Chapter 9: Arrays and Strings

In this chapter you will learn about:

- Arrays as Parameters to Functions

Arrays as Parameters to Functions

Now that you have seen how to work with arrays, a question naturally arises: How are arrays passed 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 9-5

Consider the following function:

```
void funcArrayAsParam(int listOne[], double listTwo[])
{
      .
      .
      .
}
```

The function `funcArrayAsParam` has two formal parameters: (1) `listOne`, a one-dimensional array of type `int` (that is, the component type is `int`) and (2) `listTwo`, a one-dimensional array of type `double`. In this declaration, the size of both arrays is unspecified.

Sometimes, the number of elements in the array might be less than the size of the array. For example, the number of elements in an array storing student data might increase or decrease as students drop or add courses. In such situations, we want to process only the components of the array that hold actual data. To write a function to process such arrays, in addition to declaring an array as a formal parameter, we declare another formal parameter specifying the number of elements in the array, as in the following function:

```
void initialize(int list[], int listSize)
{
    int count;

    for (count = 0; count < listSize; count++)
        list[count] = 0;
}
```

The first parameter of the function `initialize` is an `int` array of any size. When the function `initialize` is called, the size of the actual array is passed as the second parameter of the function `initialize`.

Constant Arrays as Formal Parameters

Recall that when a formal parameter is a reference parameter, then whenever the formal parameter changes, the actual parameter changes as well. However, even though an array is always passed by reference, you can still prevent the function from changing the actual parameter. You do so by using the reserved word `const` in the declaration of the formal parameter. Consider the following function:

```
void example(int x[], const int y[], int sizeX, int sizeY)
{
      .
      .
      .
}
```

Here, the function `example` can modify the array `x`, but not the array `y`. Any attempt to change `y` results in a compile-time error. It is a good programming practice to declare an array to be constant as a formal parameter if you do not want the function to modify the array.

Example

This example shows how to write functions for array processing and declare an array as a formal parameter.

```
//Function to initialize an int array to 0.
//The array to be initialized and its size are passed
//as parameters. The parameter listSize specifies the
//number of elements to be initialized.
void initializeArray(int list[], int listSize)
{
 int index;
 for (index = 0; index < listSize; index++)
      list[index] = 0;
}
```

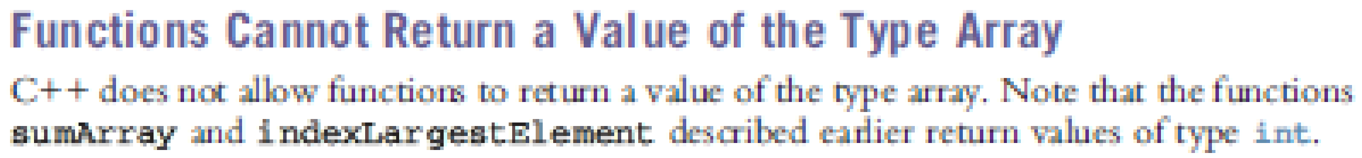
```
//Function to read and store the data into an int
array.
//The array to store the data and its size are
passed as
//parameters. The parameter listSize specifies the
number
//of elements to be read.

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.
//The array to be printed and the number of
elements
//are passed as parameters. The parameter
listSize
//specifies the number of elements to be printed.
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
//elements of an int array. The parameter listSize
//specifies the number of elements to be added.
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;
}
```



Functions Cannot Return a Value of the Type Array

C++ does not allow functions to return a value of the type array. Note that the functions **sumArray** and **indexLargestElement** described earlier return values of type **int**.

References

1. Malik, D. S. (2010). *C++ programming: Program design including data structures.* Course Technology.