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 constant expression that evaluates to a positive integer

Accessing Array Components (1 of 3)

- 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 (2 of 3)

This statement declares an array of 10 components:

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

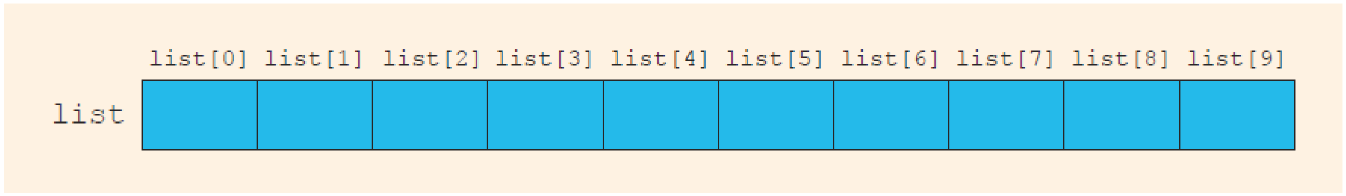

FIGURE 8-3 Array **`list`**

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

stores **34** in **`list[5]`**, the *sixth* component of the array **`list`**

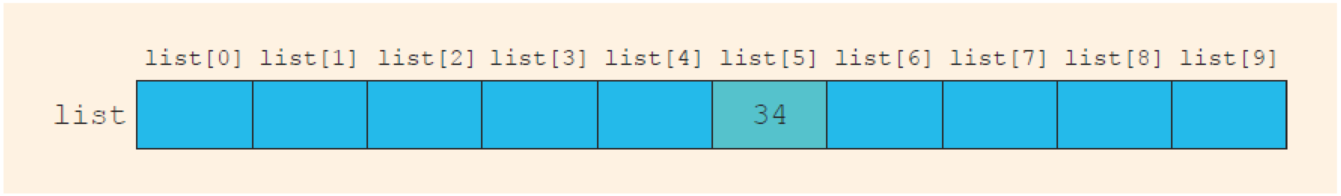

FIGURE 8-4 Array **`list`** after execution of the statement **`list[5]= 34;`**

Accessing Array Components (3 of 3)

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

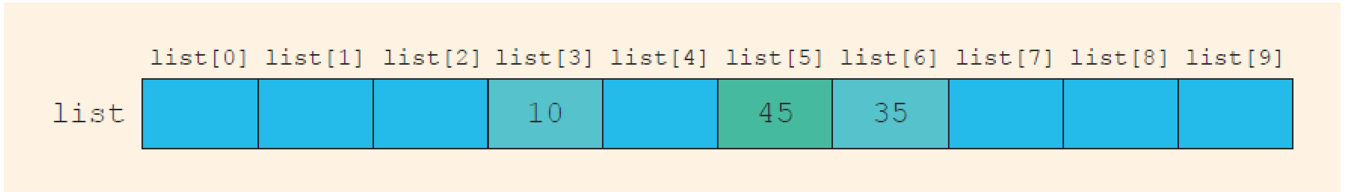

FIGURE 8-5 Array `list` after execution of the statements `list[3]= 10;`, `list[6]= 35;`, `and list[5] = list[3] + list[6];`

Processing One-Dimensional Arrays (1 of 3)

- Basic operations on a one-dimensional array include:
  - 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 using a loop

Processing One-Dimensional Arrays (2 of 3)

- 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

- The index of an array is <u>in bounds</u> if the index is between `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
  - This check is solely the programmer’s responsibility

Array Initialization During Declaration

- Arrays can be initialized during declaration
  - Values are placed between curly braces
- Example 1

```
double sales[5] = {12.25, 32.50, 16.90, 23, 45.68}
```

- Example 2: the array size is determined by the number of initial values in the braces if the array is declared without size specified

```
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 (an example of partial initialization of an array during declaration):

```
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
- The size of the array is usually omitted in the array parameter
  - If provided, it is ignored by the compiler
- The following example illustrates a function header, which includes an array parameter and a parameter specifying the number of elements in the array:

```
void initialize(int list[], int listSize)
```

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

- The base address of an array is the address (memory location) of the first array component
  - 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
- Refer to Example 8-6 in the text
  - Functions **`sumArray`** and **`indexLargestElement`**

Integral Data Type and Array Indices

- C++ allows any integral type to be used as an array index
  - Improves code readability
- The following code illustrates improved readability:

```
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

- Example 1

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

- Example 2

```
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

- 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 that is being searched
  - Continue the search until item is found or no more data is left in the list

Sorting

- Selection sort: rearrange the list by selecting an element and moving it to its proper position
- Steps for a selection sort:
  - 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

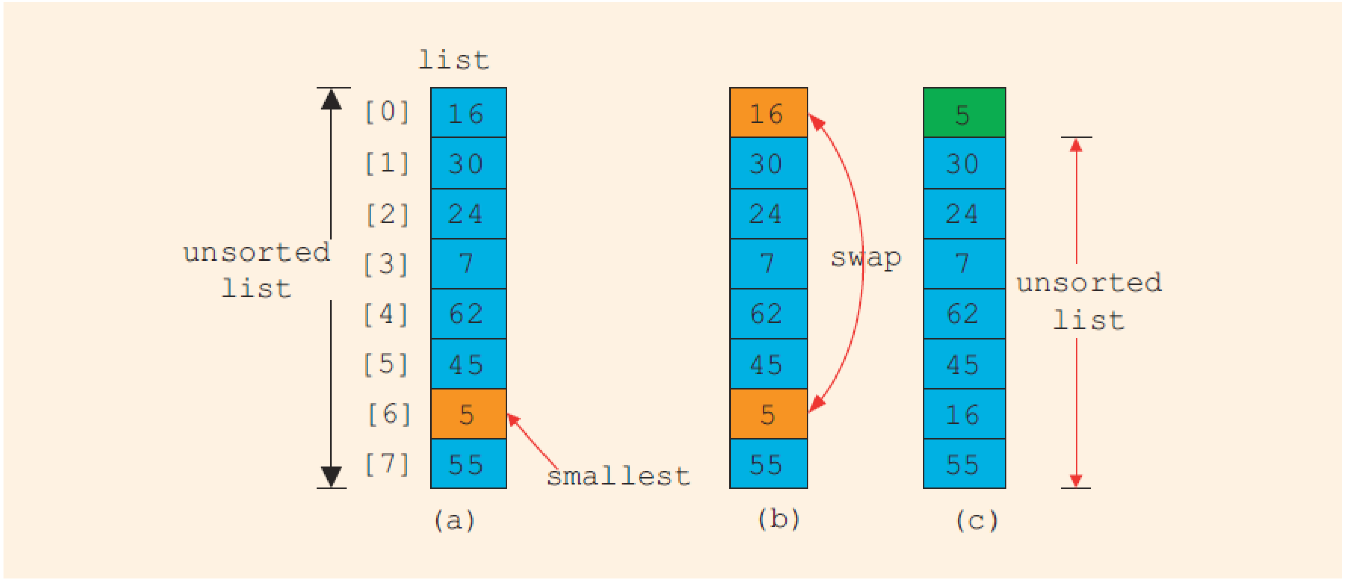

FIGURE 8-10 Elements of `list` during the first iteration

Two- and Multidimensional Arrays

- Two-dimensional array: a collection of a fixed number of components (of the same type) arranged in two dimensions
  - Sometimes called matrices or tables
- Declaration syntax
  - **`intExp1`** and **`intExp2`** are expressions with positive integer values specifying the number of rows and columns in the array

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

Accessing Array Components (1 of 2)

- Syntax to access a component 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 (2 of 2)

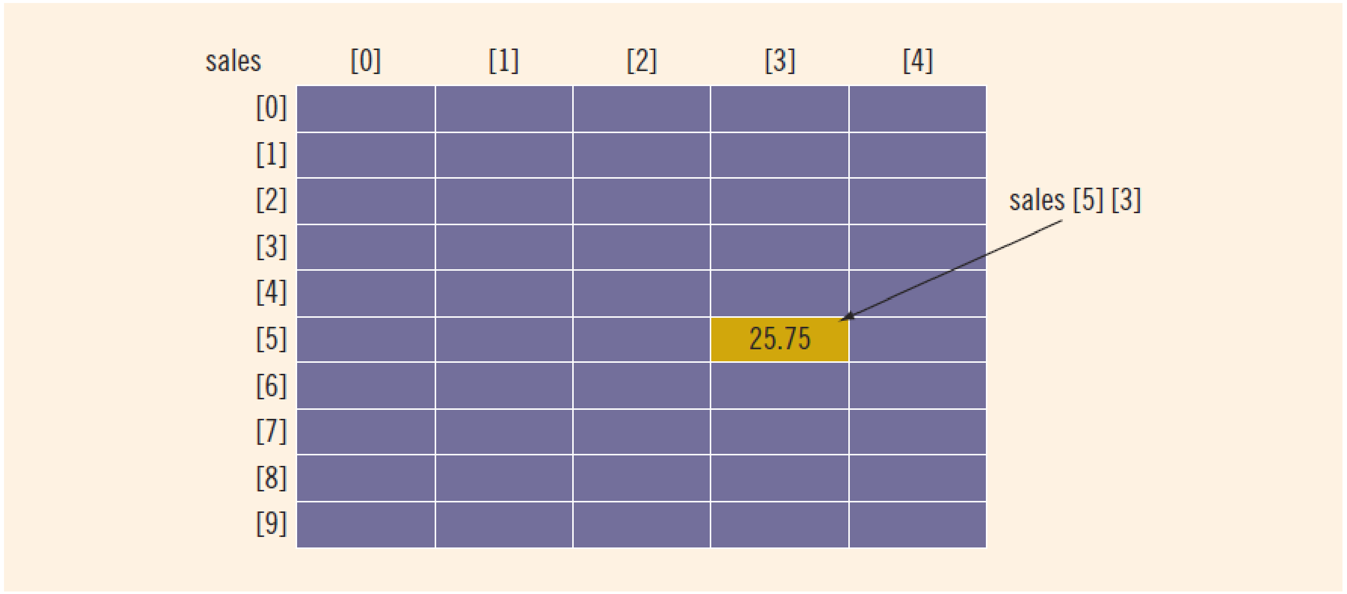

FIGURE 8-14 `sales[5][3]`

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`
- An example of two-dimensional array initialization is shown below:

```
int board[4][3] = {{2, 3, 1},
                   {15, 25, 13},
                   {20, 4, 7},
                   {11, 18, 14}};
```

Initialization

- An example initializing row number **4** (fifth row) to **0**:

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

- An example initializing 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

- An example of adding input to row number **4** (fifth row):

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

- An example of adding input to each component of matrix:

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

Multidimensional Arrays

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

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

- Code to access a component

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