**Objective**

We’ll be creating a small program that will display all the binary in an int. We’ll be manipulating the bits inside the integer and displaying them to the user. First, however, we should set up some global helper functions.

You may also choose to declare your unsigned int bitField globally so all functions can use it, otherwise you’ll have to add it as an additional parameter to all your helper functions and instead declare it in main. If you make it a parameter, ensure that it is an unsigned int reference (&) so the original value can be changed, or that the function returns the result of the bit operation as an unsigned int so you can update the value of bitField in the main code that is calling the helper function.

**TurnOn(int bit) – 10 points**

This function should take in a number to determine which bit should be turned on. You may set this up to be the bit index (like an array, but 0 would be the right-most, 1 would be the one to the left of that, etc) or you may have it represent the actual bit value itself (1, 2, 4, 8, 16, etc). Either method is fine, it just depends on how you choose to write your main code to decide which number to pass as the parameter.

HINT: “or” |

**TurnOff(int bit)** – 10 points

Similar to TurnOn, but instead ensure that specific bit is turned off.

HINT: “negate” ~, “and” &

**Toggle(int bit)** – 10 points

As in the previous two, this function will toggle (0 becomes 1, 1 becomes 0) the specific bit. If checks do not need to be used to complete this method.

HINT: “xor”, aka “exclusive or” ^.

**Negate()** – 10 points

Invert all the bits using the negate operator. Since this is a unary (single parameter) operator, you do not need to know which bit to manipulate because it manipulates all of them. Loops do not need to be used to complete this method.

**LeftShift()** and **RightShift() - 20 points**

These two separate functions will left and right shift the number by 1 in the respective direction. Similar to Negate(), they do not require a specific bit to operate on as they shift the entire bit field.

**Using a loop, array, or vector in any of the above functions is incorrect and will result in a severe grading hit.**

**Main.cpp – 40 points**

As mentioned prior, either declare your unsigned int bitField globally or at the top of main. Display the value of the bitField normally and also print the on/off state of each of the 32 bits within the int. HINT: “and” & can be used to test the value of a bit. If your bitField is a regular signed int, issues can arise because the left-most 31st bit is reserved for the signed (+ or -) state of the int instead of an actual value.

Example Output

bitField: 19

Bits: 0000 0000 0000 0000 0000 0000 0001 0011

TurnOn = 1, TurnOff = 2, Toggle = 3, Negate = 4, Left Shift = 5, Right Shift = 6

Choose operation: 1

// Choose bit index: 8

// -or-

// Choose bit value: 8

You will need to ask the user for which operation to execute, as well as the bit of interest (if applicable for that operation). Depending on your decided method for bit, you would ask either the literal bit value (8 value would be the 4th bit, AKA bit #3 because 0-based index) or the bit index (8 index would be value 256, AKA bit #9 because 0-based index).

HINT: Using Windows Calculator and setting it to Programmer Mode will help assist you with the visualization and mathematical results of all the bit operators as well as seeing the decimal value and bit states of that value.  
  
**Do not use std::bitset** to display the bits. This will be a severe grading hit.