CS 241
Data Structures and Algorithms
Section 01, CRN: 18180
Fall 2007

Syllabus & Course Policies

INSTRUCTOR: Robert England
EMAIL: englandr@rhodes.edu
OFFICE: OH 419
PHONE: 843-3725

TEXT: *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*,
by Bruno R. Preiss.
Supplemental material may be distributed in class as needed.

COURSE DESCRIPTION:

This course builds on the foundation provided by the CS 141 - CS 142 sequence to introduce the fundamental concepts of data structures and the algorithms that proceed from them. We will review the underlying philosophy of object-oriented programming, explore recursion as a design strategy, carefully examine fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), and cover the basics of algorithmic analysis.

PREREQUISITE:

CS 142.

TOPICS: [not necessarily in this order]

Review of elementary programming concepts

Fundamental data structures
- Stacks
- Queues
- Linked lists
- Hash tables
- Trees
- Graphs

Object-oriented programming
- Object-oriented design
- Encapsulation and information hiding
- Classes
- Separation of behavior and implementation
- Class hierarchies
- Inheritance
- Polymorphism

TOPICS, cont.:

Fundamental computing algorithms
- O (N lg N) sorting algorithms
- Hash tables, including collision-avoidance strategies
- Binary search trees
- Representations of graphs
- Depth- and breadth-first traversals

Recursion
- The concept of recursion
- Recursive mathematical functions
- Simple recursive procedures
- Divide-and-conquer strategies
- Recursive backtracking
- Implementation of recursion

Basic algorithmic analysis
- Asymptotic analysis of upper and average complexity bounds
- Identifying differences among best, average, and worst case behaviors
- Big "O," little "o," omega, and theta notation
- Standard complexity classes
- Empirical measurements of performance
- Time and space tradeoffs in algorithms
- Using recurrence relations to analyze recursive algorithms

Algorithmic strategies
- Brute-force algorithms
- Greedy algorithms
- Divide-and-conquer
- Backtracking
- Branch-and-bound
- Heuristics
- Pattern matching and string/text algorithms
- Numerical approximation algorithms

Overview of programming languages
- Programming paradigms

Software engineering
- Software validation
- Testing fundamentals, including test plan creation and test case generation
- Object-oriented testing

GRADING:

Program assignments:	50%	
2 in-class tests:	25%	[Thurs 20 Sept; Thurs 25 Oct]
Final exam:	25%	[Tues 11 Dec, 5:30pm]

TESTS:

Both of the regular tests and the exam will be closed book, closed notes. Typical test format is a list of multiple choice questions, one code writing problem, and one code trace problem, though there may be slight variations to this format.

ATTENDANCE:

Attendance will be checked each class lecture period. As a bonus to those who attend regularly, each student who misses no more than 2.0 class meetings will be allowed to omit selected questions on the final exam worth a total value of approximately 12 to 15 points out of 100. These 2.0 allowed absences are to cover emergencies and do not have to be explicitly excused --- no excuses for other absences will be accepted with regard to this bonus. Each tardy (arrival after the start of class) counts as 0.5 absence. After class, you should be careful to remember to sign the attendance sheet if you were tardy; otherwise, you will be counted absent. After 5 unexcused absences, each additional absence reduces the final grade for the course by one letter grade.

If you can verify that you were unavoidably absent because of a school sponsored event, the absence will not count against your 2.0 allowed absences, but you will still be responsible for all material covered in class, of course --- be careful to get lecture notes from a colleague for the class meetings you miss. The instructor's own lecture notes will not be made available for copying or review.

Aside from the possible effect of the exam bonus on your grade for the course, experience has shown that there is an unmistakable relationship between class attendance and the degree of mastery of the material presented in this course. 'Nuff said.

PROGRAM ASSIGNMENTS:

All programs assigned in this course must be written in C++. Each program assignment will each be awarded a letter grade A through X:

A: (100 pts)
'A' programs are carefully designed, efficiently implemented, well documented, and produce clearly formatted, correct output.

A- : (94 pts)
This is an 'A' program with one or two of the minor (?) problems described for grade 'B'.

B: (88 pts)
A 'B' 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); incomplete hard copies; sloppy source code format; minor efficiency problems; minor (?) memory leaks; 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.

PROGRAM ASSIGNMENTS, cont.:

C: (75 pts)
A 'C' program has more serious problems: incorrect output 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 description, very poor or inefficient design or implementation, near complete absence of documentation, etc.

D: (60 pts)
A 'D' program compiles, links, and runs, but it produces clearly incorrect output 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 compile. 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. Still, as a last resort, an 'F' program is better than no program turned in at all.

X: (0 pts)
A grade of 'X' will be recorded for each program not turned in. Each 'X' has the additional side effect of lowering the final grade for the course by one letter.

A fully documented sample program that you can use as a model for source code format will be distributed with or before the first programming assignment. The first line of each program source code file submitted for credit must be a comment that states the name of the source code file. Each student is responsible for keeping a back-up copy on disk of all source code turned in for an assignment. Failure to do so could result in loss of credit for an assignment.

Programming assignments must be turned in on the due date to receive full credit. Programs will be accepted late, but with a penalty of one letter grade per day. You get a three late day allowance.

Recommendation: *keep this list* of policies handy as a partial checklist of requirements to review before turning in each completed assignment!

ACADEMIC INTEGRITY:

All programs and tests must be the student's *own* work. Copying all or part of an assignment, or downloading code from the Internet and submitting it as your own, or having someone else write code for your assignment, or having someone else debug your assignment, or *allowing* someone else to copy from you, or coding or debugging someone else's assignment --- these are all included in the definition of reportable Honor Code violations for this course. If you have any doubts about whether or not a program development practice on an assignment is acceptable, clear it with the instructor before proceeding.

The instructor reserves the right to alter this syllabus as necessary.