Programming Practice: Logic Operations

1. Use ***bitwise logical operations*** to write a C program and to perform the following operations:

   (a) Enter an even positive integer n, where $0<n\leq 2000$.
   (b) Use n as the seed of srand() to randomly generate n bytes of data $X_0X_1X_2X_3\ldots X_{n-2}X_{n-1}$, where $0\leq X_i<256$, i=0, 1, …, n-1.
   (c) Output $X_0X_1X_2X_3\ldots X_{n-2}X_{n-1}$ using hexadecimal numerals with at most 30 bytes in a line and leave a space between two bytes.
   (d) Let the bits of two consecutive bytes $X_{2k}X_{2k+1}$, where k=0, 1, …, n/2-1, be $c_7c_6c_5c_4c_3c_2c_1c_0$ and $d_7d_6d_5d_4d_3d_2d_1d_0$, respectively. Compute $Y_{2k}Y_{2k+1}$ such that Y_{2k} is set to $d_0d_1d_2d_3d_4d_5d_6d_7$ and Y_{2k+1} is set to $\overline{c_3c_2c_1c_0c_7c_6c_5c_4}$.
   (e) Output $Y_0Y_1Y_2Y_3\ldots Y_{n-2}Y_{n-1}$ using hexadecimal numerals with at most 30 bytes in a line and leave a space between two bytes.

   Example of program execution:

```
Enter an even positive integer n (0<n<=2000): 258

The randomly generated byte data are:
71 88 B2 D4 BF 7D 71 F8 68 7B CD 29 4C B1 C3 BC 73 D9 BA CD A0 75 B9 98 29 58 D6 27 05 69
F2 2E 27 A4 A3 33 0A 61 72 F1 8E D9 47 FF 02 E6 21 AD 82 0A 0A 2F 83 B3 4B FD AC BD DC 79
E8 39 1E D2 B9 6D CC 29 CF 1B 2F F3 D9 07 92 9F 9B B3 F6 78 40 6D 07 CC F2 8B 4E 4E A9 F6
A7 B9 40 E5 F6 B6 CE EB 16 00 33 34 12 C6 F1 0C 98 51 3E A0 AA E6 57 09 97 EE 16 86 2A 54
C7 09 21 2F 38 F6 E0 A1 0F 27 67 FE 60 33 82 31 7F F0 3F 5C 13 36 A5 BF 6C E6 A3 DB 17 2B
47 D7 AF 00 31 22 F5 F3 46 5A 23 28 A9 6D 7E A0 E7 FE 2B BB 71 0A 43 FC 50 CB 2A 0D 14 DC
1C 03 0D 9F 08 E2 BE DC A2 64 8F AA B4 F6 C2 93 B4 04 A8 F3 9C 76 15 A1 F6 7B 11 D1 36 85
D0 39 4D 95 E5 73 7B B8 B2 A3 65 D1 23 59 67 9A 9C B9 2B E6 2F D9 7B 29 13 6A 53 39 51 9B
3B 56 C1 3A D3 69 35 1C B0 89 F4 C0 66 C3 6A 2F E6 1B


The processed byte data are:
11 E8 2B D4 BE 04 1F E8 DE 79 94 23 8D 3B 3D C3 9B C8 B3 54 AE F5 19 64 1A 6D E4 92 96 AF
74 D0 25 8D CC C5 86 5F 8F D8 9B 17 FF 8B 67 DF B5 ED 50 D7 F4 5F CD C7 BF 4B BD 35 9E 32
9C 71 4B 1E B6 64 94 33 D8 03 CF 0D E0 62 F9 D6 CD 46 1E 90 B6 FB 33 8F D1 D0 72 1B 6F 65
9D 85 A7 FB 6D 90 D7 13 00 9E 2C CC 63 DE 30 E0 8A 76 05 1C 67 55 90 8A 77 86 61 9E 2A 5D
90 83 F4 ED 6F 7C 85 F1 E4 0F 7F 89 CC F9 8C D7 0F 08 3A 0C 6C CE FD A5 67 39 DB C5 D4 8E
EB 8B 00 05 44 EC CF A0 5A 9B 14 CD B6 65 05 18 7F 81 DD 4D 50 E8 3F CB D3 FA B0 5D 3B BE
C0 3E F9 2F 47 7F 3B 14 26 D5 55 07 6F B4 C9 D3 20 B4 CF 75 6E 36 85 AE DE 90 8B EE A1 9C
9C F2 A9 2B CE A1 1D 48 C5 D4 8B A9 9A CD 59 89 9D 36 67 4D 9B 0D 94 48 56 CE 9C CA D9 EA
6A 4C 5C E3 96 C2 38 AC 91 F4 03 B0 C3 99 F4 59 D8 91
```

2. A *parity bit* (https://en.wikipedia.org/wiki/Parity_bit) is a bit added to a binary stream to check the number of 1-bits in the stream is even or odd. For the case of **even parity**, if the number of 1-bits in the binary stream is odd, then the parity bit must be 1; if the number of 1-bits is even, the parity bit must be 0. For the case of **odd parity**, the value of the parity bit is reversed. Usually, a parity bit is attached at the end of a binary stream for checking whether there is an error bit in the binary stream or not. Write a C program to input a pair of variables *parity* and *length*, where *parity* is a bit with value 0 denoting even parity and 1 denoting odd parity; *length* is the number of bits in the binary stream including the parity bit; and then use random number generator to generate a data sequence of □*length*/8.0□elements and each data element is of value between 0 and 255 (8-bit data). The stream starts from the most significant bit of the first byte and the extra bits at the end are padded by 0's. Assume the stream length is at most 2000 bits (including parity bit). Count the number of 1 bit in the input stream and check whether the random generated stream passes parity check, or not. Repeat the parity check until the input *length* is 0. In the program, output (a) the byte count and the bit count of the final byte in the data stream, (b) random

data sequence generated in integers, maximum 20 integers in a line, (c) the data stream in hexadecimal numerals, maximum 20 numerals in a line, (d) the data stream in binary numerals, maximum 8 numerals in a line, and (e) parity count and the result of parity check. Program execution examples:

```
Enter parity and stream length: 0 69
0 69
Input binary streams (9 bytes):

The binary stream is:
00011011 01100000 00100100 00001110 11011111 00101101 10101110 11011110
01101

**** Even Parity.    The binary stream has 36 ones. Parity check passes.

--------------------------------------------------
Enter parity and stream length: 1 124
1 124
Input binary streams (16 bytes):

The binary stream is:
11100101 11001110 11101011 00111001 11001111 00111001 01101001 11000011
01100110 00010101 00110101 01111110 00010101 10101101 10111100 0011

**** Odd Parity.    The binary stream has 70 ones. Parity check fails.

--------------------------------------------------
Enter parity and stream length: 0 0
0 0

--------------------------------
```

3. Write a C program to simulate the combinational logic design of a 32-bit binary adder. A 32-bit binary adder is a logic circuit of 32 one-bit full adders. When adding two 32-bit positive integers, $S=X+Y$, a full adder takes x_i, y_i, and c_{in} as input and produces s_i and c_{out}, where x_i, y_i, and s_i are the i-th bit of X, Y, and S, respectively, and c_{in} and c_{out} are the input and output carry bit, respectively. The logic formula of a full adder is defined:

$$s_i = (x_i \oplus y_i) \oplus c_{in}$$
$$c_{out} = (x_i \wedge y_i) \vee (c_{in} \wedge (x_i \oplus y_i))$$

Refer to Digital System Design Lecture 12, Combinational Logic Design Binary Adder-Subtractor (binary_adder_subtractor.pdf) for more details of the logic design of binary adder. The program will repeatedly input two 32-bit unsigned integers X and Y, and use a binary adder to compute S=X+Y, until both X and Y are 0's. **Do not** use addition operation in C programming language. The output will print X, Y, and S in both decimal and binary format. Also, print a message to confirm that the binary adder has the same result as the addition operation of C programming language. If the addition results in the overflow situation, print an overflow message. A sample output is shown as below:

```
D:\>binary_adder
Enter two non-negative integers between 0 and 4294967295: 656272 2344
X = 656272      Binary value: 0000 0000 0000 1010 0000 0011 1001 0000
Y = 2344        Binary value: 0000 0000 0000 0000 0000 1001 0010 1000
S = 658616      Binary value: 0000 0000 0000 1010 0000 1100 1011 1000
Correct! Adder operation test: 656272 + 2344 = 658616
--------------------------------------------------
Enter two non-negative integers between 0 and 4294967295: 90022 34728372
X = 90022       Binary value: 0000 0000 0000 0001 0101 1111 1010 0110
Y = 34728372    Binary value: 0000 0010 0001 0001 1110 1001 1011 0100
S = 34818394    Binary value: 0000 0010 0001 0011 0100 1001 0101 1010
Correct! Adder operation test: 90022 + 34728372 = 34818394
--------------------------------------------------
Enter two non-negative integers between 0 and 4294967295: 2737281309 3002782671
X = 2737281309  Binary value: 1010 0011 0010 0111 1001 1001 0001 1101
Y = 3002782671  Binary value: 1011 0010 1111 1010 1101 0011 1100 1111
S = 1445096684  Binary value: 0101 0110 0010 0010 0110 1100 1110 1100
Correct! Adder operation test: 2737281309 + 3002782671 = 1445096684
**** The addition operation is overflow.
--------------------------------------------------
Enter two non-negative integers between 0 and 4294967295: 0 0
```