Lab #1 - C Programming Basics

CMPSC311 - Introduction to Systems Programming
Spring 2024 - Prof. Sencun Zhu

Due date: February 9, 2024 (11:59pm) EST

In this lab assignment, you will write simple functions in C. The purpose of this lab assignment is to assess your basic programming skills. You should have no difficulty in completing this assignment. If you experience difficulty, you should take some time to brush up on programming.

Like all lab assignments in this class, you are prohibited from copying any content from the Internet including (discord or other group messaging apps) or discussing, sharing ideas, code, configuration, text, or anything else or getting help from anyone in or outside of the class. Consulting online sources is acceptable, but under no circumstances should anything be copied. Failure to abide by this requirement will result in penalty as described in our course syllabus.

Below are the files in this assignment and their descriptions:

1. student.h: a header file with the declarations of the functions you will implement.
2. student.c: a source file in which ***you will implement*** the functions whose declarations appear in student.h. In other words, you will provide the definitions of the functions declared in student.h. This is the only file that you will change. We will only use your student.c for grading.
3. ref.h: a header file for ref.o
4. ref.o: an object file that is needed by tester to test your code.
5. tester.c: a source file that contains unit tests for the functions that you will implement. This file will compile into an executable, tester, which you will run to see if you pass the unit tests.
6. Makefile: instructions for compiling and building tester used by the make utility.

Your workflow will consist of:

(1) If you are using an arm based machine like newer MacBooks running on M1/M2/M3 chips delete ref.o and then rename ref_arm64.o to ref.o .
(2) Implementing functions by modifying student.c
(3) Typing 'make clean' and then 'make' to build the tester.
(4) Running tester to see if you pass the unit tests
(5) Committing and pushing your code to GitHub
(6) Repeating steps 2-5 until you pass all the tests.
(7) Final commit and push to GitHub and submission of commit id on canvas

Although you only need to edit student.c for successfully completing the assignment, you can modify any file you want if it helps you in some way. When testing your submission, *however*, ***we will use the original forms of all files except student.c***. Do not forget to *add comments* to your code to explain your implementation choices.

<u>Below are the functions you need to implement:</u>

1. `smallest`: Takes an array of integers and the length of the array as input and returns the smallest integer in the array. You can assume that the input array contains at least one element.
2. `product`: Takes an array of integers and the length of the array as input and returns the product of the integers in the array.
3. `swap`: Takes pointers to two integers and swaps the values of integers.
4. `rotate`: Takes a pointer to three integers and rotates the values of integers. For example, if the inputs are pointers to integers `a`, `b`, and `c`, then after a call to `rotate`, `c` contains the value of `b`, `b` contains the value of `a`, and `a` contains the value of `c`.
5. `sort`: Takes a pointer to an array of integers and the length of the array as input and sorts the array in descending order (larger to smaller). That is, after a call to `sort` the contents of the array should be ordered in descending order. You can implement any sorting algorithm you want, but you have to implement the algorithm yourself—you cannot use sort functions from the standard C library. We recommend using something simple, such as Bubble sort or Selection sort.
6. `halve_primes`: Takes an array of integers and the length of the array as input and halves every prime element of the array. For example, if the input array contains [1, 7, -2], then after a call to `halve_primes`, the array will contain [1, 3, -2].
7. `square_armstrongs`: Takes an array of integers and the length of the array as input and *square every positive element of the array that is an Armstrong number.* An Armstrong number (also called a Narcissistic number) is an integer that is equal to the sum of its decimal digits, each raised to the power of the number of decimal digits. Read more about them here For example, 153 is an Armstrong number because it has 3 digits and $1^3 + 5^3 + 3^3 = 153$. Thus, if an input array contains [2, -153, 153], then after a call to `square_armstrongs`, the array will contain [4, -153, 23409].
8. `triple_happy`: A happy number is an integer that, when replaced by the sum of the square of each decimal digit, will eventually be 1. Reade more about them here. For example, 19 is a happy number because $1^2+9^2=82$, $8^2+2^2=68$, $6^2+8^2=100$, $1^2+0^2+0^2=1$. For this function you have to triple any happy number in the array. Thus if an input array contains [4,19,2], then after a call to `triple_happy`, the array will contain [4,57,2]

You are encouraged to write helper functions to simplify the implementations of the above functions. It is important to note that you should not use any library functions and implement all helper functions independently. Additionally, any extra #include directives are not allowed. For instance, if a function to raise number 'a' to the power 'b' is needed, you are expected to <u>implement this function from scratch</u>.

<u>Deliverables:</u> Push your code to GitHub. Submit the commit ID of your latest code to Canvas for grading. Ensure that there is no additional text, words, or comments around the commit ID. Check canvas assignment page for more details.

<u>Grading rubric</u>: The grading would be done according to the following rubric:

- ☐ Passing all test cases: 95%
- ☐ Adding meaningful descriptive comments: 5%
- ☐ If your 'make' has any errors or if your code is stuck in a loop, you will get a 0 for this lab.
- ☐ If you do not submit your commit id on canvas by the due date, you will get a 0 for this lab.