Arrays

An **array** is a consecutive group of memory locations that all have the same name and the same type.

C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

We need to understand the following about functions:

- Declaring Arrays
- Initializing Arrays
- Accessing Array Elements (Read / Write / Process Array Elements)
- Array dimension

1. Declaring Arrays

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows –

```
type arrayName [ arraySize ];
```

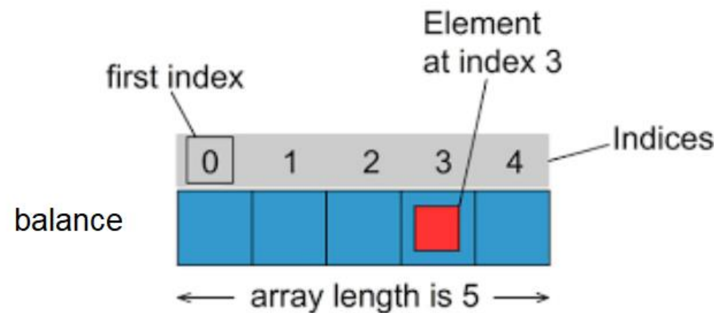
This is called a **single-dimension array**, in which

- **type** can be any valid C++ data type.
- **arrayName** is a valid identifier.
- The **arraySize** must be an integer constant greater than zero.

For example:

```
double balance[5];     // Declare an double array of 5 elements
int scores[10];         // Declare an int array called scores with 10 elements
char alphabet[5];
```

NOTE: The elements field within square brackets [ ], representing the number of elements in the array, must be a constant expression, since arrays are blocks of static memory whose size must be determined at compile time, before the program runs.



In the above example, we have defined an array of char of size five:

char alphabet[5];

The array allocates in the computer 6 bytes for the array, one for each character:

alphabet

Array Index	Contents
alphabet[0]	
alphabet[1]	
alphabet[2]	
alphabet[3]	
alphabet[4]	

2. Initializing Arrays

Initially the contents of the array variables are unpredictable values, just like other uninitialized variables. In C++, the elements in an array can be explicitly initialized to specific values when it is declared, by enclosing those initial values in braces {}.

Typically there are 2 ways to initialize your array:

a) Declare and initialize array by setting a fixed size

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
int score [4] = {4 , 3 , 4 , 5}
char alphabet [5] = {‘A’ , ‘B’ , ‘C’ , ‘E’ , ‘H’}
```

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].

Now, according to the above initialized array, **alphabet will be:**

alphabet

Array Index	Contents
alphabet[0]	A
alphabet[1]	B
alphabet[2]	C
alphabet[3]	E
alphabet[4]	H

b) Declare array by initializing your elements

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

3. Accessing Array Elements (Read/Write/Process Array Elements)

The values of any of the elements in an array can be accessed just like the value of a regular variable of the same type. The syntax is:

arrayName[index]

This mean each array index is treated like a separate integer variable.

Here are some simple ways we can access the array just like it was a variable:

int a[5];

a[3] = 54; // Stores 54 into array element 3

```
a[0] = a[3];                // Copies 54 into array element 0

a[5] = a[2+1];              // Copies contents of a[3] to a[5]

i=5;
a[i]++;                     // Increment value in a[5]

for (i=0;i< 5; i++)
      cout << a[i] << endl;       // Print each array element
```

The flexible thing here is we can programmatically access each variable, based on the index, instead of hard-coding the access by hand.

Ex1: Write a C++ program to assign value to array and print out the array elements.

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

int main () {

   int a[ 10 ];        // a is an array of 10 integers

   // initialize elements of array n to 0
   for ( int i = 0; i < 10; i++ ) {
      a[ i ] = i + 100;        // set element at location i to i + 100
   }

   // output each array element's value
   for ( int j = 0; j < 10; j++ ) {
      cout << j << " "<< a[ j ] << endl;
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
0  100
1  101
2  102
3  103
4  104
5  105
6  106
7  107
8  108
9  109
```

Ex2: Write a C++ program to enter value to array and find the summation of the array elements.

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

int main () {

   int a[ 10 ];         // a is an array of 10 integers
   int i, j, sum =0;

   // initialize elements of array n to 0
   for ( i = 0; i < 10; i++ ) {
      cin >> a[ i ];   // enter element at location i
   }

   // output each array element's value
   for ( j = 0; j < 10; j++ ) {
      sum+= a[ j ];
   }

   cout << " The sum of array elements is: "<< sum << endl;

   return 0;
}
```

Two-dimension Array

So far, all the previous examples explain the one-dimension array (also called vector). However, arrays can be of two dimensions (i.e., subscripts) to represent tables of values consisting of information arranged in rows and columns. The intersection between a row and a column represents the location of arrays' elements. Arrays that require two subscripts to identify a particular element are called two-dimensional arrays or 2-D arrays.

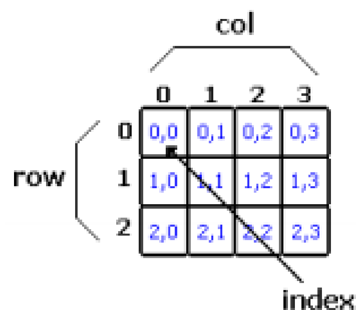


Figure above illustrates a two-dimensional array. The array contains three rows and four columns, so it’s said to be a 3-by-4 array. In general, an array with m rows and n columns is called an m-by-n array.

To access the element at the second row and third column (if the array name is a), then we can access it as a[1][2]. That is due to the first row’s index is 0 and the first column’s index is 0 also.

Note: Sometimes, an array may contain equal number of rows and columns, it is then called a matrix and expressed as n-by-n array. The main diagonal represents the elements from [0][0] to [n-1][n-1], while the secondary diagonal represents the elements from [0][n-1] to [n-1][0].

1. Declaring 2D-Arrays

Type arrayName[rows][columns];

Example:

int marks[8][7] //marks is a 2D array of 8 rows and 7 columns of int values

2. Initializing 2D-Array Elements:

Int a [2] [3] = { {1, 2, 3} , {4, 5, 6} };

1	2	3
4	5	6

3. Read / Write / Process Array Elements

Ex3: Write a C++ program to Declaring a 2D array and Using a Loop to Initialize and print the array's elements

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

int main () {

   int a[4][3];        // a is an array of 4 rows and 3 column integers

   // read elements to array a
   for ( int i = 0; i < 8; i++ ) {
        for ( int j = 0; j < 7; j++ )
              cin >> a[i][j];  // enter element at location i to 0
        }

   // output each array element's value
   for ( int j = 0; j < 8; j++ ) {
        for ( int j = 0; j < 7; j++ )
              cout << j << " "<< a[ j ] << endl;
      }
   return 0;
}
```

Ex4: Write C++ program, to read 3*4 2D-array, then find the summation of each row:

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

int main () {

   int a[3][4];   // a is an array of 4 rows and 3 column integers
   int sum;
   // read elements to array a
   for ( int i = 0; i < 4; i++ )
      for ( int j = 0; j < 3; j++ )
            cin >> a[i][j];

   for ( int j = 0; j < 4; j++ ) {
      sum = 0;
      for ( int j = 0; j < 3; j++ )
            sum+= a[i][j];
      cout << "summation of row " << i << " is: " << sum << endl;
    }
   return 0;
}
```

Multidimensional Arrays

You have seen one-dimensional and two-dimensional arrays. In C++, arrays may have any number of dimensions. To process every item in a one-dimensional array, you need one loop. To process every item in a two-dimensional array, you need two loops. The pattern continues to any number of dimensions. To process every item in an ndimensional array, you need n loops.

For example, if we wanted to declare an array of 3 dimensions, each with 10 elements, we could do so via;

```
int three_d_array[10][10][10];
```