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)
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, class prac/theory exams, and a final prac/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.

Course Timetable

Lectures are Wed, Fri 09:00 - 12:00 Rupert Myers Theatre (K-M15-1001)

The complete course timetable is available here (http://www.cse.unsw.edu.au/~cs1927/16x1/CourseTimetable.html) .

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	lecture, 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	tutorials
scholarship: digitally literate	everywhere in CSE
leadership: enterprising, innovative and creative	assignments
leadership: collaborative team workers	lab work
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

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.

Labs

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.

You will receive lab marks for successfully completing the lab exercises. The lab mark will be part of your final mark, but it's main purpose is not assessment, but to make sure you constantly practice. There will be no late submissions.

Assignments

The assignments are significant programming exercises. There will be sufficient time to complete the assignments. You should start working on the assignment as soon as you can to allow for possible sick days and similar. Extensions will only be possible in exceptional circumstances, when you were not able to work on the assignment for a substantial part of the time allowed for the assignment. Late submissions are possible for up to five days after the deadlines, with penalties of 10% of the maximum mark deducted per day. 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, and you take the risk of being caught for plagiarism. If you are caught, you will be awarded an overall mark of zero for the assignment, and possibly the course, and receive an entry in the school plagiarism register. This does not look good on your academic record.

However, you should be aware that re-use of algorithms and data structures developed outside this course is allowed. You should not re-use code however, but if you feel it is justified, you should give credit to the original author or source.

Assignment related questions may be part of the exam

Prac Exams

There will be two prac exams run during lecture times. The questions will be similar in scope and complexity to lab exercises. Each of the prac exams will account for 5% of the overall marks.

Exam

There will be a three-hour closed-book exam for this course. The exam will consist of a practical (programming) part and a theory part. The final exam contributes 55% of the final mark for this course. There will be no second-chances to pass the final exam. Attendance at the exam will be treated as an indication by you that you are feeling well enough to sit that exam. If you are feeling so unwell on the day of the exam that you cannot attend, go to the doctor and get a medical certificate. If you put in a request for supplementary assessment after already attending the original exam, your request will not be granted. Also, if your special consideration request does not provide documented evidence that you personally have been severely affected by some unforeseen situation, then your request will not be granted.

Assessment

Component	**Percentage of overall mark**
Tutorials	5%
Labs	10%
Assignment 1	10%
Assignment 2	10%
Class Exams	10% (5 each)
Final Exam	55%

Your final mark in this course will be based on components from tutorials, 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. The following formula describes precisely how the mark will be computed and how the hurdle will be enforced:

```
okExam = (final exam > 22/55) (after scaling)
mark = tutorials+labs+assignment1+assignment2+class exams+ final exam
grade = HD|DN|CR|PS if mark >= 50 && okExam
      = FL if mark < 50 && okExam
      = UF if !okExam
```

Summary

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 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 (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)

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 may change 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	Book Chapter
1	Introduction, Linked lists, Abstract Data Types	Chapter 3, Chapter 4
2	Work Complexity,Recursion and Trees	Chapter 2, Chapter 5
3	Sorting	Chapter 6.1-6.5, 7, 8
4	Graph Intro, Graph Search	Chapter 17, 18
5	DAGs, Minimum Spanning Trees, Shortest Path	Chapter 21
6	Symbol Tables, Binary Search Trees	Chapter 12
7	Balanced Trees, Binary Search Tables, HashTables	Chapter 13, Chapter 14
8	Revision	

Resources for Students

A list of topics and references to the course textbooks will be given on the course web site. The course textbooks are:

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

both 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.

The following books might also serve as additional reference material on the C programming language:

- Brian W. Kernighan and Dennis M. Ritchie: **The C Programming Language** , 2nd edition, Prentice Hall, USA, 1988, ISBN 0-13-110370-9.
- Paul Davies, **The Indispensable Guide to C** , Addison Wesley, England, 1995, ISBN 0-201-62438-9.

Course Evaluation and Development

This course is being continuously improved and we will conduct a survey at the end of session to obtain feedback on the quality of the various course components. Your participation in the survey will be greatly appreciated. Also, please feel free to contact your tutor or lecturer with any feedback or suggestions for improvement during the session.

Resource created 6 years ago (Thursday 26 November 2015, 07:44:43 PM), last modified 6 years ago (Tuesday 01 December 2015, 07:25:15 PM).

Comments

Error getting response from server. Try refreshing the page.

There are no comments yet.