

CS 131 - Final Exam Study Suggestions

last modified: 12-10-10

- * Remember: anything covered in lecture, in lab, or on a homework, is FAIR GAME.
- * You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will not be returned.

Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer, and you are expected to work individually.

(Studying *beforehand* in groups is an **excellent** idea, however!)

- * Note that final exams are **not** returned. I will retain them for at least 2 years, and you may come by my office to see them if you wish after grades are posted into Moodle.
- * The final exam is cumulative in CONCEPTS, but the LANGUAGE of the exam will be C++, to reduce cross-syntax confusion.

You will not be writing any Racket for the final.

- * so, you should still use the review suggestions for Exam 1 and Exam 2 for studying for the final exam, but substitute C++ for Racket in the Exam 1-related material. (Note that these review suggestions are still available from the public course web page, under "Homeworks and Handouts".)
- * and, you should still use your Exam 1 and Exam 2 for studying for the final exam, but imagine how those Exam 1 answers would look written in C++. Some questions may be similar in style to those asked on the first two exams, except you would answer them using C++.
- * (yes, this DOES mean I could ask questions involving recursion --
 - * but involving C++ syntax instead of Racket syntax -- a recursive C++ function is still just a function calling itself --
 - * and it STILL has one or more base cases that aren't recursive,
 - * and it STILL has one or more recursive cases that call itself on something SMALLER, so that eventually a base case WILL be reached;)
- * You are expected to follow course style standards in your exam answers, and there may be exam questions about course style standards as well.
- * You should still be comfortable with the design recipe for functions, and should be able to appropriately fill in the opening comment block "templates" we've been using (and the main and .h file templates as well).

Note: the final exam handout will include the `mainfunction` template posted on the public course web page.

- * note that answers may lose points if they show a lack of precision in terminology or syntax; for example, if I ask for a literal or an expression and you give an entire statement, instead; or, if a statement is requested that requires a semicolon, and it is not ended with one.

Interactive input

- * should be quite comfortable using `cin` for interactive input;

"complete" C++ programs, continued

- * At this point, you should be able to write a `main` function;
- * Given all of the files involved in a C++ program, you should be able to give a command to link and load and create a C++ executable program for that program;
- * Consider the precompiler directive `#include`.
 - * When do you need to use this?
 - * How do you `#include` a standard library (what needs to surround its name)?
 - * How do you `#include` the header file for a function or class that you have written (what needs to surround its name)?
 - * For this course, what line should follow all of your `#includes`?
 - * Given a description of what a function is doing/calling, you should be able to write the `#includes` that would be needed for that function;
- * Be aware that we use the precompiler directives `#define`, `#ifndef`/`#endif` in `.h` files to reduce redefinition errors involving `#includes`

while statement continued

- * you should know, at this point, that `for` loops are better for "plain" count-controlled loops than `while` loops;
- * you know that `while` loops are better for **sentinel-controlled loops**, and for "other" kinds of loops.
- * you are expected to know, and use, the **"classic" structure for a sentinel-controlled loop**, when that logic is what is desired;

arrays continued

- * there was some array coverage on Exam 2, but you've done a lot more with arrays since then;
- * expect to have to read, write, and use arrays; you should be comfortable with array-related syntax and semantics, and with common "patterns" for using arrays.
- * **expect it: you will have to write at least one loop that does something involving every element in an array!**
- * how do you pass an array as an argument? how do you declare an array as a parameter? In C++, when an array is a parameter for a function, what should also be a parameter of that function?
- * how can you do something to every element within an array? how can you use every element within an array? what particular statement is especially useful in doing such actions?

- * note that you should be comfortable with arrays of objects as well as arrays of primitive C++ data types;
- * how can you declare a dynamically-allocated array? How do you then dynamically allocate such an array? How do you free the allocated memory when you are done with that array?

classes continued

- * you know what is meant by overloading a method within a class, and how to do it;
 - * you know how one common example of overloading methods is overloading constructors, and making sure to include a 0-argument constructor along with whatever constructor(s) would be useful for the users
 - * (and once you have a 0-argument constructor for a class, you know how to call that 0-argument constructor...)
- * you are now also comfortable with modifier methods (modifiers) and other methods;
- * you should be able to write a class definition and implement that class, given specifications of what is desired; you should be able to write a test function for that class as well, including testing all of its constructor methods, selector methods, modifier methods, and "other" methods;
- * given information about a class and its methods and data fields, you should be able to use that class: declare instances of it (using both zero-argument constructors and multiple-argument constructors), call its selector methods, modifier methods, and "other" methods.
- * you know what a destructor, a copy constructor, and an overloaded assignment operator are each intended to do, and you know when you need to write them for a class; you will not have to write these kinds of methods on the Final Exam, however.

file input/output

- * why might you want a program to be able to read from a file? why might you want a program to write to a file?
- * what C++ standard library is used for the file input/output that we used?
- * how do you set up and open a file for reading in C++? how do you set up and open a file for writing in C++?
 - * ...and how do you close such a file stream when you are done?
- * once you have opened an input file stream, how can you read something from it?
- * once you have opened an output file stream, how can you write something to it?

intro to pointers

- * what is a pointer? What does it do? How can it be used?
- * How do you declare a pointer in C++? How do you set it/initialize it?
- * According to class style standards, if a pointer is currently not "pointing" anywhere, then what

special value should it be set to? What can you `#include` in order to get the special value?

- * how can you make a pointer point to something? how can you affect or use what a pointer points to?
- * I might ask a question involving the boxes-and-arrows notation used in lecture; you might have to fill in the boxes and draw the arrows to complete a diagram based on a code fragment, for example.
- * If a pointer is pointing to an object, how can `->` be used to call the methods associated with the object that the pointer is pointing to?

dynamic memory allocation

- * how can you dynamically allocate memory from the heap/freestore and have a pointer point to that memory?
 - * you should be able to do this for arrays, scalar values, and objects;
- * how can you then free or deallocate this memory when you want the pointer to point somewhere else or when you are simply done with it?
 - * what is the special syntax needed if you are doing this for a dynamically-allocated array?
 - * what might happen if you do not do this?

pass-by-value and pass-by-reference

- * What is a pass-by-value parameter? What is a pass-by-reference parameter? How can you tell them apart? What is the difference between them? What is the difference in the possible `*arguments` between pass-by-value and pass-by-reference parameters?
- * EXPECT at least one question involving pass-by-value and pass-by-reference parameters.
- * input parameters, output parameters, input/output parameters --- what are they? for each, what, in general, type of parameter passing is most appropriate?
- * in terms of class style standards, when is it more appropriate to use pass-by-reference? when is more appropriate to use pass-by-value?

also note...

- * **expect** at least one question in which you are given function headers and asked to write appropriate calls of those functions;
- * **expect** at least one question in which you are asked to write a count-controlled loop to do something a set number of times;