Assignment #6 – Functions **CS 218 Fall 2020**

Purpose: This program will explore creating and using functions in assembly.

Using a random number generator, create a list of random values. Ask the user how many values to generate (minimum 2, maximum 10,000) and their range (0 to minimum 1, maximum 100,000). Then sort the list of values using the comb sort algorithm (described below) and output the sorted values to the terminal.

This program will be written from scratch. Feel free to modify your code from previous assignments for the conversion functions. Add error checking as appropriate.

Your program must include the following four functions but you may define additional ones as needed.

Function 1: Random number generator using a linear congruential generator (lcg). The LCG will form the core of the random number generator by providing values between 1 and a prime value m in a random order. This value will be further refined by dividing by the desired range.

To obtain the next value of the LCG, multiply the previous value by the constant **a** and find the remainder of dividing by the prime value **m**. The starting value is known as the seed, which we will set to 1. Using the same seed will always result in the same sequence of values being generated.

$$lcg_n = (lcg_{n-1} * a) \bmod m$$
$$lcg_0 = seed$$

$$a = 48271$$
$$m = 2^{31} - 1$$
$$seed = 1$$

Use global constants for **a** and **m**. The previous lcg value will be passed to the function as a by-reference argument along with the range as an unsigned dword integer.

Once the next lcg value is created, use the provided range to generate the random number desired. Make sure to update the lcg value or it will keep generating the same number.

$$(lcg(n+1) \bmod (range + 1))$$

For more information see:
https://en.wikipedia.org/wiki/Linear_congruential_generator

Function 2: Comb Sort. The comb sort is a variation of a bubble sort where the comparisons are done over a gap rather than comparing neighboring elements. This allows the smallest and largest values to transition nearer to their correct positions using less swaps.

For an array of n items, start with a gapsize of n. Before each pass through the array, multiply the gapsize by 10 then divide by 13 (round down). The multiplication must be done before the division to avoid a loss of precision. Once the gapsize is 1, the algorithm begins to check if a swap is done on a given iteration of the array. If no swaps were performed, the list is sorted and the algorithm ends.

Pseudocode:

```
gapsize = n
loop
{
     gapsize = gapsize * 10 / 13
     if(gapsize == 0) gapsize = 1

     sortsDone = 0
     for ( i = 0, i < n - gapsize, i++ )
     {
          if(array[i] > array[i+gapsize])
          {
               temp = array[i]
               array[i] = array[i+gapsize]
               array[i+gapsize] = temp
               sortsDone++
          }
     }

     if(gapsize == 1 and sortDone == 0) exit loop
}
```

The function should take two arguments, an address to the dword array to sort and the length of dword array passed by value. Write the function to sort the values in ascending order (like the pseudocode above)

For more information see:
https://en.wikipedia.org/wiki/Comb_sort

Function 3
Modify the decimal to integer macro from assignments 4/5 and turn it into a function.

Function 4
Modify the integer to hexadecimal string macro from assignments 4/5 and turn it into a function.

Expected Output

```
Sorted Random Number Generator

Number of values to generate (2-10,000): 100
Maximum Value (1-100,000): 500
Sorted Random Numbers

0x00000001 0x00000007 0x0000000C 0x00000010 0x00000012
0x00000016 0x0000001B 0x0000001C 0x00000022 0x00000023
0x00000027 0x00000027 0x00000034 0x00000035 0x0000003C
0x00000041 0x00000044 0x0000004A 0x0000004D 0x00000050
0x00000051 0x00000051 0x00000055 0x00000062 0x00000064
0x0000006C 0x0000006E 0x00000076 0x00000077 0x00000078
0x0000007A 0x00000081 0x0000008B 0x0000008C 0x00000090
0x00000090 0x0000009B 0x000000A5 0x000000A8 0x000000AF
0x000000B4 0x000000CC 0x000000D0 0x000000D8 0x000000DE
0x000000E4 0x000000EC 0x000000EF 0x000000FB 0x000000FC
0x00000107 0x00000109 0x0000010F 0x00000110 0x00000110
0x00000113 0x00000113 0x00000117 0x0000011A 0x0000012D
0x00000132 0x00000132 0x00000135 0x00000138 0x0000013E
0x00000150 0x00000154 0x00000158 0x00000159 0x0000015E
0x00000170 0x00000173 0x00000173 0x00000177 0x0000018C
0x00000191 0x00000192 0x00000194 0x0000019E 0x000001A3
0x000001A6 0x000001AC 0x000001AC 0x000001B2 0x000001B6
0x000001BB 0x000001BD 0x000001BF 0x000001C7 0x000001D0
0x000001D5 0x000001D6 0x000001D7 0x000001DA 0x000001DF
0x000001E5 0x000001EA 0x000001ED 0x000001EE 0x000001F1
```

Submission
Once you are satisfied with the program, upload the assembly source code (.asm) file to the class website.