

## ARRAYS IN C++ 1-D and 2-D Array UNIT – III

# Arrays

- ◆ 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.
- ◆ 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.

# 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:

- ▮ **type arrayName [ arraySize ];**
- ▮ This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type.
- ▮ **double balance[10];**
- ▮ **int test[85];**

# Initializing Arrays

✂ In C++ array elements either one by one or using a single statement as follows:

- ▮ **double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};**
- ▮ The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].
- ▮ If size of the array is omitted, an array just big enough to hold the initialization is created.
- ▮ **double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};**
- ▮ **all arrays have 0 as the index of their first element which is also called base index.**

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

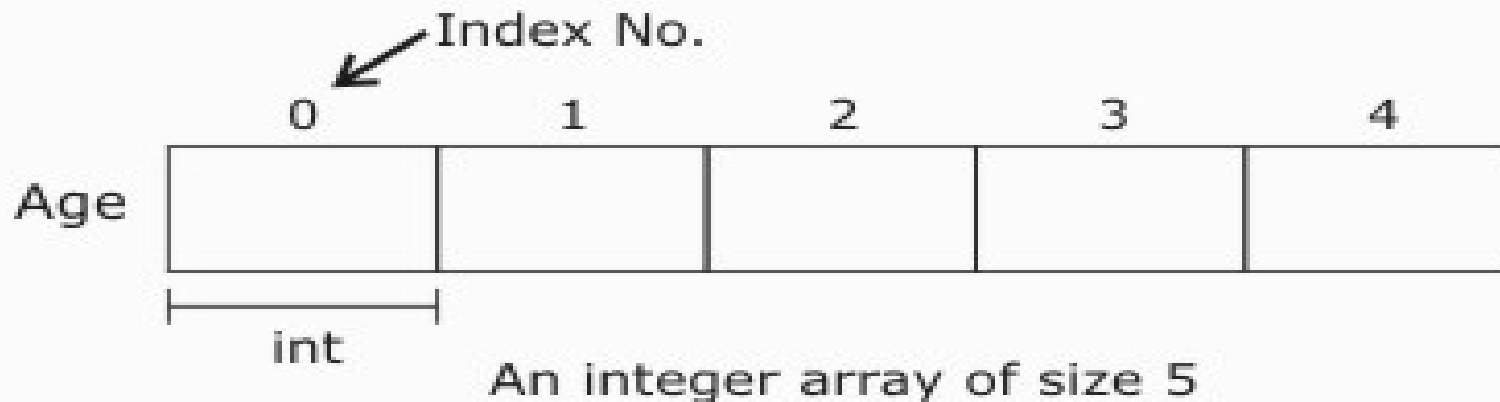
# Accessing Array Elements

✂ An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

- ▮ **double salary = balance[9];**
- ▮ **The syntax is: name[index]**
- ▮ **salary[2] = 15000;**
- ▮ **Here 3rd element of the array contains value 15000.**

# Single / One Dimensional Arrays

- ✂ Single / One Dimensional Array is an array having a single index value to represent the arrays element.
- ✂ Syntax: `type array_name[array_size]`
- ✂ Declaration of Array
  - ▢ `Type arrayName[numberOfElements];`
  - ▢ **For example, `int Age[5] ; float cost[30];`**



# Two / Multi-dimensional Arrays

✂ C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

- `type name[size1][size2]...[sizeN];`

- `int threedim[5][10][4];`

- **Two-Dimensional Arrays:** simplest form of the multidimensional array is the two-dimensional array. To declare a two-dimensional integer array of size x,y, you would write something as follows:

✂ `type arrayName [ x ][ y ];`

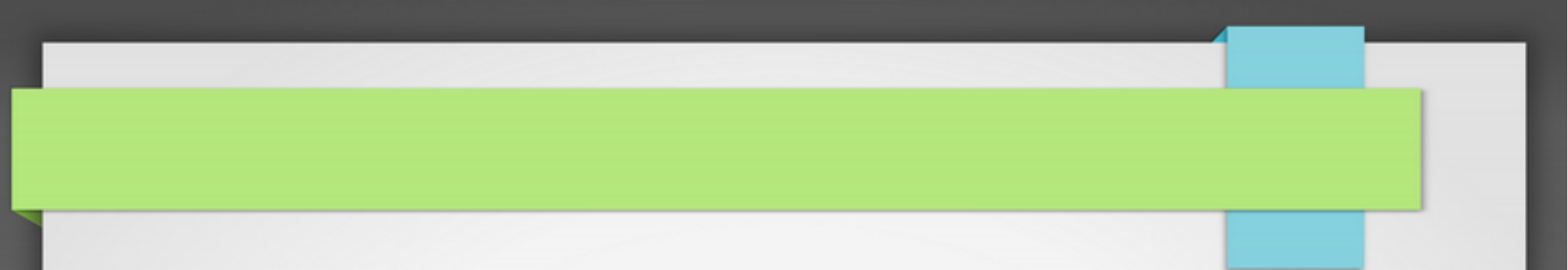
	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[ 0 ][ 0 ]</code>	<code>a[ 0 ][ 1 ]</code>	<code>a[ 0 ][ 2 ]</code>	<code>a[ 0 ][ 3 ]</code>
Row 1	<code>a[ 1 ][ 0 ]</code>	<code>a[ 1 ][ 1 ]</code>	<code>a[ 1 ][ 2 ]</code>	<code>a[ 1 ][ 3 ]</code>
Row 2	<code>a[ 2 ][ 0 ]</code>	<code>a[ 2 ][ 1 ]</code>	<code>a[ 2 ][ 2 ]</code>	<code>a[ 2 ][ 3 ]</code>

## ✂ Initializing Two-Dimensional Arrays:

- ✂ Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {  
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */  
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */  
    {8, 9, 10, 11} /* initializers for row indexed by 2 */  
};
```





1-D Array operations-

2-D Array operations-

# Arrays and Functions

✂ If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received.

✂ Formal parameters as a sized array as follows:

```
void myFunction(int param[10])
{
    ...
    ...
    ...
}
```

# Passing 1-D Array to a Function

In C++, we can pass arrays as an argument to a function.  
And, also we can return arrays from a function.

The syntax for passing an array to a function is:

```
returnType functionName(dataType arrayName[arraySize])  
{  
    // code  
}
```

Example:

```
int total(int marks[5]) {  
    // code  
}
```

// C++ Program to display marks of 5 students

```
#include <iostream>
```

```
using namespace std;
```

```
// declare function to display marks
```

```
// take a 1d array as parameter
```

```
void display(int m[5]) {
```

```
    cout << "Displaying marks: " << endl;
```

```
// display array elements
```

```
    for (int i = 0; i < 5; ++i) {
```

```
        cout << "Student " << i + 1 << ": " << m[i] << endl;
```

```
    }
```

```
}
```

```
int main() {
```

```
    // declare and initialize an array
```

```
    int marks[5] = {88, 76, 90, 61, 69};
```

```
    // call display function
```

```
    // pass array as argument
```

```
    display(marks);
```

```
    return 0;
```

```
}
```

# Passing Multidimensional Array to a Function

// C++ Program to display the elements of two  
// dimensional array by passing it to a function

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

// define a function
// pass a 2d array as a parameter
void display(int n[][2]) {
    cout << "Displaying Values: " << endl;
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 2; ++j) {
            cout << "num[" << i << "][" << j << "]: " << n[i][j] << endl;
        }
    }
}

int main() {

    // initialize 2d array
    int num[3][2] = {
        {3, 4},
        {9, 5},
        {7, 1}
    };

    // call the function
    // pass a 2d array as an argument
    display(num);

    return 0;
}
```