# CS 111 - Exam 2 Review Suggestions - Fall 2016

last modified: 2016-11-14

- 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 responsible for the material covered through the Week 11 Lab (2016-11-04), and through and including Homework 9.
- 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!)
- 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.
  - The questions will all involve C++, although many of the concepts overlap with those from the first exam (since we covered many of the same concepts in Racket and in C++).
  - Indeed, to emphasize the common concepts, note that *some* Exam 2 questions *may* be simply C++ versions of questions from Exam 1.
- Note that you will be given copies of the following posted templates along with Exam 2:
  - the course `main` function template
  - the course non-`main` function template
  - the course `.h` file template

  I believe that the ability to use such references effectively is an important skill.
- Your studying should include careful study of posted examples and notes as well as the homeworks (and posted example solutions) thus far.

## C++ basics

- What is a simple expression in C++? What is a compound expression in C++? Should be able to read these, write these, tell the type of a given expression, write an expression of a given type.
- (You are expected to be familiar with the C++ types discussed so far, including knowing their C++ type names.)
- You should know the difference between an expression and a statement; you should know how a statement is terminated in C++.
  - Note that, often/usually, C++ statements end in semicolons (unless you are talking about a block, `{ }`). Expressions, which have a value, are usually within C++ statements, and so don't need semicolons.

- Need to be able to write a C++ function using the design recipe! (Need to be able to write the steps that `funct_play2` asks you for, in the right order!)
  - How does a C++ signature differ from a Racket signature?
  - Need to be able to write appropriate specific examples/tests for C++ functions.
  - Need to be able to write a C++ function header, and a C++ function body. Need to be comfortable reading, designing, writing, testing, and calling/using C++ functions.
- Need to follow the course style standards.
- Need to be comfortable with C++ identifiers and C++ literals.
- How do you write a named constant declaration in C++?
- What types have we discussed so far in C++? How can you write literals of (most of) these? How would you declare variables of each?
- You should be comfortable with the C++ types `string` and `char*`.
  - You are expected to be comfortable with C++ string literals (anything written within double quotes); although these are really of type `char*`, note that they can be assigned to variables of type `string`.
  - You should be able to declare new-style C++ string variables (`string`, and `#include <string>`)
  - (And note that it is a course style standard that, whenever possible, when you want a function to have a string parameter or return a string value, you are expected to use type `string` rather than `char*`.)
  - You should be comfortable reading and writing C++ statements and expressions using the `string` methods `length` and `at`, and concatenating `string` instances using `+`; given a description of a "new" `string` method, you would be able to write C++ statements and expressions using that method.
- What are the basic arithmetic operators of C++? What do we mean by operator precedence? How do you write the relational operators in C++? ...the boolean operators? What happens when you divide two integers?

## C++ `if` statement and `switch` statement

- Need to be comfortable reading and writing C++ `if` statements and `switch` statements
  - What are the differences between these statements?
  - When is a `switch` statement appropriate?
- What are the types permitted for the `switch` statement's expression?
  - Why do you need to know about how to use `break;` statements within a `switch` statement?
- You should be able to write these using the course-required indentation.
- (And you still need to be able to write an appropriate set of examples for a function involving multiple categories of data -- need an example for each category, and for the boundaries between those categories!)

## example of a side-effect: screen output (`cout`)

- 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?
- Be prepared to give the precise output of fragments of C++ code; you should be comfortable knowing how `cout` will "behave" with `endl`'s, `boolalpha`, literals, and other expressions.
  - I could give you a "grid" of squares, and ask you to write out precisely what would be displayed, 1 character per square, to see if you know.

## "complete" C++ programs

- what is a C++ program? what function must be included in a C++ program? There are several acceptable headers for this function; what is the one that we have been using?
  - Given the CS 111 course style standards, what is this function expected to return?
- Note that the examples handout will include the course `main` template from the public course web page; you should be able to write a `main` function, given that template.
- 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.
  - You should be able to write a "testing" `main` function that meets the class style standards for testing a non-`main` function.
  - You should also be able to write `main` functions for purposes other than testing as well.
- Given all of the files involved in a C++ program, what `g++` command would you type to compile, link, and load to create a C++ executable program for that program?
  - as our class convention, what is the resulting C++ executable program file named?

## precompiler directives

- what does `#include`, do? Where should you put it? When is it done/"handled"?
- where would you typically find `#define`, `#ifndef`/`#endif`?
- 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 class, what line should follow all of your `#include`s?
- Note that the examples handout will include the `.h` file template; you should be able to write a header file for a non-`main` function, given that template; 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 **two** 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;

  ...from Exam 2, now you should also be comfortable with using an assignment statement to change the value of a **local** variable. You should also be comfortable using `cin` to change the value of a local variable.

## Interactive input (`cin`)

- You should be quite comfortable using `cin` for interactive input.
- You should be able to write a `main` function that uses `cin` to serve as an "interactive front end" for a non-`main` function.

## different kinds of C++ functions (so far)

- 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, for example).
- you should know the difference between a function returning something and function printing something to the screen; you should be able to write functions that can do either, 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?
- 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 within a `cout` statement:

    ```
    cout << cheer(13) << endl;
    ```
  - ...you call a function on the right-hand-side of an assignment statement:

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