CSCI210 Lab Assignment
Bit Manipulation

- Bitwise operations in C
- Working with packed data

Please type your answers out for these problems. Use the following operators to denote your operations:

- Logical shift right: **>>** followed by an integer shift count
- Arithmetic (sign propagation) shift right: **>>>**
- Shift left: **<<**
- AND: &
- OR: |
- NOT: ~
- XOR: ^

Assume the following 16 bit variables and values:

a. X = 0xFC12
b. Y = 0xCAFE

1. Show the mask and logical operation used to extract the LO three bits from these variables? Show the mask and results in hex

2. Show the mask and logical operation used to clear the HO bit from these variables? Masks should be shifted to the appropriate position. Show the mask and results in hex.

3. Show the mask and logical operations used to set the bit at offset 12 of variable Y? Masks should be shifted to the appropriate position. Show the mask and results in hex.

4. Using the low order bytes of X and Y prove to yourself that the following algorithm performs a swap. Write the resulting hex values after each operation.

```
X = X ^ Y
Y = X ^ Y
X = X ^ Y
```

5. An IPv4 address consists of 32 bits separated into 4 groupings of 8 bits called octets. Here is an example in decimal **192.168.5.130**. Obviously decimal is presented for human legibility. The numbers are ultimately bit strings. This logical address consists of two parts: The **host address** and the **network prefix.** All hosts on a network have the same network prefix (the routing prefix) and this portion of the IP address consists of a collection of the most significant bits of the octets. The **host address** and the **network prefix** can be acquired from the IP address using bitwise masking. The number of bits that comprise the network prefix are often shown in the following format **192.168.5.130/24** (the value following the slash indicates the number of bits that form the prefix). The remaining least significant bits form the host address.

   1) Create two masks for the following IP addresses. One to extract the network prefix and one to extract the host identifier. List the masks and the extracted values in decimal.

   2) 192.168.5.130/24

   3) 192.168.2.4/26

6. Open the file ***google_question.c*** and follow in the instructions contained in the comments. Show your work as additional comments.

7. **XOR Encryption:** An interesting factor of bitwise XOR is that a value can be encrypted by XORing with a **key,** and the encryption can be reversed by XORing with the same key.

   Example:

   ```
   Encrypt byte:  0xCA
   Key:           0xFF

   Binary:   1100-1010 0xCA
   Key:      1111-1111 0xFF
   XOR:      0011-0101 0x35

   Reverse:  0011-0101 0x35
   Key:      1111-1111 0xFF
   XOR:      1100-1010 0xCA
   ```

   Your task is to encrypt an image file by applying a byte-wise XOR with a key. You should be able to encrypt the image and decrypt the image back to the original.

C Functions to research:

fopen	opens a file and returns a pointer to a FILE struct you may need to specify the **wb+** modes
fclose	closes a file
fgetc	gets a char from a file
fputc	puts a char in a file
feof	will report if at end of file

Program Operation:

The program should accept two command line arguments

argv[1]: name of file to encrypt
argv[2]: name of encrypted file

```
gcc encrypt.c -o file_encrypter
./file_encrypter charles_babbage.jpg scrambled.jpg
```

Hex Dump of Original

```
00000000: ffd8 ffe0 0010 4a46 4946 0001 0102 0048  ......JFIF.....H
00000010: 0048 0000 ffdb 0043 0001 0101 0101 0101  .H.....C........
00000020: 0101 0101 0101 0202 0302 0202 0202 0403  ................
00000030: 0302 0305 0405 0505 0404 0405 0607 0605  ................
00000040: 0507 0604 0406 0906 0708 0808 0808 0506  ................
00000050: 090a 0908 0a07 0808 08ff c000 0b08 0320  ...............
00000060: 02a3 0101 1100 ffc4 001f 0000 0007 0101  ................
```

Hex Dump of Encrypted

```
00000000: 0027 001f ffef b5b9 b6b9 fffe fefd ffb7  .'..............
00000010: ffb7 ffff 0024 ffbc fffe fefe fefe fefe  .....$..........
00000020: fefe fefe fefe fdfd fcfd fdfd fdfd fbfc  ................
00000030: fcfd fcfa fbfa fafa fbfb fbfa f9f8 f9fa  ................
00000040: faf8 f9fb fbf9 f6f9 f8f7 f7f7 f7f7 faf9  ................
00000050: f6f5 f6f7 f5f8 f7f7 f700 3fff f4f7 fcdf  ..........?.....
00000060: fd5c fefe eeff 003b ffe0 ffff fff8 fefe  .\.....;........
```

The XXD command line utility was used to generate this dump but you can also view in a hex editor

8. Open the document ***FAT Whitepaper*** (“Data Sheets” module on Brightspace) and read the section pertaining to Date and Time formats on page 25. When you have completed that, parse the following 16 bit date and show your work: **0x5046**

9. Still working with the ***FAT Whitepaper***, read the sections on directory time structures. When you have completed that parse the following 16 bit time and show your work: **0x49F9**

10. Write a C program that contains the following:

a) a function called **parse_date_time** that accepts an unsigned int representing a packed date and time. The **date** is in the **high-order 16 bits** in MS-DOS format. The time is in the **low-order 16 bits** in MS-DOS format

   a. **Example:** The date/times ***9/18/2019, 9:15:16*** would be passed into the function as: **0x4F3249E8** or **1328695784_{10}**

      Print the date and time in US format using printf.

b) A function called **encode_date** that accepts ***month, day, year*** as unsigned ints (or whatever data type you find appropriate) and packs the 3 values into one unsigned int (or short, you choose) using the MS DOS encoding scheme. Return the encoded value

c) A function called **encode_time** that accepts ***hours, minutes, seconds*** as unsigned ints (or whatever data type you find appropriate) and packs the 3 values into one unsigned int (or short, you choose) using the MS DOS encoding scheme. Return the encoded value

d) https://en.wikipedia.org/wiki/C_data_types (There are more)

e) A function called **encode_date_time** that accepts ***month, day, year, hours, minutes, seconds*** and uses the functions from above to create a 32 bit packed date/time value with the **date** in the **high-order 16 bits** and the time in the **low-order 16 bits.** Both in MS-DOS format.

f) Create some test cases and ensure you can encode and parse values. This would be a great candidate for some unit testing using assertions: https://www.programmingsimplified.com/c/source-code/assert

Submission: Attach your work document and C files to the assignment item in Brightspace. C code should be well documented