CS 111 - Final Exam Review Suggestions - Fall 2017

last modified: 2016-12-09

- You are responsible for material covered in class sessions, lab exercises, and homeworks; but, here's a quick overview of especially important material.
- 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 the course Moodle site.
- 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++.
- 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 that the Final Exam will include a References handout that includes both the posted `main` function template as well as the `.h` file template and non-`main` function template.
- 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;
  - or, if you are asked for a specific code fragment, and you give an entire function.
- This will be a pencil-and-paper exam, but you will be writing and reading C++ expressions and functions in this format. You will be answering questions about concepts, expressions, and functions as well.
- Your studying should include careful study of posted examples and notes as well as the homeworks (and posted example solutions) thus far.

+=, ++, and more

+=, -=, *=, /=

- What do `+=`, `-=`, `*=`, `/=` mean? How are they used? What are their effects and semantics?
- Be able to accurately read, write code fragments containing these operators.

++ and --

- What are the prefix and postfix increment operators (`++`) and the prefix and postfix decrement operators (`--`)? How are they written? Where are they written? What are their effects and their semantics?
- If you have `int i;` and `i` has been set to some value, what is the difference between `++i` and `i++` (or between `--i` and `i--`)?
- Be able to accurately read, write code fragments containing these.

C++ `while` statement/`while` loop

- Need to be comfortable with the basics of the C++ `while` statement/loop; need to be comfortable with its syntax and semantics, need to understand how it uses mutation of a local variable to implement repetition.
- Should be able to read, write a count-controlled loop (using a `while` loop), a loop that does something a certain number of times.
- Should be very comfortable with the course-expected indentation for `while` loops.
- You should be able to design, read, and write `while` loops; you should be able to read a `while` loop, and tell what it is doing; you should be able to answer questions about what a `while` loop does when it executes.

`for` statement/`for` loop, and `while` loop, continued

- need to be comfortable with the basics of the C++ `for` loop statement; need to be comfortable with its syntax and semantics, need to understand how its uses mutation of a local variable to implement repetition
- when is a `for` loop appropriate? when is a `while` loop more appropriate?
- should be able to read, write a count-controlled loop, a loop that does something a certain number of times.
  - Now that you know about the `for`-loop, that is the loop of choice for a basic count-controlled loop
  - (One might certainly quite reasonably use a `while`-loop in some slight variations of count-controlled loops, however! So being familiar with how to use a `while`-loop in a count-controlled situation can still be useful.)
- you know that `while` loops are better for **sentinel-controlled style loops**, and for "other" kinds of loops;
- you are expected to know, and use, the "classic" structure for a **sentinel-controlled loop** style, when that logic is what is desired;
- you are also expected to know, and use, the **"question"-controlled loop** style, when that logic is what is desired;

- should be **very** comfortable with the course-expected **indentation** for `while` loops and `for` loops;
- you should be able to design, read, and write `while` loops and `for` loops; you should be able to read a `while` loop or a `for` loop, and tell what it is doing; you should be able to answer questions about what a `while` loop or `for` loop does when it executes

C++ 1-dimensional arrays

- Need to be comfortable with the basics of C++ 1-dimensional arrays.
- How do you declare an array? How can you initialize it?
  - Given an array's declaration, you should be able to say what its indices will be.
- How do you access an individual element/an individual cell within an array?
  - Given an array's declaration, you should be able to write expressions representing individual elements/cells within that array.
- How can you do something to every element/every cell within an array? How can you use every element/every cell within an array?
  - You should be able to write a a `for` loop or a count-controlled `while` loop that does something to or with every element within an array.
- 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.
- 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?
- EXPECT IT: you will have to write at least one loop that does something involving every element in an array!

different kinds of C++ functions (now including `void` functions, and functions with no parameters)

- at this point, you have written "pure" functions that expect parameters and return a result;

  you have also seen C++ `main` functions, as well as auxiliary functions that are not so "pure" (they may have side-effects, they may not return anything, they may require no parameters, etc.!)
- Given a function header, you should know how to then write a "legal" call to that function;
  - How is a `void` function called?
  - How is a function expects no parameters called?
  - When a function returns a value, how is it (typically) called? How can it also be called if you just care about its side-effects, and not about what it happens to return?
  - When a function expects one or more parameters, how is it called?
  - EXPECT IT: you **WILL** be given a function header, and be asked to write a "legal" call to that function.

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? how do you set up and open a file for writing?
  - ...and how do you close such a file stream when you are done?
  - because the `open` method requires a `char*` argument, you should know that the `string` class's `c_str` method expects no arguments and returns a `char*` version of the calling `string`.
- once you have opened an input file stream, how can you read something from it? How can you read each line from a file (until the end of file)?
- once you have opened an output file stream, how can you write something to it?
  - know that opening an output file stream for a given name creates a new file with that name if such a file does not currently exist, and deletes its current contents if it does currently exist.
- be comfortable with the `getline` function for reading in a line at a time from a given input stream
  - (and be aware that, if mixing reads using `getline` with reads using the >> operator, you MAY need to use an extra `getline` call to "finish" the preceding line before going on)
- know that the `eof` method (of an input file stream) can be used for reading from a file whose length is not known in advance,

  and that the `fail` method (of either an input file stream or an output file stream) can be used to tell if an `open` of a stream failed, so that your program can react more gracefully in that situation.

Formatting output

- how can you indicate that `bool` values should be output as `true` and `false`?
- how can you indicate that you'd like a newline in your output? how can you put a blank in your output?

Putting all of a C++ program's functions' source code in 1 `.cpp` file

- ...if you do this, what must you do to include `main` function's implementation first? (What has to come before the `main` function's implementation?)
- ...and if you don't do the above, and have the `main` implementation at the end, in what order must the other implementations of the other functions be put?
- (Remember: a C++ program consists of one or more functions, one of which is the special function `main`, where execution starts when the program is run.)

pass-by-value

- what is a pass-by-value parameter? how is the argument's value passed to the parameter in this case?
- for a (non-array) pass-by-value parameter, is the argument changed when the corresponding parameter is changed? Why not? What is the course style standard with regard to changing (non-array) pass-by-value

parameters?

pass-by-value and pass-by-reference

- What is a pass-by-reference parameter? How can you tell a pass-by-reference parameter from a pass-by-value parameter? What is the difference between them? What is the difference in the possible *arguments* between pass-by-value and pass-by-reference parameters?
- if you change a pass-by-value scalar (single-value) parameter, is the corresponding argument changed? why or why not?

  if you change a pass-by-reference scalar (single-value) parameter, is the corresponding argument changed? why or why not?

  - arrays are passed by value -- yet if you change the contents of a parameter array, the argument array is changed. Why?
- input parameters, output parameters, input/output parameters -- what are they? for each, what, in general, type of parameter passing is most appropriate?

also note...

- EXPECT a question giving you function headers & asking you to write calls of those functions;
- EXPECT a question asking you to write a count-controlled loop to do something a set number of times;