ESC180F: Final Exam: 2018-12-20

First name (please write as legibly as possible within the boxes)

Last name

Student ID number

Instructions

- It is recommended that you use PEN on the exam. If you decide to use pencil: print hard.
- Write your official first name, last name, student number and utorid, using ONLY bold capital (UPPERCASE) letters and numbers.
- Closed book; no aids; no electronic equipment allowed (cellphones, tablets, computers, calculators, etc.).
- Attempt and answer ALL questions
- Answer all questions on the space provided for the question you are answering. Answers written in the wrong spot will not be accepted.
- NO QUESTIONS WILL BE ANSWERED BY THE EXAMINER. If a particular question seems unclear, explicitly state any reasonable assumptions to enable you to proceed with the problem.
- Show all steps and present full solutions, clearly, using the concepts from the course. Non-course material will not be accepted as valid.
- For all questions, unless otherwise stated, all return values shall be of type int (C) or Boolean (Python).
- The questions start with a code (C, n) or (Python, n): the first part indicates the language to be used, and the second (n) indicates the points that a correct solution will earn.

Possibly useful information about Python

```
Types: sequential (list, string), numeric (integer, float),
       boolean (True, False)
Operators: tests (==, !=, <=, >=, >, <), and, or, not,
           +, -, *, /, %, assignment (=), list copy (L=list(M))
Extending the language: functions (def ... return), classes
```

Possibly useful information about C

```
Types: numeric (int, float), char, struct,
       arrays (TYPE NAME[SIZE]), pointer/address (TYPE *NAME),
Operators: tests (==, !=, <=, >=, >, <), &&, ||, !, +, -, *, /, %,
           assignment (=)
Extending the language: functions structs

Type Analysis:
  TYPE *X; means: *X is a TYPE
                   X is a TYPE *
                  &X is a TYPE **

Passing pointers:
 if the function is defined: RVAL_TYPE FNAME(TYPE1 *PARG1_NAME,...);
 if the caller defines:
    TYPE1 X;
    RVAL_TYPE R;
 then the caller must call FNAME via:
    R = FNAME( &X, ...)

 Since PARG1_NAME is a TYPE1 *, then inside of FNAME you can write:
     *PARG1_NAME = <some value>;
 and the value of X in the caller (which is outside of FNAME) will be set
 to that value.

 If you want to return a value of type TYPE via the arg list,
 then you must define the argument for that value as a TYPE *

Malloc syntax: if you define TYPE *X;
 then you can allocate memory for X via:
   X = (TYPE *) malloc(sizeof(TYPE) * <SIZE>);

 Then X[0] refers to a value of type TYPE at the start of that memory.
      X[<SIZE>-1] refers to the final value of type TYPE in that memory
```

1. (C; 5) A function, f, takes in an array of integers as its sole argument. No additional arguments are provided for this function. You know that the valid data items for this array are non-zero, hence the designer has decided to employ the same tactic with this array, as C employs for strings in char arrays. Write this function f, such that it returns the number of valid data items in this array.

2. (C; 10) Write a function, g, that takes in only two argument arrays (of the form described in the prior question) and returns an array that is the concatenation of the two argument arrays (in their order of appearance), with the same end-of-array tactic as in the prior question.

3. (C; 15) Consider a function, h, that takes in two argument arrays of general, float values (no value is invalid) and outputs an array that is the concatenation of two argument arrays (in their order of appearance). The output will be via an additional **final** argument, which is not caller-allocated. (4 points) To be well-posed, how many arguments are required for h? (2 points) Show the prototype of h. (9 points) Write the function.

4. (C; 20) Consider a function, i, that takes in one argument, an array (of the form described in the prior question), and computes the histogram of that array, which will be returned. The histogram will be represented via pairs of the form (n,frequency); frequency should report the number of times the value n was found in the argument array. You are to represent each element of the histogram using a struct; the set of all such pairs will be contained in an array. (2 points) Show the definition of the struct to represent a histogram element. (2 points) What must you return in addition to the histogram? (2 points) How can you, via the return statement, return the histogram plus this additional piece of information? (14 points) Write the function.

(continued; your solution to the prior problem may be written here)

5. (C; 10) Consider the function, j, that is like i (above), except that it outputs the histogram and the additional piece of information required by the histogram return value, via additional arguments (that are not caller-allocated). (4 points) Show the function prototype for j, and (6 points) show the line of code that allocates the additional arguments.

6. (Python; 15) Write the function, hist(x), that produces the histogram of x, which is a list of integers. The output should be a list of lists, where the lower-level lists are pairs as defined previously (0th element is n, 1st element is the frequency of n in x).

7. (Python; 20) You are to write a Python facility that will generate the points of a line when called repeatedly. The facility will maintain the internal state of the domain, and compute the corresponding range when requested. It works as follows:

```
x=gen_line()
x.set_slope(10)
x.set_y_intercept(9)
x.set_domain(5)
x.set_domain_increment(0.01)
x.get_domain() #returns 5
x.get() #returns the value of 10*5+9
x.get_domain() #returns 5+0.01
x.get() #returns the value of 10*(5+0.01)+9
x.get_domain() #returns 5+0.01+0.01
x.get_domain() #returns 5+0.01+0.01
```

(continued; your solution to the prior problem may be written here)

8. (C; 5) The function f calls another function g. Within f, there is an int variable, x. Function f calls g, expecting g to modify x by setting it to x*x+x. (2 points) Show the line of code in f that calls g with a single argument. (2 points) Show the function prototype for g. (1 point) Show the line of code in g that performs the aforementioned modification.

9. (C; 10) The function f calls another function g. Within f, there is an int* variable, x. Function f calls g, expecting g to modify x by setting it to point to an int array of size 100. (4 points) Show the line of code in f that calls g with a single argument. (2 points) Show the function prototype for g. (4 point) Show the line of code in g that performs the aforementioned modification.

10. isBlah test:

    - (Python; 5) A number is a blah number if it can be expressed as the product of either (a) 7 and another prime number, or (b) 4 and a prime number. Write the function isBlah(x), that returns True if x is a blah number, and False otherwise.

- (C; 5) Write the same function in C, where 1 is returned instead of True, and 0 is returned instead of False.

11. isBlah generator:

    - (Python; 10) Using the isBlah function you defined earlier, write the function genBlah(x), that returns a list of the first x blah numbers.

- (C; 10) Using the isBlah function you defined earlier, write the function genBlah(x), that returns an array of the first x blah numbers.

- (Python; 10) Using the isBlah function you defined earlier, write the function genBlahLim(x), that returns a list of all blah numbers up to x (not including x)

- (C; 10) Using the isBlah function you defined earlier, write the function genBlahLim(x), that returns an array of all blah numbers up to x (not including x)

12. (Python; 10) Write a function, compress(x), that takes a list of numbers and returns a list of the pairwise sums of adjacent numbers as in:

```
compress([]) # returns []
compress([1]) # returns []
compress([1,1]) # returns [2]
compress([1,2,1]) # returns [3,3]
```

13. (Python; 10) Write a recursive function, pascal(n), that generates the nth line of pascal's triangle; pascal(0) will return [1], and so on. You may assume the use of the prior-defined compress function (as specified by the question statement).

14. (Python; 20) Write the function, sort(x), where x will always be a list of numbers, that returns a list of the elements of x, sorted in ascending order.

15. (Python; 20) Write the function slide(x), where x will always be a list of numbers. that, for each element of x, computes the average of that element, its preceding one, and its succeeding one. The resulting list, formed by concatenating these averages, will be returned; note, it must have the same length as x. If there is no preceding elemennt, then zero is chosen; if there is no succeeding element, then zero is chosen.