CS 132 Exam #1 - Study Suggestions

* **last modified: 2-16-05**

* The test covers through HW #3, the Week 5 Lab Exercise Exercise, and material through the 2-14-05 lecture/2-16-05 lab.
* Anything that has been covered in **assigned reading** is fair game;
* Anything that has been covered in **lecture** is fair game;
* Anything covered in a **course handout** is fair game;
* Anything that has been covered in a **lab exercise** or **homework assignment** is ESPECIALLY fair game.

* But, these are some especially-significant topics to help you in your studying for the exam.

* You are responsible for being familiar with, and following, the class **style** guidelines.

* The exam will be closed-book and closed-notes, and you are expected to work individually.
* Test format: will likely be short answer, possibly with a smattering of multiple-choice questions.
    * All you need to provide is a pen or a pencil;

    * EXPECT to have to read and write C++ code, pseudocode, UML notation.

* note that you could definitely be given code and asked questions about it, as in the Week 4 Lab Exercise (answering questions about the different sort implementations).

* the only aspect of namespaces that you are responsible for on this exam is that you need to use `using namespace std;` after #include'ing standard libraries in modern, standard C++.

* data structures
    * what is a **data structure**? an organized collection of data...

    * what is an **abstract data type** (adt)? a collection of data PLUS all of the operations for acting on that data;

* phases of software development and program design recipe handouts
    * be comfortable with the basic phases of software development as given in the course text; be comfortable with the **basic function design recipe** discussed.

        * especially: for a function,
            1. figuring out what data is involved (data analysis),

2. then writing a CONTRACT,

3. then writing the HEADER corresponding to that contract (here, remember, we mean the first line of the *implementation*/definition, NOT the prototype/declaration/what goes in the .h file)

4. then writing the PURPOSE, INCLUDING the parameter names appropriately,

   writing PRECONDITIONS and POSTCONDITIONS if called for,

5. then writing the EXAMPLES, actual example calles of the function, including what the function returns or does as a result of that call,

6. and only THEN devising its algorithm, and then translating that algorithm into code.

* what is the class "syntax"/notation for a function **contract**? Given a non-main function or its description, you should be able to write a contract using this syntax/notation.

* in this class, what should be incorporated into the Purpose: statement of a function that has parameters?

* what is a precondition? what is a postcondition? what are the expectations for these?
* you should be able to read and write **assert** statements to verify a function's preconditions (for preconditions for which such tests are reasonable); you should know what happens when an assert's condition is false.

* what goes in the Examples: section of a "regular" function's opening comment block, in this class? How do we write these when the function returns, say, an int?
* when should you come up with specific examples for a function or method? (BEFORE you write it!)

* Given a function and/or its description, you should be able to write examples that adequately test it (cover all major categories of input and boundaries between those categories).

* be comfortable reading and appropriately writing code using EXIT_SUCCESS and EXIT_FAILURE. (remember the class conding standards regarding these.)

    * should be able to read, write tester programs (testing main functions) as you have been doing in class assignments.

* Lab and C++-related details
    * how can you compile a C++ function on cs-server? how can you compile and link a C++ program on cs-server?

    * what should go in a .h file for a non-main function being written in its own file? How does another function use a non-main function written in its own file?

    * how can you redirect screen output to a file in UNIX?

    * how should you declare a named constant in this class? (be familiar with both the syntax, its meaning, and the class style standards for named constants)
        * Within a class, how many "copies" of a thing declared to be **static** are there?

* running time analysis
    * what is big-O notation? What does it mean? How can it be useful?

    * given a formula representing the number of steps that some algorithm requires for a problem of size n, you should be able to give the big-O notation for such an algorithm (for example, as in Week 2 Lab Exercise problem #1)

    * what is average-case run-time complexity? worst-case? best-case? What are the differences between these?

    * you should know (or be able to figure out) the run-time complexities for "simple" operations, and express them using big-O notation;
    * you should know (or be able to figure out) the average-, worst-, and best-case time complexities for:
        * sequential search and binary search
        * selection sort, insertion sort, bubblesort, merge sort, quicksort, and radix sort

    * what phrase is equivalent to O(1)? to O(n)? to O(log n)? to $O(n^2)$? to $O(2^n)$?
        * except for $O(2^n)$, you should be able to give an example of an algorithm that takes that average-case running-time; you should be able to give an example of an algorithm that average-case running time for O(n log n), also.

    * (remember: in computer science, when log n is written, base 2 is assumed.)

* recursion
    * what is a recursive definition? what is a recursive function?

    * requirements for "good" recursion!

    * given a function --- does it demonstrate "good" recursion or not? Why? Why not?

    * what is a "base" case? Does every recursive function need one? Can a recursive function have more than one?

    * what is a recursive case? Does every recursive function need one? Can a recursive function have more than one?

    * given a recursive function, you should be able to tell what it would produce for a call of that function; given a specific call to that function, you can give the results of that call.

    * you may be asked to write a recursive function.

* searching
    * you are responsible for knowing sequential search and binary search.

    * you should be able to describe the basic algorithm for each; you should know their run-time complexities.

    * if code was given,you should be able to recognize which of the above is being implemented within that code.

    * (frankly, you should be able to code some version of sequential search at the drop of a hat...)

    * you should be able to reason about variations on these basic algorithms

* sorting
    * you are responsible for knowing selection sort, insertion sort, bubblesort, merge sort, quicksort, and radix sort

    * you should be able to describe the basic algorithm for each; you should know their run-time complexities.

    * if code was given, you should be able to recognize which of the above is being implemented within that code.

    * you should be able to reason about variations on these basic algorithms (using bubblesort within quicksort when the list size is sufficiently small, for example)

* introduction to container classes and bags
  * what do we mean by a container class?

  * for this exam, you are responsible for the idea of bags, for the bag class as discussed in lecture, and for how such bags can be used; you should be comfortable with their static-array and dynamic-array implementations, also.

  * you should be able to read and use a data structure with a given "UML" such as that given for bag in-lecture (and available from the course web page). You should be able to answer questions based on reading it, and should be able to write code using such a class (as you did in the Week 5 Lab Exercise).

  * what is an abstract data type (adt)? What are some of the benefits of using a well-designed adt class for a data structure within a program? Be comfortable, too, with such terms as information hiding and abstraction.