

C++ Programming: From Problem Analysis to Program Design, Sixth Edition

Chapter 08: ARRAYS AND STRINGS

Objectives

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
- Learn how to search an array
- Learn how to sort an array

Objectives

In this chapter, you will:

- Learn about C-strings
- Examine the use of string functions to process C-strings
- Discover how to input data into—and output data from—a C-string
- Learn about parallel arrays
- Discover how to manipulate data in a two-dimensional array
- Learn about multidimensional arrays

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];
```

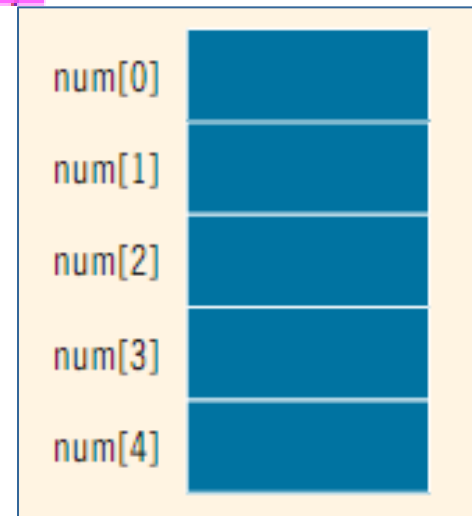


FIGURE 8-1 Array num

Accessing Array Components

The general form (syntax) used for accessing an array component is:

```
arrayName[indexExp]
```

in which **indexExp**, called the **index**, is any expression whose value is a nonnegative integer. The index value specifies the position of the component in the array.

In C++, **[]** is an operator called the **array subscripting operator**. Moreover, in C++, the array index starts at 0.

Consider the following statement:

```
int list[10];
```

This statement declares an array **list** of 10 components. The components are **list[0]**, **list[1]**, ..., **list[9]**. In other words, we have declared 10 variables (see Figure 8-3).

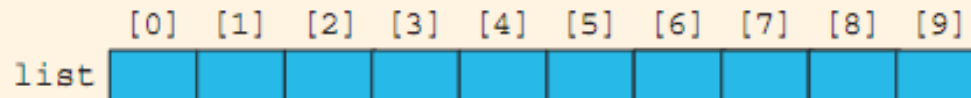


FIGURE 8-3 Array list

Accessing Array Components

The assignment statement:

```
list[5] = 34;
```

stores 34 in `list[5]`, which is the sixth component of the array `list` (see Figure 8-4).

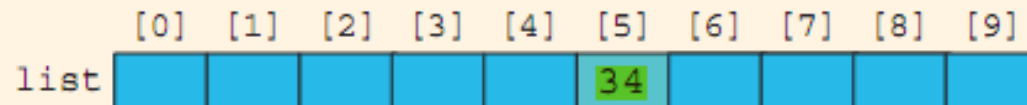



FIGURE 8-4 Array `list` after execution of the statement `list[5] = 34;`

Processing One-Dimensional Arrays

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

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

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



read 100 numbers from the keyboard and store the numbers in list

Processing One-Dimensional Arrays/Example

```
double sales[10];  
int index;  
double largestSale, sum, average;
```

- a. **Initializing an array:** The following loop initializes every component of the array `sales` to 0.0.

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

- b. **Reading data into an array:** The following loop inputs the data into the array `sales`. For simplicity, we assume that the data is entered from the keyboard.

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

- c. **Printing an array:** The following loop outputs the array `sales`. For simplicity, we assume that the output goes to the screen.

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


Processing One-Dimensional Arrays/Example

d. Finding the sum and average of an array:

```
sum = 0;
for (index = 0; index < 10; index++)
    sum = sum + sales[index];

average = sum / 10;
```

e. Largest element in the array:

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

Processing One-Dimensional Arrays/Example

Let us demonstrate how this algorithm works with an example. Suppose the array **sales** is as given in Figure 8-6.

	[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 8-6 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.

Assignments on array

1. Write a C++ program that declares an array **alpha** of 50 components of type **double**. Initialize the array so that the first 25 components are equal to the square of the index variable, and the last 25 components are equal to three times the index variable. Output the array so that 10 elements per line are printed.
2. Write a C++ function, **smallestIndex**, that takes as parameters an **int** array and its size and returns the index of the first occurrence of the smallest element in the array. Also, write a program to test your function.
3. Write a C++ function, **lastLargestIndex**, that takes as parameters an **int** array and its size and returns the index of the last occurrence of the largest element in the array. Also, write a program to test your function.