CL2001 **Data Structure Lab**	**Lab 2** **1D and 2D Dynamic Safe Pointers and jagged array**

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2025

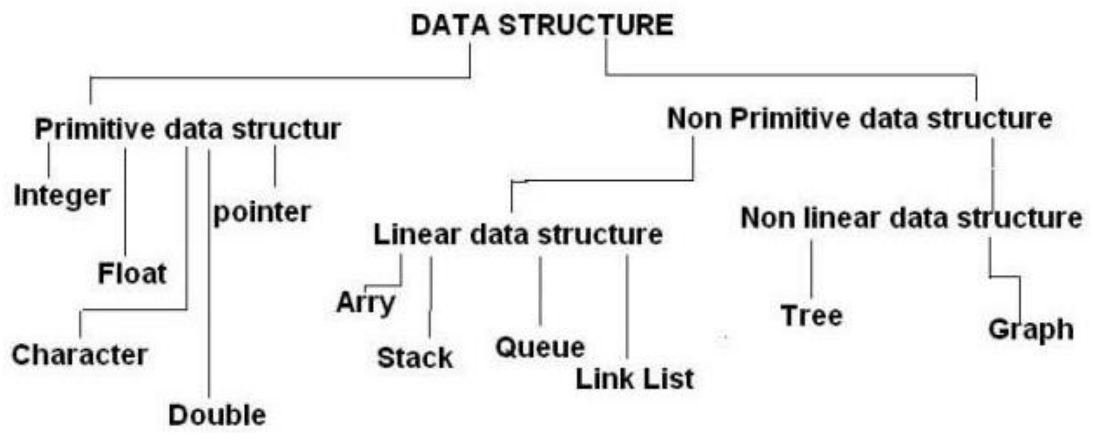
LAB MANUAL 02

Lab Content

1. 1D and 2D static Array
2. 1D and 2D Dynamic Arrays
3. Safe Arrays
4. Jagged Array

Data Structures

A data structure is a way of organizing and storing data in a computer so that it can be used efficiently. Think of it as a container that holds data in a particular arrangement so that operations like searching, inserting, deleting, or sorting can be done quickly. **Below is the classification of data structure:**



Linear Data Structures

A Linear Data Structure is a type of data structure in which data elements are arranged sequentially (one after another). For Example: Arrays, linked lists, stacks and queues. In a linear data structure, every element has a unique predecessor and a unique successor (except the first and last element).

Characteristics of Linear Data Structures:

1. **Sequential organization** - elements are stored in a sequence.
2. **Single level** - data flows in one dimension (like a straight line).
3. **Traversal** - elements are visited one by one.
4. **Memory allocation** - can be contiguous (**arrays**) or non-contiguous (**linked lists**)

Arrays

An array is a collection of elements of the same data type, stored in contiguous (continuous) memory locations. It allows us to store and access multiple values using a single variable name with an index.

Key Features of Arrays

- Stores multiple values of the same type.
- Elements are stored in contiguous memory.
- Index-based access (fast lookup).
  - Index starts from 0 (first element).
  - Last element index = size – 1.
- Fixed size (once created, size cannot change).

1D and 2D Arrays

1D array: a simple list of elements

int arr[5] = {10, 20, 30, 40, 50};

Memory:

Index: 0   1   2   3   4

Value: 10 20 30 40 50

2D array: Like a table (rows x columns)

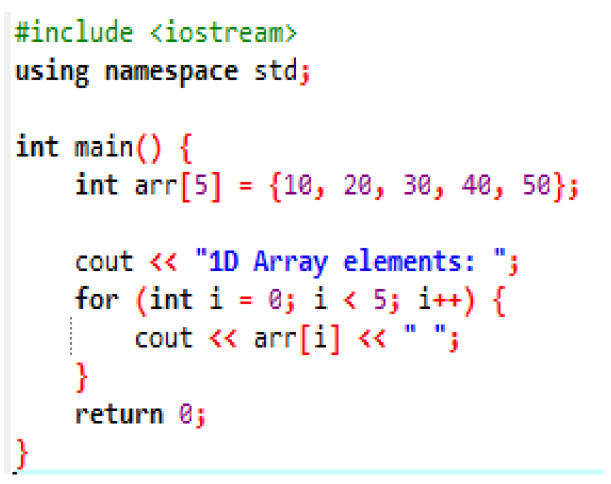
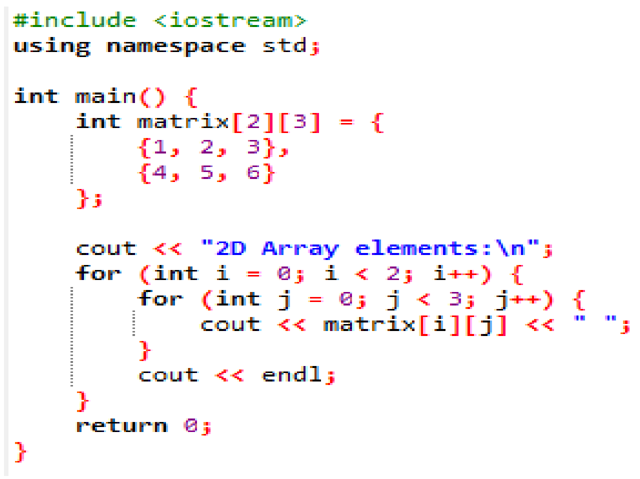
int matrix[2][3] = {{1, 2, 3},

{4, 5, 6}};

Memory:

Row 0 → [1 2 3]

Row 1 → [4 5 6]

1D array Example	2D array Example
	

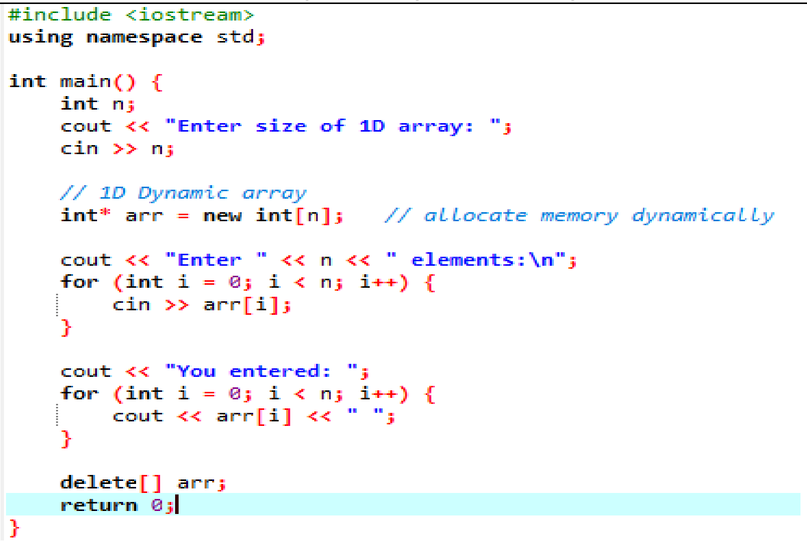
```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {10, 20, 30, 40, 50};

    cout << "1D Array elements: ";
    for (int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

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

int main() {
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    cout << "2D Array elements:\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Dynamic Arrays

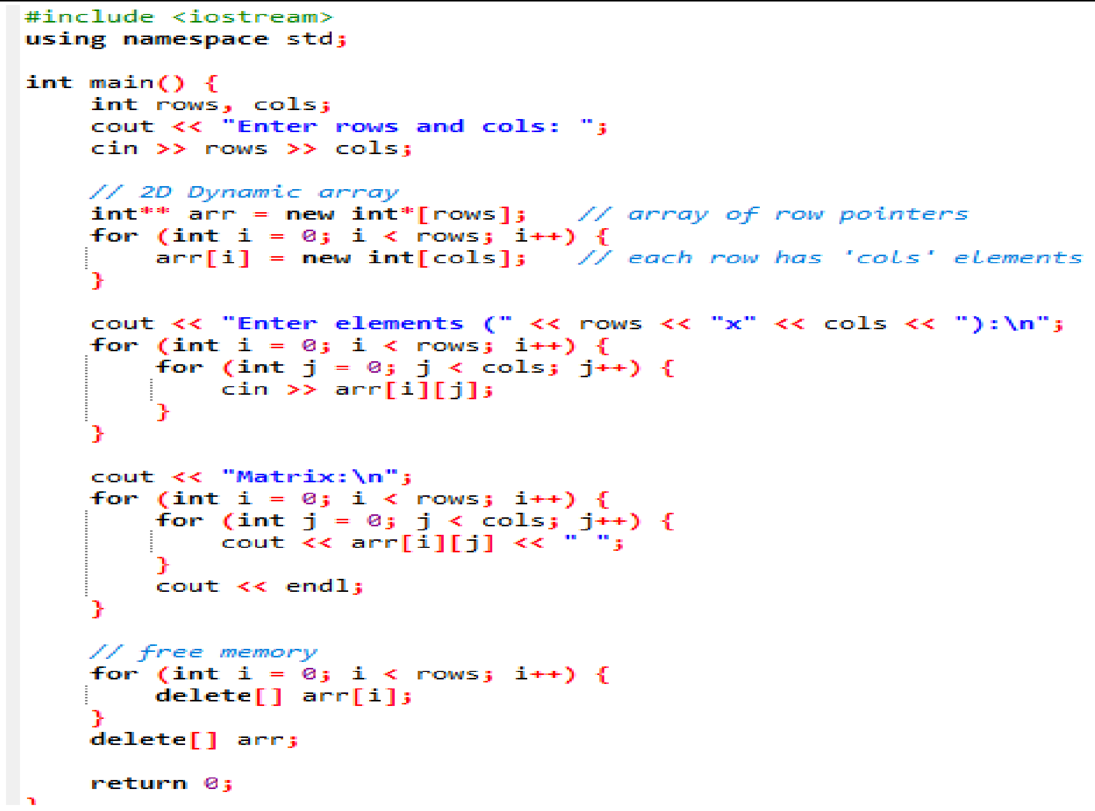
A 1D array is just a list of elements stored sequentially in memory. A dynamic array means the array size is not fixed at compile-time but is allocated at runtime using pointers and the new operator. Memory can be allocated and later freed using delete[ ].

Example 1D dynamic array



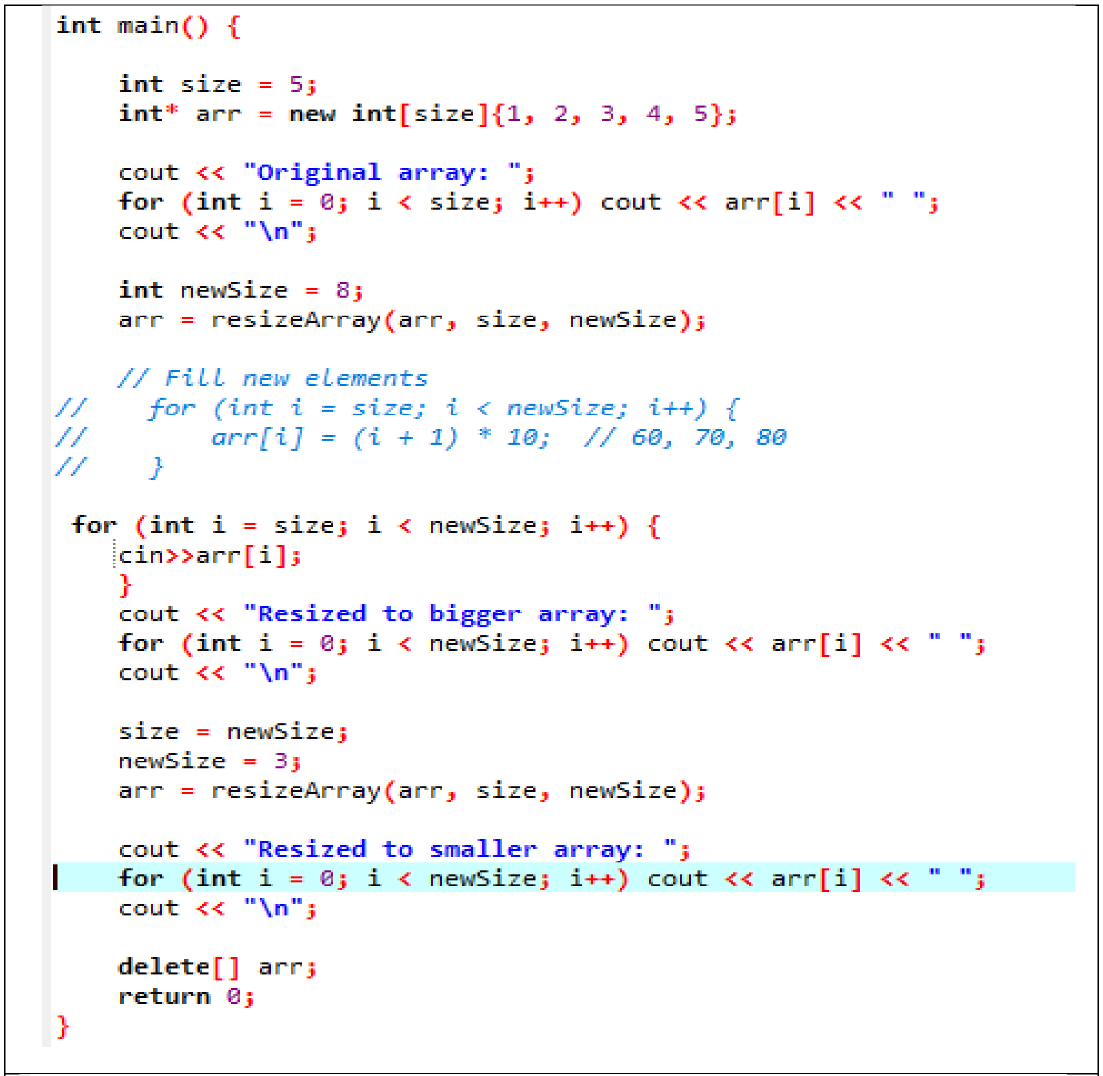
```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter size of 1D array: ";
    cin >> n;

    // 1D Dynamic array
    int* arr = new int[n];   // allocate memory dynamically

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

    cout << "You entered: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    delete[] arr;
    return 0;
}
```

2D Dynamic array

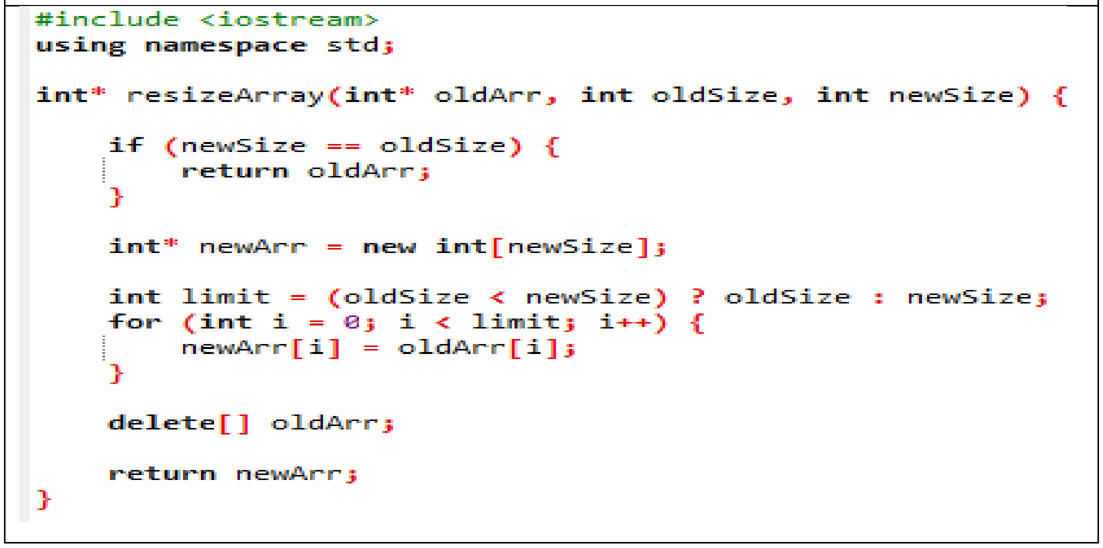


```
#include <iostream>
using namespace std;

int main() {
    int rows, cols;
    cout << "Enter rows and cols: ";
    cin >> rows >> cols;

    // 2D Dynamic array
    int** arr = new int*[rows];   // array of row pointers
    for (int i = 0; i < rows; i++) {
        arr[i] = new int[cols];   // each row has 'cols' elements
    }

    cout << "Enter elements (" << rows << "x" << cols << "):\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> arr[i][j];
        }
    }

    cout << "Matrix:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    // free memory
    for (int i = 0; i < rows; i++) {
        delete[] arr[i];
    }
    delete[] arr;

    return 0;
}
```

Resizing Dynamic Arrays



```
int main() {

    int size = 5;
    int* arr = new int[size]{1, 2, 3, 4, 5};

    cout << "Original array: ";
    for (int i = 0; i < size; i++) cout << arr[i] << " ";
    cout << "\n";

    int newSize = 8;
    arr = resizeArray(arr, size, newSize);

    // Fill new elements
//    for (int i = size; i < newSize; i++) {
//        arr[i] = (i + 1) * 10;  // 60, 70, 80
//    }

 for (int i = size; i < newSize; i++) {
    cin>>arr[i];
    }
    cout << "Resized to bigger array: ";
    for (int i = 0; i < newSize; i++) cout << arr[i] << " ";
    cout << "\n";

    size = newSize;
    newSize = 3;
    arr = resizeArray(arr, size, newSize);

    cout << "Resized to smaller array: ";
    for (int i = 0; i < newSize; i++) cout << arr[i] << " ";
    cout << "\n";

    delete[] arr;
    return 0;
}
```



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

int* resizeArray(int* oldArr, int oldSize, int newSize) {

    if (newSize == oldSize) {
        return oldArr;
    }

    int* newArr = new int[newSize];

    int limit = (oldSize < newSize) ? oldSize : newSize;
    for (int i = 0; i < limit; i++) {
        newArr[i] = oldArr[i];
    }

    delete[] oldArr;

    return newArr;
}
```

Safe Arrays

In C++, when you create a normal array (static or dynamic), there is no **automatic bounds** Checking.

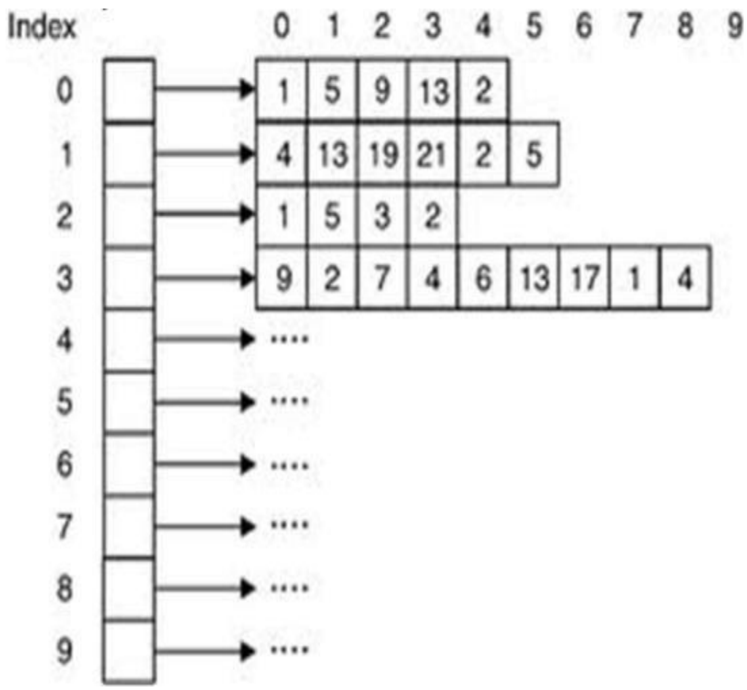
```
int arr[5];
arr[10] = 100;   // ? Undefined behavior (out of range)
```

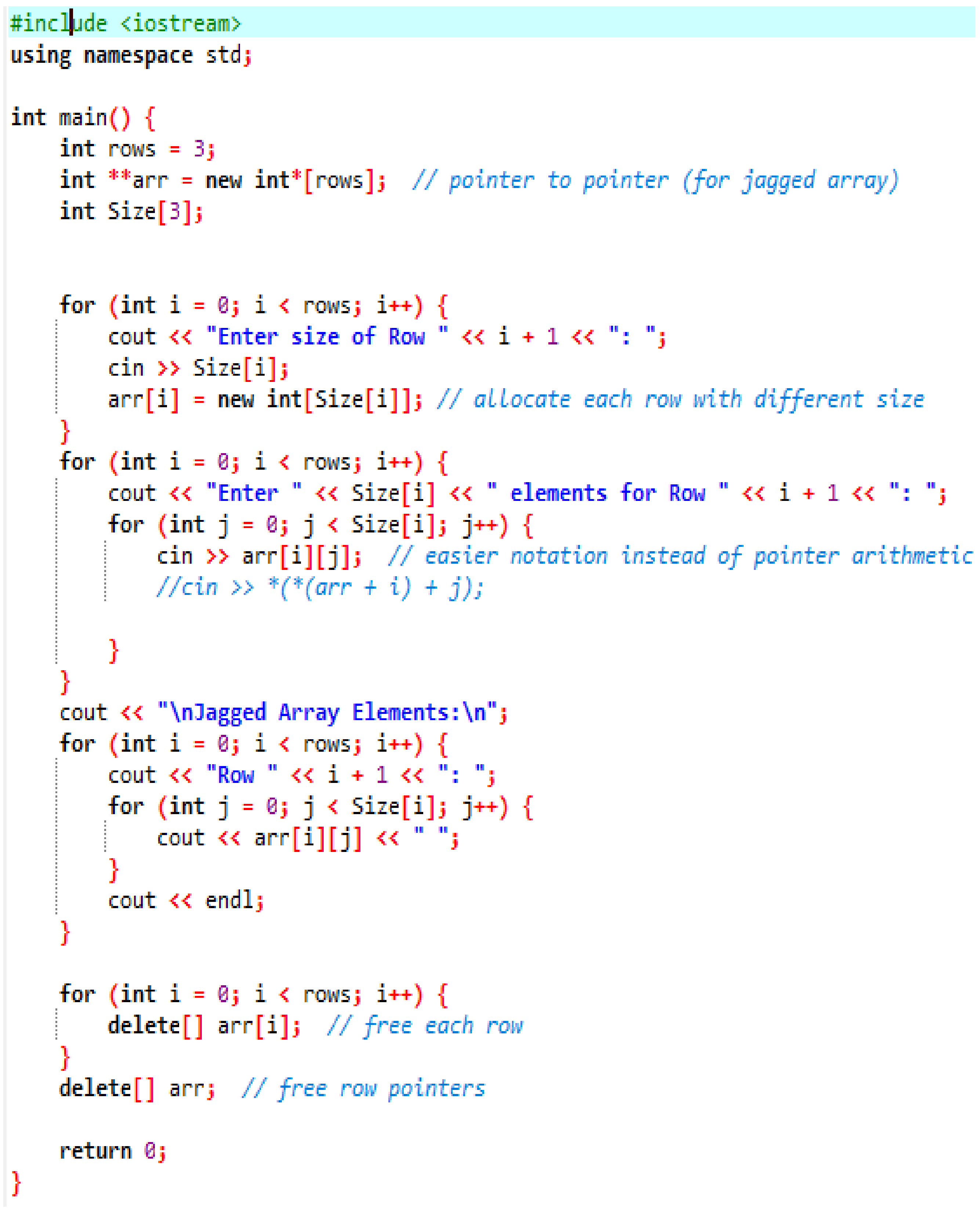
What will happen by executing above line?
Solution of above is safe array that validates the index before accessing or modifying elements. Prevents writing outside the valid range. Optionally allows custom index ranges (like starting from -5 instead of 0).
See Example below:

```
void set(int pos, Element val) {
    // check if position is valid
    if (pos < 0 || pos >= size) {
        cout << "Boundary Error\n";
    }
    else {
        Array[pos] = val;   // safe assignment
    }
}
```

Jagged Arrays

A Jagged Array is like a 2D array but each row can have a different number of columns.





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

int main() {
    int rows = 3;
    int **arr = new int*[rows];  // pointer to pointer (for jagged array)
    int Size[3];


    for (int i = 0; i < rows; i++) {
        cout << "Enter size of Row " << i + 1 << ": ";
        cin >> Size[i];
        arr[i] = new int[Size[i]]; // allocate each row with different size
    }
    for (int i = 0; i < rows; i++) {
        cout << "Enter " << Size[i] << " elements for Row " << i + 1 << ": ";
        for (int j = 0; j < Size[i]; j++) {
            cin >> arr[i][j];  // easier notation instead of pointer arithmetic
            //cin >> *(*(arr + i) + j);

        }
    }
    cout << "\nJagged Array Elements:\n";
    for (int i = 0; i < rows; i++) {
        cout << "Row " << i + 1 << ": ";
        for (int j = 0; j < Size[i]; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }

    for (int i = 0; i < rows; i++) {
        delete[] arr[i];  // free each row
    }
    delete[] arr;  // free row pointers

    return 0;
}
```

LAB TASKS

Task 1:

A teacher wants to record marks of 5 students in 4 subjects. Write a program to store them in a dynamic 2D array and calculate the average marks for each student.

Task 2:

A library allows members to borrow different numbers of books. Create a C++ program to store the borrowed books per member using a jagged array and display how many books each member borrowed.

Task 3:

Create a program to store votes for 5 candidates in a 1D array. After input, display the total votes for each candidate and declare the winner.

Task 4:

A supermarket needs to store the prices of N items purchased by a customer. Use a dynamic array to calculate the total bill and apply a 5% tax.

Task 5:

Create two dynamic matrices of size $N \times N$. Subtract them element by element and display the resulting matrix.