In this lecture you will learn:

- **Explore how to declare and manipulate data into arrays**
- **Processing One-Dimensional Arrays**
- **Arrays as Parameters to Functions**
- **Two Dimensional Array**

Array

An array is a collection of a fixed number of components all of the same data type. A one-dimensional array is an array in which the components are arranged in a list form.

The general form for declaring a one-dimensional array is:

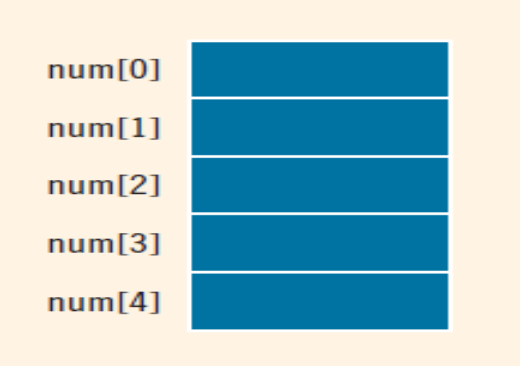
DataType arrayName [intExp];

In which **intExp** is any constant expression that evaluates to a positive integer. Also, intExp specifies the number of components in the array

The statement:

int num[5];

declares an array num of five components. Each component is of type int. The components are num[0], num[1], num[2], num[3], and num[4]. Figure below illustrates the array num.



Processing One-Dimensional Arrays

Some of the basic operations performed on a one-dimensional array are initializing, inputting data, outputting data stored in an array, and finding the largest and/or smallest element. Moreover, if the data is numeric, some other basic operations are finding the sum and average of the elements of the array. Each of these operations requires the ability to step through the elements of the array. This is easily accomplished using a loop. For example, suppose that we have the following statements:

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

The following for loop steps through each element of the array list, starting at the first element of list:

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

If processing the list requires inputting data into list, the statement in Line 2 takes the form of an input statement, such as the cin statement. For example, the following statements read 100 numbers from the keyboard and store the numbers in list:

```
for (i = 0; i < 100; i++)                 //Line 1
```

```
cin >> list[i];                    //Line 2
```

Similarly, if processing list requires outputting the data, then the statement in Line 2 takes the form of an output statement.

Example 1:

```
//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 Initialization during Declaration

Like any other simple variable, an array can be initialized while it is being declared. For example, the following C++ statement declares an array, sales, of five components and initializes these components.

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

or

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

It is not necessary to specify the size of the array. The size is determined by the number of initial values in the braces.

Partial Initialization of Arrays during Declaration

When you declare and initialize an array simultaneously, you do not need to initialize all components of the array. This procedure is called partial initialization of an array during declaration.

The statement:

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

declares list to be an array of 10 components and initializes all of the components to 0.

The statement:

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

declares list to be an array of 10 components and initializes list[0] to 8, list[1] to 5, list[2] to 12, and all other components to 0. Thus, if all of the values are not specified in the initialization statement, the array components for which the values are not specified are initialized to 0.

Arrays as Parameters to Functions

By reference only: In C++, arrays are passed by reference only.

Because arrays are passed by reference only, you do not use the symbol & when declaring an array as a formal parameter.

When declaring a one-dimensional array as a formal parameter, the size of the array is usually omitted. If you specify the size of a one-dimensional array when it is declared as a formal parameter, the size is ignored by the compiler.

Example:

```
void initialize(int list[ ], int listSize)
{
int count;
for (count = 0; count < listSize; count++)
```

```
list[count] = 0;
}

//Function to read and store the data into an int array.
void fillArray(int list[], int listSize)
{
int index;
for (index = 0; index < listSize; index++)
cin >> list[index];
}

//Function to print the elements of an int array.
void printArray(const int list[], int listSize)
{
int index;
for (index = 0; index < listSize; index++)
cout << list[index] << " ";
}
```

```
//Function to find and return the sum of the one dimensional array
int sumArray(const int list[], int listSize)
{
int index;
int sum = 0;
for (index = 0; index < listSize; index++)
sum = sum + list[index];
```

```
return sum;
}
//Function to find and return the index of the first largest element in an int
//array. The parameter listSize specifies the number of elements in the array.
int indexLargestElement(const int list[], int listSize)
{
int index;
int maxIndex = 0; //assume the first element is the largest
for (index = 1; index < listSize; index++)
if (list[maxIndex] < list[index])
maxIndex = index;
return maxIndex;
}
```

Note: Functions Cannot Return a Value of the Type Array

Searching an Array for a Specific Item

```
int seqSearch(const int list[], int listLength, int searchItem)
{
int loc;
bool found = false;
loc = 0;
while (loc < listLength && !found)
if (list[loc] == searchItem)
found = true;
else
loc++;
```

```
if (found)
return loc;
else
return -1;
}
```

Two- and Multidimensional Arrays

In the previous section, you learned how to use one-dimensional arrays to manipulate data. If the data is provided in a list form, you can use one-dimensional arrays. However, sometimes data is provided in a table form. For example, suppose that you want to track the number of cars in a particular color that are in stock at a local dealership. The dealership sells six types of cars in five different colors.

Two-dimensional array: A collection of a fixed number of components arranged in rows and columns (that is, in two dimensions), wherein all components are of the same type. The syntax for declaring a two-dimensional array is:

```
dataType  arrayName[intExp1][intExp2];
```

The statement:

double sales[10][5];

Declares a two-dimensional array sales of 10 rows and 5 columns, in which every component is of type double. As in the case of a one-dimensional array, the rows are numbered 0. . .9 and the columns are numbered 0. . .4.

inStock	[RED]	[BROWN]	[BLACK]	[WHITE]	[GRAY]
[GM]	10	7	12	10	4
[FORD]	18	11	15	17	10
[TOYOTA]	12	10	9	5	12
[BMW]	16	6	13	8	3
[NISSAN]	10	7	12	6	4
[VOLVO]	9	4	7	12	11

Two-Dimensional Array Initialization during Declaration

Like one-dimensional arrays, two-dimensional arrays can be initialized when they are declared. The following example helps illustrate this concept. Consider the following statement:

int board[4][3] = {{2, 3, 1}, {15, 25, 13}, {20, 4, 7}, {11, 18, 14}};

This statement declares board to be a two-dimensional array of four rows and three columns. The components of the first row are 2, 3, and 1; the components of the second row are 15, 25, and 13; the components of the third row are 20, 4, and 7; and the components of the fourth row are 11, 18, and 14, respectively.

PROCESSING TWO-DIMENSIONAL ARRAYS

A two-dimensional array can be processed in three ways:

1. Process the entire array.
2. Process a particular row of the array, called row processing.
3. Process a particular column of the array, called column processing.

Initialization

Suppose that you want to initialize row number 4, that is, the fifth row, to 0.

the following for loop does this:

```
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
matrix[row][col] = 0;
```

If you want to initialize the entire matrix to 0, you can also put the first index, that is, the row position, in a loop. By using the following nested for loops, we can initialize each component of matrix to 0:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
matrix[row][col] = 0;
```

Print

By using a nested for loop, you can output the components of matrix. The following nested for loops print the components of matrix, one row per line:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
cout << setw(5) << matrix[row][col] << " ";
cout << endl;
}
```

Input

The following for loop inputs the data into row number 4, that is, the fifth row of matrix:

```
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
cin >> matrix[row][col];
```

As before, by putting the row number in a loop, you can input data into each component of matrix. The following for loop inputs data into each component of matrix:

```
for (row = 0; row < NUMBER_OF_ROWS; row++)
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
cin >> matrix[row][col];
```

Sum by Row

The following for loop finds the sum of row number 4 of matrix; that is, it adds the components of row number 4.

```
sum = 0;
row = 4;
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
sum = sum + matrix[row][col];
```

Once again, by putting the row number in a loop, we can find the sum of each row separately. Following is the C++ code to find the sum of each individual row:

```
//Sum of each individual row
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
sum = 0;
```

```
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
sum = sum + matrix[row][col];
cout << "Sum of row " << row + 1 << " = " << sum << endl;
}
```

Sum by Column

As in the case of sum by row, the following nested for loop finds the sum of each individual column:

//Sum of each individual column

```
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{
sum = 0;
for (row = 0; row < NUMBER_OF_ROWS; row++)
sum = sum + matrix[row][col];
cout << "Sum of column " << col + 1 << " = " << sum
<< endl;
}
```

Largest element

The following function determines the largest element in each row:

```
void largestInRows(int matrix[][NUMBER_OF_COLUMNS],
int noOfRows)
{
int row, col;
int largest;
//Largest element in each row
```

```
for (row = 0; row < noOfRows; row++)
{
largest = matrix[row][0]; //Assume that the first element
//of the row is the largest.
for (col = 1; col < NUMBER_OF_COLUMNS; col++)
if (largest < matrix[row][col])
largest = matrix[row][col];
cout << "The largest element of row " << (row + 1)
<< " = " << largest << endl;
}
}
```

References:

[1] Deitel, P. & Deitel, R., "C++ How to Program", Eighth Edition, Pearson Education Inc., 2012.

[2] Stefan B., " C++ Primer Plus ", Sixth Edition, Pearson Education Inc., 2012.