



CS118 – Programming Fundamentals

Lecture # 21

Monday, November 11, 2019

FALL 2019

FAST – NUCES, Faisalabad Campus

Zain Iqbal

Objectives

2

In this chapter, you will:

- Learn about arrays
- Explore how to declare and manipulate data into arrays
- Learn about “array index out of bounds”
- Become familiar with the restrictions on array processing
- Discover how to pass an array as a parameter to a function

Introduction

4

- A data type is called simple if variables of that type can store only one value at a time
- A structured data type is one in which each data item is a collection of other data items

Example

5

```
//Program that takes five numbers print their average
//and the numbers again
#include<iostream>
using namespace std;
int main()
{
    int n1, n2, n3, n4, n5;
    double average;
    cout << "Enter five integers : " ;
    cin >> n1 >> n2 >> n3 >> n4 >> n5 ;

    average = (n1 + n2 + n3 + n4 + n5) / 5.0 ;

    cout << "The average of the given numbers = " << average ;
    cout << "\nand the numbers are n1 = " << n1 << " n2 = " << n2
        << " n3 = " << n3 << " n4 = " << n4
        << " n5 = " << n5 << endl ;
    return 0;
}
```

Example

6

- Five variables must be declared because the numbers are to be printed later
- All variables are of type `int`—that is, of the same data type
- The way in which these variables are declared indicates that the variables to store these numbers all have the same name—except the last character, which is a number

Arrays

7

- **Array:** A collection of a fixed number of components where all of the components have the same data type
- In a one-dimensional array, the components are arranged in a list form
- Syntax for declaring a one-dimensional array:

```
dataType arrayName[intExp];
```

- **intExp** evaluates to a positive integer

Arrays

8

Example:

```
int num[6];
```

num[0]	
num[1]	
num[2]	
num[3]	
num[4]	
num[5]	

	num[0]	num[1]	num[2]	num[3]	num[4]	num[5]
num						

	[0]	[1]	[2]	[3]	[4]	[5]
num						

Defining Arrays

9

- When defining arrays, specify

- Name
- Type of array
- Number of elements

arrayType arrayName[numberOfElements];

- Examples:

```
int c[ 10 ];
```

```
float myArray[ 3284 ];
```

- Defining multiple arrays of same type

- Format similar to regular variables
- Example:

```
int b[ 100 ], x[ 27 ];
```


Accessing Array Components

10

- General syntax:

```
arrayName[indexExp]
```

- Where indexExp, called an index, is any expression whose value is a nonnegative integer
- Index value specifies the position of the component in the array
- [] is the **array subscripting operator**
- The array index always starts at 0

Accessing Array Components (cont'd.)

11

```
int list[8];
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list								

```
list[5] = 75;
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list						75		

Accessing Array Components (cont'd.)

`list[3] = 20 ;`

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list				20				

`list[6]=100;`

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list				20			100	

`list[2]= list[3] + list[6];`

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
list			120	20			100	

Accessing Array Components (cont'd.)

Suppose `i` is an `int` variable. Then, the assignment statement:

```
list[3] = 63;
```

is equivalent to the assignment statements:

```
i = 3;  
list[i] = 63;
```

If `i` is 4, then the assignment statement:

```
list[2 * i - 3] = 58;
```

stores 58 in `list[5]` because `2 * i - 3` evaluates to 5. The index expression is evaluated first, giving the position of the component in the array.

Accessing Array Components (cont'd.)

Array can also be declared as

```
const int SIZE_OF_ARRAY = 20;  
int array[SIZE_OF_ARRAY] ;
```

First declare a named constant and then use it to declare an array of this specific size.

When an array is declared its size must be known. You cannot do this:

```
int arr_size;  
cout << "Enter size of array ";  
cin >> arr_size;  
  
int arr[arr_size];
```

Processing One-Dimensional Arrays

15

- Some basic operations performed on a one-dimensional array are:
 - **Initializing**
 - **Inputting** data
 - **Outputting** data stored in an array
 - **Finding** the largest and/or smallest element
- Each operation requires ability to **step through** the elements of the array
 - Easily accomplished by a **loop**

Processing One-Dimensional Arrays

(cont'd.)

16

- Consider the declaration

```
int list[100]; //array of size 100
int i;
```

- Using for loops to access array elements:

```
for (i = 0; i < 100; i++) //Line 1
    //process list[i]      //Line 2
```

- Example:

```
for (i = 0; i < 100; i++) //Line 1
    cin >> list[i];       //Line 2
```

Processing One-Dimensional Arrays

(cont'd.)

```
double scores[10];  
int index;  
double largest, sum, average;
```

Initializing an array

```
for (index = 0 ; index < 10 ; ++index)  
    scores[index] = 0.0 ;
```

Reading data into array

```
for (index = 0 ; index < 10 ; ++index)  
    cin >> scores[index] ;
```

Printing the array

```
for (index = 0 ; index < 10 ; ++index)  
    cout << scores[index] << " " ;
```


Processing One-Dimensional Arrays (cont'd.)

Finding sum and average of an array

```
sum = 0.0;
for (index = 0 ; index < 10 ; ++index)
    sum = sum + scores[index];
average = sum / 10;
```

Largest element in the array

```
maxIndex = 0;
for (index = 1 ; index < 10 ; ++index)
    if (scores[maxIndex] < scores[index])
        maxIndex = index;
largest = scores[maxIndex];
```

Processing One-Dimensional Arrays

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
sales	12.50	8.35	19.60	25.00	14.00	39.43	35.90	98.23	66.65	35.64

FIGURE 9-6 Array sales

index	maxIndex	sales [maxIndex]	sales [index]	sales[maxIndex] < sales[index]
1	0	12.50	8.35	12.50 < 8.35 is false
2	0	12.50	19.60	12.50 < 19.60 is true ; maxIndex = 2
3	2	19.60	25.00	19.60 < 25.00 is true ; maxIndex = 3
4	3	25.00	14.00	25.00 < 14.00 is false
5	3	25.00	39.43	25.00 < 39.43 is true ; maxIndex = 5
6	5	39.43	35.90	39.43 < 35.90 is false
7	5	39.43	98.23	39.43 < 98.23 is true ; maxIndex = 7
8	7	98.23	66.65	98.23 < 66.65 is false
9	7	98.23	35.64	98.23 < 35.64 is false

After the **for** loop executes, maxIndex = 7, giving the index of the largest element in the array sales. Thus, largestSale = sales[maxIndex] = 98.23.

Arrays by an Example ...

20

```
1  // C++ Program
2  // Initializing an array.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     int n[ 10 ]; // n is an array of 10 integers
15
16     // initialize elements of array n to 0
17     for ( int i = 0; i < 10; i++ )
18         n[ i ] = 0; // set element at location i to 0
19
20     cout << "Element" << setw( 13 ) << "Value" << endl;
21
22     // output contents of array n in tabular format
23     for ( int j = 0; j < 10; j++ )
24         cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
25
26     return 0; // indicates successful termination
27
28 }
```

Declare a 10-element array of integers.

Initialize array to 0 using a for loop. Note that the array has elements **n[0]** to **n[9]**.

Output

21

Element	value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Example

22

```
//Program to read five numbers, find their sum, and
//print the numbers in reverse order.

#include <iostream>

using namespace std;

int main()
{
    int item[5]; //Declare an array item of five components
    int sum;
    int counter;

    cout << "Enter five numbers: ";

    sum = 0;

    for (counter = 0; counter < 5; counter++)
    {
        cin >> item[counter];
        sum = sum + item[counter];
    }

    cout << endl;

    cout << "The sum of the numbers is: " << sum << endl;
    cout << "The numbers in reverse order are: ";

    //Print the numbers in reverse order.
    for (counter = 4; counter >= 0; counter--)
        cout << item[counter] << " ";

    cout << endl;

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

Enter five numbers: 12 76 34 52 89

The sum of the numbers is: 263

The numbers in reverse order are: 89 52 34 76 12

Array Index Out of Bounds

23

- If we have the statements:

```
const ARRAY_SIZE = 10;  
double num[ARRAY_SIZE];  
int i;
```
- The component `num[i]` is valid if `i = 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9`
- The index of an array is in bounds if the `index >= 0` and the `index <= ARRAY_SIZE-1`
- Otherwise, we say the index is out of bounds
- **In C++, there is no guard against indices that are out of bounds**

Array Index Out of Bounds

24

A loop such as the following can set the index out of bounds:

```
for (i = 0; i <= 10; i++)  
    list[i] = 0;
```

Here, we assume that `list` is an array of 10 components. When `i` becomes 10, the loop test condition `i <= 10` evaluates to `true` and the body of the loop executes, which results in storing 0 in `list[10]`. Logically, `list[10]` does not exist.

How array element is calculated

25

- Each element is placed at consecutive location
- We can calculate address of any element of array by performing simple arithmetic

$$\text{address_of_index_x} = \text{base_address} + \text{size_of_datatype} * x$$

- Example

```
int array[10] ;
```

Assume this array starts at address 100 in memory. To calculate address of index 5.

$$\text{address_of_index_5} = 100 + 4 * 5 = 120$$

[0]		[1]		[2]		[3]		[4]		[5]		[6]		[7]		[8]		[9]	
100	103	104	107	108	111	112	115	116	119	120	123	124	127	128	131	132	135	136	139

Note: Compiler just knows the address of the first element of the array known as the base address of the array. All other indexes are calculated relative to base address.

How array element is calculated

26

- Can we have -ve indexes in the array
- Like **array[-2]**



Array Initialization During Declaration

- Arrays can be initialized during declaration
- In this case, it is not necessary to specify the size of the array
- Size determined by the number of initial values in the braces
- Example:

```
double sales[5] = {12.25, 32.50, 16.90, 23, 45.68};
```

The values are placed between curly braces and separated by commas—here, `sales[0] = 12.25`, `sales[1] = 32.50`, `sales[2] = 16.90`, `sales[3] = 23.00`, and `sales[4] = 45.68`.

- `double sales[] = {12.25, 32.50, 16.90, 23, 45.68};`

Partial Initialization of Arrays During Declaration

- The statement:

```
int list[10] = {0};
```

declares list to be an array of 10 components and initializes all of them to zero

- The statement:

```
int list[10] = {8, 5, 12};
```

declares list to be an array of 10 components, initializes list[0] to 8, list[1] to 5, list[2] to 12 and all other components are initialized to 0

Partial Initialization of Arrays During Declaration (cont'd.)

➤ The statement:

```
int list[] = {5, 6, 3};
```

declares list to be an array of 3 components and initializes list[0] to 5, list[1] to 6, and list[2] to 3

Partial Initialization of Arrays During Declaration (cont'd.)

30

- `int list[10] = {2, 5, 6, , 8}; //illegal`
- In this initialization, because the fourth element is uninitialized, all elements that follow the fourth element must be left uninitialized

Arrays by an Example

31

```
1 // C++ Program
2 // Initializing an array with a declaration.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     // use initializer list to initialize array n
15     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
16
17     cout << "Element" << setw( 13 ) << "Value" << endl;
18
19     // output contents of array n in tabular format
20     for ( int i = 0; i < 10; i++ )
21         cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
22
23     return 0; // indicates successful termination
24
25 }
```

Note the use of the initializer list.

C:\WINDOWS\system32\cmd.exe

Element	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

Press any key to continue . . .

Some Restrictions on Array Processing

- Consider the following statements:

```
int myList[5] = {0, 4, 8, 12, 16}; //Line 1
```

```
int yourList[5]; //Line 2
```

- C++ does not allow aggregate operations on an array:

```
yourList = myList ; //illegal
```

- Solution:

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

```
    yourList[i] = myList[i] ;
```

Some Restrictions on Array Processing (cont'd.)

33

- The following is illegal too:

```
cin >> yourList ; //illegal
```

- Solution:

```
for(int i=0 ; i<5 ; ++i)  
    cin >> yourList[i] ;
```

- The following statements are legal, but do not give the desired results:

```
cout << yourList ;  
if(myList <= yourList)
```

•

•

•

Examples Using Arrays

34

➤ Array size

- Can be specified with constant variable (const)
`const int size = 20;`
- Constants cannot be changed
- Constants must be initialized when declared
- Also called named constants or read-only variables

Arrays by an Example - III ...

35

```
1 // C++ Program
2 // Initialize array s to the even integers from 2 to 20.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     // constant variable can be used to specify array size
15     const int arraySize = 10;
16
17     int s[ arraySize ]; // array s has 10 elements
18
19     for ( int i = 0; i < arraySize; i++ ) // set the values
20         s[ i ] = 2 + 2 * i;
21
22     cout << "Element" << setw( 13 ) << "Value" << endl;
23
24     // output contents of array s in tabular format
25     for ( int j = 0; j < arraySize; j++ )
26         cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
27
28     return 0; // indicates successful termination
29
30 } // end main
```

Note use of **const** keyword.
Only **const** variables can
specify array sizes.

The program becomes more scalable when we set the array size using a **const** variable. We can change **arraySize**, and all the loops will still work (otherwise, we'd have to update every loop in the program).

Program Output ...

36

Element	value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Arrays by an Example -IV...

37

```
1      // C++ Program
2      // Compute the sum of the elements of the array.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      int main()
9      {
10         const int arraySize = 10;
11
12         int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13
14         int total = 0;
15
16         // sum contents of array a
17         for ( int i = 0; i < arraySize; i++ )
18             total += a[ i ];
19
20         cout << "Total of array element values is " << total << endl;
21
22         return 0; // indicates successful termination
23
24     } // end main
```

Output:
Total of array element values is 55

Arrays by an Example -V...

38

```
1      // C++ Program
2      // Histogram printing program.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      #include <iomanip>
9
10     using std::setw;
11
12     int main()
13     {
14         const int arraySize = 10;
15         int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
16
17         cout << "Element" << setw( 13 ) << "Value"
18              << setw( 17 ) << "Histogram" << endl;
19
20         // for each element of array n, output a bar in histogram
21         for ( int i = 0; i < arraySize; ++i ) {
22             cout << setw( 7 ) << i << setw( 13 )
23                  << n[ i ] << setw( 9 );
24
25             for ( int j = 0; j < n[ i ]; ++j ) // print one bar
26                 cout << '*';
```

Prints asterisks corresponding to size of array element, **n[i]**.

Arrays by an Example -V...

39

```
27
28     cout << endl;  // start next line of output
29
30     } // end outer for structure
31
32     return 0;  // indicates successful termination
33
34 } // end main
```

Program Output ...

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

Arrays by an Example –VI ...

40

```
1 // C++ Program
2 // Roll a six-sided die 6000 times.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>
13 #include <ctime>
14
15 int main()
16 {
17     const int ARRAY_SIZE= 7;
18     int frequency[ARRAY_SIZE ] = { 0 };
19
20     srand( time( 0 ) ); // seed random-number generator
21
22     // roll dice 6000 times
23     for ( int roll = 1; roll <= 6000; roll++ )
24         ++frequency[ 1 + rand() % 6 ];
```

An array is used instead of 6 regular variables, and the proper element can be updated
This creates a number between 1 and 6, which determines the index of **frequency[]** that should be incremented.

```
26
27     cout << "Face" << setw( 13 ) << "Frequency" << endl;
28
29     // output frequency elements 1-6 in tabular format
30     for ( int face = 1; face < ARRAY_SIZE; face++ )
31         cout << setw( 4 ) << face
32             << setw( 13 ) << frequency[ face ] << endl;
33
34     return 0; // indicates successful termination
35
36 }
```

Program Output ...

Face	Frequency
1	1003
2	1004
3	999
4	980
5	1013
6	1001

Arrays by an Example –VII ...

42

```
1      // C++ Program
2      // Student poll program.
3      #include <iostream>
4
5      using std::cout;
6      using std::endl;
7
8      #include <iomanip>
9
10     using std::setw;
11
12     int main()
13     {
14         // define array sizes
15         const int RESPONSE_SIZE = 40;    // size of array responses
16         const int FREQUENCY_SIZE = 11;   // size of array frequency
17
18         // place survey responses in array responses
19         int responses[RESPONSE_SIZE] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
20             10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
21             5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
22
23         // initialize frequency counters to 0
24         int frequency[FREQUENCY_SIZE] = { 0 };
25
```

```
26 // for each answer, select value of an element of array
27 // responses and use that value as subscript in array
28 // frequency to determine element to increment
29 for ( int answer = 0; answer < RESPONSE_SIZE; answer++ )
30     ++frequency[ responses[answer] ];
31
32 // display results
33 cout << "Rating" << setw( 17 ) << "Frequency" << endl;
34
35 // output frequencies in tabular format
36 for ( int rating = 1; rating < FREQUENCY_SIZE ; rating++ )
37     cout << setw( 6 ) << rating
38         << setw( 17 ) << frequency[ rating ] << endl;
39
40 return 0; // indicates successful termination
41
42 } // end main
```

responses[answer] is the rating (from 1 to 10). This determines the index in **frequency[]** to increment.

Program Output ...

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Questions

45

