INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101
Programming with C++

Module-3:
Aggregate Data-types

About Subject

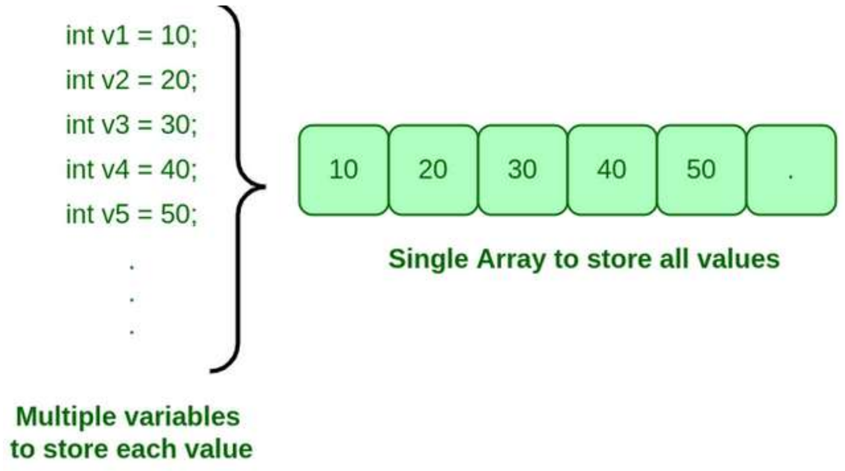
- Aggregate Data-types:
  - Arrays
  - Pointers
  - Structures
  - Dynamic data and Pointers
  - Dynamic arrays

Why Arrays

- If I want to store a list of marks of all the students in EEC-101, I need to declare 204 variables (one for each student); how long will it take to write the declaration part by using normal variable declaration?
- For example, int mark1, mark2, mark3, ..., mark204;

- It would be challenging to manipulate and maintain the data.

- **The idea of an array is to represent many instances in one variable.**

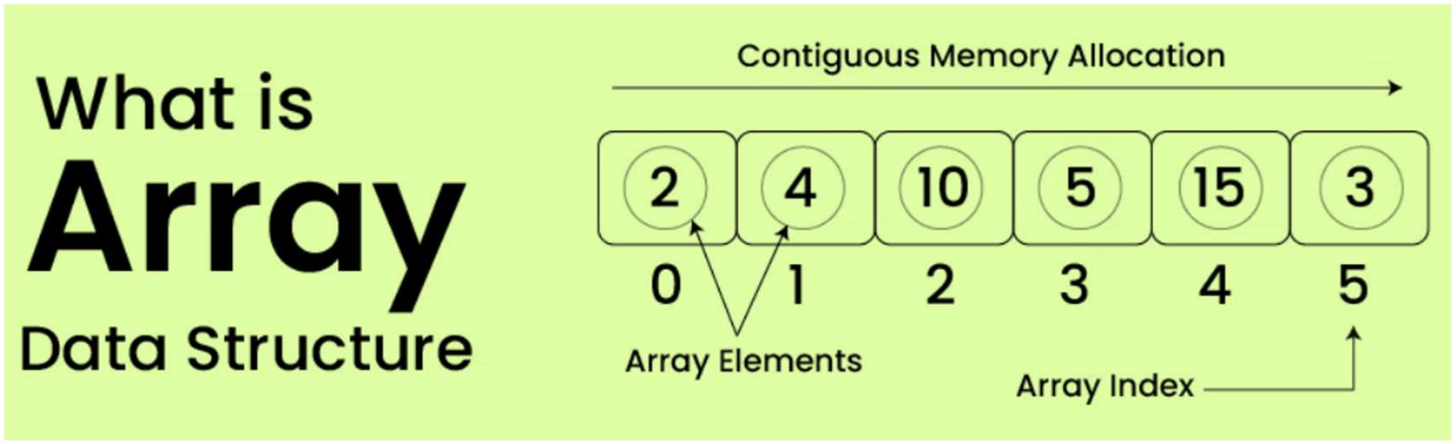

int v1 = 10;
int v2 = 20;
int v3 = 30;
int v4 = 40;
int v5 = 50;
10
20
30
40
50
Single Array to store all values
Multiple variables to store each value

Arrays

- Array is a linear data structure where all elements are arranged sequentially.

- It is a collection of elements of the same data type stored at contiguous memory locations

sharing a common border; touching.
next or together in sequence.

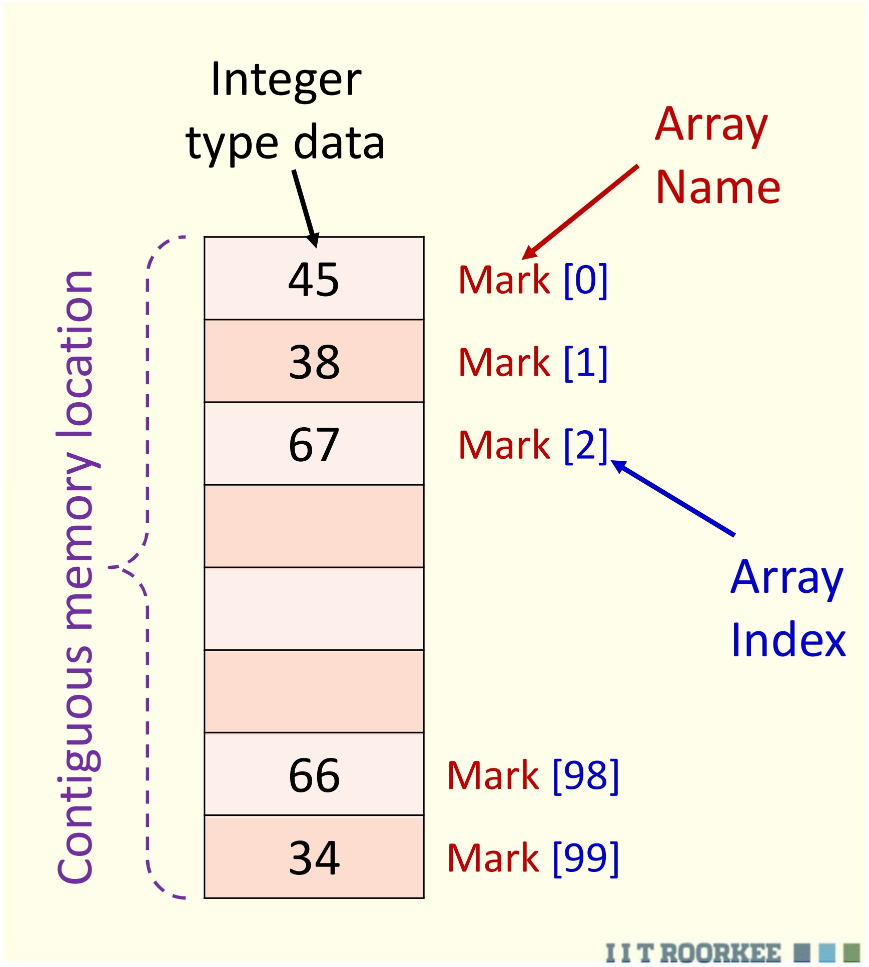


Array Declaration

- By using an array, we just declare like this:

  int mark[100];

- This will reserve 100 contiguous/sequential memory locations for storing the integer data type.



Arrays

- Array is a contiguous space in memory where all the locations are referred to by a group name.

- Array name is a collective name for all locations in the array.

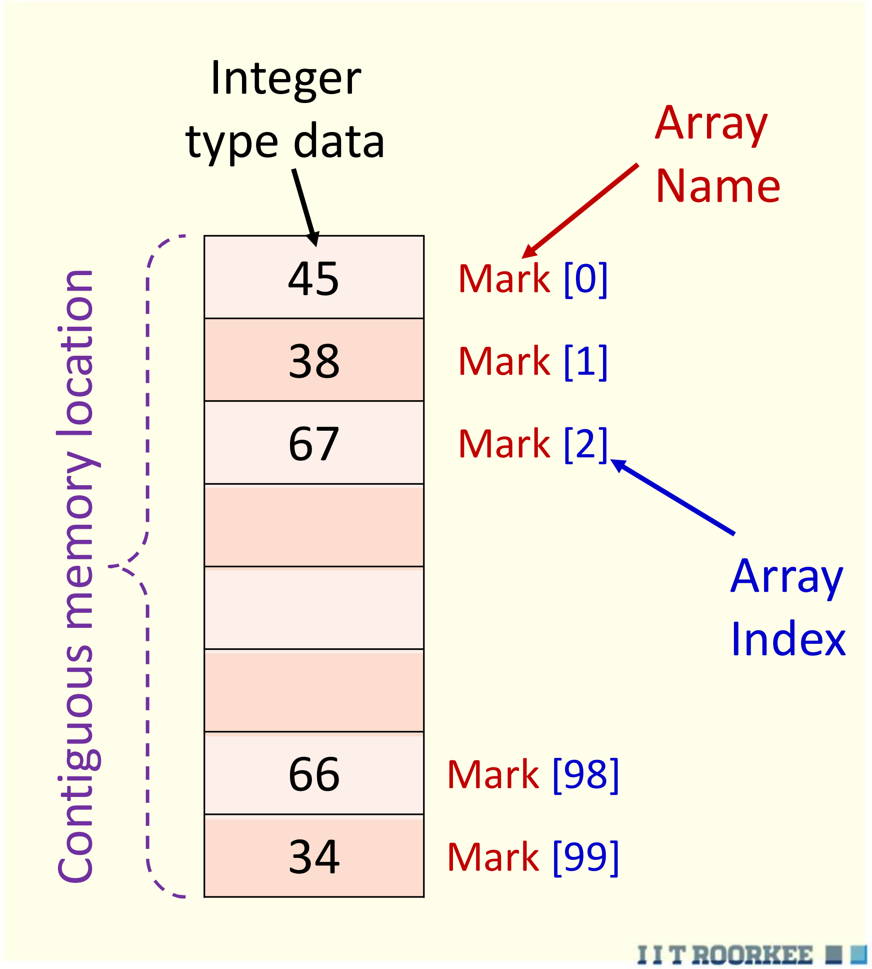
- The collection is homogeneous, i.e., all elements are of the same type.

- A particular location is referred to by the array name and an index value.
  - index values start from 0
  - index is also known as subscript

- **for declaring an array, mention**
  - **the type of elements**
  - **array name**
  - **size of array**

Basic Terminologies

- Array Index: In an array, elements are identified by their indexes. Array index starts from 0.
- Array Element: Elements are items stored in an array and can be accessed by their index.
- Array Length: The length of an array is determined by the number of elements it can contain.



Characteristics

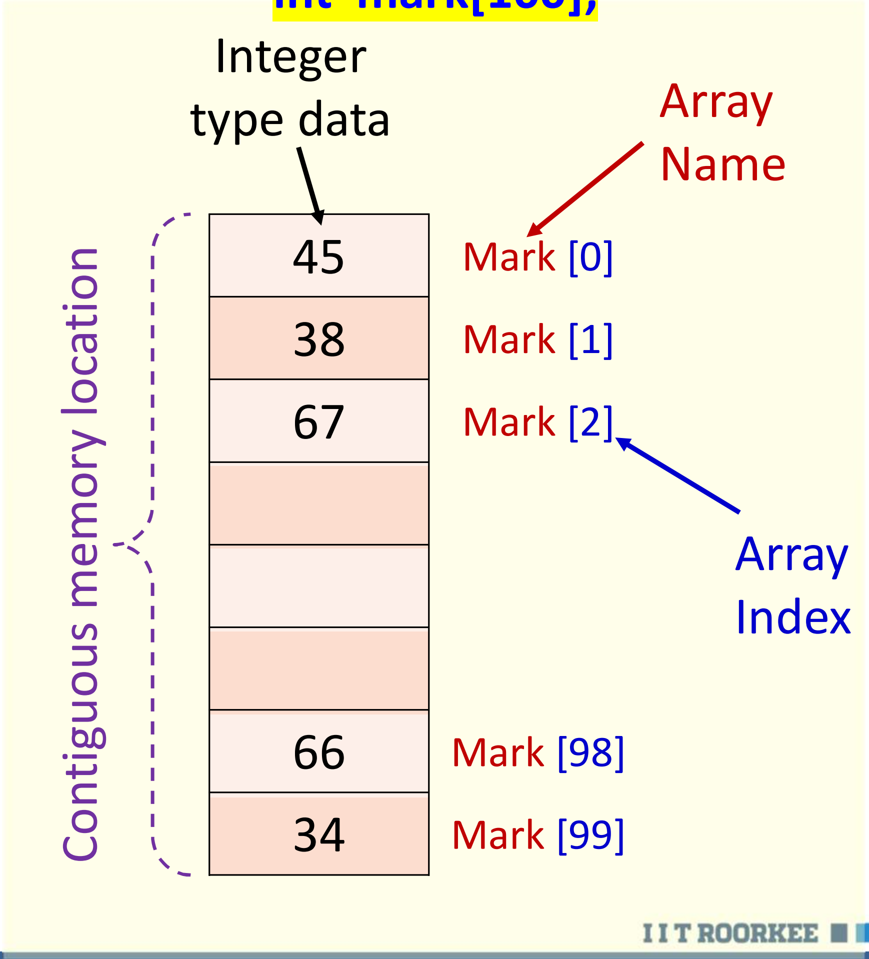
- Fixed size
- Elements are all the same type
- Stored contiguously in memory
- Individual elements can be accessed by their position or index

- First element is at index 0
- Last element is at index size-1

- No checking to see if you are out of bounds

- Always initialize arrays
- Very efficient
- Iteration (looping) is often used to process

```
int  mark[100];
```



Characteristics

- N-element array c

  `c[0],c[1] … c[n - 1]`

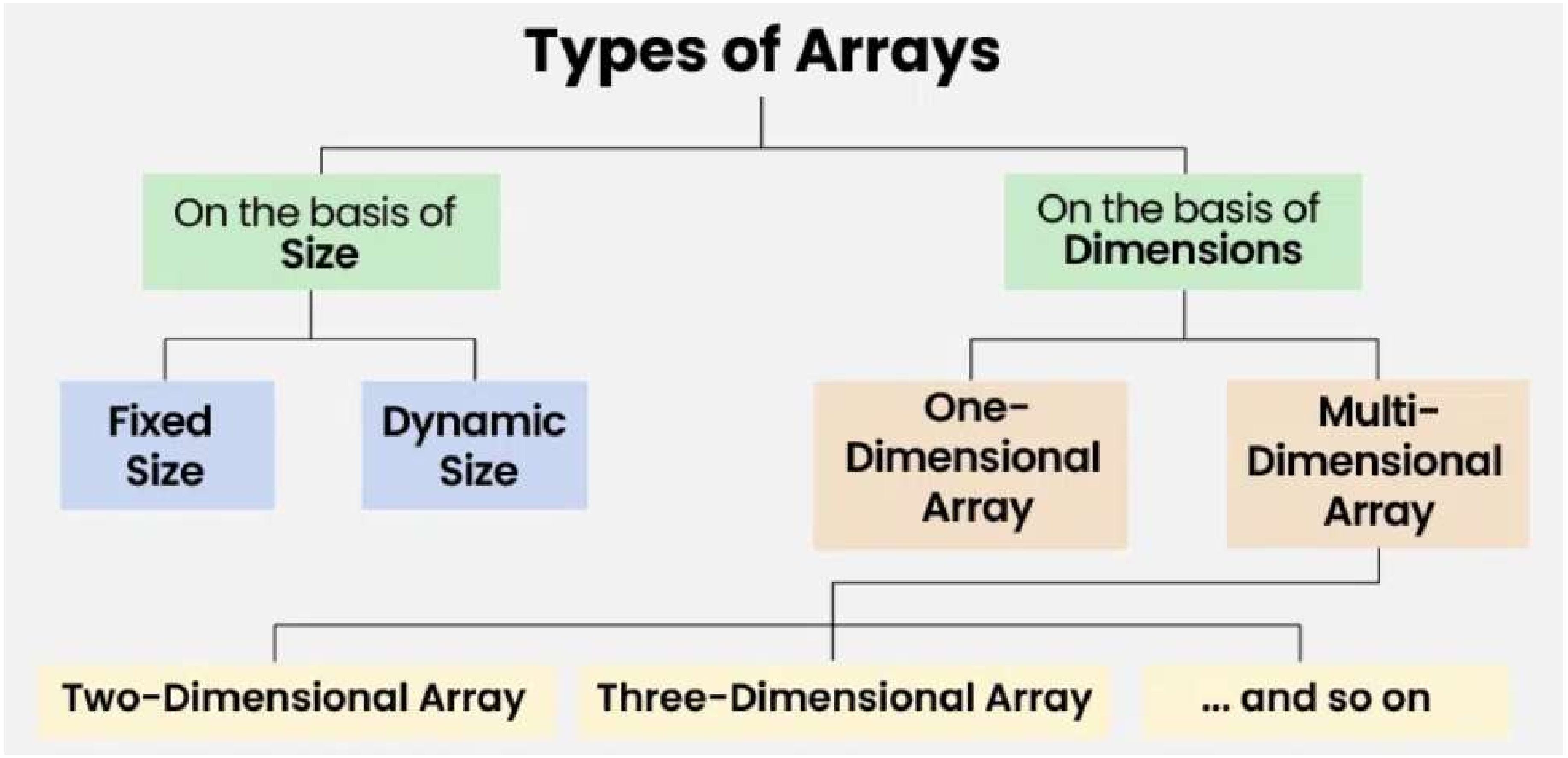
  - N^th element at position N-1

- Array elements are like other variables
  - Assignment, printing for an integer array **c**

```
c[0] =  3;
cout << c[ 0 ];
```

- Can perform operations inside subscript

  `c[ 5 – 2 ]` same as `c[3]`


Types of Arrays
On the basis of Size
Fixed Size
Dynamic Size
On the basis of Dimensions
One-Dimensional Array
Multi-Dimensional Array
Two-Dimensional Array
Three-Dimensional Array
... and so on

Declaring Arrays (1-D Array)

- **When declaring arrays, specify**
  1. **Name**
  2. **Type of array-Data type**
  3. **Number of elements**

  – *Syntax: Datatype ArrayName* [*ConstantExpression*] ;

```
int d[ 20];  // array of 20 integers
float e[ 4325 ]; // array of 4325 floats
int value[7*j+4];// index is an expression=11 for j=1
```

- **Declaring multiple arrays of the same type**
  - **Use comma separated list, like regular variables**

```
int b[ 200 ], y[ 38 ];
```

Initializing Arrays

1. For loop
   - Set each element
2. Initializer list
   - Specify each element when array declared `int age[ 5 ] = {18,20,17,21,19};`
   - If not enough initializers, rightmost elements 0
3. To set every element to 0
   - `int n[ 5 ] = { 0 };`
   - We can initialize the array with all elements as ‘0’ by specifying ‘0’ inside the curly braces. This will happen in the case of zero only. If we try to initialize the array with a different value, say ‘2’ using this method, then ‘2’ is stored at the 0th index only.
4. If array size is omitted, initializers determine the size

   ```
   int n[] = { 1, 2, 3, 4, 5 };
   ```

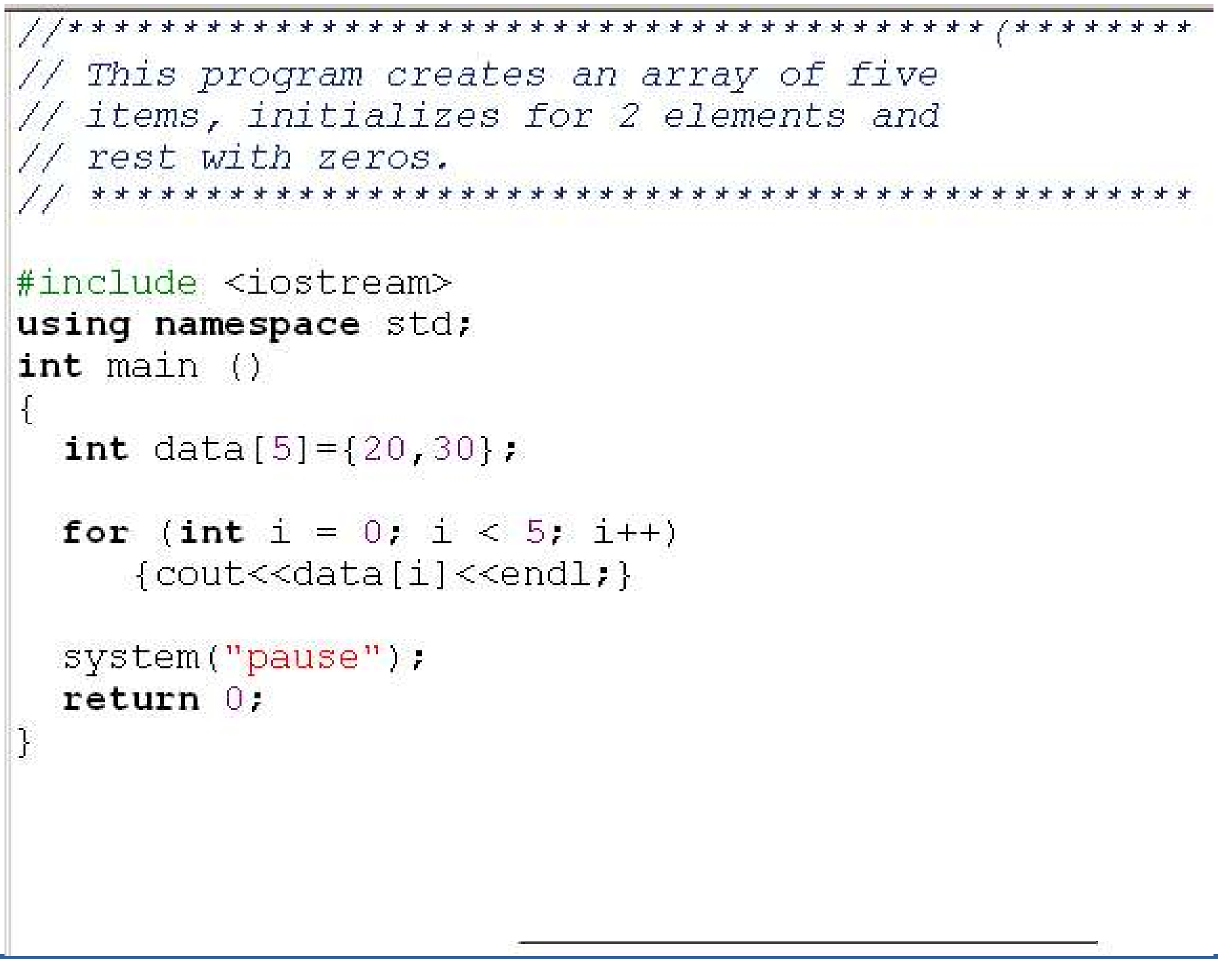
   - 5 initializers, therefore 5 element array

Accessing an Element of an Array in C++

- Elements of an array can be accessed by specifying the name of the array and then the index of the element enclosed in the array subscript operator [].

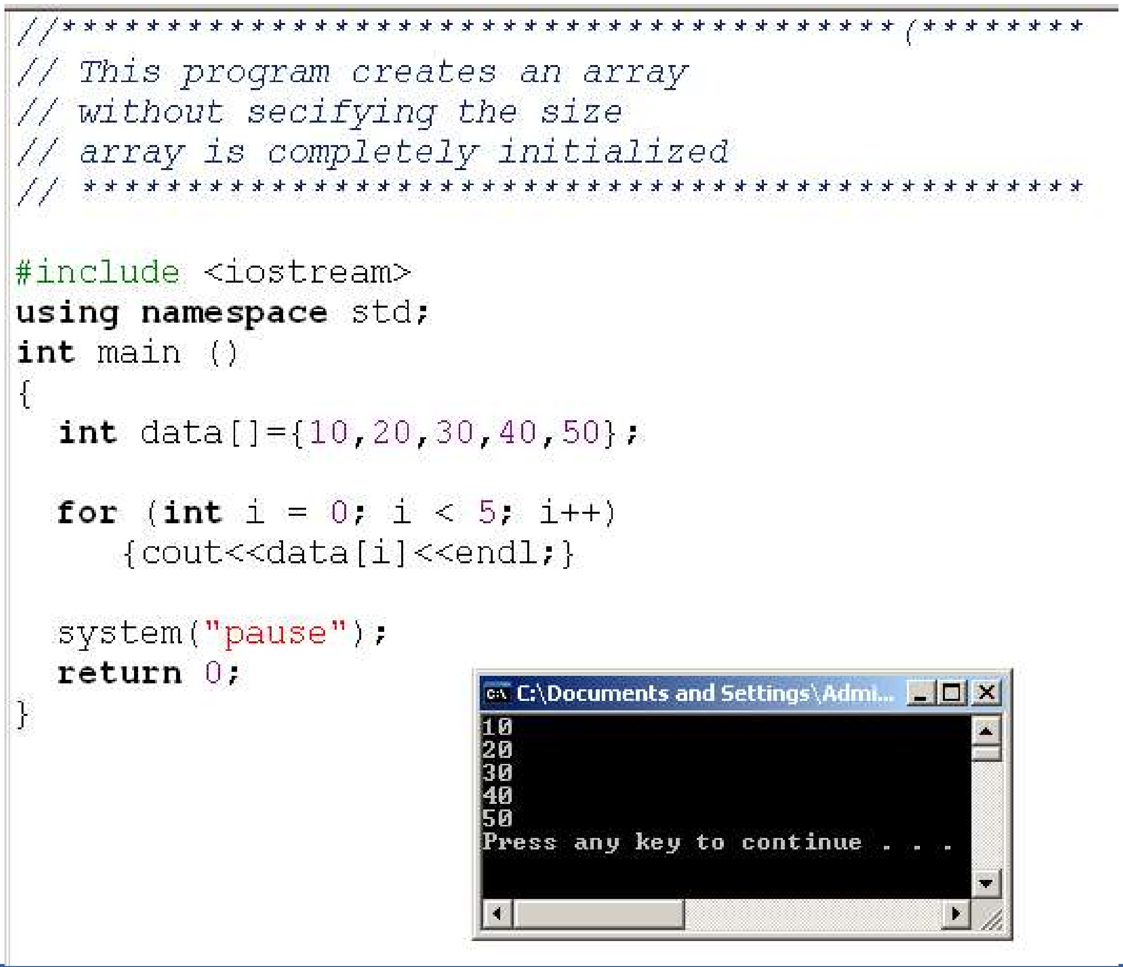
- For example, cout<<mark[3];

Example

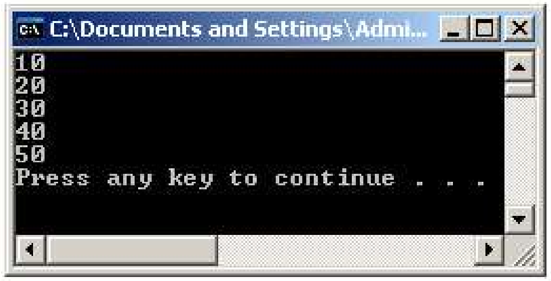


```
//*****************************************(********
// This program creates an array of five
// items, initializes for 2 elements and
// rest with zeros.
// ***********************************************

#include <iostream>
using namespace std;
int main ()
{
  int data[5]={20,30};

  for (int i = 0; i < 5; i++)
     {cout<<data[i]<<endl;}

  system("pause");
  return 0;
}
```

Example

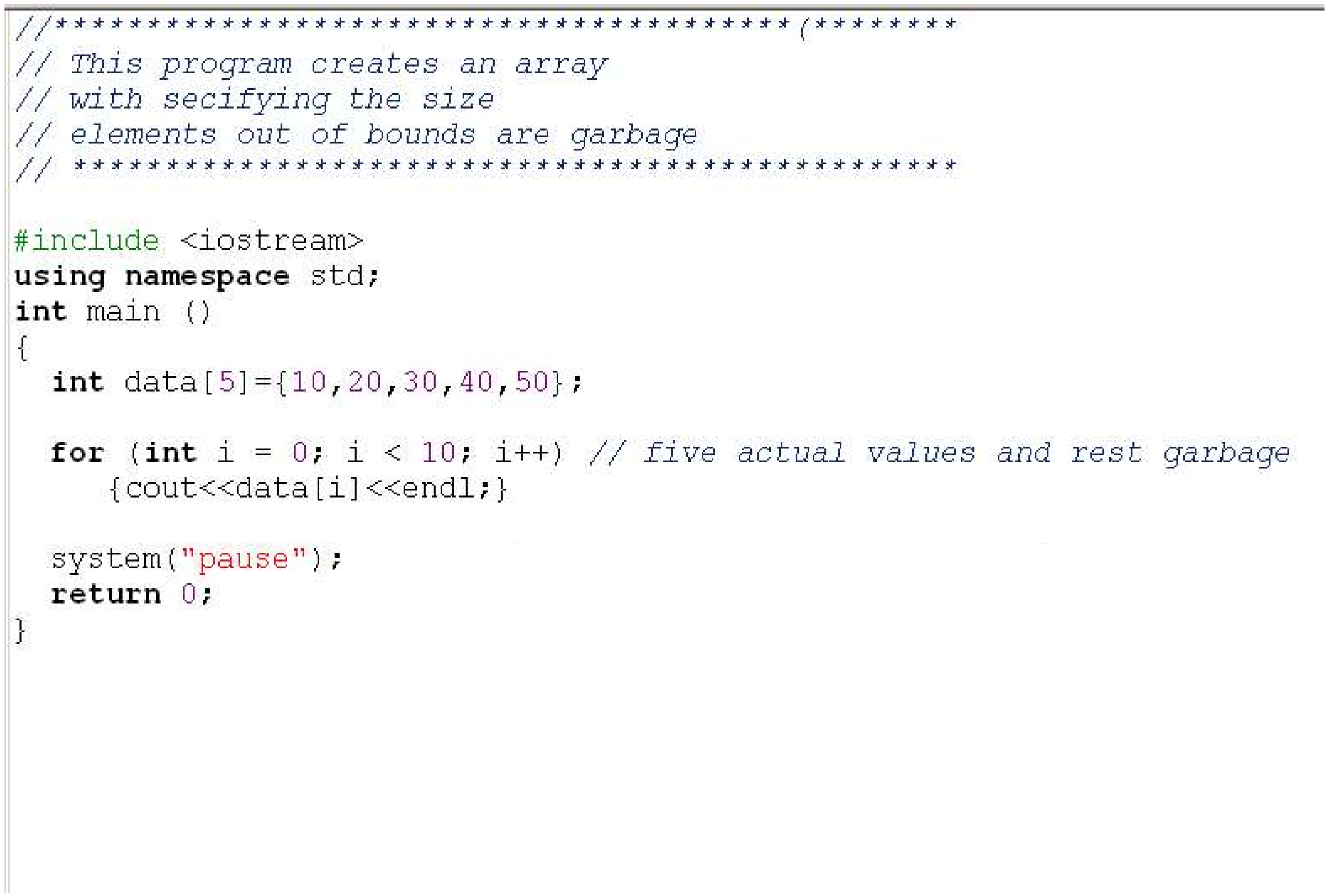


```
//*****************************************(********
// This program creates an array
// without secifying the size
// array is completely initialized
// ***************************************************

#include <iostream>
using namespace std;
int main ()
{
  int data[]={10,20,30,40,50};

  for (int i = 0; i < 5; i++)
     {cout<<data[i]<<endl;}

  system("pause");
  return 0;
}
```



```
10
20
30
40
50
Press any key to continue . . .
```

Example



```
//****************************************(********
// This program creates an array
// with secifying the size
// elements out of bounds are garbage
// ************************************************

#include <iostream>
using namespace std;
int main ()
{
  int data[5]={10,20,30,40,50};

  for (int i = 0; i < 10; i++) // five actual values and rest garbage
     {cout<<data[i]<<endl;}

  system("pause");
  return 0;
}
```

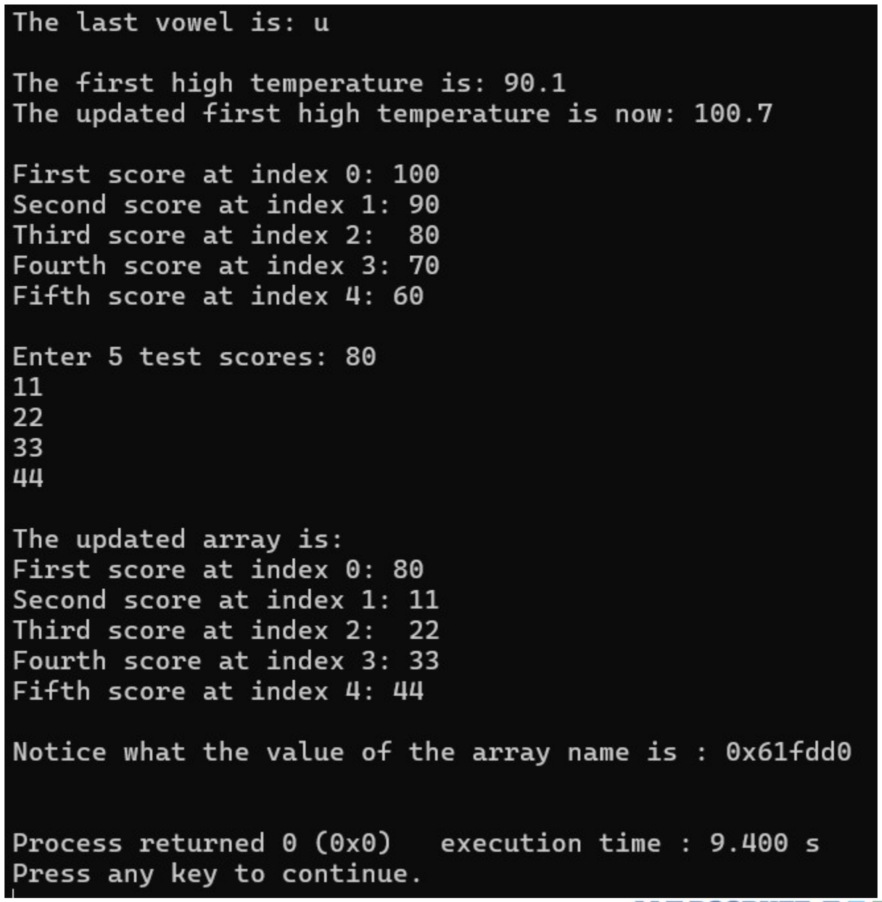
Example

For loop to assign array elements

```
// Initialize array num to the even integers from 2 to 30.
      #include <iostream>
      #include <iomanip>
      using namespace std;
      int main()
    {
      const int Size = 15;// constant or variable can be used to specif
      int num[Size];  // array s has 10 elements
      for ( int i = 0; i < Size; i++ )  // set the data values
            num[i] = 2 + 2 * i;

// output contents of array s in tabular format
       cout << "Element" << setw( 13 ) << "Value" << endl;
      for ( int j = 0; j < Size; j++ )
            cout << setw( 7 ) << j << setw( 13 ) << num[ j ] << endl;
       return 0;  // indicates successful termination
    } // end main
```

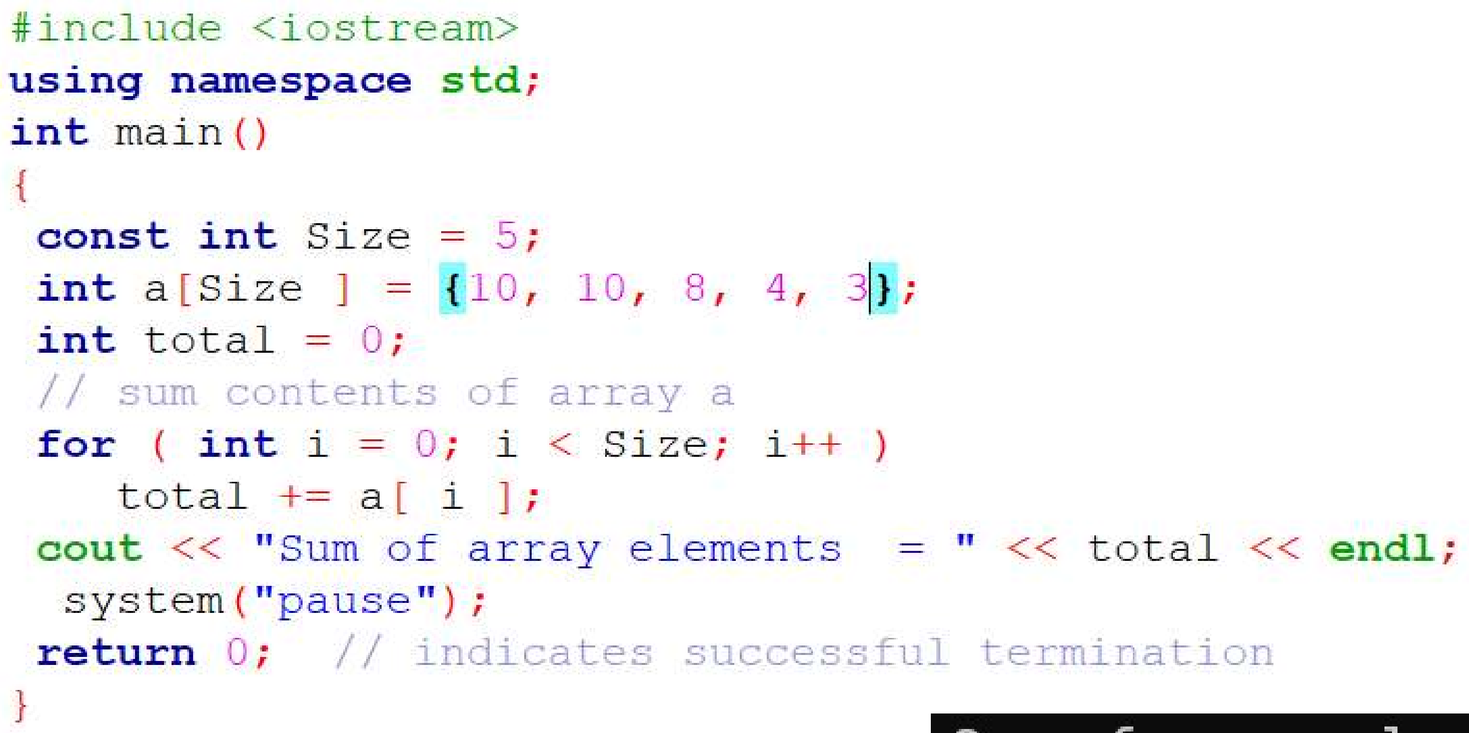
Example

```cpp
int main() {

    char vowels[]  {'a' ,'e', 'i', 'o', 'u' };
    cout << "\nThe first vowel is: " << vowels[0] << endl;//first index is 0
    cout << "The last vowel is: " << vowels[4] << endl;// last index is 4

    //cin >> vowels[5];  //out of bounds - don't do this!!

    double hi_temps []  { 90.1, 89.8, 77.5, 81.6};
    cout << "\nThe first high temperature is: " << hi_temps[0] << endl;

    hi_temps[0] = 100.7;    // set the first element in hi_temps to 100.7

    cout << "The updated first high temperature is now: " << hi_temps[0] << endl;


    int test scores [] {100, 90, 80,70, 60};
   cout << "\nFirst score at index 0: " << test_scores[0] << endl;
   cout << "Second score at index 1: " << test_scores[1] << endl;
   cout << "Third score at index 2:  " << test_scores[2] << endl;
   cout << "Fourth score at index 3: " << test_scores[3] << endl;
   cout << "Fifth score at index 4: " << test_scores[4] << endl;

   cout << "\nEnter 5 test scores: ";
   cin >> test_scores[0];
   cin >> test_scores[1];
   cin >> test_scores[2];
   cin >> test_scores[3];
   cin >> test_scores[4];

   cout << "\nThe updated array is:" << endl;
   cout << "First score at index 0: " << test_scores[0] << endl;
   cout << "Second score at index 1: " << test_scores[1] << endl;
   cout << "Third score at index 2:  " << test_scores[2] << endl;
   cout << "Fourth score at index 3: " << test_scores[3] << endl;
   cout << "Fifth score at index 4: " << test_scores[4] << endl;

   cout << "\nNotice what the value of the array name is : " << test_scores << endl;
   cout << endl;
   return 0;
```

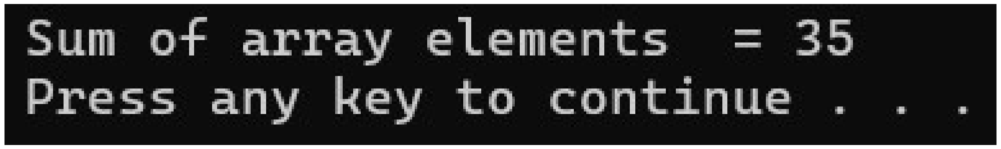


```
The last vowel is: u

The first high temperature is: 90.1
The updated first high temperature is now: 100.7

First score at index 0: 100
Second score at index 1: 90
Third score at index 2:  80
Fourth score at index 3: 70
Fifth score at index 4: 60

Enter 5 test scores: 80
11
22
33
44

The updated array is:
First score at index 0: 80
Second score at index 1: 11
Third score at index 2:  22
Fourth score at index 3: 33
Fifth score at index 4: 44

Notice what the value of the array name is : 0x61fdd0


Process returned 0 (0x0)   execution time : 9.400 s
Press any key to continue.
```

WAP to find sum of elements in an array



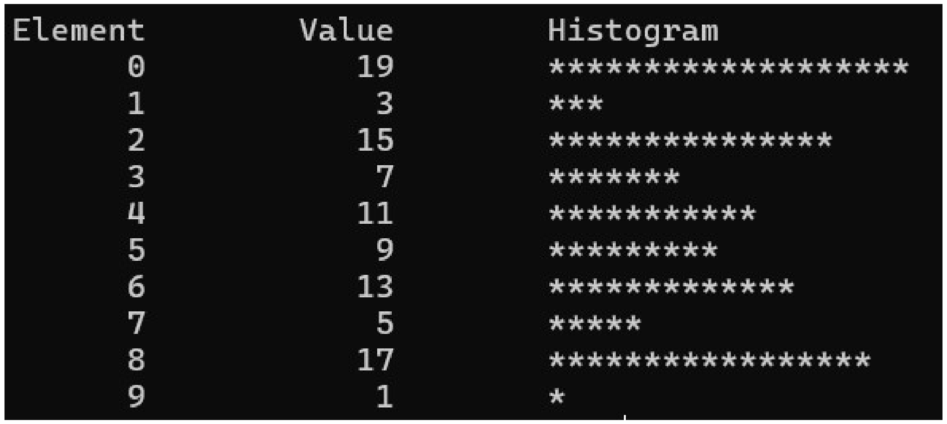
```cpp
#include <iostream>
using namespace std;
int main()
{
 const int Size = 5;
 int a[Size ] = {10, 10, 8, 4, 3};
 int total = 0;
 // sum contents of array a
 for ( int i = 0; i < Size; i++ )
    total += a[ i ];
 cout << "Sum of array elements  = " << total << endl;
  system("pause");
 return 0;  // indicates successful termination
}
```



```
Sum of array elements  = 35
Press any key to continue . . .
```

WAP to print Histogram of array value

```cpp
int main()
{
   const int arraySize = 10;
   int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };

   cout << "Element" << setw( 13 ) << "Value"
        << setw( 17 ) << "Histogram" << endl;

   // for each element of array n, output a bar in histogram
   for ( int i = 0; i < arraySize; i++ ) {
      cout << setw( 7 ) << i << setw( 13 )
           << n[ i ] << setw( 9 );

      for ( int j = 0; j < n[ i ]; j++ )   // print one bar
         cout << '*';
      cout << endl;  // start next line of output

   } // end outer for structure
   system("pause");
   return 0;  // indicates successful termination
```



```
Element          Value            Histogram
      0             19            *******************
      1              3            ***
      2             15            ***************
      3              7            *******
      4             11            ***********
      5              9            *********
      6             13            *************
      7              5            *****
      8             17            *****************
      9              1            *
```

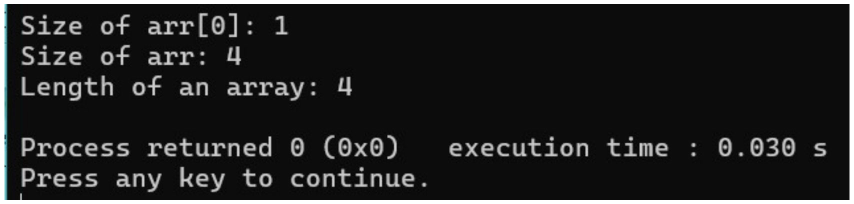
Example

- WAP to find the size of an array?

~~int [n] = {~~

int n [ ] = { }

```cpp
// C++ Program to Illustrate How to Find the Size of an
// Array
#include <iostream>
using namespace std;
int main()
{
    //int arr[] = { 1, 2, 3, 4, 5,6,7,8,9,10};
    char arr[] = { 'a', 'b', 'a', 'd'};
    // Size of one element of an array
    cout << "Size of arr[0]: " << sizeof(arr[0]) << endl;
    // Size of array 'arr'
    cout << "Size of arr: " << sizeof(arr) << endl;
    // Length of an array
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Length of an array: " << n << endl;
    return 0;
}
```



```
Size of arr[0]: 1
Size of arr: 4
Length of an array: 4

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Out of Bound Array Indexes

- This is the case when an index value is either less than 0 or greater than the (array size minus -1)

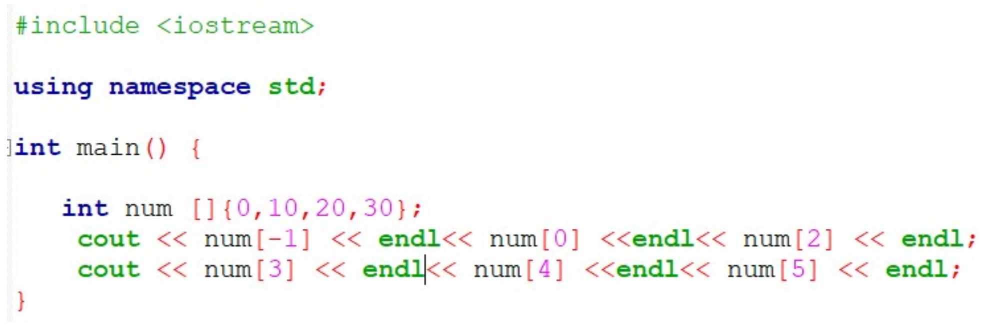
- Example

float alpha[100];

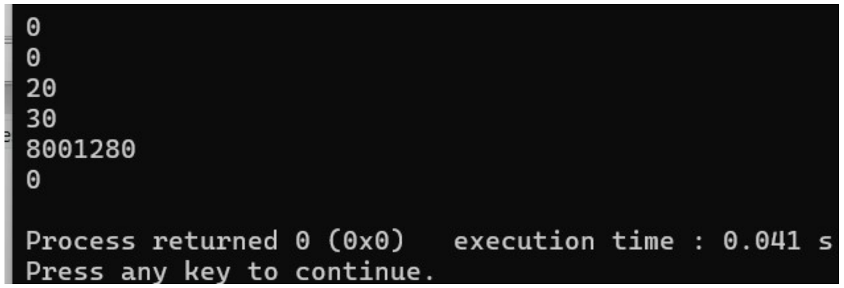
alpha[i]=65.8;

  – if i is less than 0 or when i is greater than 99, memory location outside the array is accessed. C++ does not check for invalid(out-of-bounds)
  – if i = 100, then 65.8 is stored in the next memory location past the end of the array

Example



```
#include <iostream>

using namespace std;

int main() {

   int num [] {0,10,20,30};
    cout << num[-1] << endl<< num[0] <<endl<< num[2] << endl;
    cout << num[3] << endl<< num[4] <<endl<< num[5] << endl;
}
```



```
0
0
20
30
8001280
0

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

Aggregate Operations

```
int x[50];
int y[50];
x=y ; // Not valid


// to copy array use this
for (int index =0; index < 50;index++)
    x[index]=y[index];

if (x==y) // Not valid
cout<<x; // Not valid to print all elements (displays the base  address)
 x= x+y; // Not valid
```

Array and Functions

- Arrays can be passed as parameters to a function
- Arrays automatically use pass-by-reference

  void processArray(int n[ ], int size); OR

  void processArray(int [ ], int);
- **The size of an array is not given with the array parameter**
- A typical call to the function is: processArray(n, 5);
- The size of array is passed separately so that only a specific number of elements are processed
- To prevent array modification in the calling environment, use pass-by-const-reference

Array and Functions

- To process arrays in a large program, we need to pass them to functions
  - Passing individual elements
  - Passing the whole array

Passing Individual Element

- Passed to a function like an ordinary variable

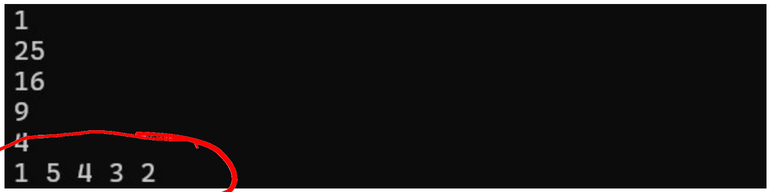
- As long as the array element type matches the function parameter type, it can be passed

- It will be passed as a Value Parameter (Pass by value)

(means the function can not change the value of the element in the calling function)

Example

```cpp
//passing individual array element
#include <iostream>
using namespace std;
void square (int);
int main()
{
    int arr[] = { 1, 5, 4, 3, 2};
    for (int i=0;i<5;i++)
        square (arr[i]);
    for (int i=0;i<5;i++)
        cout<<(arr[i])<<" ";
    return 0;
}
void square (int a){
cout<<a*a<<endl;
}
```



```
1
25
16
9
4
1 5 4 3 2
```

Passing Whole Array

- If passed as value to a function, it would take a lot of memory and time to pass large arrays every time we wanted to use it in the function (e.g. 10,000 elements)
- If an array containing 10,000 elements were passed by value to a function, then another 10,000 elements would have to be allocated in the function and each element would have to be copied from one array to the other
- Thus, in C++, passing the Whole Array passes the address of the array

- In C++ name of the array is a primary expression whose value is the address of the first element in the array

- Indexed references are calculated addresses.

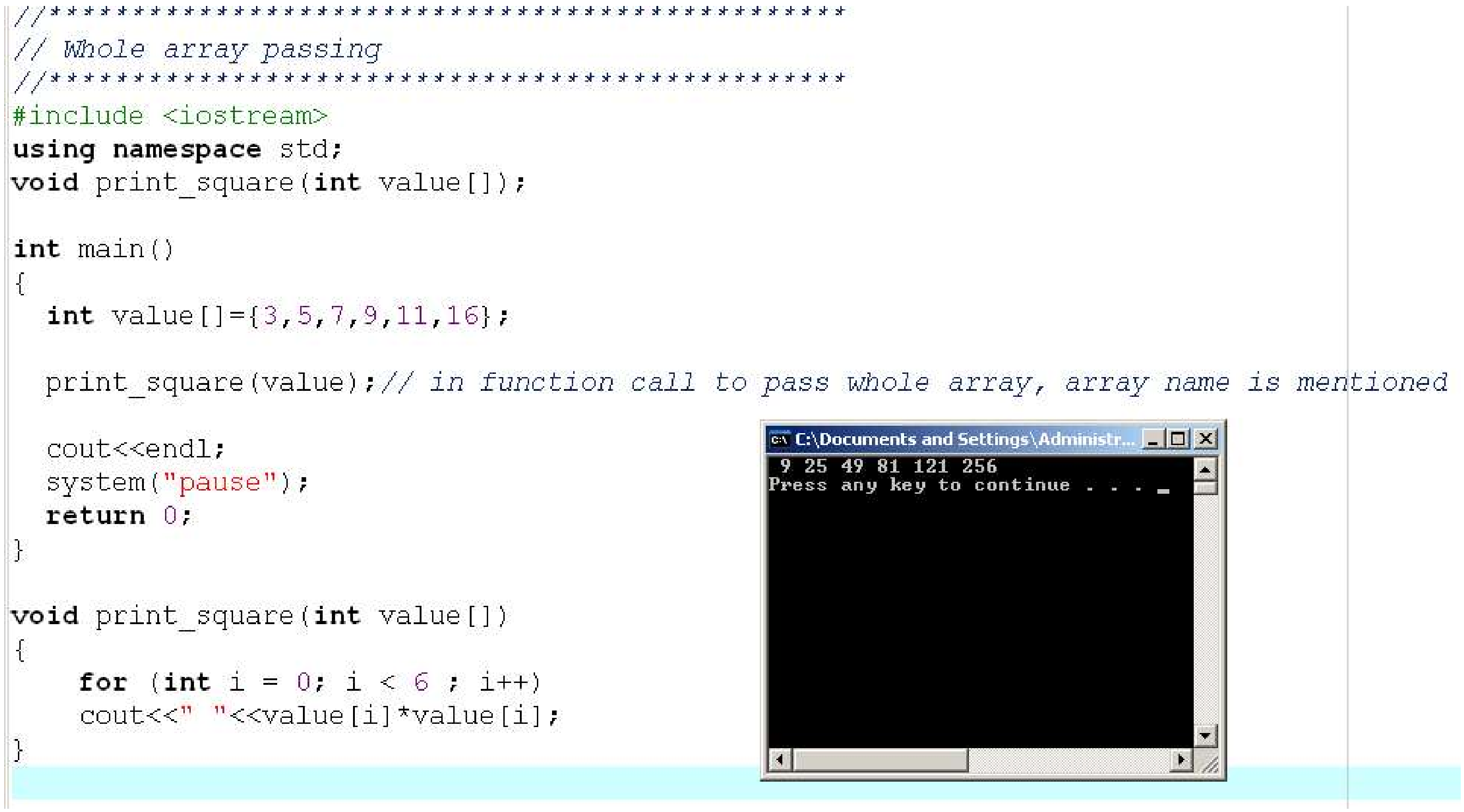
- Passing the array name allows the function to refer to the array back in the calling function

How to pass whole array?

- Function must be called by passing only the name of the array

- In the function definition,
  - the formal parameter must be an array type;
  - the size of the array need not be specified

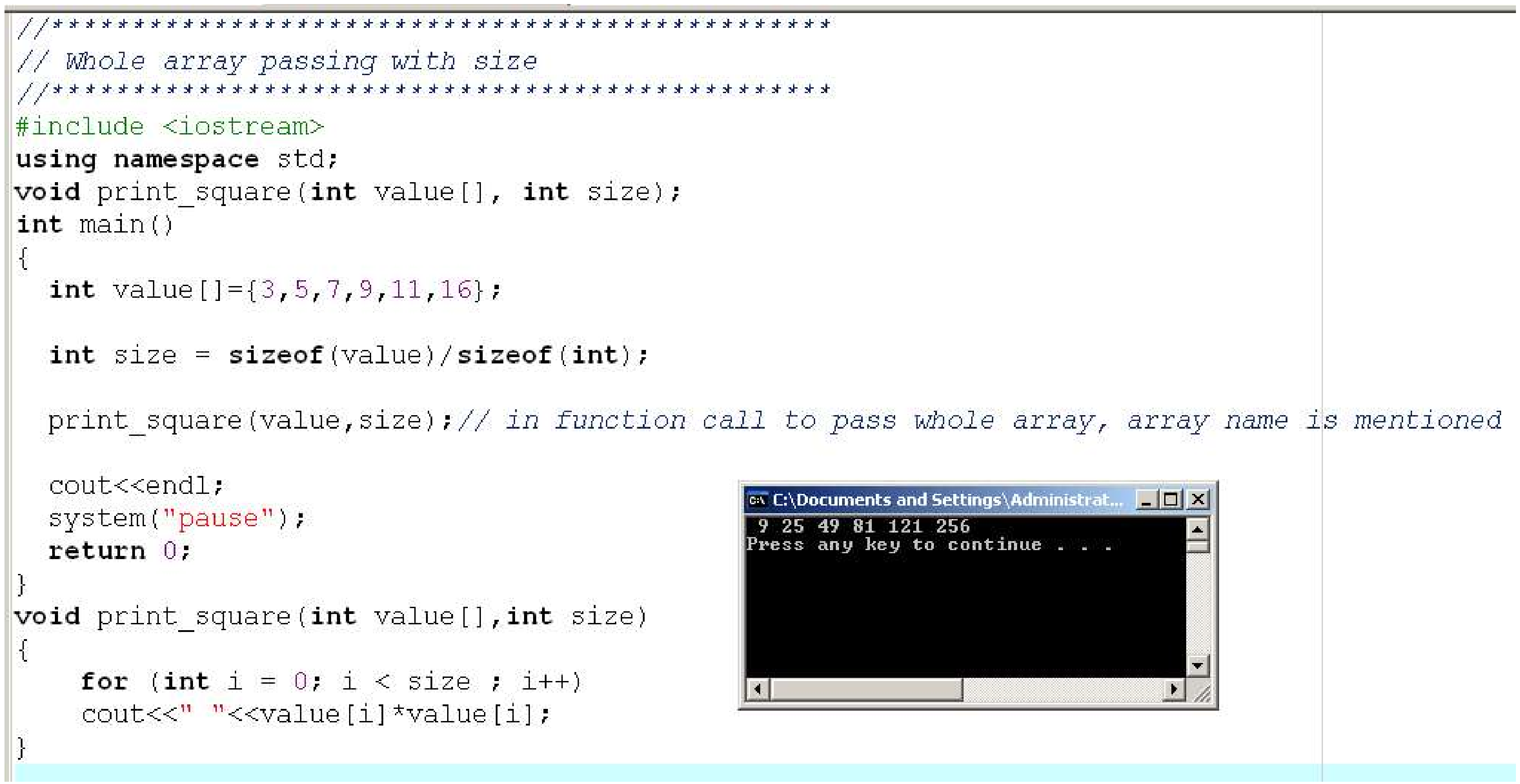
Example



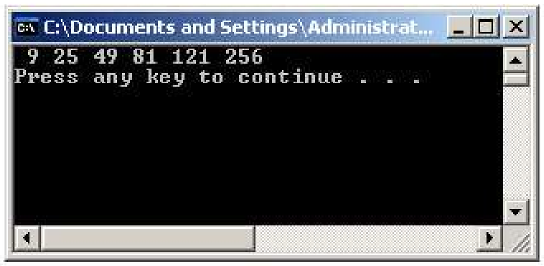
```
//*************************************************
// Whole array passing
//*************************************************
#include <iostream>
using namespace std;
void print_square(int value[]);

int main()
{
  int value[]={3,5,7,9,11,16};

  print_square(value);// in function call to pass whole array, array name is mentioned

  cout<<endl;
  system("pause");
  return 0;
}

void print_square(int value[])
{
    for (int i = 0; i < 6 ; i++)
    cout<<" "<<value[i]*value[i];
}
```

```
 9 25 49 81 121 256
Press any key to continue . . .
```

Example:size



```
//**************************************************
// Whole array passing with size
//**************************************************
#include <iostream>
using namespace std;
void print_square(int value[], int size);
int main()
{
  int value[]={3,5,7,9,11,16};

  int size = sizeof(value)/sizeof(int);

  print_square(value,size);// in function call to pass whole array, array name is mentioned

  cout<<endl;
  system("pause");
  return 0;
}
void print_square(int value[],int size)
{
    for (int i = 0; i < size ; i++)
    cout<<" "<<value[i]*value[i];
}
```

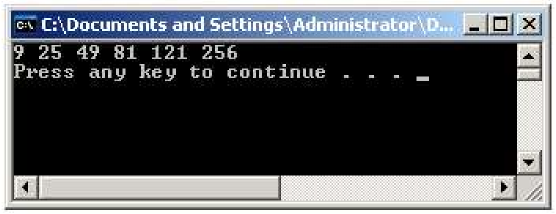


C:\Documents and Settings\Administrat...

```
 9 25 49 81 121 256
Press any key to continue . . .
```

Example: original value

```
//*************************************************
// To demostrate Whole array passing (as pass by refrence)
//*************************************************
#include <iostream>
using namespace std;
void square(int value[], int size);
int main()
{
  int value[]={3,5,7,9,11,16};

  int size = sizeof(value)/sizeof(int);

  square(value,size);// in function call to pass whole array, array name is mentioned

  for(int i =0; i<size;i++)
  cout<<value[i]<<" ";

  cout<<endl;
  system("pause");
  return 0;
}
void square(int value[],int size)
{
    for (int i = 0; i < size ; i++)
    value[i]= value[i]*value[i];
}
```


C:\Documents and Settings\Administrator\D...

```
9 25 49 81 121 256
Press any key to continue . . . _
```

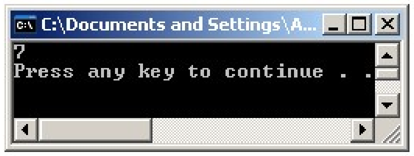
IIT ROORKEE

Passing Arrays as Constants

- Array is not changed by the called function
- Integrity is ensured by passing it as a constant
- Use type modifier **const**, in the function header parameter list

Write a function to return the average of an Array

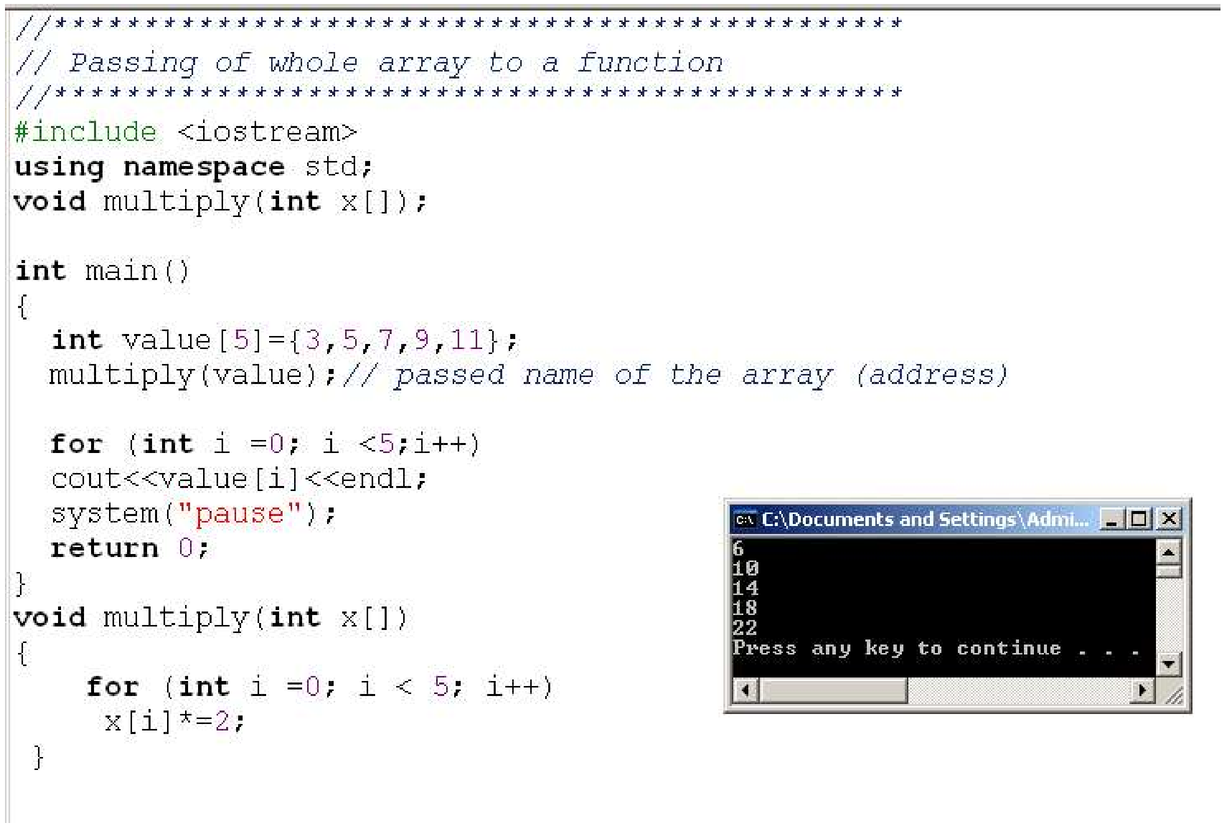
```cpp
#include <iostream>
using namespace std;

double average (const int x[]);

int main()
{
  double ave;

  int value[5]={3,5,7,9,11};

  ave = average(value);// passed name of the array (address)

  cout<<ave<<endl;

  system("pause");
  return 0;
}

double average(const int x[])
{
    double sum =0;
    for (int i =0; i < 5; i++)
    sum += x[i];
    return (sum/5.0);
}
```


C:\Documents and Settings\A...

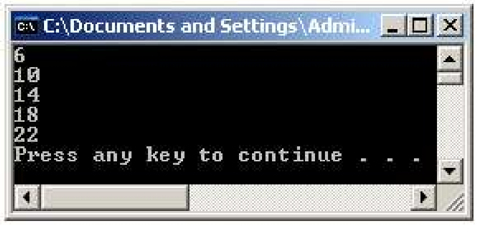
```
7
Press any key to continue . . .
```

Passing Arrays for Updating (without using constant modifier)

- To change the value of the array elements by passing the array name(by reference)



```cpp
//**************************************************
// Passing of whole array to a function
//**************************************************
#include <iostream>
using namespace std;
void multiply(int x[]);

int main()
{
  int value[5]={3,5,7,9,11};
  multiply(value);// passed name of the array (address)

  for (int i =0; i <5;i++)
  cout<<value[i]<<endl;
  system("pause");
  return 0;
}
void multiply(int x[])
{
    for (int i =0; i < 5; i++)
     x[i]*=2;
 }
```

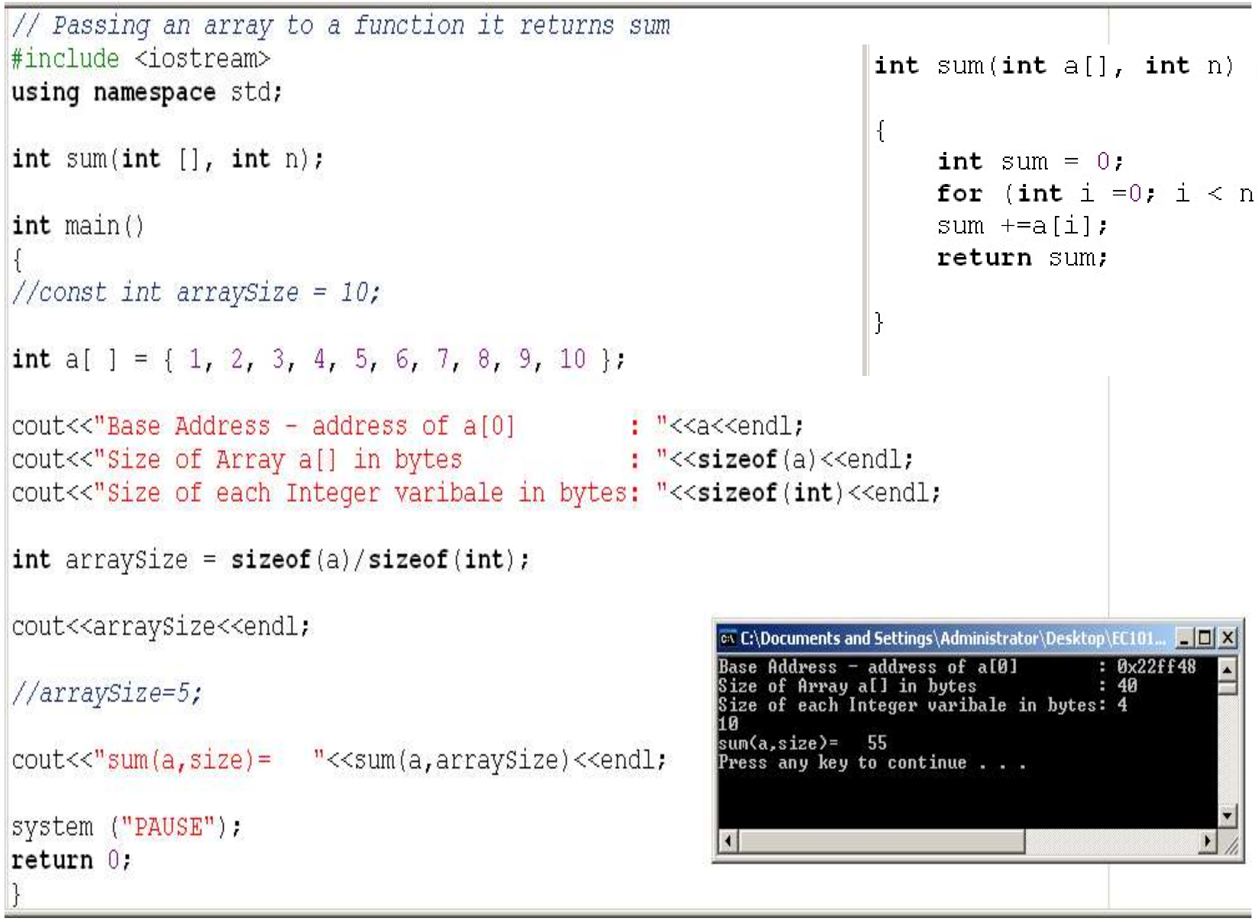


C:\Documents and Settings\Admi...

```
6
10
14
18
22
Press any key to continue . . .
```

Passing Arrays to Functions

- When a function is passed an array this way, it is actually passed only the address of the memory cell where the array starts. This value is represented by the array's name **value**.
- The function can then change the contents of the array by directly accessing the memory cells where the array's elements are stored.
- Although the name of the array is passed by value, its elements can be changed just as if they had been passed by reference.



```cpp
// Passing an array to a function it returns sum
#include <iostream>
using namespace std;

int sum(int [], int n);

int main()
{
//const int arraySize = 10;

int a[ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

cout<<"Base Address - address of a[0]        : "<<a<<endl;
cout<<"Size of Array a[] in bytes            : "<<sizeof(a)<<endl;
cout<<"Size of each Integer varibale in bytes: "<<sizeof(int)<<endl;

int arraySize = sizeof(a)/sizeof(int);

cout<<arraySize<<endl;

//arraySize=5;

cout<<"sum(a,size)=   "<<sum(a,arraySize)<<endl;

system ("PAUSE");
return 0;
}
```

```cpp
int sum(int a[], int n) // function where array and its size is passed

{
    int sum = 0;
    for (int i =0; i < n; i++)
    sum +=a[i];
    return sum;

}
```

C:\Documents and Settings\Administrator\Desktop\EC101...

```
Base Address - address of a[0]        : 0x22ff48
Size of Array a[] in bytes            : 40
Size of each Integer varibale in bytes: 4
10
sum(a,size)=   55
Press any key to continue . . .
```

```
//*************************************************
// This program reverses the numbers in an array
//*************************************************
#include <iostream>
#include<string>
using namespace std;

int main()
{
  int value[10];
  int number;

  for (number = 0; number < 10; number++) // for loop for cin - enter 10 integers
  cin >> value[number];

   for (number = 9; number >= 0; number--) // for loop for reversing the stored intergers in array named value
  cout << value[number] << endl;

 system("pause");
  return 0;
}
```

Summary

- Specify name without brackets
  - To pass array **`myArray`** to **`myFunction`**

```
int myArray[ 25 ];
myFunction( myArray, 25 );
```

  - Array size usually passed, but not required
    - Useful to iterate over all elements

Summary

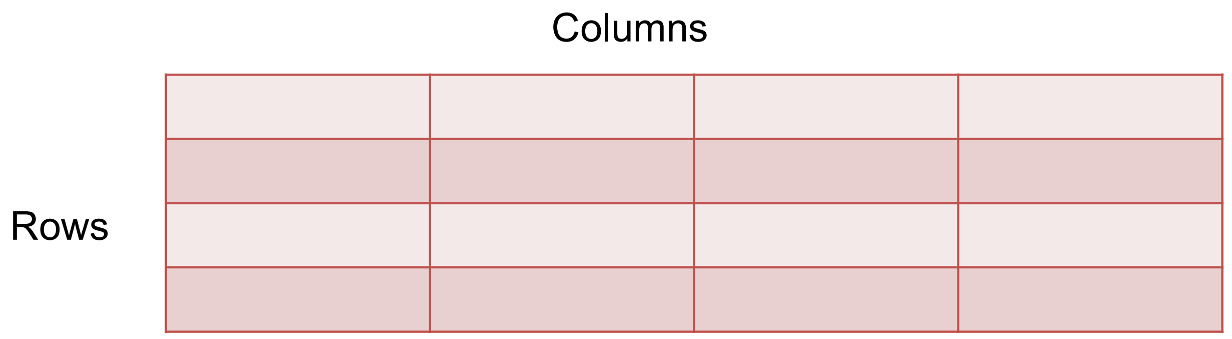
- Arrays passed-by-reference
  - Functions can modify original array data
  - Value of name of array is address of first element
    - Function knows where the array is stored
    - Can change original memory locations
- Individual array elements passed-by-value
  - Like regular variables
  - `square( myArray[5] );`

Summary

- Functions taking arrays
  - Function prototype
    - `void modifyArray( int b[], int arraySize );`
    - `void modifyArray( int [], int );`
      - Names optional in prototype
    - Both take an integer array and a single integer
  - No need for array size between brackets
    - Ignored by compiler
  - If declare array parameter as **`const`**
    - Array elements cannot be modified (compiler error)
    - `void doNotModify( const int [] );`

Two-dimensional Arrays

- In C++, a two-dimensional array is a grouping of elements arranged in rows and columns. Each element is accessed using two indices: one for the row and one for the column, which makes it easy to visualize as a table or grid.
- A common example:
  - Table – 2D array consists of rows and columns
  - Rows – First Dimension
  - Columns – Second Dimension



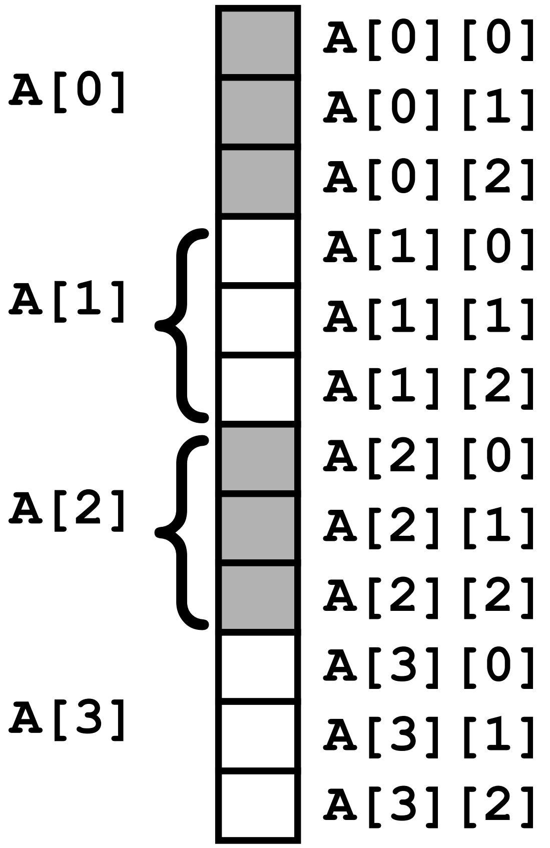
- 2D array is an array of one-dimensional arrays

2-D Memory Organization

```
char A[4][3];
```

A is an array of size 4.

Each element of A is an array of 3 chars



Declaring and Defining 2D Arrays

- Like one-dimensional arrays, 2D arrays must be declared and defined before being used
- Declaration and definition –
  - type of each element
  - name of the array
  - size of each dimension
- A two-dimensional array has two subscripts/indexes.
  - The first subscript refers to the row
  - The second subscript refers   to the column

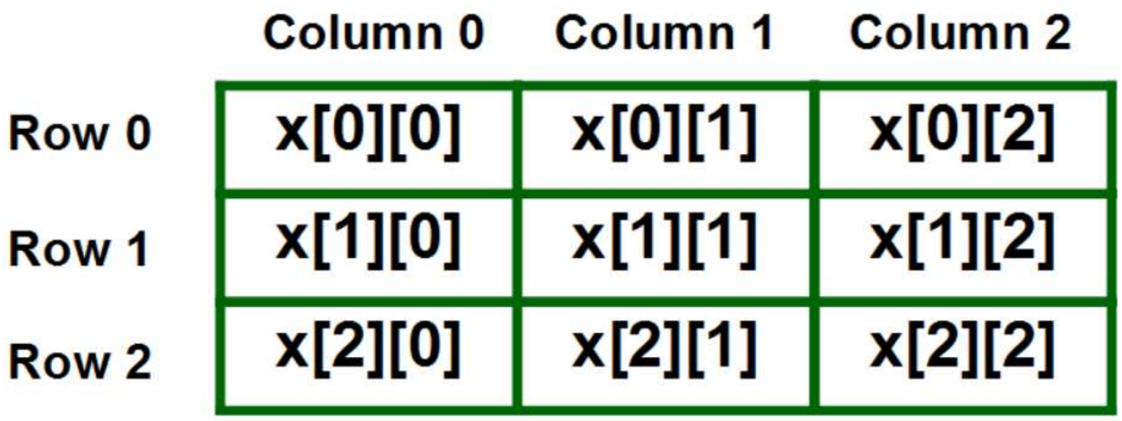
- Its declaration has the following form:
  - **data_type   array_name[number of rows][number of columns];**

- For example:

  int x[3][3]; //declares x as an integer array with 3 rows and 3 columns

  float matrix[20][25]; // declares matrix as a floating-point array with 20 rows and 25 columns

- int x[3][3] can be depicted as follows:



	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]

Initializing 2D array

```
int movie_rating [3][4]
{
      { 0, 4 ,3, 5},
      { 2, 3, 3, 5},
      { 1, 4, 4, 5}
};
```

	0	1	2	3
0	0	4	3	5
1	2	3	3	5
2	1	4	4	5

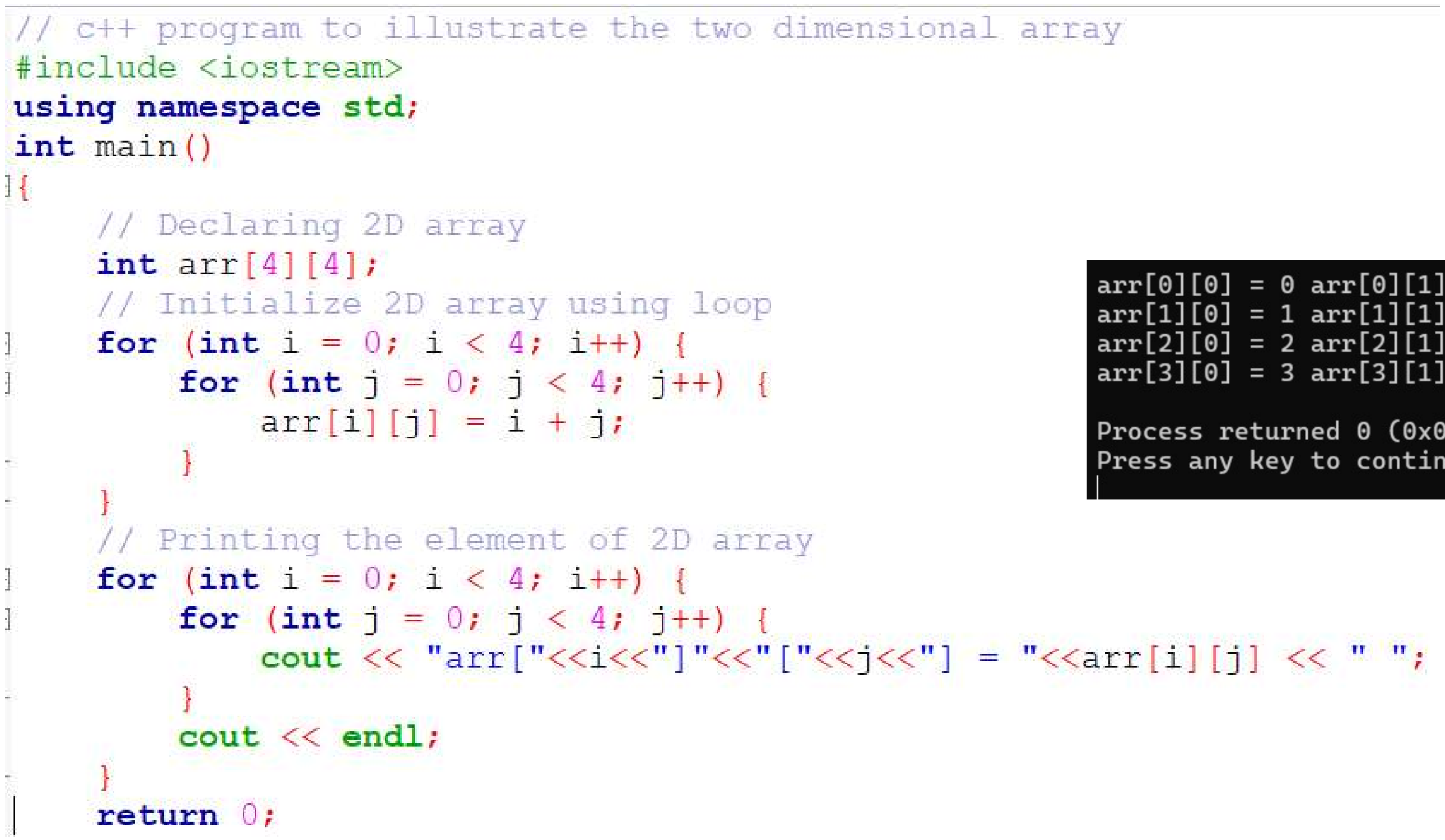
Int x [3] [4] = {{1,2,3}, {5,6,7}{9,10,11}}

x [0] [0]   x [1] [0]   x [2] [0]

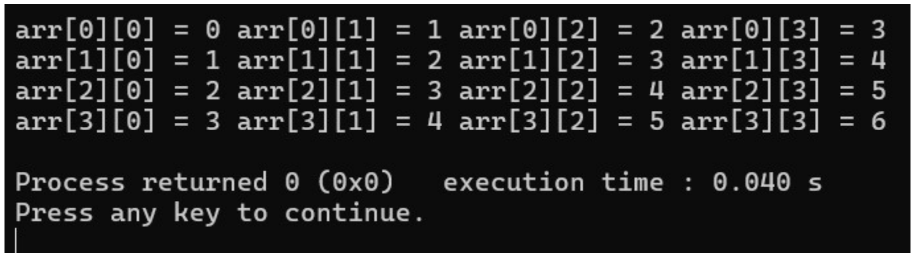
In this declaration an initial value of zero will be assigned to

x[0][3], x[1][3] and x[2][3].

Example



```cpp
// c++ program to illustrate the two dimensional array
#include <iostream>
using namespace std;
int main()
{
    // Declaring 2D array
    int arr[4][4];
    // Initialize 2D array using loop
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            arr[i][j] = i + j;
        }
    }
    // Printing the element of 2D array
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cout << "arr["<<i<<"]"<<"["<<j<<"] = "<<arr[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
```



```
arr[0][0] = 0 arr[0][1] = 1 arr[0][2] = 2 arr[0][3] = 3
arr[1][0] = 1 arr[1][1] = 2 arr[1][2] = 3 arr[1][3] = 4
arr[2][0] = 2 arr[2][1] = 3 arr[2][2] = 4 arr[2][3] = 5
arr[3][0] = 3 arr[3][1] = 4 arr[3][2] = 5 arr[3][3] = 6

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

- For n rows and m columns, the total size equal to m*n
- The item list is read starting from the first row from left to right and then goes to the next row, and so on.

A set of strings can be stored in a two-dimensional character array with **the left index specifying the number of strings** and the **right index specifying the maximum length of each string**.

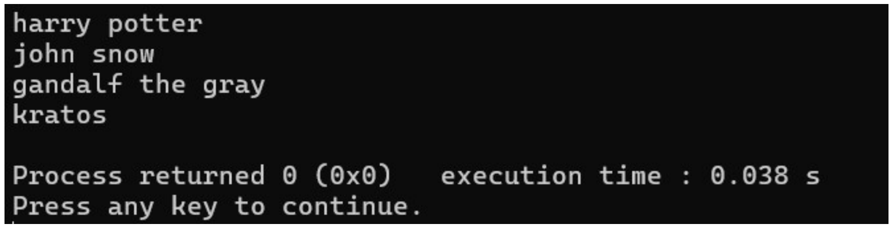
Note: A 2D array that can store 4 names, each 20 characters long(including null character) is defined as

```
char    name[4][20];
```

Where, name[0] is first name, name[1] is second name and so on.

Example

```cpp
#include <iostream>

using namespace std;

int main()  {

    char name  [4][20]{"harry potter","john snow","gandalf the gray","kratos"};
    for  (int i=0;i<4;i++){
     cout<<name[i]<<endl;
    }
    return  0;
}
```



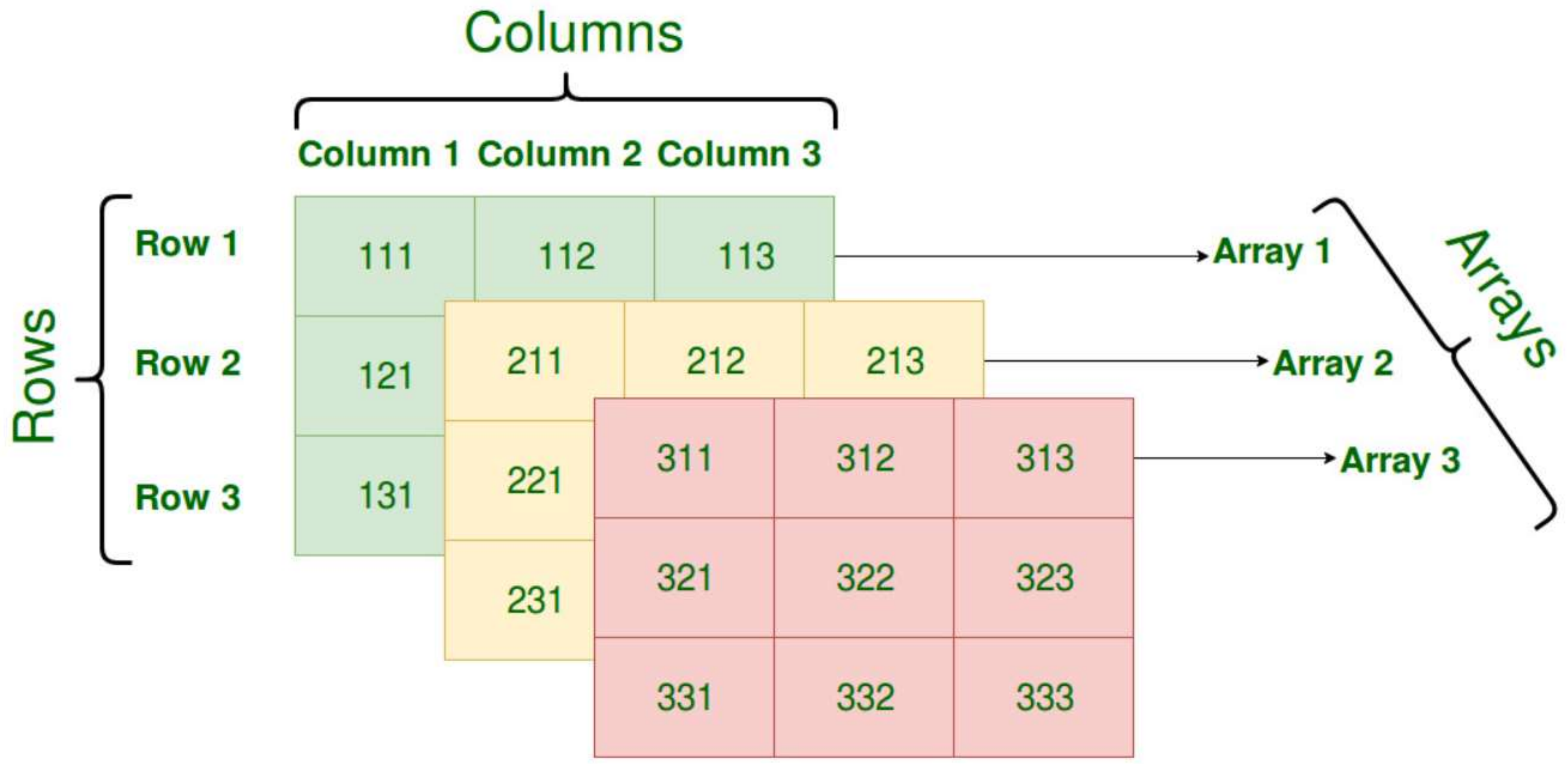
```
harry potter
john snow
gandalf the gray
kratos

Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

3D Array

- The 3D array uses three dimensions. A collection of various two-dimensional arrays piled on top of one another can be used to represent it. Three indices—the row index, column index, and depth index are used to uniquely identify each element in a 3D array.
- To declare a 3D array in C++, we need to specify its third dimension along with 2D dimensions.
- Data_Type Array_Name[D][R][C];

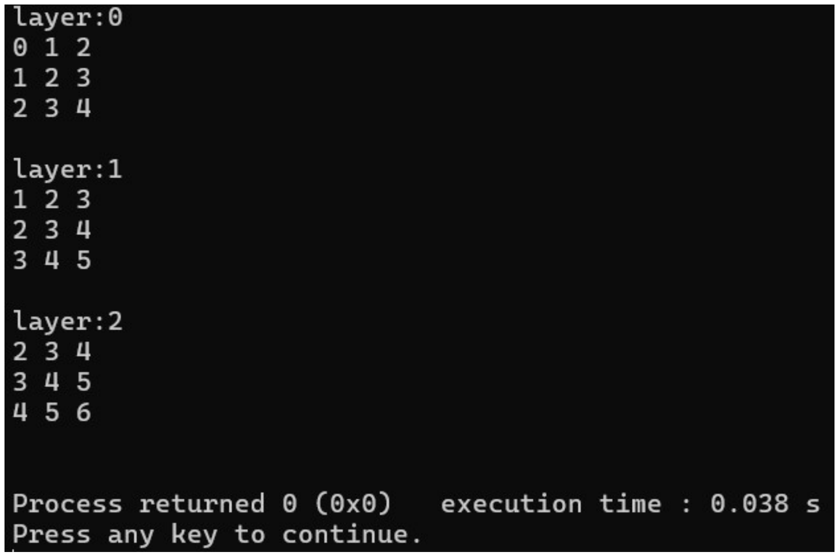
- Data_Type: Type of data to be stored in each element.
- Array_Name: Name of the array
- D: Number of 2D arrays or Depth of array.
- R: Number of rows in each 2D array.
- C: Number of columns in each 2D array.

- int array[3][3][3];



Example

```cpp
#include <iostream>
using namespace std;
int main()
{
    // declaring 3d array
    int arr[3][3][3];
    // initializing the array
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                arr[i][j][k] = i + j + k;
            }
        }
    }
    // printing the array
    for (int i = 0; i < 3; i++) {
        cout << "layer:" <<i << endl;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                cout << arr[i][j][k] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }
    return 0;
}
```



```
layer:0
0 1 2
1 2 3
2 3 4

layer:1
1 2 3
2 3 4
3 4 5

layer:2
2 3 4
3 4 5
4 5 6


Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

Thanks