CSE 3150 — C++ Essentials — Yufeng Wu —Fall 2024
Syllabus ——- Please read carefully.

An important message about this course: You will need to do a **significant** amount of C++ programming in this course. In particular,

1. Expect weekly programming assignments or projects.
2. The amount of code for these assignments would grow from almost trivial programs in the beginning to a project that you may have to write at least one thousand lines of code (or more) near the end of the semester.
3. You must be able to write and debug programs on your own. All assignments are individual. My past experience of teaching this course is that often most students can write code, but quite a number of students cannot debug the code. If you haven't acquired the skill of debugging a non-trivial piece of program, you will have to pick up that skill quickly.
4. Taken from the SET report from Fall 2023 class (with 20 students): (i) 67% students felt the course is more difficult and 33% felt the course is much more difficult than other courses. (ii) 33% students felt they learned more from the course and 67% felt they learned much more from this course than other courses. (iii) On the number of hours spent weekly on the course outside class: 4 to 6 for 50% students, 7 to 9 for 33% students and 10 to 14 for 17% students.

I strongly believe a student in Computer Science major should possess good programming skills, no matter what career he or she will pursue. Please note "buggy" or "spaghetti" code has literally no value. As a trained computer science professional, you are supposed to write quality code that really works [1]. The best way to improve coding skills is by practicing on some programming tasks that are close to some real world applications. In the past, I heard some students said some assignments in the class (especially projects) are challenging. A senior student complained that the assigned programming work in this course was way more than what he or she did in the senior design project. I believe the amount of the programming tasks in this course, while non-trivial, are certainly doable within reasonable amount of time. CSE 3150 is not an entry-level course. I expect a student has already known C programming well (I will conduct a C code practice session in the beginning of the class to check your C programming skills). I know this course may feel hard for quite a number of students. One thing I am certain is: if you stay in the class till the end, you will learn a lot. Quote from a student from fall 2022 class: this course "felt like the most applicable course I have taken in CSE so far, and as hard as it was, really solidified for me that I want to pursue a career in software engineering".

1 Overview

This course is about learning the C++ language and applying C++ in practice. There are two main objectives for this course.

1. Learn the C++ as a programming language. C++ is a powerful language. It is not as fancy as newer programming languages such as Java and Python. Nonetheless, C++ still has a significant market share in programming languages. During the first several weeks, each student needs to pick up C++ and can use it as a working programming language.

[1] If you haven't heard about, there are machine learning based automated programming systems that can generate code.

2. Learn object oriented design and programming in C++. C++ is one of the object oriented (OO) programming languages, so are Java and Python. However, it is important to note that programming in an object oriented language doesn't mean your programs follow the object orientation paradigm. A main objective of this course is teaching you how to perform OO design and programming in C++. I believe OO using C++ is one of the most important applications for C++. I know many of you may already have taken courses that cover some aspects of OO. Still, I believe you will learn a lot in this course by practicing OO on a somewhat larger scale using C++.

2 Administrative issues

Instructor Yufeng Wu. Office hour: Wednesday: 10 am to 12 pm (after class). ITEB 235. Email: `yufeng.wu@uconn.edu`. Please avoid emailing me for questions regarding assignments. Post in Piazza instead.
TA: Check HuskyCT and Piazza for office hours of TAs.

1. Alexander Hamilton, `alexander.2.hamilton@uconn.edu`
2. Amogh Parmatma Chaubey, `amogh.chaubey@uconn.edu`
3. Joshua Kaplan, `joshua.a.kaplan@uconn.edu`
4. Chuanyu Xue, `chuanyu.xue@uconn.edu`
5. Daniel Baker, `daniel.baker@uconn.edu`
6. Mainak Mondal, `mainak.mondal@uconn.edu`
7. Yiming Zhang, `yiming.zhang.cse@uconn.edu`
8. Ankit Bhardwaj, `ankit.bhardwaj@uconn.edu`

TAs' office hours will be posted in HuskyCT/Piazza shortly.
Textbook: There is **no** required textbook. I do recommend the following books.

1. Accelerated C++, A. Koenig and B. E. Moo, Andison-Wesley, 2000. This is a concise textbook on C++. This book can be useful to get started with C++.
2. Effective C++ by Scott Meyers. This is a great book for learning C++. I personally learned a lot by reading this book. I won't go through this book in depth in class. But quite a number of topics in the book will be covered in the class assignments.
3. More Effective C++ also by Scott Meyers. If you like the previous one, you may also like this one.
4. Design patterns: Elements of Reusable Object-Oriented Software. By E. Gamma, et al., Addison-Wesley, 1994. I highly recommend this book. This is a great book for learning C++ based object orientation. We will try to use the techniques in this book in the course project. Note this book is a great book about object orientation and design patterns in C++. But it doesn't really teach you the C++ language itself.

Prerequisites: You need to know how to write programs. This is a 3000-level course. I won't try to teach you basics of programming (such as conditionals and loops), which I suppose you have learned in a course like CSE 1010. *All* programming assignments will be *individual*-based. In the industry, team work is essential. However, in order to be a good team player, you need to know how to write code yourself first. If you cannot write programs on your own, you will likely struggle in this course. Note if

you cannot write programs, you will also likely have difficulty in landing good jobs after you graduate. So be *prepared*; as I said earlier, you will write significant amount of code in this class. **CSE 3100**: CSE 3100 is officially a prerequisite. You need to take CSE 3100 *before* taking this course. Since C is a subset of C++, you will need to know how to program in C before learning the C++ specifics.
Lectures: Most lectures are likely be given as pre-recorded videos. Some of the scheduled class time will be used mainly for review and practice, and sometimes tests. Some other scheduled class meetings will be used for watching these videos. Please make sure to watch the lecture videos **within** the week when the videos are assigned. Some lecture videos are marked as "optional". These optional videos, while useful, are not the most important parts. You should watch them if you have time or are really interested in learning all of C++. But these optional videos won't be covered much in assignments.
Programming assignments: I will assign programming assignments. These programming assignments will focus on writing C++ programs to solve relatively small and well-defined problems. The purpose of programming assignments is helping you get comfortable with programming in C++.
Exams: We will have several in-class exams. I expect to hold the following possibly paper-based exams:

1. C++ basics.
2. Object orientation, design patterns and other topics not covered by exam one.

These tests are designed to test whether you understand basic concepts (e.g., of C++ language or object orientation). The current plan (again subject to change) is that there won't be final exams. I will assign a final project that will be in lieu of the final exam.
In-class exercises: During class and lab time we may work on some relatively small programming or other types of exercises. The purpose is to review the lectures/readings done during the past week. Some of the exercises will be closely related to the lectures (e.g., writing the code for the examples taught in the lecture videos). These exercises are not exams. Please note: these exercises will be **graded**. I plan to do live programming on many of the exercises in class. You will be given extra time to finish after the class. I have heard from past students that these exercises can be an important part for learning in this course. So please attend the class and complete and submit these exercises.
Lab session: The TAs will run the weekly lab sessions. During lab sessions, the TA may cover programming assignments which are not completed during class time. There will be assignments in the lab sessions. Lab submissions will be graded. Lab attendance is optional but students are responsible to know the topics covered in the lab and make the required submission. At the end of the semester, the TA will come up with a lab grade which is based on the lab submissions.
Laptop: I assume each student has a laptop (Windows, Mac or Linux). I haven't got a student who doesn't have one in the past several times I taught this course. If you happen to not own a laptop, I believe it is time to buy one. A Computer Science student without a laptop feels like a soldier without weapon. When you come to the class, please bring your laptop. We will do live coding in class.

1. Windows. If you use Windows, I highly recommend you to install WSL 2. This is the Windows implementation of Linux.
2. MacOS or Linux. You can just use the console environment for development.

Autograding of programming assignments: I plan to use Gradescope's autograding for grading programming assignments (but not the projects). We expect most (if not all) programming assignments will be graded by autograder. Please note: we don't have the resource to perform manual re-grading for auto-graded assignments. That is, the score reported by the autograder will be the *final* score for that assignment, unless otherwise stated. A frequent request by students is asking for partial credits of the code that the student claim to be "almost working". I don't believe there is much value for a code that is "almost working": you cannot sell such code to a customer when working in the industry. Moreover, it is not easy to verify the claim that the code is "almost working". So please spend time to make sure your code passes the test cases.

Extension of programming assignment submission: I frequently get requests for extension of programming assignment submissions. So I am making my policy clear: every student gets one-day automatic extension of all programming assignments (unless otherwise stated). There are circumstances where we don't allow extension. We will explicitly state so in those cases. Other than the automatic one-day extension, we won't give more extensions (unless there are unusual circumstances): a deadline is a deadline. In the real world, you will have to work with deadlines. So try to complete your work on time.

Project: We will have course projects. Course projects are somewhat more complex than programming assignments in the following sense.

1. A project usually comes in more than one parts. You will need to build code for the next part on top of what you did in the previous part. Therefore, it is important to maintain a working code base. That is, make sure to fix the bugs in your code for the earlier parts; otherwise, your code for the next part may not work.
2. A project usually has less given starter code. Likely I will only specify the necessary part from a client's point of view. You will have to furnish more details to make the code work.
3. A project may need some careful design and application of good OO principle.

I will provide more details about projects later in the course.

Grading: Programming assignments (30%), exams (30%), in-class exercises (10%), labs (5%), and projects (25%). *Letter grade* will be decided based on the semester grade (the accumulative grade as computed above). However, I may or may not use the standard conversion from the semester grade (as computed above) to letter grades. I guess some of you are wondering that this means. It just means that I have the flexibility of deciding whether to curve the grade or not. In the past, semester course overall scores vary. Sometimes the whole class grades are low. I expect to use the standard conversion (e.g., 90% for A/A-, 80% for B-/B/B+, and so on) as long as it leads to a reasonable distribution of letter grades. Typically 20 to 30% of students will get A (including A-). But of course this doesn't necessarily mean the grades for this time will be like these. I may consider curving the grades if the course grades are significantly lower than expected.

3 Organization of the course

This course will be heavily focused on practice. There are three main parts. I don't plan to give many lectures in this course. We will use the class meeting time mainly for discussion and lab.

1. Learning C++ basics. This part will be fast paced. You will need to watch video-based lectures (in HuskyCT, recorded by Prof. Laurent Michel) which cover the C++ language. I Our class meeting time during this period will be mainly devoted to discussion on how to work with C++, and sometimes coding or paper-based tests.
2. Learning OO in C++. After you have learned the basics of C++, we will switch to focus on object orientation. Here, you will need to read the design pattern book. During class time, we will discuss design patterns.
3. More topics in C++. I will cover more advanced topics in C++. Topics may include Lambda and C++ pointers, among others. Exactly what topics and how many such topics to be covered will depend on the progress of the course. It is possible that we won't have enough time for these. Let us see how the class progresses.

4 C++ specifics

In this course, I don't plan to use any C++ IDE. We will use console-based (i.e., command line) C++ development. Programs are usually compiled using a Makefile. I will show you how to write simple Makefile in the class. I do recommend to use a good program editor. I am aware of web-based services where you can code and test your code online. While this can work for many programming assignments, it may not work for the course projects. So please make sure you can code on your own computer.

5 Academic integrity

Students in this class are expected to follow the academic integrity code of UConn. For all assignments (except the exams), you are allowed to discuss with other students. However, **you must write programs yourself**. **Note**: I do check the submitted code frequently. During the past several years, I did catch several students who cheated in the programming assignments *each* time when I teach the class. Thus I will make my policy clear.

1. If a student is caught cheating for the first time, I will meet with the student (with the presence of a TA) to discuss this matter. The student would get zero for the assignment. The student has to promise there is no more cheating in this class.

2. If a student is caught for the second time, I will meet with the student again (with a TA). The student will get an F for this class.

5.1 More about academic integrity

Some students in the past asked me why I insist all assignments being individual. Why cannot students collaborate? Isn't collaboration the way how code is written in the industry? My past experience in the industry is, team work is important in the industry but doesn't work very well in a class like CSE 3150. If two students work together in a project, often the code is mostly written by the student with stronger coding skills. This leaves the other student not contributing and not learning.

We are in the era where technologies are fast changing. I know that parts of the class assignments may be done through latest AI (e.g., ChatGPT). I strongly suggest you not copying code from AI services like ChatGPT. Some simpler problems may be solvable by ChatGPT. But my experience indicates for more complex tasks (e.g., projects), ChatGPT just doesn't work. Moreover, the very purpose of the earlier (simpler) assignments is meant to let you learn gradually how to work with C++. If you just copy code (from ChatGPT or from someone else), you would miss this important learning step.

I am frequently surprised that students in the class don't actually understand what is "academic misconduct". The following is taken from UConn Community Standards: "Academic misconduct includes, but is not limited to: Providing or receiving assistance on academic work (papers, projects, examinations) in a way that was not authorized by the instructor " `https://community.uconn.edu/student-undergraduate-faq/`. Please keep in mind: if you send your code to your friend, that is cheating! If you copy code from some online resource (e.g., ChatGPT), that is cheating too.