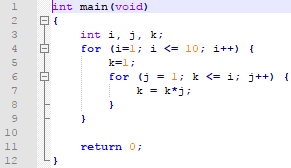
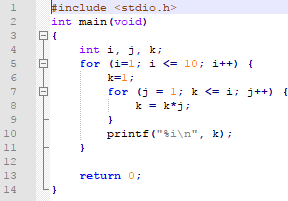
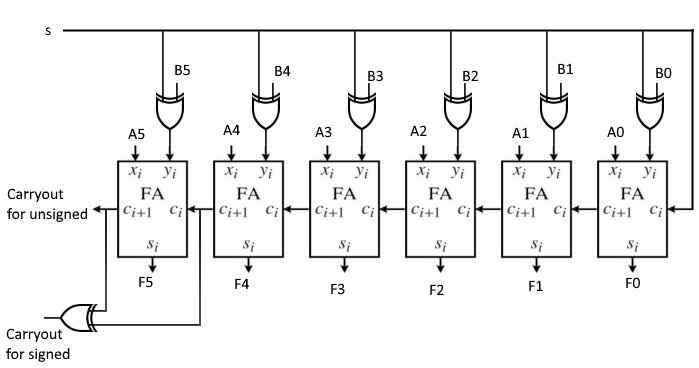
## Question #1



1. This program first establishes the int variables *i*, *j*, & *k*. Then it uses a for() loop which will loop a total of ten times. Each time this loop triggers, it sets *k* to 1 and then starts another loop which loops until *k* is larger than *i*. Every time this inner loop goes, *k* is multiplied by *j*. So for example, for the first loop, it will find that *i* equals 1. Thus, the outer loop will succeed since 1 is less than 10, setting *k* = 1. Then it initiates the inner loop, which will loop twice because when *j* = 1 then “*k*=*k\*j”* = 1, then j increments so that “*k*=*k\*j”* = 2, breaking the loop since *k* is greater than *i* (which is still 1 at this point). From here the loop will continue to execute like this, running the inner loop until *i* reaches a number larger than 10.
2. To print out the *k*-values all we have to do is add a printf() statement after the 2nd for loop:  
     
   This means we also have to include the package “stdio.h”, otherwise printf() won’t work.
3. *k* = 2, *k* = 6,  
   *k* = 6, *k* = 6.

## Question #2



This is a 6–bit adder–subtractor combinational circuit I recreated using the 1**–**bit adder in the HW4.pdf, and an XOR gate I took from Google. It can add up to 5 signed bits or 6 unsigned bits.

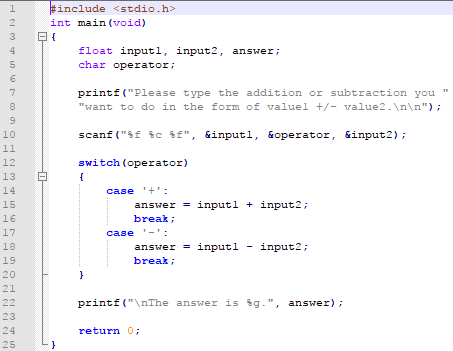
If we want to do addition using this circuit, then we set *s*=0. This will take the *B* inputs as given. Say we want to do the calculation -5+21, the input for -5 will be 111011 (due to 2’s complement) and 21 will be 010101. Doing the addition:  
 111011  
+ 010101 Ci=0  
------------  
 010000  
Out C5 & Cout will both be 1, so using the carryout for signed (XOR) we get 0. Thus, we have no overflow.

Conversely, if we want to do subtraction such as 7-4, we can do so. 7 in binary is 000111, while 3 is 000100. Since we need to get the negative version of 4, we can use use the XOR gates when s=1 to get 111011. From here, we do normal addition, considering a carry in of 1 from s=1:

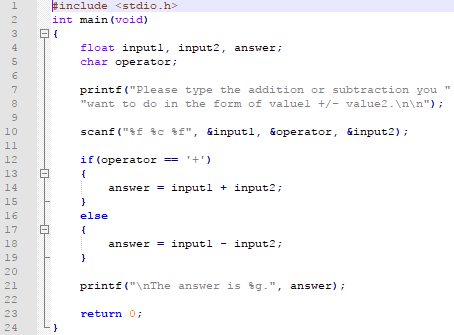
000111  
+ 111011 Ci=1  
------------  
= 000011  
Our C5 & Cout will both be 1, so once again we will have no overflow.

As for how this device works, it is no different than normal addition. Assuming s=0, then the 1–bit adders simply take the raw inputs that we give it and adds them together. The first 1–bit adder adds the first bit of each number, carries out to the second if needed, and outputs the answer. The second takes in the carry out from the first, adds the 2nd bits from both numbers, and so on and so forth until the addition is finished. The subtraction method is quite simple too. If s=1, then the program simply flips the bits of the 2nd number and adds 1 to the calculations, creating the 2’s complement of this number. Essentially, instead of performing the calculation for 7-3, the program computes 7+(-3). From here it runs the calculations as previously mentioned until it gets the answer.

## Question #3



## Question #4



## Question #5

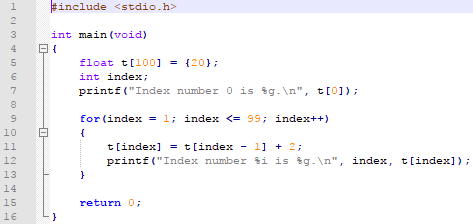
1. The break statement breaks you out of the loop completely, while the continue statement acts as if the loop completed. That is, it will activate the step–variable (assuming the loop is a for() style loop) and then redo the test variable (unless, of course, it is a do–while loop).
2. For the expression !a||b && c||d, the program would first evaluate the logical not operator !, then the logical AND operator &&, and finally the logical OR operators ||.

## Question #6

The mathematical matrix defined by m[3][2] would look as follows:

And the indices for array element 4 would be m[1][1].

Similarly, if we want the array , if we consider *N* to be the size of the array, then the array is 100 elements large, with indices from 0-99.



## Question #7

