Lab Experiment No. 1

Objective

To implement and analyze various operations on arrays such as **insertion, deletion, searching, traversing, and updation** using C++.

T-1.1 Insertion
T-1.2 Deletion
T-1.3 Searching
T-1.4 Traversing
T-1.5 Updation

Theory

An array is one of the most fundamental and widely used data structures in computer science. It is a collection of elements stored in contiguous memory locations. Each element in an array is identified by an index or a key, which represents its position in the array.

Advantages of Arrays

1. **Fast Access**:
    - Elements can be accessed in O(1) time using their indices.
2. **Memory Efficiency**:
    - Arrays use contiguous memory, which reduces memory overhead.
3. **Ease of Implementation**:
    - Arrays are simple to implement and use in most programming languages.
4. **Cache Friendliness**:
    - Contiguous memory allocation improves cache performance, making arrays faster for sequential access.

Operations on Arrays

1. **Insertion**: Adding an element at a specific position.
2. **Deletion** : Removing an element from a specific position.
3. **Searching**: Finding an element in the array.
4. **Traversing**: Visiting and displaying all elements.
5. **Updation**: Modifying an element at a given index.

Algorithm & Implementation

T-1.1. Insertion

Algorithm:

1. Check if the array is full.
2. Shift elements to the right from the insertion position.
3. Insert the new element.
4. Increase array size.

T-1.2. Deletion

Algorithm:

1. Check if the array is empty.
2. Shift elements to the left from the deletion position.
3. Reduce the array size.

T-1.3. Searching

Algorithm:

1. Traverse the array.
2. If the element is found, return the index.
3. If not found, return -1.

T-1.4. Traversing

Algorithm:

1. Start from the first index.
2. Visit and display each element.

T-1.5. Updation

Algorithm:

1. Check if the index is valid.
2. Replace the element at the given index.

Code

#include <iostream> using namespace std;	// Function to update an element at a specific position

```
// Function to traverse and display array elements
void traverse(int arr[], int n) {
    cout << "Array elements: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

// Function to insert an element at a specific position
void insert(int arr[], int& n, int element, int position, int capacity) {
    if (n >= capacity) {
        cout << "Insertion failed: Array is full." << endl;
        return;
    }
    if (position < 0 || position > n) {
        cout << "Insertion failed: Invalid position." << endl;
        return;
    }
    for (int i = n; i > position; i--) {
        arr[i] = arr[i - 1];
    }
    arr[position] = element;
    n++;
    cout << "Inserted " << element << " at position " << position << "." << endl;
}

// Function to delete an element from a specific position
void remove(int arr[], int& n, int position) {
    if (position < 0 || position >= n) {
        cout << "Deletion failed: Invalid position." << endl;
        return;
    }
    cout << "Deleted element " << arr[position] << " from position " << position << "." << endl;
    for (int i = position; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    n--;
}

void update(int arr[], int n, int position, int newElement) {
    if (position < 0 || position >= n) {
        cout << "Update failed: Invalid position." << endl;
        return;
    }
    cout << "Updated position " << position << " from " << arr[position] << " to " << newElement << "." << endl;
    arr[position] = newElement;
}

int main() {
    const int capacity = 100; // Maximum capacity of the array
    int arr[capacity];
    int n = 0; // Current number of elements in the array

    // Inserting elements
    insert(arr, n, 10, 0, capacity);
    insert(arr, n, 20, 1, capacity);
    insert(arr, n, 30, 2, capacity);
    insert(arr, n, 40, 3, capacity);
    insert(arr, n, 50, 4, capacity);

    // Traversing array
    traverse(arr, n);

    // Deleting an element
    remove(arr, n, 2);
    traverse(arr, n);

    // Searching for an element
    int pos = search(arr, n, 40);
    if (pos != -1) {
        cout << "Element 40 found at position " << pos << "." << endl;
    } else {
        cout << "Element 40 not found." << endl;
    }

    // Updating an element
    update(arr, n, 1, 25);
    traverse(arr, n);

    return 0;
```

```
// Function to search for an element and return its position
int search(int arr[], int n, int element) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == element) {
            return i;
        }
    }
    return -1; // Element not found
}
```

Sample Output

```
Inserted 10 at position 0.
Inserted 20 at position 1.
Inserted 30 at position 2.
Inserted 40 at position 3.
Inserted 50 at position 4.
Array elements: 10 20 30 40 50
Deleted element 30 from position 2.
Array elements: 10 20 40 50
Element 40 found at position 2.
Updated position 1 from 20 to 25.
Array elements: 10 25 40 50
```

Complexity Analysis

Operation	Best Case	Average Case	Worst Case
Insertion	O(1)	O(n)	O(n)
Deletion	O(1)	O(n)	O(n)
Searching	O(1)	O(n)	O(n)
Traversing	O(n)	O(n)	O(n)
Updation	O(1)	O(1)	O(1)

Conclusion

We successfully implemented and analyzed various **array operations**. Arrays are a simple yet powerful data structure that provides fast access to elements and is widely used in programming.