In this assignment, you will write three programs that involve string arrays. You will get used to comparing string characters as well as iterating through strings. You will also gain a thorough understanding of two’s compliment bit storage and integer addition.

(1) Morse Code

Write a program called `morseCode.c` that asks the user for an English sentence and then displays the sentence in Morse code. Each letter should be replaced by the code shown on the right (i.e., using dash and dot characters). You will have to convert any lowercase characters to upper case. Also, if any non-alphabetic (and non-space) characters are entered, they are to be represented simply by this string: "?". You must leave a space character between each letter and you should leave 3 spaces between each word.

Here are a couple of sample runs of the program:

```
Enter the sentence to be converted:
I LOVE C
Here is your sentence in Morse Code:
..   .-.-.-- ....- ...

Enter the sentence to be converted:
This sentence has 2 #'s and 2 symbols
Here is your sentence in Morse Code:
- .... . .-.-. - .... .- .- -. ... ... ? ? ? ... .-

-.-. . . ? ... -.-. -- =-= =- =- =- =- =- =- ...
```

(2) Reverse Morse Code

Write a program called `reverseMorseCode.c` that takes in a morse code string sequence from the user (as produced from the program above) and converts it back into the original ASCII sentence as follows:

```
Enter the Morse Code sequence:
..   .-.-.-- ....- ...

Here is the decoded message:
I LOVE C
```

Note that the spacing between the words in the final output is arbitrary and may vary according to whatever you want, as long as the words come out properly. The ‘?’ characters should stay as ‘?’ character in the output.
(3) Binary Math

Write a program called `bitMath.c` that asks the user for two 8-bit sequences that correspond to integers stored in the two's compliment representation. It should then add the bit sequences together to produce a resulting bit sequence that is the sum of the two numbers. The program should display the bit sequences as well as their decimal value with output in the format shown in the following output (assuming the program was run 8 times):

```
student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 00001010
Enter 2nd 8-bit sequence: 00000111

00001010 (10)
00000111 (7)
----------------
00010001 (17)

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 11110110
Enter 2nd 8-bit sequence: 11111001

11110110 (-10)
11111001 (-7)
----------------
11101111 (-17)

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 10000000
Enter 2nd 8-bit sequence: 00000001

10000000 (-128)
00000001 (1)
----------------
10000001 (-127)

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 01110011
Enter 2nd 8-bit sequence: 00111001

01110011 (115)
00111001 (57)
----------------
OVERFLOW DETECTED

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 10000000
Enter 2nd 8-bit sequence: 10000001

10000000 (-128)
10000001 (-127)
----------------
OVERFLOW DETECTED

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 000111
Enter 2nd 8-bit sequence: 1001
Error: You must enter two valid 8-bit string sequences

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 1hh1139
Enter 2nd 8-bit sequence: kd232811
Error: Bit sequence 1 has non-binary characters

student@COMPBase:~ $ ./bitMath
Enter 1st 8-bit sequence: 10011001
Enter 2nd 8-bit sequence: 10k03r43
Error: Bit sequence 2 has non-binary characters

student@COMPBase:~ $ 
```
To write this code, it makes sense to have at least 4 functions:

- a main function,
- a function for flipping bits (for the negative numbers),
- a function for adding two bit sequences, and
- a function for converting a bit sequence to a decimal value.

It would be good to follow the following steps to obtain your solution systematically. Don’t move on until you get each step working:

1. Get the input from the user
2. Ensure that each input sequence is 8 characters long
3. Check to ensure that there are only 1’s and 0’s in the bit sequence
4. Display the input sequences along with their decimal equivalents (test only positive numbers)
5. Write the function to add the bit sequences and produce the result (test only positive numbers)
6. Write the function to flip the bits of a sequence
7. Adjust your code to work for negative numbers
8. Add the overflow checking
9. Test thoroughly

Test your code thoroughly and hand in the output from your test cases in a file called output.txt. You will be marked according to the thoroughness of your test cases. Think of the tricky test cases and boundary test cases. If you miss some “obvious” boundary test cases, you will lose marks … so test more than you need if you are not certain.

IMPORTANT SUBMISSION INSTRUCTIONS:

Submit all of your c source code files as a single tar file containing:

1. A Readme text file containing
   - your name and studentNumber
   - a list of source files submitted
   - any specific instructions for compiling and/or running your code
2. All of your .c source files and all other files needed for testing/running your programs.
3. Any output files requested (e.g., output.txt).

The code MUST compile and run on the course VM, which is COMP2404B-W19.

- If your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late … so make sure to submit the assignment WELL BEFORE it is due!

- You WILL lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not written neatly with proper indentation and containing a reasonable number of comments. See course notes for examples of what is proper indentation, writing style and reasonable commenting).