Lab 08: Pointers

Objectives

- To be able to use pointers.
- To be able to generate random numbers using the **`rand()`** function
- To be able to use pointers to pass arguments to functions using call-by reference.
- To understand the close relationships among pointers and arrays.
- To understand the use of pointers to functions.

Pointers

Pointers are variables that points to the locations of elements in the memory. They can be used to point to variables as in the following code:

```
#include <stdio.h>
int main(void){
   int a;           /* a is an integer */
   int * aPtr;      /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a;/* Assign the address of a to aPtr */

   printf( "The address of a is %p\n"
           "The value of aPtr is %p\n", &a, aPtr );

   printf( "The value of a is %d\n"
           "The value of (*aPtr) is %d\n", a, (*aPtr) );

   printf( "Showing that * and & are complements of each other\n"
           "(&*aPtr) = %p\n(*&aPtr) = %p\n", (&*aPtr), (*&aPtr) );
}
```

Pointers also can be used with array variables as in the following code:

```
#include <stdio.h>
int main(void){
    int Array[] = {4, 1, 87, -66, 2, -27};
    int * ptr;
    int i;

    printf("\n1) Show Array using array notation\n");
    printf("\t Element \t\tAddress of element\n");
    printf("\t-------------------------------------------------\n");
    for (i = 0 ; i < 6 ; i++) {
        printf("\t Array[%d] = %d\t\t%p\n", i, Array[i], &Array[i]);
    }

    /* point our pointer to the first element of Array */
    ptr = &Array[0]; /* equivalent to ptr = Array */
```

```
    printf("\n2) Show Array using a pointer (ptr)\n");
    printf("\t Element \t\tAddress of element\n");
    printf("\t-------------------------------------------------\n");
    for(i = 0 ;i < 6; i++) {
        printf("\t(*ptr) = %d \t\t%p\n", (*ptr), ptr);
        ptr++;
    }
}
```

Activity 1

Write a program that generates an array of random numbers between **`MIN`** and **`MAX`** using **`rand()`**. After that, the program shifts the array of several PLACES to the right or to the left. Print the array before and after the shift.

```
#define MaxSize 20
#define MIN 1
#define MAX 100
#define PLACES 3

void printArray(int *s_ptr, int *e_ptr);
void shiftArrayToRightBy(int array[], int size, int places);
void shiftArrayToLeftBy(int array[], int size, int places);
```

Task 1.1: Define the function:

```
void printArray(int *s_ptr, int *e_ptr);
```

that prints the given array. The parameters are the starting address of the array (first element) and the ending address of the array (last element).

Task 1.2: Define the function:

```
void shiftArrayToRightBy(int array[], int size, int places);
```

that shifts the given array to the right by a number of places.

Task 1.3: Define the function:

```
void shiftArrayToLeftBy (int array[], int size, int places);
```

that shifts the given array to the left by a number of places.

Task 1.4: change the program in activity 1 and let the user to choose in which direction is the array should be shifted. Use the following new type of direction.

```
typedef char direction;
#define right 'R' || 'r'
#define left 'L' || 'l'
```

For example, if the user enters 'R' or 'r', that means the program should shift the array to the right. Alternatively, 'L' or 'l' means the program should shift the array to the left.

Activity 2

Consider the following C programs that randomly generate an integer array between 1 and 100 and then filter it and distinguish even and odd numbers in separate arrays. Note that, all arrays must end with a zero. Print array using pointers in a function.

```
#define MIN 1
#define MAX 100
#define LEN 10 // only 10 elements

/* protoypes */

    // generate random numbers between MIN and MAX
    // print the array
    // generate even array by filtering allNumbers array and printing it

    // generate odd array by filtering allNumbers array and printing it
```

Task 2.1: define the function

```
int * generateArray(int min, int max, int length){
```

that generates an array of random numbers between min and max with a specific length and then returns the array (the pointer of the array). Since this is a function, the created array must be allocated in the memory using `malloc` function. The array must be terminated by zero (i.e., the value of last location must be zero).

Task 2.2: define the function

```
int * filterEvens(int * arrayPtr);
```

that takes a pointer to an array '`arrayPtr`' and then generates a separate array of only even numbers taken from the '`arrayPtr`'. The array must be generated dynamically using `malloc`.

The generated array must also be terminated by zero. The function must return the pointer of the new array of locations.

Task 2.3: define the function

```
int * filterOdds(int * arrayPtr);
```

that takes a pointer to an array ‘`arrayPtr`’ and then generates a separate array of only odd numbers taken from the ‘`arrayPtr`’. The array must be generated dynamically using `malloc`. The generated array must also be terminated by zero. The function must return the pointer of the new array of locations.