CIS 130 Final Examination Review Suggestions

* last modified: 12-07-05

* you may bring into the final exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish. 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.)

* final is CUMULATIVE!
  * if it was fair game for exams #1 or #2, it is fair game for the final;

  * Thus, using the posted review suggestions for exams #1 and #2 in your studying for the final would be a good idea. (Note that they are still available from the public course web page, under "Homeworks and Handouts".)

  * studying your Exam #1 and Exam #2 would also be wise.

  * there may indeed be similar styles of questions on the final as on those exams.

* IN GENERAL...
  * Anything that has been covered in **assigned reading** is fair game;
  * Anything that has been covered in **lecture** is fair game;
  * Anything covered in a **course handout** is fair game;
  * Anything covered in a **lab exercise** or **lecture exercise** or **homework assignment** is ESPECIALLY fair game.

  * But, the Exam #1 Review Suggestions, Exam #2 Review Suggestions, and these review suggestions provide a quick overview of especially important material.

* 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 fill in the opening comment block "templates" we've been using appropriately.

* note that answers may lose points if they show a lack of precision in terminology; 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.

*** high points from the material SINCE Exam #2:**

* Given a .h file template (such as that on the public course web page) or an example of a .h file, you should be able to write a .h file for another auxiliary function.

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

  you have also written 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, they may change the values of pass-by-reference 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.

  * you should know the difference between a function **accepting** something as a parameter and a function **prompting** a user for input (and then **reading** in such input); you should be able to write functions that can do either, depending on what's specified.

  * you should know the difference between an **expression** and a **statement**; you should know how a **statement** is terminated in C++.

  * 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 accepts 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?

* What statement is used to perform interactive input into a C++ program? What statement is used to perform interactive or screen output from a C++ program?

* Be prepared to give the precise output of fragments of C++ code; you should be comfortable knowing how cout will "behave" with endl's, literals, and other expressions.

* You may also be expected to modify existing code, add to existing code, and/or correct erroneous code.

* know what an **algorithm** is; you know what **pseudocode** is.

* you are expected to be comfortable with C++ string literals (anything written within double quotes);

    * although you now know that you CAN declare a string variable by using either the type **string** (for the standard **string** class, along with **#include <string>**) or **char [size]** (for the old-style C string), I will avoid requiring you to use this on the final examination.

    * note that C++ string literals are old-style C strings, but can be assigned to **string** variables (C++ will convert them as part of the assignment). However, passing them as parameters is another matter...! (sigh!)

* the **char** type

    * how do you write a **char** literal?

    * how do you declare a variable of type char?

    * you should be able to write a function that takes a **char** parameter; you should be able to write a function that returns a value of type **char**. You should be able to declare a variable of type **char**, and should be able to set a **char** variable's value appropriately.

* **switch statement**

    * another C++ statement that implements a kind of **branch** structure!

    * what is the syntax of the C++ switch statement? What are its semantics?

    * should be very comfortable with the course-expected indentation for switch statements.

    * what is the purpose of **break** within a switch statement? What happens when it is omitted?

    * what are legal types for the control expression and the case expressions within a switch statement?

    * be comfortable designing, reading, and writing switch statements; you should be able to read a switch statement, and tell what it is doing; you should be able to give its output.

* **for statement**

    * another C++ statement that implements a kind of **repetition** structure!

    * what is the syntax of the C++ for statement/for loop? What are its semantics?

    * should be very comfortable with the course-expected indentation for **for** statements/**for** loops;

    * when are **for** loops appropriate? Should be able to decide when a **while** loop is more appropriate, and when a **while** loop is more appropriate;

        * you should be able to convert from a count-controlled while loop to the equivalent for-loop, and vice versa;

    * be comfortable designing, reading, and writing for loops; you should be able to read a for loop, and tell what it is doing; you should be able to give its output.

* **arrays**

    * what is an array? (we'll be sticking with one-dimensional arrays...)

    * 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.

    * what 3 pieces of information does the compiler need to know to declare an array? how do you declare an array in C++?

    * how can you initialize an array? (there is more than one way...)

    * what is an array index? what is the size of an array? if you know an array's size, what are the indices of that array? what is the type of an array?

    * how do you write an expression representing a single element in an array? how do you write an expression representing the entire array?

    * how do you pass an array as an argument? how do you declare an array as a parameter?

    * 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?

* **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.)

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

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

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

* **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?