Assignment #1 - C Programming Basics

CMPSC311 - Introduction to Systems Programming

Fall 2022 – Dr. Suman Saha

Due date: September 23, 2022 (11:59pm) EST

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

Like all 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 in or outside of the class. Failure to abide by this requirement will result dismissal from the class 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 that 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`
3. `reference.h`: a header file with the declarations of the functions that are identical to those defined in `student.h` except they are prefixed with `reference`. These are the reference functions against which your implementations will be tested.
4. `reference.o`: an object file that contains the reference implementations of the functions declared in `reference.h`. This is a binary file that contains compiled reference implementations of functions.
5. `reference-m1.o (for MAC m1)`: an object file that contains the reference implementations of the functions declared in `reference.h`. This is a binary file that contains compiled reference implementations of functions. (If you are a mac m1 user delete the reference.o file and rename `reference-m1.o to reference.o`)
6. `tester.c`: unit tests for the functions that you will implement. Each unit test passes an input to your implementation of a function and to the reference implementation of the same function and compares the outputs. This file will compile into an executable, `tester`, which you will run to see if you pass the unit tests.
7. `Makefile`: instructions for compiling and building `tester` used by the `make` utility.

You workflow will consist of (1) implementing functions by modifying `student.c`, (2) typing `make` to build the `tester`, and (3) running `tester` to see if you pass the unit tests, and repeating these three steps until you pass all the tests. 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.

Below are the functions you need to implement:

1. `squareOfSmallest`: Takes an array of integers and the length of the array as input, and returns the square of the smallest integer in the array. You can assume that the input array contains at least one element.
   Example : If the array contains [-4,8,9,56,70] you have to return 16 (-4*-4) as the square of the smallest number -4.

2. `findMin`: Takes a rotated sorted array of integers and the length of the array as input and return the smallest element in the list. Example: if the input array contains [3, 4,5, 1,2], then after a call to findMin, you have to return 1.

3. `isPalindrome:` Take a string of characters and check if the string is palindrome or not. A palindrome is a word, phrase, or sequence that reads the same backward as forward, Example: if the input string is "madam", then after a call to isPalindrome you have to return True. You can assume that string contains only numbers and alphabet.

4. `freqOfChar:` Takes a string (`str[]`) and a char (`key`) as the inputs, and returns the amount of times that specific char appears within the string. You can assume that the length of the string is greater than 0 (i.e. the string contains at least one element); however, you will not be given the length of the string AND you cannot use ARRAY_SIZE().
   Example : If the string is "introtosystemsprogramming" and the char is 't' you would return 3.

5. `sort`: Takes an array of integers and the length of the array as input and sorts the array. That is, after a call to `sort` the contents of the array should be ordered in ascending 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.

6. `twoSum:` You are given an array of integers, length of the array and a target number. Two of the numbers from the array will add up to give the target. Return the array of these two numbers. Example: if the input array is [1,2,3,4], len of the array is 4,and target is 7. The answer here will be [3,4]. You need to return this array to pass this test. Note – You can assume that 2 numbers from the list will always add up to give the target.

7. `decryptPointer:` Takes an array of integers (`array[]`), the length of the array (`length`), and an array of pointers (`key[]`) as the inputs and returns an array that contains the number at each position in the original array added to the number held at the addresses at each position in `key[]`. You can assume that the length of `array[]` and `key[]` are equal and greater than zero.
   Example : If the array contains [-4, 8, 9, 56, -100] and the array of pointers gives you addresses to other integers held in memory (the addresses will be denoted `addr_i`) [`addr_1` points to 3, `addr_2` points to 7, `addr_3` points to -13, `addr_4` points to 6, `addr_5` points to 20] you would return [-1, 15, -4, 62, -80].