Assignment 1 – C Programming Basics

CMSC257 - Computer Systems

Due date: Oct 9, 2017

In this assignment, you will develop a C program to manage several data types, data structures and arrays.

Instructions:

1. Download the following file from BlackBoard and copy it to your UNIX account on the server.

   assign1-starter.tgz

2. Create a directory for your assignments and copy the file into it. Change into that directory.

   ```
   % mkdir cmsc257
   % mv assign1-starter.tgz ./cmsc257/
   % cd cmsc257
   % tar xvzf assign1-starter.tgz
   ```

   Once unpacked, you will have the following starter files in the asgn1 directory: Makefile, cmsc257-s17-assign1.c and a1support.h. The Makefile contains commands to make your program from the source code. cmsc257-s17-assign1.c contains the main function which reads in values from the standard input, as well as calls the functions you are to create as part of the exercise. The a1support.h partially defines functions that you are to implement in the course of this assignment (see below).

3. You are to complete the cmsc257-s17-assign1 program. It receives 20 float values, one per line. The code for reading those values from standard input are provided in the assignment source code starter file.

4. You are to create a new file a1support.c. This will include the code for each of the functions defined in a1support.h. You are also to complete the function definitions in a1support.h. These functions are defined in Table 1.

6. Complete the code in the cmsc257-s17-assign1.c and a1support.h files. Places where code or declarations needs to be added are indicated by ???. See in file comments for hints. The program shall perform the following functions in order as implemented within the main() function:

   (a) Read in 20 float values from the terminal and place them in an array. Note the code to perform this step is already provided.

   (b) In the main function, create a second array of integer values. For each value in float array, convert as follows:

   - *Truncate* the float value to an integer.
   - Convert all integers to positive values by taking their *absolute* values.
   - Convert these positive integers to numbers in the range 0,…,15 by implementing the *mod* operation (i.e. number *mod* 16).

   (c) Print out the values of each array on their line using the `float_display_array` and `integer_display_array` functions.

   (d) For each integer, print out the number of '1' bits in the resulting binary representation by calling

the function countBits.

(e) Sort the integer array using the integerQuickSort function. Print out the sorted integer array again using the showInts function.

(f) Create two functions that take an array and prints out the number of even values. The first function float_evens should ignore the part of the number to the right of the decimal point to determine if it is even. The second function integer_evens should count the number of even numbers as normal. Call both of these functions from the main function inside a **single** print statement annotating the returned values.

(g) Write a function most_values to figure out which values occur in the integer array calculated in the preceding step most frequently. The function will receive three parameters:

- arr - the array itself
- range - the number of elements in the array
- maxval - the largest possible value in the array

The function will print out the value which occurs the most times in the array. If there are more than one that occur as the highest number, print them all. Hint: You can assume that maxval will never exceed 16.

(h) Cast each integer to an unsigned short type and compute a number with bits reversed by calling the reverseBits function. Print out a binary representation of each of the numbers by calling the binaryString on two string arrays and printing out the resulting text.

7. Add comments to all of your files stating what the code is doing. Fill out the comment function header for each function you are defining in the code. A sample header you can copy for this purpose is provided for the main function in the code.

8. **Try to use bit-wise operators for each of the functions mentioned above**. Specifically, use bit-wise operators for the following:

a) calculating the "mod" and "absolute" values
b) countBits function (where you keep dividing by 2)
c) reverseBits function
d) finding even numbers
e) swapping variables in sorting without temporary variables

You can truncate floats to ints using a type-cast. The goal is to not use functions from the math.h library.

To turn in:

1. Create a tarball file containing the assign1 directory, source code and build files as completed above. Upload the program on BlackBoard by the assignment deadline (before class on the day of the assignment; no extensions allowed). The tarball should be named LASTNAME-EID-assign1.tgz, where LASTNAME is your last name in all capital letters and EID is your VCU E-ID.

2. Before sending the tarball, test it using the following commands (in a temporary directory – NOT the directory you used to develop the code):
   % tar xvzf LASTNAME-EID-assign1.tgz
   % cd asgn1

% make
--- (Test the program)

Function	Parameters	Description
float_display_array	A reference to the array of floats and an integer length of the array.	This function prints out an array of floats on a single line. The display width should be the same for each value. The float should only print the first two numbers to the right of the decimal point.
integer_display_array	A reference to the array of integers and an integer length of the array.	This function prints out an array of integers on a single line. The display width will be the same for each value.
integer_evens	This function should receive a reference to an array of integers and the array length.	The function should return the number of even values in the array passed.
float_evens	This should receive a reference to an array of floats and the array length.	The function should return the number of even values of the array passed (truncate float value to its lowest integer).
countBits	This should receive an integer to count bits from.	The function should return the number of nonzero bits in the number. Note that any negative sign should be ignored for the purposes of counting bits.
reverseBits	This should receive an unsigned short integer of the number to reverse.	The function should return the number (in integer format) whose bits are reversed, i.e., the top bit of the original number is the bottom bit of the returned number, the second from the top bit of the original number is the second to the bottom bit of the returned number.
binaryString	This should receive a pointer to a string, a length and a number to convert to binary.	This function should fill the text string with a binary representation of the number suitable for printing. If the string is too long, just print as many digits as is possible (don't forget to NULL terminate the string).
most_values	This function should receive a reference to the integer array, the number of elements and the maximum possible value.	The function should return the value(s) that occurs the most.
integerQuickSort	This function should receive a reference to the integer array and the index of a left and right element to sort.	The function should sort the values in the array using a quick sort from lowest to highest. You can use the algorithm listed on the corresponding Wikipedia page.

Table 1: Functions to define and implement.