CSE/ECE 474 C-Programming Assignment 1: C Programming with bit manipulation and structs

Updated: April 10, 2023

In this assignment we will learn how to inspect and modify individual bits in memory. Remember that everything in computers is represented by just a series of 0's and 1's. This means that the software has to remember how each part of memory is converted from 0's and 1's to something else. Here's a simple example: Suppose the memory location 0xAFF5C3 contains the following:
0xAFF5C3 →

0	0	1	0	1	0	1	0

We will focus on the exact bits in memory and the powerful (but yes, low-level) C-functions which allow you to work with them. This 8-bit segment of memory could be a `uint8_t` type (an 8-bit integer between 0-255). In that case, 00101010 equates to 2+8+32 = 42. However, if it is a `char` (character), the same pattern of 00101010 equates to '*' (the asterisk character).

Working with bits is important because hardware control bits need to be set or cleared in order to control a computer chips. You could turn on or off on-chip peripheral devices to save battery life for example.

Instructions

We have provided template C files, **`c_prog2.c`** and **`c_prog2.h`** which contain comments indicating where to put each part of your code as well as directions and sample outputs. **These are the two files you will turn in on canvas.** In addition to this we have provided the file **`c_prog2_arduino.ino`** contains a set of function calls in the **`setup()`** function that will run your code. A sample output is provided in **`sample_output.txt`**.

Please read the entire file before starting work so you understand where things go. Please do not modify the test code.

Turning in the assignment: Upload `c_prog2.c` and `c_prog2.h` on Canvas

Running your code. This code is designed to run in the Arduino IDE which is available to download here: https://www.arduino.cc/en/software
Clicking the "Upload" icon will compile and upload your code to the Arduino board. The output will be displayed in the Serial monitor (Tools > Serial Monitor). The code will run once and will be repeated if you press the RESET button on the board.

For those with an existing C development environment you are welcome to use other tools as well, however we will not go over this setup in class and *will evaluate the code using the Arduino environment described above*. To do this you will need to write a new **`main.c`** file with the testing code in **`setup()`** and redefine the printing functions in **`c_prog2_arduino.ino`** to use **`printf`**.

1 C Bitwise Operations

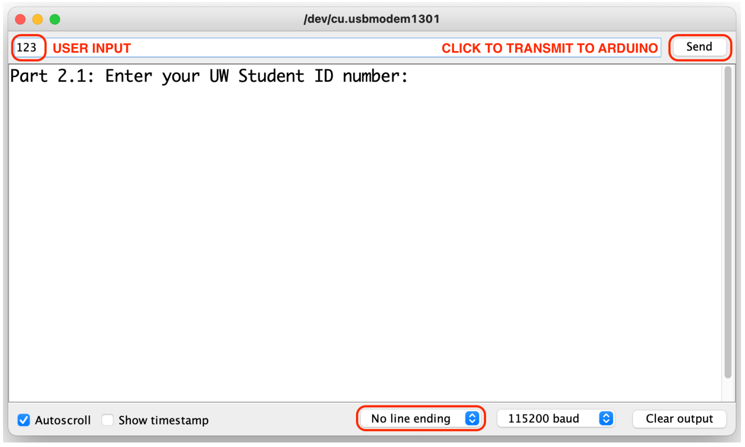
1.1 Bitwise operators

We are going to take a number (such as your UW student ID) and mangle it beyond recognition. Write the function **`long mangle(long SID)`** which performs the following three steps on the input:

1. Right shift SID by 2 bit positions
2. Clear bit 6 (after the shift, counting from the LSB=0)
3. Complement bit 3 (e.g. 00001000). If bit 3 is one, make it 0. If bit 3 is 0, make it 1.

```
OUTPUT EXAMPLE:
Part 2.1: Enter your UW Student ID number:
 You entered 51218
 Your mangled SID is 12812
```

User input on Arduino. The testing code will prompt you to enter a value for your SID to test. To do this, type a value into the Serial monitor on Arduino and click "Send". This will send the value from your computer to the Arduino's processor. Note that there are options for formatting the way your output is sent such as automatically adding a newline character to the end. Select "no line endings" as the testing code is not designed to handle these extra characters.



```
Part 2.1: Enter your UW Student ID number:
```

1.2 More bit manipulation

Write the function **`int bit check(int data, int bits_on, int bits_off)`**, to check an int to see if a specific set of bits (aka a "bit mask", called bits_on) is SET, AND that another set of bits (bits_off) is CLEAR. Returns 1 if the int called data matches the bit masks, and 0 if not. However, note there is a special case. For example, suppose I write:

```
int data = 0xFF; // 0b11111111
bit_check(data, 0x32, 0x20); //0b11111111, 0b00110010, 0b00100000
```

There's a problem! Because 0x32 & 0x20 = 0x20 , we are asking that bit 0x20 must be BOTH off and on! Your function should return -1 for any case of testing that any bit is BOTH off and on.

2 Pointer declaration and usage

2.1 Basic Pointer Declaration and use

In this part we will use pointers to designate capital letters of the alphabet which are put into memory through the pre-defined array:

```
char a_array[] = {'A','B','C','D','E','F','G','H','I','J','K',
'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
```

Write a function **`char* pmatch(char c)`** which will take a character and return a pointer to the element in **`a_array`** which matches it. If there is no such character (e.g. pmatch('m')), then return the NULL pointer.

```
Part 2.1: Enter a capital letter: J
 You entered: J
```

2.2 Pointer Arithmetic

You can meaningfully do certain kinds of math with pointers (but e.g. x = sqrt(ptr) is meaningless). Write the function **`char nlet(char* ptr)`** where ptr is a pointer returned by **`pmatch`**. This function will return the next letter of the alphabet (NOT a pointer to the next letter of the alphabet) unless the pointer points to 'Z'. In that case it will return -1. If the argument does not point to a capital letter A-Y, return -1.

```
 Part 2.2: The next letter after J is K
 Part 2.2: The next letter after Z is -1
```

2.3 More Pointer Arithmetic

Write the function **`int ldif(char c1, char c2)`** to find the alphabet distance between two letters of the alphabet using pointer arithmetic. Include **`c2`** in the count but not **`c1`**. For example: **`ldiff('A','E') == 4`** is true and **`ldiff('E','A') == -4`** is true. Check for errors. If either letter is not a capital letter, return a negative number less than −26.

```
 Part 2.3: M and Q are 4 positions apart
 Part 2.3: x and Q are -999 positions apart
```

3 Working with Structs

3.1 Defining a struct

You might have used classes in a previous programming course/language. In C we don't have classes but structs provide similar functionality to group data fields together but do not include methods. Here you are going to set up a struct to represent some data about a person, and a couple of functions which work with pointers to those structs. Modify the placeholder definition of Person in **`c_prog2.h`** to include the following fields:

1) A max 20 char string: FirstName
2) A max 30 char string: LastName
3) A max 80 char string: StreetAddr
4) A max 6 char string: ZipCode
5) A double: Height // height in meters
6) A float: Weight // weight in kg
7) A long int: DBirth // birthday (days since 1-Jan-1900) * /

3.2 Size of a struct

Write a function **`int personSize(Person p)`** that returns the number of bytes required to store the person struct in memory. What do you think the number will be? What happens after we store values into it, does the size change? HINT: You can use built in library functions for this.

3.3 Displaying a Person

Write a function **`char* per_print(person * p)`** which returns a formatted string to neatly print out all the data about a Person. The street address can store 80 characters (see above), if the person's address is longer than 60 characters, only print the first 60. Do not erase memory of the last 20 characters if present.

You can use functions from the standard string libraries. For example, the **`sprintf`** function can help create formatted strings with a mix of words and numbers (see the testing code in **`c_prog2_arduino.ino`** for examples). The function strcat can also be useful for concatenating strings together.

Also some microcontrollers do not have hardware support for floating point operations. You'll notice that Arduino does not support certain standard C functions like the ability to use '%f' to print floating point numbers with **`sprintf`**. Define the helper function **`floats_to_ints(float f, int* output)`** that takes in a floating point number and converts it to 2 integers in the int array output. You may find built in math functions such as round helpful.

```
OUTPUT EXAMPLE:
 --- person report: ----
 First Name:       Blake
 Last Name:        Hannaford
 Address:          124 N. Anystreet / Busytown, WA
 Zip:              99499
 Height:           1.97
 Weight:           81.81
 DOB 1/1/1900:     34780
 -----------------------
```