COMP2401—Tutorial 3
Functions and bit manipulation

Learning Objectives

After this tutorial, you will be able to:

- Write functions in C that manipulate bits
- Gain insight into bit operations and binary representations.

Note:

For operations on bits refer to Chapter 2.2 “Bitwise Operations”.

1 Tutorial

1.1 Extract files

Download tutorial_3.tar.gz file. The file was compressed using the Linux gzip utility program. To uncompress the file use the program gunzip. Use “man gzip” and “man gunzip” to read more about file compression in Linux. Extract the files by:

a. Uncompressing the file using the “gunzip” program
b. Extracting the files using the “tar” program

1.2 Coding exercises

1. Determine whether a bit is set to 1

Write a function that accepts as input an unsigned char, and an integer bitNum (range 0-7) and using the “shift” and “and” operators returns a 1 if bit bitNum is set (bit bitNum is 1) and 0 otherwise.

For example, if the bit sequence of a char is 00010100 and bitNum == 4 then the answer should return 1.

Function prototype

int isBitSet(unsigned char c, int bitNum);

Instructions

Steps:

1. Creating a mask – in order to accomplish this you need to expose the required bit. This is done using the “left shift” operator on the number 1 (the integer 1 has a bit representation of 00000001, which is a 1 in bit 0). For example, to create a mask that exposes the bit at position 1 (the second bit from the right) one case use 1<<1 will create the bit sequence 00000010.

2. Using the “and” operator between the mask and the char c will create a true/false statement.
3. Return a 1 if the answer to step 2 is true otherwise return a 1

2. Setting a bit to 1

Write a function that accepts as input an unsigned char, and an integer bitNum (range 0-7) and using the “shift” and “or” operators sets the bit at position bitNum to 1. The function returns modified character.

For example, if the bit sequence of a char is 00010100 and bitNum == 5 then the answer should return 00110100.

Function prototype

unsigned char setBit(unsigned char c, int bitNum);

Instructions:
This can be accomplished in a similar way to exercise 2 above

3. Clearing a bit

Write a function that accepts as input an unsigned char, and an integer bitNum (range 0-7) and using the “shift”, “and” and “not” operators clears the bit at position bitNum (namely, sets the bit at position bitNum to 0). The function returns modified character. Note, that no assumption is made whether the bit at position bitNum is set to 1.

For example, if the bit sequence of a char is 00010100 and bitNum == 4 then the answer should return 00000100.

unsigned char clearBit(unsigned char c, int bitNum);

4. Printing bits iteratively

Write a function that accepts as input an unsigned char and prints the bits of the character. The function should print the bits iteratively (namely using a “for loop”). The function needs to print the bits that the MSB (bit 7) is printed first.

Use the following statements when printing:

1. To print a 0 use - printf(“0”);
2. To print a 1 use - printf(“1”);
3. To go the next line use - printf(“\n”); Note that you can combine the new line with any other printing statement e.g., to print a 0 on the screen and skip to the next line use printf(“0\n”);

For example if the c is ‘A’ then the function output should be

01000001

Function prototype

void printBitsIterative(unsigned char c);

Instructions:
Here the function will need to iterate on all the bits. There are several ways of accomplishing it. Below is one way of doing it.
Option 1:
Here you will use the isBitSet() function that you coded.

1. Loop (using a for loop) on all the possible bits (bits 0-7) and check each bit whether it is set
    a. If the bit is set, then print a 1 otherwise print a 0

5. Printing bits recursively

Write a recursive function that accepts as input an unsigned char and using the functions above prints the bits of the character. In this printing scheme leading 0's are not printed. Namely if c == 4 then the function will print 100.

For example if the c is 'A' then the function output should be

1000001

Function prototype

void printBitsRecursive(unsigned char c);

pseudo code:

// base case – print the write most bit if c is either 0 or 1

// if c is either 0 or 1 then print 0 or 1 depending on the state of the least significant bit (LSB)

// recursive step

// call the function recursively while using right shift by 1 on c

// print the LSB of C (either 0 or 1)

// note that here the "work of the recursive step" is carried out after the recursive step

1.3 submission

create a tar file t3.tar include the files main.c bit_functions.c and bit_functions.h and submit it on cuLearn

End of tutorial section for submission

2 Additional Exercises

Below are additional exercises that you can do to make sure that you understand the concept of bit manipulations.
You do not need to submit those

2.1 On paper

2.1.1 Binary representation

1. Binary to octal and hexadecimal representations.
   Compute the value of the following. Once when the numbers are given in 1's complement and once when they are given in 2's complement.
   a. 10000101 + 11111110
   b. 11111101 +01110100

2. Floating point
   You decided to represent a real number using a floating point convention as follows:
   Bit 7 – the sign bit
   Bits 4-6 – is the exponent part. Here you will be using **Excess-4** to represent the values of the exponent.
   Bits 0-3 represent the mantissa

   Write the bit pattern of
   a. 0.5
   b. 2.375
   c. -0.25

2.1.2 Bit operations

1. Solve the following cases on a sheet of paper. Then verify your results using the printf statement (printf("%x \n",cc);). For each case that your result differs from the program output, try to understand why there is a discrepancy.

   a. cc = 1;
      cc = ~cc;
   b. cc = 5;
      cc |= 0x0f;
   c. cc = 5;
      cc &=0x0f;
   d. cc = -32;
      cc >> 2; // pay attention to the carry of the sign bit.
   e. cc = -32;
      cc << 1;
   f. cc = (1 << 3);
   g. cc = ~(1 << 3);
   h. cc = 0xff;
      cc &= ~(1 << 3);
   i. cc = ~1 << 3;

j. cc =5 ^ 0x0f;
k. cc = 9 ^ 0x0f;
   cc ^= 0x0f;
l. cc = 32;
   cc = cc >> 1;
m. cc = 32;
   cc = cc >> 3;
n. cc = -32;
   cc = cc << 1;

2.2 Coding exercises

Parity bit

A parity bit is a bit that is used to detect whether a given sequence of bytes was tempered with. A byte is tempered when one or more of its bits were changed (either on purpose or accidently). For example, assume that the variable c is declared as char and that its initial value is 0x7B. If bit 4 was changed (in this case from a 1 to 0) then the value of c has changed to 0x6B. Note that if c represented the money in your bank account then your account would have $107 instead of $123.

An **even parity bit** is set to either 0 or 1, depending on the number of bits that are set to 1 in the sequence of bytes. If the number of bits that are set to 1 is odd then the parity bit is set to 1 otherwise it is set to 0.

For example: here we use bit # 7 as the parity bit. If the char c has the value 0x5D (see table below). In this case the number of bits that are set to 1 among bits 0-6 is 5. Therefore, bit 7 will be set 1. This will ensure that the total number of bits that are set to one in the character (bits 0-7) is even.

Data	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
bit #	7	6	5	4	3	2	1	0

In this case the number of bits that are set to 1 among bits 0-6 is 5. Therefore, bit 7 will be set 1. This will ensure that the total number of bits that are set to one in the character (bits 0-7) is even.

Data	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
bit #	7	6	5	4	3	2	1	0

Write two functions that compute the parity bit of a char that is stored in bit 7. Note in this case one assumes that the data is stored in bits 0-6. Use the function(s) that you created the first part of the tutorial

1 Write a function counts the number of bits that are set to 1 in a char. The function will accept as input a char c and will return the number of bits that are set to one. Since bit 7 is the parity bit the function should ensure that bit 7 is set to 0.

```
int countBits(char c);
```

2 Write a function that computes the parity of bit of a char. The parity bit is at position 7. In doing so use the function countBits().

Use the return value from countBits() to determine whether the parity bit needs to be set to 1. Here you can either use the % operator (mod).

Modify the parity bit of c.

function prototype

void setParityBit(**char *c**);

3 Test your function on the characters “A” and “B”