Course Outline

THIS DOCUMENT IS A DRAFT.
Parts of this outline are not yet finalised for the 21T1 offering.

Contents

- Course Details
- Course Summary
  - Assumed Knowledge
  - Learning Outcomes
- Teaching Strategies and Rationale
  - Lectures
  - Tutorials
  - Laboratory Classes
  - Assignments
  - Weekly Tests
- Student Conduct and Academic Integrity
  - Student Conduct
  - Academic Integrity
- Assessment
- Course Schedule
- Resources for Students
- Course Evaluation and Development

Course Details

Course Code	COMP2041, COMP9044
Course Title	Software Construction
Units of Credit	6
Course Contact	<cs2041 at cse.unsw.edu.au>
Convenor/Lecturer	Dr Andrew Taylor <andrewt at unsw.edu.au>
Admin	Dylan Brotherston <d.brotherston at unsw.edu.au>
Classes	**Lectures**: Tue 09-11, online; Thu 14-16, online; ... timetable for all classes
Course Website	https://cgi.cse.unsw.edu.au/~cs2041/21T1/
Handbook Entry	https://www.handbook.unsw.edu.au/undergraduate/courses/current/COMP2041/ https://www.handbook.unsw.edu.au/postgraduate/courses/current/COMP9044/

Course Summary

This course is designed for students who have mastered the basics of programming. It aims to broaden your knowledge of techniques and tools for software construction.

Assumed Knowledge

Before commencing this course, students should ...

- have a sound understanding of a procedural programming language such as C or Python;
- be able to produce a correct procedural program from a spec;
- have an understanding of fundamental data structures and algorithms (sorting, searching);
- appreciate the use of abstraction in computing.

For undergraduate (COMP2041) students, the above material will typically have been covered in COMP1511 (or its predecessor, COMP1917) and COMP2521 (or its predecessor, COMP1927).

For postgraduate (COMP9044) students, the above material will have been covered in COMP9021 and COMP9024, or in equivalent courses in their undergraduate degree.

A significant fraction of the class has previously used the C programming language, and some knowledge of that language may be assumed during some lecture examples. Students who have not programmed C (or C++) may need to do some extra reading to understand these examples.

Almost all the class has previously used the Python programming language. Students who have not programmed in Python previously can expect to do extra work for assignment 2, where you may not modify or extend supplied Python code.

Learning Outcomes

After completing this course, students will have:

1. practical experience in programming with the Unix shell and Perl;
2. broader and deeper knowledge of building software systems;
3. appreciation of the use of specific technologies and strategies during software development;
4. exposure to tools for version control, performance improvement, configuration, and debugging;
5. improved their ability to articulate and communicate concepts related to programming and systems.

This course contributes to the development of the following graduate capabilities:

Graduate Capability	**Acquired in**
scholarship: understanding of their discipline in its interdisciplinary context	lectures, assignments
scholarship: capable of independent and collaborative enquiry	lab work, assignments
scholarship: rigorous in their analysis, critique, and reflection	tutorials
scholarship: able to apply their knowledge and skills to solving problems	tutorials, lab work, assignments
scholarship: ethical practitioners	all course-work, by doing it yourself
scholarship: capable of effective communication	blog, tutorials
scholarship: digitally literate	everywhere in CSE
leadership: enterprising, innovative and creative	assignments
leadership: collaborative team workers	lab work, assignments
professionalism: capable of operating within an agreed Code of Practice	all prac work

Teaching Strategies and Rationale

This course uses the standard set of practice-focussed teaching strategies employed by most CSE foundational courses:

- Lectures â€¦ introduce concepts, show examples
- Tutorials â€¦ reinforce concepts and provide additional examples
- Lab Work â€¦ provide examples of using various technologies
- Assignments â€¦ allow you to solve larger problems

This course has a heavy practical orientation. Do not take this course unless you wish to do a very large amount of coding. This will place large demands on your time, particularly in the second half of session.

Lectures

Lectures will be used to present the theory and practice of the techniques and tools in this course. Lectures will include extensive practical demonstrations of various technologies. Lecture notes will be available on the course web pages before each lecture.

We plan to use broadcast lectures using Teams live events with the opportunity to ask questions via chat. Recording of all lectures will linked to the class home page. Lectures may be pre-recorded on some topics.

Tutorials

We plan to use Blackboard Collaborate for tutorials and labs. Your will be emailed links.

From week 1, you will also be expected to participate in a one-hour tutorial session to clarify ideas from lectures and work through exercises based on the lecture material. You should make sure that you use them effectively by examining in advance the material to be covered in each week's tutorial, by asking questions, by offering suggestions, and by generally participating. The tutorial questions will be posted on the Web in the week before each tutorial. There are no marks for tutorial participation.

Laboratory Classes

Following the tutorial class each week, there will be a two-hour laboratory class, during which you will work on a variety of small practical problems involving the tools introduced in lectures. Because this course is practical in nature, laboratory classes are **important**: if you do not put a great deal of effort into the lab classes, you risk failing the final exam.

Each week, there will be one or more exercises to work on. These exercises will be released in the week preceding the lab class.

Completed exercises need to be submitted. You must submit exercises before the deadline using **give** to obtain a mark for a lab exercise, The usual lab exercise submission deadline will be Tuesday 12:00 noon but some lab exercises may have an extended deadline.

Most lab exercises will be automarked. There will be partial marks for attempts which do not pass autotests. Tutors will separately provide feedback on your approach to the problem and on the style of your solution. Some labs may contain exercises which will be assessed during the lab.

Submission of any work that it is not your own will result in an automatic mark of zero for the entire lab component.

The lab exercises for each week are worth in total 2 marks. All of your lab marks will be summed to give you a mark out of 18; if their sum exceeds 15, your total mark will be capped at 15.

Challenge exercises may be specified for some labs. Challenge exercises may be involve concepts not covered in lectures and range in difficulty from not-very-hard to almost-impossible. Their contribution to lab marks will be limited so that you can obtain a you can obtain a HD (85+%) for the lab component without completing challenge exercises.

Weekly Tests

There will be weekly tests from weeks 3â€“10 designed to give you timely and realistic feedback of your understanding of the course material. Tests may be programming exercises, multiple choice questions, or both.

These will be conducted in your own time under self-enforced exam-like conditions. Each test will specify the conditions, but typically these will include:

1. no assistance permitted from any person;
2. a time limit of one hour;
3. no access to materials (written or online) except specified language documentation or man pages.

Each test is worth 1.7 marks, and will be automarked. Your total mark for the tests component is computed as a sum of your best 6 of 8 test marks. Any violation of the test conditions will result in a mark of zero for the entire test component.

Assignments

There are two assessable programming assignments. Assignments give you the chance to practice what you have learnt on relatively large problems (compared to the small exercises in the labs). Assignments are a *very important* part of this course, therefore it is essential that you attempt them yourself.

- Assignment 1, on shell programming; due Week 7; worth 15%
- Assignment 2, on Perl programming; due Week 10; worth 15%

The assignment weighting and deadlines may be slightly varied when the assignment designs are complete.

Late assignments submissions will be penalised. The exact penalty will be specified in the assignment specification: often, it is a 2% reduction in the maximum achievable mark for every hour late.

Final Exam

There will be a three-hour final exam, to be held online, during the exam period.

This will be centrally timetabled, and appear in your UNSW exam timetable.

The exam will contain short-answer questions, which may require you to read code, and implementation tasks, where you will be required to write code. During this exam you will be able to execute, debug and test your answers. The implementation tasks will be similar in nature to those encountered in lab exercises.

You will not be expected to remember all the details of programming languages used in the course; reference information will be provided along with the exam paper, giving a summary of any language that we expect you to use.

There is a hurdle requirement on the final exam. If you do not score at least 40% (18.0/45) on the exam (after scaling), you cannot pass this course. If your overall course score exceeds 50%, despite scoring very poorly (<40%) on the exam, the hurdle will be enforced via a grade of UF.

Student Conduct and Academic Integrity

Student Conduct

The **Student Code of Conduct** (Information, Policy) sets out what the University expects from students as members of the UNSW community. As well as the learning, teaching and research environment, the University aims to provide an environment that enables students to achieve their full potential and to provide an experience consistent with the University's values and guiding principles. A condition of enrolment is that students *inform themselves* of the University's rules and policies affecting them, and conduct themselves accordingly.

Students have the responsibility to observe standards of equity and respect in dealing with every member of the University community. This applies to all activities on UNSW premises and all external activities related to study and research. This includes behaviour in person as well as behaviour on social media: for example, in Facebook groups set up for the purpose of discussing UNSW courses or course work. Behaviour that is considered in breach of the Student Code Policy as discriminatory, sexually inappropriate, bullying, harassing, invading another's privacy, or causing any person to fear for their personal safety is serious misconduct, and can lead to severe penalties, including suspension or exclusion.

If you have any concerns, you may raise them with your lecturer, or approach the School Ethics Officer, Grievance Officer, or one of the student representatives.

Academic Integrity

UNSW has an ongoing commitment to fostering a culture of learning informed by academic integrity. All UNSW staff and students have a responsibility to adhere to this principle of academic integrity.

Plagiarism is defined as using the words or ideas of others and presenting them as your own. Plagiarism undermines academic integrity, and is not tolerated at UNSW. Instances of plagiarism are treated by UNSW and CSE as acts of academic misconduct, which carry penalties as severe as being excluded from further study at UNSW. There are several on-line resources to help you understand what plagiarism is and how it is dealt with at UNSW.

- Plagiarism and Academic Integrity
- UNSW Plagiarism Procedure

Make sure that you read and understand these. Ignorance is not accepted as an excuse for plagiarism. In particular, at CSE you are responsible for ensuring that your assignment files are not accessible by anyone but you by setting correct permissions in your CSE home directory and for any code repositories you may use. Note also that plagiarism includes paying or asking another person to do a piece of work for you, and then submitting it as your own work.

The pages below describe the policies and procedures in more detail:

- Student Code Policy
- Student Misconduct Procedure
- Plagiarism Policy Statement
- Plagiarism Procedure

You should also read the following page which describes your rights and responsibilities in the CSE context:

- Essential Advice for CSE Students

Assessment

Item	Topics	Due	Marks	LOs
Tests	all topics	Weeks 3-10	10	1-9
Assignment 1	shell programming	Week 7	15	1-5
Assignment 2	Perl programming	Week 10	15	1-5
Labs	most topics	most weeks	15	1-5,9
Final Exam	all topics	exam period	45	1-9

Your final mark for this course will be computed using the above assessments as follows:

CourseWorkMark	=	TestMark + LabMark + Ass1Mark + Ass2Mark	out of 55

ExamMark			out of 45
ExamOK	=	ExamMark â‰¥ 18.0/45	true/false
FinalMark	=	CourseWorkMark + ExamMark	out of 100
FinalGrade	=	UF, if ! ExamOK && FinalMark â‰¥ 50 FL, if FinalMark < 50/100 PS, if 50/100 â‰¤ FinalMark < 65/100 CR, if 65/100 â‰¤ FinalMark < 75/100 DN, if 75/100 â‰¤ FinalMark < 85/100 HD, if FinalMark â‰¥ 85/100	

Course Schedule

The following is a rough schedule of when topics will be covered. This will most likely change over the session as topics take more or less time to cover.

Week	Content
1	Course introduction. Unix filters.
2	Unix filters (cont'd). Shell programming.
3	Shell programming (cont'd).
4	Introduction to Version Control with Git.
5	Perl programming.
6	Flexibility week
7	Perl programming. (cont'd). **Assignment 1 due.**
8	Git in more detail
9	Tools for Performance Analysis and Deployment.
10	Revision. Exam Information. **Assignment 2 due.**

Resources for Students

There is no required textbook for the course. Useful (both for this course and beyond) reference books include:

- BrianÂ W.Â Kernighan and RobÂ Pike. 1998. *The Practice of Programming*. Addison-Wesley.
  (The inspiration for 2041, both in overall philosophy and in tool details.)
- SteveÂ McConnell. 2004. *Code Complete* (2nd ed.). Microsoft Press.
  (Many interesting case studies and practical ideas.)
- TomÂ Christiansen, brianÂ dÂ foy, LarryÂ Wall, and JonÂ Orwant. 2012. *Programming Perl* (4th ed.). O'Reilly Media.
  (The original, and the best, Perl reference manual. Previous editions are still excellent.)
- RandalÂ L.Â Schwartz, brianÂ dÂ foy, and TomÂ Phoenix. 2016. *Learning Perl* (7th ed.). O'Reilly Media.
  (A gentle and careful introduction to Perl. Previous editions are still excellent.)
- TomÂ Christiansen and NathanÂ Torkington. 2009. *Perl Cookbook* (2nd ed.). O'Reilly Media.
  (Lots and lots of interesting Perl examples.)
- RandalÂ L.Â Schwartz, brianÂ dÂ foy, and TomÂ Phoenix. 2012. *Intermediate Perl* (2nd ed.). O'Reilly Media.
  (A great book to read after this course â€" picks up where this course finishes.)
- RobertÂ W.Â Sebesta. 1999. *A Little Book on Perl*. Prentice-Hall.
  (A very concise introduction to Perl.)
- JonÂ Orwant, JarkkoÂ Hietaniemi, and JohnÂ Macdonald. 2011. *Mastering Algorithms with Perl*. O'Reilly Media.
  (Algorithms and data structures, implemented in Perl.)
- StephenÂ G.Â Kochgan and PatrickÂ Wood. 2013. *UnixÂ® Shell Programming*. Sams Publishing.
  (A careful introduction to shell programming.)
- CarlÂ Albing and JPÂ Vossen. 2017. *bash Cookbook* (2nd ed.). O'Reilly Media.
  (An example-based introduction to shell programming.)
- ShelleyÂ Powers, JerryÂ Peek, TimÂ O'Reilly, and MikeÂ Loukides. 2009. *Unix Power Tools* (3rd ed.). O'Reilly Media.
  (A comprehensive guide to common Unix tools.)
- MikeÂ Loukides and AndyÂ Oram. 1996. *Programming with GNU Software*. O'Reilly Media.

(A tutorial on the GNU programming tools: gcc, gdb, etc.)

- ArnoldÂ Robbins. 2009. *Unix in a Nutshell* (4th ed.). O'Reilly Media.
  (A concise guide to Unix and its toolset.)
- BrianÂ W.Â Kernighan and RobÂ Pike. 1984. *The Unix Programming Environment*. Prentice-Hall.
  (Predecessor to *The Practice of Programming*; a good introduction to Unix tools.)

Pointers to other useful reading material, including documentation for all of the tools used in the practical work, are provided from the course web pages.

Course Evaluation and Development

The course will be evaluated at the end of the session using the myExperience system; feedback will be used to make continual improvements to the course. Students are also encouraged to provide informal feedback during the session, and to let course staff know of any problems as soon as they arise. Suggestions will be listened to openly, positively, constructively, and thankfully, and every reasonable effort will be made to address them.

In the most recent session feedback was very favourable, probably as results of changes based on the previous session's feedback. Some lab exercises and lecture topics will be updated to better reflect current practice.