0

If something is happening that:

- you don't understand;
- that seems to contradict what we've already told you;
- ...

Ask questions!

The cost of not asking is often higher.

Cheating and collusion

All code which you submit for assessment must be, either

- supplied by teaching staff **for this run** of the course.
  OR
- written entirely individually by you.

All code you submit will be checked for collusion and plagiarism (including but not limited to using M.O.S.S.[1]). If similarity is found, a school investigation may be started (these are not fun).
Seek help from course staff in plenty of time if you are having difficulty with assessment.
Be very careful when seeking outside help — outside tutors may facilitate cheating (and get you caught) rather than your learning.

[1] Not the `moss` computer

Misconduct — Things not to do

1. Do not show your code to other students
2. Do not look at other student's code
3. Do not use code given to you in other semesters or courses
4. Do not **try** to get other people to write code for you
5. Do not store your code in any online repository which **could** be accessible to others. (This will be deemed as if you have broken Item 1).
6. Do not start a few days before the deadline and then claim that cheating was the only way to get it done in time.

Assessment

The ECP is the authoritative version of rules, deadlines etc.

A_1	Assignment 1	Mon 23/3/20 18:00
A_2	Assignment 2	Mon 6/4/20 18:00
E_m	Mid semester exam	Thu 23/4/20 18:00
A_3	Assignment 3	Fri 8/5/20 20:00
A_4	Assignment 4	Fri 29/5/20 20:00
E_f	Final Exam	*Exam period*

$$\text{Mark} = \sqrt{\frac{\sum A_i}{4} \times \max \left\{ \begin{array}{l} 0.15 \cdot E_m + 0.85 \cdot E_f \\ 0.30 \cdot E_m + 0.70 \cdot E_f \end{array} \right\}}$$

Assessment

- 40% hurdle on the overall exam component
- No late submissions for assignments
  - We will mark the last "on-time" commit.
- Serious sickness or misadventure?.
  - 2–3 days extension (varies with assignment).
  - Online applications only.
  - See ECP.

Contact sessions

All students should be enrolled in a single contact session (once a week).
Contacts will generally contain some mixture of:

- Theory questions
- Instruction on tools
- Exercises
- **Assignment help**

You can go to as many as you like, but people enrolled have priority for seats.

Information and help

- Discussion board
  - Questions which are not specific to you.
  - Eg: about lectures or content or assignment requirements.
- `csse2310@helpdesk.eait.uq.edu.au`
  - Questions / issues that are specific to you.
  - Eg: need X remarked or This code[2] caused the tool to crash.
- `joelfenwick@uq.edu.au`
  - Things not fitting into the above

[2] Don't ever post your assignment code to the discussion board

Accountability

Chain of command (try not to jump levels):

- Tutors work for me.
- I work for ITEE Deputy Head (teaching and learning).
- DHOS works for HOS.

What is 2310 about?

"How programs on a modern general purpose computer interact with the system and with each other"

Goals:

1. Cover technical material
2. Make you better programmers:
   - Practice
   - Understanding material in #1
   - Using material in #1 **in context**.

How?

C programs and the Linux operating system with `svn` version control.

Compiler: `gcc`
Target: `moss.labs.eait.uq.edu.au`

Your code must compile and run on moss.

On your own system?

1. You can access moss via ssh/putty.
2. Make a VM and install linux in it.
3. Install and use tools directly on your host OS.

The only option we directly support is #1.

"Challenges"

Some (potentially) challenging aspects of the course:

- Code must compile (Essays vs Programs)
- Writing whole programs
- Writing larger programs
- Automated testing

Essays . . .

How not to write an essay:

1. Write "enough" words
2. . . .
3. cleanup / edit if there is time

The target of an essay is a human mind. Minds are wonderful things, they can (generally) handle problems like:

- contradiction
- missing steps
- spelling and grammar errors

We expect our target to compensate for small errors.

. . . vs Programs

However, programs are targetted at computers. Computers do not handle those problems well.

If your code does not compile, the computer can't do anything with it. Regardless of how "small" the error is.

Note: When people say that "programs are written for people", there is an implied ". . . as well as computers" at the end.

So, in this course code which *does not compile* will get zero.
Note: a program could compile and not do everything correctly.

Rationale: Whole programs

- The blank page/file/program/. . . is intimidating.
  - This is understandable but programmers need to be able to move past that.
- "All of that is mine" is rewarding.
- To deal with the mindset that programming is "somebody else's job, I just fill in a few stubs".

Rationale: Larger programs (1/4)

Remember: Programming is a practical discipline:

- theory is good
- but you won't really improve if you aren't doing it.

Rationale: Larger programs (2/4)

1. Even the assignments in this course are quite small. For example, in this course you will be using:

A shell (eg `bash`)	193267 LOC[3]
Encrypted connection (eg `ssh`)	128242 LOC

2. To get better at writing larger programs you need to write larger programs.

[3] LOC=lines of code, ignores blank lines and comments

Rationale: Larger programs (3/4)

3. Professionally, you need to be the expert on your own work. In general:
   - Tutors can help you learn to debug
   - Tutors may be able to suggest common types of errors
   - You wrote it, somebody else is not going to instantly know what is wrong with it.
4. In smaller programs, you may have avoided good practices. Do you really need to comment, name things sensibly, . . . ?
5. Larger programs force you to get better at debugging.

Rationale: Larger programs (4/4)

6. Programs need to include context
   - eg: use a network connection not just set one up

Larger programs — Objection

"It's hard to write that much code at the last minute."

In this course, assignments are *learning activities*.

- A lot of what you learn in this course will come from coding.
- You are supposed to be working on them over a period of weeks.
  - Time to realise problems (debugging)
  - Time to ask questions
  - Time to replace code which is causing problems.
  - Time to take a break when you go "on tilt".

Rationale: Automated testing

Functionality is tested by running your programs against test cases and checking that the output matches exactly.

Why?

1. Much of the time, code (programs/libraries/...) is used by other code, not people.
   - Computers are not good at flexibility / inconsistancy
2. Allows for much faster marking
   - get feedback and results back faster
   - if there are problems, remarking is faster
3. Allows for more consistant marking

Style is marked by (tool-assisted) humans.

Tips (1/2)

- Keep a notebook or logfile of your questions as the occur to you
  - More efficient use of tutor time
- Be willing to try things out
- Use test scripts when we make them available.
- Allow time to deal with problems.
- Test and debug as you go.
  - It's easier to remove eggshell before you've mixed it into the cake.
  - Write a test program if necessary.

Tips (2/2)

- Don't be more than an hour from working code.
  - Comment out blocks of code
  - Get a working version from version control.
- Commit *working* code when you've made a reasonable improvement.
  - Only commit working code.