Chapter 8
Arrays and Strings

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

Arrays

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

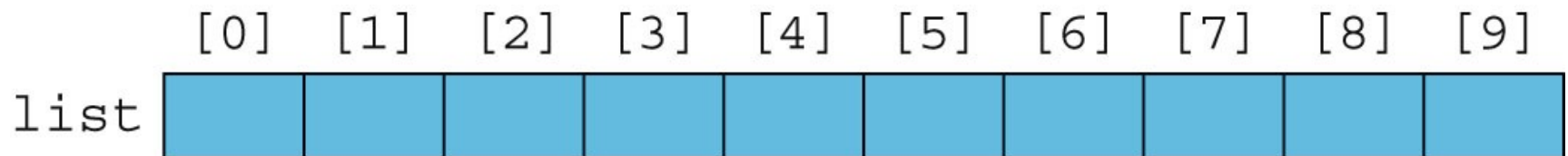# Accessing Array Components (cont'd.)

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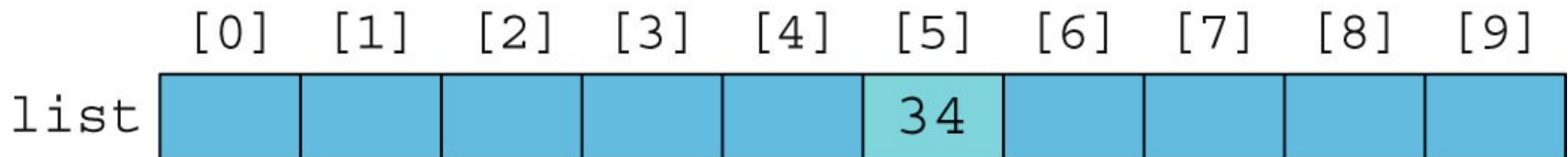

FIGURE 8-3 Array list

```
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];
```
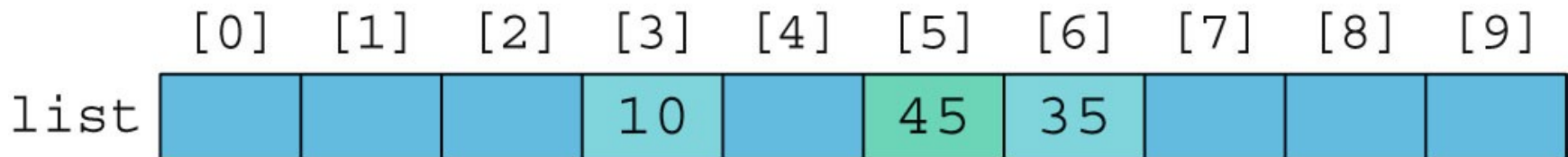
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
list				10		45	35			

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
```

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:

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

- Solution:

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

Arrays as Parameters to Functions

- Arrays are passed <u>by reference only</u>

- 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:

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

Base Address of an Array and Array in Computer Memory

- <u>Base address</u> 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

Integral Data Type and Array Indices

- C++ allows any integral type to be used as an array index
  - Improves code readability
- Example:

```cpp
enum paintType {GREEN, RED, BLUE, BROWN, WHITE, ORANGE, YELLOW};
double paintSale[7];
paintType paint;

for (paint = GREEN; paint <= YELLOW;
                    paint = static_cast<paintType>(paint + 1))
     paintSale[paint] = 0.0;

paintSale[RED] = paintSale[RED] + 75.69;
```

Other Ways to Declare Arrays

- Examples:

```cpp
const int NO_OF_STUDENTS = 20;
int testScores[NO_OF_STUDENTS];
```

```cpp
const int SIZE = 50;              //Line 1
typedef double list[SIZE];        //Line 2

list yourList;                    //Line 3
list myList;                      //Line 4
```

Searching an Array for a Specific Item

- <u>Sequential search</u> (or <u>linear search</u>):
  - Searching a list for a given item, starting from the first array element
  - Compare each element in the array with value being searched for
  - Continue the search until item is found or no more data is left in the list

Selection Sort

- <u>Selection sort</u>: rearrange the list by selecting an element and moving it to its proper position

- Steps:
  - Find the smallest element in the unsorted portion of the list
  - Move it to the top of the unsorted portion by swapping with the element currently there
  - Start again with the rest of the list

Selection Sort (cont'd.)

FIGURE 8-10   Elements of list during the first iteration
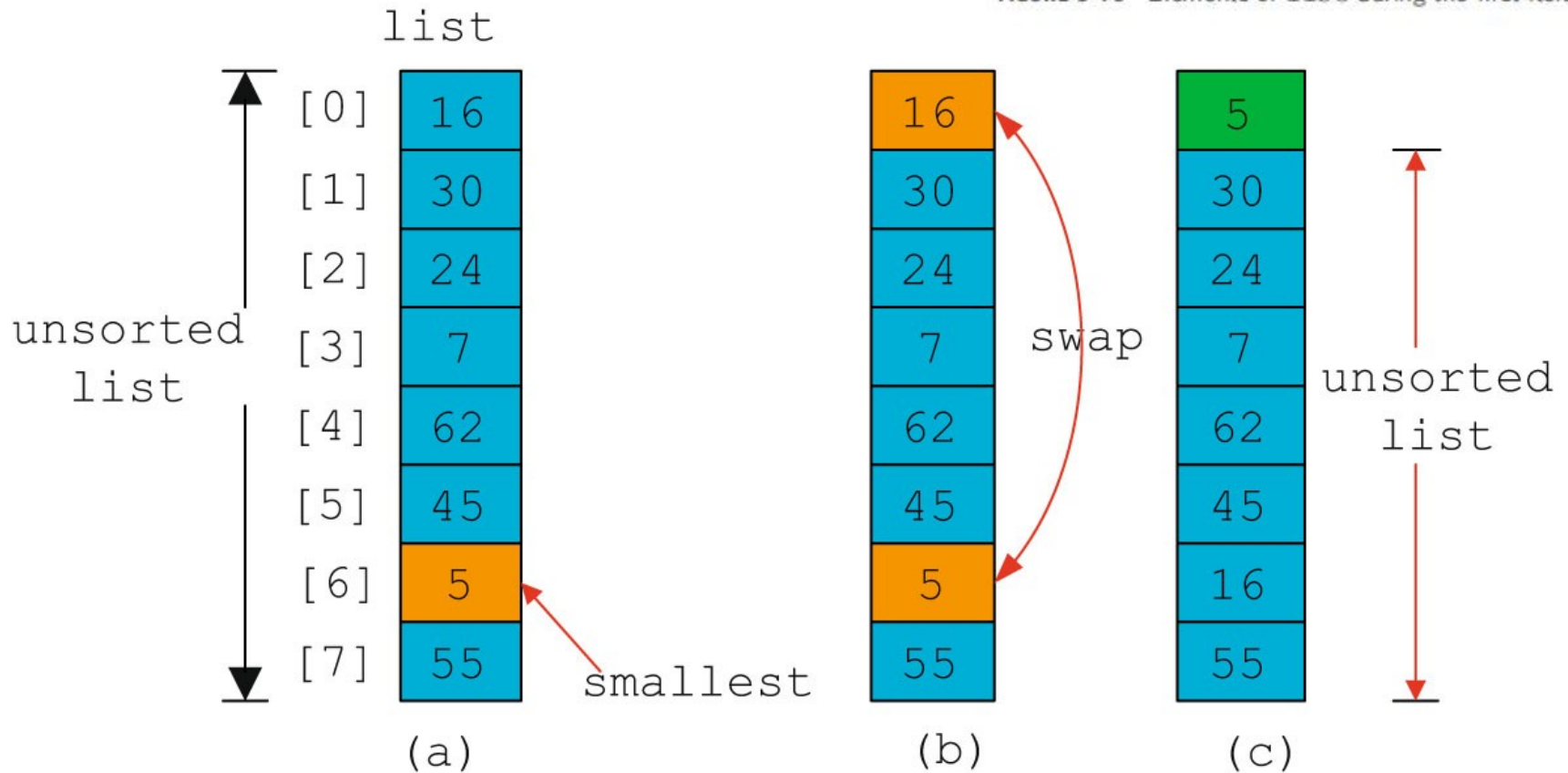
C-Strings (Character Arrays)

- Character array: an array whose components are of type `char`

- `C`-strings are null-terminated (`'\0'`) character arrays

- Example:
  - `'A'` is the character `A`
  - `"A"` is the `C`-string `A`
  - `"A"` represents two characters, `'A'` and `'\0'`

C-Strings (Character Arrays) (cont'd.)

- Example:

  ```
  char name[16];
  ```

- Since `C`-strings are null terminated and `name` has 16 components, the largest string it can store has 15 characters

- If you store a string whose length is less than the array size, the last components are unused

C-Strings (Character Arrays) (cont'd.)

- Size of an array can be omitted if the array is initialized during declaration

- Example:

  ```
  char name[] = "John";
  ```

  – Declares an array of length 5 and stores the C-string "John" in it

- Useful string manipulation functions

  – `strcpy`, `strcmp`, and `strlen`

String Comparison

- `C`-strings are compared character by character using the collating sequence of the system
  - Use the function `strcmp`

- If using the ASCII character set:
  - "Air" < "Boat"
  - "Air" < "An"
  - "Bill" < "Billy"
  - "Hello" < "hello"

Reading and Writing Strings

- Most rules for arrays also apply to C-strings (which are character arrays)

- Aggregate operations, such as assignment and comparison, are not allowed on arrays

- C++ does allow aggregate operations for the input and output of C-strings

String Input

- Example:

```
cin >> name;
```

  - Stores the next input C-string into name

- To read strings with blanks, use `get` function:

```
cin.get(str, m+1);
```

  - Stores the next `m` characters into `str` but the newline character is not stored in `str`

  - If input string has fewer than `m` characters, reading stops at the newline character

String Output

- Example:

  ```
  cout << name;
  ```
  - Outputs the content of name on the screen
  - `<<` continues to write the contents of `name` until it finds the null character
  - If `name` does not contain the null character, then strange output may occur
    - `<<` continues to output data from memory adjacent to `name` until a `'\0'` is found

Specifying Input/Output Files at Execution Time

- User can specify the name of an input and/or output file at execution time:

```
ifstream infile;
ofstream outfile;

char fileName[51];       //assume that the file name is at most
                         //50 characters long

cout << "Enter the input file name: ";
cin >> fileName;

infile.open(fileName);   //open the input file
```

`string` Type and Input/Output Files

- Argument to the `open` function must be a null-terminated string (a `C`-string)
  - If using a `string` variable for the name of an I/O file, the value must first be converted to a `C`-string before calling `open`
  - Use the `c_str` function to convert

- Syntax:

`strVar.c_str()`

  - Where `strVar` is a variable of type `string`

Parallel Arrays

- Two (or more) arrays are called <u>parallel</u> if their corresponding components hold related information

- Example:

```
int studentId[50];
char courseGrade[50];
```

```
23456 A
86723 B
22356 C
92733 B
11892 D
.
.
.
```

Two- and Multidimensional Arrays

- <u>Two-dimensional array</u>: collection of a fixed number of components (of the same type) arranged in two dimensions
  - Sometimes called matrices or tables

- Declaration syntax:

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

  - `intExp1` and `intExp2` are expressions with positive integer values specifying the number of rows and columns in the array

Accessing Array Components

- Accessing components in a two-dimensional array:

  > arrayName[indexExp1][indexExp2]

  - Where `indexExp1` and `indexExp2` are expressions with positive integer values, and specify the row and column position

- Example:

  ```
  sales[5][3] = 25.75;
  ```

Accessing Array Components (cont'd.)

FIGURE 8-14 sales[5][3]

Two-Dimensional Array Initialization During Declaration

- Two-dimensional arrays can be initialized when they are declared:
  - Elements of each row are enclosed within braces and separated by commas
  - All rows are enclosed within braces
  - For number arrays, unspecified elements are set to 0

Two-Dimensional Arrays and Enumeration Types

- Enumeration types can be used for array indices:

```
const int NUMBER_OF_ROWS = 6;
const int NUMBER_OF_COLUMNS = 5;
enum carType {GM, FORD, TOYOTA, BMW, NISSAN, VOLVO};
enum colorType {RED, BROWN, BLACK, WHITE, GRAY};

int inStock[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];


inStock[FORD][WHITE] = 15;
```

Processing Two-Dimensional Arrays

- Ways to process a two-dimensional array:
  - Process entire array
  - <u>Row processing</u>: process a single row at a time
  - <u>Column processing</u>: process a single column at a time
- Each row and each column of a two-dimensional array is a one-dimensional array
  - To process, use algorithms similar to processing one-dimensional arrays

Initialization

- Examples:
  - To initialize row number 4 (fifth row) to 0:

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

  - To initialize the entire matrix to 0:

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

Print

- Use a nested loop to output the components of a two dimensional array:

```cpp
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

- Examples:
  - To input into row number 4 (fifth row):

```cpp
row = 4;

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

  - To input data into each component of matrix:

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

Sum by Row

- Example:
  - To find the sum of row number 4:

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

Sum by Column

- Example:
  - To find the sum of each individual column:

```cpp
//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 in Each Row and Each Column

- Example:
  - To find the largest element in each row:

```cpp
//Largest element in each row
for (row = 0; row < NUMBER_OF_ROWS; 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 in row " << row + 1 << " = "
         << largest << endl;
}
```

Passing Two-Dimensional Arrays as Parameters to Functions

- Two-dimensional arrays are passed by reference as parameters to a function
  - Base address is passed to formal parameter
- Two-dimensional arrays are stored in <u>row order</u>
- When declaring a two-dimensional array as a formal parameter, can omit size of first dimension, but not the second

Arrays of Strings

- Strings in C++ can be manipulated using either the data type `string` or character arrays (`C`-strings)

- On some compilers, the data type `string` may not be available in Standard C++ (i.e., non-ANSI/ISO Standard C++)

Arrays of Strings and the `string` Type

- To declare an array of 100 components of type `string`:

  ```
  string list[100];
  ```

- Basic operations, such as assignment, comparison, and input/output, can be performed on values of the `string` type

- The data in `list` can be processed just like any one-dimensional array

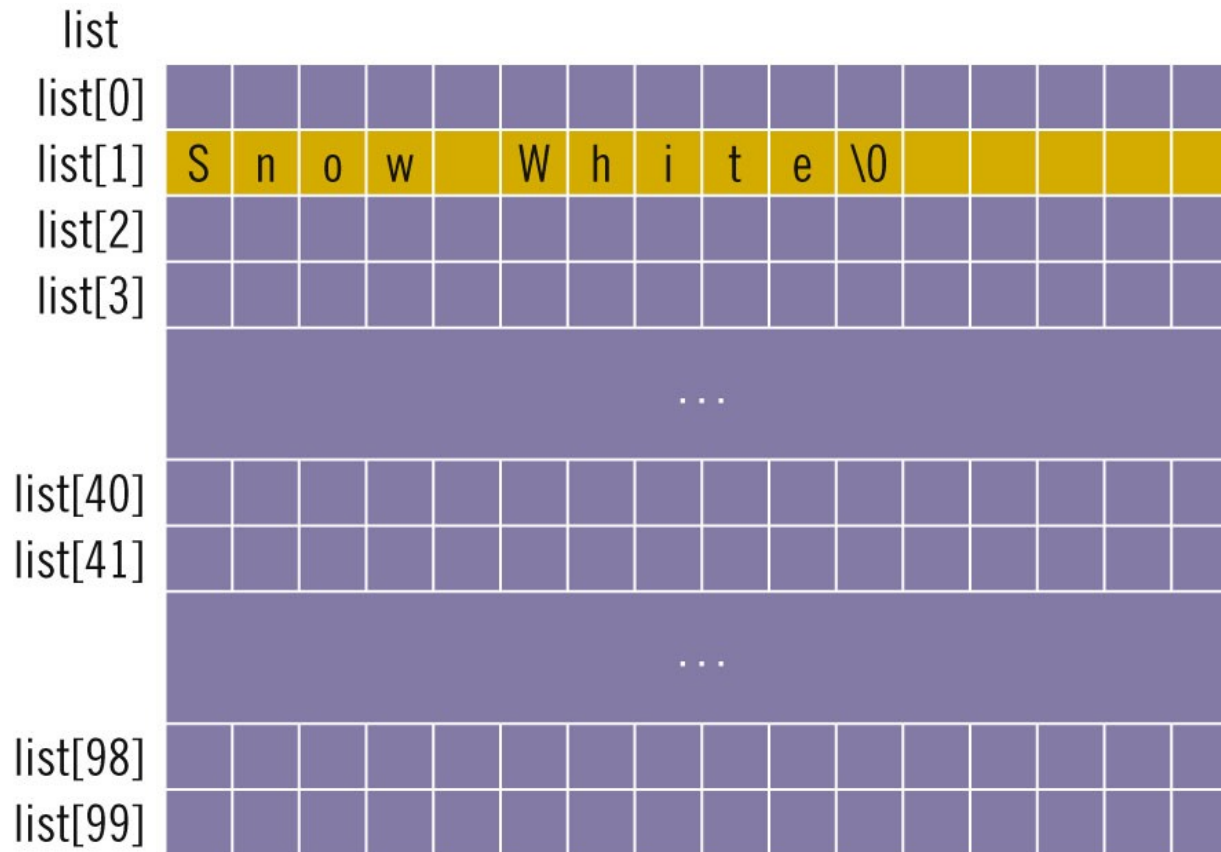# Arrays of Strings and C-Strings (Character Arrays)

FIGURE 8-20 Array `list`, showing `list[1]`

Another Way to Declare a Two-Dimensional Array

- Can use `typedef` to define a two-dimensional array data type:

```
const int NUMBER_OF_ROWS = 20;
const int NUMBER_OF_COLUMNS = 10;

typedef int tableType[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
```

- To declare an array of 20 rows and 10 columns:

```
tableType matrix;
```

Multidimensional Arrays

- *n*-dimensional array: collection of a fixed number of elements arranged in *n* dimensions ($n >= 1$)

- Declaration syntax:

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

- To access a component:

```
arrayName[indexExp1][indexExp2] ... [indexExpn]
```

Summary

- Array: structured data type with a fixed number of components of the same type
  - Components are accessed using their relative positions in the array

- Elements of a one-dimensional array are arranged in the form of a list

- An array index can be any expression that evaluates to a nonnegative integer
  - Must always be less than the size of the array

Summary (cont'd.)

- The base address of an array is the address of the first array component

- When passing an array as an actual parameter, use only its name
  - Passed by reference only

- A function cannot return an array type value

- In C++, $\mathbb{C}$-strings are null terminated and are stored in character arrays

Summary (cont'd.)

- Commonly used `C`-string manipulation functions include:

  - `strcpy`, `strcmp`, and `strlen`

- Parallel arrays are used to hold related information

- In a two-dimensional array, the elements are arranged in a table form

Summary (cont'd.)

- To access an element of a two-dimensional array, you need a pair of indices:

  - One for row position, one for column position

- In row processing, a two-dimensional array is processed one row at a time

- In column processing, a two-dimensional array is processed one column at a time