

C++ Programming Language

Presented By:

T.A. Asmaa Hamad El-saied

E-mail: eng.asmaa134@gmail.com

Contents



Arrays

Array

- ❖ An **Array** is a powerful data structure that stores a collection of data, having the *same* data 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.

Array(cont.)

❖ Declaring Arrays

- *<Data 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.

Example

```
int myArray [5];
```

0	
1	
2	
3	
4	

myArray

Array(cont.)

❖ Initializing Arrays

- You can initialize C++ array elements either one by one or using a single statement as follows:

`<arrayName>[arraySize] = <relevant data>`

- `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 []

Array(cont.)

❖ Initializing Arrays (cont.)

- If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

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

You will create exactly the same array as you did in the previous example.

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

`balance[4] = 50.0;`

- The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index.

Array(cont.)

❖ 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 above statement will take 10th element from the array and assign the value to salary variable.

Array(cont.)

- ❖ Just like ordinary variables, arrays should be initialised, otherwise scrap data will remain stored in them
 - If we want to initialize 2 whole 20-sized integer and boolean arrays to 0 and false respectively, we do it like this:

```
    int myIntArray [20];  
    bool myBoolArray [20];  
For (int i = 0;i<20; i++)  
{  
    myIntArray[i] = 0;  
    myBoolArray[i] = false;  
}
```


Array(cont.)

```
#include <iostream>
using namespace std;
int main ()
{
    int a[5];
    a[0] = 12;
    a[1] = 23;
    a[2] = 34;
    a[3] = 45;
    a[4] = 56;
    return 0;
}
```

0	12
1	23
2	34
3	45
4	56

Array(cont.)

- ❖ It is a lot easier when you use a loop to access the values in an array. Here is an example of reading in 5 values into an array:

```
#include <iostream>
using namespace std;
int main ()
{
    int a[5];
    for(int i = 0;i<5;i++)
        cin>>a[i];
    for(int i = 0;i<5;i++)
        cout<<a[i];
    return 0;}
```

Array(cont.)

❖ Example:

```
#include <iostream>
using namespace std;
int main ()
{
    int n[ 10 ]; // n is an array of 10 integers
    for ( int i = 0; i < 10; i++ )
        { n[ i ] = i + 100; // set element at location i to i + 100 }
    cout << "Index \t\t" << "\tValue" << endl;
    // output each array element's value
    for ( int j = 0; j < 10; j++ )
        { cout << j << "\t\t" << n[ j ] << endl; }
    return 0; }
```

Array(cont.)

❖ Example:

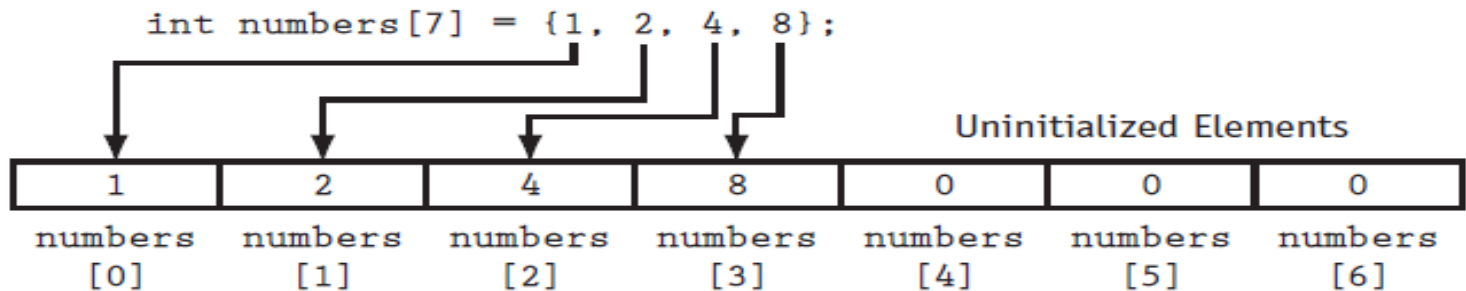
Output

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

Array(cont.)

❖ Partial Array Initialization

- An initialization list cannot have more values than the array has elements, but it may have fewer values than there are elements. That is, C++ does not require a value for every element. It's possible
- to only initialize part of an array, such as
 - `int numbers[7] = {1, 2, 4, 8};`
- This definition only initializes the first four elements of a seven element array



Array(cont.)

❖ Partial Array Initialization(cont.)

- If an array is partially initialized, the uninitialized elements will be set to zero for numeric arrays or to the null character for character arrays. This is true even if the array is defined locally.
- (If a local array is completely uninitialized, its elements will contain “garbage,” like other local variables.)

Array(cont.)

❖ Partial Array Initialization(cont.)

■ Example

```
#include <iostream>
using namespace std;
int main ()
{   const int SIZE = 7;
    int numbers[SIZE] = { 1, 2, 4, 8 }; // Initialize the first 4 elements.
    cout << "Here are the contents of the array:\n";
    for (int index = 0; index < SIZE; index++)
        cout << numbers[index] << " ";
    cout << endl;   return 0; }
```

Program Output Here are the contents of the array:

1 2 4 8 0 0 0

Array(cont.)

❖ Processing Array Contents

- Processing array elements is no different than processing other variables. For example, the following statement multiplies `hours[3]` by the variable `rate`:

```
» pay = hours[3] * rate;
```

- And the following are examples of pre-increment and post-increment operations on array elements:
 - `int score[5] = {7, 8, 9, 10, 11};`
 - `++score[2];` // Pre-increment operation on the value in `score[2]`
 - `score[4]++;` // Post-increment operation on the value in `score[4]`

Array(cont.)

❖ Processing Array Contents(cont.)

- **Note:**
- When using increment and decrement operators, be careful not to confuse the subscript with the array element. The following example illustrates the difference.
- `amount[count]--;` // This decrements the value stored in `amount[count]`.
- `amount[count--];` // This decrements the variable `count`, but does // nothing to the value stored in `amount[count]`..

Array(cont.)

❖ Summing the Values in a Numeric Array

```
const int NUM_UNITS = 6;  
int units[NUM_UNITS] = { 16, 20, 14, 8, 6, 10};  
int total = 0; // Initialize accumulator  
for (int count = 0; count < NUM_UNITS; count++)  
    total += units[count];
```

Array(cont.)

❖ Finding the Average of the Values in a Numeric Array

```
const int NUM_SCORES = 5;
double scores[NUM_SCORES] = {90, 88, 91, 55, 33};
double total = 0; // Initialize accumulator
double average; // Will hold the average
for (int count = 0; count < NUM_SCORES; count++)
    total += scores[count];
average = total / NUM_SCORES;
```

Array(cont.)

❖ Finding the Highest and Lowest Values in a Numeric Array

```
int highest;
```

```
highest = numbers[0];
```

```
for (count = 1; count < SIZE; count++)
```

```
{   if (numbers[count] > highest)
    highest = numbers[count]; }
```

```
int lowest;
```

```
lowest = numbers[0];
```

```
for (count = 1; count < SIZE; count++)
```

```
{   if (numbers[count] < lowest)
    lowest = numbers[count]; }
```

Two-Dimensional Arrays

Two-Dimensional Arrays

- An array is useful for storing and working with a set of data. Sometimes, though, it's necessary to work with multiple sets of data. For example, in a grade-averaging program a teacher might record all of one student's test scores in an array of doubles. If the teacher has 30 students, that means 30 arrays of doubles will be needed to record the scores for the entire class. Instead of defining 30 individual arrays, however, it would be better to define a two-dimensional (2D) array.

Two-Dimensional Arrays(cont.)

- Two-dimensional arrays can hold multiple sets of values. It's best to think of a two dimensional array as a table having rows and columns of elements

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

- shows an array of test scores that has three rows and four columns. Notice that the three rows are numbered 0 through 2 and the four columns are numbered 0 through 3. There are a total of 12 elements in the array

Two-Dimensional Arrays(cont.)

- To define a two-dimensional array, two size declarators are required: the first one is for the number of rows and the second one is for the number of columns.
- Here is an example definition of a two-dimensional array with three rows and four columns:

– `double score[3][4];`

 ↓ ↓
 rows columns

- The elements in row 0 are

- » `score[0][0]`
- » `score[0][1]`
- » `score[0][2]`
- » `score[0][3]`

Two-Dimensional Arrays(cont.)

- The elements in row 1 are
 - » `score[1][0]`
 - » `score[1][1]`
 - » `score[1][2]`
 - » `score[1][3]`
- And the elements in row 2 are
 - » `score[2][0]`
 - » `score[2][1]`
 - » `score[2][2]`
 - » `score[2][3]`

Two-Dimensional Arrays(cont.)

- ❖ the following statement assigns the value 92.25 to the element at row 2, column 1 of the score array:
 - `score[2][1] = 92.25;`
- ❖ And the following statement displays the element at row 0, column 2:
 - `cout << score[0][2];`

Two-Dimensional Arrays(cont.)

❖ Initializing Two-Dimensional Arrays:

- Two-Dimensional Arrays 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},
 {4, 5, 6, 7} ,
 {8, 9, 10, 11} };`

Or

- `int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};`

Two-Dimensional Arrays(cont.)

❖ Accessing Two-Dimensional Array Elements:

- An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.

For example:

```
» int val = a[2][3];
```

- The above statement will take 4th element from the 3rd row of the array

Two-Dimensional Arrays(cont.)

❖ Simple Example:

```
#include <iostream>
using namespace std;
int main ()
{ // an array with 5 rows and 2 columns.
  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
  // output each array element's value
  for ( int i = 0; i < 5; i++ )
    for ( int j = 0; j < 2; j++ )
      { cout << "a[" << i << "][" << j << "]: ";
        cout << a[i][j]<< endl; }
  return 0; }
```

Two-Dimensional Arrays(cont.)

❖ Simple Example:

Output

a[0][0]: 0

a[0][1]: 0

a[1][0]: 1

a[1][1]: 2

a[2][0]: 2

a[2][1]: 4

a[3][0]: 3

a[3][1]: 6

a[4][0]: 4

a[4][1]: 8

Two-Dimensional Arrays(cont.)

❖ Summing All the Elements of a Two-Dimensional Array

```
const int NUM_ROWS = 3; // Number of rows
```

```
const int NUM_COLS = 5; // Number of columns
```

```
int total = 0; // Accumulator
```

```
int numbers[NUM_ROWS][NUM_COLS] = {{2, 7, 9, 6, 4},  
                                     {6, 1, 8, 9, 4},  
                                     {4, 3, 7, 2, 9}};
```

```
// Sum the array elements
```

```
for (int row = 0; row < NUM_ROWS; row++)
```

```
    for (int col = 0; col < NUM_COLS; col++)
```

```
        total += numbers[row][col];
```

```
cout << "The total is " << total << endl;
```

Two-Dimensional Arrays(cont.)

❖ Summing the Rows of a Two-Dimensional Array

```
const int NUM_STUDENTS = 3; // Number of students
```

```
const int NUM_SCORES = 5; // Number of test scores
```

```
double total; // Accumulator
```

```
double average; // Holds a given student's average
```

```
double scores[NUM_STUDENTS][NUM_SCORES]
```

```
= { {88, 97, 79, 86, 94},  
    {86, 91, 78, 79, 84},  
    {82, 73, 77, 82, 89}};
```


Two-Dimensional Arrays(cont.)

❖ Summing the Rows of a Two-Dimensional Array

// Sum each student's test scores so his or her

// average can be calculated and displayed

```
for (int row = 0; row < NUM_STUDENTS; row ++)
```

```
{
```

```
    total = 0;
```

```
    for (int col = 0; col < NUM_SCORES; col++)
```

```
        total += scores[row][col];
```

// Compute and display the average for this student

```
    average = total / NUM_SCORES;
```

```
    cout << "Score average for student " << (row + 1) << " is " << average << endl;
```

```
}
```

Two-Dimensional Arrays(cont.)

❖ Summing the Columns of a Two-Dimensional Array

```
const int NUM_STUDENTS = 3; // Number of students
```

```
const int NUM_SCORES = 5; // Number of test scores
```

```
double total; // Accumulator
```

```
double average; // Holds average score on a given test
```

```
double scores[NUM_STUDENTS][NUM_SCORES] =
```

```
    { {88, 97, 79, 86, 94},
```

```
      {86, 91, 78, 79, 84},
```


```
      {82, 73, 77, 82, 89} };
```

Two-Dimensional Arrays(cont.)

❖ Summing the Columns of a Two-Dimensional Array

```
for (int col = 0; col < NUM_SCORES; col++)  
{ total = 0;  
    for (int row = 0; row < NUM_STUDENTS; row++)  
        total += scores[row][col];  
  
    // Compute and display the class average for this test  
    average = total / NUM_STUDENTS;  
    cout << "Class average for test " <<<< (col + 1) << " is "  
<< average << endl;  
}
```

some Examples

- 
- ❖ Write a C++ program to sort 10 elements of array.
 - ❖ Write a C++ program to search an element in array
If found display index that is elements in it ,if not
display “not found”.



Thanks