**Arrays in C++**

* Array is a data structure which contains the collection of similar type of elements
* Elements in the array are stored in contiguous memory location.
* Index / subscript is used to keep track all the elements in the array
* Index starts from 0 and ranges upto n-1 when n elements are in the array.
* Array is a Composite / Reference / Derived data type.
* Primitive values or objects can be stored in an array in Java.
* Like C/C++, single dimensional or multidimensional arrays are created in Java.
* Fixed set of elements can be stored in a Java array.



### Advantages

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

### Disadvantages

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

### Types of Array

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

## Single Dimensional Array

**Syntax to Declare an Array**

dataType[size] arr; (or)

dataType [size]arr; (or)

dataType arr[size];

**Example :**

* + Array declaration
    - int [10 ] arr;
  + This can be rewritten as,

int[10 ] arr ;

int arr[ 10 ];

**Memory requirement for the array**

* The array contains 10 elements of integer data type, so the memory requirement will be
  + 10 \* 2 = 20 bytes

10 ----- Size of the array

2 ------ Size of int datatype

* **Size of the array**

Number of elements stored in the array

### ****How to Find the Size of an Array in C++****

In C++, you can find the **number of elements** in a **statically declared array** using the following formula:

sizeof(arr) / sizeof(arr[0])

* **Initialization of values in single dimensional array**

**int** a[ ]= {10, 20, 30, 40 ,50 }; //declaration,  instantiation and  initialization

* **Assigning values in the array**

a [ 0 ] = 10;

a [ 1 ] = 20;

a [ 2 ] = 30;

a [ 3 ] = 40;

a [ 4 ] = 50;

* **Assigning values in an array using for loop**

for ( int i = 0; i < n ; i++ )

{

cin << a [ i ] ;

}

* **Printing the values in the array**

for ( int i = 0; i < n ; i++ )

{

cout << a [ i ] ;

}

* **Accessing the array elements using for . . . . each loop**

int arr [10];

**for** (**int** i : arr)

cout<< i <<endl;

**/\* Program to declare an array of size ‘n’ and find out the sum of array elements and average of them \*/**

**#include <iostream>**

**using namespace std;**

int main ( )

{

int sum = 0, n ;

float avg;

cout << “Enter the number of elements : “;

cin >> n ;

**int** a[n]; //Array declaration and instantiation

cout << “ Enter the array elements”;

// Accessing the array elements

for ( int i = 0; i < n ; i++ )

{

cin >> a [ i ] ;

}

// Processing the array elements

for ( int i = 0; i < n ; i++ )

{

sum += a [ i ] ;

}

avg = (float )sum / n;

//Printing array

cout >> “Array Elements” ;

**for**(**int** i=0; i< n ; i++)

cout << a [ i ] ;

  cout << “Sum of array = “ << sum ;

cout << “Average of array = “ << avg;

}

* **Passing array to the function in C++**
  + Array can be passed to the function as one of the argument like other data types
  + Passing array to the function is called as Pass/Call By reference
    - The address/reference of the array is passed to the function
    - If any changes are made to the array elements in the function it affects the original value
    - When an array is passed to a function, what actually gets passed is a **pointer to its first element**.

/\* Java Program to demonstrate the way of passing an array to method

Program to find the largest element in the array by passing the function to the array \*/

**class** ArraysToFunction {

//creating a method which receives an array as a parameter

**void** largest( **int** arr[ ] ) {

**int** large=arr[0];

**for**(**int** i=1; i<arr.length; i++)

**if**( arr[i] > large )

   large = arr[i] ;

cout<<“Largest value in the array = “ << large<<endl;

}

**void** main ( String args[ ] ) {

**int** a[ ] = {-27, 78 ,56 -12 , 9, 33}; //declaring and initializing an array

largest ( a ); //passing array to method

}

}

**Method 1: Passing the array as a Pointer**

Example :

#include <iostream>

using namespace std;

void printArray(int \*arr, int size) {

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

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

}

cout << endl;

}

int main() {

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

int length = sizeof(numbers) / sizeof(numbers[0]);

printArray(numbers, length);

return 0;

}

### Method 2 : ****Method 2: Using Array Syntax****

void printArray(int arr[], int size) {

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

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

}

}

NOTE: int arr[] and int \*arr are treated the same inside the function.

### ****For-each Loop in C++**** (also called ****Range-based for loop****)

Introduced in **C++11**, the **for-each loop** simplifies iterating over arrays or containers like vector, array, etc.

### 🔹 ****Syntax:****

for (datatype variable : collection) {

// statements using variable

}

### ****Example with Array:****

#include <iostream>

using namespace std;

int main() {

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

for (int num : numbers) {

cout << num << " ";

}

return 0;

}

**Output:**

10 20 30 40 50

### ****Example with Vector:****

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<string> names = {"Alice", "Bob", "Charlie"};

for (string name : names) {

cout << name << endl;

}

return 0;

}

### ****Using Reference (to Modify Elements):****

cpp

CopyEdit

for (int &num : numbers) {

num = num \* 2; // modifies the original array

}

### ****Advantages of For-each Loop:****

* Cleaner and simpler syntax.
* Avoids using index variables.
* Works with arrays, vectors, maps, sets, and other STL containers.
* **Multidimensional ( 2 – Dim ) array**
* The 2-Dim array contains row and column values, where as each element in the array is keep tracked with row and column index.
* 2-Dim array declaration

dataType arrayRefVar[row ][column ];

**Example to instantiate Multidimensional Array**

**int mat**[ row ][ column ] ; // 3 row and 3 column

**Example to initialize Multidimensional Array**

arr[0][0]=11;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=14;

arr[1][1]=5;

arr[1][2]=36;

arr[2][0]=7;

arr[2][1]=28;

arr[2][2]=9;

These values are arranged as a matrix like this:

11 2 3

14 5 36

7 28 9

### Example of Multidimensional Array

Let's see the simple example to declare, instantiate, initialize and print the 2-Dimensional array.

// Program to illustrate the use of multidimensional array

#include <iostream>

#include <vector>

using namespace std;

int main() {

//declaring and initializing 2D array

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

//printing 2D array

**for**(**int** i=0;i<3;i++) // Loop for rows

{

**for**(**int** j=0;j<3;j++) // Loop for columns

{

    cout << arr[i][j] << "  " );

 }

  cout << “ \n “;

}

}

**Program to add 2 matrices**

#include <iostream>

using namespace std;

int main() {

int rows1, cols1, rows2, cols2;

cout << "Enter number of rows and columns of 2 matrices: ";

cin >> rows1 >> cols1 >>rows2 >>cols2;

int matrix1[10][10], matrix2[10][10], sum[10][10];

if( rows1==rows2 && cols1==cols2)

{

cout << "\nEnter elements of first matrix:\n";

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols1; j++) {

cin >> matrix1[i][j];

}

}

cout << "\nEnter elements of second matrix:\n";

for (int i = 0; i < rows2; i++) {

for (int j = 0; j < cols2; j++) {

cin >> matrix2[i][j];

}

}

// Adding the matrices

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols1; j++) {

sum[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

// Displaying the result

cout << "\nSum of the two matrices:\n";

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols1; j++) {

cout << sum[i][j] << "\t";

}

cout << endl;

}

}

else

{

cout << “Matrix addition is not possible”;

}

return 0;

}

**Question:**

* 1. Program to multiply the matrix value by a unique number and display the resultant matrix
  2. Program to find the sum of the diagonal values in the matrix