COMP 241 — Computer Science III: Data Structures and Algorithms — Fall 2024

Instructor:	Nate Phillips
Times:	2:00 PM-3:15 PM Tuesday/Thursday
Classroom:	Briggs 119
Course website:	`http://www.cs.rhodes.edu/241` or `https://ncp38.github.io/cs241-f24`
Email:	phillipsn@rhodes.edu
Office:	Briggs 210
Office hours:	Tues/Thurs 3:15-4:15 PM & Wed 1-3 PM
	Also available by appointment and via Zoom

Official Course Description: An introduction to the fundamental concepts of data structures and the algorithms that arise from them, using the object-oriented design paradigm. Data structures covered include stacks, queues, linked lists, hash tables, trees, and graphs. Other topics include an introduction to iterative and recursive algorithmic strategies and basic algorithm analysis.

Unofficial Course Description: So far, you have acquired proficiency in programming. This course will start your transformation from a programmer to a computer scientist. One of the important tasks of a computer scientist is to make efficient use of computational resources. This course will teach you the different ways of organizing data to facilitate such efficient use, and will also discuss efficient techniques to perform some fundamental operations in computer science.

You will use the techniques discussed in this course to solve commonly encountered computational problems efficiently. This course will also teach you to analyze the efficiency of your programs in a mathematically rigorous manner. Material you learn in this course is critical to your becoming a good software developer later.

This course will focus on fundamental algorithms, mathematical analyses of those algorithms, and the use of basic and advanced data structures. Among the specific data structures covered are lists, stacks, queues, maps, trees and graphs. Recursion will also be covered. Finally, some standard computer science algorithms (sorting and searching) will be discussed.

Though we will do a good deal of programming in this course, the focus of the course is not learning to program, but rather the efficiency of programs and programming. In this course, we will examine various meanings of "efficient." For instance, it is efficient to (a) minimize the running time of code, especially code that is destined for re-use; (b) minimize the memory and storage needs of code, recognizing that there may be a trade-off between speed and memory requirements; (c) re-use code, instead of re-writing code; and (d) select only the features you need to solve a particular problem without having to use costly extra features you do not need.

Course Objectives: At the end of this course, you should be able to:

- Describe fundamental data structures, algorithms and programming techniques, including lists, stacks, queues, search trees, hash tables, several sorting algorithms, and search algorithms for graphs.

- Apply the techniques from the course when solving programming/algorithmic problems. These techniques include methods for sorting and searching, basic graph algorithms, recursion, dynamic programming, and time and space analysis of programs.
- Select the best algorithm and/or data structure when solving a given programming problem.
- Analyze the time and space required for the execution of a program, as well as the correctness of a program.
- Formulate a given programming task as an algorithmic problem, in order to select the best method for solving it.
- Combine and modify algorithms and data structures, in order to design an efficient program.

Course Topics:

- Abstract data types (ADTs) and data structures
- Asymptotic analysis
- List ADT: arrays and singly- and doubly-linked lists
- Stack and queue ADTs
- Set and map ADTs: trees, binary search trees, tree traversals, hash tables
- Sorting: $O(n^2)$ and $O(n \log n)$ sorting algorithms
- Priority queue ADT: heaps
- Graph ADT: adjacency matrix and adjacency list representations, Dijkstra's algorithm

Text: There is no required textbook for this class. Materials will be distributed in class or online.

Prerequisites: The course assumes successful completion of CS142 and CS172.

Coursework:

	Tentative weight	Tentative date
Programming projects	40%	
Written homework	20%	
Exam 1	12.5%	Thursday, October 3, in class
Exam 2	12.5%	Thursday, November 7, in class
Comprehensive final exam	15%	Monday, December 16, 5:30 PM

Grades of A–, B–, C–, and D– are correlated with final course grades of 90%, 80%, 70%, and 60%, respectively. If your final course grade falls near a letter grade boundary, I may take into account participation, attendance, and/or improvement during the semester.

Assignments are due at 11:59pm unless otherwise specified. **Late work is unprofessional** and will generally be **penalized at a rate of 8% per 24 hours late** or 1 point per day for homework assignments. If the assignment is submitted three days after it is due, it will not be accepted except in special cases, which might include sickness, injury, or other unforeseen events. In these situations, you should contact the professor as soon as is feasible to come up with a plan for submission.

Office Hours: In addition to regular office hours, I am often available immediately before or after class for short questions. You don't need an appointment to see me during regular office hours; you can just come by.

Don't be shy about coming by my office or sending me emails!! My goal for this course is to help you understand course concepts and reach your full potential. I think proactive communication is important and necessary in achieving this goal.

Attendance: Prompt attendance is expected for each class and is necessary for your success. If you know ahead of time that you will be absent from class for any reason, please discuss this with me at the beginning of the semester or as early as possible. Otherwise, if your attendance deteriorates, you may be referred to the dean and asked to drop the course. Note that in-class assignments or bonuses, if missed, will not be available to retake.

Attendance and participation may be considered when assigning a final grade. Attendance will be checked each class lecture period, with consequences for **excessive absences (more than five)**. For each additional absence, points will be deducted from your grade, up to and including **dropping a letter grade**.

Workload: It is important to stay current with the material. You should be prepared to devote at least 2–3 hours outside of class for each in-class lecture. In particular, you should expect to spend a significant amount of time for this course working on the computer trying example programs and developing programming assignments. For the projects, it is expected to take **an average of 25 hours** to complete. It is recommended to start and complete these assignments early; do not wait until the last minute to start your programming assignments.

Even with intentional focus and preparation, it is possible for issues to come up that delay assignment submission. As such, a **once-a-semester extension** is available upon request, allowing for up to 48 hours' grace for submissions. This extension may not be used for the last project in the semester but is otherwise available.

You are encouraged to form study groups with colleagues from the class. The goal of these groups is to clarify and solidify your understanding of the concepts presented in class, and to provide for a richer and more engaging learning experience. However, you are expected to turn in your own code that represents the results of your own effort.

Getting Help: Students can get help with their coursework several different ways. The first place to reach out for help is to come to office hours. Scheduled office hours are listed on Canvas, the class website, and this syllabus. Additional meeting times are available by request. You may also email the instructor.

There will be tutoring available five nights a week specifically for this course. More information will be provided once the tutoring schedule is determined.

Programming Assignments:

- All programs assigned in this course must be written in Java, unless otherwise specified. When turning in assignments, submit only the Java source code files (`.java`) and other files necessary to allow your program to run (image files, data sets, etc.). Do not submit any files generated by the IDE (e.g., `.class`).

- I recommend backing up your code somewhere as you're working on your assignments. Computer crashes or internet downtime are not valid excuses for missing a deadline.
- Grades for programming projects will be based on correctness of the program output, efficiency and appropriateness of the algorithms used in the code, and style and documentation of the source code. In addition, bonus points may be offered on assignments for meeting particular challenges or for creative or excellent program functionality.
- To ensure your programs recieve as high a score as possible:
  - **Start work early**, and turn your programs in on time! Even if you are struggling to resolve a minor issue, it is often better to turn the program in on time than to spend a few extra days working on the issue.
  - **Look at the programming assignment page**–typically, there is a clear list of milestones your program needs to accomplish and documents to turn in. This is a good list to review before you start and before you submit your assignment.
  - **Create a plan.** How are you going to:
    - Budget time for your program in light of your other commitments?
    - Use what you've learned to solve the problem? Often, thinking out and designing the program beforehand results in a much better (and faster!) process.
  - **Document any errors** you are aware of in your program! This enables your instructor to help you understand/address the underlying issue and is a sign of having tested your program well.
  - Finally, don't hesitate to **come see me or the tutors** if you get stuck. We're here to help you understand and grow in your programming knowledge!

Rules for Completing Assignments Independently

- Unless otherwise specified, programming assignments handed in for this course are to be done *independently.*
- Talking to people (faculty, other students in the course, others with programming experience) is one of the best ways to learn. I am willing to answer your questions or provide hints if you are stuck, and the tutors are an excellent resource for programming help and instruction as well. But when you ask other people for help, sometimes it is difficult to know what constitutes legitimate assistance and what does not. In general, follow these rules:
  - **Rule 1: Do not look at anyone else's code for the same project, or a different project that solves a similar or identical problem.**
    Details: "Anyone else" here refers to other members of the class, people who have taken the class before, people at other schools enrolled in similar classes, or any code you find online or in print. "Similar or identical problem" here should allow you to look at code that uses techniques applied in different situations that you can then adapt to your project. However, if you find yourself copying-and-pasting code or directly transforming code line by line to fit into your program, then that is considered plagiarism. This includes code from the internet, automated programs like ChatGPT, or other people; as such, be careful that your programs are your own work and aren't copied from other sources.

Exception: You may help someone else debug their program, or seek assistance in debugging yours. However, this requires the person writing the code being debugged to have made a good-faith attempt to write the program in the first place, and the goal of the debugging must be to fix one specific problem with the code, not re-write something from scratch.

- **Rule 2: Do not write code or pseudocode with anyone else.**
  Details: You must make a good faith effort to develop and implement your ideas independently before seeking assistance. Feel free to discuss the project *in general* with anyone else before you begin and as you're developing your program, but when you get to the level of writing code or pseudocode, you should be working independently.

A violation of these rules constitutes plagiarism and is not acceptable behaviour for Rhodes students. Such violations will be dealt with harshly and will have consequences that are reflected in your grades, which can include failing the assignment, dropping a letter grade for the entire course, or even failing the course. However, if you have any questions about what is acceptable for this course, please send me an email with the details relating to your specific case.

At any point, you may be asked to explain your code or ideas or to reflect on aspects of your coding style, approach, or the assignment itself. This self-analysis is part of your assignment, but, even more, thoughtful responses will help you better understand your own code and design practices!

The underlying idea is that course work and outside assistance should genuinely help you to learn the material (as opposed to just getting the assignment done). Programming assignments are graded as a benefit to you; they are your chance to show what you have learned under circumstances less stressful than an exam. In return, I ask only that your work fairly reflect your understanding and your effort in the course.

Coding Style: Designing algorithms and writing the corresponding code is not a dry, mechanical process, but an art form. Well-written code has an aesthetic appeal while poor form can make other programmers (and instructors) cringe. Programming assignments will be graded based on correctness and style. To receive full credit for graded programs, you must adhere to good programming practices. Therefore, **your assignment must contain the following:**

- A comment at the top of the program that includes the author of the program, the date, and a brief description of what the program does.
- Concise comments that summarize major sections of your code, along with a comment for each function in your code that describes what the function does.
- Meaningful variable and function names
- Well-organized code
- White space or comments to improve legibility

Class Conduct:

- I encourage everyone to participate in class. Raise your hand if you have a question or comment. Please don't be shy about this; if you are confused about something, it is likely that someone else is confused as well. Teaching and learning is a partnership between the instructor and the students, and asking questions not only helps you understand the material, it also helps me better connect with you in my teaching.

- When in class, only use cell phones or other electronic devices for classwork and keep the volume on silent.
- If you cannot make it to class for whatever reason, make sure that you know what happened during the lecture that you missed. The information from that class is your responsibility. (A good way of doing this is to ask a classmate!)
- If you have to leave a class early, inform the instructor in advance.

Additional policies: On the class webpage, there are additional class and college policies covering accommodations, academic integrity, diversity, sexual misconduct disclosure, and recording lectures.