COMP(2041|9044) - Software Construction 19T2

Lecturer/Admin: Andrew Taylor, andrewt@cse.unsw.edu.au

https://www.cse.unsw.edu.au/~cs2041/

Course Goals

Overview: to expand your knowledge of programming.
First programming courses deals with ...

- one language (C or Python atCSE)
- some aspects of programming (e.g. basics, correctness)
- on small tightly-specified examples

COMP(2041|9044) deals with ...

- other languages (Shell, Perl
- other aspects of programming (e.g. testing, performance)
- on larger (less-small) less-specified examples

Course Goals

Introduce you to:

- building software systems from components
- treating software as an object of experimental study

Develop skills in:

- using software development tools (e.g. git)
- building reliable, efficient, maintainable, portable software

Ultimately: get you to the point where you could build some software, put it on github, have people use it and have it rated well.

Inputs

At the start of this course you should be able to:

- produce a correct procedural program from a spec
- understand fundamental data structures + algorithms
  (char, int, float, array, struct, pointers, sorting, searching)
- appreciate the use of abstraction in computing

Outputs

At the end of this course you should be able to:

- understand the capabilities of many programming tools
- choose an appropriate tool to solve a given problem
- apply that tool to develop a software solution
- use appropriate tools to assist in the development
- show that your solution is reliable, efficient, portable

Syllabus Overview

1. Qualities of software systems
   - Correctness, clarity, reliability, efficiency, portability, ...
2. Techniques for software construction
   - Analysis, design, coding, testing, debugging, tuning
   - Interface design, documentation, configuration
3. Tools for software construction
   - Filters (*grep, cut, sort, uniq, tr, sed*...)
   - Scripting languages (*shell*, *Perl*, *Python*, some Javascript)
   - Intro to Programming for the web
   - Software tools (git, ...)

Lectures

Lectures will:

- present a brief overview of theory
- give practical demonstrations of tools
- demonstrate problem-solving methods

Lecture notes available on the Web before each lecture.
Feel free to ask questions, but otherwise *Keep Quiet*.

Tutorials

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 start in week 2.
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

To get the best out of tutorials

- attempt the problems yourself
- if you understand OK and get feasible answer, fine

Ask your tutor if ...

- if you aren't sure your answer is feasible/correct
- if you don't know *how* the solution was reached
- if you don't understand a question or how to solve it

Do *not* keep quiet in tutorials ... talk, discuss, ...
Your tutor may ask for your attempt to start a discussion.

Lab Classes

Each tutorial is followed by a two-hour lab class.
Lab exercises aim to build skills that will help you to

- complete the assignments
- pass the final exam

Lab classes give you experience applying tools/techniques.
Each lab exercise is a small implementation/analysis task.
Labs often includes challenge exercise(s)
Do them yourself!

Lab Classes

Lab exercises contribute 9% to overall mark.
In order to get a marks most lab exercises:

- submitted via `give` before Sunday midnight
- automarked (partial marks if fails autotests)
- tutors will give feedback separately

Some lab exercises must be completed and/or assessed during lab class.
Most labs will contain challenge exercises.
More marks available for lab exercises than needed for full marks for lab component.
Can miss 1 week and some challenge exercises and still get full marks.
Any submission of work not your own - zero for lab component.

Lab Exemption

COMP9041 students can apply for lab exemption if they have:

- full-time work or other commitments
- previous programming, particularly scripting, experience
- good marks (70+) in previous courses

If granted - final mark calculated without lab component.
Strongly recommended they still complete labs.

Weekly Programming Tests

Programming tests in weeks 3-10 contribute 6% to overall mark.
Immediate reality-check on your progress.
Done in your own time under self-enforced exam conditions.
Each test specifies conditions, typically:

- No assistance from any person.
- Time limit of 1 hour
- No access to materials (written or online)
  except language documentation or man pages.
- automarked (we'll try to be generous here)
- best 6 of 8 tests used to calculate the 6
- any violation of the test conditions, zero for whole component

Assignments

Assignments give you experience applying tools/techniques
(but to 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

Organising your time $\Rightarrow$ no penalty.

Plagiarism

Labs and Assignments must be entirely your own work.
No group work in this course.
Plagiarism = submitting someone else's work as your own.
Plagiarism will be checked for and *penalized*.
Plagiarism may result in suspension from UNSW .
International students may lose their visa.
Supplying your work to any another person may result in loss of all your marks for the lab/assignment.
Assignments may allow use of code snippets (< 10 lines) with **attribution**
For example, can use 3 lines of code from Stack Overflow if clear you are not author

Final Exam

3-hour closed-book exam during the exam period.
The exam contributes 55% to total mark for the course.
Some multiple-choice/short answer questions - similar to tut questions.
Six (probably) implementation questions - similar to lab exercises.
COMP(2041|9044) hurdle requirement:

- you must score 23/55 on the final exam.
- you must successfully complete two implementation questions

If you do not complete two implementation questions, your exam mark will be capped at 22/55.
A question still regarded as complete if it contains a minor bug.
You can not pass COMP(2041|9044) unless you meet the hurdle.
If you fail to meet hurdle and get 50+, you receive a UF grade.

Final Exam

Format: Held in the CSE Labs (must know lab environment)

- limited on-line language documentation available
- we give you programming task
- some tasks specify a language (sh, perl, Javascript)
- some tasks may allow any of (sh, perl, Javascript, Python, C)
- similar in style/difficulty to labs

How to Pass this Course

Coding is a *skill* that improves with practice.
The more you practise, the easier you will find assignments/exams.
Don't restrict practice to lab times and two days before assignments due.
It also helps to pay attention in lectures and tutorials.

Reading Material

General References:

- *Kernighan & Pike*, The Practice of Programming, Addison-Wesley, 1998.

  (Inspiration for 2041 - philosophy and some tool details)

- *McConnell*, Code Complete (2ed), Microsoft Press, 2004.

  (Many interesting case studies and practical ideas)

Reading Material

Perl Reference Books:

- *Wall, Christiansen & Orwant* , Programming Perl (3ed), O'Reilly, 2000. (Original & best Perl reference manual)
- *Schwartz, Phoenix & Foy*, Learning Perl (5ed), O'Reilly, 2008. (gentle & careful introduction to Perl)
- *Christiansen & Torkington*, Perl Cookbook (2ed), O'Reilly, 2003. (Lots and lots of interesting Perl examples)
- *Schwartz & Phoenix*, Learning Perl Objects, References, and Modules (2ed), O'Reilly, 2003. (gentle & careful introduction to parts of Perl mostly not covered in this course)
- *Schwartz, Phoenix & Foy*, Intermediate Perl (2ed), O'Reilly, 2008. (good book to read after 2041 - starts where this course finishes)
- *Sebesta*, A Little Book on Perl, Prentice Hall, 1999. (Modern, concise introduction to Perl)
- *Orwant, Hietaniemi, MacDonald*, Mastering Algorithms with Perl, O'Reilly, 1999. (Algorithms and data structures via Perl)

Reading Material

Shell Programming:

- *Kochgan & Wood 2003*, UnixÂ® Shell Programming, Sams Publishing 2003 (Careful intoduction to Shell Programming)
- *Peek, O'Reilly, Loukides*, Bash Cookbook, O'Reilly, 2007. (Recipe(example) based intro to Shell programming)

Reading Material

Unix Tools Reference Books:

- *Powers, Peek, O'Reilly, Loukides*, Unix Power Tools (3ed), O'Reilly, 2003. (Comprehensive guide to common Unix tools)
- *Loukides & Oram*, Programming with GNU Software, O'Reilly, 1997. (Tutorial on the GNU programming tools (gcc,gdb,...))
- *Robbins*, Unix in a Nutshell (4ed), O'Reilly, 2006. (Concise guide to Unix and its toolset)
- *Kernighan & Pike*, The Unix Programming Environment, Prentice Hall, 1984. (Pre-cursor to the textbook, intro to Unix tools)

Reading Material

All tools in the course have extensive on-line documentation.
Links to this material are available in the course Web pages.
You are expected to master these systems largely by reading the manuals.
However ...

- we will also give introductory lectures on them
- the lab exercises will give practice in using them

Note: "The ability to read software manuals is an invaluable skill" (jas,1999)

Home Computing

All of the tools in the course are available under Unix and Linux.
Most have also been ported to MS Windows.
(generally via the CygWin project)
All tools should be available on Mac.
(given that Mac OS X is based on FreeBSD Unix)
Links to downloads will be placed on course Web site.

Home Computing

There may be minor incompatibilities between Unix, Windows and Mac versions of tools.

Therefore ... *test your assignments at CSE* before you submit them.

Note: we expect that any software that *you* produce will be portable to all platforms.

This is accomplished by adhering to *standards*.

Conclusion

The goal is for you to become a better programmer

- more confident in your own ability
- producing a better end-product
- ultimately, enjoying the programming process