Aim ↬ To perform basic operations on 1-D arrays.

Objectives ↬

i. Write a program to find max and min elements in a 1-D array.
ii. Write a program to create a 1-D Integer Array using dynamic memory allocation.
iii. Enter the values of Array elements using the keyboard. Perform the following operations on it:
    a) Traverse the Array from first to last.
    b) Traverse the Array from last to first.
    c) Search a particular number in the Array.
iv. Write a program to create a 1-D Integer array. Perform the following operations on it:
    a) Insert an element at a given position
    b) Delete an element present at a given position.

Software Required ↬ Visual Studio Code

Code 1 ↬

```
#include <stdio.h>

int main()

{

    int max, min, length;

    printf("Enter the length of array: ");

    scanf("%d", &length);

    int arr[length];

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

        printf("Element %d : ", i + 1);

        scanf("%d", &arr[i]);

    }

    printf("Array: ");

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

        printf(" %d", arr[i]);
```

```
    }
    printf("\n");
    max = min = arr[0];
    for (int i = 0; i < length; i++){
        if (arr[i] > max)
            max = arr[i];
        if (arr[i] < min)
            min = arr[i];
    }
    printf("Maximum element of array: %d\n", max);
    printf("Minimum element of array: %d\n", min);
    return 0;
}
```

Output ↬

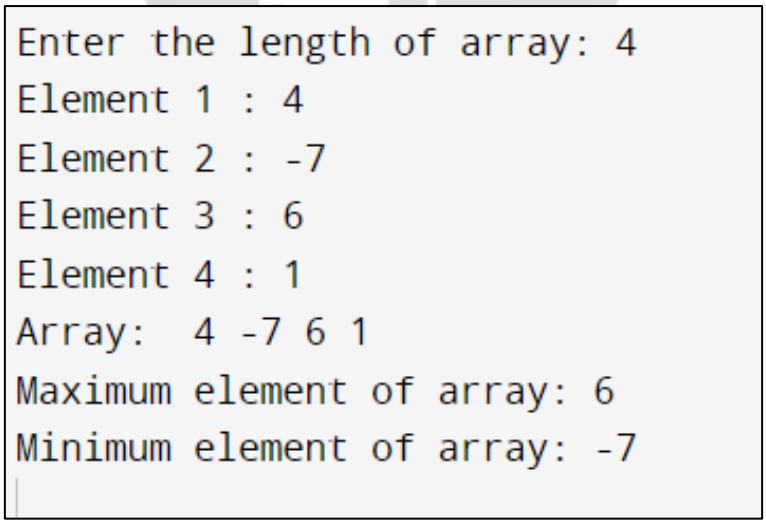

Fig. i) Max and Min elements in Array

Code 2 ↬

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

```
int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    int *arr = (int *)malloc(size * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed. Exiting the program.\n");
        return 1;
    }
    printf("Enter %d elements for the array:\n", size);
    for (int i = 0; i < size; ++i) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);}

    // Traverse the array from first to last
    printf("Traversing the array from first to last:\n");
    for (int i = 0; i < size; ++i) {
        printf("%d ", arr[i]);}
printf("\n");

    // Traverse the array from last to first
    printf("Traversing the array from last to first:\n");
    for (int i = size - 1; i >= 0; --i) {
        printf("%d ", arr[i]); }
    printf("\n");
```

```
    // Search for a particular number in the array
    int searchNumber;
    printf("Enter a number to search in the array: ");
    scanf("%d", &searchNumber);
    int foundIndex = -1;
    for (int i = 0; i < size; ++i) {
        if (arr[i] == searchNumber) {
            foundIndex = i;
            printf("Number found at index %d.\n", foundIndex);
            break;
        }
    }
    if (foundIndex == -1)
        printf("Number not found in the array.\n");
    free(arr);
    return 0;
}
```

Output ↬

```
Enter the size of the array: 4
Enter 4 elements for the array:
Element 1: 6
Element 2: 7
Element 3: 2
Element 4: 5
Traversing the array from first to last:
6 7 2 5
Traversing the array from last to first:
5 2 7 6
Enter a number to search in the array: 6
Number found at index 0.
```

Fig. ii) Traversing in Array

Code 3 ↬

```
#include <stdio.h>
#define MAX_SIZE 100
void displayArray(int arr[], int size) {
    printf("Array elements: ");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void insertElement(int arr[], int *size, int position, int element) {
    if (*size >= MAX_SIZE) {
        printf("Array is full, cannot insert more elements.\n");
        return;
    }

    if (position < 0 || position > *size) {
        printf("Invalid position for insertion.\n");
        return;
    }
    for (int i = *size; i > position; i--)
        arr[i] = arr[i - 1];
    arr[position] = element;
    (*size)++;
    printf("Element %d inserted at position %d.\n", element, position);
}
```

```
void deleteElement(int arr[], int *size, int position) {
    if (*size <= 0) {
        printf("Array is empty, cannot delete elements.\n");
        return;
    }

    if (position < 0 || position >= *size) {
        printf("Invalid position for deletion.\n");
        return;
    }
    int deletedElement = arr[position];
    for (int i = position; i < *size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    (*size)--;
    printf("Element %d deleted from position %d.\n", deletedElement, position);
}

int main() {
    int arr[MAX_SIZE];
    int size = 0;
    insertElement(arr, &size, 0, 5);
    insertElement(arr, &size, 1, 10);
    insertElement(arr, &size, 2, 15);
    displayArray(arr, size);
    deleteElement(arr, &size, 1);
```

```
    displayArray(arr, size);

    return 0;

}
```

Output ↬

```
Element 5 inserted at position 0.
Element 10 inserted at position 1.
Element 15 inserted at position 2.
Array elements: 5 10 15
Element 10 deleted from position 1.
Array elements: 5 15
```

Fig. iii) Insertion & Deletion in Array

Result ↬

The programs were successfully implemented to perform basic operations on 1-D arrays, including finding maximum and minimum elements, dynamic memory allocation, traversing arrays, searching, insertion, and deletion. The outputs matched the expected results for all operations, demonstrating correct functionality.

Conclusion ↬

The experiment effectively demonstrated various fundamental operations on 1-D arrays using C programming, enhancing the understanding of array manipulation, dynamic memory allocation, and the implementation of common array operations.

Precautions ↬

- Ensure proper input validation for array size, positions for insertion and deletion to avoid runtime errors.
- Avoid accessing memory beyond the array bounds to prevent segmentation faults or undefined behavior.
- Free dynamically allocated memory after use to prevent memory leaks.