Course Outline

COMP6771
Advanced C++ Programming
Semester 2 2016

Contents

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

Course Aims

This course aims to introduce the students to intermediate and advanced programming using C++. Students will be exposed to the core features of C++ with emphasis on how to incorporate them into well designed C++ programs. The first part of the course will introduce the basic features of C++, including references, functions, C++ STL, operator overloading, classes, copy control, and scope and access control mechanisms. The second part will focus on more advanced topics such as exception handling, memory management, templates and generic programming, C++ template metaprogramming, input/output with C++ iostreams, C++ and efficiency issues, RTTI and effective C++ design guidelines. The structure, principles, and fundamental uses of the C++ Standard Library (including the Standard Template Library) will be introduced.

C++14 is the most recent iteration of C++, including some additions to the core language. As of GCC 6.1, GCC's C++11 mode implements all of the major features of the C++14 standard produced by the ISO C++ committee. In the 5th edition of our text book, *C++ Primer*, the authors have completely revised the popular book to use the C++11 standard. C++14 is intended to be a small extension of C++11, featuring some bug fixes and small improvements.

The course will be useful to those who wish to get a strong understanding of practical aspects of C++.

Graduate attributes (Engineering contextualised graduate attributes) are the qualities, skills and understandings the faculty agrees its students should develop during their studies. Some of these graduate attributes will be developed and assessed in this course in terms of the knowledge and skill outcomes listed below.

Student Learning Outcomes

Knowledge Outcomes:

- Understand the core features of C++ that object-oriented C++ would typically deploy, and why
- Understand the subset of C++ that generic (template-based) C++ would typically deploy, and why
- Gain a sound grounding in C++ fundamentals from which they can build upon and learn other OO languages such as Java and C#

Skill Outcomes:

- Understand, write and maintain C++ programs
- Use the core features of C++ effectively
- Ability to use a range of provisions from the standard library, including streams, exceptions, containers, iterators, algorithms and function objects;
- Ability to create function templates, class template and iterators
- Ability to deploy good C++ programming practices
- Practical ways to deal with memory and performance issues associated with C++

Assumed Knowledge

- **Pre-requisites**
  - Undergraduate Students: COMP2911.
  - Postgraduate Students: COMP9024.

This is not a first course on programming languages. Neither is this an introductory course on object-oriented programming languages. It is assumed that students are already fluent in at least one programming language and have had some programming experience in an object-oriented language so that we can cover the topics quickly. Some topics may be covered only briefly or not covered at all, and it is the students' responsibility to keep up with the assigned reading.

In particular, we are assuming that you know a reasonable amount of C so that you are comfortable with pointers, arrays, linked lists, trees, dynamic memory allocation, hashing and function pointers. If this is not the case for you, it is your responsibility to teach yourself enough of these in the first few weeks.

In addition, we are also assuming that you have acquired a basic understanding of key OO concepts such as inheritance and polymorphism in some earlier courses. These standard OO concepts will be introduced only to show how they are supported in C++. This is not a course on OO design, which is actually covered by COMP2911.

Understanding how a compiler works can usually get a motivated beginning programmer into C++ more quickly. You may consider taking COMP3131/9102 Programming Languages and Compilers in semester 1.

NB. You should have clear reasons for enrolling in COMP6771 and be prepared to invest a relatively large amount of time and hard work if you wish to perform well. If your only reason for taking COMP6771 is that it fits well into your timetable or you think it would look good on your CV, you may wish to reconsider.

Previously, we have found that students lacking the pre-requisite knowledge, a genuine interest in C++, and/or self-motivation performed rather poorly.

Course Staff

Staff Name	Role	Email	Phone
Jingling Xue	Course Convenor	j.xue@unsw.edu.au	
Bradford Heap	Lecturer	b.heap@unsw.edu.au	
Hayden Smith	Lecturer	hsmith@cse.unsw.edu.au	

Course Details

Course Code:	COMP6771
Course Title:	Advanced C++ Programming
Units of Credit:	6
Course WebSite:	http://www.cse.unsw.edu.au/~cs6771/
Handbook Entry:	http://www.handbook.unsw.edu.au/undergraduate/courses/2016/COMP6771.html

Resources for Students

Textbook

> *Stanley B. Lippman, Jose Lajoie, Barbara E. Moo, "**C++ Primer**", 5th Edition, 2013. ISBN: 0-321-71411-3.*

- An e-version at A$56.95.
- A print version at A$65.97 (currently running with a 35% discount).

Some other good C++ books can be found in the C++ Books resource page.

Course Forum (MessageBoard)

The course forum is an invaluable help tool. It should be your first port of call if you have a question. It may well be that your question has already been asked and answered. You should visit the forum often and never be afraid of posting questions or answers. The advantages of using the forum are:

- your question may have already been answered
- your question is likely to help others
- you are likely to increase your learning by reading the forum discussions
- you are likely to receive a timely response
- you can share your insights with each other

Course Schedule

Here is a high-level syllabus for this course:

- Introduction
- C++ Basics
- Sequential Containers
- Associative Containers
- Generic Algorithms

- C++ Classes (object-based programming)
- Copy Control (the Big Three/Five)
- Function Overloading
- C++ Classes (object-oriented programming)
- Templates and Generic Programming
- Memory management and exception handling
- Multithreading
- Single and Multiple Inheritance
- C++ Metaprogramming

Material covered may vary somewhat, depending on contingent factors.

Teaching Rationale

This is a one semester course consisting of three-hour lectures per week, one tutorial per week, unlimited email consultation, five programming assignments and one three-hour final exam.

Programming is about building and using abstractions. The approach we take in teaching this intermediate to advanced C++ course is to emphasise on using abstractions (from the C++ Standard Library) first and then concentrate on exploring how to build abstractions in C++. Therefore, we will introduce the Standard Library and generic programming earlier in the course. This will allow students to write programs in terms of those abstractions, and defer thinking about how they may work until they became comfortable with using them.

Teaching Strategies

How much you get out of this course is proportional with how much effort you put into the course. C++ is a complex language complemented by extensive library resources. It is impossible to cover every feature in detail and, as such, we will often introduce material in brief and then expect you to expand your understanding through reading, tutorial questions and programming assignments. You are expected to attend all lectures and tutorials.

The programming assignments are serious projects and they will get more and more challenging as we cover more advanced topics. The only way to learn programming is to work at it. So be prepared to spend a lot of time in front of a computer.

The first part of the course focuses on "using abstractions". We make a serious attempt at learning how to write C++ programs effectively using the existing abstractions provided by the C++ Standard Library (including the C++ Standard Template Library).

The second part of the course focuses on "building abstractions". We look at how abstractions are built in C++ and learn how to define new abstractions.

Tutorials

Tutorials give you a chance to reinforce material from lectures and set readings, and to practice your problem-solving skills in a small (and hopefully more personal) class with the assistance of a tutor. You should make sure that you use them effectively by attempting in advance the questions to be covered in each week's tutorial and by participating in tutorial discussions. Your tutor will help with some of the trickier questions, but often there will not be enough time to examine the answers to all questions. There will be two sets of tutorial questions:

- **Class Activity Questions**: Your tutor will go through these questions interactively with you in your tutorial session. The answers for a week's Class Activity Questions will be made available online at the end of the week.
- **Extra Questions**: Your tutor may go through some of these questions if there is spare time left or no questions at all. Their answers are available online at the same time.

Tutorials are one hour long and they are scheduled to start in Week 2 of semester.

Tutorial participation is strongly encouraged and we will keep track of tutorial attendance.

Special Enrollments

Some students have been given special permission to enroll in this course, for example:

- timetable clash
- accelerated program
- pre-requisite waiver

If you are one of those students, you have been made aware of the difficulty of this course and the risk you are taking when your permission to enroll was granted. It is your responsibility to ensure that enough time and effort are allocated to this course to make up for any shortfall in prior knowledge or class attendance.

You will not be given any special consideration on the grounds that you were disadvantaged.

Assessment

The mark breakdown between the programming assignments and the final exam is:

Component	Marks
Programming Assignments	45
Final Exam	55

Your final mark is the total of the marks achieved in all components.

To pass this course, you must **pass** the final exam **and** attain at least 50 overall. This means that you can fail the course in spite of any astounding performances on the programming assignments (this is often a good indication of outside help!).

Final Exam

There will be a **three-hour, closed-book (MCQ and/or written)** exam for this course in the November exam period.

All material covered in this course, including material in set readings, tutorials and assignments, is examinable in the final exam.

The final exam is a hurdle requirement of this course.

Programming Assignments

There will be **five** individual programming assignments, which are designed to help you improve your mastery of C++ and develop your programming skills in building and using abstractions in C++. These assignments will become progressively harder (perhaps only conceptually for the last assignment). There is a good reason for this: a lot of ground must be covered first before some interesting programming assignments (e.g., those on templates) can be attempted. The first assignment is relatively easy, but you shouldn't let your guard down.

It is essential that you make every effort to complete these programming assignments. The programming practice will put you in good stead for the final exam.

Discussion and problem solving (on paper!) with your peers is encouraged, but you are not allowed to derive code together during such discussions. Students are also warned not to send code fragments of the assignments to each other in any form (e.g., as e-mail or listings). In addition, copying/purchasing of code that is available on the Web is also not permitted.

Please see further plagiarism information below.

Lateness

The pace of this course makes it difficult for students to catch up once they have fallen behind. So students are strongly encouraged to complete all of their programming assignments on time.

Late programming assignments will be penalised unless you have legitimate reasons to convince the LIC otherwise. Any assignment submitted after the due date will attract a reduction of 20% per day to your mark. A day is defined as a 24-hour day and includes weekends and holidays. **No assignment will be accepted more than three days after its due date.**

In the event of an illness or misadventure that makes it impossible for a student to complete an assignment on time, extensions may be granted beyond its due date. To request an extension for an assignment, email the LIC **no later than 24 hours before the assignment is due**. You will also have to formally apply for special consideration.

Special Consideration

If you believe that your performance in an assessable component of the course has been affected by illness or misadventure, you can apply for special consideration with the university. This is a formal process and you must provide appropriate proof. It is important that you apply for special consideration in a timely manner. Late applications are likely to be summarily dismissed.

Click on the following for more information:

- UNSW Policy: https://my.unsw.edu.au/student/atoz/SpecialConsideration.html
- CSE Policy: http://www.cse.unsw.edu.au/people/studentoffice/policies/yellowform.html#special

Please note that you will almost always be required to complete the assessment component or a similar substitute, even if your special consideration application is approved by the university.

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.

UNSW and CSE are committed to detecting plagiarism in assignments. In this course, we run a special program that detects similarity between assignment submissions of different students and then manually inspect those with high similarity to guarantee that the suspected plagiarism is apparent. The plagiarism detection software is sophisticated enough to ignore computationally irrelevant changes!

If you receive a written letter or email relating to suspected plagiarism, please contact the LIC within the specified deadline. While those students can collect their assignments, their marks will only be finalised after we have reviewed your explanation regarding their suspected plagiarism.

Course Evaluation and Development

Student feedback on this course will be obtained via electronic survey at the end of semester, and will be used to make continual improvements to the course. Students are also encouraged to provide informal feedback during the semester, and to let the lecturer in charge know of any problems, as soon as they arise. All reasonable suggestions/concerns will be carefully considered and every effort will be made to address them.

Here are answers to some of the questions raised in the 2015 student feedback.

Start to move towards C++14 as it is the new standard

C++14, released on 15 December 2014, is a version of the standard for the programming language C++. it is intended to be a small extension over C++11, featuring mainly bug fixes and some small improvements.

Lay more stress on multithreading

This will be covered a bit more this year.

More focus should be given to teaching more practical things but not something too advanced

This is a course on advanced C++ programming, focusing on teaching real-life skills that are essential in writing practical C++ applications.