# CST8234 – C Programming
## Array Manipulations

## Setup:

**Create a directory** Lastname_02**. You are going to develop your lab here**

## Lab Learning Goals

1. Write programs with format input and output with printf() and scanf()
2. Programs using functions and arrays
3. To compile and execute basic C programs

## Submissions

**Important**: Make sure to put your name and any other information (like any sources you used or collaborations you may have had) in comments at the top of your source code. You should get into the habit of doing this for any source code you submit for this (and probably every) class.

Before the deadline, you must:

1. Demo to your lab professor
    - You have 2weeks to finish this lab and demo afterwards
2. Submit on Brightspace before the deadline.
    - What to submit: a zip **Lab1_Fname_Lname.zip** folder containing the .c and .h source files only

- Where to submit: Brightspace =>Assignments =>Activities => Lab2

## Compiling your code

To compile your code, use `gcc` the gnu c compiler using the following command:

```
gcc -Wall -g arraymanip.c -o arraym
```

The `-Wall` option will provide additional warnings. The `-g` option will include debugging symbols in case you want to use the debugger.

## Lab Description and Requirements

You will write a single program called manipulator which will take in user input to construct an array, and then perform basic manipulations over that array as desired by the user. Here is a basic run of the program for reference:

```
$ ./arraym
Enter the length:
6
Enter 6 numbers (space separated):
1 2 3 4 5 6
Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
1
{ 6 5 4 3 2 1 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
2
{ 4 1 5 2 6 3 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
3
{ 1 2 3 4 5 6 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
0
```

## Part 1 – Reading input from the user

The first part of the lab is to get user input using `scanf()`, error checking using the concept of field width and populate the array. In this part of the lab you will write a simple program that takes input from the user and manipulates that input in some way depending on the user choice. You must also error check user input. Below is some additional information to help you complete this task.

```
RETURN VALUES

     This function returns the number of input items assigned.  This
     can be fewer than provided for, or even zero, in the event of a
     matching failure.  Zero indicates that, although there was input
     available, no conversions were assigned; typically, this is due to
     an invalid input character, such as an alphabetic character for a
     `%d' conversion.  The value EOF is returned if an input failure
     occurs before any conversion such as an end-of-file occurs.  If
     an error or end-of-file occurs after conversion has begun, the
     number of conversions which were successfully completed is
     returned.
```

What this means is that if a user enters something that is not a number, then scanf() will return 0, indicating that it could not properly convert the input into a number given the %d format directive. We can easily check for such a condition in our code:

```
int a, b;
printf("Enter a number:\n");
b = scanf("%d",&a);
if( b == 0){
  printf("ERROR: Invalid input");
        //exit, return, or take some other action
}
```

## Part 2: Reversing the array

In this part of the lab you are going to complete the function to reverse the array.

**Requirements**

- You must complete the `reverse_array()` function and properly call it in the main() function when the user provides the option 2.
- You must print the array following the completion of the operation. (Hint: you've already written this operation; don't do something twice!)

### Sample output:

```
$ ./arraym
Enter the length:
5
Enter 5 numbers (space separated):
0 1 2 3 4
Choose an operation:
(0) : exit
(1) : reverse array
```

```
(2) : randomize array
(3) : sort array
1
{ 4 3 2 1 0 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
0
```

## Part 3 – Randomizing the array

### Random Integers

A random number generator is built into the C standard library: random() returns a random integer between 0 and 264-1. For the purposes of this lab, you'll only need random numbers in the range of the length of the array. To bound the random numbers being generated, use the modulo operator:

```
int r = random() % bound; //produce random number between 0 and bound-1
```

### Seeding Random Numbers

Additionally, as you test your program, you will likely want your random numbers to be deterministic, that is, be random but predictably random so that each run you get the same random numbers. To make things easier for you, I have seeded the random number generator like so:

```
srandom(1964); //seed random number generator with seed 1964
```

### Pseudocode for array randomization

Randomizing an array of values is equivalent for performing random swaps for items in the array. Here is some pseudocode to use:

```
foreach index i in array:
  choose a random index j
  swap array[i] with array[j]
```

Sample output:

```
$ ./arraym
Enter the length:
5
Enter 5 numbers (space separated):
0 1 2 3 4
Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
2
{ 3 1 2 4 0 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
0
```

## Part 4 – Sorting the array

The last part of the assessment is the trickiest. You must write an array sort routine. You may or may not have had experience writing sort functions before, so I encourage you to look up something like the insertion sort article on Wikipedia (Insertion sort) which is perhaps the easiest to implement.

Sample output:

```
$ ./arraym
Enter the length:
5
Enter 5 numbers (space seperated):
0 1 2 3 4
Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
2
{ 3 1 2 4 0 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
```

```
 3
{ 0 1 2 3 4 }

Choose an operation:
(0) : exit
(1) : reverse array
(2) : randomize array
(3) : sort array
0
```

## Marking

Refer to the rubric