CSC 362 Programming Assignments #4 & 5
Due Date: Friday, April 17

These two assignments require that you implement two Intel assembly language programs embedded in C programs. If you are using Windows, you will have to use Visual Studio. If you are using a Mac, you will have to use Xcode. Otherwise, you will have to log in to the NKU virtual lab to access a virtual Windows 10 computer running Visual Studio. In these programs, all instructions MUST be in assembly language accept for any variable declarations, input and output. For program #4, you can also write a while that surrounds your assembly code so that you can do input/process/output for each of your inputs rather than rerunning the program, once per input. See the skeleton code below. Make sure you comment your code well.

Program #4: A perfect number is a number whose factors (not including the number itself) sum up to the number. For example, 6 is a perfect number: its factors are 1, 2 and 3, and 1+2+3 = 6. There are very few perfect numbers (1, 28, 496 and 8128 are the only other perfect numbers less than 100,000). Other numbers might be close to being perfect. We will consider the *closeness* that a number is to being perfect by using the following formula: **| (sum of factors – number) | / number**. For instance, 12 has factors 1, 2, 3, 4, 6 which sum up to 16 so that 12 has a closeness of |(16–12)|/12 = 33% of a perfect number. assembly program, embedded in C code, to compute how close an input int value comes to being perfect. Your program is to output the input value, the sum of its factors, and the percentage of closeness. All program input, output, and variable declarations should be written in C code. You may also initialize variables in C. All remaining code (loops, arithmetic statements, selection statements) **must** be done in Intel assembly language. You *may* place a while loop in your C code to repeat the process of initialization-input-assembly code-output so that you can run your program one time and handle all of the inputs (rather than having to rerun your program multiple times). Your program might look like this:

```
void main( ) {
        // variable declarations in C
        // input the value to be tested in C
        while (//input > 1)  {    //  assume all input values will be 2 or greater
                // initialize any needed variables here in C or in assembly, your choice
                _ _ asm{        // compiler directive to enter assembly language code
                                // assembly code here to compute the sum of the factors
                                // assembly code here to compute the percentage of how closely
                                //      the input number comes to being a perfect number
                                // store the resulting percentage in a variable as declared in C above
                        }
        // output statements in C for the input number, sum of factors, and percentage
        // get next input
        }
}
```

As the Intel assembly code does not include float/double computations, you will have to compute | (sum of factors – number) | as an int and multiply it by 100 and then divide by number to obtain a fractional value as an integer. For instance, instead of 0.25, you would get 25 which you can output as 25%. Also note that there is no absolute value operation in Intel assembly language. If you just do sum – number, you may obtain the wrong value if sum < number, so before doing the subtraction, determine which of the two is greater. Alternatively, you can do the subtraction and then if the result is negative, use the NEG command to negate it.

Test your program on 12 (sum of 16, 33% from being a perfect number), 50 (sum of 43, 14% from being a perfect number), 88 (sum of 92, 4% from being a perfect number), 28 (sum of 28, 0% from being a perfect number). Then, run your program on the following inputs. Submit your source code and the outputs from these inputs: 4, 32, 50, 51, 64, 81, 500, 8128, 90000

Program #5: Implement the Insertion Sort algorithm for an array of int values in Intel assembly language. The algorithm, in C, is given below, assuming **int a[ ], n;** where n is the number of elements stored in the array.

```
for(i=1;i<n;i++)
{
        temp=a[i];
        location=i-1;
        while(location>=0&&a[location]>temp) {
                a[location+1]=a[location];
                location--;
        }
        a[location+1]=temp;
}
```

Implementing this in Intel assembly language can be tricky because you are limited to the 4 data registers and you might need to be accessing other values as well. This may require that you move values back and forth between variables and registers. Write the program in C first to make sure you understand the logic, and then either convert the instructions into assembly instructions or write it from scratch using your C code as a basis.

Your program will be organized as follows:

- initialize the int array and the number of elements of the array in C code (you can either hardcode the array with the initial list of values, or input the list from the keyboard)
- enter assembly code to perform the sort
    - all logic for the sorting algorithm must be done in assembly code
- exit assembly code and output the sorted array (use a for loop in C to iterate through the array for output purposes)

Use only one array (do not manipulate a copy of the array). Run the program on the 2 data sets below. Remember to comment your code, especially with respect to how it helps accomplish sorting. You can either use an outer loop in C to iterate through the two data sets, or write the program for data set 1, compile and run it, change the program for data set 2, compile and run it. This program should be contained in one file and use only one function (main).

Run #1: 100, 99, 97, 95, 90, 87, 86, 83, 81, 77, 74, 69, 63, 50, 44, 43, 39, 31, 29, 12
Run #2: 123456, 342100, 87539, 606006, 443322, 198371, 99109, 88018, 707007

Hand in your source code and the output from the two runs.