Programming Assignment - Bit Manipulations

Some things about this programming assignment.

1. Be careful and back up your files occasionally. It is very easy to have a typo delete your files.

   The line below will compile your files. The line below it will ***ERASE*** your bits.cc.

   g++ -o bits bits.cc main.cc

   g++ -o bits.cc main.cc

   (The ***-o bits.cc*** means to create an output file named bits.cc from the compile result of main.cc. The compiler will create a new output file named bits.cc which will overwrite your bits.cc. The file main.cc will probably not compile which will leave bits.cc empty.)

2. We are learning C but we are using a C++ compiler. That does NOT mean you should be using C++ in your programs. Mostly, don't use CIN and COUT. I will be asking questions about scanf and printf (and other things we cover in class) on the exams and you need practice using those functions. You can safely use anything in my notes or in the book.

3. The book does have chapters on C and they are really good. Chapters 11 through 16 covers what you will need to know for this class in C.

4. Some of you may know C and have some experience. Some of you may look things up on the internet and find code that others have written that would make some of the tasks I've given you easier. So, to be perfectly clear YOU ARE NOT ALLOWED TO USE ANY EXTRA LIBRARIES. For this assignment you ARE ONLY ALLOWED TO INCLUDE STDIO.H. You need to learn the basics of C in this class.

5. I only want you to create the functions I have specifically asked for. While breaking a problem into pieces and writing functions to solve those pieces is normally a good practice, for this assignment each function needs to be its own self contained unit. It also helps keep the complexity of the tests down too. So you should submit a bits.cc with ONLY the three functions requested inside.

Programming Assignment - Bit Manipulations

Instructions:

1. Create a C program file named ***bits.cc***. Create functions named ***printIP, reverseEndian,*** and ***countGroups.*** These functions will be described in the following steps. Note that Linux is VERY case sensitive, so do not name you file Bits.cc.

2. Function ***printIP*** should take a 32 bit integer as parameters and print the integer as if it were an IP address.

   PRINT A SINGLE NEWLINE AT THE END OF THE IP ADDRESS.

   IP addresses really are simply 32 bit numbers. They are written using the familiar dot notation to make it easier for humans to remember them. The values and groupings of each 8 bit section of an IP address have meaning which I will be happy to discuss with you outside of class.

   The following 32 bit number can be divided into four bytes. Each byte is simply written as a decimal number with dots in between.

   For example, the number 33,783,932 in decimal

   is 0x0203807C in hex

   is 00000010000000111000000001111100 in binary

00000010	00000011	10000000	01111100	Break into bytes
2	3	128	124	Each byte to decimal

   2.3.128.124

   SO

   printIP(33783932);

   Should print

   2.3.128.124

   Followed by a newline of course.

Programming Assignment - Bit Manipulations

3. Function ***reverseEndian*** takes a 32 bit number as a parameter and **RETURNS** the number with its endianness reversed.

   Big endian and little endian are two different ways numbers can be stored in a computer's memory. A computer will usually use one method or the other exclusively, but there are some hardware implementations that can use both.

   Using 33,783,932 as an example, a byte addressable memory would have to store the number a 4 bytes as we did before. The bytes would be 2, 3, 127, and 124 (hex 02, 03, 80, 7C).

   Assuming the 32 bit integer is stored at memory location 1008 the two methods of storing the number are illustrated in the table below.

1004	1005	1006	1007	1008	1009	100A	100B	
00	00	00	00	02	03	80	7C	Big Endian
00	00	00	00	7C	80	03	02	Little Endian

   Which is correct? Either. Which is best? It depends on the application and the engineer.

   Note that it doesn't affect how we write the number in hex in any way. Regardless of how it is stored on the computer we ALWAYS write hex with the most significant digits to the left.

   So the number 33783932 will always be written 0x0203807C.

   The function reverseEndian should simply reverse the order of the bytes of the value passed as an argument.

   Calling reverseEndian with the following argument

   ```
   reverseEndian(33783932);
   ```

   will return (AND NOT PRINT)

   ```
   2088764162
   ```

   Of course you can and should print the return value to make sure.

   What would the following return? Does your function return correctly?

   ```
   reverseEndian(1111111);

   reverseEndian(5);

   reverseEndian(0xFAFBFCFD);
   ```

   Note that testing is much easier to verify if you pass arguments and print in hex.

Programming Assignment - Bit Manipulations

4. Function ***countGroups*** takes an integer as a parameter and counts the number of groups of adjacent 1 bits and returns that number.

   Calling

   ```
   countGroups(207);
   ```

   would return 2 since the binary for 207 contains two groups of 1s.

   11001111 One group is highlighted in blue the other highlighted in red.

   ```
   countGroups(0x1ACE);
   ```

   should return 4

   0001101011001110

5. Create a file named main.cc with a main method inside. Use this main method to test your bits.cc file. DO NOT PUT A MAIN FUNCTION IN bits.cc. Make sure to test your functions thoroughly using values ***OTHER*** than the ones given above. Make sure to check boundary values.

6. There is a main.cc attached to this document and main.cc is also provided as a download on AsULearn.
   DO NOT SUBMIT MAIN.CC TO WEB-CAT.
   DO NOT SUBMIT A bits.cc THAT HAS A MAIN MEHOD IN IT.

7. Compile and run the program
   a. Compile with the following command. Cut and paste may not work. If you cut and paste and get errors try typing over the dashes. You MUST clear all warnings.

      ```
      g++ -Werror -o bits main.cc bits.cc
      ```

   b. Run by typing the following.

      bits

      or on student or other linux machines you may have to type

      ./bits

Programming Assignment - Bit Manipulations

8. When you have completed your project, upload bits.cc (DO NOT SUBMIT MAIN.CC. DO NOT SUMBIT bits.cc WITH A MAIN IN IT.) to Web-CAT for grading. You will lose points for excessive uploads so you must thoroughly test your work **BEFORE** uploading. You will lose 10 points per day that it is late. After 2 days late you will fail the assignment. You have 5 attempts to submit successfully without penalty. You will lose 1 point for every submission after 5.

   If you have issues come see me early, especially if you don't know why you are failing the tests. If you use all 5 of your non-penalized submissions you won't get them back even if it was a silly error or simple misunderstanding on your part.

   Here is a link to Web-CAT for this course. Note that it is NOT the same server you used in CS1440 or CS2440.

   SAMPLE MAIN ON NEXT PAGE

Programming Assignment - Bit Manipulations

```
//The following is a sample main.cc.  Modify it to test your program.
//Make sure to test thoroughly.  All the pieces below have to be included.
#include <stdio.h>

//These are called prototypes and they have to be here.
//Any function in bits.cc that you also call from main must be listed here.
//For now, make sure you include the following 3 lines in main.cc
void printIP(int);
int reverseEndian(int);
int countGroups(int);

//This function will run when you run your program from the command line.
int main(void)  {
        int answer;

        //For the print functions you will have to verify that the correct
        //output was printed yourself.
        printf("Verify the following manually\n");
        printIP(33783932);
        printIP(0x1234567A);

        //Test reverseEndian.  Note that this only prints tests that fail.
        //Put an else in if you want to see tests that pass also.

        //reverseEndian test 1
        answer = reverseEndian(0x12345678);
        if (answer != 0x78563412)     {
                printf("Failed reverseEndian test 1\n");
                printf("Expected: %X\t", 0x78563412);
                printf("Found: %X\n", answer);
        }

        //Put more reverseEndian tests here

        //Test countGroups. Note that this only prints tests that fail.
        //Put an else in if you want to see tests that pass also.

        //countGroups test 1
        answer = countGroups(207);
        if (answer != 2)       {
                printf("Failed countGroups test 1\n");
                printf("Expected: %d\t", 2);
                printf("Found: %d\n", answer);
        }

        //Put more countGroups tests here

        return 0;
}
```