CSCI 1730 Breakout Lab 03

Bitwise And, Bitwise Or, and Bitwise Xor

Learning Outcomes

- Understand how unsigned integers are represented and stored in memory by converting unsigned integers from binary to base-10.
- Understand binary representation of integers in memory and use bitwise operators to solve problems.
- Trace, design, and implement software solutions to non-trivial problems using the C programming language.
- Design and implement programs that use command line arguments.

Problem / Exercise

The purpose of this lab is to get experience with the bitwise and operator, the bitwise or operator, and the bitwise xor operator. You will implement a C program that will process three command line arguments: a bit string, a flag to represent a bitwise operator, and a second bit string, and it should output the result of a bitwise operator in binary and in base-10 as shown in the examples. The three possible flags for this assignment are `-and` to denote the `&` operator, -or to denote the | operator, and `-xor` to denote the `^` operator. Let n be the length of the longest bit string inputted on the command line. You should assume $1 \leq n \leq 64$ for this assignment. For command line arguments with bit strings of different lengths, the shorter length bit string should be padded with leading zeros to make it the same length as the longer bit string as shown in the examples. You may assume all of the bit strings in this assignment represent unsigned integers. Before you write any code for this assignment, you should work through all examples with paper and pencil.

Examples

Your C source code should be in a file called `lab03.c`, and it should be compiled into an executable called `lab03.out`. Your C program must look exactly like the examples below when run on the command line on odin, it must compile correctly, and it must run correctly with any valid sequence of inputs provided as command line arguments. You may assume that the command line arguments will be entered in a correct manner as demonstrated by the examples. Each example is a separate execution of a correct program for this assignment.

```
./lab03.out 1 -and 110000
000001 & 110000 evaluates to 000000 using bit strings of length 6
1 & 48 evaluates to 0 using unsigned 6-bit integers

./lab03.out 0010 -and 011
0010 & 0011 evaluates to 0010 using bit strings of length 4
2 & 3 evaluates to 2 using unsigned 4-bit integers

./lab03.out 001000010 -or 0
001000010 | 000000000 evaluates to 001000010 using bit strings of length 9
66 | 0 evaluates to 66 using unsigned 9-bit integers

./lab03.out 1011100 -or 11000
1011100 | 0011000 evaluates to 1011100 using bit strings of length 7
92 | 24 evaluates to 92 using unsigned 7-bit integers

./lab03.out 1100001011 -xor 01111000
1100001011 ^ 0001111000 evaluates to 1101110011 using bit strings of length 10
779 ^ 120 evaluates to 883 using unsigned 10-bit integers
```

```
./lab03.out 0011 -xor 0101111100011
0000000000011 ^ 0101111100011 evaluates to 0101111100000 using bit strings of length 13
3 ^ 3043 evaluates to 3040 using unsigned 13-bit integers
```

1 C Program

1.1 General Requirements

1. Place your C source code for this assignment in a file named `lab03.c`. See the Examples section to see what your C program should output. Compile and link your C program using the following command. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.

   ```
   gcc -Wall lab03.c -o lab03.out
   ```

2. Place the files for this assignment in a directory called `lab03`.

3. Your program's source file(s) must compile on odin using the required compiler for this course (`gcc version 11.2.0`).

4. Do *NOT* include `math.h` in your source code. Do *NOT* call any functions in `math.h`.

5. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).

2. All structs, unions, and enums must be commented.

3. All global variables and static variables must be commented.

4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.

5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 03. Submit only the following files.

1. All source files required for this lab: lab03.c

2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	7 points
README.txt file missing or filled out incorrectly	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Not submitting this assignment before its late period expires	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.