CS 111 - Final Exam Review Suggestions - Fall 2025

last modified: 2025-12-10

Final Exam BONUS Opportunity

- You can receive (a maximum) ***5 POINTS BONUS*** on the Final Exam if you do the following:
  - Make a **hand-written** Final Exam study sheet (a single sheet of paper, no larger than 8.5" by 11", on which you have hand-written as much as you would like on one or both sides)
  - Submit a photo or scan of it saved as a `.pdf`, `.png`, `.jpg`, or `.tiff` to Canvas **by 8:00 am on Thursday, December 18** such that I can read at least some significant CS 111 ***post-Exam-2*** material on it.
  - You are **encouraged** to have this **with you** as you are taking the Final Exam.
  - **NOTE**: if this is typed rather than handwritten, you will **not** receive bonus credit, and you will **not** be allowed to use it during the Final Exam.
  - Please let me know if you have any questions about this, and I hope it helps you in reviewing course concepts more effectively before the Final Exam.

Final Exam Set-up

- You will take the Final Exam in **NR 101** at **8:00 am** on **Thursday, December 18**.
  - You are expected to work **individually** on the exam -- it is not acceptable during the exam to discuss anything on the exam with anyone else.
  - You may have your Final Exam study sheet and ***also*** your Exam 1 and Exam 2 handwritten one-page study sheets on hand during the exam. Otherwise, the exam is closed-note, closed-book, and closed-computer/closed-electronic-devices.
- 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 Exam.
  - 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 Exam 1 and Exam 2 review suggestions handouts are still available on the public course web site, under "Homeworks and Handouts", as well as on Canvas under "Modules" in the "Exams" section.
- I expect there will be some multiple-choice questions, and the rest will be short- to medium-answer questions.
  - You will be reading and writing C++ expressions, including C++ function and named constant definitions.

  - You will be answering questions about concepts as well.
- Your studying should include careful study of posted examples and notes.
- You are responsible for material covered in class sessions, lab exercises, and homeworks.
  - This review handout is a quick overview of especially important material since Exam 2.
  - TIP: It is **perfectly fine** to retake/read over the short-answer questions in Canvas from course Homeworks as you are studying for the Final Exam!

    These are set up for unlimited retakes, and only keep the highest score, so you will not hurt your grade by doing so!
- Remember that C++ **is case sensitive** - for example, **`String`** is not the same data type as **`string`**. You are expected to use the correct case in your answers.
- You are also expected to follow **CS 111 course style guidelines and coding standards** in your answers (**including indentation**).
- You should be comfortable with the design recipe for functions, and should be able to write an opening comment block for a function including its signature, purpose statement, and tests.
  - You now know, in a function's purpose, to also **describe any side-effects** it has ("has the side-effects of...") in addition to describing what it expects ("expects ...") and what it returns ("returns...)
- 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, you will lose some credit;
  - or, if just an expression is asked for, and you put a semicolon at the expression's end;
  - 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.
- Final exams are **not returned**, although they will be kept on file for at least 2 years, and you are welcome to come by my office to look over your graded exam once it has been graded.
- A reference page will be given out with the Final Exam (***in addition*** *to* your optional handwritten page of notes); this is intended both for reference and for use directly in some exam questions.
  - This will include a copy of the posted **`111template.cpp`**.
    - I believe that the ability to use such a reference effectively is an important skill.

C++ `switch` statement

- Need to be comfortable reading and writing C++ **`switch`** statements
  - What are the differences between an **`if`** statement and a **`switch`** statement?
  - When is a **`switch`** statement appropriate?
- What are the types permitted for the **`switch`** statement's expression?
- Know how to use **`break;`** statements within a **`switch`** statement; know what they do, and what can

happen (depending on the statements involved) if you leave them out.

  - (Also know that it is a CS 111 course style standard that `break` may **ONLY** be used within `switch` statements; it is considered against course style standards to use them anywhere else.)
  - (Likewise, it is also a CS 111 course style standard that `continue` not be used.)

- You should be able to write **`switch`** statements using the course-required indentation.

Example of a side-effect: screen output (`cout`), continued

- You should be more experienced with this based on your programming since Exam 2.
- Should be able to read and write code that has side-effects such as simple screen output; should be comfortable with the object **`cout`** provided by the C++ stream input/output standard library, **`iostream`**
- How can you print the value of an expression to the screen? How can you make sure it is on its own line (is followed by a newline character)?
- How can you print a blank within your printed output?
- Be prepared to give the precise output of fragments of C++ code; you should be comfortable knowing how **`cout`** will "behave" with **`endl`**, **`boolalpha`**, literals, and other expressions.
  - **NOTE** that including **extra** newlines, blank lines, spaces, quotes, etc. will **NOT** be counted as correct for such output.

"Complete" C++ programs

- For Exam 2, you should have been able to use the posted **`111template.cpp`** (which will be included in the Final Exam Reference page as well) to write a "testing" **`main`** function that meets the class style standards for testing a non-`main` function.
  - For the Final Exam, you ***also*** should be able to write a **`main`** function that is not just for testing -- for example, a **`main`** to serve as an interactive "front end" for a non-**`main`** function or functions.
- You should be able to read a **`main`** function; you should be able to tell, from a collection of functions making up a program, what that program would do when it is run.

Local variables, mutation, and assignment statements

- What is a local variable? How do you declare a local variable in C++? How can you assign to it? (Right now, you know at least **three** ways to assign to it.)
- What is the difference between **=** and **==**?
  - If you have **`int i;`** and **`i`** has been set to some value, what does **`i = i + 1;`** do?
- Should be able to read a fragment of code and answer questions about it; should be able to say what the value of a variable is at any point within that fragment.
- For Exam 1, you should have understood that a parameter is assigned the value of its argument's expression when a function is called;

...for the Final Exam, now you should also be comfortable with using an assignment statement to change the value of a **local** variable. You should also be able to use **`cin`** to change the value of a local variable.

C++ `while` statement and `for` statement

- Need to be comfortable with the basics of the C++ **`while`** statement; 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 a fragment of C++ code including a **`while`** statement, and be able to tell what it is doing; you should be able to answer questions about what a **`while`** statement does when it executes.
  - Should be able to write a **`while`** statement.
  - Should know the course-expected indentation for **`while`** statements.
- Need to be comfortable with the basics of a C++ "classic" **`for`** statement; need to be comfortable with its syntax and semantics, need to understand how its uses mutation of a local variable to implement repetition.
  - Should be able to read a fragment of C++ code including a **`for`** statement, and be able to tell what it is doing; you should be able to answer questions about what a **`for`** statement does when it executes.
  - Should be able to write a **`for`** statement.
  - Should know the course-expected indentation for **`for`** statements.
- When is a **`for`** statement more appropriate? When is a **`while`** statement more appropriate?
- Should be able to read, write a count-controlled loop (using either a **`while`** statement or a **`for`** statement), a loop that does something a certain number of times.
  - If an exam question asks you to write a ***count***-controlled loop and does **not** specify that it must use a particular kind of statement, then you may choose to write that using either a `while` **or** a `for` statement.
- Should also be able to read a more-general **`while`** statement (that does something repeatedly, although not necessarily a known-in-advance number of times).

Using operator >> with `cin` for interactive input

- You should be comfortable using **`cin`** for interactive input with its **>>** operator.

  ```
  cin >> desired_local_variable;
  ```

- You should be able to write a **`main`** function that uses **`cin`** to serve as an "interactive front end" for a non-`main` function.
  - (But you should also be aware that *any* function can happen to use either of these to have a side-effect of interactive input, in addition to what that function expects and returns.)

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 within an array?
  - Given an array's declaration, you should be able to write expressions representing individual elements within that array.
- How can you do something to every element within an array? How can you use every element within an array?
  - You should be able to write a a count-controlled 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 can you write a function with an array parameter?
  - In C++, what is usually also included as one of the parameters when a function has an array parameter?
  - How do you indicate an array parameter in a function signature comment?
  - How do you declare an array parameter in a function header?
  - How do you call a function with an array argument?
- EXPECT IT: expect to write at least one loop that does something involving every element in an array.

Different kinds of C++ functions

- 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, etc.!)
- You should know the difference between a function **returning** something and a function **printing** something to the screen or to a file; you should be able to write functions that can do either or both, depending on what is specified.
- Given a function header, you should know how to then write a "legal" call to that function;
  - 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: given a function header, write a "legal" call to that function.
- You should know what happens when:
  - ...you call a function (especially one that has side-effects) by itself as a statement:

```
cheer(13);
```

- ...you call a function that returns something within a **`cout`** statement:

```
cout << cheer(13) << endl;
```

- ...you call a function that returns something on the right-hand-side of an assignment statement:

```
int looky;
looky = cheer(13);
```

Preprocessor directives

- what does **`#include`** do? Where should you put it? When is it done/"handled"?
- how do you **`#include`** a standard library (what needs to surround its name)? For CS 111, what line should follow all of your **`#include`**s? (**`using namespace std;`**)

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? What **`#include`**, then, do you need to include in each **`.cpp`** file containing a function body that does file input or file output?
- 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?
- 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?
  - 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.

C++ "shorthand" operators

*+=, -=, *=, /=*

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

++ and --

- What are the prefix and postfix increment operators (++) and the prefix and postfix decrement operators (--)? How are they used? What are their effects and their semantics?
- Be able to accurately read code fragments containing these operators.