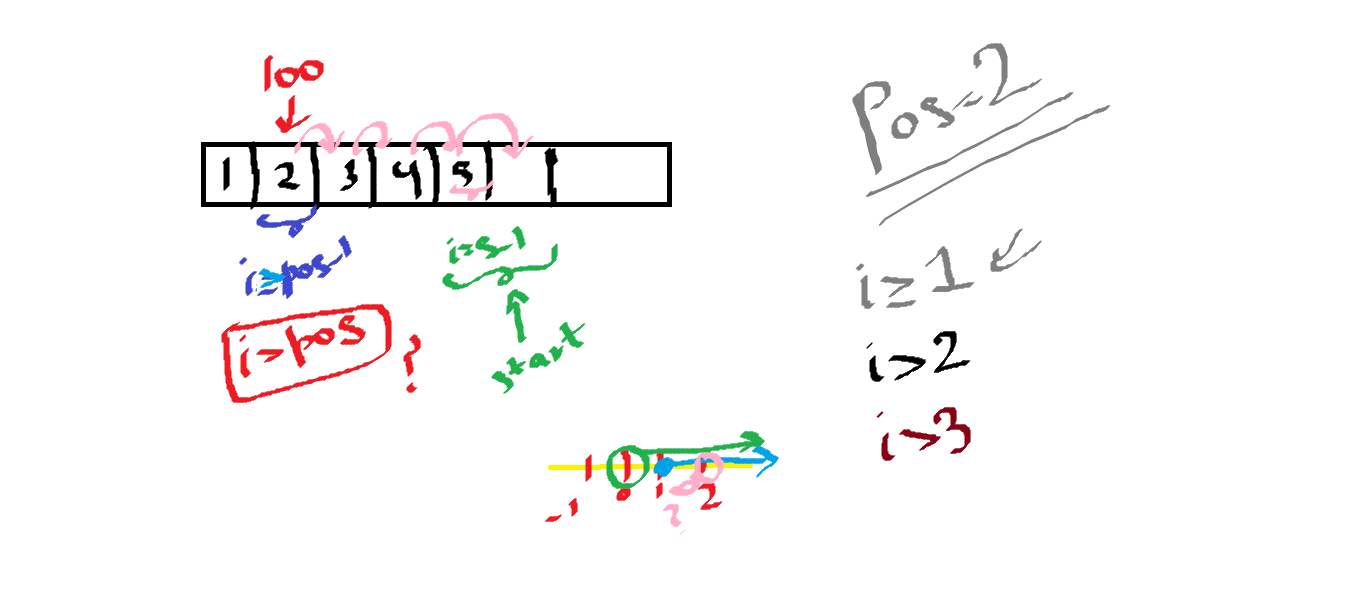
Shifting loop reason, why I >= pos – 1, and not i>pos or i>pos+1 ( see number line)



**COMPREHENSIVE GUIDE TO THE CODE 🡪**

**1. Input Function**

**Function:** void Input(int arr[], int Size)  
**Purpose:** Populate an array with user input.

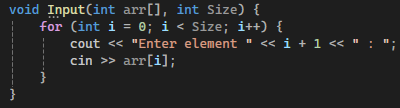
**Key Points:**

* Iterates from index 0 to Size-1.
* Reads integers via cin and stores them in arr[i].
* Assumes Size is valid (≥0 and ≤100).

**Trivia / Note:**

* Arrays in C++ have **fixed size**, so you need to ensure Size does not exceed the allocated memory (arr[100]).
* Inputting a negative size or a size exceeding array capacity can cause **undefined behavior** if not handled.

**Code snippet:**



## ****2. Traverse Function****

**Function:** void Traverse(int arr[], int Size)  
**Purpose:** Print array elements along with its current size.

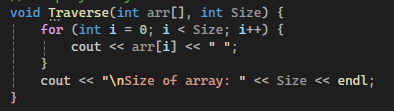
**Key Points:**

* Traverses from index 0 to Size-1.
* Prints elements in order.
* Prints current Size of the array for clarity.

**Trivia:**

* Even if Size = 0, this function handles it gracefully: prints nothing for elements and Size: 0.

**Code Snippet:**

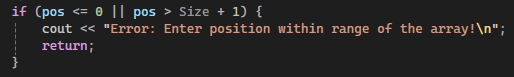


## ****3. Insert Function****

**Function:** void Insert(int arr[], int &Size)  
**Purpose:** Insert a new element at a given position in the array.

**Key Points / Steps:**

1. **Input:** Reads the element to insert and the position (pos).
2. **Position Validation:**

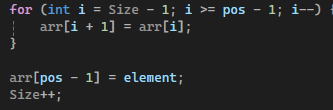


1. if (pos <= 0 || pos > Size + 1)
   * Must allow pos = Size + 1 for insertion at the end.
   * pos = 0 or negative is invalid.
   * **Trivia:** The +1 accounts for arrays being 0-indexed but users thinking in 1-based positions.
2. **Max Size Check:** Prevents insertion beyond array capacity (100).
3. **Shifting Elements:**
   * All elements from pos-1 to Size-1 are shifted right by 1 to make space.
4. **Insert Element:** Place element at arr[pos-1].
5. **Increment Size:** Update Size after insertion.

**Edge Cases / Exceptions:**

* Insert into **empty array (Size=0)** → must only allow pos=1.
* Insert at **end** → shifts no elements, simply adds at arr[Size].
* Insert when **array full** → blocked by Size >= 100.

**Snippet:**



**Trivia:**

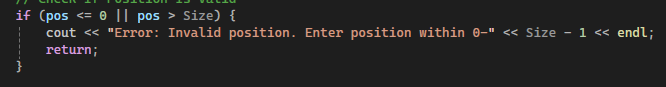
* If we checked pos > Size instead of pos > Size + 1, **insertion at the end would fail**.

## ****4. Delete Function****

**Function:** void Delete(int arr[], int &Size)  
**Purpose:** Delete an element from a given position in the array.

**Key Points / Steps:**

1. **Input:** Reads the position (pos) to delete.
2. **Position Validation:**



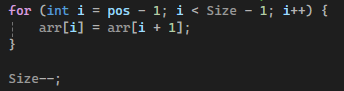
* + Position must be within [1, Size].
  + **No +1 here** because you **cannot delete beyond the current last element**.

1. **Empty Array Check:** If Size = 0, deletion is invalid.
2. **Shifting Elements:**
   * Shift all elements from pos to Size-1 **left by 1**, overwriting the deleted element.
3. **Decrement Size:** Reduce Size after deletion.

**Edge Cases / Exceptions:**

* Delete **first element (pos=1)** → shifts all elements left.
* Delete **last element (pos=Size)** → no shifting beyond array.
* Delete from **empty array** → handled with error message.

**Snippet:**



**Trivia:**

* **Important distinction:** Insertion allows pos = Size + 1 (new element at end) but deletion must **never exceed Size**.

FULL CODE 🡪

// Array insertion n deletion n traversal

#include<iostream>

using namespace std;

void Input(int arr[], int Size) {

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

cout << "Enter element " << i + 1 << " : ";

cin >> arr[i];

}

}

void Traverse(int arr[], int Size) {

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

cout << arr[i] << " ";

}

cout << "\nSize of array: " << Size << endl;

}

// pass by reference to change size of array in main too

void Insert(int arr[], int &Size) {

int element = 0, pos = 0;

//

cout << "Enter value you want to insert: ";

cin >> element;

cout << "Enter position where you want to enter " << element << " : ";

cin >> pos;

// Checking is Pos is in range

// Pos > Size + 1, not Size, to ensure insertion can be done at the end ( Pos = Size + 1)

if (pos <= 0 || pos > Size + 1) {

cout << "Enter Position within the range of the array !" << endl;

return;

}

// Checking if Max Size is reached

// equality ensures Max Size never exceeds 100

if (Size >= 100) {

cout << "Max Size (100) reached. No further insertions are possible " << endl;

return;

}

// Insertion logic by Shifting elements one cell from right to left

for (int i = Size - 1; i >= pos - 1;i--) {

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

}

arr[pos - 1] = element;

Size++;

}

// pass by reference to change size of array in main too

void Delete(int arr[], int &Size) {

int pos;

cout << "Enter position where you want to Delete: ";

cin >> pos;

// Checking is Pos is in range

if (pos <= 0 || pos > Size) {

cout << "ERROR. Invalid Position. Enter position within 0-" << Size - 1 << endl;

return;

}

// Checking if array is empty

if (Size == 0) {

cout << "Error. Can't perform deletion on empty list ! Returning" << endl;

return;

}

// No Max Size check as size is incremented in Insert() and decremented in Delete()

for (int i = pos - 1; i < Size - 1; i++) {

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

}

Size--;

}

int main() {

int arr[100];

int size = 0, element = 0, Insertpos = 0, Deletepos = 0;

cout << "Enter the size of the array: ";

cin >> size;

if (size > 100 || size < 0) {

cout << "Enter valid size" << endl;

return;

}

cout << "\nEnter the original array --> " << endl;

Input(arr, size);

cout << endl;

Traverse(arr, size);

cout << "\nInserting element into array -->\n" << endl;

Insert(arr, size);

cout << endl;

Traverse(arr, size);

cout << "\nDeleting element from array -->\n" << endl;

Delete(arr, size);

cout << endl;

Traverse(arr, size);

return 0;

}