Course Outline

Contents

- Course Details
- Course Summary
- Course Timetable
- Course Aims
- Student Learning Outcomes
- Assumed Knowledge
- Teaching Rationale
- Teaching Strategies
- Assessment
- Academic Honesty and Plagiarism
- Course Schedule
- Resources for Students
- Course Evaluation and Development

Course Details

Course Code	COMP1927
Course Title	Computing 2
Units of Credit	6
Course Website	http://cse.unsw.edu.au/~cs1927/ (http://cse.unsw.edu.au/~cs1927/) (which eventually links back to here)
Handbook Entry	http://www.handbook.unsw.edu.au/undergraduate/courses/current/COMP1927.html (http://www.handbook.unsw.edu.au/undergraduate/courses/current/COMP1927.html)

Course Summary

This course explores data structures, algorithms and software implementation techniques. You explore these ideas in tutorials and lab classes, but mainly via two assignments. Assessment involves labs, tutes, on-line quizzes, a prac exam, and a theory exam. At the end of the course, we want you to be a solid programmer, with knowledge of a range of useful data structures and programming techniques, capable of building significant software systems in a team environment, and ready to continue with further specialised studies in computing.

Executive Summary

A summary of the critical things to know about COMP1927:

- attempt all of the labs, quizzes and assignments yourself
- always try to produce a better program than last time
- in lectures, think critically about what's being said/shown
- the textbook is a useful reference source beyond this course
- assessment: quizzes:5%, labs:7%, assignments:23%, exam:65%
- *enjoy the course!*

Now, please read the rest of this document.

Course Timetable

All lectures are in CLB7, Wed 4pm-6pm, Thu 11am-1pm. The complete course timetable is available here (/COMP1927/15s2/timetable) .

Course Aims

The aim of this course is to get you to **think like a computer scientist** . Certainly sounds like a noble goal, but what does it really mean. How does a *scientist* , let alone a computer scientist, actually think?

What many types of scientists try to do is to understand natural systems and processes. For example, a geologist tries to understand the structure of the earth, a biologist tries to understand living organisms, a chemist tries to understand materials and reactions, and so on. Computer scientists don't, as the name might suggest, simply try to understand the structure and behaviour of computers, but are more

concerned with understanding software systems (and the interaction between the software and the hardware on which it runs). Also, unlike other scientists, computer scientists frequently build the objects that they study.

During this course, we'll be looking at ways of creating, analysing and understanding software. Ultimately, you should be able to answer the question "Is this piece of software any good?" and be able to provide sound reasons to justify your answer.

This course follows on from the introductory C programming courses COMP1917 or COMP1921. We cover additional aspects of the C programming language that were not covered in these course, and also look at some programming tools which were not covered (in detail) in COMP1917 or COMP1921. However, this course is not simply a second C programming course. The focus is on the ideas and abstractions behind the data structures and algorithms that are used.

COMP1927 is a critical course in the study of computing at UNSW, since it deals with many concepts that are central to your future studies in the area. Whether you are studying Computer Science, Software Engineering, Bioinformatics, Computer Engineering, or even a discipline outside the realm of computing, understanding a range of algorithms and data structures and how to use them will make you a much more effective computing problem solver in the future.

Student Learning Outcomes

After completing this course, students will:

- be familiar with fundamental data structures and algorithms
- be able to analyse the performance characteristics of algorithms
- be able to measure the performance behaviour of programs
- be able to choose/develop an appropriate data structure for a given problem
- be able to choose/develop appropriate algorithms to manipulate this data structure
- be able to reason about the effectiveness of data structures and algorithms for solving a given problem
- be able to package a set of data structures and algorithms as an abstract data type
- be able to develop and maintain software systems in C that contain thousands of lines of code

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

Assumed Knowledge

The official pre-requisite for this course is either COMP1917 or COMP1921.

Whether or not you satisfy the pre-requisite, we assume that:

- you can program in the C programming language, and are familiar with arrays, strings, pointers, dynamic memory allocation, recursion
- you are able to design, implement, debug, test and document small C programs (up to several hundred lines of code)
- you are familiar with the Linux environment on CSE computers

Installing Linux, possibly as a virtual machine, on your own computer would be a major bonus.

Teaching Rationale

Computer Science is, to a large extent, a practical discipline, and so COMP1927 has an emphasis on practice. Lectures will include exercises where we examine the practice of developing and analysing programs. Tutorials aim to develop analysis and understanding via practical case studies. Lab Classes also provide practice in program development and analysis. Assignments provide large case studies of software development.

Teaching Strategies

COMP1927 involves lectures, tutorials, labs, assignments and a text book.

Lectures aim to convey basic information about the course content and to model the practices and techniques involved in software development (i.e. we do demos). The most important components of the course, however, are the tutorials, labs and assignments. Tutorials aim to clarify and refine the knowledge that you got from lectures and from reading the text book and notes. Labs and

assignments are where you get to put together and practise all of the ideas from the lectures, tutes and text. The only way to develop the skills to do effective software development is by practising them. If you slack off on the assignments and lab exercises (or, worse, rely on someone else to do them for you), you're wasting the course's most valuable learning opportunities.

The University requires us to assess how well you have learned the course content, and the primary approach to achieving this is via a final exam. An exam is the ultimate summative assessment tool; it gives you a chance, at the end of the course, to demonstrate everything that you've learned. Labs and assignments are a *learning* tool, not an assessment tool, so, in an ideal world, I would have them as pure learning exercises and award no marks for them. However, to give a more concrete incentive to do them (in a timely fashion), there are marks tied to them.

Lectures

Each week there will be four hours of lectures during which theory, practical demonstrations and case-studies will be presented. Three of these hours (Wed 4-6, Thu 11-1) will be used for dealing with core material. The fourth hour of lectures will be used to look at programming tools and techniques (e.g. compilers, builders, debuggers, profilers, version control, etc.)

Lectures convey a small amount of information about the course content, but their main aim is to try to stimulate you to think about concepts and techniques. Feel free to ask questions at any stage, but otherwise please respect the right of other people around you who are trying to listen and (shhhhhh) keep quiet.

Note that I will *not* post the lecture slides before the lecture, to encourage you to concentrate on the fresh material. However, I will always give you textbook references to read for each lecture in the previous lecture and the notes will always be available before the lecture.

Text vs Notes vs Slides vs Videos

There are three sources of material available for you to study in this course:

- **textbook** : contains all material for the course (and more), available in UNSW Bookstore at start of semester, describes material in lots of detail and is very well-written
- **notes** : my summary notes on the material in the course, available online at start of semester, much less detail than the textbook
- **slides** : the material I use in lectures, available online *after* the lecture, very terse, much less detail than the notes
- **videos** : each lecture will be video'd and placed online after the lecture (takes some time to process and upload the video material to YouTube)

If you want to look at the material from the last time I ran the course, you can find it at

- http://webapps.cse.unsw.edu.au/webcms2/course/index.php?cid=2359 (http://webapps.cse.unsw.edu.au/webcms2/course/index.php?cid=2359) (http://webapps.cse.unsw.edu.au/webcms2/course/index.php?cid=2359)

Note that there will be differences (sometimes major differences) between the order of presentation and the actual lecture content between the current and previous offerings of the course.

Tutorials

Tutorials aim to clarify ideas from lectures and to get you to think about design/analysis issues. There will be a number of exercises set for each tutorial class. The aim of the class is *not* to simply get the tutor to give you the answers; the aim is to focus on just one or two of the exercises and work through them in detail, discussing as many aspects, alternative approaches, fine details, etc. as possible. You must be active and ask questions in tutorials. Ideally, students should run the entire tute themselves, with the tutor being a moderator and occasionally providing additional explanations or clarifications.

Lab Classes

Lab classes aim to give you practice in problem-solving and program development. Each week, there will be one or two small exercises to work on. These exercises will be released in the week preceding the lab class. Labs will be done in pairs, and you and you partner should discuss the exercises *before* going to the lab, to maximise the usefulness of the class. The exercises will need to be submitted (for our records) and will be assessed by your tutor. During the lab, your tutor will provide feedback on your approach to the problem and on the style of your solution.

Pairs will also be asked to do code reviews in the tutorials, to explain how they tackled a particular problem and describe interesting features of their solution.

If you complete all of the labs, to a high standard, a bonus mark is available.

Assignments

In the assignments, you will work on more substantial (hundreds of lines of code) programming exercises The first assignment is an individual assignment; the second will be completed in groups. We expect all members of a pair or group to contribute to the assignments; part of your assignment mark will be tied to this. As noted above, assignments are the primary vehicle for learning the material in this course. If you don't do them, or simply copy and submit someone else's work, you have wasted a valuable learning opportunity.

Quizzes

There will be a number of on-line quizzes during the semester, which you take in your own time. The primary aim of the quizzes is to review material from the previous couple of weeks' lectures, but they will also contribute towards your final mark.

Blogs

We expect all students to maintain a reflective blog during the semester. This is useful for us to keep up with what you're doing, but, more importantly, it is helpful for you to reflect on the improvement of your skills in programming, analysis, teamwork, etc. and to think about how you'll do it better in the future. There will be a up to two bonus marks available for (a) maintaining a blog consistently

(weekly) throughout the semester, and (b) writing thoughtful things in your blog.

Assessment

Your final mark in this course will be based on components from the assignment work, labs, quizzes, and the exam. Note that the exam is a hurdle, so that if you fail the exam badly enough, you cannot pass the course. If your failure in the exam is due solely to failing the Prac part, and the rest of your marks are satisfactory (okEffort), you will be given a second chance to complete the Prac.

The following formula describes precisely how the mark will be computed and how the hurdle will be enforced:

```
quizzes   = mark for online quizzes        (out of  5)
labs      = mark for lab exercises         (out of  7)
ass1      = mark for assignment 1          (out of  9)
ass2      = mark for assignment 2          (out of 14)
pracExam  = mark for Prac part of exam     (out of 35)
theoExam  = mark for Theory part of exam   (out of 30)
exam      = (pracExam+theoExam)            (out of 65)

okEffort  = (quizzes>=3 && labs>=4 && ass1>=4 && ass2>=6)
okExam    = (exam > 26/65)              (after scaling)

mark      = quizzes+labs+ass1+ass2+exam
grade     = HD|DN|CR|PS  if mark >= 50 && okExam
          = FL           if mark <  50 && okExam
          = UF           if !okExam
```

Academic Honesty and Plagiarism

Plagiarism is defined as (https://student.unsw.edu.au/what-plagiarism) *using the words or ideas of others and presenting them as your own* . UNSW and CSE treat plagiarism as serious academic misconduct.

In the context of Computing, plagiarism most frequently occurs as people submitting programs written by others as their own work. We use plagiarism-checking software to detect this. Such software has gotten quite sophisticated over the years, and is capable of recognising most techniques of attempting to disguise a copied program.

Similarly, getting other people do your prac work for you is treated as serious academic misconduct. Anyone discovered engaging in this will have their details forwarded to the UNSW Registrar for further investigation.

Make sure that you read and understand these. Ignorance is not accepted as an excuse for plagiarism.

Note that the penalty for academic misconduct can be as serious as being excluded from further study at UNSW, which will leave a serious black mark on your academic record.

How to avoid this? It's *simple* . Do all of your prac work yourself, or in conjunction with your allocated lab partner or assignment group. A side-effect of doing this is that you will learn how to program and have a much better chance of passing the Prac Exam.

There are several on-line resources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Learning Centre: Plagiarism and Academic Integrity (http://www.lc.unsw.edu.au/plagiarism/index.html)
- MyUNSW: Plagiarism (https://student.unsw.edu.au/plagiarism) and Academic Misconduct (https://www.gs.unsw.edu.au/policy/documents/studentmisconductprocedures.pdf)
- CSE: Addendum to UNSW Plagiarism Guidelines (http://www.cse.unsw.edu.au/~chak/plagiarism/plagiarism.html)
- CSE: Yellow Form (http://www.cse.unsw.edu.au/people/studentoffice/policies/yellowform.html) (whose terms you have agreed to)

Course Schedule

The (current and subject to change) schedule of lecture topics is:

Week	**Lectures**	**Tutes/Labs**
Week 1	Introduction, Data, Algorithms, ADTs, List Revision	-
Week 2	Algorithmic Complexity, Sorting: Simple Algorithms	ADTs, Lists
Week 3	Sorting: More Efficient Algorithms	
Week 4	Sorting: External (File) Sorting Algorithms	Profile Analysis
Week 5	Graphs: Definitions, Properties, Representation	Sort Detective
Week 6	Graphs: Traversal, Searching	Sort Detective (cont)
Week 7	Graphs: Weighted, Acyclic, MST, Cycles	Graph Construction, Geo Data
Week 8	Graphs: MST & Shortest Path Algorithms	Weighted Graphs, Geo Data

Week 9	Searching: Scanning, Binary Search, Trees	Shortest Path
Week 10	Searching: Binary Search Trees	Tree Construction
Week 11	Searching: Balanced Trees	Level-order Traversal
Week 12	Searching: Hashing, Tries	Balanced Trees
Week 13	-	Hash Tables

Each topic will be dealt with in tutes/labs in the week after it's covered in lectures.

Resources for Students

COMP1927 follows the contents of the pair of books:

- *Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching* (3rd Edition)
  by Robert Sedgewick, Addison-Wesley
- *Algorithms in C, Part 5: Graph Algorithms* (3rd Edition)
  by Robert Sedgewick, Addison Wesley

These two books are available as a bundle from the UNSW bookshop. They are expensive, but are useful well beyond this course, and will serve as a useful reference on the bookshelf of any serious programmer.

You may also be able to find on-line resources related to the text books. Robert Sedgewick has a series of videos on the topics in this course, but unfortunately they all seem to be in Java (which he has used for the new edition of his book). If you find any useful on-line resources, please let me know and I will add them to the Resources section of the course web site (with credit to the finder).

This website also has links to the auxiliary material/documentation that you'll need for the course. Solutions for all tutorial questions and lab exercises will also be made available. I'll review quiz and assignment solutions in the lectures.

Course Evaluation and Development

This course is evaluated each session using the CATEI system.

In the previous offering of this course, the most obvious themes in the "What could be improved" section were: make lectures more engaging, explain the practical relevance of algorithms/data-structures, reduce the workload (of tutes/labs), ensure that everyone pulled their weight in group-work (labs and assignments). Based on these comments, I will change the way I run lectures (more problem-solving than presentation), make labs easier with extension work to challenge advanced students, and introduce some peer assessment into the labs and assignments.

Resource created 6 years ago (Wednesday 22 July 2015, 01:20:13 PM), last modified 6 years ago (Sunday 26 July 2015, 11:27:14 AM).

Comments

Error getting response from server. Try refreshing the page.

There are no comments yet.