# Lab 09 Arrays and Pointers

**Objectives:**

- Learn the basic operations on arrays (e.g. traversal, value swapping, retrieving indices etc)
- Accessing array elements via pointers.
- Practice array operations by implementing simple sorting algorithms.

**Reading Task 1: Working with Arrays**

Chapter 08 Arrays (pages 269 to 288) from the book: “Let us C” by Yashavant Kanetkar

**Reading Task 2: Selection Sort Algorithm**

In computer science, selection sort is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

**Example:**

Here is an example of this sort algorithm sorting five elements:

| Sorted sublist | Unsorted sublist | Least element in unsorted list |
|---|---:|---|
| ( ) | (11, 25, 12, 22, 64) | 11 |
| (11) | (25, 12, 22, 64) | 12 |
| (11, 12) | (25, 22, 64) | 22 |
| (11, 12, 22) | (25, 64) | 25 |
| (11, 12, 22, 25) | (64) | 64 |
| (11, 12, 22, 25, 64) | ( ) | |

Selection sort can also be used on list structures that make add and remove efficient, such as a linked list. In this case it is more common to *remove* the minimum element from the remainder of the list, and then *insert* it at the end of the values sorted so far. For example:

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

*Selection Sort Example*

**In-Lab Task 1: Finding Minimum and Maximum Values in an Array**

Your task is to perform some functions on integer arrays. Specifically you will write a C program that does the following:

1. Declare an array of size 20.
2. Initialize the array with random values (use loop, and rand() function).
3. Print all the elements in the array.
4. Print all the elements in the array in the reverse order.
5. Print the array such that every N$^{th}$ element gets printed. N is user input.

**In-Lab Task 2: Implementing Selection Sort**

You are given a C program in **Code Listing 1**, that does the following:

1. Declares an integer array with 50 elements (not initialized).
2. Populates the array with random positive numbers. (Uses a loop and rand() function)
3. Calls the function ‘`int` **`find_max`**`(int * ptr_array, int size)`’ and prints the value and index of the largest number.

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

#define ASCENDING 0
#define DESCENDING 1

#define ARRAY_SIZE 50

int find_max(int * ptr_array, int size);
int find_min(int * ptr_array, int size);
void selection_sort(int * ptr_array, int size, int order);

int main()
{
    int num_array[ARRAY_SIZE];  /// Declare an integer array

    int * ptr_ar = &num_array[0];   /// A pointer to the start of the array

    for(int i =0; i < ARRAY_SIZE; i++)
    {
      num_array[i] = rand()%100;  /// Initialize the array with random numbers in range 0 to 99
      printf("%d ", num_array[i]);/// and print it.
    }

    int mx_idx = find_max(ptr_ar, ARRAY_SIZE);  /// Print the maximum value and its index
    printf("\nThe maximum number is %d at index %d \n", num_array[mx_idx], mx_idx);

    int mn_idx = find_min(ptr_ar, ARRAY_SIZE);  /// Print the minimum value and its index
    printf("\nThe minimum number is %d at index %d \n", num_array[mn_idx], mn_idx);

    selection_sort(num_array, ARRAY_SIZE, ASCENDING);   /// Sort the array using Selection Sort
    for(int i=0; i<ARRAY_SIZE; i++)                     /// Print the sorted array
        printf("%d ", num_array[i]);

    return 0;
}

int find_max(int * ptr_array, int size)
{
    int max_val = 0;
    int max_idx = 0;

    for(int i=0; i<size; i++)
    {

        if(*(ptr_array+i) > max_val)
        {
            max_val = *(ptr_array+i);
            max_idx = i;
        }
    }

    return(max_idx);

}
int find_min(int * ptr_array, int size)
{
    return 0;
}
void selection_sort(int * ptr_array, int size, int order)
{

}
```

*Code Listing 1*

Write a similar function ‘`int find_min(int * ptr_array, int size)`’ and print the value and index of the smallest number in the array as well.

## Implementing Selection Sort

Your second task is to implement the Selection Sort algorithm by making a function with the following prototype;

```
void selection_sort(int * ptr_array, int size, int order);
```

This function takes as input a pointer to the start of the array, and the array size and sorts it in-place. The last input to the function is the sorting order (0 for ascending and 1 for descending). You should call the functions (**find_max**(), **find_min**()) developed in lab-task 1 to implement Selection Sort algorithm.

## Post-Lab Task: Implement Insertion Sort Algorithm

Your second task is to implement the Insertion Sort algorithm by making a function with the following prototype;

```
void insertion_sort(int * ptr_array, int size, int order);
```

This function takes as input a pointer to the start of the array, and the array size and sorts it in-place. The last input to the function is the sorting order (0 for ascending and 1 for descending).

## References:

1. https://www.toptal.com/developers/sorting-algorithms
2. https://en.wikipedia.org/wiki/Selection_sort
3. https://en.wikipedia.org/wiki/Insertion_sort