CSCI 222L – Computer Organization Lab
Lab 04 – Integers & Floating Point Numbers

You may complete this lab individually or in pairs. No groups larger than 2 are allowed.

Goals

This lab will help you gain experience using signed and unsigned numeric data types in C. You will also learn how to overcome the limitations of finite sized data types and to evaluate code for security risks resulting from the nuances of using signed and unsigned data types. The best way to learn this is to analyze and write C code.

Setup & Submission

Clone the assignment repository using the link you received via email from gitkeeper. Make sure that all files you create are added to the repository. Make sure to commit all files that were added or modified. Make sure to push your solutions. Check your email for successful submission email.

All submissions are due at the end of the lab period.

Question 1 [7 points]

In the previous lab we presented a C implementation for the `factorial` function that only worked for `n <= 12`. In today's lab you will learn to overcome the limitations of finite size data types by writing a new `factorial` function that will compute `n!` for `n > 12`. The prototype for your new `factorial` function is:

```
unsigned int factorial(unsigned int n, unsigned char
result[]);
```

The result of this function will store `n!` in array `result` and return the number of significant digits in the array. Note that the digits in the array are `unsigned char` and are stored in little endian. For example, to compute the `factorial` of `13` (which is `6,227,020,800`) the new function should return `10` and set `result` the following way:

Value:	0	0	8	0	2	0	7	2	2	6
Index:	0	1	2	3	4	5	6	7	8	9

This function will contain a loop that, during the i^{th} iteration, will compute `i!` based on the invariant property that `(i-1)!` is stored in `result`. This will require you to write an inner loop that will traverse `result` starting at position `0`. At each position, `result[index]`, you will need to multiply the digit by `i` and determine the new value of `result[index]` and the carry over to `result[index+1]`.

Use the given `display_array` function to display the digits in the correct order (i.e., from most to least significant). Test your function by modifying the main function so that it computes `n!` from `0` to `15`. Output the value of `n` and the value of `n!` to standard output. Use the redirection operator in Unix to send the output of `a.out` to a file named `fact.txt` that will contain the output of your program. Make sure this file is added to the repository. Do not add `a.out` to the repository.

Question 2 [7 points]

a. Make a copy of the `factorial.c` program and name it `evaluate.c`. Consider the `display_array` function in `evaluate.c`. Modify the `for` statement so that the loop variable is an `unsigned int` and remove the `int` typecast. Modify the `main` function to call `display_array` with parameters that will cause a runtime error in the modified function. What runtime error do you get? How many times does the for loop execute in your modified code? Why does this runtime error occur? Put your answer in a comment block at the end of the file.
b. Explain your understanding of the operation of this `for` statement to leave no doubt you understand the need for the `int` loop variable and the `int` typecast. Put your answer in a comment block at the end of the file.

Take a few moments to study the source code in file `copy_safe.c` (provided with your lab instructions). When you compile this code you will get a warning message. Read the warning message and execute the code anyway. When you run `copy_safe.c` you should get:

```
src is a buffer of size 46
dst is a buffer of size 10
Copy successful!
```

When you feel confident with the code, answer the items below. Put your answers in a comment block at the end of the file.

c. Do you think there is something wrong with the code (yes/no)?

d. If you answered yes to the previous question, describe what is wrong with the code and how would you correct the code.

Question 3 [6 points]

Write a function called `show_float` that takes a `float` as argument and displays its *sign bit*, the *exponent,* and *mantissa* bits. A float is stored in IEEE 754 format. The signed bit is the leftmost bit (bit 0). The exponent is stored in bits 1-8. The mantissa is stored in the remaining 23 bits. Reference your textbook for additional information on IEEE 754 formats.

Test your function using the following examples. Redirect the output of your program to a file named `float.txt`. Make sure to add this file to your repository.

```
show_float(1/3)          >> 0 01111101 01010101010101010101011
show_float(3.75)         >> 0 10000000 11100000000000000000000
show_float(-3.75)        >> 1 10000000 11100000000000000000000
show_float(M_PI)         >> 0 10000000 10010010000111111011011
show_float(NAN)          >> 0 11111111 10000000000000000000000
show_float(INFINITY)     >> 0 11111111 00000000000000000000000
show_float(-INFINITY)    >> 1 11111111 00000000000000000000000
show_float(0)            >> 0 00000000 00000000000000000000000
show_float(-0)           >> 0 00000000 00000000000000000000000
```

Question 4 [5 points]

Write a C program in file `compute.c` that will compute the binary fields of a floating point number, as in Question 3, and will convert the binary representation of the floating point numbers to their equivalent floating point values. Test your program on 4 various floating point values. Output the results to file `compute.txt` by redirecting standard output.

After you have completed the lab, make sure to add all files you created to the repository. Do not add any a.out files, or files with ~ at the end of their filename. Make sure to commit all files that were added or modified during the lab. Make sure to push your solutions. Check your email for a successful submission message.