# Array Deletion

---

## 🧠 1️⃣ Deletion at the Beginning

### 🔷 Algorithm

1. Input array elements and its size `n`.
2. To delete from beginning, shift all elements one step left.
3. Decrease `n` by 1.

---

### 🔷 C++ Code

```
#include <iostream>
using namespace std;

int main() {
    int arr[100], n;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    // Deletion from beginning
    for (int i = 0; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    n--; // reduce size
    cout << "Array after deletion from beginning: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

---

## 🧠 2️⃣ Deletion at the End

### 🔷 Algorithm

1. Input array and its size `n`.
2. To delete last element, just reduce size by 1 (`n--`).
3. No shifting needed.

---

### 🔷 C++ Code

```
#include <iostream>
using namespace std;

int main() {
    int arr[100], n;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    // Deletion from end
    n--; // simply reduce size
    cout << "Array after deletion from end: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

## 🧠 3️⃣ Deletion from a Specific Position (Index-Based)

### 🔷 Algorithm

1. Input array and size `n`.
2. Input the position `pos` to delete (0-based index).
3. Check if `pos` is valid (`pos >= 0 && pos < n`).
4. Shift elements from `pos` to `n-1` one step left.
5. Decrease size `n` by 1.

### 🔷 C++ Code

```
#include <iostream>
using namespace std;

int main() {
    int arr[100], n, pos;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    cout << "Enter position to delete (0-based index): ";
    cin >> pos;

    if (pos < 0 || pos >= n) {
        cout << "Invalid position!" << endl;
        return 0;
    }

    // Shift elements left
    for (int i = pos; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    n--;

    cout << "Array after deletion from position " << pos << ": ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

---

## 🧠 4️⃣ Deletion by Value

### 🔷 Algorithm

1. Input array and size `n`.
2. Input value `val` to delete.
3. Search for the value in array.
4. If found, store its index `pos`.
5. Shift all elements after `pos` one step left.

6. Decrease `n` by 1.
7. If not found, print “Value not found”.

---

## 🔷 C++ Code

```
#include <iostream>
using namespace std;

int main() {
    int arr[100], n, val, pos = -1;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    cout << "Enter value to delete: ";
    cin >> val;

    // Find value
    for (int i = 0; i < n; i++) {
        if (arr[i] == val) {
            pos = i;
            break;
        }
    }

    if (pos == -1) {
        cout << "Value not found!" << endl;
        return 0;
    }

    // Shift elements
    for (int i = pos; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    n--;

    cout << "Array after deleting value " << val << ": ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

---

## 🧮 Time Complexity Summary

| Type of Deletion | Description | Time Complexity |
|---|---|---|
| Beginning | Shift all elements | **O(n)** |
| End | Just reduce size | **O(1)** |
| Specific Position | Shift elements after position | **O(n)** |
| By Value | Search + Shift | **O(n)** |

---

## 🧩 Example Execution

**Input:**

```
5
10 20 30 40 50
```

**Delete position 2**

**Output:**

```
10 20 40 50
```