CS 131 - Exam 2 Study Suggestions

last modified: 11-11-10

* Remember: anything covered in lecture, in lab, or on a homework, is FAIR GAME.

* You are responsible for all of the material covered through Week 12 - Lecture 1 (11-9-10) (including Homework 9)

* These are simply suggestions of some especially important concepts, to help you in your studying.

* 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.

* You will be writing and reading expressions, functions, and programs on this exam; you will be answering questions about concepts, expressions, functions, and programs 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++)
  * note that some Exam 2 questions may be simply C++ versions of questions from Exam 1

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 ~st10/funct_play2 asks you for!)
  * how does a C++ contract differ from a Racket contract?

  * 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 be able to write appropriate specific examples/tests for C++ functions

* need to follow the course style standards (for Racket and C++)

* 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 these? how would you declare variables of each?
  * 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>`)

* 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

* need to be comfortable reading and writing C++ `if` statements

* (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!)

* you should be able to write these using the course-required indentation;

C++ classes

* need to be comfortable reading, writing, and using C++ classes (note that there will be an example C++ class `.h` file and `.cpp` file included in the packet of example code given out with the exam).

* need to be able to write a data definition comment for a C++ class; need to be able to write a template for a C++ function expecting a class instance as a parameter

* what is a class data field? what is a class constructor/constructor method? (what is its name?) what is a class selector/selector method? what is a class modifier/modifier method? What is our class style for naming these?

* need to be able to write a class definition with a class `.h` file; should be able to implement a class's methods in a class `.cpp` file

* need to be able to write a function that tests a class;

* how do you declare an object (an instance of a class)? How do you call a method for a particular object/class instance?
  * how do you declare an object using a zero-argument constructor?

* how do you write a function expecting a class instance?

* need to know what is meant by overloading a method within a class, and how to do it;
  * need to 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, need to know how to call that 0-argument constructor...)

* should be 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.

mutation - assignment statements, +=, ++, and more

* how do you declare a local variable in C++? How do you assign to it?

* what is the difference between = and ==?

  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

+=, -=, *=, /=

* What do +=, -= *=, /= mean? How are they used? What are their effects and semantics?

* Be able to accurately read, write code fragments containing them, too;

prefix increment operator, postfix increment operator

* What are the prefix and postfix increment operators (++)? How are they written? Where are they written? What are their effects and their semantics?

* Be able to accurately read, write code fragments containing them;

* (You should be able to handle the prefix and postfix decrement operators (--) also.)

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? what are its indices? How do you access an individual element?

* 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?

* 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?

C++ loops (iteration/mutation-based repetition)

* need to be comfortable with the basics of the C++ `while` loop and `for` loop statements; need to be comfortable with their syntax and semantics, need to understand how they use mutation of a local variable to implement repetition;

* should be able to read, write a count-controlled loop (using both a `while`-loop and a `for`-loop), a loop that does something a certain number of times;

* when is a `for` loop appropriate?

* 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

example of a side-effect: screen output

* should be able to 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 do 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 course style standards, what is this function expected to return?

* Note that the examples handout will include the **`main`** template **`main_template.txt`** from the public course web page; you should be able to write a **`main`** function, given that; 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

* Given all of the files involved in a C++ program, how can you link and load and create a C++ executable program for that program?

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, they may not return anything, they may require no parameters, etc.!)

* 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;
    * If a function is a `void` function, how is it called?
    * If a function expects no parameters, how is it called?
    * If a function returns a value, how is it (typically) called?
    * If a function expects one or more parameters, how is it called?

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?

precompiler directives

* what do `#include`, `#define`, `#ifndef`/`#endif` do? Where should you put them? When are they done/handled?

* how can `#ifndef`, `#define`, and `#endif` help in reducing redefinition errors involving #includes?
    * doing this, what do we consider to be good practice/good style in terms of what we should name the `#define` involved?

* 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 #includes?