CS130 Lab Exercise #2

Overview

This lab exercise is intended to give students experience in dealing with operations at what is sometimes referred to as “bitwise” level. In CS116, you were expected to master logical operations with constructs such as AND (`&&`), OR (`||`), NOT (`!`), `if`, `else if`, and `while`. In this exercise, we will be dealing with bitwise logical operators `&`, `|`, `~`, `^` (XOR), Left Shift (`<< n`), and Right Shift (`>> n`). It is entirely possible that these operators are new to you, but operations such as these are germane to our upcoming studies in CS130 this semester.

This document is a required lab exercise for all students in the class who opt to not do the more-difficult Extra Credit exercise. The file name for the Extra Credit document is ‘PP CS130 Extra Credit Lab Exercise #2.PDF’ and it too is posted on “Canvas”.

A companion “Help” document is provided to students, posted on “Canvas” as ‘Demo Program To Display Bits Source.Txt’. This document – a text file – contains constructs that may be copied and pasted into your source code to save time in typing. Some modification of the text will likely be required to match up with the names you have chosen for variables in your design.

Requirements

Following is a list of requirements for this lab exercise, all of which must be met:

1.) Using the construct `sizeof()`, determine the size of unsigned long integers (declare them as `long long unsigned`) on your computer. Display the result on the console (“monitor”).
2.) Prompt the User to enter an unsigned long integer (Range: 0 to 18,446,744,073,709,551,615).
3.) Validate this entry as an unsigned integer and reject invalid entries. A method to effect this validation process is provided on the companion document posted on “Canvas”. See “Validating Keyboard Inputs Source.Txt”.
4.) Prompt the User to enter a 2nd unsigned long integer (Range: 0 to 18,446,744,073,709,551,615). Prompt the User to ensure that this value is greater than the first integer entered.
5.) Validate this entry as an unsigned long, long integer and reject invalid entries.
6.) Each input value or entry must be labeled. Align the displayed binary values so that the entered values and any output value are aligned horizontally such that Bit 0 of each entry or output value is directly above or below Bit 0 of all other input values or output results.
7.) Add the two values together.
8.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
9.) Pause the screen, prompting the User to press ‘any key’ to continue.
10.) Examine the binary representation of the sum and note its value.
11.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (12)
12.) Subtract the second value that was entered from the first. Remember that the result will be negative so you must store the difference in an integer that can be either positive or negative.
13.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
14.) Pause the screen, prompting the User to press any key to continue.
15.) Examine the binary representation of the difference and note how a negative value is formed.
16.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (17)
17.) Using the bitwise operator `&`, logically AND these two integers.
18.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
19.) Pause the screen, prompting the User to press any key to continue.
20.) Examine the binary representation of the sum and note its value.
21.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (22)
22.) Using the bitwise operator `|`, logically OR these two integers.
23.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
24.) Pause the screen, prompting the User to press any key to continue.
25.) Examine the binary representation of the sum and note its value.

26.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (27)
27.) Using the bitwise operator **^**, logically XOR these two integers.
28.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
29.) Pause the screen, prompting the User to press any key to continue. Examine the result.
30.) Examine the binary representation of the sum and note its value.
31.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (32)
32.) Using the bitwise operator **~**, logically NOT the first integer.
33.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
34.) Pause the screen, prompting the User to press any key to continue.
35.) Examine the binary representation of the sum and note its value.
36.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (37)
37.) Using the unary operator -, negate the first integer. (Careful here!)
38.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
39.) Pause the screen, prompting the User to press any key to continue.
40.) Examine the binary representation of the sum and note its value.
41.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (42)
42.) Prompt the User that a left shift operation will next be performed on the first integer. Prompt the User to enter the number of places this integer will be shifted.
43.) Qualify this number to ensure that (a) it is a positive number and (b) that is does not exceed 63.
44.) Using the left shift operator **<<**, perform a logical shift on the first integer.
45.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
46.) Pause the screen, prompting the User to press any key to continue.
47.) Examine the binary representation of the sum and note its value.
48.) When a key is pressed, clear the screen, redraw the two entered integer values, and execute (49)
49.) Prompt the User that a right shift operation will be performed on the first integer. Prompt the User to enter the number of places this integer will be shifted.
50.) Qualify this number to ensure that (a) it is a positive number and (b) that is does not exceed 63.
51.) Using the right shift operator **>>**, perform a logical shift on the first integer.
52.) Display each of the three values left-to-right as (a) binary, (b) hexadecimal, and (c) decimal.
53.) Pause the screen, prompting the User to press any key to continue.
54.) Examine the binary representation of the sum and note its value.
55.) Upon the next press of a keyboard key, terminate the program.

Additional Requirement

While the screen is paused for each of the operations listed above, examine both the input binary values and the resulting output binary values. Ensure that (a) you know what the result should be, and (b) that your most magnificent program performs properly, showing the correct output.

Screenshots

Go to the file “Program Output Screenshots.Zip” posted on “Canvas” to see what a sequential set of possible outputs might look like. You will need to use **`setw()`** to stabilize the column width for hexadecimal and decimal values since the number of displayed digits for these values is not constant.

Final Action

When your program is “Mary Poppins Perfect”, submit [ONLY] your source code to me at:

RSturlaCS130@GMail.com

Naming Convention

For this and all subsequent submissions, name your files as follows. Substitute your **last name** exactly as it appears on the class roster for mine. Format:

Sturla Lab #2 Source.cpp

If you prefer, you may submit your source code as a text file. Example:

Sturla Lab #2 Source.Txt