Lab #1 - C Programming Basics

CMPSC311 - Introduction to Systems Programming
Spring 2025 - Prof. Suman Saha

Due: February 14th, 2024 (11:59 PM EST)

Like all lab assignments in this class, you are prohibited from copying any content from the Internet or discussing, sharing ideas, code, configuration, text, or anything else or getting help from anyone (AI or human) in or outside of the class. Failure to abide by this requirement will result in a penalty as described in our course syllabus.

Overview

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

Files Provided

Below are the files included in this assignment and their descriptions:

- **student.h**: A header file with the declarations of the functions you will implement.
- **student.c**: A source file where **you will implement** the functions declared in `student.h`.
- **ref.o**: An object file that is needed by tester to test your code.
- **ref.h**: A header file for `ref.o`
- **tester.c**: A source file containing unit tests for the functions you will implement. This file will compile into an executable, tester, which you will run to see if you pass the unit tests.
- **Makefile**: Instructions for compiling and building `tester` using the `make` utility.

Workflow

Your workflow will consist of the following steps:

1. Rename the appropriate reference object files based on your machine architecture:
    - For ARM-based machines (e.g., MacBooks with M1/M2/M3 chips), rename `ref_arm64.o` to and `ref.o`.
    - For Intel-based x86 machines, rename the given ref_x86.o to ref.o

2. Implement the required functions in `student.c`.
3. Run `make clean` followed by `make` to build the tester.
4. Execute `tester` to check if you pass the unit tests.
5. **Commit and push your code to GitHub.**
6. Repeat steps 2–5 until all tests are passed.
7. Make a final commit and push it to GitHub, and submit the commit ID on Canvas.

Note: While you only need to edit `student.c`, you may modify other files for debugging or testing. However, when grading your submission, the original forms of all files except `student.c` and `student.h` will be used. Do not forget to add comments to your code to explain your implementation choices.

Functions to Implement

You are required to implement the following functions:

1. **smallest_pos**: Takes an array of integers and its length as input and returns the smallest positive integer. If not possible, return 0.
2. **sum_modulo**: Takes an array of integers, its length, and an integer k as inputs. The function should calculate the modulo k for each element in the array, sum these individual results, and return the total. If an error occurs the function should return 1."
   Example input: product_modulo([10,22,6],3,5)
   Expected output: 3
   Explanation: 10 mod 5 + 22 mod 5 + 6 mod 5 = 3 *Hint:* What is the modulo of a negative number?
3. **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.
4. **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.
5. **find_building** Takes an apartment number (int) and returns: 0 if the apartment belongs to Building 0 (0-99), 1 if it belongs to Building 1 (100-199), 2 if it belongs to Building 2 (200-299), and -1 for invalid apartment numbers (outside 0-19). Example input: findBuilding(15) Expected output: 0 Explanation: apartment 15 is in building 0
6. **cube_armstrongs**: Cube every positive Armstrong number in an array. 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. 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 cube_armstrongs, the array will contain [8, - 153, 3581577].

7. **double_spy**: Double every spy number in an array. Spy numbers are positive integers where the sum of their digits equals the product of their digits. For example, 123 is a spy number since $1+2+3 = 1*2*3$. Thus given an array[11,123,-44] then after a call to `double_spy` the array should will be [11,246,-44].
   You may refer to: https://thebrainboxtutorials.com/2022/07/what-is-a-spy-number.html

8. **second_bit**: Takes a non-negative integer as input and returns the second bit from the right in its binary form.
   Example input: second_bit(5)
   Expected output: 0
   Explanation: 5 in binary is 101, and 0 is the second bit from the right-hand side.

Note: 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 implement this function from scratch.

Grading Rubric

The grading will be based on the following criteria:

- Passing all test cases: **95%**
- Adding meaningful descriptive comments: **5%**

Important Notes:

- If `make` generates errors or if your code enters an infinite loop, you will receive a **0** for this lab.
- Failure to submit your commit ID on Canvas by the due date will result in a **0**.

Penalties: 10% per day for late submission (up to 3 days). The lab assignment will not be graded if it is more than 3 days late.

Asking for help:

If you are stuck and need assistance, here's the recommended approach:

1. Check Campuswire
   - Start by searching Campuswire for similar issues. Often, other students may have encountered and resolved the same problem.
   - If you don't find an answer, you can create a new post describing your issue.
     Important: Do not post your code directly on Campuswire. If your question is code-specific and requires a TA to review your work, you may share your commit ID. The commit ID is unique and only visible to you and the TAs, ensuring the integrity of your work

.

2. Email or Visit Office Hours
    - If Campuswire doesn't resolve your issue, email your assigned TA or visit their office hours.
    - Do not email multiple TAs separately or use BCC. Instead, email one TA, and if you do not hear back from them in a timely manner, you can send an email to other TAs in the same email chain and clearly indicate your issue. This ensures transparency and avoids duplication of effort among TAs.

Following these guidelines will help ensure timely and effective assistance while maintaining fairness and efficiency for all.