Lesson 4

Arrays

Introduction

- Simple data type: variables of these types can store only one value at a time
- Structured data type: a data type in which each data item is a collection of other data items

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 ... ates 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];
```

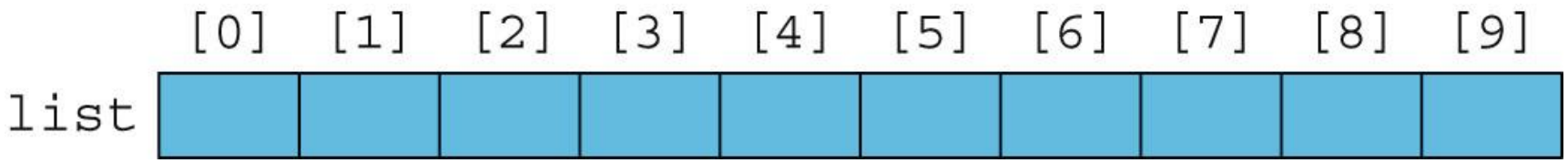

FIGURE 8-3 Array `list`

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

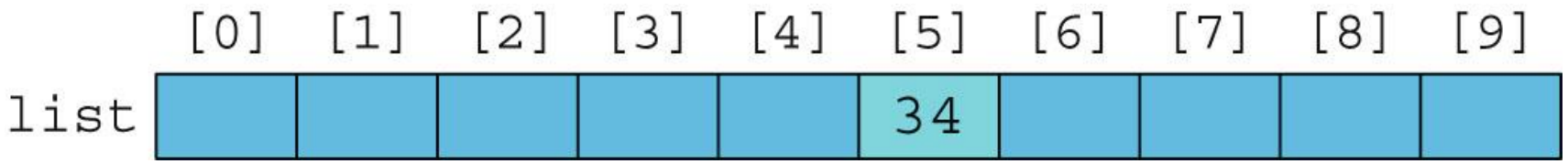

FIGURE 8-4 Array `list` after execution of the statement `list[5]= 34;`

Accessing Array Components (cont'd.)

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

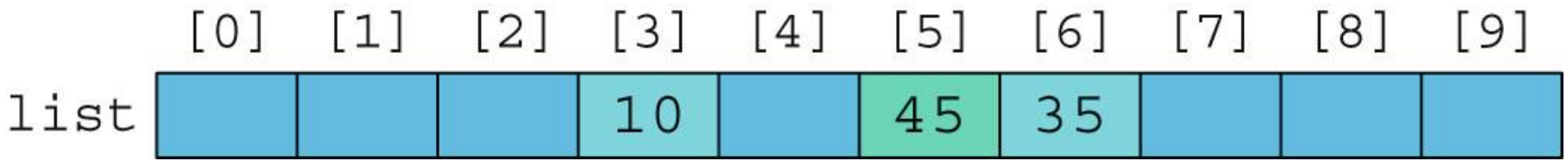

FIGURE 8-5 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];`

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

- <u>Aggregate operation</u>: 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
```

- Solution:

```
yourList = myList;  //illegal
```

```
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:

```
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

Searching an Array for a Specific Item

- Sequential search (or linear search):
  - 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

Sorting

- <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.)

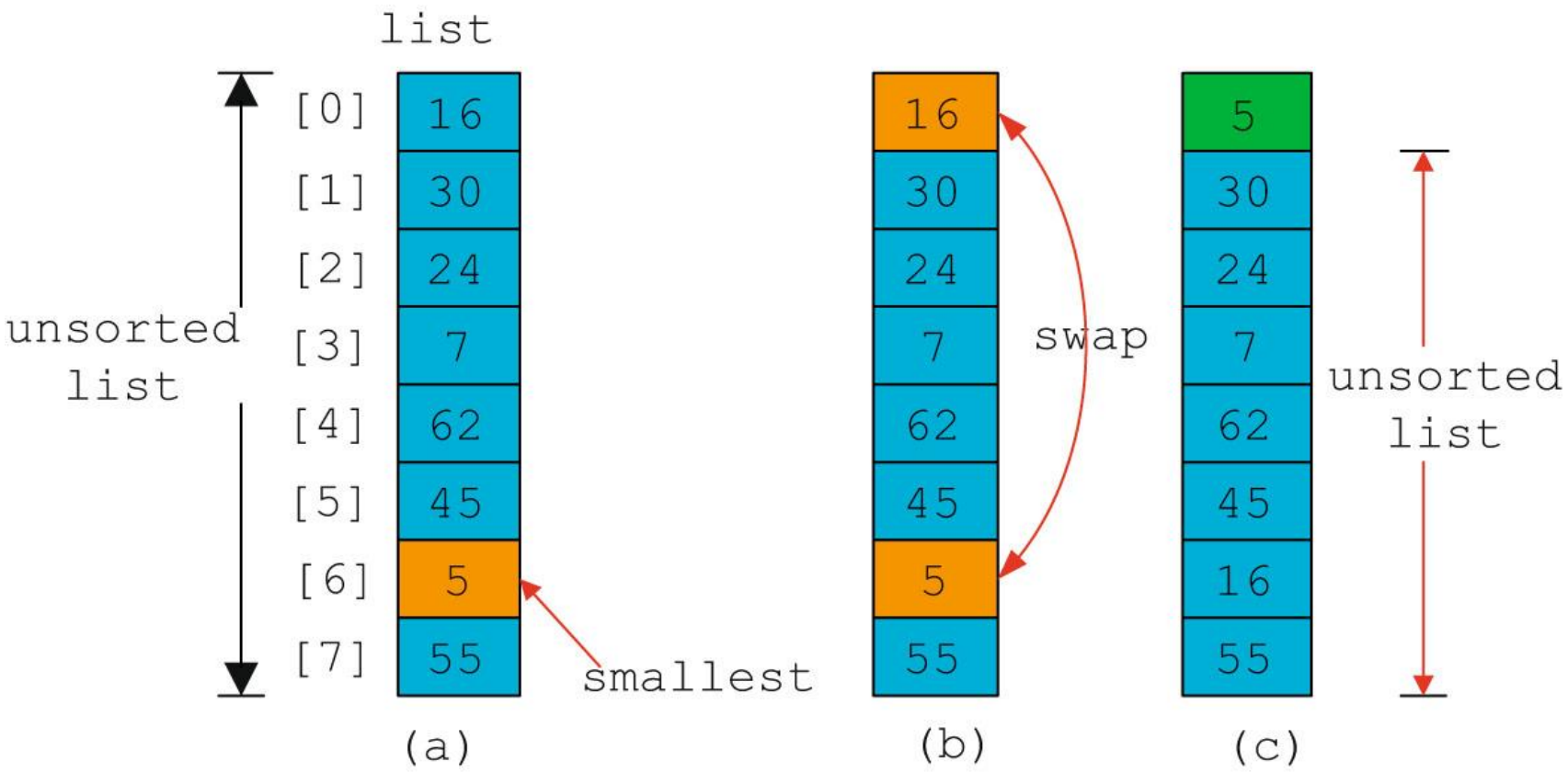

FIGURE 8-10 Elements of `list` during the first iteration

Auto Declaration and Range-Based `For` Loops

- C++11 allows auto declaration of variables
  - Data type does not need to be specified

```
auto num = 15; // num is assumed int
```

- Range-based `for` loop

```
sum = 0;
for (double num : list) // For each num
    sum = sum + num;     // in list
```

C-Strings (Character Arrays)

- <u>Character array</u>: 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

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

- Two-dimensional array: 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.)

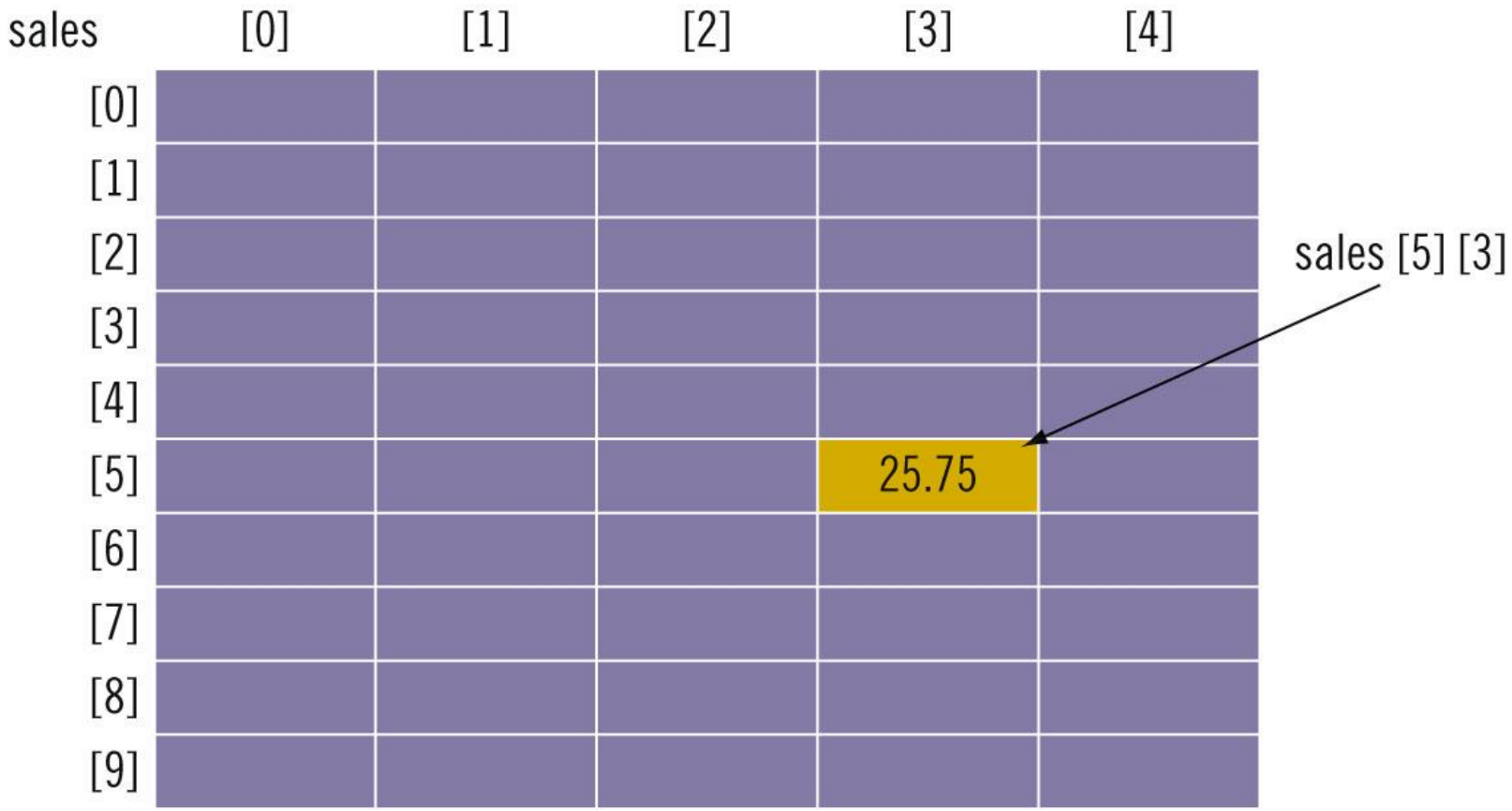

FIGURE 8-14 sales[5][3]

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

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:

```
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):

```
row = 4;

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

  - To input 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

- 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:

```
  //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:

```
  //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;
}
```