# CSCE 3600: Systems Programming

## Major Assignment 1 – GitLab and Bitwise Operators

### Due: 11:59 PM on Wednesday, February 12, 2020

### COLLABORATION

You should complete this assignment as a group assignment with the other members of your group as assigned on Canvas using our GitLab environment. If desired, students may submit their program to Canvas, but all "production" source code that will be graded *must* be committed to the master branch in GitLab.

There are group components as well as individual components identified in this assignment. For the group component, all group members are responsible (and can work together completely) for this functionality and will be graded as such, although peer evaluations may reduce or increase an individual group member's score on the group component. For the individual component of this assignment, individual group members will be solely responsible for the respective code and graded as such. This includes submission of all respective code to GitLab, which will be verified when grading. Individuals may receive collaborative assistance from other members of their group (only) for their portion, but ultimately each individual is responsible for his/her own individual component in GitLab.

Please make sure that all group members who participated in the assignment are listed in the code and README files.

### PROGRAM DESCRIPTION

In this introductory major assignment, you will write a complete C program that will implement a set of operations supported by a menu that will prompt for the operation ("Power of 2", "Reverse Bits", "Replace Bit Position from Mask", and "Palindrome") along with a positive integer less than 2 billion, and then perform that operation on that integer (based on possible additional input) to produce the result. More details about the operations are found below. The twist in this assignment is that each operation must be performed using bitwise operators rather than the traditional operations. That said, relational operators and the like can be used in branching and loops, but data manipulation must be done using bitwise operators.

This project will be organized in a header file called **`major1.h`**, a source code file called **`major1.c`**, which will contain the `main()` function, and four *individual* source code files called **`power.c`**, **`reverse.c`**, **`replace.c`**, and **`palindrome.c`** that will contain the function definition for their respective arithmetic operations. The entire group will be responsible in general for non-operation specific functionality in the **`major1.h`** and **`major1.c`** source code files while each individual group member is responsible for his/her own "functional operation" in the appropriate files (including the function declaration in the header file and the function call inside the `main()` function).

In particular, you are expected to have the following functionality for each file:

- **major1.h**

  This file is the overall header file for the project and will contain any preprocessor directives, such as `include` and `define` directives, and function prototypes. While the `include` directives are general for the team, each member is expected to add their own function prototype (i.e., function declaration) for the operation that he/she is responsible for.

- **major1.c**

  This is the code file with the `main()` function that will do the following: (1) display the menu, (2) read in the user's response for the menu selection, (3) prompt for and read in a positive integer less than two billion, and then, based on the menu selection, (4) call the appropriate function call for the specified operation, passing the integer operand as a parameter to that function. This functionality will be contained in a loop that will continue to iterate until the user selects the option to end the program. If the user enters a valid outside of the 1 – 5 range, you will print a meaningful error message and re-display the menu. Additionally, you will continue to prompt for and read in the integer operand until the user enters an acceptable value (no error message is needed here). While the code to display the menu as well as prompt for and read in the integer operand are considered to be part of the group component, each group member is expected to add the function call for the operation he/she is responsible for.

- **power.c**

  This code will contain a single function that accepts a single positive integer less than two billion (and the include directive to your header file) to perform the following functionality: determine if the passed-in integer parameter is a power of two (i.e., is there some integral value N for which the positive integer $2^N$ less than two billion exists, such as 32, which is $2^5$) and if it is not a power of two, calculate the next integer higher than the passed-in integer parameter that is a power of two. For example, if the user enters the positive integer 12 (which is not a power of two), the next higher integer that is a power of two is 16. The operations to determine whether or not the positive integer is a power of two, plus the calculation of the next higher integer that is a power of two, must be done using bitwise operators. One team member, and only one team member, will be responsible for the source code in this file in GitLab, though collaboration with other group members may be done if needed.

- **reverse.c**

  This code will contain a single function that accepts a single positive integer less than two billion (and the include directive to your header file) to perform the following functionality: reverse the bits (all 32 of them) and then print out the decimal value of the new integer (i.e., the one with the bits reversed). For example, if the user enters the positive integer 2 (which is 00…0010), this function would reverse the digits to 0100…00, which is 1073741824. The operations to reverse the bits must be done using bitwise operators. One team member, and only one team member, will be responsible for the source code in

this file in GitLab, though collaboration with other group members may be done if needed.

- **`replace.c`**

  This code will contain a single function that accepts a single positive integer less than two billion (and the include directive to your header file) to perform the following functionality: (1) prompt for and read in a positive integer "mask" less than three billion and continue to prompt for and read in the positive integer mask until the user enters and acceptable value (no error message is needed here), (2) prompt for and read in the bit replacement position from the mask and continue to prompt for and read in the bit replacement position from the mask until the user enters an acceptable value (no error message is needed here, but note that since we are working with 32-bit unsigned integers, the value should be between 0 and 31, inclusively), and finally (3) replace the single bit in the original positive integer less than two billion passed to the function with the single bit from the positive integer mask specified by the user (i.e., in the bit replacement position from the mask). For example, if the user initially enters 7 (in binary, 00…00111) as the positive integer less than two billion, then enters 8 (in binary, 00…01000) for the positive integer mask and 3 for the bit replacement position, you will replace the third bit from the positive integer 7 (a 0) with the third bit from the positive integer mask (a 1), resulting in the new value 15 (in binary, 00…01111). The operations to replace the bit must be done using bitwise operators. Additionally, no loops may be used (except in error checking when prompting for and reading in the positive integer mask and bit replacement position from the mask) to achieve this functionality. One team member, and only one team member, will be responsible for the source code in this file in GitLab, though collaboration with other group members may be done if needed.

- **`palindrome.c`**

  This code will contain a single function that accepts a single positive integer less than two billion (and the include directive to your header file) to perform the following functionality: print out the binary representation of the positive integer and then determine if the positive integer is a palindrome (based on all 32-bits). For example, if the user enters 1073741826 (in binary, 0100…0010), it is a palindrome. Check that 1073741826 is not a palindrome. The operations to determine whether or not the positive integer less than two billion is a palindrome must be done using bitwise operators. As a hint, you may use an array, but remember that the significant logic to determine whether or not the positive integer is a palindrome must be done using bitwise operators. One team member, and only one team member, will be responsible for the source code in this file in GitLab, though collaboration with other group members may be done if needed.

Note that the expectation for this assignment assumes that a group contains 4 students, but if, for some reason, a team has only 3 students, then only 3 of the operations would need to be supported. This means that each group member is responsible for one and only one operation (plus the group component of the assignment). If you have any

questions on what is acceptable, please discuss this with your instructor.

**SAMPLE OUTPUT** (user input shown in **bold**):

```
$ make clean
rm -f *.o binops
$ make
gcc -c -Wall power.c -lm
gcc -c -Wall reverse.c -lm
gcc -c -Wall replace.c -lm
gcc -c -Wall palindrome.c -lm
gcc -c -Wall major1.c -lm
gcc -o binops power.o reverse.o replace.o palindrome.o major1.o -lm
$ ./binops
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 0
Error: Invalid option. Please try again.
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 6
Error: Invalid option. Please try again.
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 1
Enter a positive integer less than 2 billion: 0
Enter a positive integer less than 2 billion: 2000000000
Enter a positive integer less than 2 billion: 32
32 is a power of 2
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 1
Enter a positive integer less than 2 billion: 72346
72346 is not a power of 2
Next higher integer that is power of 2 is: 131072
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
```

```
--> 2
Enter a positive integer less than 2 billion: 0
Enter a positive integer less than 2 billion: 2000000000
Enter a positive integer less than 2 billion: 2
2 with bits reversed is 1073741824
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 2
Enter a positive integer less than 2 billion: 1073741824
1073741824 with bits reversed is 2
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 2
Enter a positive integer less than 2 billion: 237834
237834 with bits reversed is 1350942720
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 3
Enter a positive integer less than 2 billion: 0
Enter a positive integer less than 2 billion: 2000000000
Enter a positive integer less than 2 billion: 7
Enter a positive integer mask up to 3 billion: 0
Enter a positive integer mask up to 3 billion: 3000000000
Enter a positive integer mask up to 3 billion: 8
Enter the bit replacement position from mask (0-indexed): -1
Enter the bit replacement position from mask (0-indexed): 32
Enter the bit replacement position from mask (0-indexed): 3
New integer with bit 3 from mask 8 is 15
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 3
Enter a positive integer less than 2 billion: 238475
Enter a positive integer mask up to 3 billion: 2983434345
Enter the bit replacement position from mask (0-indexed): 18
New integer with bit 18 from mask 2983434345 is 238475
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
```

```
--> 3
Enter a positive integer less than 2 billion: 238475
Enter a positive integer mask up to 3 billion: 2983434345
Enter the bit replacement position from mask (0-indexed): 22
New integer with bit 22 from mask 2983434345 is 4432779
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 4
Enter a positive integer less than 2 billion: 0
Enter a positive integer less than 2 billion: 2000000000
Enter a positive integer less than 2 billion: 1073741826
The binary representation is: 01000000000000000000000000000010
1073741826 is a palindrome
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 4
Enter a positive integer less than 2 billion: 298554990
The binary representation is: 00010001110010111001011001101110
298554990 is not a palindrome
Enter the menu option for the operation to perform:
(1) POWER OF 2
(2) REVERSE BITS
(3) REPLACE BIT POSITION FROM MASK
(4) PALINDROME
(5) EXIT
--> 5
Program terminating. Goodbye.
```

## REQUIREMENTS

Your header file, source code files, **`makefile`**, and README file will be committed to the master branch in the GitLab repository as follows. No other files than those mentioned here should be present in the GitLab repository.

- Your source code and header files should be well documented in terms of comments. For example, good comments in general consist of a header (with your name(s), course section, date, and brief description), comments for each variable, and commented blocks of code.
- A **`makefile`** for compiling your source code, including a clean directive. To ensure that your C code is compiled correctly, you will need to create a simple **`makefile`**. This will allow our scripts to just run `make` to compile your code with the right libraries and flags. When using `gcc` to compile your code, please use the `-Wall` switch to ensure that all warnings are displayed. Do not be satisfied with code that merely compiles – it should compile with no warnings! You will lose points if your code produces warnings when compiled.

- A **`README`** file will contain some basic documentation about your code. Specifically, this file should contain the following four components:
    - Your name(s).
    - *Organization of the Project*: Identify the responsibilities for each group member, including which operations each individual was responsible for and what role(s) he/she served as (i.e., ScrumMaster, Product Owner, or Team). Roles may be re-assigned for each *sprint*. Note that this may be used in assessment of grades for this project.
    - Known Bugs or Problems: A list of any features that you did not implement or that you know are not working correctly.
- A completed group assessment evaluation (given at a later date) for each team member. Each team member will be asked to explicitly rate himself/herself along with every other team member. *A student receiving a poor evaluation with regards to their performance on the team will have his/her grading marks reduced by an appropriate amount, based on the evaluation. In addition, the rubric for this assignment allows modification of each individual's portion of the group grade to account for individual participation and contribution to the group's submission, which may result in a member of the group receiving a much higher or much lower grade than other members of the group. A student found to not contribute to the team, will be removed from the team.*
- Your program will be graded based largely on whether it works correctly on the CSE machines (e.g., `cse01`, `cse02`, …, `cse06`), so you should make sure that your program compiles and runs on a CSE machine.

**SUBMISSION**

- Each team will ensure that all source code and header files, the **`makefile`**, and the **`README`** file are committed to the master branch in the GitLab repository by the due date and time. If desired, one student may all applicable files to Canvas, but it is not required.
- Points in the grade (based on 100 points) will be allocated as follows: 30% of the points will be based on the group components (e.g., the menu, prompting for and reading in the integer operands, etc.) while 70% of the points will be based on the individual contribution specifically associated with the individual operation each student is responsible for in the group.