PRACTICE PROBLEMS - SET 4

#1. Write a program in just main ( ), that allocates an array of 10 integers and fills the integers with random numbers between 1 and 100. How to do that? You can generate a random integer between 1 and 100 with the following lines of code:

```
#include <stdlib.h>
#include <time.h>

...

int number;

srand ((unsigned) time(NULL));  // call this function once; it "seeds" the RNG

// now you can call rand ( ) as often as you like; each time it will generate a random int
// between 0 and RAND_MAX, some big number; to generate random numbers within
// a range, use the % operator

number = rand ( )%100+1;  // the % operator yields 0 - 99; adding 1 yields 1 - 100
```

Then calculate the average, minimum, and maximum values in the array.

#2. Write a program that asks a user for an integer, then counts the number of each digit in the integer, and displays a list of all digits as follows:

```
Enter an integer ::  123232488

 digit   how many?
      1      1
      2      3
      3      2
      4      1
      8      2
```

#3. Write a program that asks a user to enter a word in which one letter appears more often than any other. Then output which letter that is.

How? Allocate an integer array of 26 elements, and initialize them all to 0. Then write a loop to use getchar() to read each letter of the word. For each letter, increment the counter in the array that corresponds to that letter. For example, M is the 13th letter in the alphabet, and so you would increment count[12], which is the 13th element in the array. Finally, once you've read each letter, search through the array for the highest value, remember the index where that highest value was stored, and convert that index to the corresponding letter.

For example:

```
Enter a word:   Bubble
The letter b/B appears most often.
```

#4. Write a program with two functions: main ( ) and

```
int find_index (int num, int j[]);
```

In main(), allocate an array int j[100], and then fill each element of the array with a random number between 1 and 100, where repeating numbers are fine.

Then ask a user for a number num between 1 and 100, and then call find_index, which should search the array, and return the first position (or index, or offset) in the array where num is stored. That returned value will range from 0 to 99 if in fact num is stored in the array; if find_index can't find an instance of num, then the function should return -1.

#5. Assume an array of N elements contains a sorted list of positive non-repeated integers. Write a function that corresponds to the following prototype:

```
int binary_search (int array[], int N, int value);
```

The function does a binary search, and returns the position in the array where value is stored. If value isn't in the array, the function returns -1.

[A binary search is a quick way of searching a sorted list of numbers. Begin at the middle of the array, and decide if the value you're looking for is in the first half of the array or the second. Then look halfway again, and halfway again, and …]

For example, the following main ( ) function, plus the binary_search function that you write:

```
int main ( ) {

  int array[100], i, j;

  for (i=0; i<100; i++)
    array[i] = i*2 + i/2;
  j = binary_search (array, 100, 42)
  printf ("value 42 is stored in array[%2d]\n", j);

  return 0;
}
```

would yield the following output:

```
value 42 is stored in array[17]
```

#6. Write a program that allocates a 2D array of size ROWS x COLS, where these two values are constants specified using #define. Initialize the array with random numbers between 1 and 100, where repeating numbers are fine. Then pass this array to a function corresponding to the following prototype:

```
int largest (int array[ROWS][COLS]);
```

that determines the row or column that sums to the largest number, and returns that sum to main ( ).

#7. Write a program that randomly assigns values between 0 and 9 to a 5 x 5 array. Print the array to the screen, and then calculate the sum of values around the "outside" (border) of the array. For example, if the random array is:

```
8 7 2 0 4
8 4 0 2 5
1 9 4 6 5
0 2 6 9 3
1 7 3 9 5
```

then the sum around the outside is 68

#8. Write a program that randomly assigns the values 1 to 25 to a 5 x 5 array, not repeating any values. Then print the array to the screen. For example:

```
  3 25 12  4 19
  5 17 21 10  6
  1  8 16  2  7
 13 22 24 11 20
 23 14 15  9 18
```

Notice that you'll have to keep track of which numbers you've already placed in the array, or, you can loop from 1 to 25 and randomly select locations in the 5 x 5 array.

If you do this in a simple, but inefficient way, you'll notice it takes longer and longer to place the next value, or find the next location. That's OK for a 5 x 5 array, but won't be appropriate for a much larger (say 1000 x 1000) array. So then what? Try writing a version of this that's efficient, that takes the same amount of time to place the first number as the last.