Course Outline

COMP1927
Computing 2

Semester 1 2013

Contents

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

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. 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.

Executive Summary

Since I know that many of you won't read all of the details below, here's 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: labs:10%, quizzes:6%, assignments:16%, diary:3%, exam:65%
- enjoy the course

Now, please read the rest of this document.

Course Details

Course Code:	COMP1927
Course Title:	Computing 2
Units of Credit:	6
Course WebSite:	http://www.cse.unsw.edu.au/~cs1927/

Handbook Entry: http://www.handbook.unsw.edu.au/undergraduate/courses/2013/COMP1927.html

Course Staff

Contact Details

Staff Name	Role	Email	Phone
John Shepherd	Course Convenor	jas at cse.unsw.edu.au	93856494
Vanessa Ung	Course Admininstrator	vanessau at cse.unsw.edu.au	
Oliver Tan	Lecturer	otan at cse.unsw.edu.au	
John Wiseheart	Lecturer	jcaw at cse.unsw.edu.au	

Consultations

Staff Name	Day/Time	Location
John Shepherd	Monday 1300-1400	K17-410
John Shepherd	Thursday 1500-1600	K17-410

Assumed Knowledge

Before commencing this course, students should:

- know certain things ...
- be able to do other things ...

These are assumed to have been acquired in the courses ...

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 bonus.

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: capable of effective communication	blog
leadership: enterprising, innovative and creative	assignments
leadership: collaborative team workers	lab work
professionalism: capable of independent, self-directed practice	assignments

Teaching Rationale

Computer Science is, to a large extent, a practical discipline, and so there 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. Assignment provide large case studies of software development.

Teaching Strategies

COMP1927 includes 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. 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, 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 ask you to do them because you wanted to learn, but award no marks for them. However, if I made the assignments and labs worth zero marks, I suspect that few people would attempt them (tell me I'm wrong!), so I will persist with the current compromise: prac work worth around 1/3 of the marks, exam worth 2/3 of the marks.

Lectures

Each week there will be four hours of lectures during which theory, practical demonstrations and case-studies will be presented. The fourth hour of lectures (Fri 2-3) will be used primarily for review, but may also introduce new material if we don't cover all of the scheduled topics in the first three hours.

Day	Time	Location
Wednesday	1pm-3pm	Mathews D
Friday	1pm-3pm	CLB 3

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.

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 providing additional explanations.

Lab Classes

Lab classes aim to give you practice in problem-solving and program development. Each week, there will be a number of small exercises to work on. These exercises will be released on the Friday of the week preceding the lab class. You should attempt them *before* going to the lab, to maximise the usefulness of the class. The exercises will need to be submitted (for our recrods) and assessed by your tutor during the lab. The tutor should provide feedback on your approach to the problem and on the style of your solution.

Assignments

In the assignments, you will work on more substantial (hundreds of lines of code) programming exercises All assignments will be completed *individually*; this means that you should do them *yourself* without assistance from others, except for asking advice from the teaching staff (lecturers and tutors) in this course. As noted above, they 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 three 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.

Reflective Diary

We'll expect all students to maintain a diary of their work via the WebCMS Blog. This is useful for us to keep an eye on what you're doing, but, more importantly, for you to relfect on the process of developing software and to learn how do it better in future. The quality of your diary contributes towards your final mark.

Assessment

Your final mark in this course will be based on components from the assignment work, labs and the exam. Note that the exam is a hurdle, so that if you fail the exam badly enough, you cannot pass the course. The following formula describes precisely how the mark will be computed and how the hurdle will be enforced:

```
labs    = total mark for all labs   (out of 10)
ass1    = mark for assignment 1     (out of  7)
ass2    = mark for assignment 2     (out of  9)
quizzes = mark for three quizzes    (out of  6)
diary   = mark for reflective diary (out of  3)
exam    = mark for final exam       (out of 65)

okExam  = (exam > 26/65)        (after scaling)

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

Exam

The exam will be a single 3-hour exam consisting of a combination of practical work (programming) and analytical work (theory). The exam is conducted in the CSE labs and all answers are entered and submitted online (even the theory questions). The practical questions will be like lab questions (but smaller to allow for the restricted time in the exam); the analytical questions will be like tutorial questions. The practical questions will be worth a little more than half of the total marks for the exam.

Academic Honesty and Plagiarism

Plagiarism is defined as "using the words or ideas of others and presenting them as your own". UNSW and CSE treat plagiarism as academic misconduct, which means that it carries penalties as severe as being excluded from further study at UNSW. There are several on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Learning Centre: Plagiarism and Academic Integrity
- MyUNSW: Plagiarism and Academic Misconduct
- CSE: Addendum to UNSW Plagiarism Guidelines
- CSE: Yellow Form (whose terms you have agreed to)

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

Course Schedule

The following is a summary of the topics that will be covered in this course. The schedule is **tentative** and will be refined over the next week; it may also vary as the semester proceeds. There will be different levels of detail on the different topics (some will provide overviews, some will be more detailed).

Week	**Topic** (Text ref)
Week 1	Introduction; C Revision (Ch.3)
Week 2	Abstract Data Types (Ch.4)
Week 3	Recursion and Trees I (Ch.5)
Week 4	Recursion and Trees II (Ch.5)
Week 5	Elementary Sorting Methods (Ch.6)
Week 6	Quicksort & Merging and Mergesort (Ch.7+8)
Week 7	Symbol Tables & Binary Search Trees (Ch.12)
Week 8	Balanced Trees (Ch.13)
Week 9	Hashing (Ch.14)
Week 10	Graph Properties and Types (Ch.17)
Week 11	Graph Search (Ch.18)
Week 12	DAGs, Minimum Spanning Trees, Shortest Path (Ch. 21)

Full details of the schedule will be given on the course web site.

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 have relevance well beyond this course, and will serve as a useful reference on the bookshelf of any serious programmer.

We will also make available our lecture slides, and all of the exercises from lectures. Each lecture will be video'd and the video will be attached to the course web site. Solutions for all tutorial questions, lab exercises, assignments and quizzes will also be made available (after the due date, of course)

You may also be able to find on-line resources related to the text book. 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).

Course Evaluation and Development

This course is evaluated each session using the CATEI system.

Feedback on the previous offering of COMP1927 (12s2) noted that most aspects of the course were ok, but requested more live coding in lectures and lecture videos. We will be doing both of these in 13s1. Other feedback suggested that too much time had been spent on lists, which had been reasonably well covered in COMP1917; I will check on your confidence with lists in the first lecture. Still other feedback suggested that some real-world context was needed to explain how the various algorithms and data structures might be used in practice; we will write new labs and assignments which try to ensure that the relevance of the algorithms is clear, and say a little more about this in the lectures.

Summary

By the end of this course, we want you to know a range of core data structures and algorithms and be able to use them to build demonstrably high-quality (correct, efficient) software using the C programming language.

Â

Enjoy the course! ... *John Shepherd*, March 2013