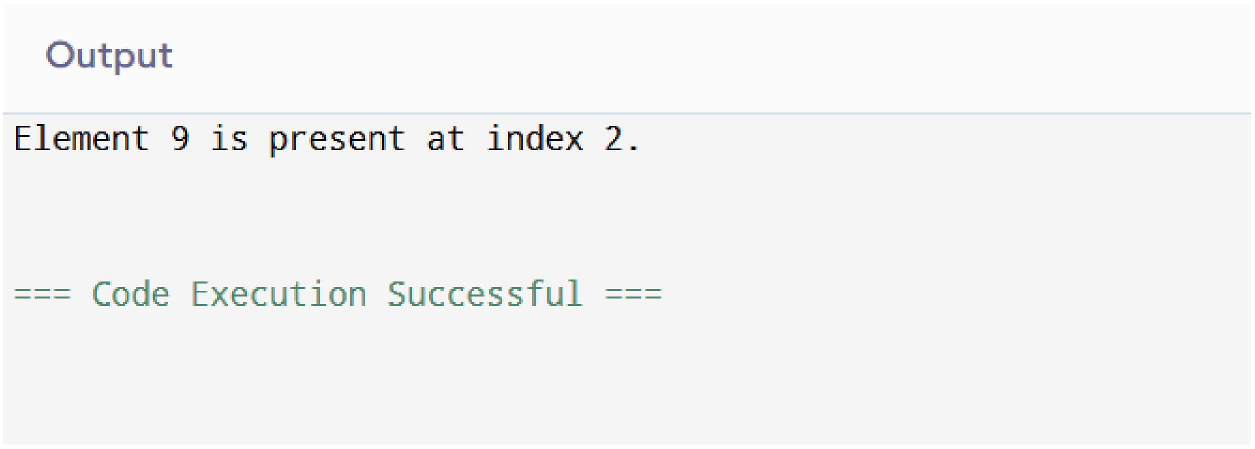
LAB ASSIGNMENT 2

1.Write a program to check whether a given number is present in an array or not (Linear search).

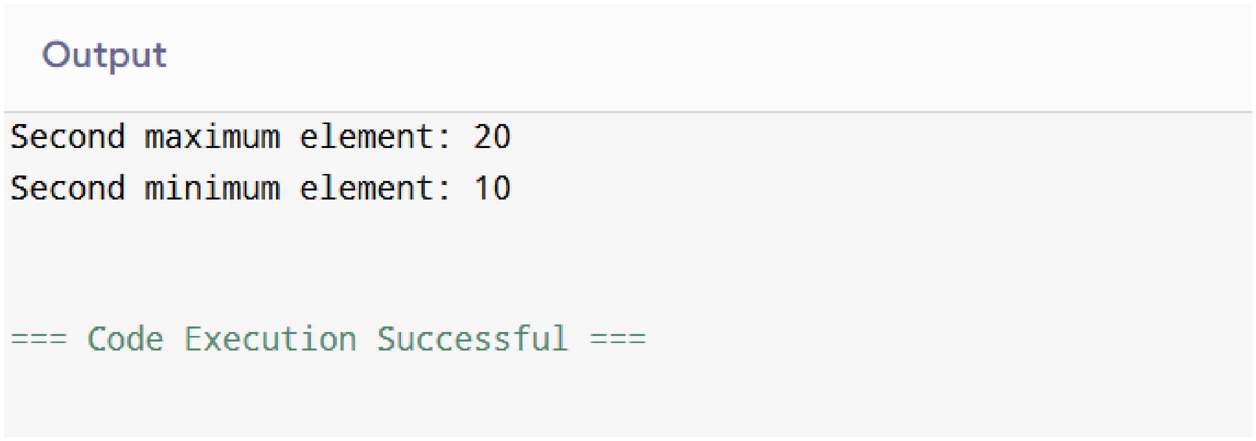
```
#include <stdio.h>

// Function to perform linear search
int linearSearch(int arr[], int size, int key) {

 // Loop through each element of the array
 for (int i = 0; i < size; i++) {

// Check if the current element is equal to the key
if (arr[i] == key) {

// If found, return the index
return i;
}
}
// If the loop completes without finding the key, return -1
return -1;
}

int main() {
    int arr[] = {5, 12, 9, 3, 15, 8};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 9;

    // Call the linear search function
    int result = linearSearch(arr, size, key);

    // Check the result and print the appropriate message
    if (result != -1) {
        printf("Element %d is present at index %d.\n", key, result);
    } else {
        printf("Element %d is not present in the array.\n", key);
    }

    return 0;
}
```


Output

```
Element 9 is present at index 2.


=== Code Execution Successful ===
```

2.Write a program to get second maximum and second minimum elements in an array.

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

// Function to find the second maximum element
int findSecondMax(int arr[], int size) {
int max1 = INT_MIN, max2 = INT_MIN;

for (int i = 0; i < size; i++) {
if (arr[i] > max1) {
max2 = max1;
max1 = arr[i];
}
else if (arr[i] > max2 && arr[i] != max1) {
max2 = arr[i];
}
}
return max2;
}

// Function to find the second minimum element
int findSecondMin(int arr[], int size) {
int min1 = INT_MAX, min2 = INT_MAX;
for (int i = 0; i < size; i++) {
if (arr[i] < min1) {
min2 = min1;
min1 = arr[i];
} else if (arr[i] < min2 && arr[i] != min1) {
min2 = arr[i];
}
}
return min2;
}
int main() {
int arr[] = {10, 5, 20, 15, 25};
int n = sizeof(arr) / sizeof(arr[0]);
// Check if the array has at least two elements
if (n < 2) {
printf("The array must have at least two elements.\n");
return 1;
}

int secondMax = findSecondMax(arr, n);
int secondMin = findSecondMin(arr, n);

printf("Second maximum element: %d\n", secondMax);
printf("Second minimum element: %d\n", secondMin);

return 0;
}
```



Output

```
Second maximum element: 20
Second minimum element: 10


=== Code Execution Successful ===
```

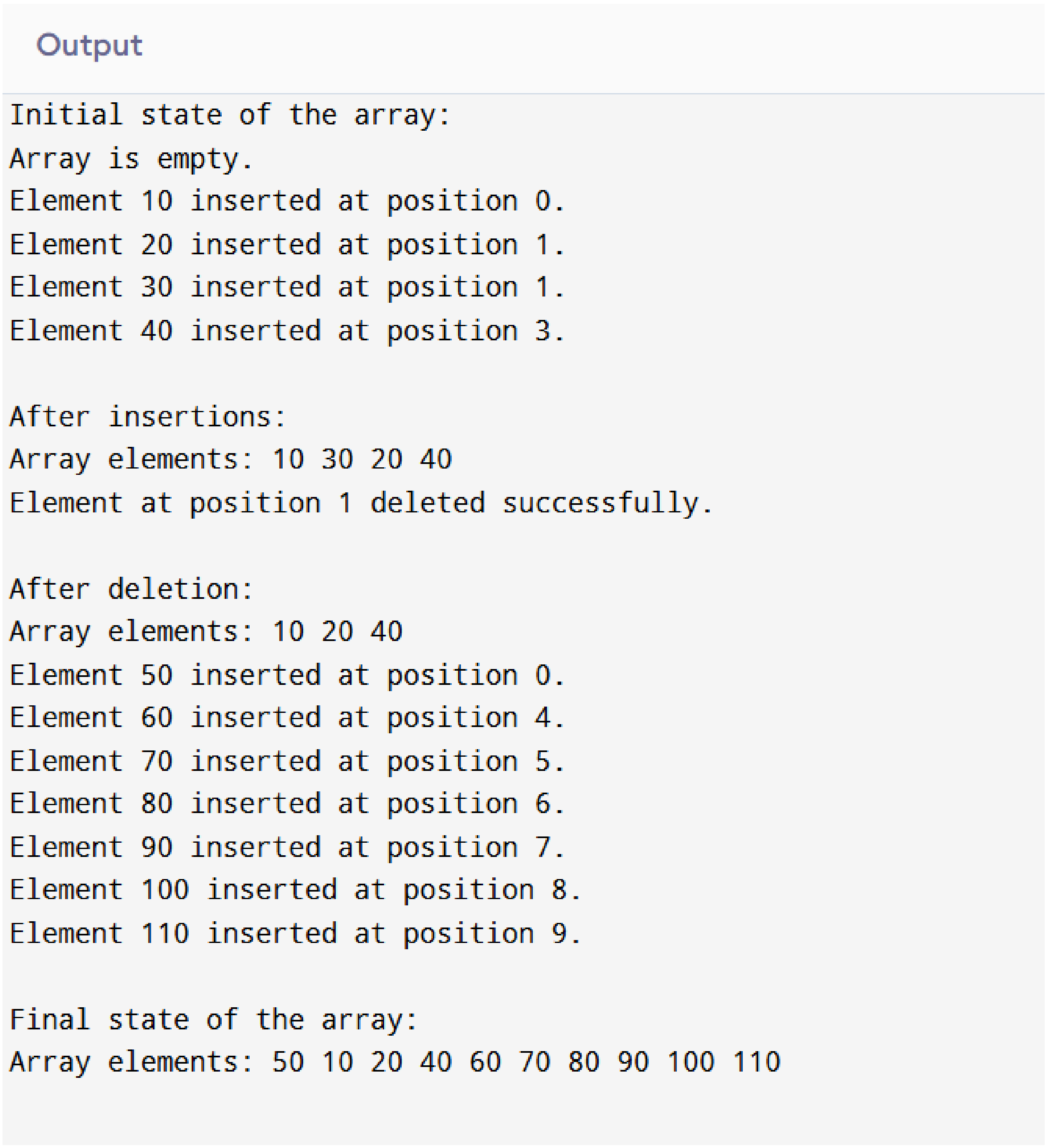
3.Write a program to perform insertion (any location), deletion (any location) and traversal in an array.

```
#include <stdio.h>

// Function to traverse and print the elements of the array
void traverseArray(int arr[], int size) {
if (size == 0) {
printf("Array is empty.\n");
return;
```

```
}
printf("Array elements: ");
for (int i = 0; i < size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}

// Function to insert an element at a specified position
int insertElement(int arr[], int size, int element, int position, int capacity) {
// Check if the array is full or the position is invalid
if (size >= capacity) {
printf("Error: Array is full. Insertion failed.\n");
return size;
}
if (position < 0 || position > size) {
printf("Error: Invalid position for insertion.\n");
return size;
}
// Shift elements to the right to make space for the new element
for (int i = size - 1; i >= position; i--) {
arr[i + 1] = arr[i];
}

// Insert the new element
arr[position] = element;
printf("Element %d inserted at position %d.\n", element, position);
return size + 1; // Return the new size of the array
}

// Function to delete an element from a specified position
int deleteElement(int arr[], int size, int position) {
// Check if the array is empty or the position is invalid
if (size == 0) {
printf("Error: Array is empty. Deletion failed.\n");
return size;
}
if (position < 0 || position >= size) {
printf("Error: Invalid position for deletion.\n");
return size;
}

// Shift elements to the left to overwrite the element to be deleted
for (int i = position; i < size - 1; i++) {
```

```
arr[i] = arr[i + 1];
}

printf("Element at position %d deleted successfully.\n", position);
return size - 1; // Return the new size of the array
}

int main() {
int capacity = 10;
int arr[capacity];
int size = 0; // Current number of elements in the array

// Initial traversal
printf("Initial state of the array:\n");
traverseArray(arr, size);

// Insertion
size = insertElement(arr, size, 10, 0, capacity);
size = insertElement(arr, size, 20, 1, capacity);
size = insertElement(arr, size, 30, 1, capacity);
size = insertElement(arr, size, 40, 3, capacity);

// Traversal after insertions
printf("\nAfter insertions:\n");
traverseArray(arr, size);

// Deletion
size = deleteElement(arr, size, 1);

// Traversal after deletion
printf("\nAfter deletion:\n");
traverseArray(arr, size);

// More insertions to demonstrate capacity check
size = insertElement(arr, size, 50, 0, capacity);
size = insertElement(arr, size, 60, 4, capacity);
size = insertElement(arr, size, 70, 5, capacity);
size = insertElement(arr, size, 80, 6, capacity);
size = insertElement(arr, size, 90, 7, capacity);
size = insertElement(arr, size, 100, 8, capacity);

// Attempting to insert into a full array
size = insertElement(arr, size, 110, 9, capacity);
```

```
// Final traversal
printf("\nFinal state of the array:\n");
traverseArray(arr, size);

return 0;
}
```



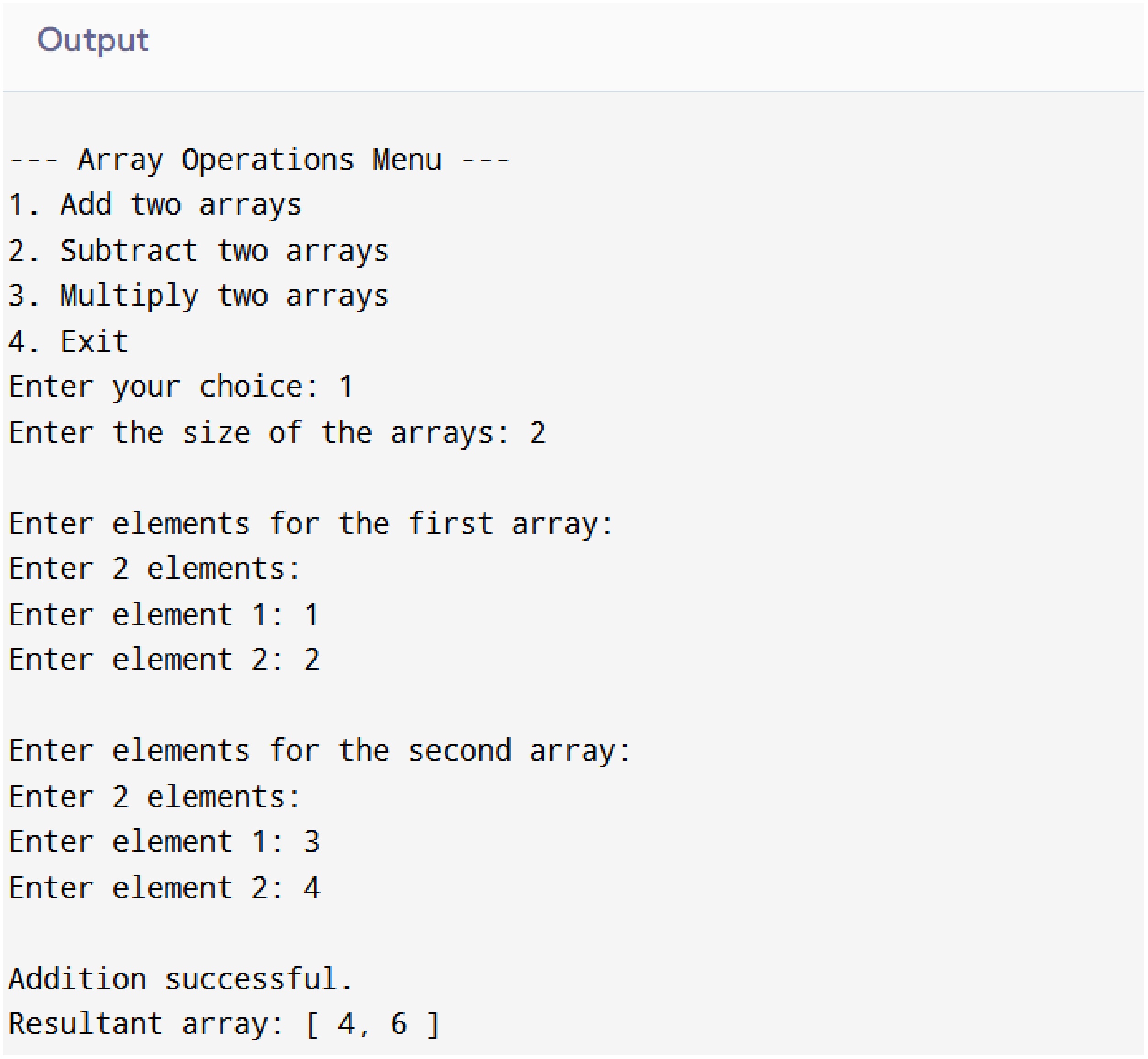
Output

```
Initial state of the array:
Array is empty.
Element 10 inserted at position 0.
Element 20 inserted at position 1.
Element 30 inserted at position 1.
Element 40 inserted at position 3.

After insertions:
Array elements: 10 30 20 40
Element at position 1 deleted successfully.

After deletion:
Array elements: 10 20 40
Element 50 inserted at position 0.
Element 60 inserted at position 4.
Element 70 inserted at position 5.
Element 80 inserted at position 6.
Element 90 inserted at position 7.
Element 100 inserted at position 8.
Element 110 inserted at position 9.

Final state of the array:
Array elements: 50 10 20 40 60 70 80 90 100 110
```

4.Write a menu driven program to perform addition, multiplication and subtraction of 2 arrays.

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

// Function to get array elements from the user
void getArrayElements(int arr[], int size) {
printf("Enter %d elements:\n", size);
for (int i = 0; i < size; i++) {
printf("Enter element %d: ", i + 1);
scanf("%d", &arr[i]);
}
}

// Function to print the elements of an array
void printArray(int arr[], int size) {
printf("Resultant array: [ ");
for (int i = 0; i < size; i++) {
printf("%d", arr[i]);
if (i < size - 1) {
printf(", ");
}
}
printf(" ]\n");
}

// Function to perform addition of two arrays
void addArrays(int arr1[], int arr2[], int result[], int size) {
for (int i = 0; i < size; i++) {
result[i] = arr1[i] + arr2[i];
}
printf("\nAddition successful.\n");
}

// Function to perform subtraction of two arrays
void subtractArrays(int arr1[], int arr2[], int result[], int size) {
for (int i = 0; i < size; i++) {
result[i] = arr1[i] - arr2[i];
}
printf("\nSubtraction successful.\n");
}

// Function to perform element-wise multiplication of two arrays
```

```
void multiplyArrays(int arr1[], int arr2[], int result[], int size) {
for (int i = 0; i < size; i++) {
result[i] = arr1[i] * arr2[i];
}
printf("\nMultiplication successful.\n");
}

int main() {
int choice;
int size;
int *arr1 = NULL;
int *arr2 = NULL;
int *result = NULL;

do {
printf("\n--- Array Operations Menu ---\n");
printf("1. Add two arrays\n");
printf("2. Subtract two arrays\n");
printf("3. Multiply two arrays\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
case 1:
case 2:
case 3:
printf("Enter the size of the arrays: ");
scanf("%d", &size);
// Allocate memory for the arrays
arr1 = (int *)malloc(size * sizeof(int));
arr2 = (int *)malloc(size * sizeof(int));
result = (int *)malloc(size * sizeof(int));

if (arr1 == NULL || arr2 == NULL || result == NULL) {
printf("Memory allocation failed. Exiting.\n");
return 1;
}

printf("\nEnter elements for the first array:\n");
getArrayElements(arr1, size);

printf("\nEnter elements for the second array:\n");
getArrayElements(arr2, size);
if (choice == 1) {
addArrays(arr1, arr2, result, size);
} else if (choice == 2) {
subtractArrays(arr1, arr2, result, size);
} else {
multiplyArrays(arr1, arr2, result, size);
}
printArray(result, size);
// Free allocated memory
free(arr1);
free(arr2);
free(result);
break;
case 4:
printf("Exiting program. Goodbye!\n");
break;
default:
printf("Invalid choice. Please try again.\n");
break;
}

} while (choice != 4);

return 0;
}
```

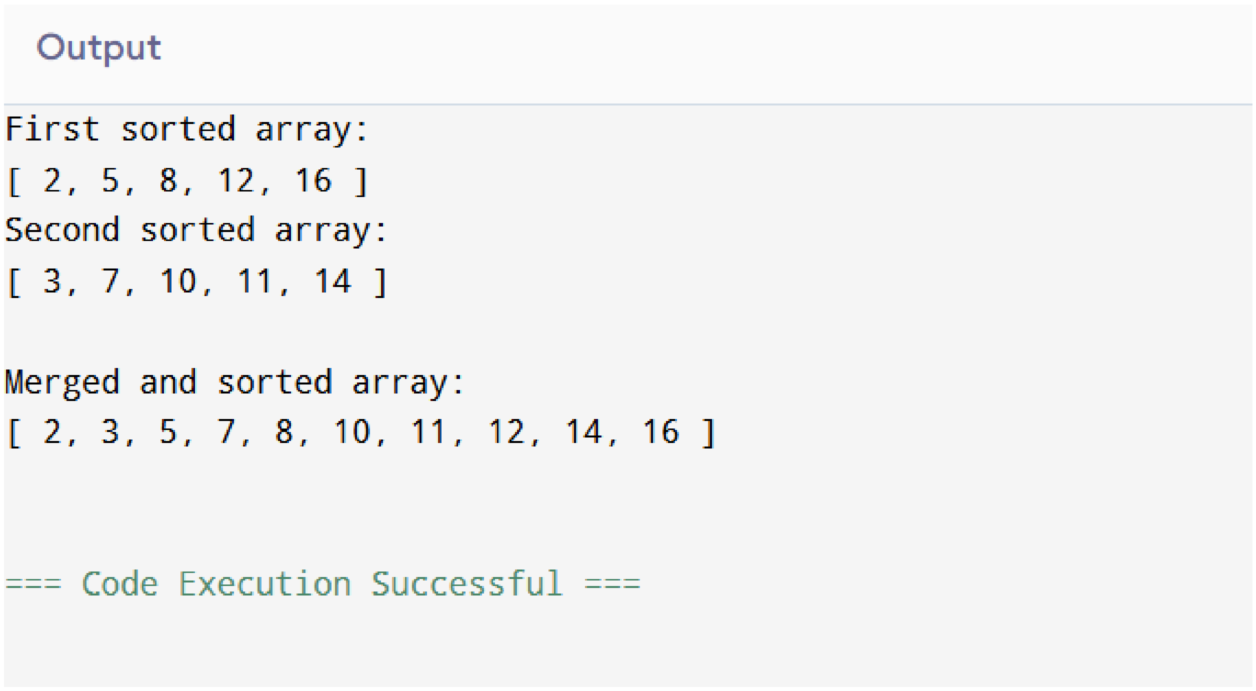

Output

```
--- Array Operations Menu ---
1. Add two arrays
2. Subtract two arrays
3. Multiply two arrays
4. Exit
Enter your choice: 1
Enter the size of the arrays: 2

Enter elements for the first array:
Enter 2 elements:
Enter element 1: 1
Enter element 2: 2

Enter elements for the second array:
Enter 2 elements:
Enter element 1: 3
Enter element 2: 4

Addition successful.
Resultant array: [ 4, 6 ]
```

5.Write a program to perform sorting while merging (Merge two sorted arrays into one sorted array).

```
#include <stdio.h>

// Function to merge two sorted arrays into one
void mergeSortedArrays(int arr1[], int size1, int arr2[], int size2, int result[]) {
int i = 0; // Pointer for the first array (arr1)
int j = 0; // Pointer for the second array (arr2)
int k = 0; // Pointer for the result array

// Compare elements from both arrays and add the smaller one to the result array
```

```
while (i < size1 && j < size2) {
if (arr1[i] <= arr2[j]) {
result[k] = arr1[i];
i++;
} else {
result[k] = arr2[j];
j++;
}
k++;
}

// Add any remaining elements from the first array
while (i < size1) {
result[k] = arr1[i];
i++;
k++;
}

// Add any remaining elements from the second array
while (j < size2) {
result[k] = arr2[j];
j++;
k++;
}
}

// Function to print the elements of an array
void printArray(int arr[], int size) {
printf("[ ");
for (int i = 0; i < size; i++) {
printf("%d", arr[i]);
if (i < size - 1) {
printf(", ");
}
}
printf(" ]\n");
}

int main() {
int arr1[] = {2, 5, 8, 12, 16};
int size1 = sizeof(arr1) / sizeof(arr1[0]);

int arr2[] = {3, 7, 10, 11, 14};
int size2 = sizeof(arr2) / sizeof(arr2[0]);
int resultSize = size1 + size2;
int resultArray[resultSize];

printf("First sorted array:\n");
printArray(arr1, size1);

printf("Second sorted array:\n");
printArray(arr2, size2);

// Call the merge function
mergeSortedArrays(arr1, size1, arr2, size2, resultArray);

printf("\nMerged and sorted array:\n");
printArray(resultArray, resultSize);

return 0;
}
```



Output

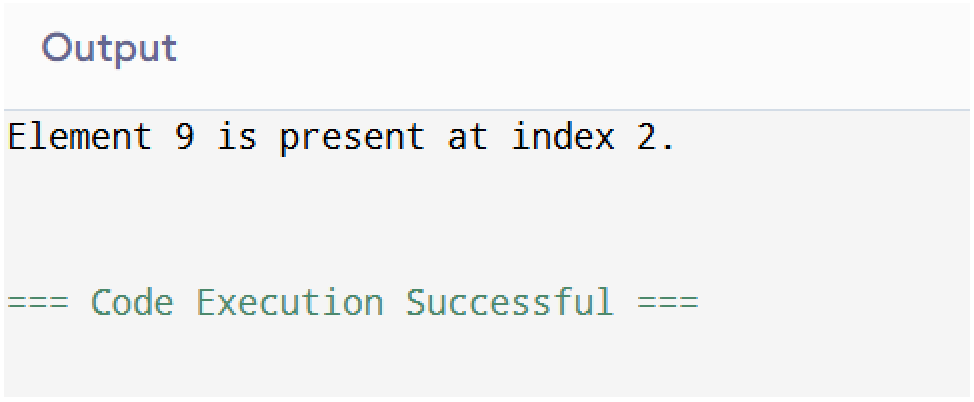
```
First sorted array:
[ 2, 5, 8, 12, 16 ]
Second sorted array:
[ 3, 7, 10, 11, 14 ]

Merged and sorted array:
[ 2, 3, 5, 7, 8, 10, 11, 12, 14, 16 ]


=== Code Execution Successful ===
```

6.Write the above programs (1,2, and 3) using functions and call by address only

1.

```
#include <stdio.h>

// Function to perform a linear search using call by address for the result
// The 'foundIndex' pointer is used to store the result directly in main's memory space.
```

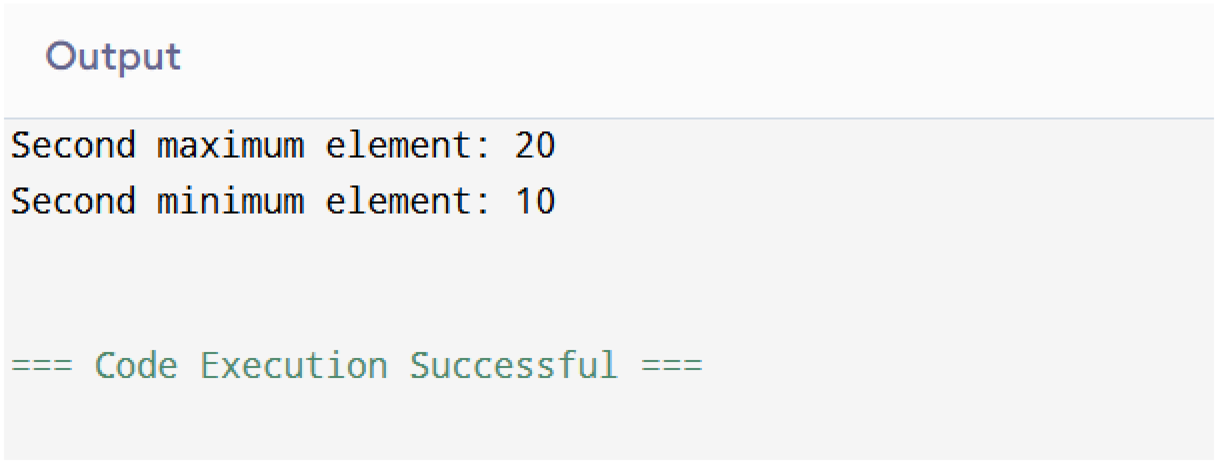
```
void linearSearch(int *arr, int size, int key, int *foundIndex) {
*foundIndex = -1; // Default to not found

for (int i = 0; i < size; i++) {
if (*(arr + i) == key) {
*foundIndex = i; // Store the index at the given memory address
return;
}
}
}

int main() {
int arr[] = {5, 12, 9, 3, 15, 8};
int size = sizeof(arr) / sizeof(arr[0]);
int key = 9;
int resultIndex; // Variable to receive the result

// Pass the address of the array and the address of the result variable
linearSearch(arr, size, key, &resultIndex);

if (resultIndex != -1) {
printf("Element %d is present at index %d.\n", key, resultIndex);
} else {
printf("Element %d is not present in the array.\n", key);
}

return 0;
}
```



```
Element 9 is present at index 2.


=== Code Execution Successful ===
```

2.

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

// Function to find the second max and min using call by address
// Pointers 'secondMax' and 'secondMin' are used to store results.
void findSecondMaxAndMin(int *arr, int size, int *secondMax, int *secondMin) {
int max1 = INT_MIN, max2 = INT_MIN;
int min1 = INT_MAX, min2 = INT_MAX;

if (size < 2) {
*secondMax = INT_MIN;
*secondMin = INT_MAX;
return;
}

for (int i = 0; i < size; i++) {
// Update max values
if (arr[i] > max1) {
max2 = max1;
max1 = arr[i];
} else if (arr[i] > max2 && arr[i] != max1) {
max2 = arr[i];
}
// Update min values
if (arr[i] < min1) {
min2 = min1;
min1 = arr[i];
} else if (arr[i] < min2 && arr[i] != min1) {
min2 = arr[i];
}
}

*secondMax = max2;
*secondMin = min2;
}

int main() {
int arr[] = {10, 5, 20, 15, 25};
int n = sizeof(arr) / sizeof(arr[0]);
int secondMaxResult, secondMinResult;
findSecondMaxAndMin(arr, n, &secondMaxResult, &secondMinResult);

if (secondMaxResult == INT_MIN || secondMinResult == INT_MAX) {
printf("The array must have at least two unique elements.\n");
} else {
printf("Second maximum element: %d\n", secondMaxResult);
printf("Second minimum element: %d\n", secondMinResult);
}

return 0;
}
```



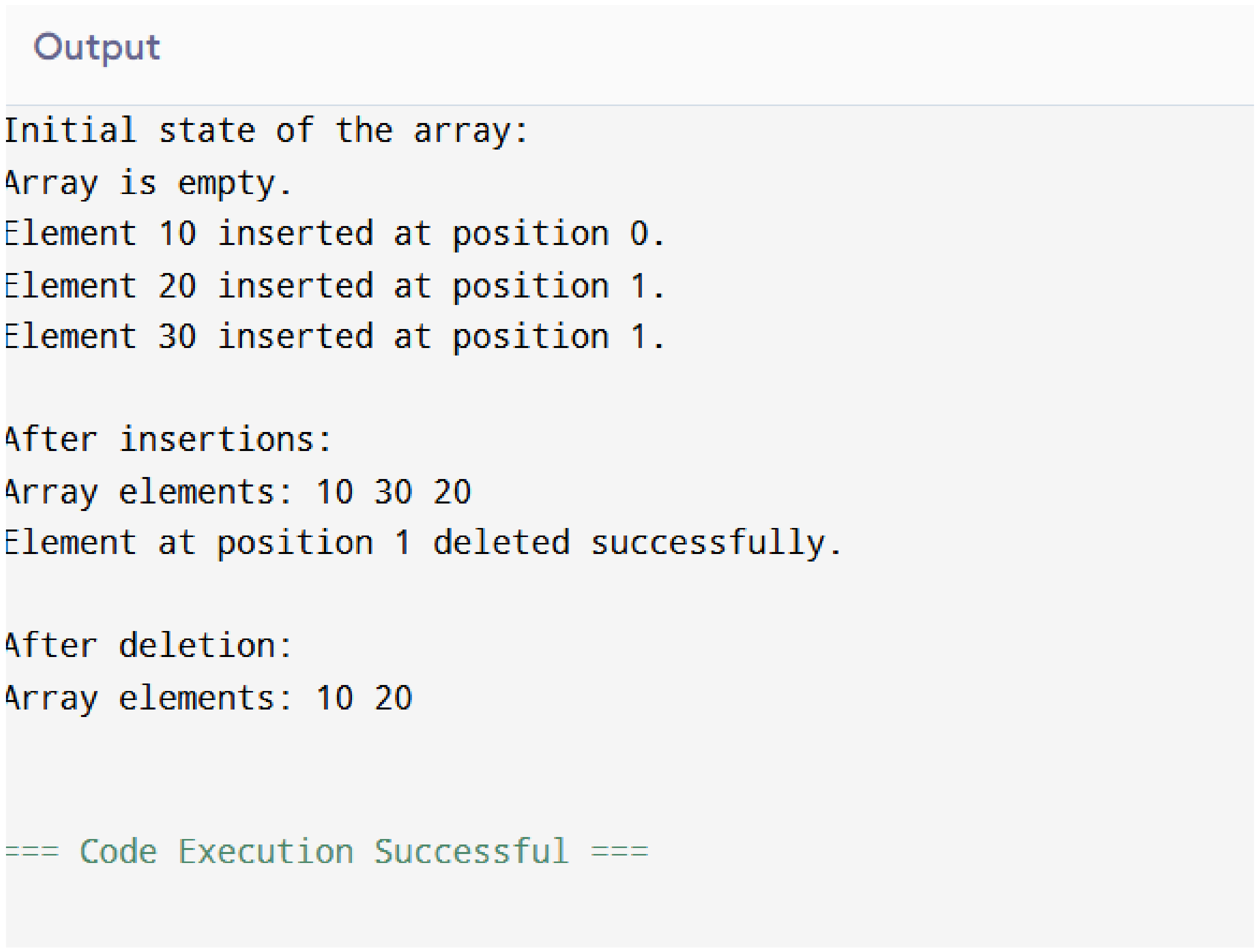
Output

```
Second maximum element: 20
Second minimum element: 10


=== Code Execution Successful ===
```

3.

```
#include <stdio.h>

// Traverses the array. The 'size' parameter is passed by address for consistency.
void traverseArray(int *arr, int *size) {
if (*size == 0) {
printf("Array is empty.\n");
return;
}
printf("Array elements: ");
for (int i = 0; i < *size; i++) {
printf("%d ", arr[i]);
}
printf("\n");
}
```

```
// Inserts an element. The 'size' is passed by address and modified.
void insertElement(int *arr, int *size, int element, int position, int capacity) {
if (*size >= capacity) {
printf("Error: Array is full. Insertion failed.\n");
return;
}
if (position < 0 || position > *size) {
printf("Error: Invalid position for insertion.\n");
return;
}

for (int i = *size - 1; i >= position; i--) {
arr[i + 1] = arr[i];
}
arr[position] = element;
(*size)++; // Increments the value at the size's memory address
printf("Element %d inserted at position %d.\n", element, position);
}

// Deletes an element. The 'size' is passed by address and modified.
void deleteElement(int *arr, int *size, int position) {
if (*size == 0) {
printf("Error: Array is empty. Deletion failed.\n");
return;
}
if (position < 0 || position >= *size) {
printf("Error: Invalid position for deletion.\n");
return;
}

for (int i = position; i < *size - 1; i++) {
arr[i] = arr[i + 1];
}
(*size)--; // Decrements the value at the size's memory address
printf("Element at position %d deleted successfully.\n", position);
}

int main() {
int capacity = 10;
int arr[capacity];
int size = 0; // The current size of the array

printf("Initial state of the array:\n");
traverseArray(arr, &size);
insertElement(arr, &size, 10, 0, capacity);
insertElement(arr, &size, 20, 1, capacity);
insertElement(arr, &size, 30, 1, capacity);

printf("\nAfter insertions:\n");
traverseArray(arr, &size);

deleteElement(arr, &size, 1);

printf("\nAfter deletion:\n");
traverseArray(arr, &size);

return 0;
}
```



Output

```
Initial state of the array:
Array is empty.
Element 10 inserted at position 0.
Element 20 inserted at position 1.
Element 30 inserted at position 1.

After insertions:
Array elements: 10 30 20
Element at position 1 deleted successfully.

After deletion:
Array elements: 10 20


=== Code Execution Successful ===
```