Course Staff

- Convenor/Lecturer: Andrew Taylor
- Course Admin: Jashank Jeremy
- Tutors:
  - Alexandra Alexus
  - Annie Liu
  - Aydin Itil
  - Christine Wu
  - Clifford Sesel
  - Dongzhu Huang
  - Dylan Brotherston
  - Emily Chen
  - Jashank Jeremy
  - Lance Young
  - Mariya Shmalko
  - Michael Manansala
  - Nhat Nguyen
  - Oliver Scott
  - Peter Kerr
  - Ryan Fallah
  - Sasha Varman
  - Selina Chua
  - Yue Wu
  - Yunzhu Duan
Students in this course have completed:

- COMP1511 or COMP1911

Everyone has learned *fundamental C programming*

COMP1511 also studied *linked lists, ADTs*

Since not everyone has seen these, we won’t use them

For this week . . .

- review/strengthen C knowledge
- introduce/revise core data structures (stacks, queues)
Course Goals

COMP1511/1911 . . .
• gets you thinking like a *programmer*
• solving problems by developing programs
• expressing your solution in the C language

COMP1521 . . .
• gets you thinking like a *systems programmer*
• with a deep understanding of run-time behaviour
• and better able to reason about your C programs

**Note:** these are *not* the same goals as COMP2121
COMP1511/1911 vs COMP1521

COMP1511/1911 . . .
COMP1511/1911 vs COMP1521

COMP1521
COMP1511/1911 vs COMP1521

or maybe ...
Course Context
Themes

Major themes . . .

1. software components of modern computer systems
2. how C programs execute (at the machine level)
3. how to write (MIPS) assembly language
4. Unix/Linux system-level programming
5. how operating systems and networks are structured
6. introduction to concurrency, concurrent programming

Goal: you are able to understand execution of software in detail
There is no prescribed textbook
Material has been drawn from:

- "Introduction to Computing Systems: from bits and gates to C and beyond", Patt and Patel
- "From NAND to Tetris: Building a modern computer system from first principles", Nisan and Schocken
- COMP2121 Course Web Site, Parameswaran and Guo

Note: always give credit to your sources
Recommended reference . . .

- *Computer Systems: A Programmer’s Perspective*,
  Bryant and O’Halloren

- covers most topics, and quite well
- but uses a different machine code

Available in UNSW Bookshop
Textbooks . . .
  • cover some parts of syllabus in great detail
Course Notes . . .
  • cover whole syllabus, but not in great detail
Lecture Slides . . .
  • summarize Course Notes and add exercises
Prac work based on *Linux* tools

- all tools available on the *CSE lab machines*
- can use *VLAB* to connect to CSE from home

Compilers: *dcc* on CSE machines (clang or gcc elsewhere)
Assembly language: MIPS on *QtSpim* (also *Xspim* on CSE)
Use your own favourite text editor
Other tools: *make*, *gdb*, *man*, *bc -l*
Learn to love the *shell* and command-line ... very useful
Lectures

- Wednesday 11:00 - 13:00
- Thursday 09:00 - 11:00

Feel free to ask questions, but otherwise quiet so others can hear. Recordings & slides from streams will be posted to class web page. Students in web stream free to attend lectures if there are seats. In past terms, plenty of seats.
Lectures will:

- present a brief overview of theory
- focus on practical demonstrations of coding
- demonstrate problem-solving (testing, debugging)
- Lecture slides available on the web before lecture.
- Feel free to ask questions, but otherwise *quiet please*.
- Lectures recorded and linked to home page.
Tutorials aim to:

- clarify any problems with lecture material
- work through problems related to lecture topics
- give practice with design skills  *(think before coding)*
- Tutorials and labs start in week 1.
- Tutorial questions available on the web the week before.
- Tutorial answers available on the web after the week’s last tutorial.
- Use tutorials to discuss *how* solutions were reached.
Tutorials

- Attempt the problems yourself beforehand
- Your tutor may ask for your attempt to start a discussion.
- Do *not* keep quiet in tutorials . . . talk, discuss, . . .
- Don’t let you tutor go too fast (interact!)
- Extra tute questions each week for revision.
Lab Classes

Each tutorial is followed by a two-hour lab class.

- Lab exercises mostly small coding tasks.
- Lab exercise build skills need for assignments & exam.
- Lab exercises done in pairs.
- Tutors will form pairs and reorganize them every 4 weeks.
- Both members of pair must submit with give
- Automarked (with partial marks)
- 10% of final mark.
- Labs often include individual challenge exercises.
- Challenge exercises may be silly, confusing, or impossibly difficult.
- Full marks possible without completing any challenge exercises
Weekly Tests

• programming tests or multiple-choice quizzes in weeks 3–10
• immediate reality-check on your progress.
• done in your own time under self-enforced exam conditions.
• time limit of 1 hour
• automarked (with partial marks)
• contribute 9% of final mark.
• best 6 of 8 tests used to calculate the 9%
• any violation of the test conditions ⇒ zero for whole component
Assignments

• Ass1: Assembly Language, weeks 4-7, 9 marks
• Ass2: C Programming, weeks 7-10, 11 marks
• Assignments give you experience with larger programming problems than the lab exercises
• Assignments will be carried out individually.
• They always take longer than you expect.
• Don’t leave them to the last minute.
• There are late penalties applied to maximum assignment marks, typically 2%/hour
COMP1521 will offer an inclusive learning environment for all students.
In anything connected to COMP1521, including social media, these things are student misconduct and will not be tolerated:

- racist/sexist/offensive language or images
- sexually inappropriate behaviour
- bullying, harassing or aggressive behaviour
- invasion of privacy

Show respect to your fellow students and the course staff
Plagiarism

What is plagiarism?
Presenting the (thoughts or) work of another as your own.
Cheating of any kind constitutes academic misconduct and carries a range of penalties. Please read course intro for details.

Examples of inappropriate conduct:
• groupwork on individual assignments (discussion OK)
• allowing another student to copy your work
• getting your hacker cousin to code for you
• purchasing a solution to the assignment
Plagiarism

What is plagiarism?
Presenting the (thoughts or) work of another as your own.

Cheating of any kind constitutes academic misconduct and carries a range of penalties. Please read course intro for details.

Examples of inappropriate conduct:

- groupwork on individual assignments (discussion OK)
- allowing another student to copy your work
- getting your hacker cousin to code for you
- purchasing a solution to the assignment

Remember
You are only cheating yourself and chances are you will get caught!
Plagiarism

- Labs must be entirely the work of your pair.
- Assignments must be entirely your own work.
- You can not work on assignment as a pair (or group).
- Plagiarism will be checked for and penalized.
- Plagiarism may result in suspension from UNSW.
- Scholarship students may lose scholarship.
- International students may lose visa.
- Supplying your work to any another person may result in loss of all your marks for the lab/assignment.
Exam

Held in the CSE Labs (must know lab environment)

- on-line documentation
- most questions ask you to write C or assembler
- some questions will ask you to read C or assembler
- also may ask you to answer written questions

How to pass the exam:

- do the lab exercises
- do the assignments *yourself*
- practise programming outside classes
- treat extra tutorial questions like a mini prac exam
Assessment

- 10% Labs
- 9% Weekly Programming Tests
- 13% Assignment 1 — due week 7
- 13% Assignment 2 — due week 10
- 55% Final Exam

Above marks may be scaled to ensure an appropriate distribution.

To pass COMP1521, you must:

- score 50/100 overall
- score 22/55 on final exam

For example:
55/100 overall and 20/55 on final exam ⇒ 55 UF not 55 PS
Assumed Knowledge

Assumed knowledge:

- design an algorithmic solution
- describe your solution in C code, using ... 
  - variables, assignment, tests (==, !, <=, &&, etc)
  - if, while, for, break, scanf(), printf()
  - functions, return, prototypes, *.h, *.c
  - arrays, structs, pointers, malloc(), free()

Not assumed knowledge:

- linked structures, file operations, ADTs, sorting, recursion, bit operations