CSE 232

Programming with C++

Lecture 7

Array, Matrix, Vector and Pointers

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
Email: mijan@jkkniu.edu.bd
Web: www.mijanrahman.com

Contents

Array, Matrix, Vector and Pointers

ARRAY, MATRIX, AND VECTOR IN C++

- An array is a collection of elements of the same data type, stored contiguously in memory.
- In C++, a matrix isn't a built-in data structure but can be represented in various ways, such as 2D arrays and vectors. 2D array is an array of arrays, where each inner array represents a row.
- In C++, vector is a dynamic array-like data structure that can store elements of any data type.

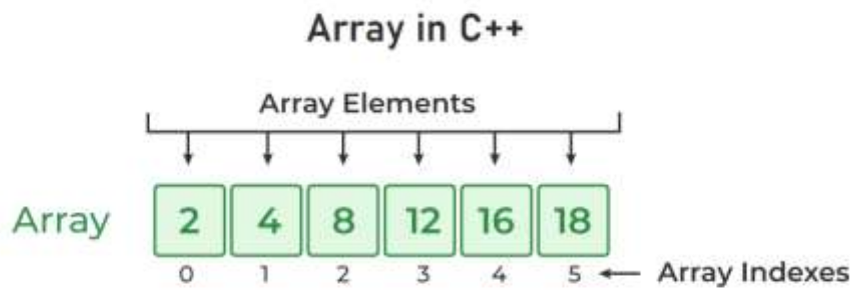
ARRAY, MATRIX, AND VECTOR IN C++

- **Key Features and Differences:**

Feature	Array	Matrix	Vector
Data Type	Same type for all elements	Can be individual elements of different types	Any data type
Size	Fixed at declaration	Depends on representation	Dynamically resizable
Access	Efficient by index	Depends on representation	Efficient by index
Use cases	Random access, fixed data	Linear algebra, image processing	Dynamic collections, storing various data

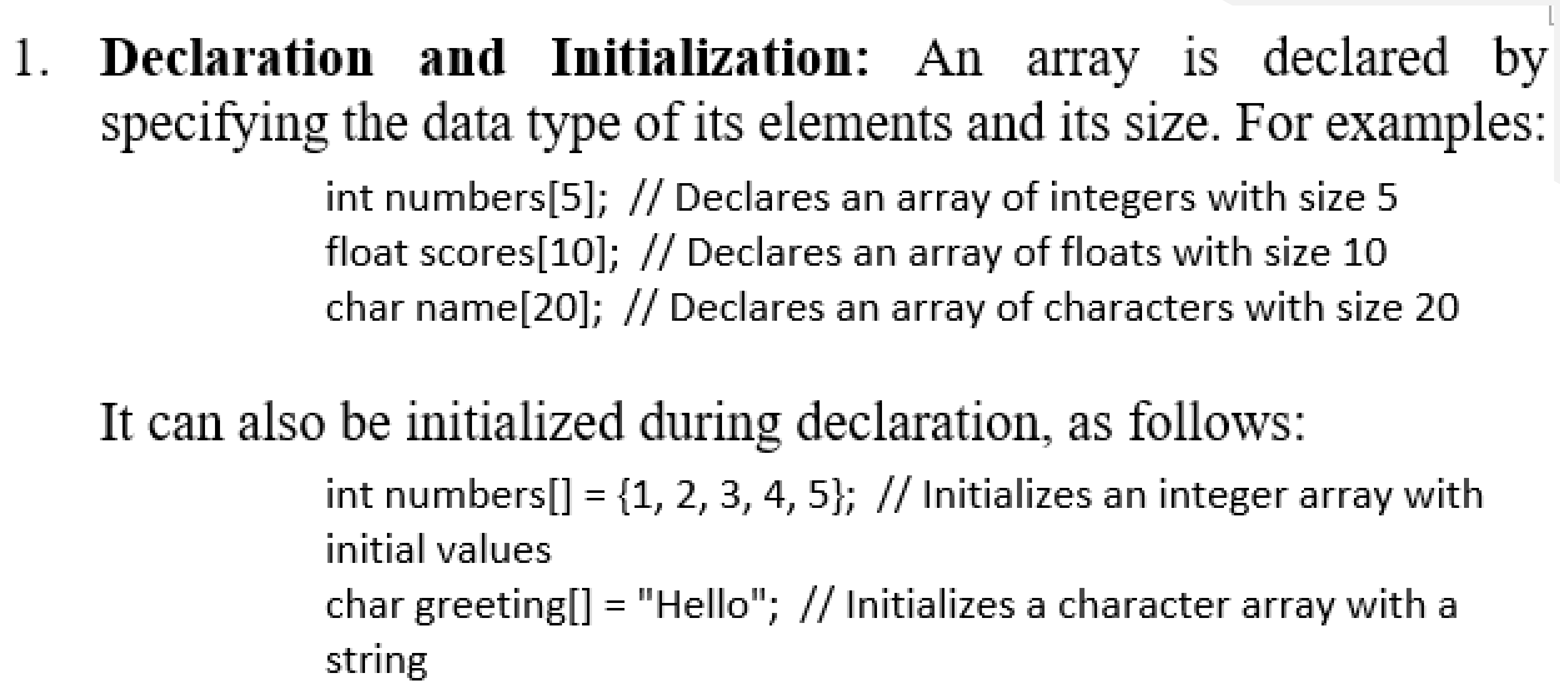
Array in C++

- In C++, an array is a fixed-size sequence of elements of the same data type. It provides a way to store and access multiple values of the same type using a contiguous block of memory.
- Arrays in C++ have a fixed size, which needs to be specified at the time of declaration. Once an array is created, its size cannot be changed.
- Additionally, arrays do not perform bounds checking, so it's essential to ensure that the index used to access an element is within the valid range of the array.



Array in C++

- The following is a brief overview of arrays in C++:



1. **Declaration and Initialization:** An array is declared by specifying the data type of its elements and its size. For examples:

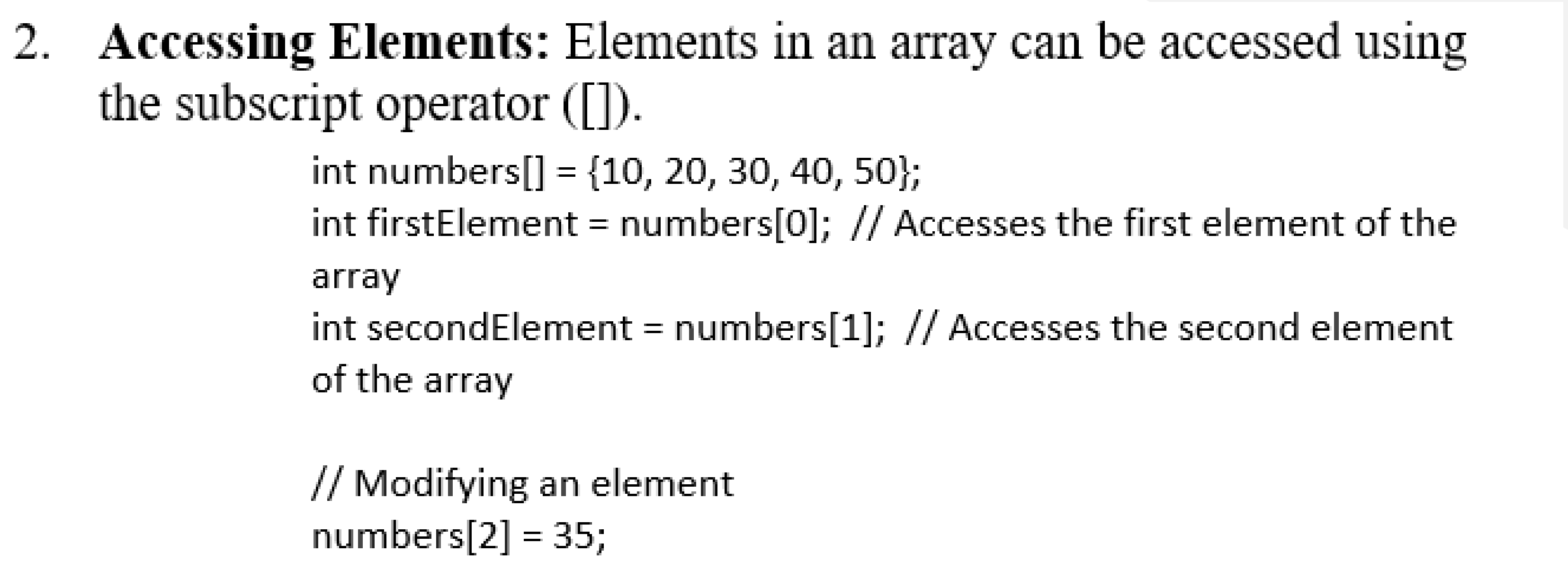
```
int numbers[5];  // Declares an array of integers with size 5
float scores[10];  // Declares an array of floats with size 10
char name[20];  // Declares an array of characters with size 20
```

It can also be initialized during declaration, as follows:

```
int numbers[] = {1, 2, 3, 4, 5};  // Initializes an integer array with
initial values
char greeting[] = "Hello";  // Initializes a character array with a
string
```

Array in C++

- The following is a brief overview of arrays in C++:

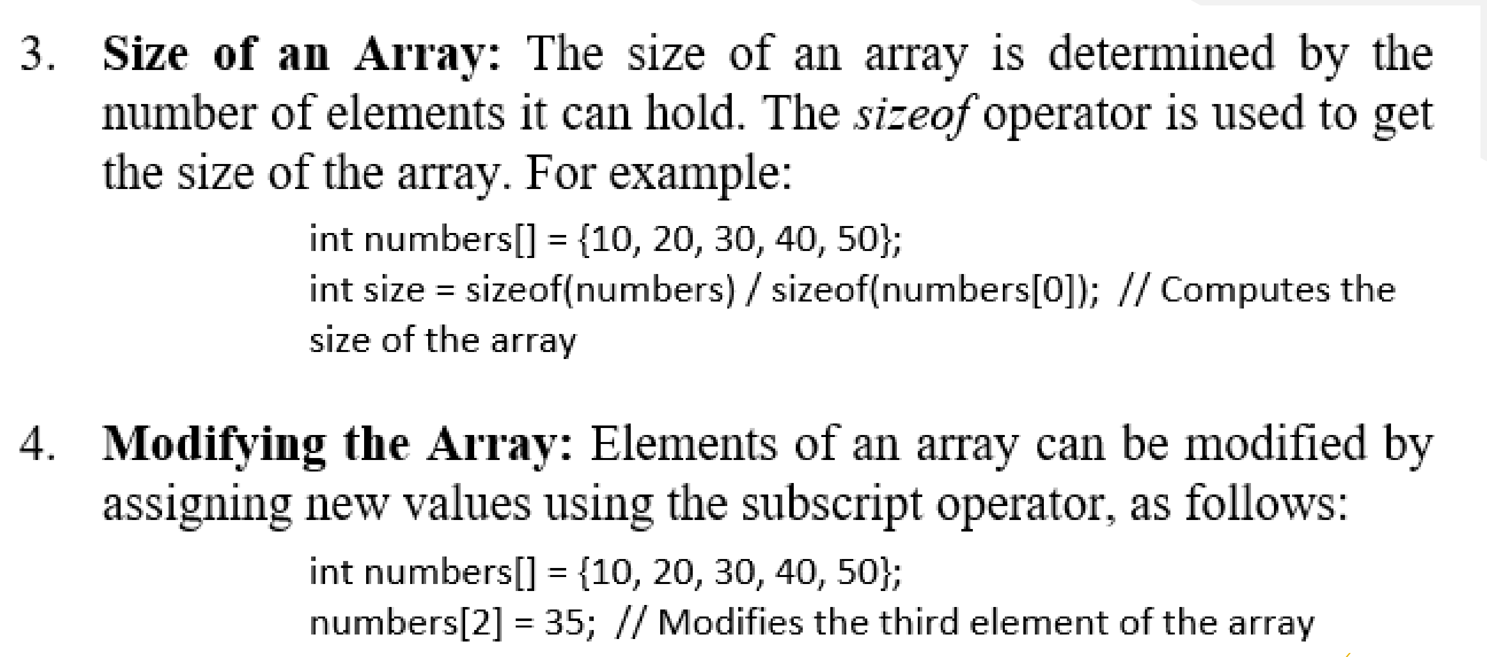


2. **Accessing Elements:** Elements in an array can be accessed using the subscript operator ([]).

```
int numbers[] = {10, 20, 30, 40, 50};
int firstElement = numbers[0];  // Accesses the first element of the
array
int secondElement = numbers[1];  // Accesses the second element
of the array

// Modifying an element
numbers[2] = 35;
```

Array in C++

- The following is a brief overview of arrays in C++:



3. **Size of an Array:** The size of an array is determined by the number of elements it can hold. The *sizeof* operator is used to get the size of the array. For example:

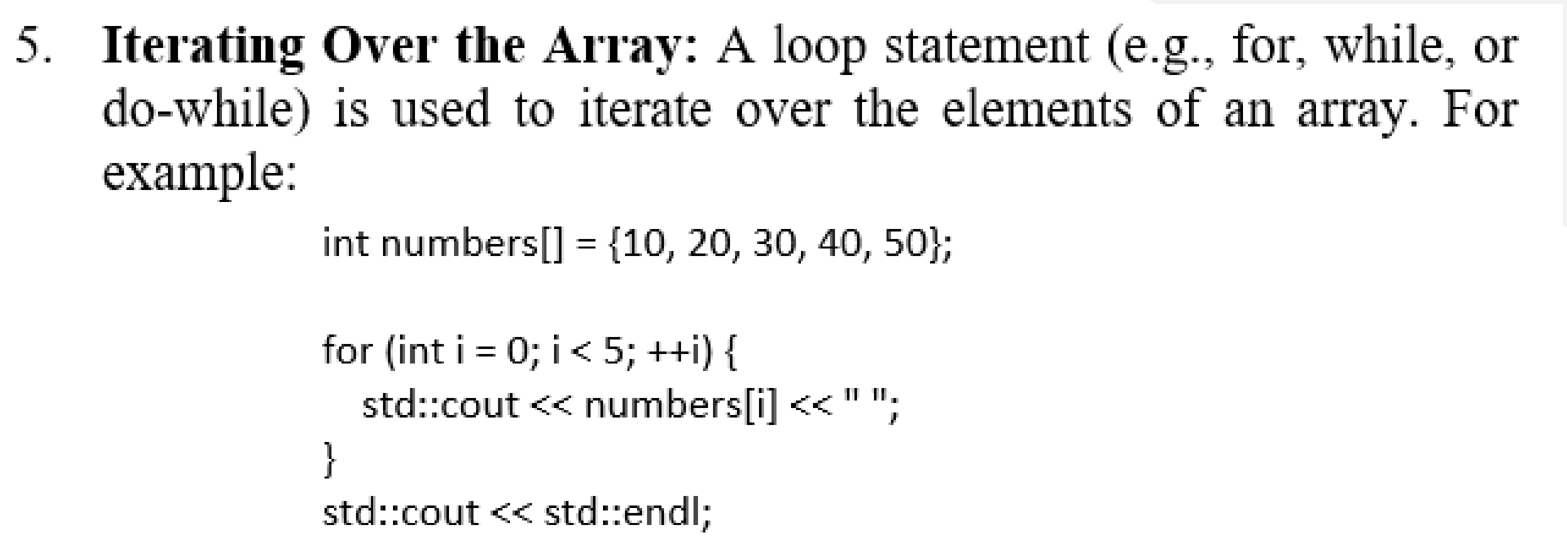
```
int numbers[] = {10, 20, 30, 40, 50};
int size = sizeof(numbers) / sizeof(numbers[0]);  // Computes the
size of the array
```

4. **Modifying the Array:** Elements of an array can be modified by assigning new values using the subscript operator, as follows:

```
int numbers[] = {10, 20, 30, 40, 50};
numbers[2] = 35;  // Modifies the third element of the array
```

Array in C++

- The following is a brief overview of arrays in C++:

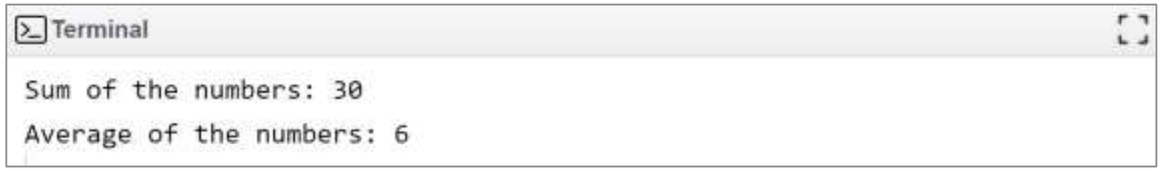


5. **Iterating Over the Array:** A loop statement (e.g., for, while, or do-while) is used to iterate over the elements of an array. For example:

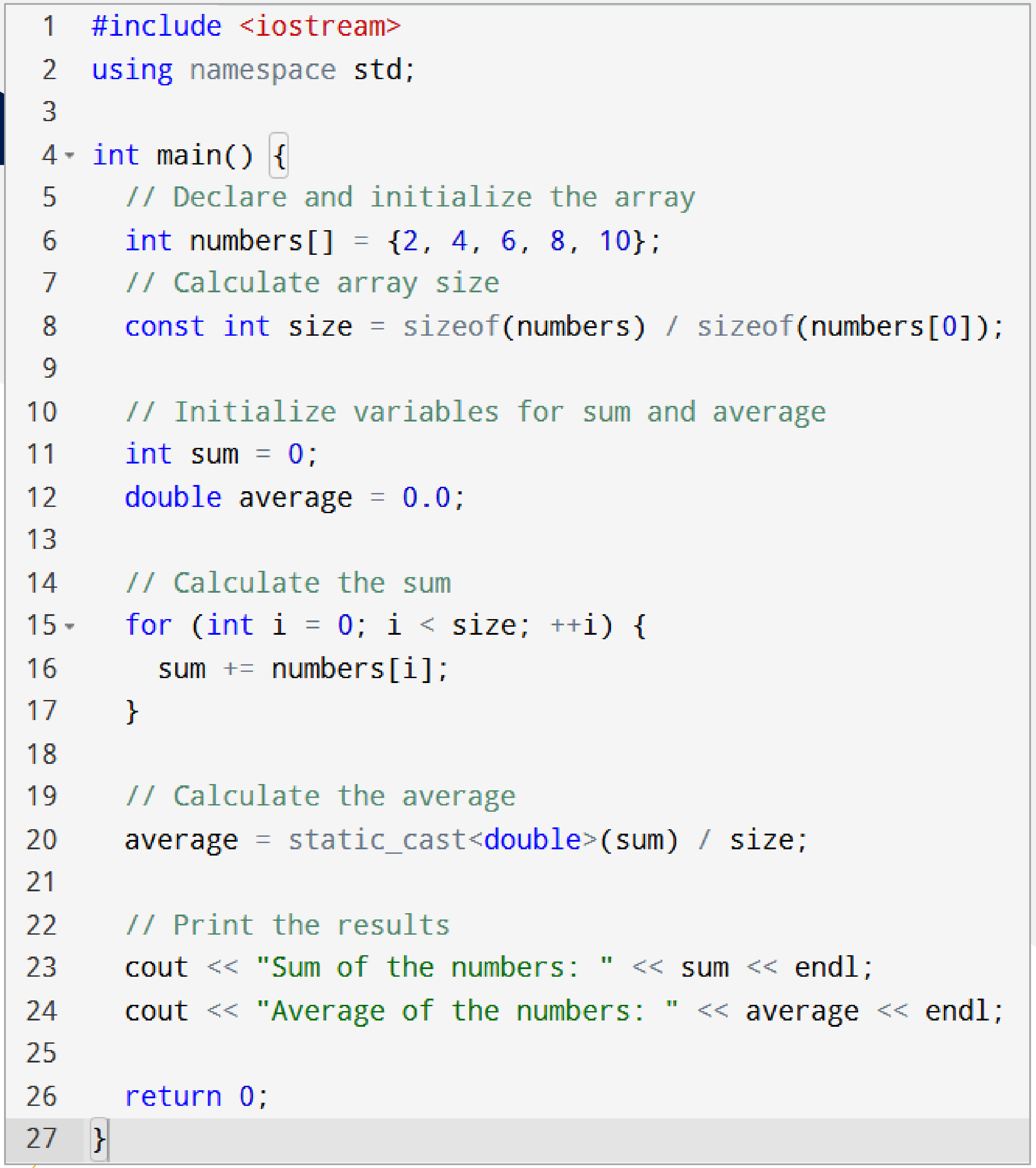
```
int numbers[] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; ++i) {
   std::cout << numbers[i] << " ";
}
std::cout << std::endl;
```

Array in C++

- A C++ program using arrays.



```
Terminal
Sum of the numbers: 30
Average of the numbers: 6
```



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

int main() {
  // Declare and initialize the array
  int numbers[] = {2, 4, 6, 8, 10};
  // Calculate array size
  const int size = sizeof(numbers) / sizeof(numbers[0]);

  // Initialize variables for sum and average
  int sum = 0;
  double average = 0.0;

  // Calculate the sum
  for (int i = 0; i < size; ++i) {
    sum += numbers[i];
  }

  // Calculate the average
  average = static_cast<double>(sum) / size;

  // Print the results
  cout << "Sum of the numbers: " << sum << endl;
  cout << "Average of the numbers: " << average << endl;

  return 0;
}
```

C++ Multidimensional Array

- A multidimensional array is an array with more than one dimension. It is the homogeneous collection of items where each element is accessed using multiple indices.
- Multidimensional Array Declaration:

  datatype arrayName[size1][size2]...[sizeN];

- where, datatype: Type of data to be stored in the array.

  arrayName: Name of the array.

  size1, size2,..., sizeN: Size of each dimension.

- Example:

  Two dimensional array: int two_d[2][4];

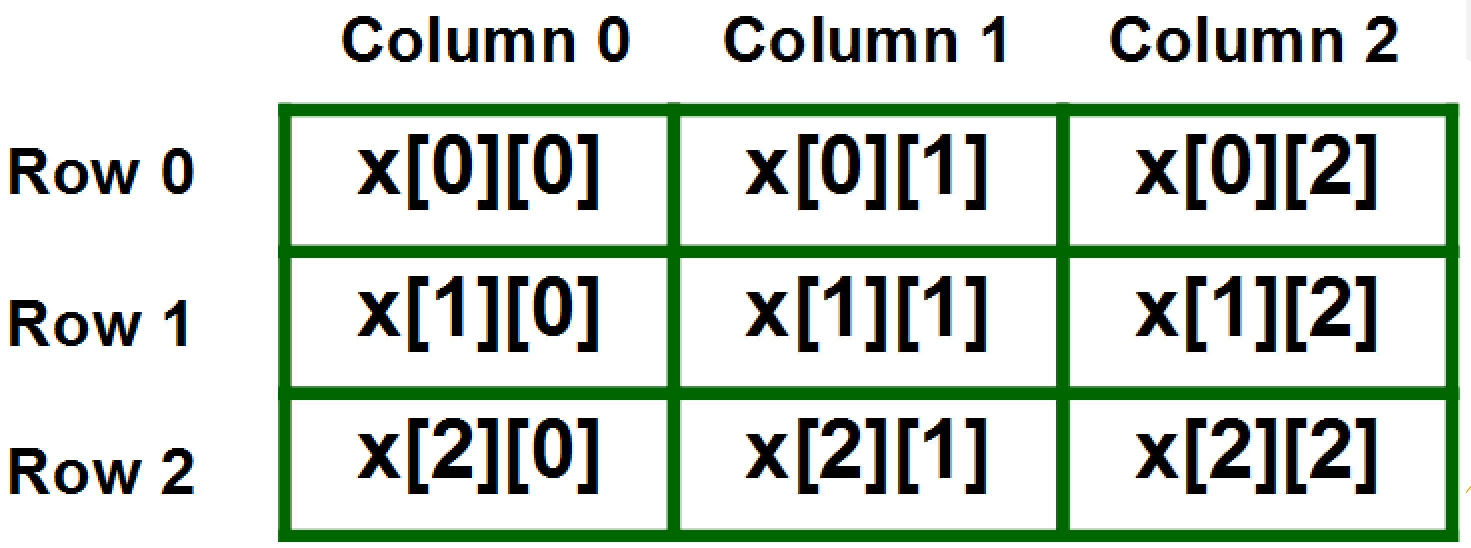
  Three dimensional array: int three_d[2][4][8];

Matrix (or 2D Array)

- In C++, a 2D array, also known as a matrix, is an array of arrays. It represents a table-like structure with rows and columns. Each element in the 2D array can be accessed using two indices: one for the row and another for the column.

- 2D arrays are useful for representing grids, matrices, and other tabular structures. Remember that 2D arrays have a fixed size, and each row can have a different number of columns.

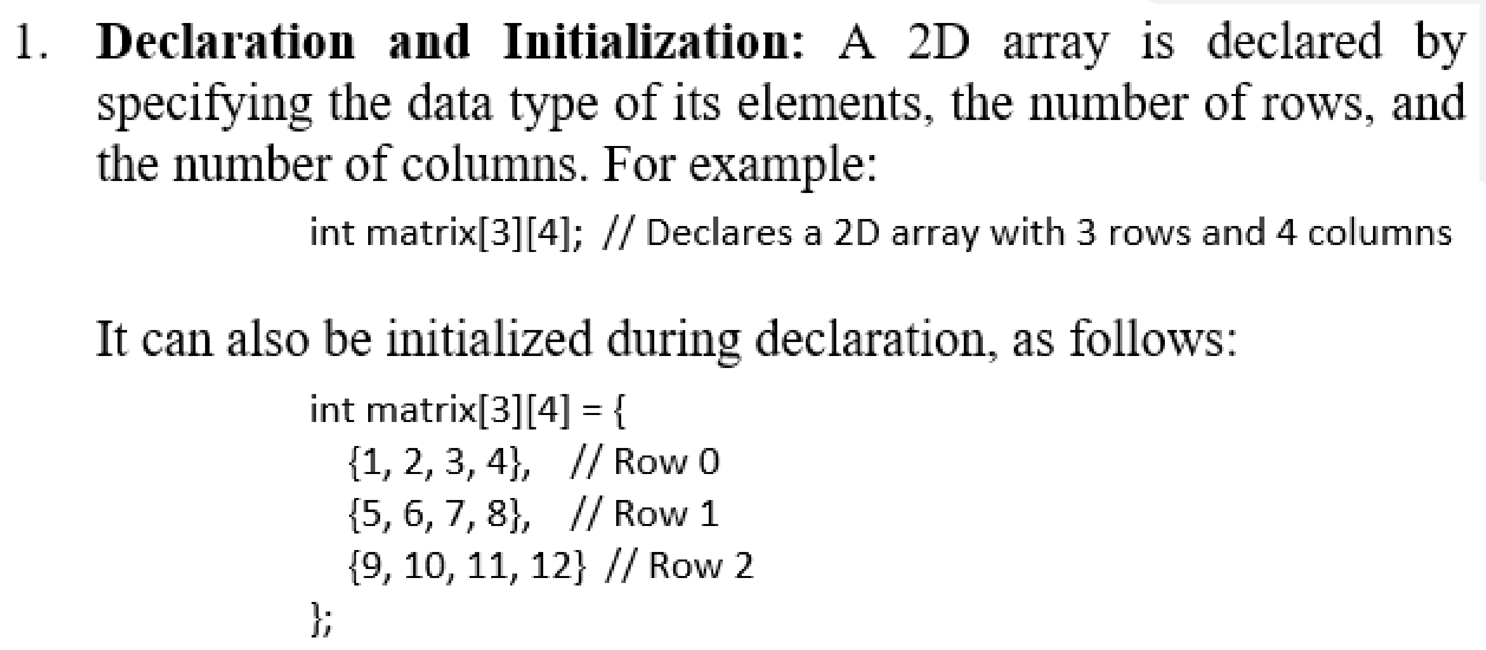
Matrix (or 2D Array)



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Matrix (or 2D Array)

- A brief overview of working with 2D arrays in C++ is given below:



1. **Declaration and Initialization:** A 2D array is declared by specifying the data type of its elements, the number of rows, and the number of columns. For example:

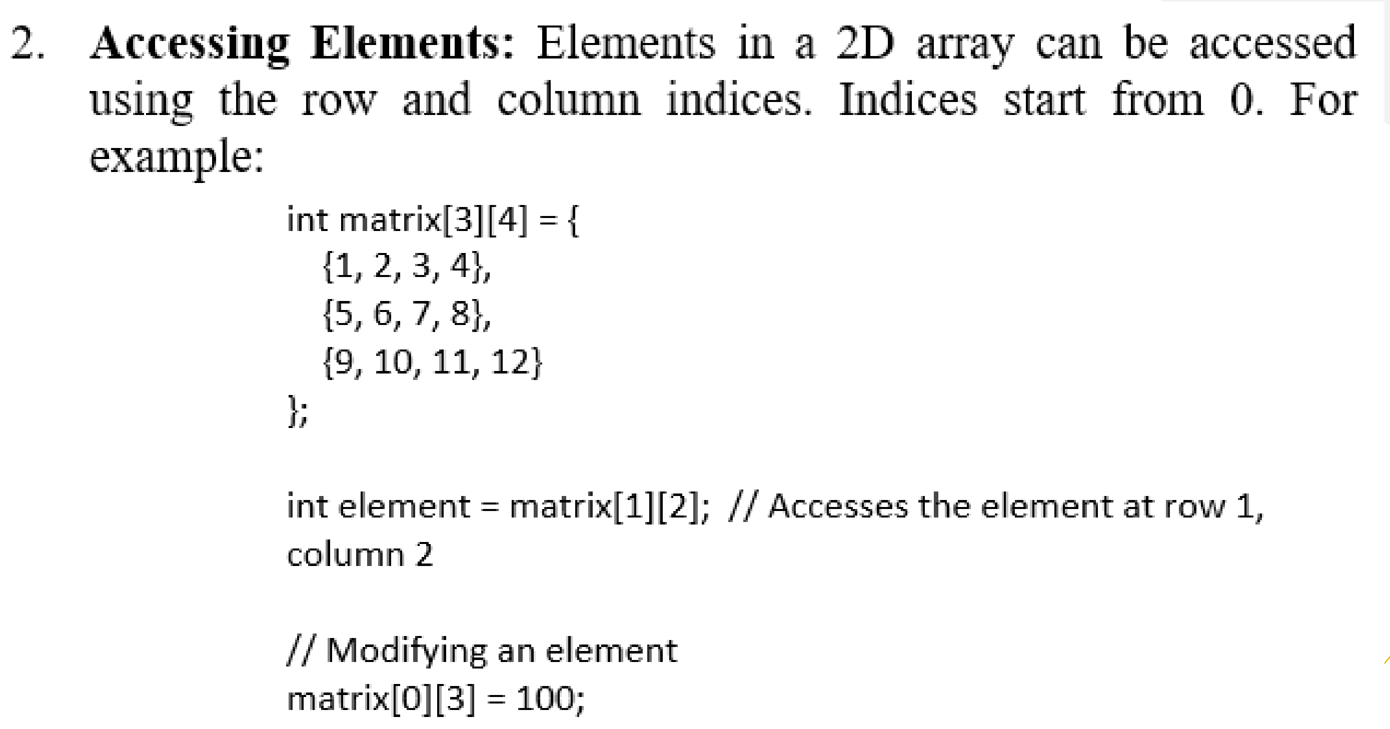
```
int matrix[3][4];  // Declares a 2D array with 3 rows and 4 columns
```

It can also be initialized during declaration, as follows:

```
int matrix[3][4] = {
   {1, 2, 3, 4},    // Row 0
   {5, 6, 7, 8},    // Row 1
   {9, 10, 11, 12}  // Row 2
};
```

Matrix (or 2D Array)

- A brief overview of working with 2D arrays in C++ is given below:

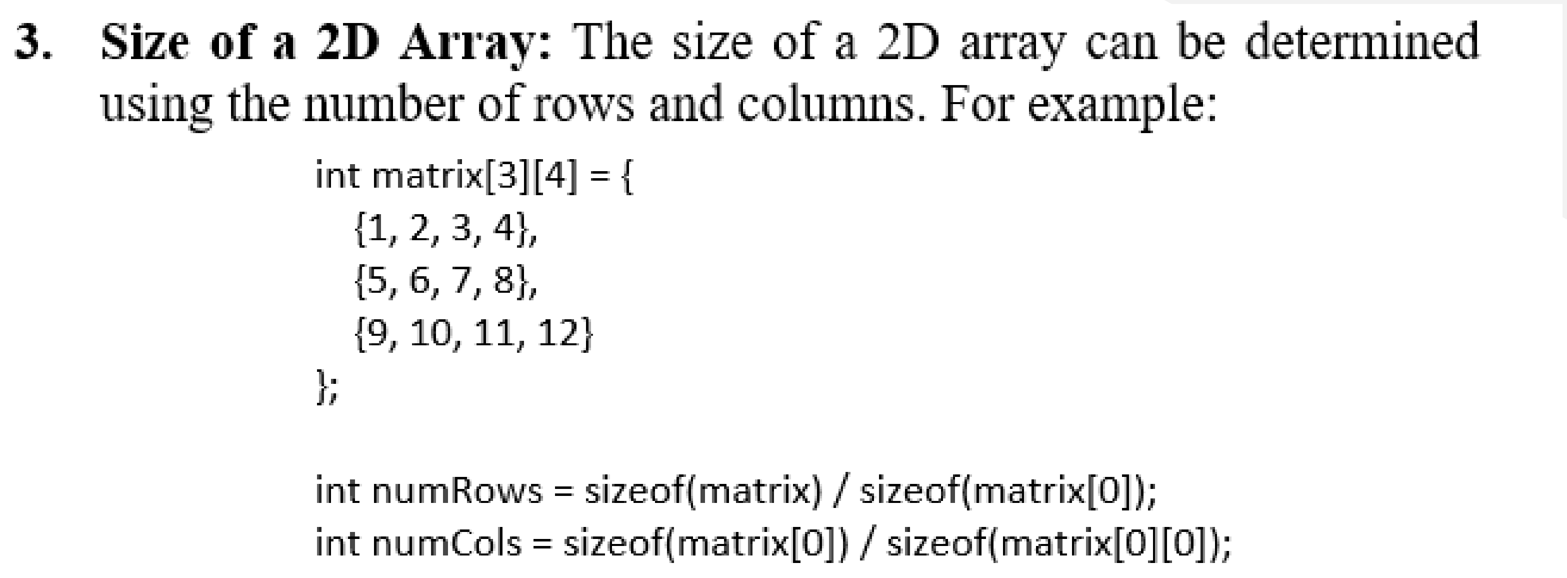


2. **Accessing Elements:** Elements in a 2D array can be accessed using the row and column indices. Indices start from 0. For example:

```
int matrix[3][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
};

int element = matrix[1][2];  // Accesses the element at row 1,
column 2

// Modifying an element
matrix[0][3] = 100;
```

Matrix (or 2D Array)

- A brief overview of working with 2D arrays in C++ is given below:

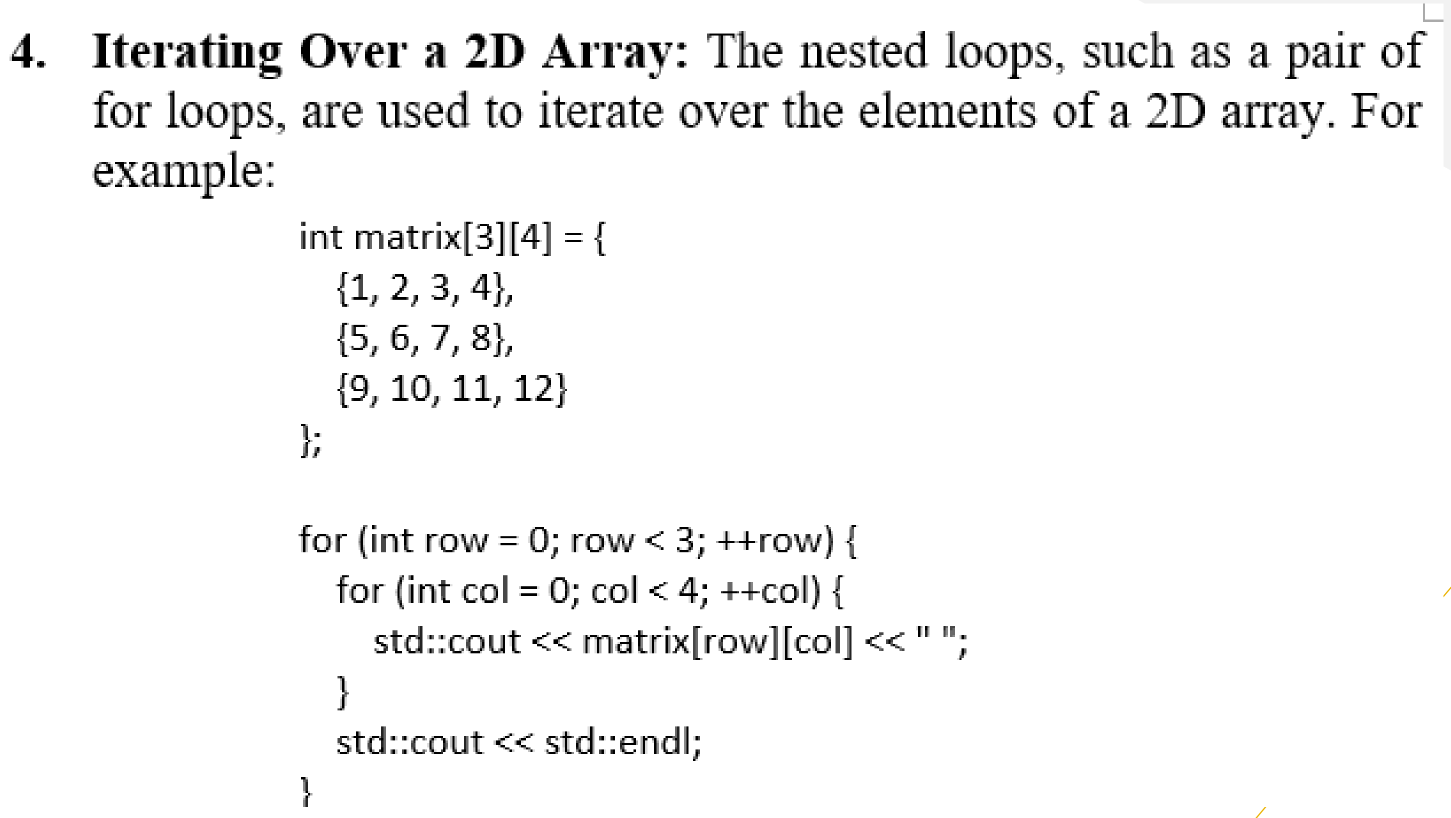


3. **Size of a 2D Array:** The size of a 2D array can be determined using the number of rows and columns. For example:

```
int matrix[3][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
};

int numRows = sizeof(matrix) / sizeof(matrix[0]);
int numCols = sizeof(matrix[0]) / sizeof(matrix[0][0]);
```

Matrix (or 2D Array)

- A brief overview of working with 2D arrays in C++ is given below:

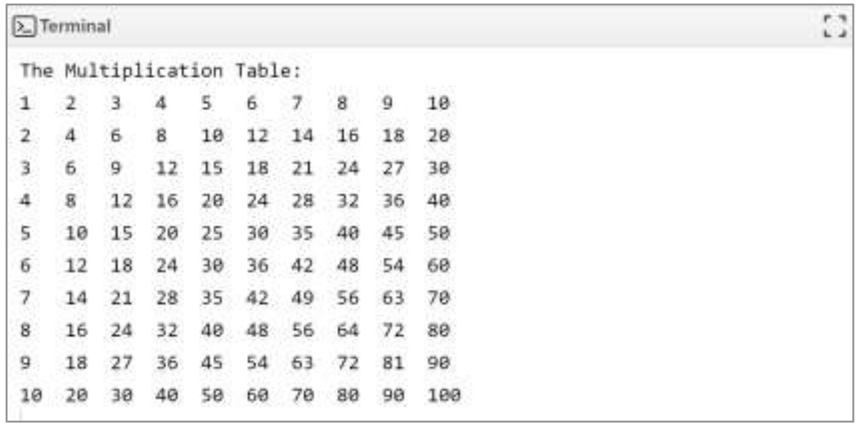


4. **Iterating Over a 2D Array:** The nested loops, such as a pair of for loops, are used to iterate over the elements of a 2D array. For example:

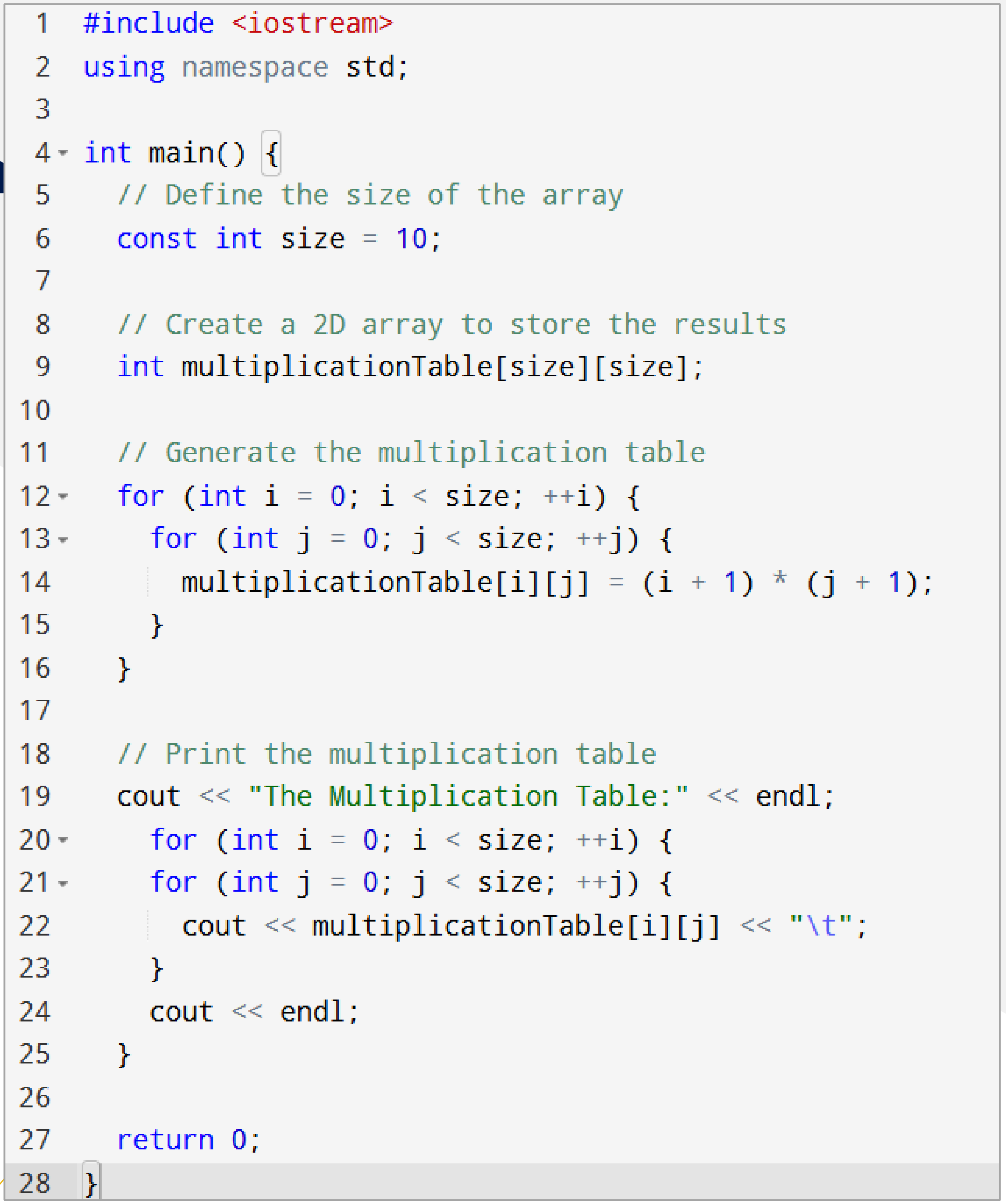
```
int matrix[3][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
};

for (int row = 0; row < 3; ++row) {
   for (int col = 0; col < 4; ++col) {
      std::cout << matrix[row][col] << " ";
   }
   std::cout << std::endl;
}
```

Matrix (or 2D Array)

- A C++ program using 2D array: This program uses a 2D array to represent a multiplication table



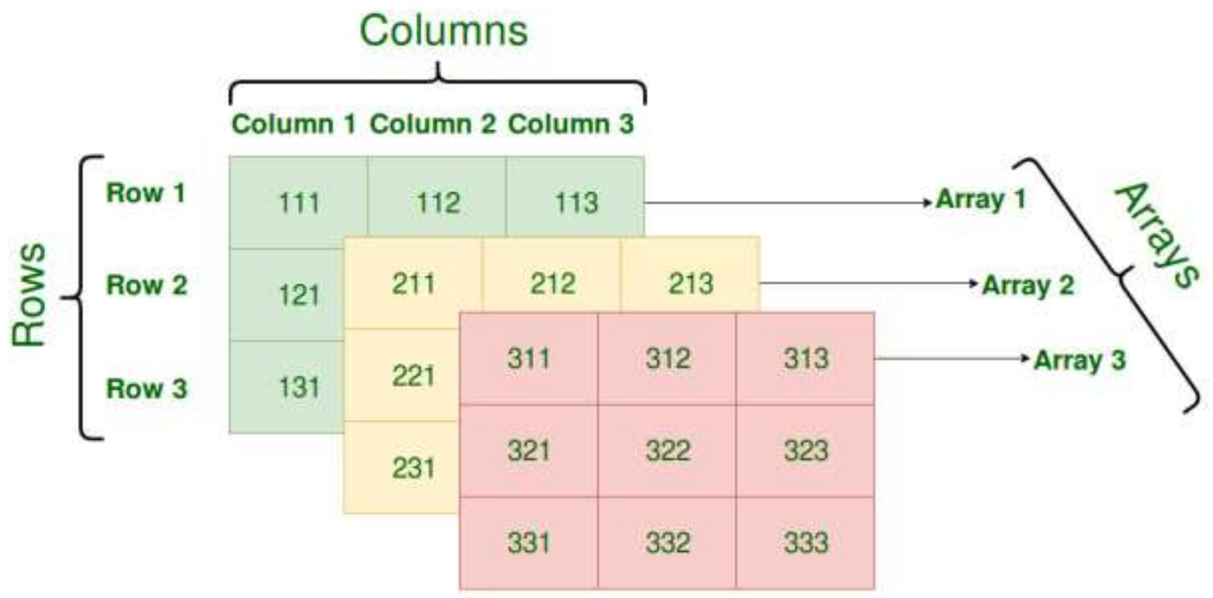
```
The Multiplication Table:
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100
```



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

int main() {
  // Define the size of the array
  const int size = 10;

  // Create a 2D array to store the results
  int multiplicationTable[size][size];

  // Generate the multiplication table
  for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
      multiplicationTable[i][j] = (i + 1) * (j + 1);
    }
  }

  // Print the multiplication table
  cout << "The Multiplication Table:" << endl;
    for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
      cout << multiplicationTable[i][j] << "\t";
    }
    cout << endl;
  }

  return 0;
}
```

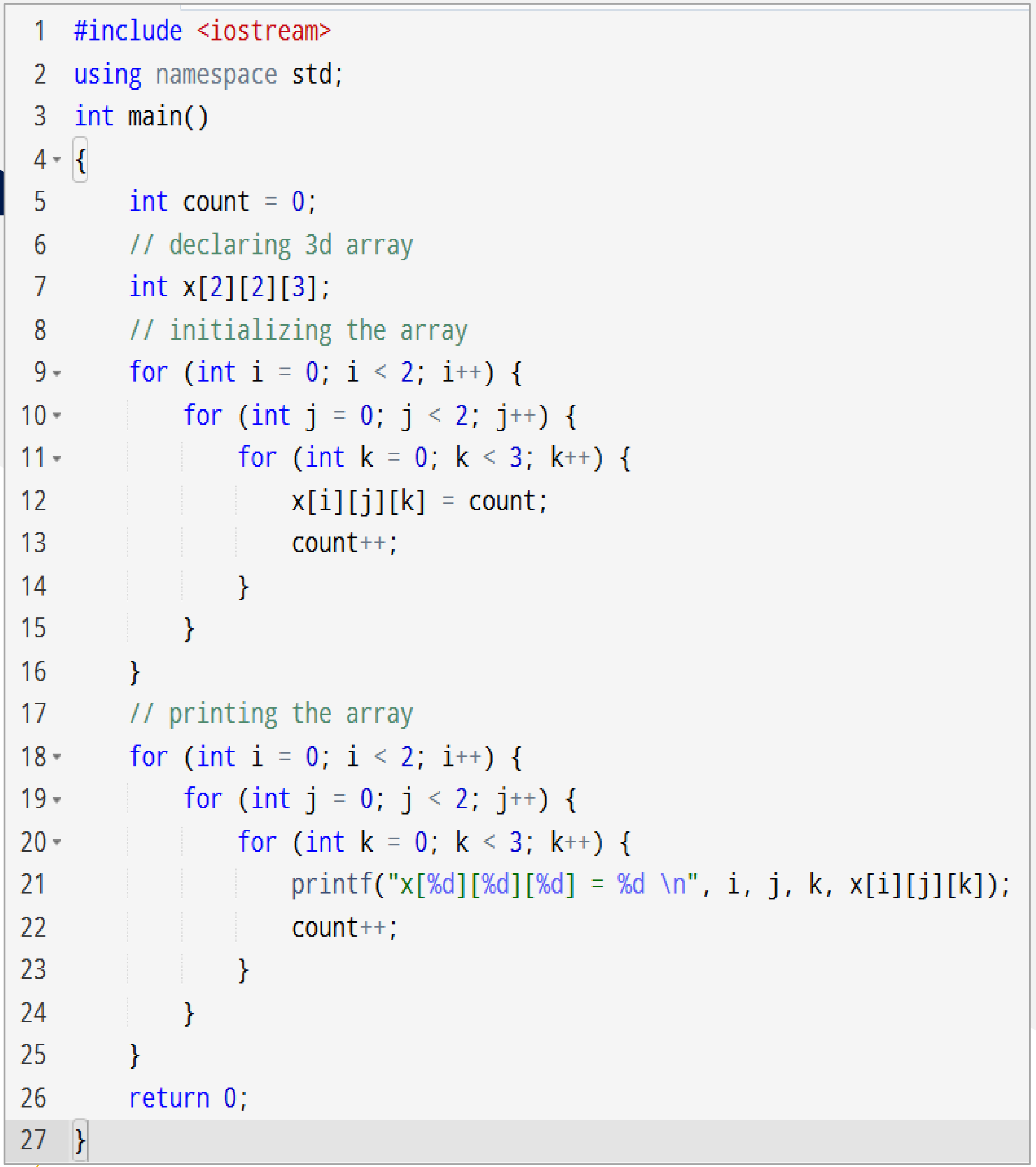
Three-Dimensional Array in C++

- The 3D array is a data structure that stores elements in a three-dimensional cuboid-like structure. It can be visualized as a collection of multiple two-dimensional arrays stacked on top of each other. Each element in a 3D array is identified by its three indices: the row index, column index, and depth index.

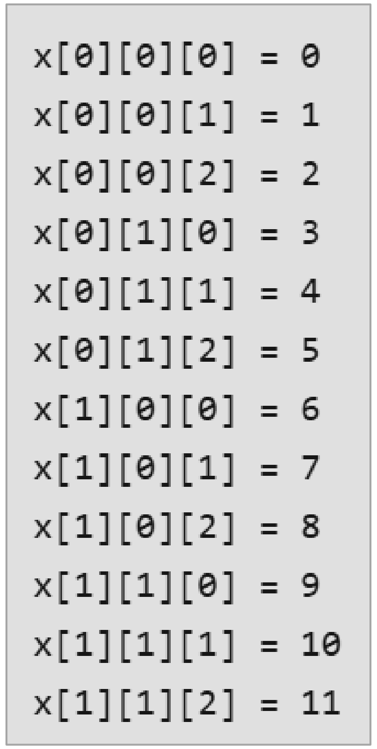


Three-Dimensional Array in C++

- A C++ program using 3D array.



```cpp
#include <iostream>
using namespace std;
int main()
{
    int count = 0;
    // declaring 3d array
    int x[2][2][3];
    // initializing the array
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 3; k++) {
                x[i][j][k] = count;
                count++;
            }
        }
    }
    // printing the array
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 3; k++) {
                printf("x[%d][%d][%d] = %d \n", i, j, k, x[i][j][k]);
                count++;
            }
        }
    }
    return 0;
}
```



```
x[0][0][0] = 0
x[0][0][1] = 1
x[0][0][2] = 2
x[0][1][0] = 3
x[0][1][1] = 4
x[0][1][2] = 5
x[1][0][0] = 6
x[1][0][1] = 7
x[1][0][2] = 8
x[1][1][0] = 9
x[1][1][1] = 10
x[1][1][2] = 11
```

Vector in C++

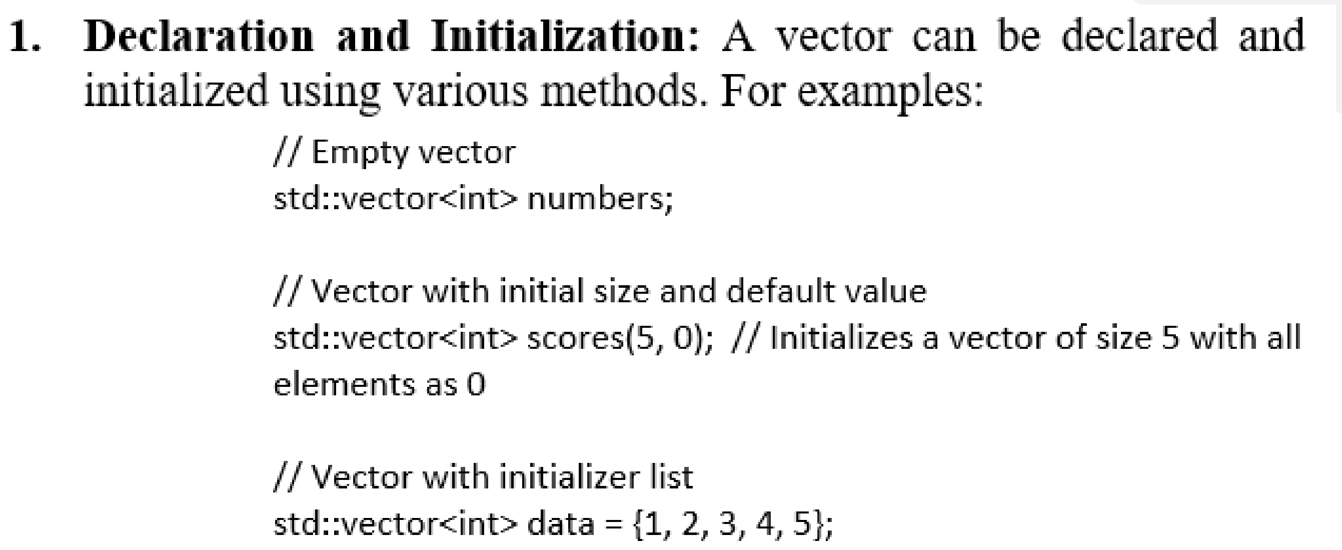
- In C++, the std::vector is a dynamic array that provides a flexible and convenient way to store and manipulate collections of elements.
- It is part of the Standard Template Library (STL) and offers several useful functions and features.
- **Syntax to Declare Vector in C++:**

```
std::vector<dataType> vectorName;
```

- where the data type is the type of data of each element of the vector. You can remove the std:: if you have already used the std namespace.

Vector in C++

- The following is a brief overview of a vector (std::vector) in C++:

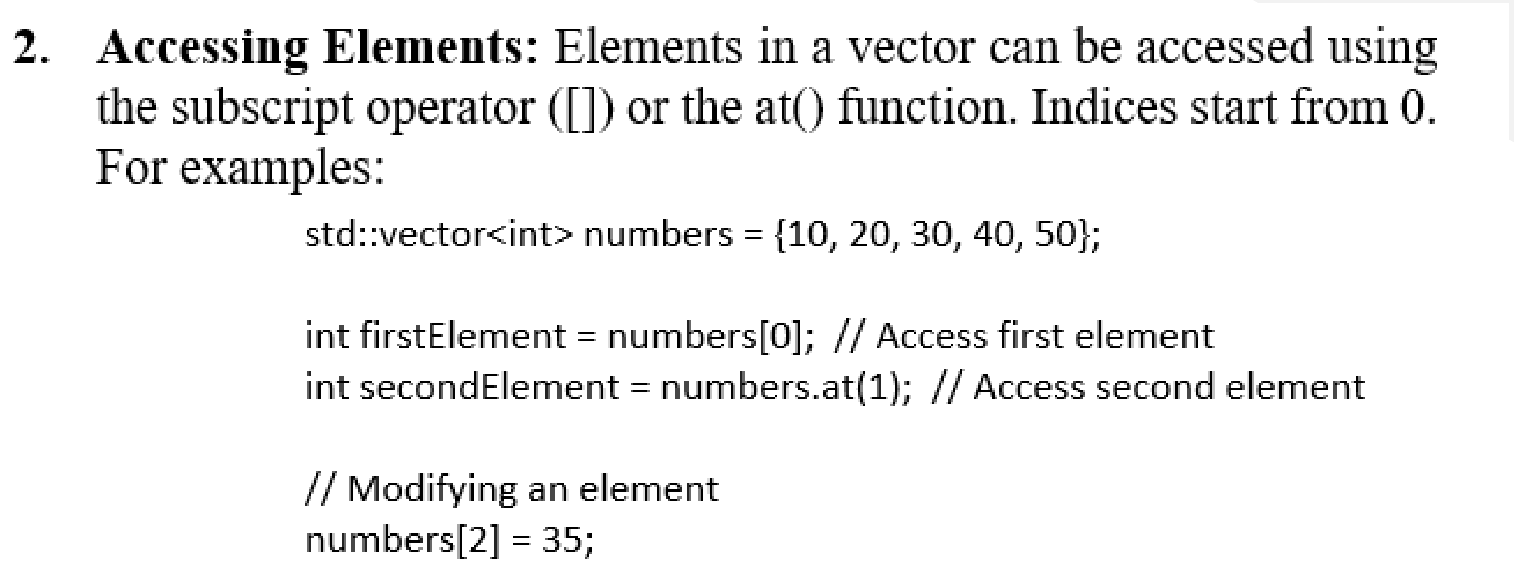


1. **Declaration and Initialization:** A vector can be declared and initialized using various methods. For examples:

```
// Empty vector
std::vector<int> numbers;

// Vector with initial size and default value
std::vector<int> scores(5, 0);  // Initializes a vector of size 5 with all
elements as 0

// Vector with initializer list
std::vector<int> data = {1, 2, 3, 4, 5};
```

Vector in C++

- The following is a brief overview of a vector (std::vector) in C++:

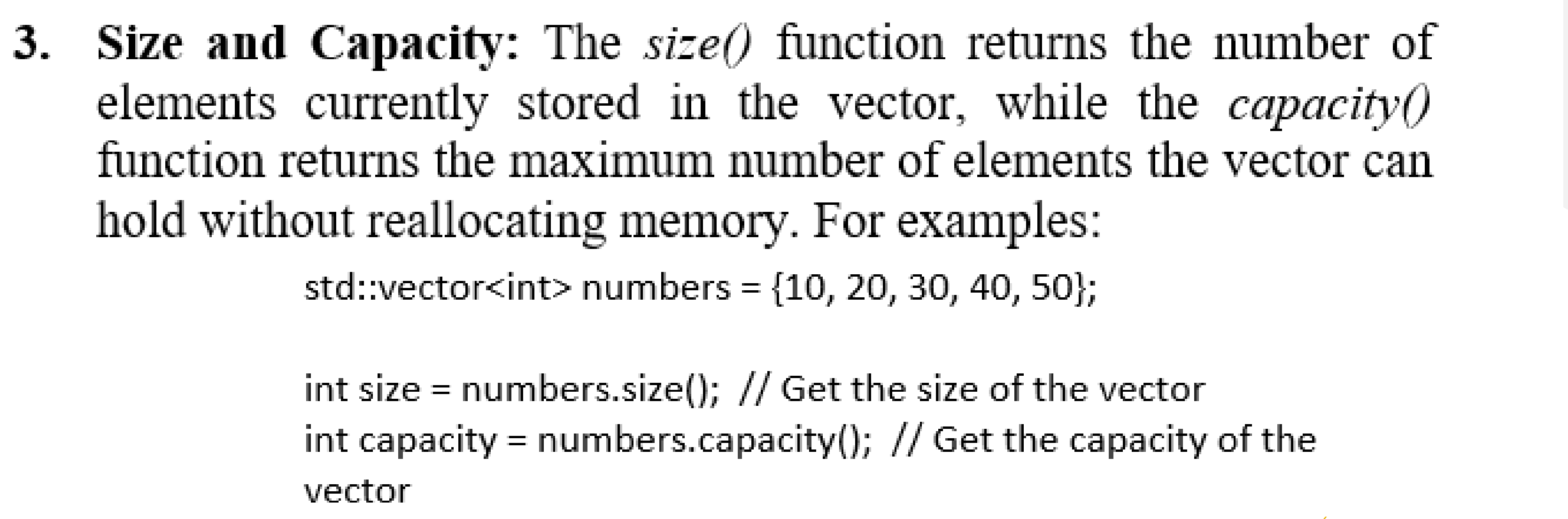


2. **Accessing Elements:** Elements in a vector can be accessed using the subscript operator ([]) or the at() function. Indices start from 0. For examples:

```
std::vector<int> numbers = {10, 20, 30, 40, 50};

int firstElement = numbers[0];  // Access first element
int secondElement = numbers.at(1);  // Access second element

// Modifying an element
numbers[2] = 35;
```

Vector in C++

- The following is a brief overview of a vector (std::vector) in C++:

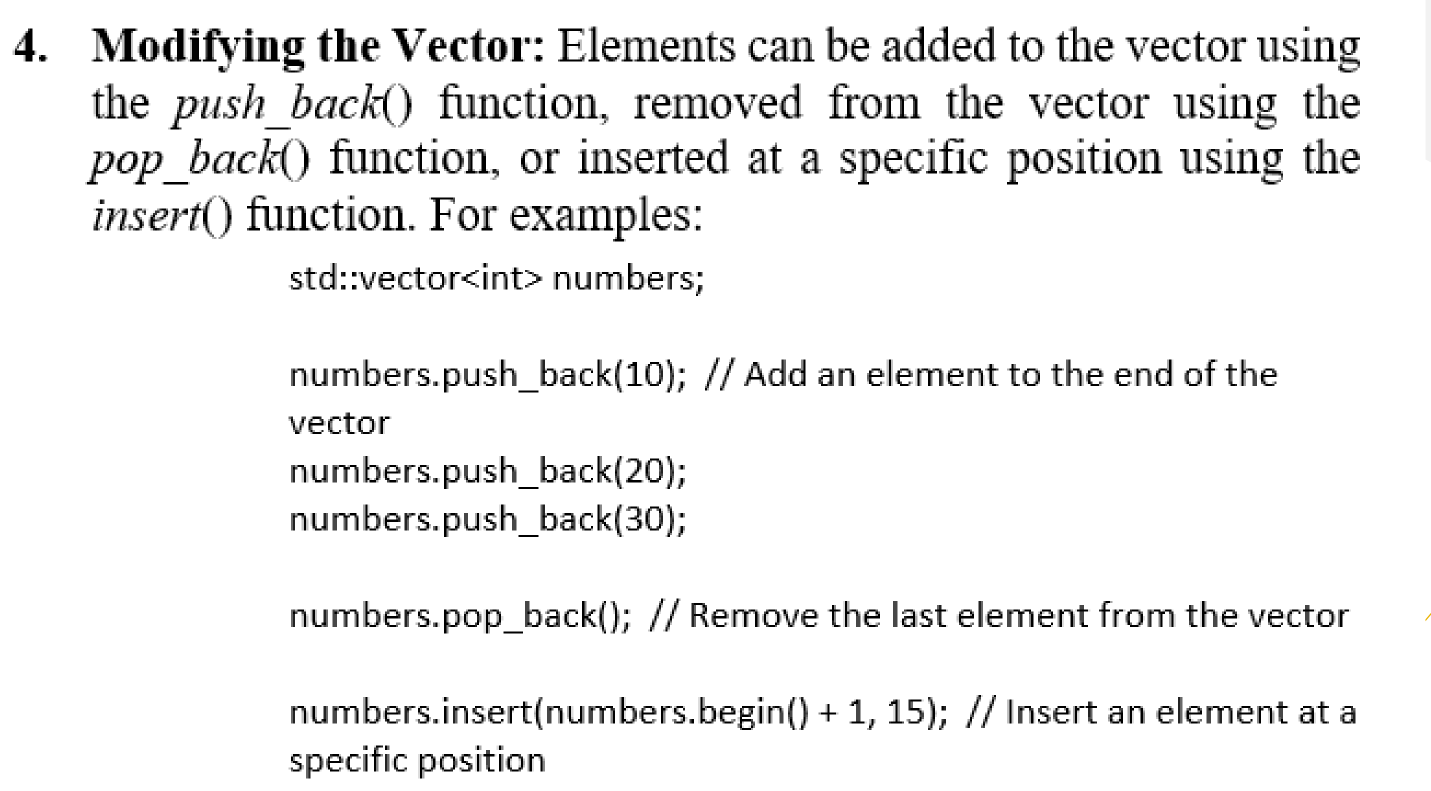


3. **Size and Capacity:** The *size()* function returns the number of elements currently stored in the vector, while the *capacity()* function returns the maximum number of elements the vector can hold without reallocating memory. For examples:

```
std::vector<int> numbers = {10, 20, 30, 40, 50};

int size = numbers.size();  // Get the size of the vector
int capacity = numbers.capacity();  // Get the capacity of the
vector
```

Vector in C++

- The following is a brief overview of a vector (std::vector) in C++:

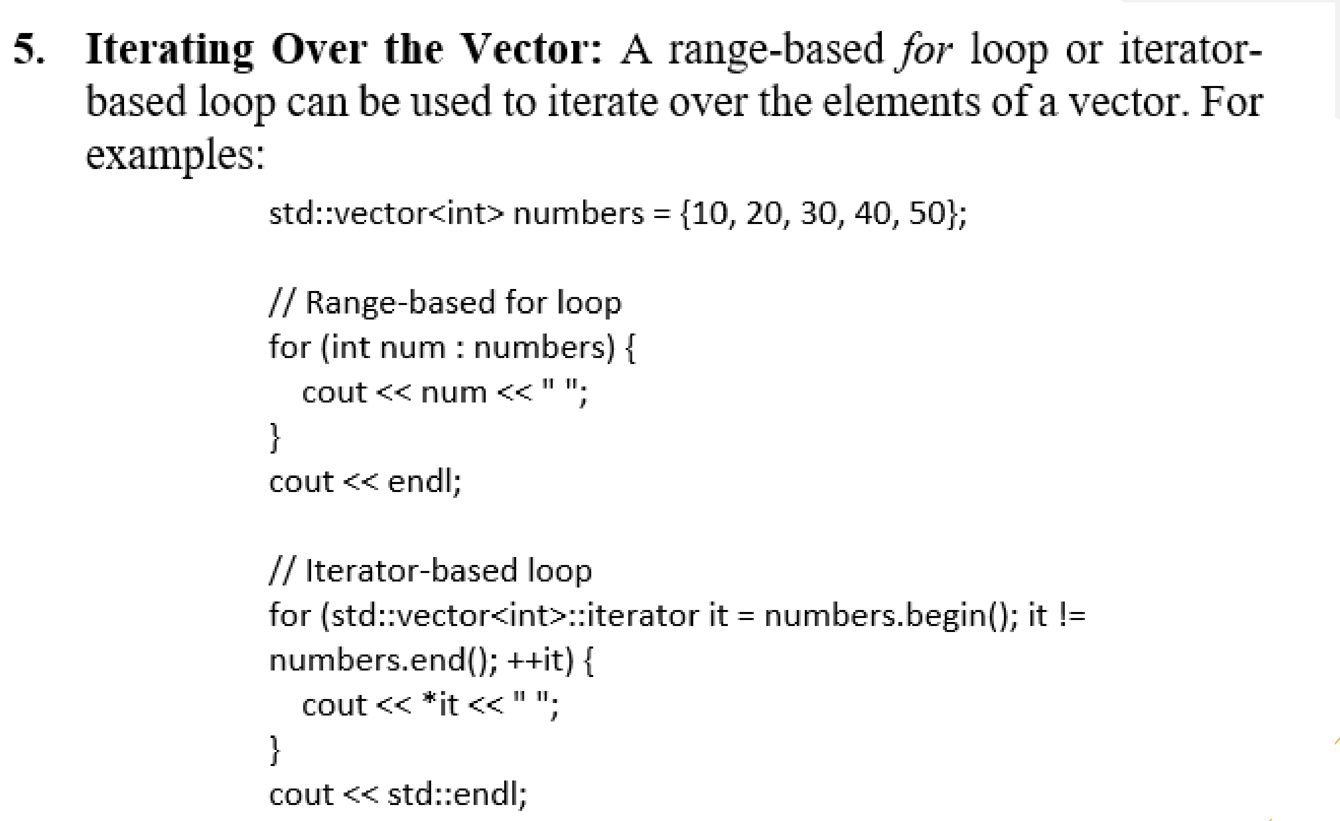


4. **Modifying the Vector:** Elements can be added to the vector using the *push_back*() function, removed from the vector using the *pop_back*() function, or inserted at a specific position using the *insert*() function. For examples:

```
std::vector<int> numbers;

numbers.push_back(10);  // Add an element to the end of the
vector
numbers.push_back(20);
numbers.push_back(30);

numbers.pop_back();  // Remove the last element from the vector

numbers.insert(numbers.begin() + 1, 15);  // Insert an element at a
specific position
```

Vector in C++

- The following is a brief overview of a vector (std::vector) in C++:

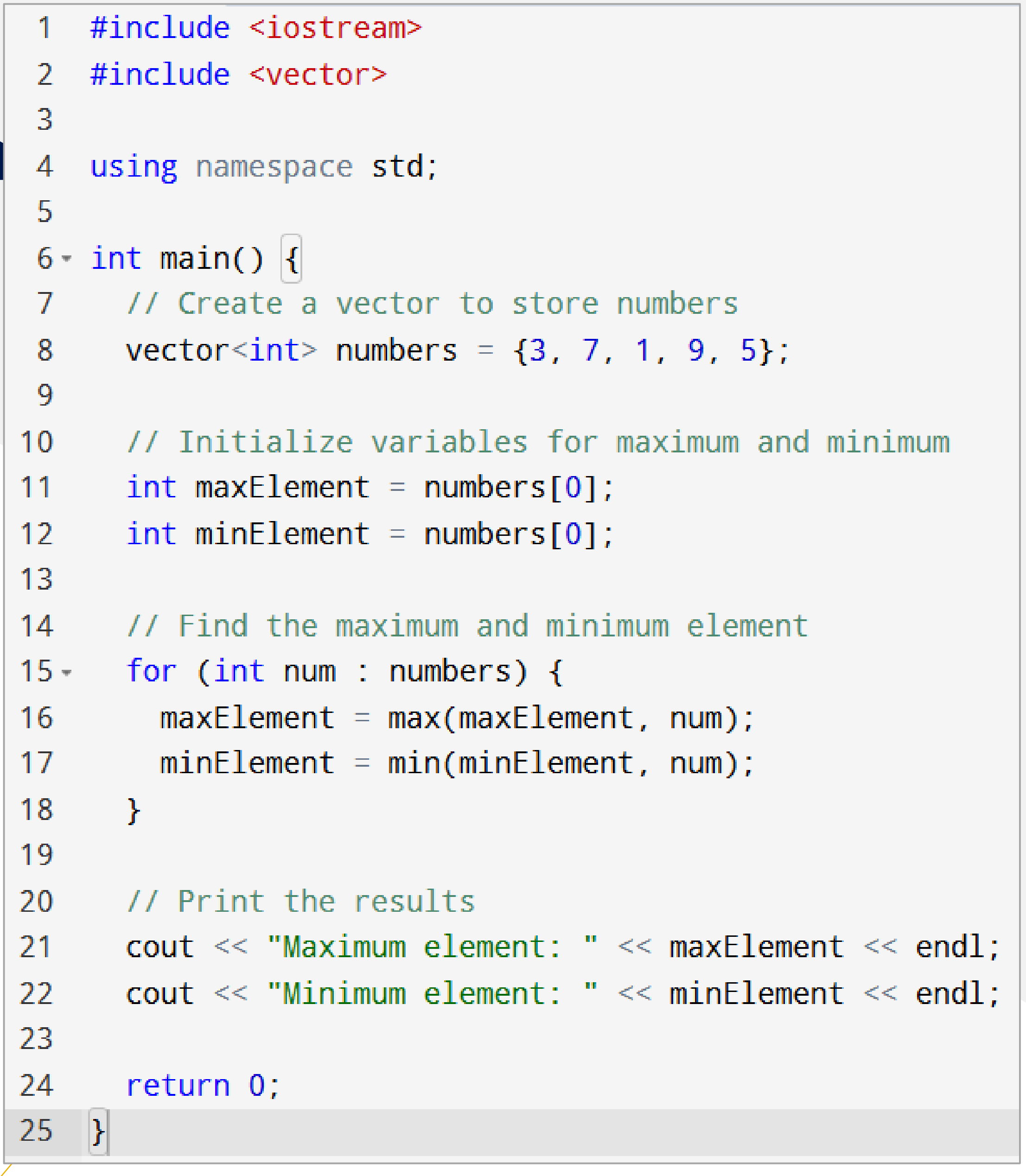


5. **Iterating Over the Vector:** A range-based *for* loop or iterator-based loop can be used to iterate over the elements of a vector. For examples:

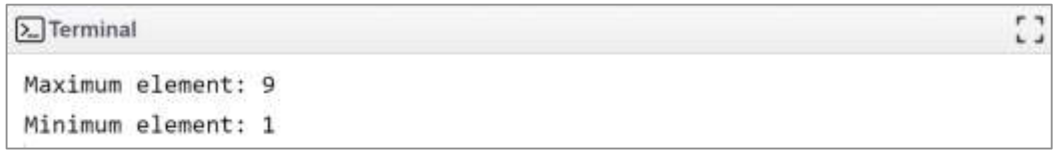
```
std::vector<int> numbers = {10, 20, 30, 40, 50};

// Range-based for loop
for (int num : numbers) {
   cout << num << " ";
}
cout << endl;

// Iterator-based loop
for (std::vector<int>::iterator it = numbers.begin(); it !=
numbers.end(); ++it) {
   cout << *it << " ";
}
cout << std::endl;
```

Vector in C++

- A C++ Program using vector:



```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
  // Create a vector to store numbers
  vector<int> numbers = {3, 7, 1, 9, 5};

  // Initialize variables for maximum and minimum
  int maxElement = numbers[0];
  int minElement = numbers[0];

  // Find the maximum and minimum element
  for (int num : numbers) {
    maxElement = max(maxElement, num);
    minElement = min(minElement, num);
  }

  // Print the results
  cout << "Maximum element: " << maxElement << endl;
  cout << "Minimum element: " << minElement << endl;

  return 0;
}
```



```
Maximum element: 9
Minimum element: 1
```

C++ Pointers

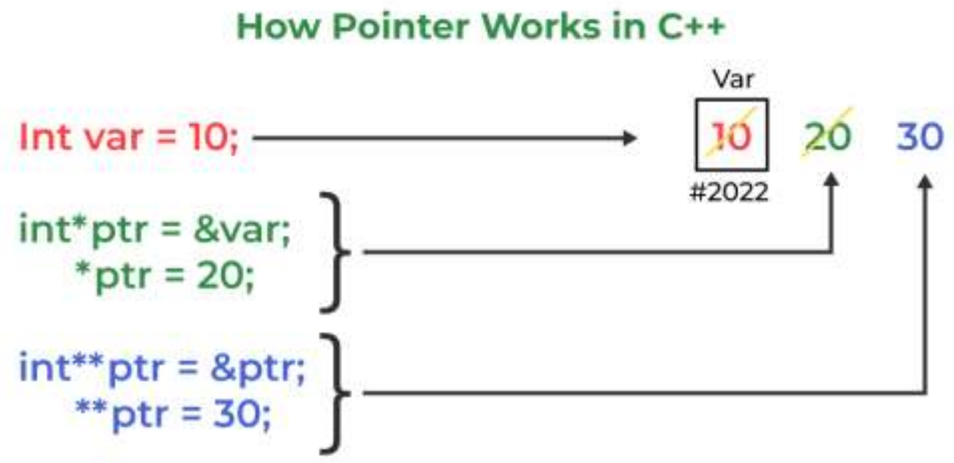
- Pointers are symbolic representations of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. Iterating over elements in arrays or other data structures is one of the main use of pointers.
- The address of the variable you're working with is assigned to the pointer variable that points to the same data type (such as an int or string).

- **Syntax:**

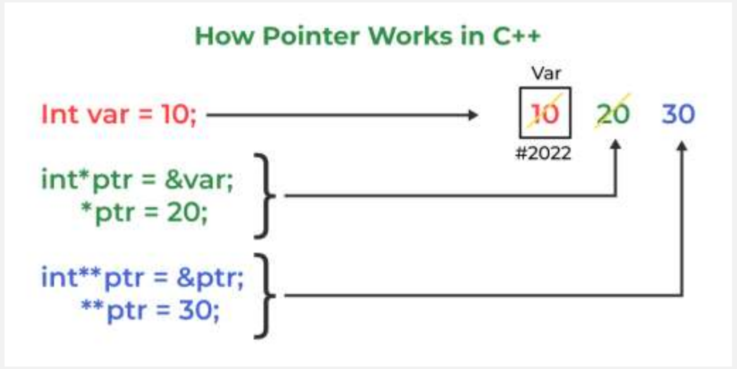
```
datatype *var_name;
int *ptr;   // ptr can point to an address which holds int data
```

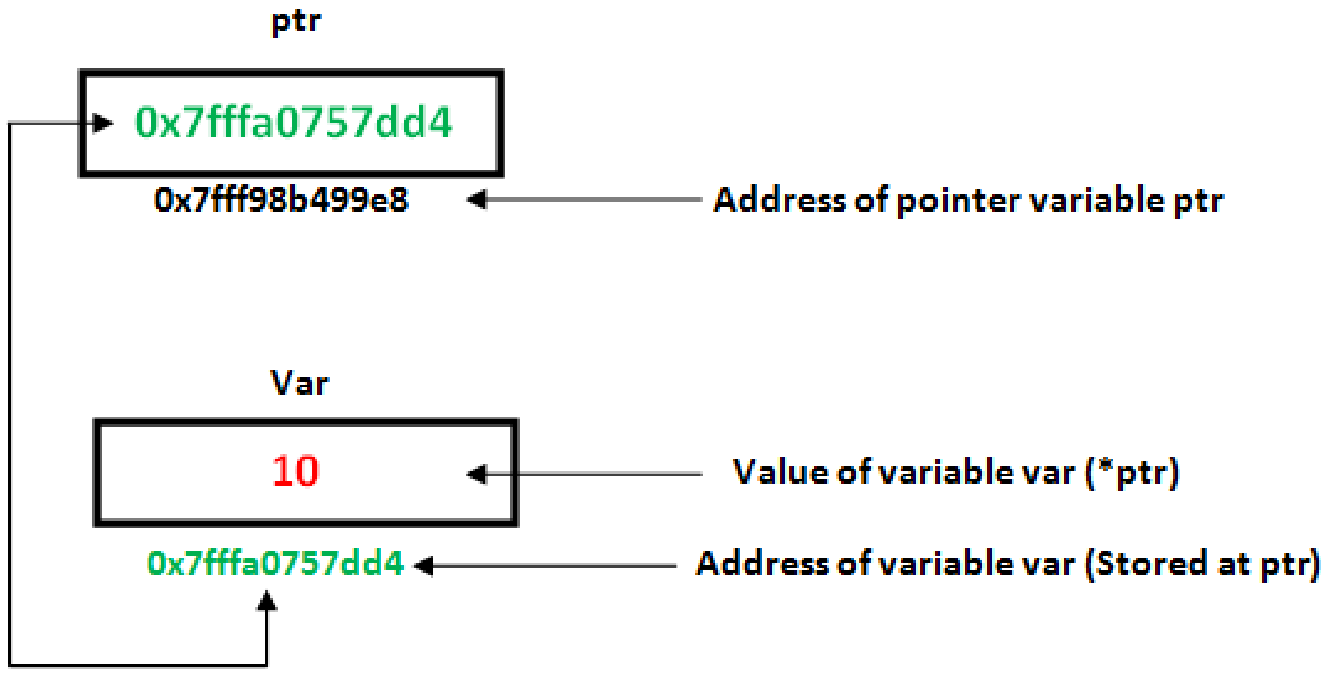
C++ Pointers

- **How to use a pointer?**
  1. Define a pointer variable
  2. Assigning the address of a variable to a pointer using the unary operator (&) which returns the address of that variable.
  3. Accessing the value stored in the address using unary operator (*) which returns the value of the variable located at the address specified by its operand.



C++ Pointers


How Pointer Works in C++
Int var = 10;
Var
10
20
30
#2022
int*ptr = &var;
*ptr = 20;
int**ptr = &ptr;
**ptr = 30;


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (Stored at ptr)

C++ Pointers

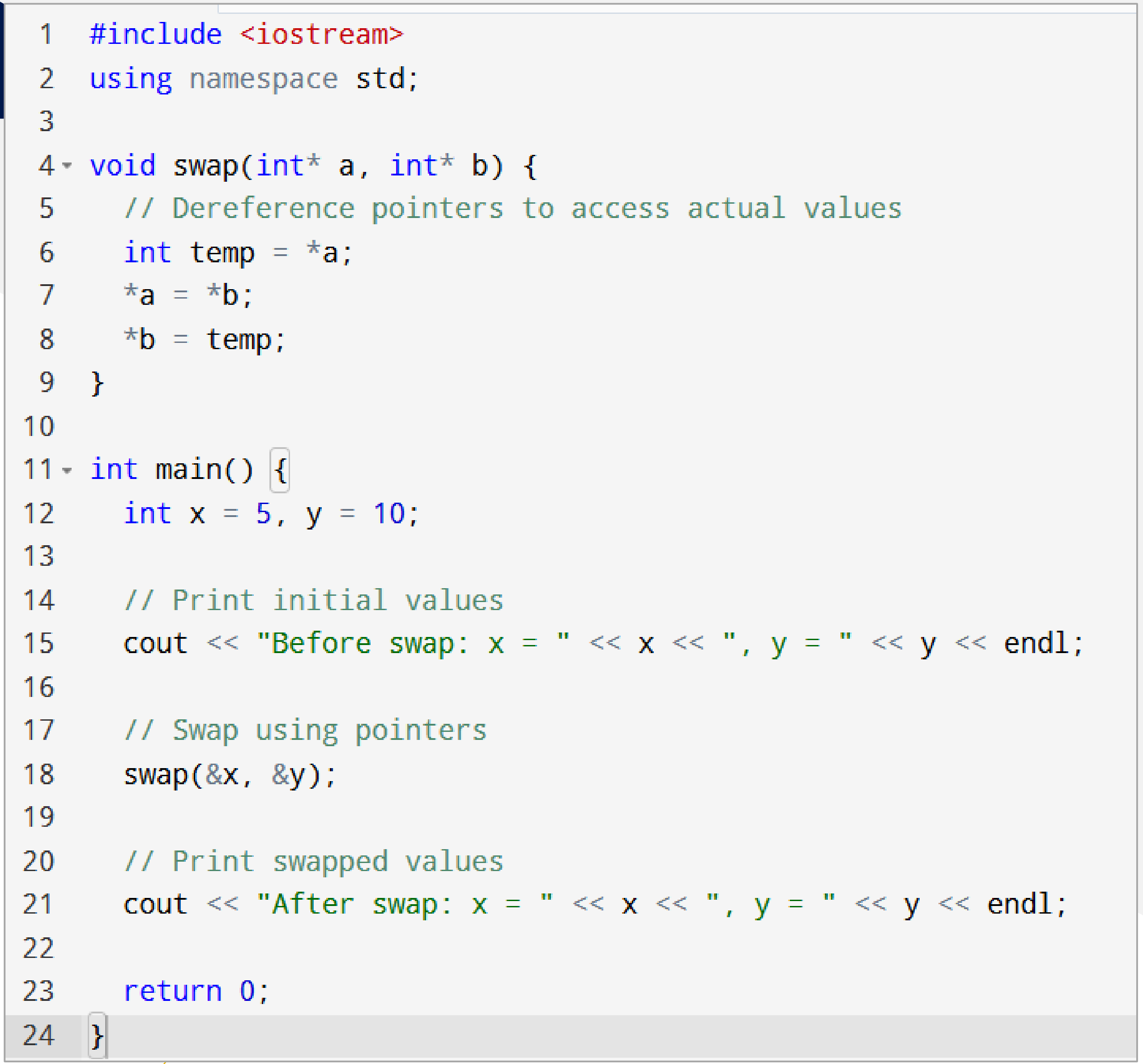
- C++ program to illustrate Pointers:

```
#include <iostream>
using namespace std;
void pFunc()
{
    int var = 100;

    int* ptr;
    ptr = &var;

    // assign the address of a variable to a pointer
    cout << "Value at ptr = " << ptr << "\n";
    cout << "Value at var = " << var << "\n";
    cout << "Value at *ptr = " << *ptr << "\n";
}

int main()
{
  pFunc();
  return 0;
}
```

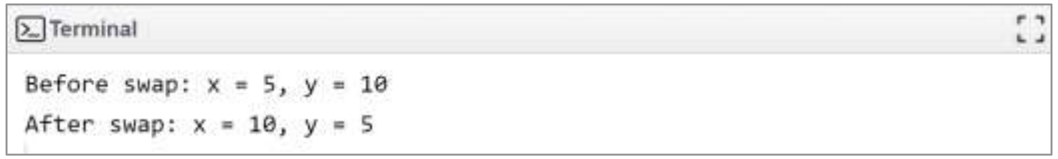
```
Value at ptr = 0x7fff3378e6d4
Value at var = 100
Value at *ptr = 100
```

C++ Pointers

- C++ program to illustrate Pointers:



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

void swap(int* a, int* b) {
  // Dereference pointers to access actual values
  int temp = *a;
  *a = *b;
  *b = temp;
}

int main() {
  int x = 5, y = 10;

  // Print initial values
  cout << "Before swap: x = " << x << ", y = " << y << endl;

  // Swap using pointers
  swap(&x, &y);

  // Print swapped values
  cout << "After swap: x = " << x << ", y = " << y << endl;

  return 0;
}
```



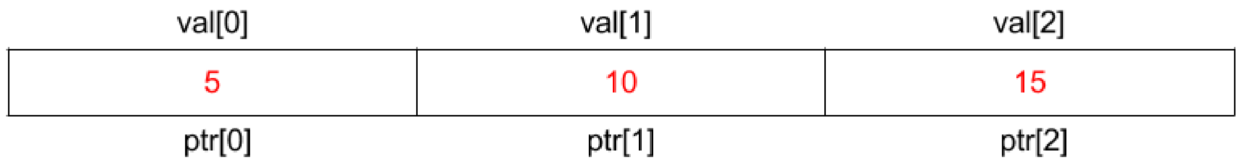
```
Before swap: x = 5, y = 10
After swap: x = 10, y = 5
```

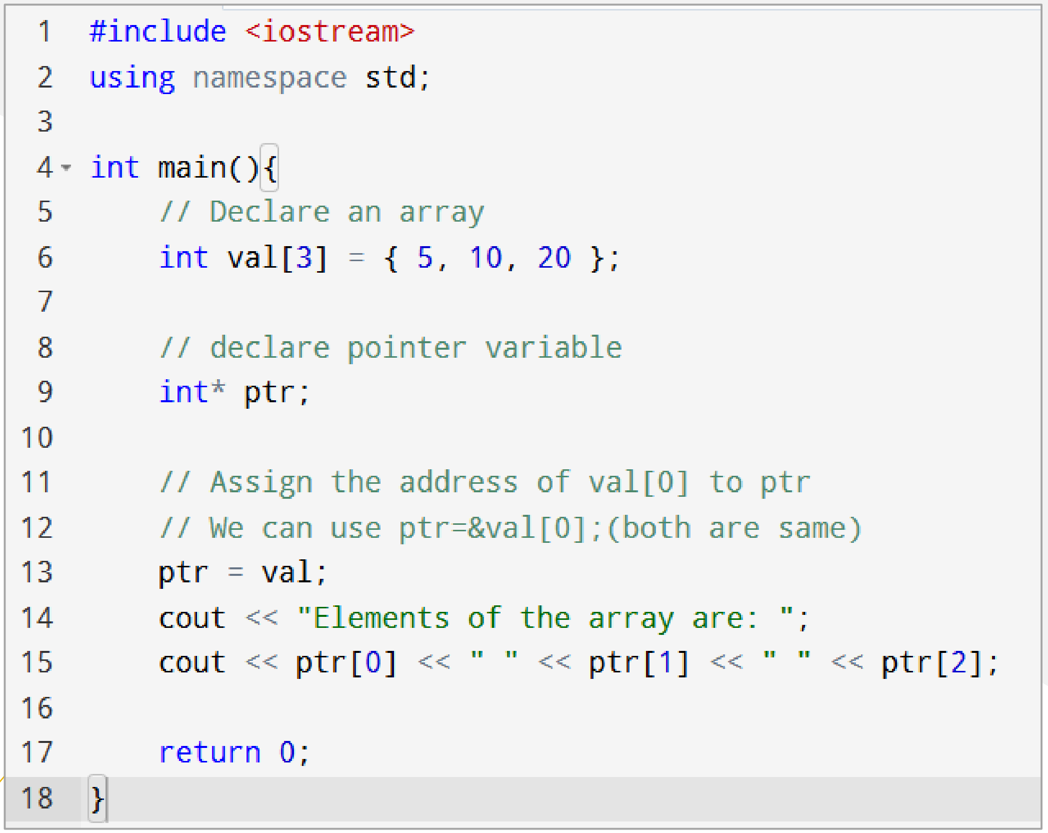
C++ Pointers

- **Array Name as Pointers:**
- An array name contains the address of the first element of the array which acts like a constant pointer. It means, the address stored in the array name can’t be changed.
- For example, if we have an array named **val** then **val** and **&val[0]** can be used interchangeably.

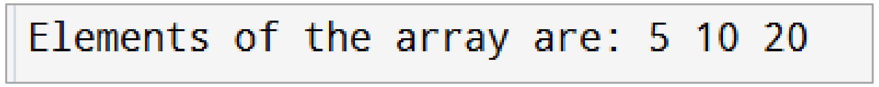
C++ Pointers

- C++ program to illustrate Array Name as Pointers:





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

int main(){
    // Declare an array
    int val[3] = { 5, 10, 20 };

    // declare pointer variable
    int* ptr;

    // Assign the address of val[0] to ptr
    // We can use ptr=&val[0];(both are same)
    ptr = val;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];

    return 0;
}
```



```
Elements of the array are: 5 10 20
```

Lecture 7

Array, Matrix, Vector and Pointers

?

THE END