syllabus for
data structures
comp 218

Instructor: **Mark LeBlanc**
mleblanc@wheatoncollege.edu
http://cs.wheatoncollege.edu/mleblanc

SC-1322 508.286.3970
Open Hours: **MON-TUE 1-3pm**
or by appointment
ping me, I'm here often

Meeting Times: **MWF 9:30 – 10:20** DC 1315 ("csLab")
Wed Lab 3:30-5:20 DC 1313

Data Structures Using C++ (2nd Edition). Malik, D.S (2010).
How to think like a computer scientist (C++ Version). Allen B. Downey (2012).
C++ Programming. Wikibooks.org
Select pages from texts will be available online. I also strongly recommend that you buy a folder. I'll pass out lots of handouts.

Content: This course is the second half of a year-long introduction to computer science in C++ for majors and minors [Prerequisite: COMP 118 Object-oriented Programming]. The content includes an introduction to the theoretical and practical aspects of data structures. Emphasis is on abstract data types and the use of the class mechanism to support object-oriented implementations, including exposure to the C++ Standard Template Library (STL). Examples of data structures include arrays, matrices, stacks, queues, linked lists, and general trees and their applications. Pointers, memory management, and recursion are discussed, implemented in lab, and used in some assignment implementations.

C++ is often used in mission-critical applications so 'code that crashes' is not an option, and thus, with much power comes much responsibility. Developing today's data-centric "apps" involves complex ways to "structure your data", a.k.a. "*data structures*". We will use C++ object-oriented features to both implement our own data structures, as well as apply higher-level objects made available in libraries (e.g., the STL). We will repeatedly emphasize good software engineering skills, including the steps that lead to verify "program correctness." Specifically, you will continue to write good (internal) documentation in your programs, including pre- and post-conditions for your functions and methods and logical invariants for your loops. You will test your solutions with unit tests. In summary, this course is where you really appreciate and practice the art of writing software, of programming, of scripting. Writing software is a craft and given the evasive nature of software in our world, programmers should be efficient *and* correct. In short, I often tell students that once they complete this course, they'll "really (start to) get it." Are you ready to "get it?"

In computer science, if you are almost correct you are a liability.
Fred Kollett (1941-1997), MathCS, Wheaton College, Norton, MA

Your Grade:

Things to do	Grading	Frequency
Labs	10%	weekly
6 Programs (1) medPing (2) class TextMiner (3) class myString (4) Qsim (5) class Vector (6) Twenty Questions	45%	TBA all deadlines will be agreed upon during lectures and posted on Canvas
Quizzes (approximately 4)	10%	TBA
Exams (2)	20%	Wed., Oct. 1 Wed., Nov. 12
Final Exam	15%	Wed., Dec. 10, 9-12

Week	Topic Areas	Readings[*] (exact pages will be assigned in class)
1-2	**August 27 – September 5** `hello data structures` Abstract Data Types (ADTs) Object-oriented Programming (OOP) Knowing your memory Documentation: Pre-Post conditions	websites Chapter 1 and 2 Appendix E
3-4	**September 8 - September 19** C-style strings “Smart” strings Operator overloading Working together on an API (Application Programmers Interface) Documentation: Loop Invariants	websites Chapter 3, Appendix D
5	**September 22 - September 26** Stacks	Chapter 7
6-7	**September 29 - October 10** Building dynamic data structures Pointers, Link Lists **Exam 1** The Standard Template Library (STL)	Chapter 3 Chapter 5 Chapter 4
	Fall Break – October 13-14	
8-10	**October 15 – October 31** Queues Deques Priority Queues	Chapter 8
11	**November 3 – November 25** Recursion	Chapter 6
12	Inheritance, Polymorphism **Exam 2**	Chapter 2
13-14	Hashing STL map Search Trees	Chapter 9, 11
	Thanksgiving Break – November 26-28	
15	**December 1 – December 5** “Big Oh” Analysis Review	Chapter 9
16	Final Exam - **Wed., Dec. 10**, 9am -12pm, DC 1315	

* Readings from “Chapters” are from *Data Structures Using C++* by Malik
All daily deadlines and exact pages for readings will be listed on our online class website in Canvas.

Course AI Policy

This course will follow Wheaton’s “Unrestricted Use of AI with Acknowledgment” policy **[**AI Use in the Classroom: Guidelines for syllabus statements and framework for the development of a college policy**]**. In this course, **I encourage you to use all the tools at your disposal**. However, as with any other resource you use to aid your work, **you must acknowledge the AI tools that you use in the development of your work. That is especially true when co-writing code along with bots**. We will lean on and leverage bots to help with code snippets and workflows, as well as with writing documentation and unit tests, but we will also agree to include citations and acknowledgments in our software and results. As programmers, we should know this best: the vetting, testing, and acknowledgment of bot-written code falls to us, the programmers.

And to be fair, from my faculty point of view, I will include a clear statement in any class materials or assessments that involve AI use.

Learning Goals

1. Understand and apply core abstract data types (e.g., stacks, queues, linked lists, trees) using object-oriented principles.
2. Design and analyze algorithms for problem-solving, including recursive and iterative approaches.
3. Develop strong programming proficiency in C++, integrating both high-level abstractions and low-level memory management.
4. Select appropriate data structures based on problem requirements and performance trade-offs.
5. Apply software engineering principles such as encapsulation, modularity, and code readability.
6. Demonstrate responsible and secure coding practices that address correctness, efficiency, and reliability.
7. Understand runtime memory organization and manage dynamic memory

Learning Objectives:

By the end of the course, students will be able to:

1. Implement and test object-oriented versions of fundamental data structures in C++.
2. Compare and contrast the performance of alternative algorithms and data structures.
3. Write and debug recursive algorithms, identifying base and recursive cases.
4. Use pointers, dynamic memory allocation, and manual memory management safely.
5. Trace program execution and identify logic, runtime, and memory-related errors.
6. Apply decomposition and abstraction techniques to design maintainable programs.
7. Use the C++ Standard Template Library (STL) effectively when solving practical problems.
8. Justify design decisions regarding data structure selection and implementation strategies.

Late Submissions:
Due is due. Always submit program assignments on time. Something submitted on time is much better than not having it accepted because it is late. Meeting deadlines takes organization and effort and the deadlines are part of the assessment and enforced; so late is not an option. Note that the Canvas page associated with this course will have submission areas that are time triggered; if you are late, submissions will be blocked. Note: If a programming assignment is "due" on, say, a Monday, I will allow you to submit your programs up to 4 am of the following day. Thus, a program due on a Monday can safely be submitted up until early Tuesday morning at 4 am. In short, if you are willing and/or need to work on and test your solution late into the night that it is due, you are granted some grace time. Note: See the paragraph above: *due is due*.

Honor Code Revisited:
It goes without saying that all submitted work will be the student's own, in keeping with the Wheaton Honor Code, unless the assignment has assigned groups. For labs, you may get "help" from fellow classmates, but remember that all completed work must be your own. Use discretion; don't ask another student nor a chatBot for "the" answer or for lines of code. One good suggestion: don't copy/paste from your chatBot into your own code. However, I do encourage you to discuss (with other students or your bots) the problem in general, such as the type of statements or functions one might use. For programming assignments, your answers and software must be your own from beginning to end. Here is an analogy. Almost no one would every "use/steal" a line or two from another person's poem. Consider it the same with your programs. Don't "borrow/use" lines or sections of code from another classmate. Your program is (like) your poem; everyone's program should be unique. Be wise. If a fellow student is asking you for too much help, be honest and remind them your program is just that, *your* program.

Homework: It is expected that you spend at least 2^{++} hours on reading and practice problems for every 50 minutes of lecture. This computes to at least 6 hours of work in the Data Structures texts per week. This should be done throughout the semester and not just when studying for quizzes or exams. The material is cumulative in a big way; for example, week 5 depends heavily on weeks 1 through 4. It is expected that you spend at least 6 hours per week on your current programming assignment. WARNING: Programmers typically underestimate the time it takes to complete a software project; 6 hours per week on your programming assignment may be one of those "underestimations."

Labs: The labs are a critical part of the course. In almost all cases, the current lab will be preparing you for the current programming assignment. That is, if you complete and understand the lab, you should be well on your way to a solution for the programming assignment. In order to best grasp the material in some labs, I strongly suggest that you completely redo any labs that you find difficult. (Read that last sentence again, unless of course you've already reread it once).

Quizzes: In addition to two exams spaced evenly over the semester, you will be quizzed regularly (the dates TBA, for example two quizzes before Exam1 and two others before Exam2). Each quiz may address material that has been assigned in the text, but not yet presented by the instructor in class. NOTE: you'll have to read the material and practice the sample problems to be successful with the quizzes. Quizzes start at the beginning of a lecture and last approximately 10-15 minutes. After some quizzes, immediate (peer) grading will be followed by a discussion, which will focus on the concepts in the quiz.

Quizzes and Exams: There will be no makeup quizzes nor exams, nor will the lowest grade be dropped. If you are an athlete, in the theatre, and/or have a conflict with a quiz date, please see me well beforehand. When studying for exams and quizzes, have someone pick problems from the notes and text, mix them up, and present them to you as if you were taking a quiz (not telling you from what section a problem comes). Use your chatBot to generate quiz questions (e.g., Gemini's "Guided Learning" mode). While practicing, do not peek at the textbook or an earlier solution. NOTE: it's easy to "read" code; it's much more difficult to "write" code from scratch. Quizzes and exams will sometimes ask you to write code from scratch; practice ownership of the concepts!

HELP

I have listed my office hours on the syllabus, but be bold: schedule a time to meet, for example, email me and ask to meet at *such and such* a day/time. Wise students peek at my Google Calendar so you know when I am busy and thus unavailable. Study, study, study, and discuss the material with me and your peers as often as you can.

Please don't wait too long before you see me;
a quick chat in my office can often clear things up.
Yes, I will help you with your programs! 😊
I'm here a lot...

Accessibility Statement
Wheaton College is committed to providing equitable access and supportive services for all students to fully access and thrive in the academic, residential and social aspects of student life at Wheaton College. Affirmatively, Wheaton provides appropriate accommodations for eligible students with documented disabilities to afford equal access to educational programs and services. Individuals with disabilities and other access concerns requiring accommodations or information on accessibility should reach out to Accessibility Services at the Filene Center, either via email at accessibility@wheatoncollege.edu or via phone at (508) 286-3794.