CSSE 120 Final Exam – What to expect

Spring, 2012-2013

Time: You have 4 hours to complete the exam – no time extensions without prior approval.

External resources allowed (same as Test 2, except includes a "cheat sheet" for the Paper-and-Pencil part):

- There is a ***Paper-And-Pencil*** part. For this part, the only external resource you may use is a single 8½ by 11 sheet of paper, with whatever you want on it. You may use BOTH sides of the sheet.
- There is an ***On-The-Computer part***. For this part, the only external resources you may use are:
  - Your computer and anything on it.
  - Your SVN repositories (your own and any whose ownership you share with teammates or partners).
  - Any directly reachable from the CSSE 120 web site.

  You may NOT use any search engine – no googling.

For both parts of the exam, you must not communicate with anyone except your instructor and his assistants at the exam, if any.

Languages: Both Python and C are featured in compare-and-contrast problems on the paper-and-pencil part of the exam. See details below. The on-the-computer part is entirely in C.

For the **Paper-And-Pencil portion** of the final exam, you should be able to:

1. Trace C code, especially code that includes pointers, and:
   a. Draw the ***box-and-pointer diagram*** appropriate to the code's execution, and
   b. Show what gets printed by *printf* statements during the code's execution.
2. ***Compare and contrast C and Python***. In particular, you should be able to explain how the following are similar and how they are different, giving examples to support your explanations:
   a. Arrays (in C) and lists (in Python)
   b. (Static) types in C and (dynamic) types in Python
   c. Explicit pointers in C and implicit references in Python
   d. Execution of a program in C (using code produced by a compiler) and execution of a program in Python (using an interpreter)
   e. C-style strings (in C) and strings (in Python)
   f. Structures (in C) and classes (in Python)
   g. Characters (in C) and characters (in Python)
   h. Blocks in C and blocks in Python
   i. Doing input/output in C (`scanf` /`printf`) and in Python (`input`/`print`).
3. ***List several features*** that an application might require that would make you ***choose C*** (as opposed to Python) as the language in which to implement the application.

4. ***List several features*** that an application might require that would make you ***choose Python*** (as opposed to C) as the language in which to implement the application.
5. Explain what happens in C ***when a program crashes***, and what happens in Python when a program crashes. Which language provides more help when a crash occurs? Explain.
6. Explain what happens in C if a program accesses an array beyond its bounds, and what happens in Python if a program accesses a list beyond its bounds.
7. In C, which of the following can happen when you reference an array out of bounds? (Choose all that apply.)
    a. The program might print a meaningful error message at run time.
    b. The program might crash at that point.
    c. The program might crash at a subsequent point (as a direct result of this error, not another error).
    d. A different variable in the program might have its value changed (by the array reference out of bounds).
    e. The program might run but give incorrect behavior (as a direct result of this error, not another error).
    f. The program might run to completion without any wrong behavior.
8. Repeat the previous question (with the same options), but with the mistake being an incorrect use of a pointer variable.
9. Why are pointers valuable in C? Give two *different* reasons.
10. Explain the difference between *static* memory allocation (from the ***stack***) and *dynamic* memory allocation (from the ***heap***, using ***`malloc`*** or ***`realloc`***). Also explain how each of these work, and the notation for each.
11. What is the ***`free`*** operation in C accomplish? Why is it important? What happens if you fail to use ***`free`***?
12. Explain the relationship between arrays and pointers.
13. Explain (statement by statement) what each of the following statements mean in C:

```
float a, b;
float* x;
float* y;
x = &a;
*x = 45;
y = x;
```

14. Consider the following statement, where ***x*** and ***y*** are lists (in Python) or arrays (in C) (and where the C statement would have a semi-colon at the end of it):

```
x = y
```

    This works fine in Python but does not compile in C. Explain what it means in Python and why it does not compile in C.

15. Which are generally easier to debug: programs in Python or programs in C? Explain your answer with some concrete examples.

For the **On-The-Computer, in C, portion** of the final exam, you should be able to do the following ***in C***:

1. Implement ***input and output*** (using ***`scanf`*** and ***`printf`*** on the types ***`int`***, ***`float`***, ***`double`***, and ***`char`*** and on C-style strings).
2. Use ***function definitions***, ***function calls***, ***assignment***, ***conditionals*** and ***loops*** at a level of maturity similar to what we expected in Python.
3. Write ***function prototypes***.
4. Implement functions that use ***nested loops*** to display 2-dimensional patterns on the console (like those that we did many exercises on, in Python and in C)
5. Declare ***arrays*** and use them successfully in function calls and as parameters of functions. In particular, you should be able to apply the following ***array patterns*** (as well as variations and combinations of them):
    a. Accessing the k^th^ element of the array, for a given k.
    b. Printing (or otherwise processing) all the values of an array, from beginning to end.
    c. Printing (or otherwise processing) all the values of an array, backwards, or using only a portion of the array.
    d. Counting the number of elements in the array that meet a given condition.
    e. Summing the elements in an array of numbers, perhaps just those that meet a given condition.
    f. Finding the largest (or smallest) number in an array of numbers.
    g. Finding a given element in an array, or an element that meets a given condition (and returning its position in the array).
    h. Initializing the elements of an array to:
        i. Random or pre-specified values.
        ii. Values that are a function of the index.
        iii. Values input by the user, asking the user how big the array should be.
        iv. Values input by the user with a sentinel loop.
    i. Patterns that refer to another element while processing the current element, e.g. determining whether an array is:
        i. Sorted from smallest to largest.
        ii. A palindrome.
    j. Loop-within-a-loop array patterns, e.g. printing (or otherwise processing) all the values of a two-dimensional array.
6. Declare instances of a given ***structure*** and use those instances successfully, including accessing their fields.
7. Use arrays of structures, or structures that contain arrays.
8. Declare and use ***pointer variables***, in particular to:
    a. "Send back" multiple values from a function (by using parameters that are pointers to variables in the caller).
    b. Send an array to a function and use the array as a parameter.
9. Declare and use ***C-style strings***, explicitly and via string functions in the **`string.h`** library.

This term we did not do two-dimensional arrays, so you are NOT responsible for them.