ECE 3331, Dr. Hebert, Fall 2023, Program 06, due Sunday 10/01 at 11:59 pm

In this project, you will:

A) Declare a 1000-cell array of integers int vec[1000].

B) Ask the user for the number of pseudo-random integer random variables N<=1000 that they require.

Verify that the user requested less than 1000 pseudorandom numbers. If not, ask the user for a new number N until N<=1000 has been entered.

C) Call a subroutine to fill-in N pseudo-random integer values into the array.

The subroutine will take two arguments: the array name int vec[ ] and the integer number of integer cells N that will be filled with pseudorandom values. See chapter 6 for using arrays as arguments for subroutines.

D) Call a second subroutine to sort the pseudo-random values from smallest to largest.

E) Print the median of the pseudo-random integers to the display.

Background 1.

Note that rand( ) generates values from 0 to RAND_MAX. (#include <stdlib.h>)
RAND_MAX is a macro that, in most installations of C, is typically = 32767.
The macro RAND_MAX is typically located in the header file stdlib.h or in math.h.
Note that 15 bits can only denote 2^15=32768 unique values. The values 0 to 32767 are exactly 32768 unique integers.
Therefore, rand( ) only generates an unsigned 15-bit random integer value 0 to 32767.
We could use rand( ) to generate a **signed** 15-bit random integer value between -16384 and +16383 using
y= using (rand( )-16384) . (-16384 to +16383 is a total of 32768 unique values).
However, an int value is 32-bits and can therefore denote 2^32=4,294,967,296 unique values,
-2,147,483,648 to +2,147,483,647.
A single call to rand( ) cannot generate a pseudorandom 32-bit integer value.
We could use two calls to rand( ) to generate a 30-bit pseudorandom integer value as
int ix=pow(2,15)*rand( ) +rand( );
Here, ix would be an integer value between 0 and 32767* 32768 +32767 = 1,073,741,823.
This is a total of 1,073,741,823 unique values. Note that 2^30=1,073,741,824 as we expect.
We could generate a 30-bit **signed** integer value as
int ix= (pow(2,15)*rand( ) +rand( )) - 536,870,912;
whereby ix would be between -536,870,912 and +-536,870,911.
A pseudorandom integer value (an int variable) requires 2 more pseudorandom bits.
We therefore need a 3rd call to rand( ) to get a 32-bit pseudorandom integer value.
int ix = (rand( )%4)*pow(2,30) +(pow(2,15)*rand( ) +rand( )) - 2,147,483,648;
yields a pseudorandom value between -2,147,483,648 and +2,147,483,647, the exact range of values for a signed int variable. Use this method in this project.

Background 2.

An “in-place sort” will sort a 1-D array of values into ascending or descending order, changing the order of the values in the array. An “in-place-sort” only uses one additional storage cell to temporarily hold one value from the 1-D array.
For ascending order, a simple code is

```
int vi[100], N=100,temp;
**some code to fill-in value for vi[ ]. **
for(i=0; i<N-1; i++){
  for(j=i+1; i<N; i++){
    if(vi[j]<vi[i]){
      temp=v[i];
      v[i]=v[j];
      v[j]=temp;
    }
  }
}
```

When i=0; we iterate j through all the remaining cells of the 1-D array. As we visit a remaining jth cell, we interchange its value with that of the 0th cell if the remaining jth cell value is less than the ith cell. After visiting all of the remaining j cells (j=1 thru N-1), the smallest value of the array is in the 0th cell. Then we move to the i=1 cell, and repeat. Etc..

Background 3.

Median. No background info needed. You’ve got this.