Project 1 – C Programming Basics and Bitwise Operations
CMSC257 - Computer Systems
Due date: Feb 21, 2019 by 11:59 pm

In this assignment, you will develop a C program to practice arrays and bitwise operators.

Instructions:

1. Download the following file from BlackBoard and copy it to your UNIX account on the server. **p1-starter.tgz**
2. Extract files from **p1-starter.tgz**
   % tar xvzf p1-starter.tgz

   Once unpacked, you will have the following starter files in the asgn1 directory: **Makefile, cmsc257-s20-p1.c** and **p1-support.h**. The **Makefile** contains commands to make your program from the source code. To compile your program, you will just need to type **make.** To delete the files created by make you will need to type **make clean.** You can read following page to get more information about make. We will also have examples in class: https://www.gnu.org/software/make/

   cmsc257-s20-p1.c contains the main function which reads in values from the standard input, as well as calls the functions created under **p1-support.c**. The **p1-support.h** declares functions that you are to implement in the course of this assignment

3. You are to complete the **cmsc257-s20-p1.c** program. It receives 10 numbers from the command line.

4. You are to create a new file **p1-support.c**. This will include the code for each of the functions declared in **p1-support.h**. These functions are described in Table 1.

6. Places where code or declarations needs to be added are indicated by ??? you may add extra helper functions as needed. See in file comments for hints. The program shall perform the following functions in order as implemented within the main() function:

   (a) Read in 10 integers from the command line and store them in **int_array1**. This part is complete.

   (b) Convert numbers in **int_array1** to positive values by taking their absolute values and save them in **int_array2**. Use only binary operations when calculating the absolute value. Print all numbers in a single line using **display_array** function.

   (c) Convert these positive integers to numbers in the range 0,…,127 by implementing the mod operation (i.e. number mod 128. Hint: This can be achieved by masking the 7 least significant digits in the integer number) and save them back into **int_array2**. Print all numbers in a single line using *display_array* function.

(d) For each integer in **int_array2**, in a separate line: print the number, number of '1' bits in the resulting binary representation by calling the function **countBits** and print whether the number is even or odd using **evenOrOdd** function.
e.g. Assume number 0 was 24
[number 0 : 24 # of 1 bits : 2 even]

(e) Cast each element of int_array2 to unsigned int and store them into **uint_array1**.

(f) Reverse the order of array elements in **uint_array1** using **swap** function. i.e. swap first and last element then 2nd and 9th and so on.

(g) Update each element of uint_array1 by using **reverseBits** function.

(h) Print each element of **uint_array1** in a separate line along with binary representation of each of the numbers using **binaryString** function.
e.g. Assume number
[number 0 : reversed: 1342177280 0b01010000000000000000000000000000]]

7. Add comments. 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. You can add comments explaining the functions in the header file (p1-support.h)

To turn in:

1. Create a tarball file containing the p1 directory, source code and build files as completed above. Upload the program on BlackBoard by the assignment deadline. The tarball should be named username_p1.tgz

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 username -p1.tgz
% cd p1
% make
--- (Test the program)

Function	Type	**Parameters**	**Description**
display_array	void	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 needs to be the same for each value. (Hint: You can have display width of 10 by changing the format specifier as %10d) .
countBits	int	This should receive an integer to count bits from.	You are expected to use bitwise (such as >>, &) operations for this function.
reverseBits	unsigned int	This should receive an unsigned integer number to reverse.	The function should return the number (in unsigned 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. You are expected to use bitwise operations for this function.
binaryString	void	This should receive a pointer to a string, an unsigned integer number to convert to binary.	This function should fill the text string with a binary representation of the number suitable for printing. You should be using some bitwise operations to achieve this, maybe **shifting(<<)** and **and** (&) operation, you can aso use bit_get function. Binary representation of the number needs to be prepended with "0b" You are expected to use bitwise operations for this function.

bitwise-mod128	int	This should receive an integer as an input	This function should return mod 128 using bitwise operations. (Hint: This can be achieved by using bitwise AND operation using a mask, mask with 7 least digits of 1s. If the number is negative this function should return 0. You are expected to use bitwise operations for this function
bitwise_abs	int	This should receive an integer as an input.	This function should return absolute value of the input integer. (If a number is negative (if the most significant bit is 1, use bit_get) Flip all bits and add one. You are expected to use bitwise operations for this function
bit_get	int	This should receive two integer values as input.	This function should extract the specified bit from a given number. Such as bit_get(num, 0) should return the leftmost binary digit in num. You are expected to use bitwise operations for this function
evenOrOdd	char*	This should receive an integer value.	Should return 1 if number is even and 0 if number is odd. You can find this out by looking at the least significant binary digit in a number. (You can use bit_get)
swap_ints	void	This should receive two unsigned integer pointers as input.	Should swap the numbers without using temp variable. This can be achieved using exclusive or operation.(bitwise operation) Look at the example in the textbook . You are expected to use bitwise operations for this function

Table 1: Functions to define and implement.

For the following sample run: ./cmsc257-s20-p1 0 1 7 131 515 -1 -7 -131 -90 -100

Your output should look like:

```
{   0,   1,   7, 131, 515,   1,   7, 131,  90, 100}
{   0,   1,   7,   3,   3,   1,   7,   3,  90, 100}
[number          0 :           0 # of 1 bits:          0    Even]
[number          1 :           1 # of 1 bits:          1     Odd]
[number          2 :           7 # of 1 bits:          3     Odd]
[number          3 :           3 # of 1 bits:          2     Odd]
[number          4 :           3 # of 1 bits:          2     Odd]
[number          5 :           1 # of 1 bits:          1     Odd]
[number          6 :           7 # of 1 bits:          3     Odd]
[number          7 :           3 # of 1 bits:          2     Odd]
[number          8 :          90 # of 1 bits:          4    Even]
[number          9 :         100 # of 1 bits:          3    Even]
[number          0 : reversed : 637534208    0b00100110000000000000000000000000]
[number          1 : reversed :1509949440    0b01011010000000000000000000000000]
[number          2 : reversed :3221225472    0b11000000000000000000000000000000]
[number          3 : reversed :3758096384    0b11100000000000000000000000000000]
[number          4 : reversed :2147483648    0b10000000000000000000000000000000]
[number          5 : reversed :3221225472    0b11000000000000000000000000000000]
[number          6 : reversed :3221225472    0b11000000000000000000000000000000]
[number          7 : reversed :3758096384    0b11100000000000000000000000000000]
[number          8 : reversed :2147483648    0b10000000000000000000000000000000]
[number          9 : reversed :          0    0b00000000000000000000000000000000]
```