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

Instructor:	Phillip Kirlin (Office: Briggs 209)
Meetings:	Tu/Th 9:30am–10:45 or 11am–12:15pm
Course website:	See Canvas
Email:	kirlinp@rhodes.edu (please include "CS 241" somewhere in the subject)
Office hours:	See website for scheduled office hours. I am also available by appointment.

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, different 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.

Text: There is no official textbook for the class. Reference materials will be provided online.

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

Coursework:

	Tentative weight	Tentative date
Programming projects	30%	
Written homework	20%	
Midterm 1	15%	Tuesday, February 20, in class
Midterm 2	15%	Thursday, April 4, in class
Comprehensive final exam	20%	Wed., May 1, 8:30am; or Sat., May 4; 1:30pm

Grades of A–, B–, C–, and D– are guaranteed 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.

Written assignments are due at the beginning of class on the assigned date. Programming projects are due on Canvas by 11:59pm on the assigned date. Homework assignments should be written neatly. Poorly written work will not be graded. All pages of assignments should be stapled together.

Course Topics:

- Abstract data types 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

Late Work and Makeup Assignments: In general, late work will not be accepted without arranging an extension in advance with the instructor, and will often come with a late penalty. Please make every effort to submit assignments on time.

If you have a valid reason for a makeup exam, inform your instructor as soon as you know. A valid reason is a medical emergency, a death in the family, religious observation, a college-sponsored off-campus activity, and, quite frankly, very little else. Generally, assignment extensions will only be granted for *unplanned* circumstances (e.g., the first two reasons above).

Office Hours: In addition to regular office hours, am also available immediately after class for short questions. Appointments for office hours should be scheduled in advance when possible, even if it's only five minutes ahead of time. Outside of regular office hours, feel free to email me or send me a message on Slack, and if I have time, I'll try to help you. If I don't have time at that moment, we'll set up an appointment for a different time. Don't be shy about sending me a Slack message or sending me email if you can't make my regular office hours. I always set aside time each week for "unscheduled" office hours.

Attendance: Attendance is expected for each class. If your attendance deteriorates, you will be referred to the dean and asked to drop the course. Attendance, participation, and apparent overall improvement trend may be considered in assigning a final grade. Attendance will be checked each class lecture period. After five unexcused absences, each additional absence will reduce the final grade for the course by one 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 a computer trying example programs and developing programming assignments. Do not wait to the last minute to start your programming assignments.

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.

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 know what I'm doing right or wrong.
- If you cannot make it to class for whatever reason, make that you know what happened during the lecture that you missed. It is your responsibility, and nobody else's, to do so. The best way to do this is to ask a classmate.

Collaboration: Students should talk to each other about the subject matter of this class and help each other. It is fine to discuss the readings, lectures, and problems and ask questions about them. I encourage such questions in class as well as elsewhere. However, there is a line past which you must not go, e.g., copying a solution from a fellow student, book, website, artificial intelligence tool such as ChatGPT, etc., will cause you to fail the course, or worse. If a significant part of one of your solutions is due to someone else, or something you've read, then you must acknowledge your source. Failure to do so is a serious academic violation. Of course, even after you acknowledge your source you must still understand your solution and write it in your own words. Copying a solution from someone or someplace else will result in failure even if you acknowledge your source, unless you put it in quotation marks and say something like, "Here is Amy's solution, but I don't understand it enough to absorb it and write it in my own words." However, this won't get you much — if any — credit.

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`); do not submit any files generated by the IDE.
- Back up your code somewhere as you're working on your assignments. Computer crashes or internet downtime are not valid excuses for missing a deadline.
- Programming grades will be graded on correctness of the program output, efficiency and appropriateness of the algorithms used in the code, and style and documentation of the source code.
- Grades are assigned to programs as follows by this general guideline:
  - A (100 pts): The program is carefully designed, efficiently implemented, well documented, and produces clearly formatted, correct output.
  - A- (93 pts): The program is an 'A' program with one or two of the minor problems described for grade 'B.'
  - B (85 pts): The program typically could easily have been an 'A' program, but it may have minor/careless problems such as poor, inadequate, or incomplete documentation; several literal values where symbolic constants would have been appropriate; wrong file names (these will be specified per program assignment); sloppy code format; minor efficiency problems; etc. (This is not an exhaustive list.) You would be wise to consider 'B' the default grade for a working program — this might encourage you to review and polish your first working draft of an assignment to produce a more professional quality final version of your program.
  - C (75 pts): The program has more serious problems: incorrect output or crashes for important special cases (the "empty" case, the "maxed-out" case, etc.), failure to carefully follow design and implementation requirements spelled out in the assignment, very poor or inefficient design or implementation, near complete absence of documentation, etc.
  - D (60 pts): The program runs, but it produces clearly incorrect output or crashes for typical cases. Or, it may deviate greatly from the design or implementation requirements stated in the assignment description.
  - F (35 pts): Typically, an 'F' program produces no correct output, or it may not even run. It may "look like a program" when printed as a hard copy, but there remains much work to be done for it to be a correct, working program.

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 always willing to answer your questions or provide hints if you are stuck. 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 (including generated by AI tools such as ChatGPT) 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.
  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.

The underlying idea is that you are entitled to seek assistance in ways which will 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 or dates, 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
- Avoidance of large blocks of copy-and-pasted code

Additional Information: To streamline this syllabus, I have moved a number of policies common to all my classes to a separate document called "Additional Course Policies." Those policies should be interpreted as a part of this syllabus.