COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 8
Arrays and Strings

Objectives

- In this chapter, you will:
  - Learn the reasons for arrays
  - Explore how to declare and manipulate data into arrays
  - Understand the meaning of ''array index out of bounds''
  - Learn how to declare and initialize arrays
  - Become familiar with the restrictions on array processing

Objectives (cont’d.)

- Discover how to pass an array as a parameter to a function
- Learn how to search an array
- Learn how to sort an array
- Become aware of `auto` declarations
- Learn about range-based `for` loops
- Learn about `C`-strings

Objectives (cont'd.)

- 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

Introduction

<u>Simple data type</u>: variables of these types can store only one value at a time

<u>Structured data type</u>: a data type in which each data item is a collection of other data items

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

#include <iostream>

using namespace std;

int main()
{
    int item0, item1, item2, item3, item4;
    int sum;

    cout << "Enter five integers: ";
    cin >> item0 >> item1 >> item2 >> item3 >> item4;
    cout << endl;

    sum = item0 + item1 + item2 + item3 + item4;

    cout << "The sum of the numbers = " << sum << endl;
    cout << "The numbers in the reverse order are: ";
    cout << item4 << " " << item3 << " " << item2 << " "
         << item1 << " " << item0 << endl;

    return 0;
}
```

Arrays

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

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

- intExp: any constant expression that evaluates to a positive integer

Accessing Array Components

- General syntax:

```
arrayName[indexExp]
```

- `indexExp`: called the index
  - An expression with a nonnegative integer value
- Value of the index is the position of the item in the array
- `[]`: array subscripting operator
  - Array index always starts at 0

Accessing Array Components (cont'd.)

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

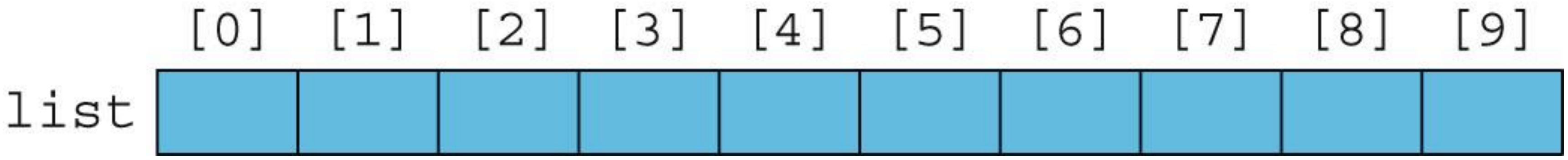


FIGURE 8-3 Array `list`

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

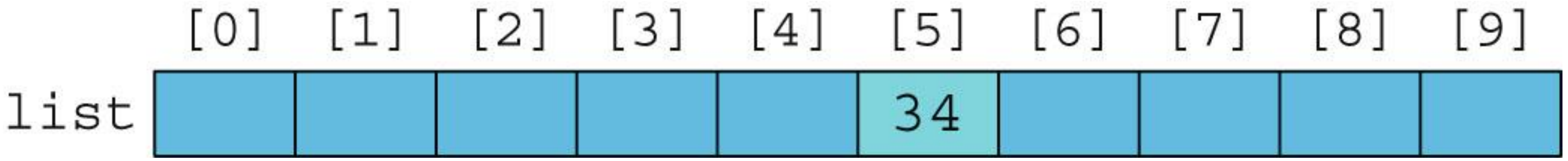


FIGURE 8-4 Array `list` after execution of the statement `list[5]= 34;`

Accessing Array Components (cont'd.)

```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```

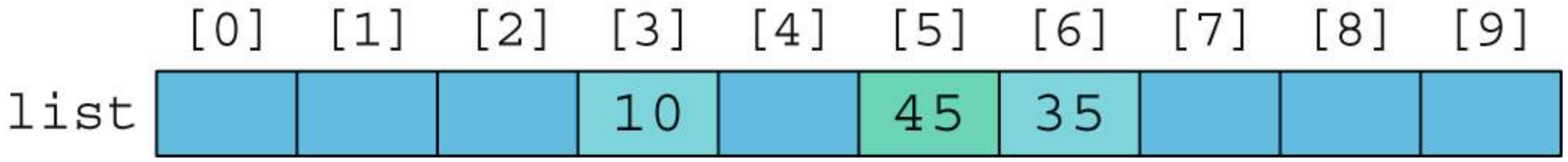


FIGURE 8-5 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];`

Processing One-Dimensional Arrays

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

Processing One-Dimensional Arrays (cont'd.)

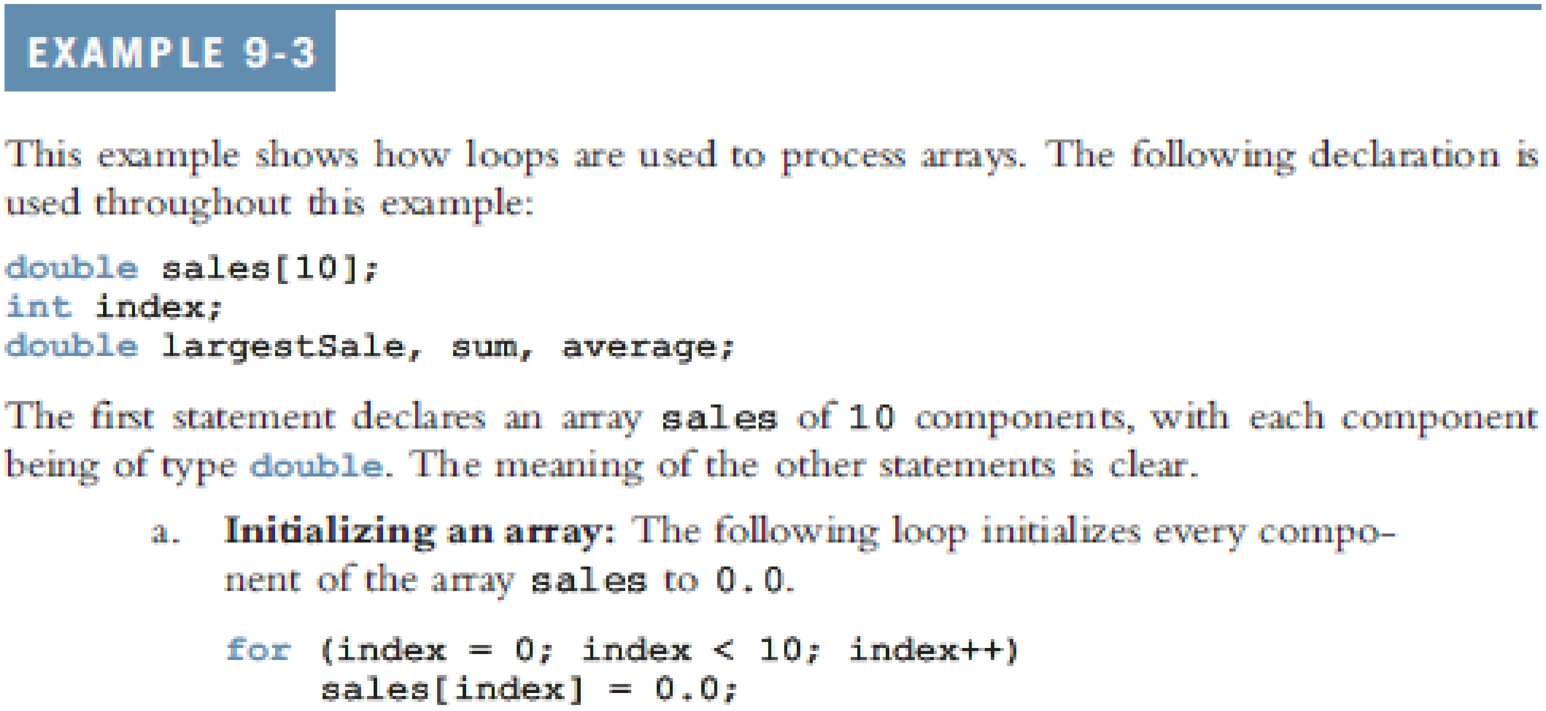
- Given the declaration:

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

- Use a `for` loop to access array elements:

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

Processing One-Dimensional Arrays (cont'd.)



EXAMPLE 9-3

This example shows how loops are used to process arrays. The following declaration is used throughout this example:

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

The first statement declares an array `sales` of `10` components, with each component being of type `double`. The meaning of the other statements is clear.

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;
```

Processing One-Dimensional Arrays (cont'd.)

Reading data into an array: The following loop inputs the data into the array sales.

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

Printing an array: The following loop outputs the array sales.

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

Finding the sum and average of an array:

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

Processing One-Dimensional Arrays (cont’d.)

Largest element in the array:

The algorithm is as follows:

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

largestSale = sales[maxIndex];
```

Array Index Out of Bounds

- Index of an array is <u>in bounds</u> if the index is `>=0` and `<= ARRAY_SIZE-1`
  - Otherwise, the index is <u>out of bounds</u>
- In C++, there is no guard against indices that are out of bounds

Array Initialization During Declaration

- Arrays can be initialized during declaration
  - Values are placed between curly braces
  - Size determined by the number of initial values in the braces
- Example:

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

Partial Initialization of Arrays During Declaration

- The statement:

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

  - Declares an array of 10 components and initializes all of them to zero

- The statement:

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

  - Declares an array of 10 components and initializes `list[0]` to 8, `list[1]` to 5, `list[2]` to 12
  - All other components are initialized to 0

Some Restrictions on Array Processing

- Aggregate operation: any operation that manipulates the entire array as a single unit
  - Not allowed on arrays in C++
- Example:

```
int myList[5] = {0, 4, 8, 12, 16};  //Line 1
int yourList[5];  //Line 2
 yourList = myList;  //illegal
```

- Solution:

```
for (int index = 0; index < 5; index ++)
    yourList[index] = myList[index];
```

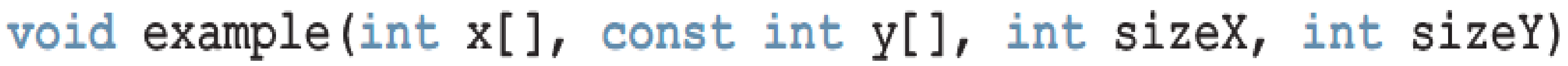
Arrays as Parameters to Functions

- Arrays are passed by reference only
- Do not use symbol `&` when declaring an array as a formal parameter
- Size of the array is usually omitted
  - If provided, it is ignored by the compiler
- Example:

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

Constant Arrays as Formal Parameters

- Can prevent a function from changing the actual parameter when passed by reference
  - Use `const` in the declaration of the formal parameter
- Example:



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

Base Address of an Array and Array in Computer Memory

- Base address of an array: address (memory location) of the first array component
- Example:
  - If `list` is a one-dimensional array, its base address is the address of `list[0]`
- When an array is passed as a parameter, the base address of the actual array is passed to the formal parameter

Functions Cannot Return a Value of the Type Array

- C++ does not allow functions to return a value of type array