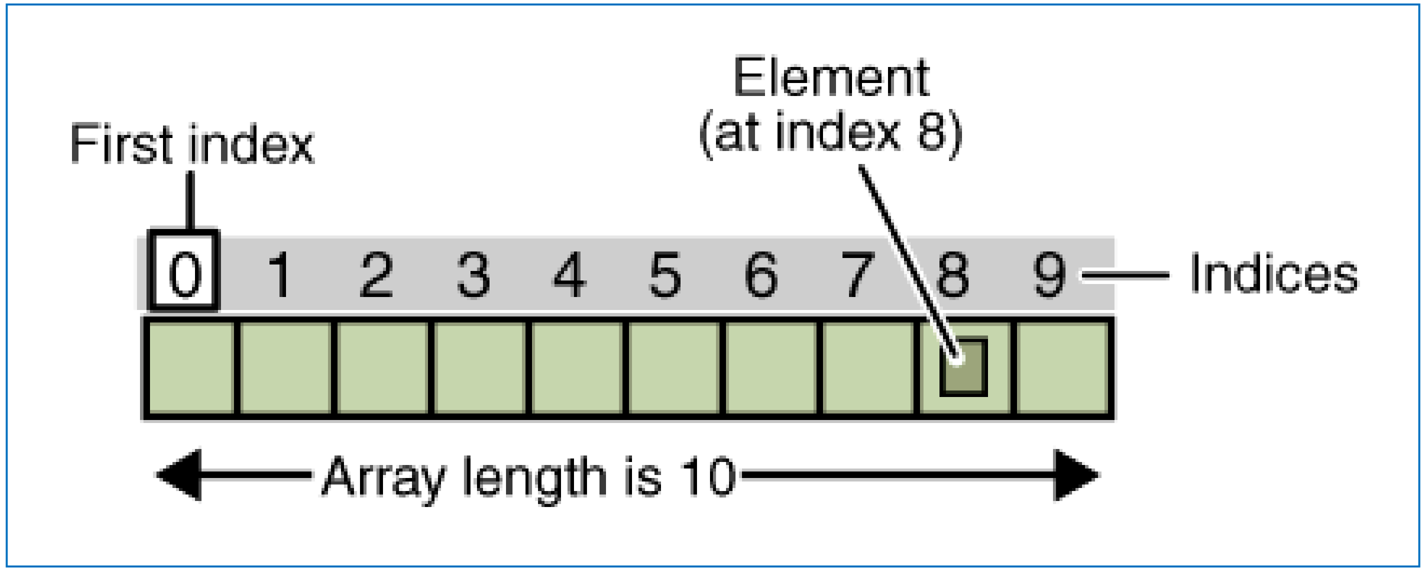
Lecture 09

M M Imran

Arrays

- An array is a series of values of the same data type.
- Each value in an array is called an element or member.
- Each element in an array has an address called an index or subscript.
- Indices are numbered from zero to one less than the number of elements in the array.


Element
(at index 8)
First index
0 1 2 3 4 5 6 7 8 9
Indices
Array length is 10

Example

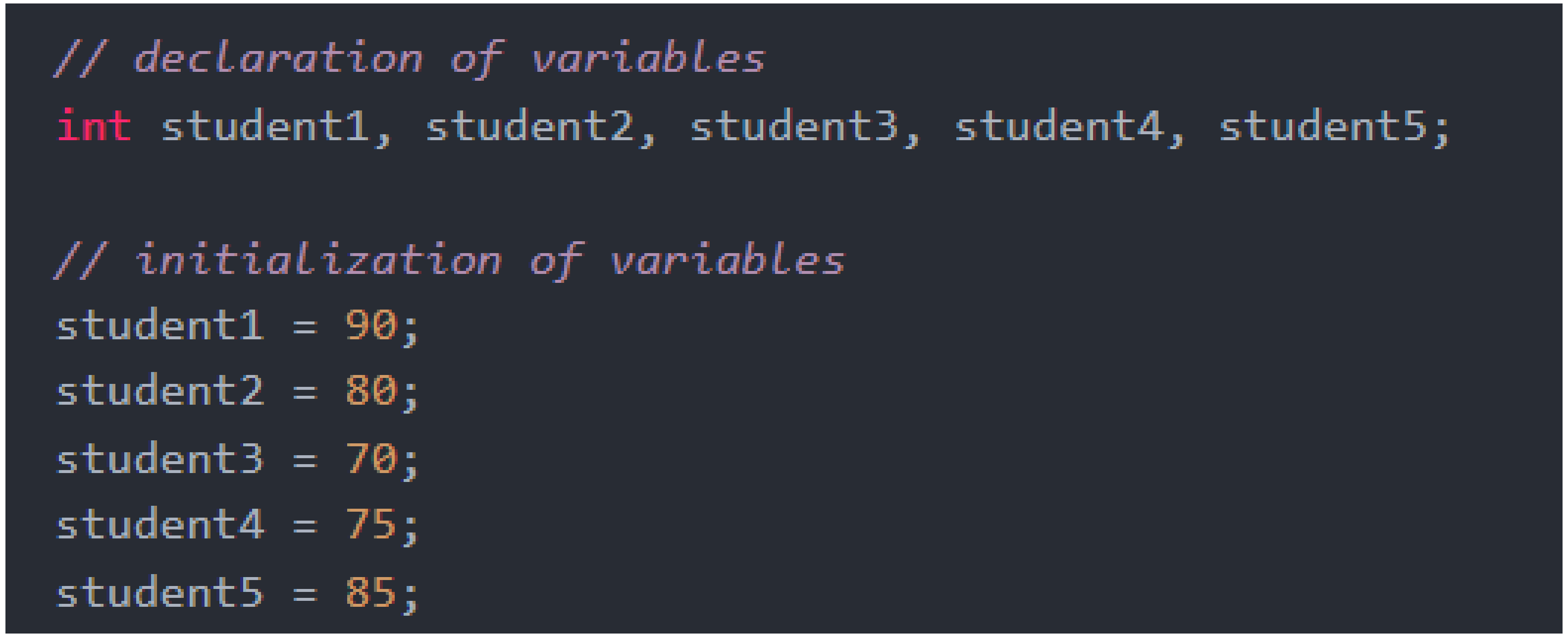
- Suppose a class has 27 students, and we need to store the grades of all of them. Instead of creating 27 separate variables, we can simply create an array:

  double grade[27];

- Here, grade is an array that can hold a maximum of 27 elements of double type.
- In C++, the size and type of arrays cannot be changed after its declaration.

Example

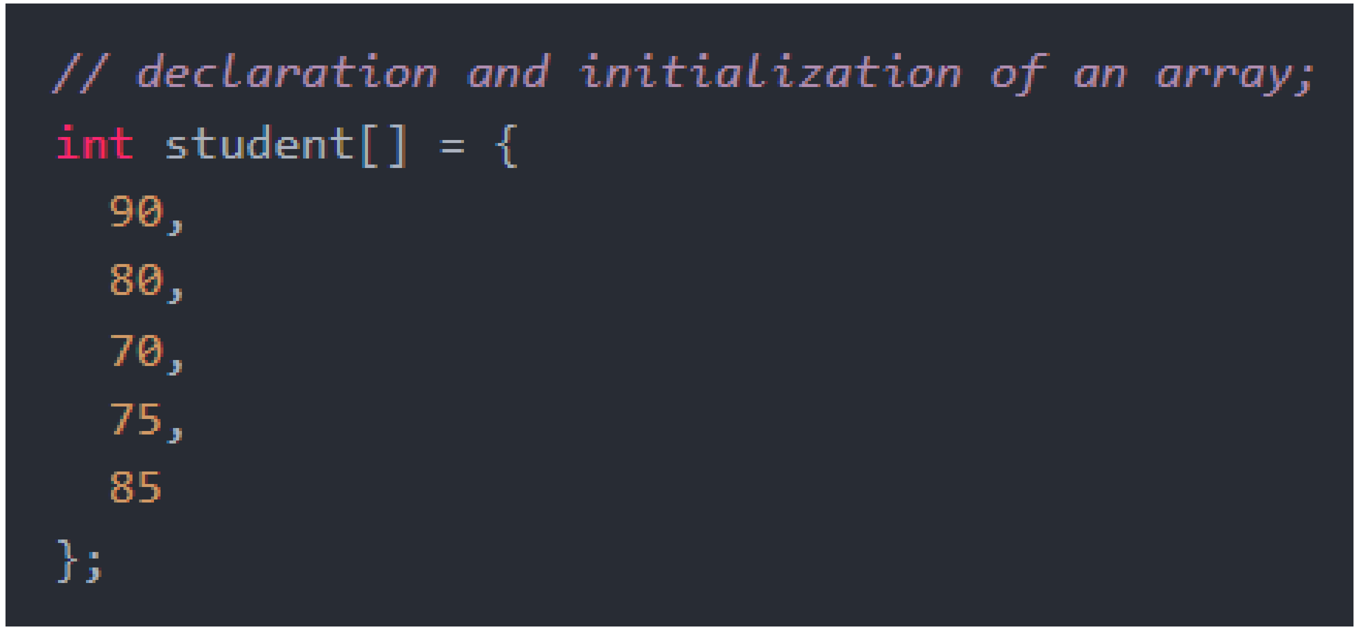
Consider a situation where marks of 5 students are required to store without using arrays. Here, five different variables are created to store the marks of 5 students.



```
// declaration of variables
int student1, student2, student3, student4, student5;

// initialization of variables
student1 = 90;
student2 = 80;
student3 = 70;
student4 = 75;
student5 = 85;
```

Example

However, a single array of 5 elements can store the marks of 5 students. There is no need to create and remember the names of 5 different variables in the program.

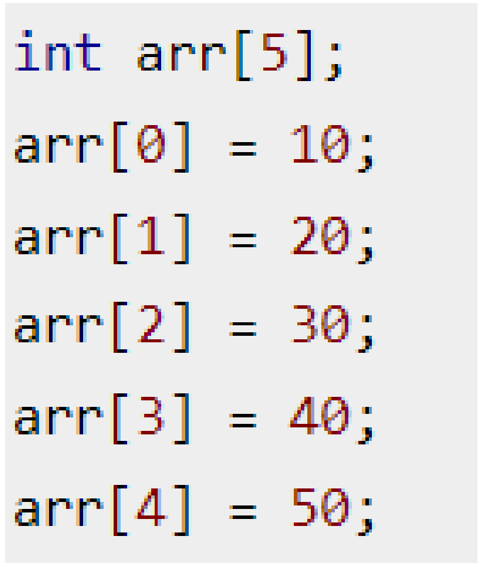


```
// declaration and initialization of an array;
int student[] = {
  90,
  80,
  70,
  75,
  85
};
```

Array declaration

There are couple of ways to declare an array.

Method 1: **<data type> <array-name>[integer-expression>];**

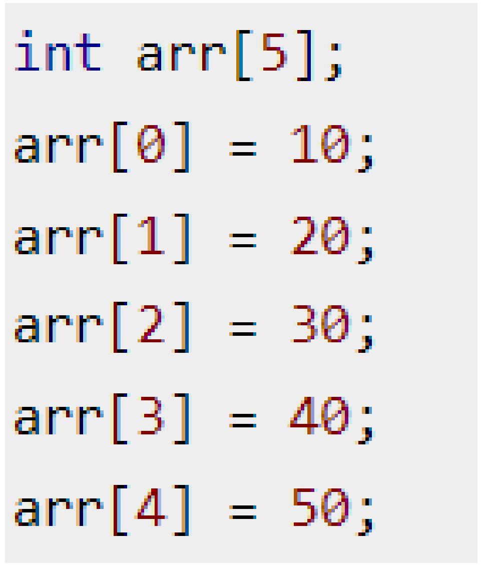


```
int arr[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

Array declaration

There are couple of ways to declare an array.

Method 1: **<data type> <array-name>[integer-expression>];**



```
int arr[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

Array declaration

Method 2: **<data type> <array-name>[] = {initializer-list};**

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

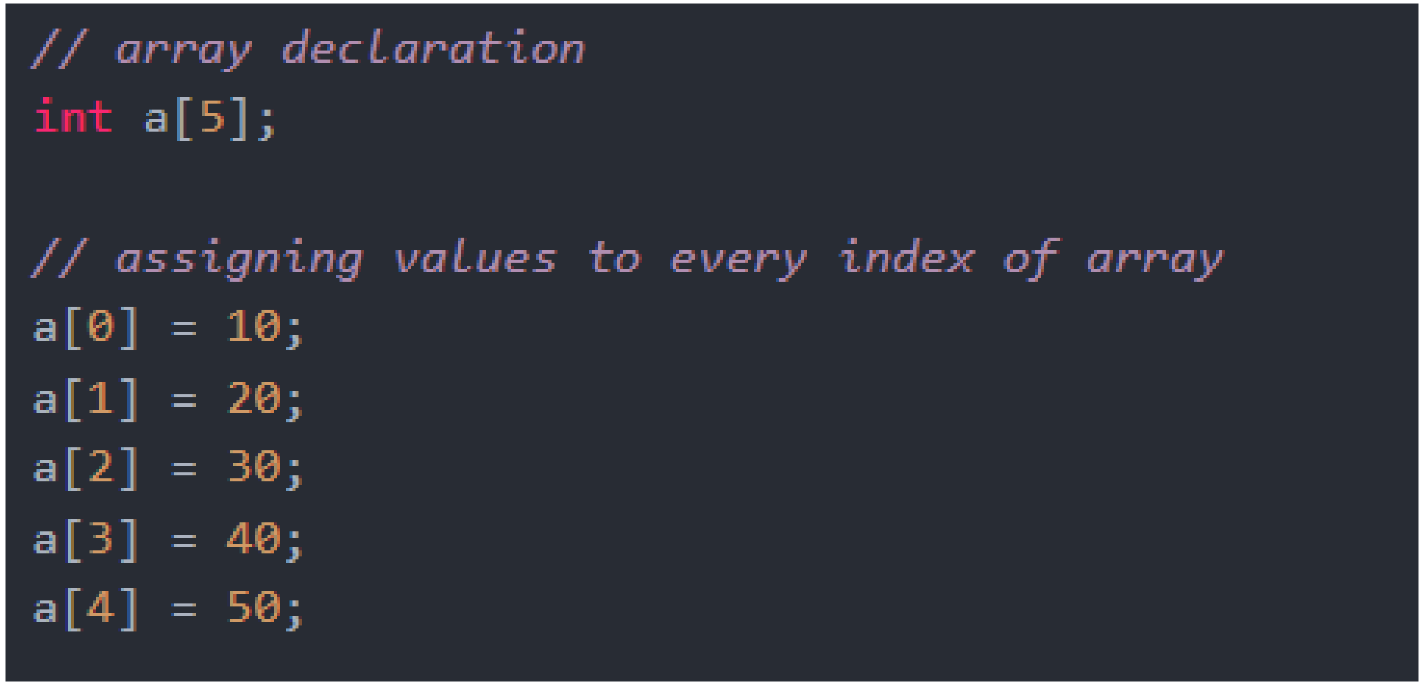
Method 3: **<data type> <array-name>[integer-expression>] = {initializer-list};**

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

Access the Elements of an Array

- Array elements are accessed using indexes.
- In C++, array indexing starts from 0, which implies that the first element in the array is placed at zeroth index.
- If an array has five elements, then indexing will be done from 0 to 4.
- Syntax: **arrayName[index]**

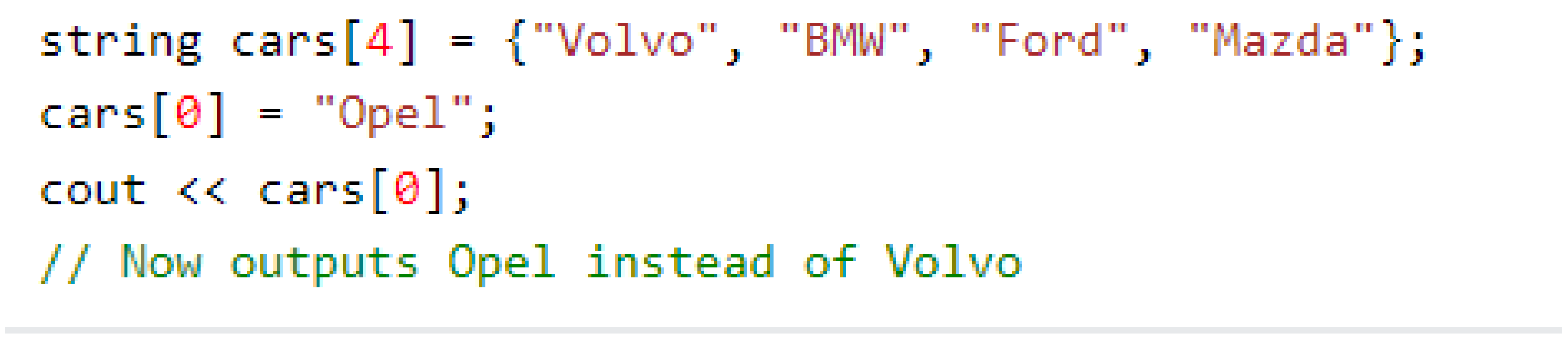
```cpp
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[0];
// Outputs Volvo
```



```cpp
// array declaration
int a[5];

// assigning values to every index of array
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
a[4] = 50;
```

Change an Array Element

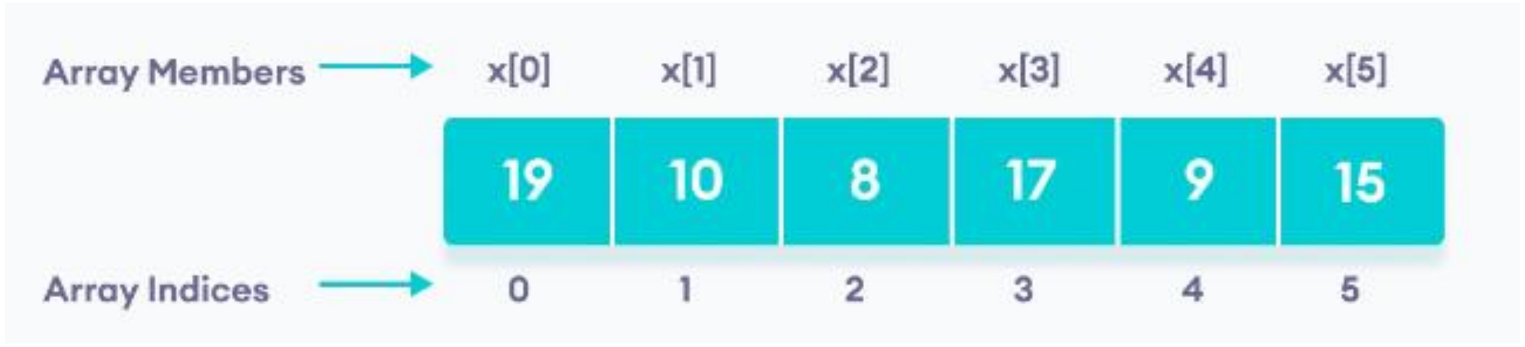
To change the value of a specific element, refer to the index number:



```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
cout << cars[0];
// Now outputs Opel instead of Volvo
```

Different Array Initializations

int x[6] = {19, 10, 8, 17, 9, 15};



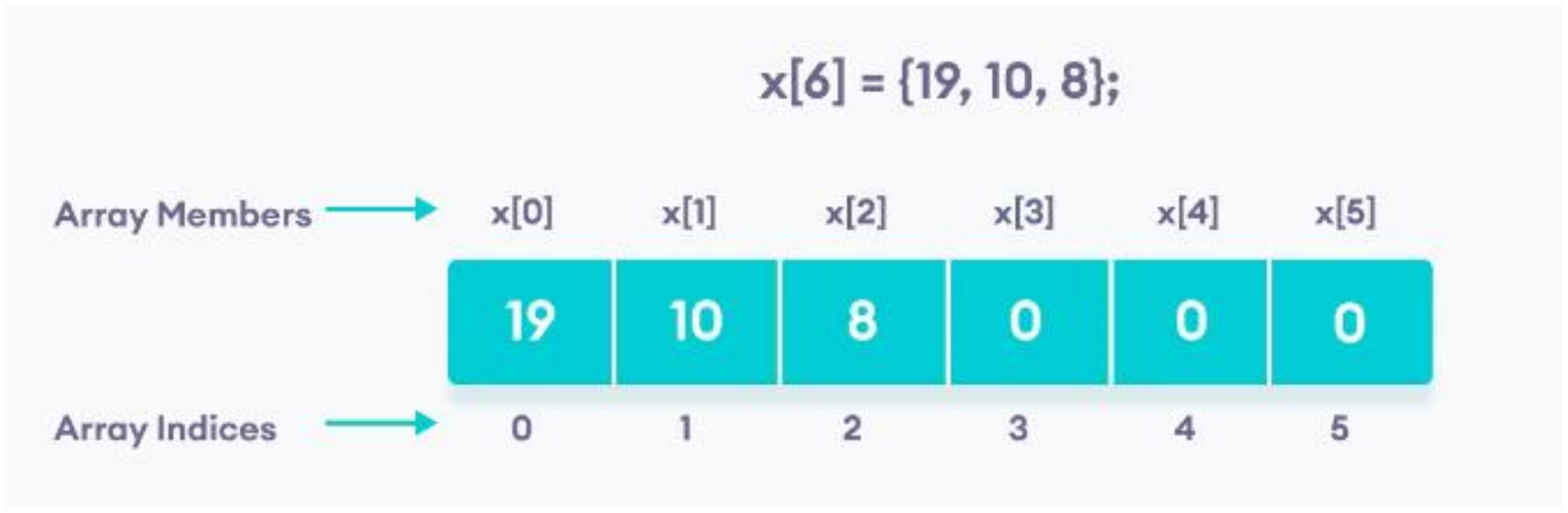
int x[] = {19, 10, 8, 17, 9, 15};

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

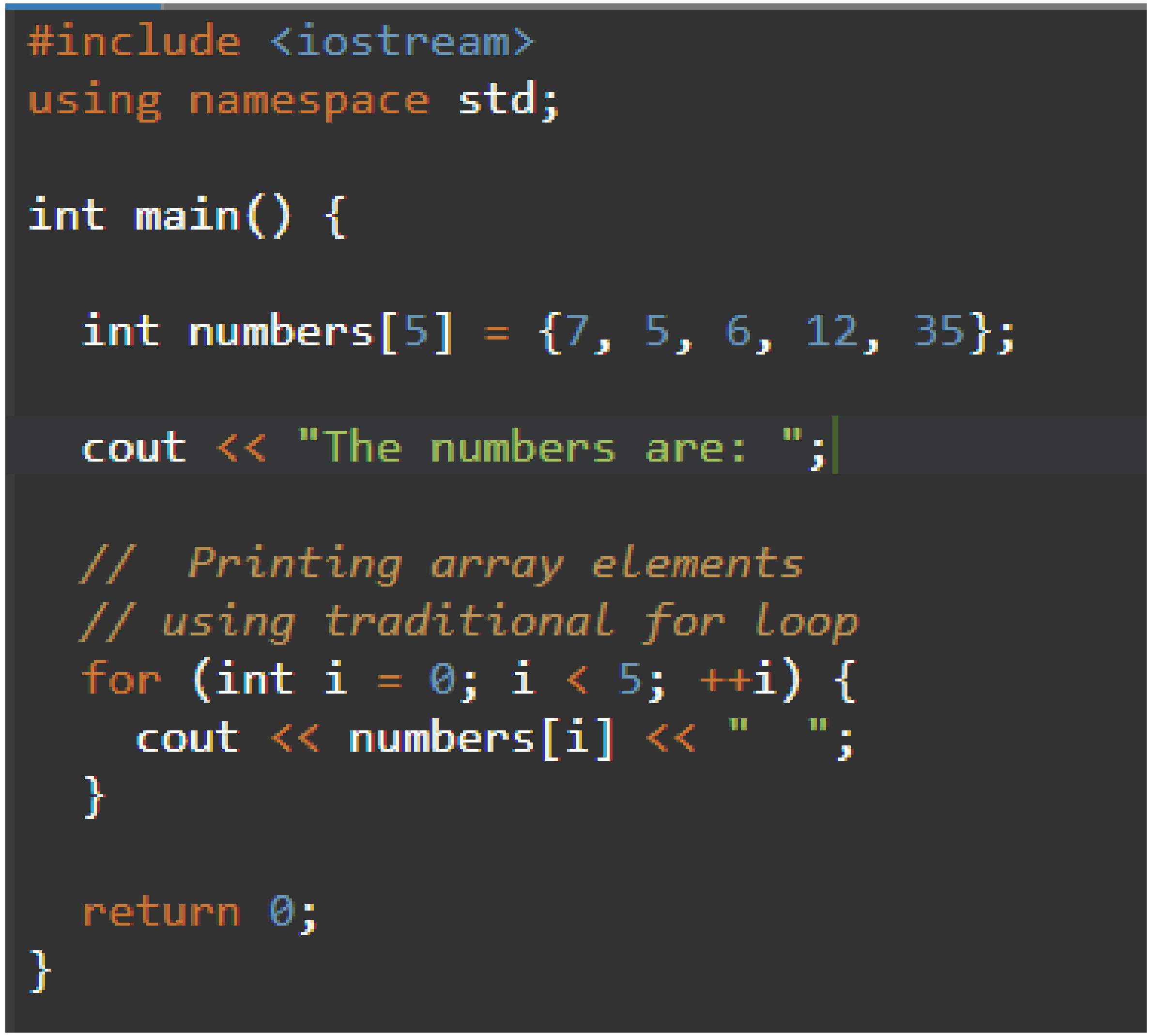
Different Array Initializations

int x[6] = {19, 10, 8};

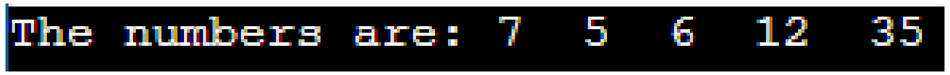
Here, the array x has a size of 6. However, we have initialized it with only 3 elements. In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.



Example: Displaying Array Elements

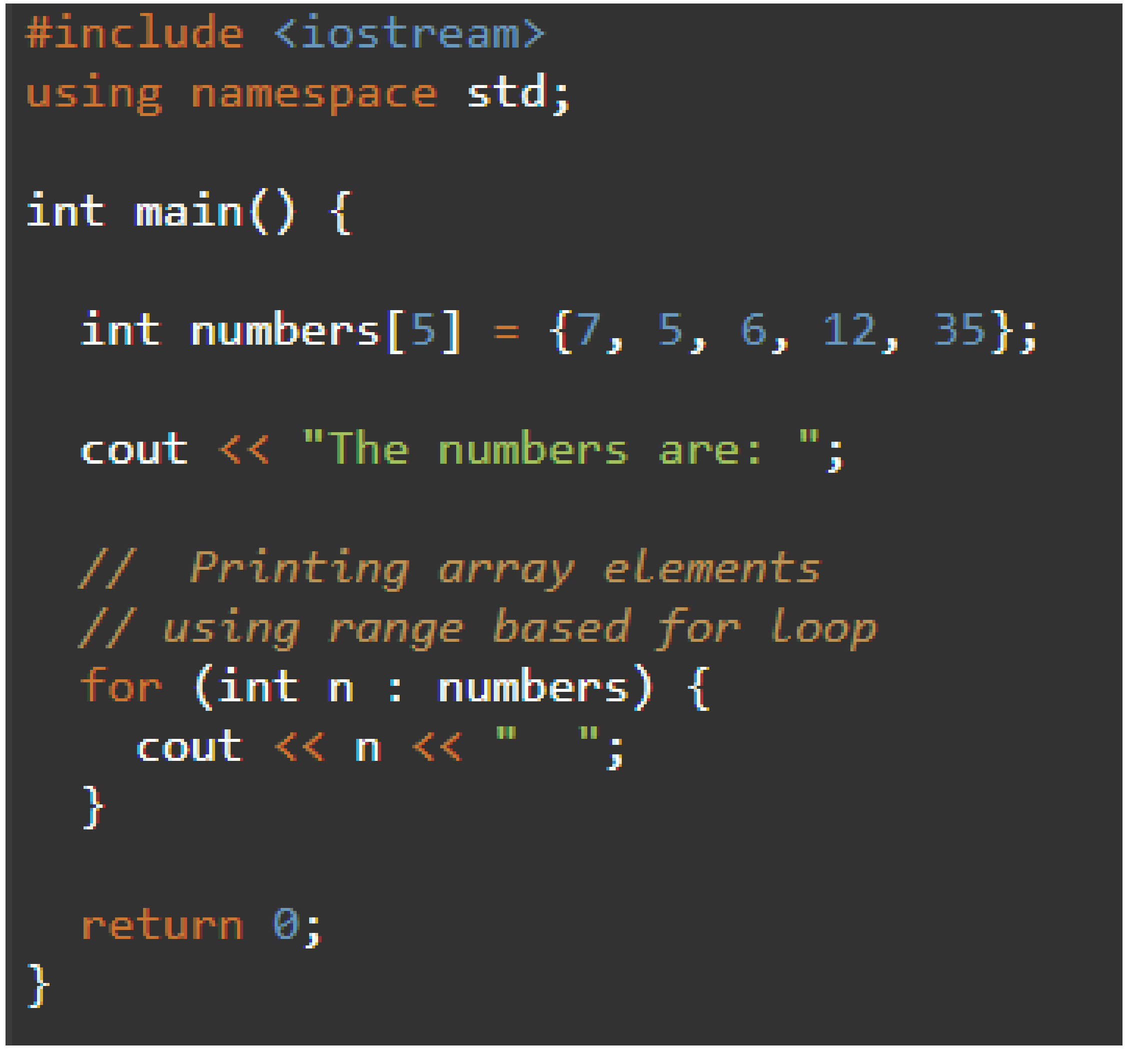


```
#include <iostream>
using namespace std;

int main() {

  int numbers[5] = {7, 5, 6, 12, 35};

  cout << "The numbers are: ";

  //  Printing array elements
  // using traditional for loop
  for (int i = 0; i < 5; ++i) {
    cout << numbers[i] << "  ";
  }

  return 0;
}
```

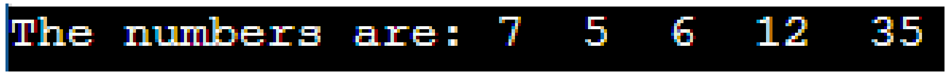


```
The numbers are: 7  5  6  12  35
```

Example: Displaying Array Elements

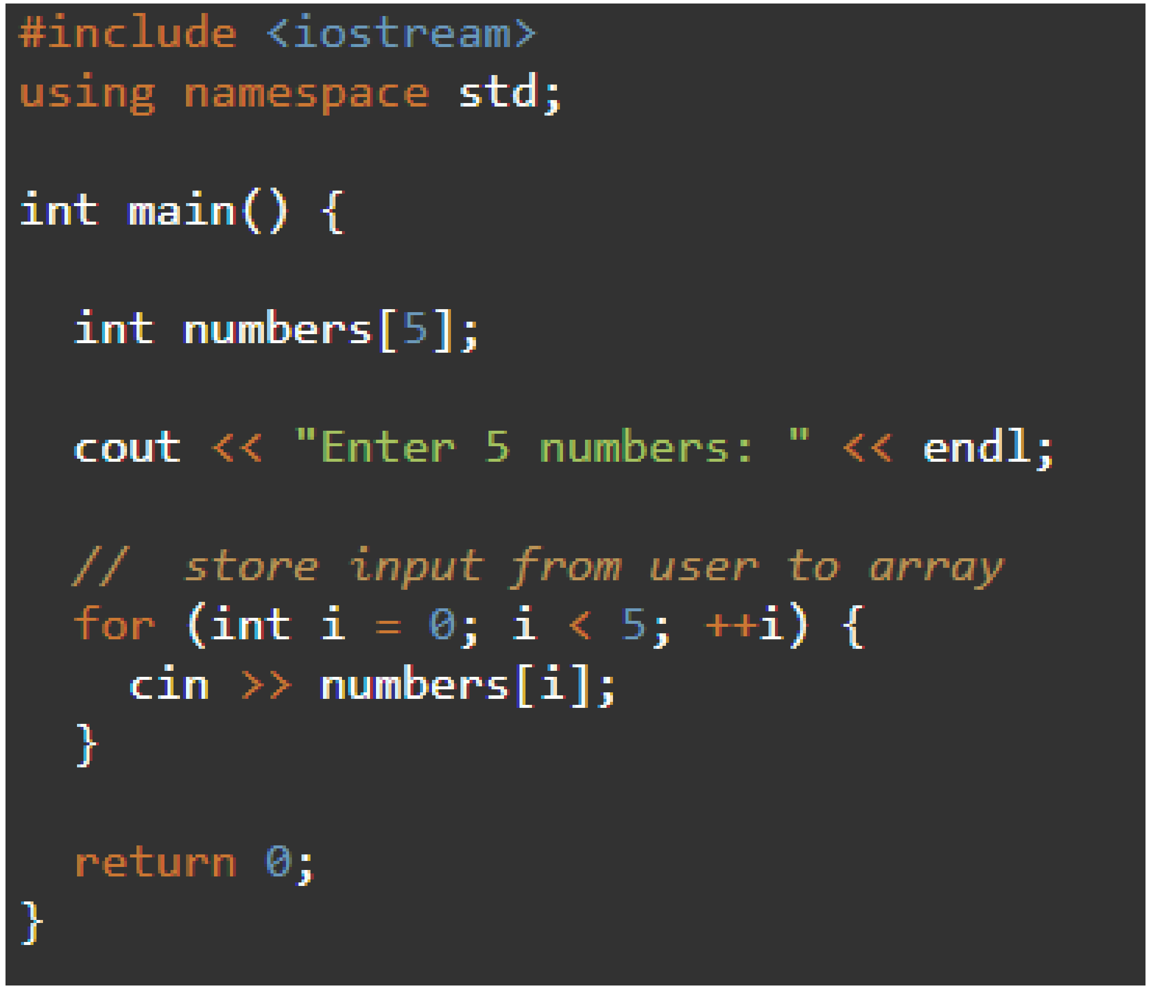


```cpp
#include <iostream>
using namespace std;

int main() {

   int numbers[5] = {7, 5, 6, 12, 35};

   cout << "The numbers are: ";

   //  Printing array elements
   // using range based for loop
   for (int n : numbers) {
     cout << n << "  ";
   }

   return 0;
}
```



```
The numbers are: 7  5  6  12  35
```

Example: Take Inputs from User

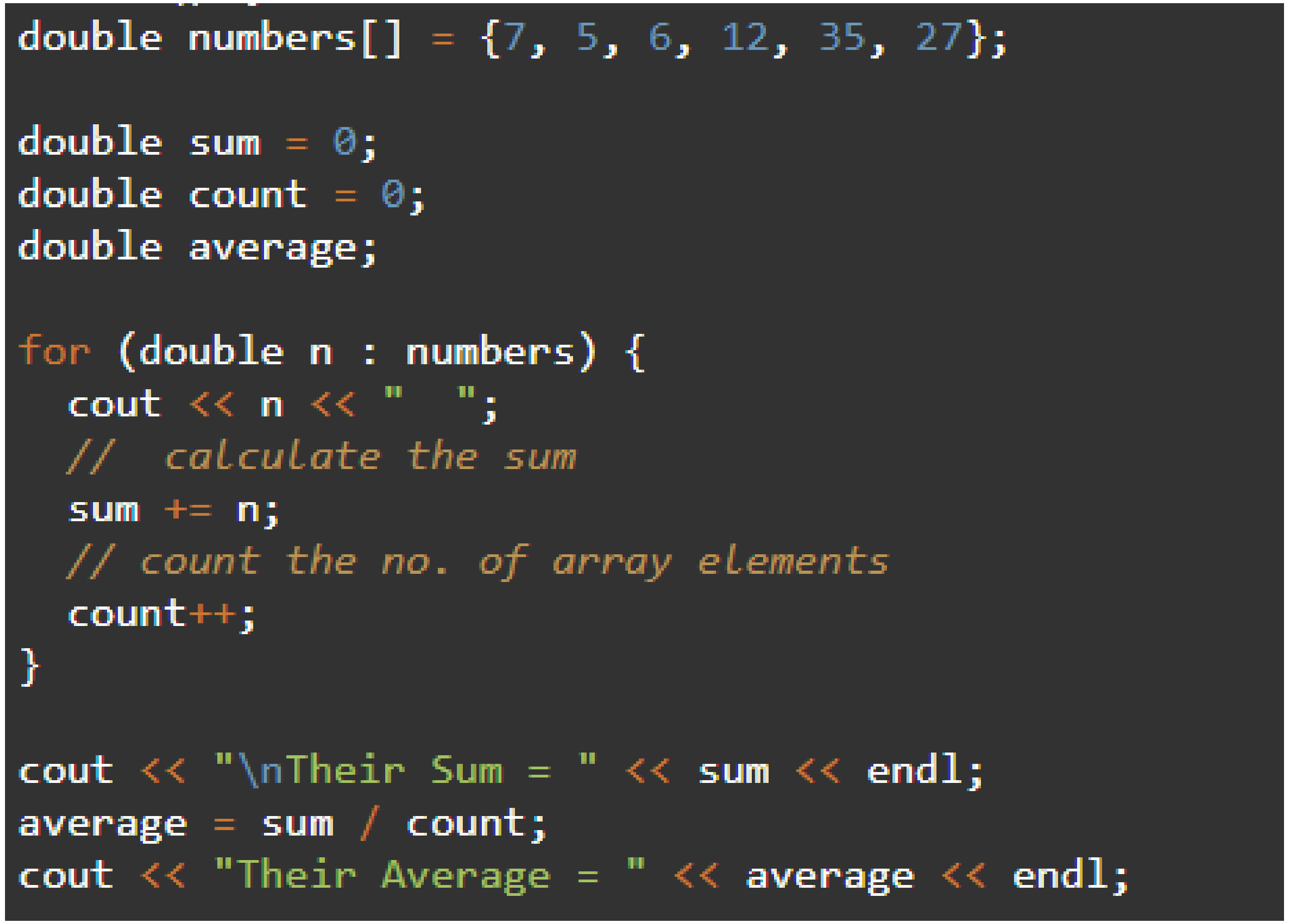


```cpp
#include <iostream>
using namespace std;

int main() {

  int numbers[5];

  cout << "Enter 5 numbers: " << endl;

  //  store input from user to array
  for (int i = 0; i < 5; ++i) {
    cin >> numbers[i];
  }

  return 0;
}
```

Practice

- Take Inputs from User and Store Them in an Array
- Print array elements

Example: Display Sum and Average of Array



```cpp
double numbers[] = {7, 5, 6, 12, 35, 27};

double sum = 0;
double count = 0;
double average;

for (double n : numbers) {
  cout << n << "  ";
  //  calculate the sum
  sum += n;
  // count the no. of array elements
  count++;
}

cout << "\nTheir Sum = " << sum << endl;
average = sum / count;
cout << "Their Average = " << average << endl;
```

Array Out of Bounds

- An invalid array element reference would be:
  - Less than zero.
  - Greater than or equal to the size of the array.
- In either case, an array out-of-bounds error results.
- C++ allows an array out-of-bounds error, but this is not good!
- If the array element reference is:
  - Less than zero, the program shows whatever is stored in memory before the array.
  - Greater than or equal to the size of the array, the program shows whatever is stored in memory after the array.

Advantages of an Array

- Elements of an array can be accessed in O(1) time.
- Minimizes the length of the code by putting values of multiple variables into a single array.
- Updation of array elements is done in constant time.
- Arrays can be easily traversed using a single loop.
- The memory of array elements are very close to one another, and therefore, the cache can easily access them from the CPU.
- Managing and sorting array elements requires fewer lines of code.

Disadvantages of an Array

- The size of an array can not be increased or decreased once defined during declaration. The use of arrays is not suitable when size is not defined earlier.
- Arrays are homogeneous. All the elements of an array should belong to the same datatype.
- The process of insertion and deletion of elements in arrays is costly.
- Garbage value is thrown while accessing any index out of range.

Multi-Dimensional Arrays

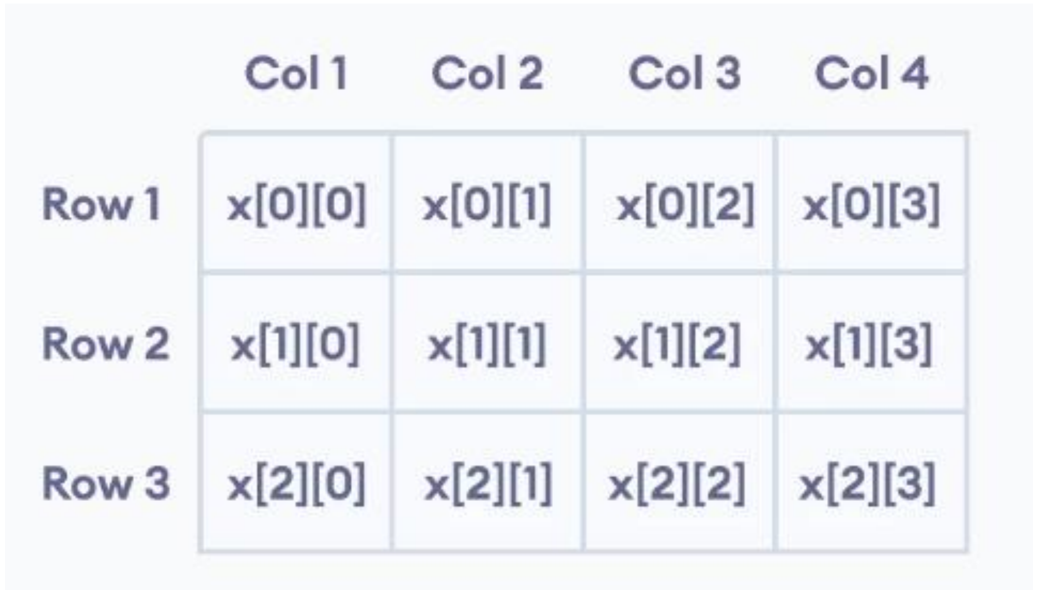
- A multi-dimensional array is an array of arrays.
- To declare a multi-dimensional array,
  - Define the variable type,
  - Specify the name of the array followed by square brackets which specify how many elements the main array has,
  - Followed by another set of square brackets which indicates how many elements the sub-arrays have.

dataType arrayName[size1d][size2d]...[sizeNd]

Two-dimensional array

int x[3][4];

- Here, x is a two-dimensional array. It can hold a maximum of 12 elements.
- We can think of this array as a table with 3 rows and each row has 4 columns as shown below.



	Col 1	Col 2	Col 3	Col 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

```
string letters[2][4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};
```

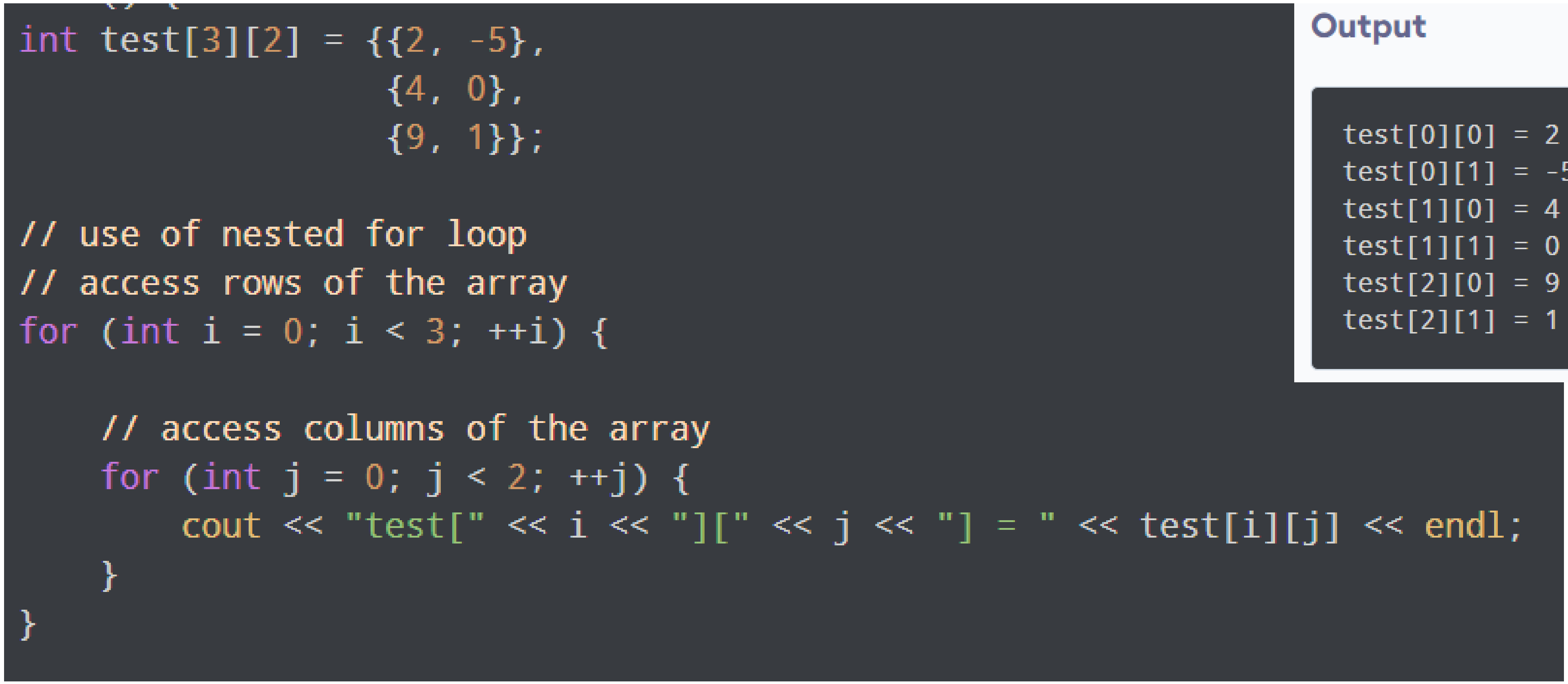
Three-dimensional array

float x[2][4][3];

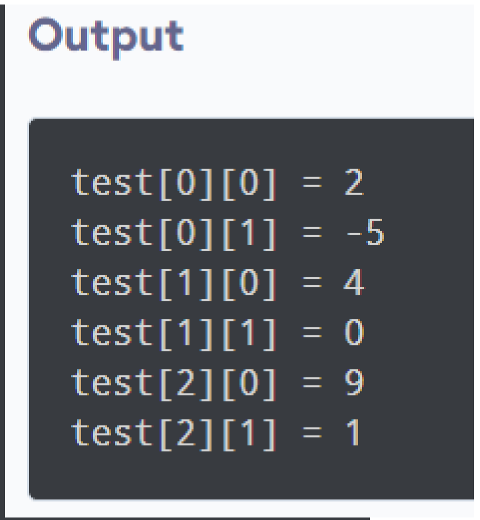
- This array x can hold a maximum of 24 elements.
- We can find out the total number of elements in the array simply by multiplying its dimensions:
    - 2 x 4 x 3 = 24

```
string letters[2][2][2] = {
  {
    { "A", "B" },
    { "C", "D" }
  },
  {
    { "E", "F" },
    { "G", "H" }
  }
};
```

Example: Print Two-Dimensional Array



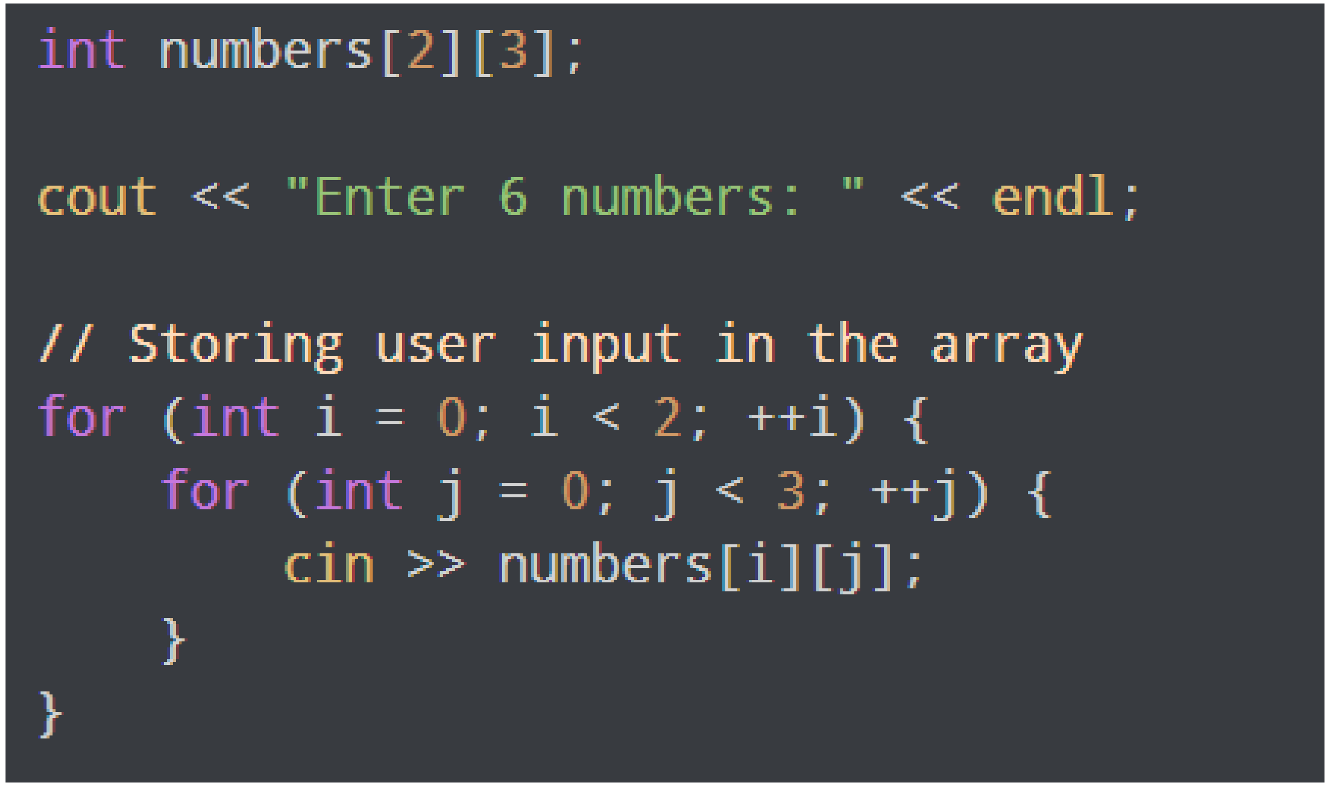
```
int test[3][2] = {{2, -5},
                  {4, 0},
                  {9, 1}};

// use of nested for loop
// access rows of the array
for (int i = 0; i < 3; ++i) {

    // access columns of the array
    for (int j = 0; j < 2; ++j) {
        cout << "test[" << i << "][" << j << "] = " << test[i][j] << endl;
    }
}
```



Output

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```

Example: Taking Input for 2-D Array

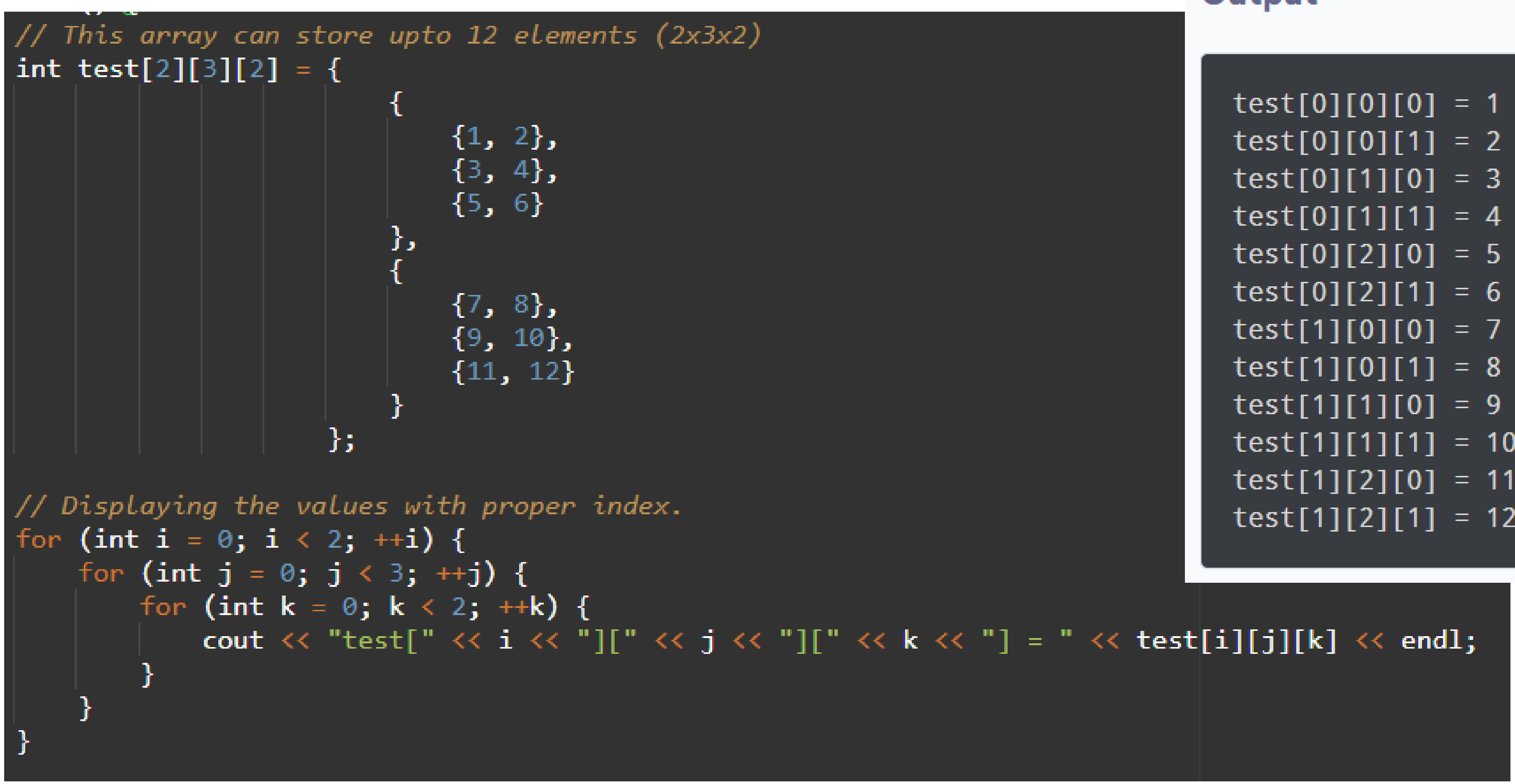


```
int numbers[2][3];

cout << "Enter 6 numbers: " << endl;

// Storing user input in the array
for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 3; ++j) {
        cin >> numbers[i][j];
    }
}
```

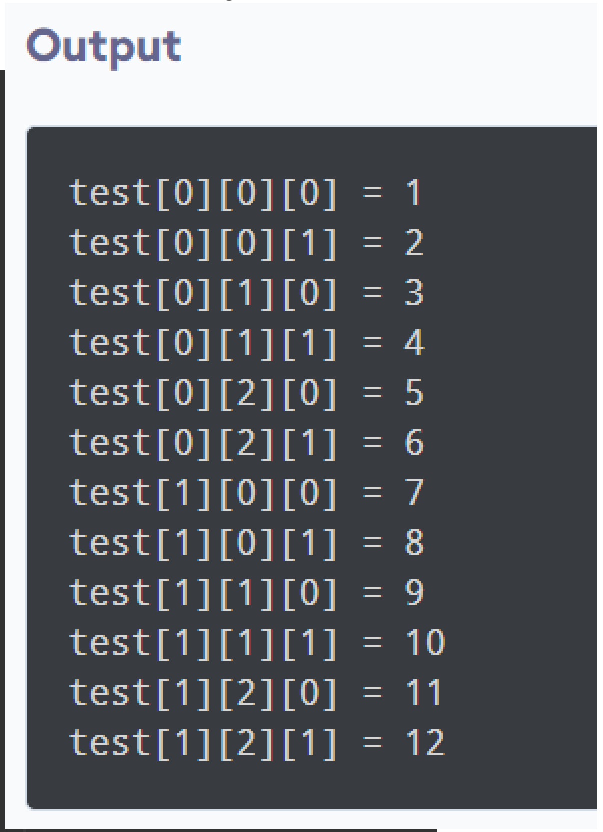
Practice

- Take Inputs from User and Store Them in an 2-D Array
- Print array elements

Example: Print Three-Dimensional Array



```
// This array can store upto 12 elements (2x3x2)
int test[2][3][2] = {
                        {
                            {1, 2},
                            {3, 4},
                            {5, 6}
                        },
                        {
                            {7, 8},
                            {9, 10},
                            {11, 12}
                        }
                    };

// Displaying the values with proper index.
for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 3; ++j) {
        for (int k = 0; k < 2; ++k) {
            cout << "test[" << i << "][" << j << "][" << k << "] = " << test[i][j][k] << endl;
        }
    }
}
```



Output

```
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```