COMP 241 — Computer Science III: Data Structures and Algorithms — Spring 2018
CRN 28286

Instructor:	Phillip Kirlin
Meetings:	Tu/Th 2-3:15, Briggs 119
Course website:	`http://www.cs.rhodes.edu/241`
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: Larry Nyhoff, *ADTs, Data Structures, and Problem Solving with C++*, 2nd edition.

Supplemental materials will be distributed in class. The textbook is designed to serve as a reference and there will be material discussed in class that the textbook does not cover.

Prerequisites: The course assumes successful completion of CS142. CS172 is a co-requisite.

Coursework:

	Tentative weight	Tentative date
Programming projects	35%	
Written homework	15%	
Midterm 1	15%	Thursday, February 22, in class
Midterm 2	15%	Tuesday, March 27, in class
Comprehensive final exam	20%	Wednesday, May 2, 5:30pm

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.

Office Hours: In addition to regular office hours, am also available immediately after class for short questions. You never need an appointment to see me during regular office hours; you can just come by. Outside of regular office hours, feel free to stop by my office, 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 coming by my office 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.

Programming Assignments:

- All programs assigned in this course must be written in C++, unless otherwise specified. When turning in assignments, submit only the C++ source code files (`.cpp, .h`); 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, what constitutes legitimate assistance, and when does it cross the border and become cheating? As a practical guide, use the following rule:

Rule: In working on an assignment, you cannot look at any *correct program* or *correct piece of code* for the same assignment which someone else has written.

- Here are some sample applications of this rule:
  - Q: If my program does not work, may I ask another student to see if they can spot what is wrong with it?
    A: Yes, since you are showing them incorrect code.
  - Q: May they suggest the needed changes (e.g.: "You need to assign `x = 0` up here.")?
    A: Yes.
  - Q: May they write the needed changes for me?
    A: No, since then they would be showing you correct code.
  - Q: May they show me one of their programs from earlier in the course which uses the same technique?
    A: Yes, since that is not for the same assignment.
  - Q: May the two of us compare the outputs from our programs?
    A: Yes. Output is not program code.
  - Q: If our outputs differ, may we compare our programs to see what the difference is?
    A: No. Assume one of them is correct; the other person should not be looking at it.
  - Q: May I write code together with someone else?
    A: No. If you're writing code together with someone else, then you'll end up seeing their program and your program at the same time. Most likely one of the programs is correct, and the other person should not be looking at it.

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

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.
- Do not use your cell phone while in class, and keep the ringer 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. It is your responsibility, and nobody else's, to do so. The best way to do this is to ask a classmate.
- If you have to leave a class early, inform the instructor in advance. It is rude to walk out in the middle of a lecture.

Students With Disabilities: If you have a documented disability and wish to receive academic accommodations, please contact the Office of Student Disability Services at x3885 as soon as possible.

Academic Integrity: Plagiarism, cheating, and similar anti-intellectual behavior are serious violations of academic ethics and will be correspondingly penalized. If you are concerned about a possible violation of this kind, please talk with me. I understand that being a student at Rhodes can be stressful sometimes and you will have many demands on your time. However, I would much rather have you turn in a partially-completed assignment or do poorly on a test than have you violate the Rhodes Honor Code. I can — and very much want to — help you if you don't understand the material, but violations of academic integrity will be dealt with harshly.

Unless otherwise specified, everything you submit in this course must be your own work and represent your individual effort. These are all included in the definition of reportable Honor Code violations for this course: copying all or part of a solution to a problem, downloading a solution from the internet and submitting it as your own, having someone else provide the solution for you, or allowing someone else to copy from you. If you have any doubt about what type of behavior is acceptable, please talk with me.

Diversity: A diverse learning community is a necessary element of a liberal arts education, for self-understanding is dependent upon the understanding of others. We are committed to fostering a community in which diversity is valued and welcomed. To that end any discrimination or harassment on the basis of race, gender, color, age, religion, disability, sexual orientation, gender identity or expression, genetic information, and national or ethnic origin, will not be tolerated in the classroom.

We are committed to providing an open learning environment. Freedom of thought, a civil exchange of ideas, and an appreciation of diverse perspectives are fundamental characteristics of a community that is committed to critical inquiry. To promote such an academic and social environment we expect integrity and honesty in our relationships with each other and openness to learning about and experiencing cultural diversity. We believe that these qualities are crucial to fostering social and intellectual maturity and personal growth.

Intellectual maturity also requires individual struggle with unfamiliar ideas. We recognize that our views and convictions will be challenged, and we expect this challenge to take place in a climate of open-mindedness and mutual respect.

Course Topics:

- Review of C++ concepts, object-oriented programming
- Abstract data types and data structures
- Asymptotic analysis
- Vector ADT
- List ADT: 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