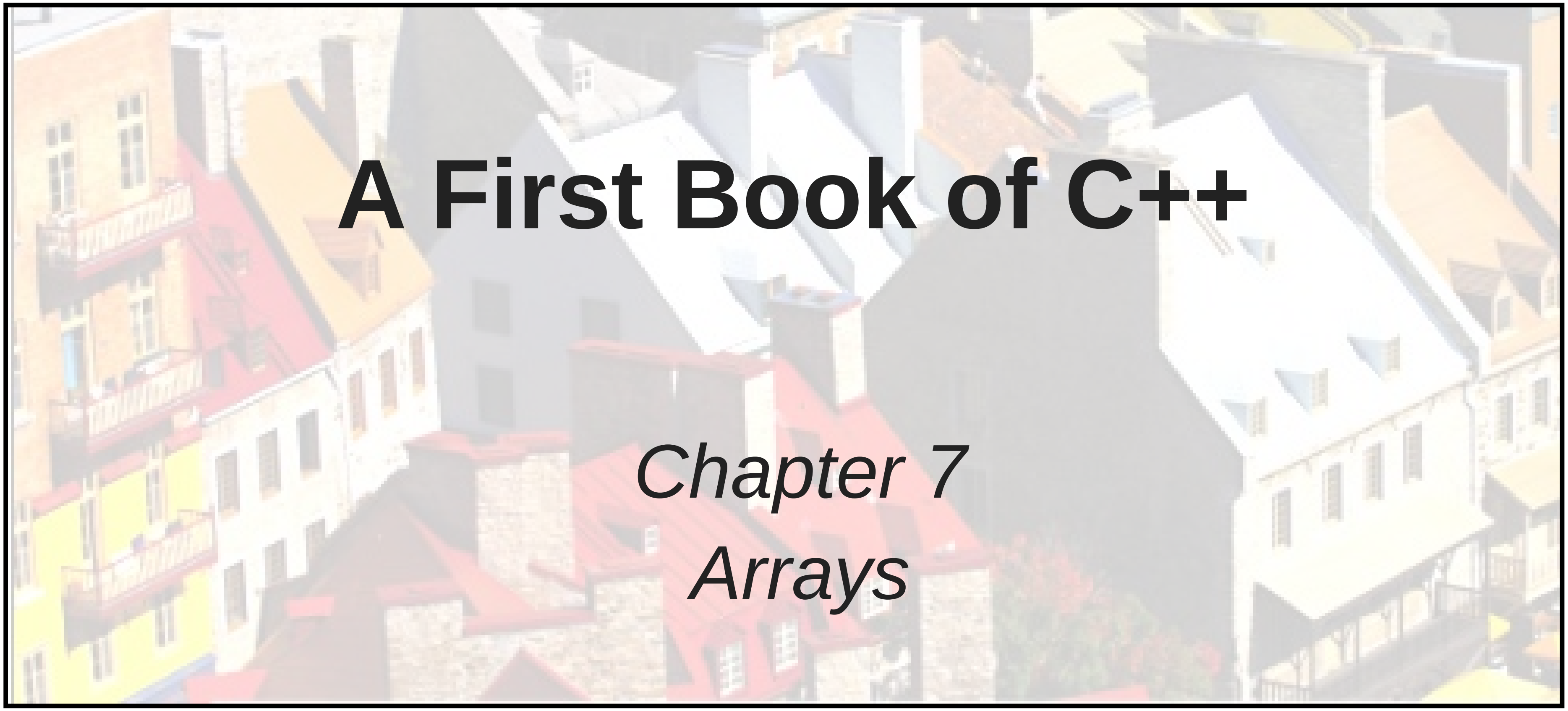


A First Book of C++

Chapter 7
Arrays

Objectives

- In this chapter, you will learn about:
  - One-Dimensional Arrays
  - Array Initialization
  - Arrays as Arguments
  - Two-Dimensional Arrays
  - Common Programming Errors
  - Searching and Sorting Methods

One-Dimensional Arrays

- One-dimensional array (single-dimension array or vector): a list of related values
  - All items in list have same data type
  - All list members stored using single group name
- Example: a list of grades

```
    98, 87, 92, 79, 85
```

  - All grades are integers and must be declared
    - Can be declared as single unit under a common name (the array name)

One-Dimensional Arrays (cont'd.)

- Array declaration statement provides:
  - The array (list) name
  - The data type of array items
  - The number of items in array
- Syntax

```
dataType arrayName[numberOfItems]
```

  - Common programming practice requires defining number of array items as a constant before declaring the array

One-Dimensional Arrays (cont'd.)

- Examples of array declaration statements:

```
const int NUMELS = 5; // define a constant
     // for the number of
          // items
int amts[NUMELS];   // declare the array

const int NUMELS = 4;
char code[NUMELS];

const int SIZE = 100;
double amount[SIZE];
```

One-Dimensional Arrays (cont'd.)

- Each array allocates sufficient memory to hold the number of data items given in declaration
- Array element (component): an item of the array
- Individual array elements stored sequentially
  - A key feature of arrays that provides a simple mechanism for easily locating single elements

One-Dimensional Arrays (cont'd.)

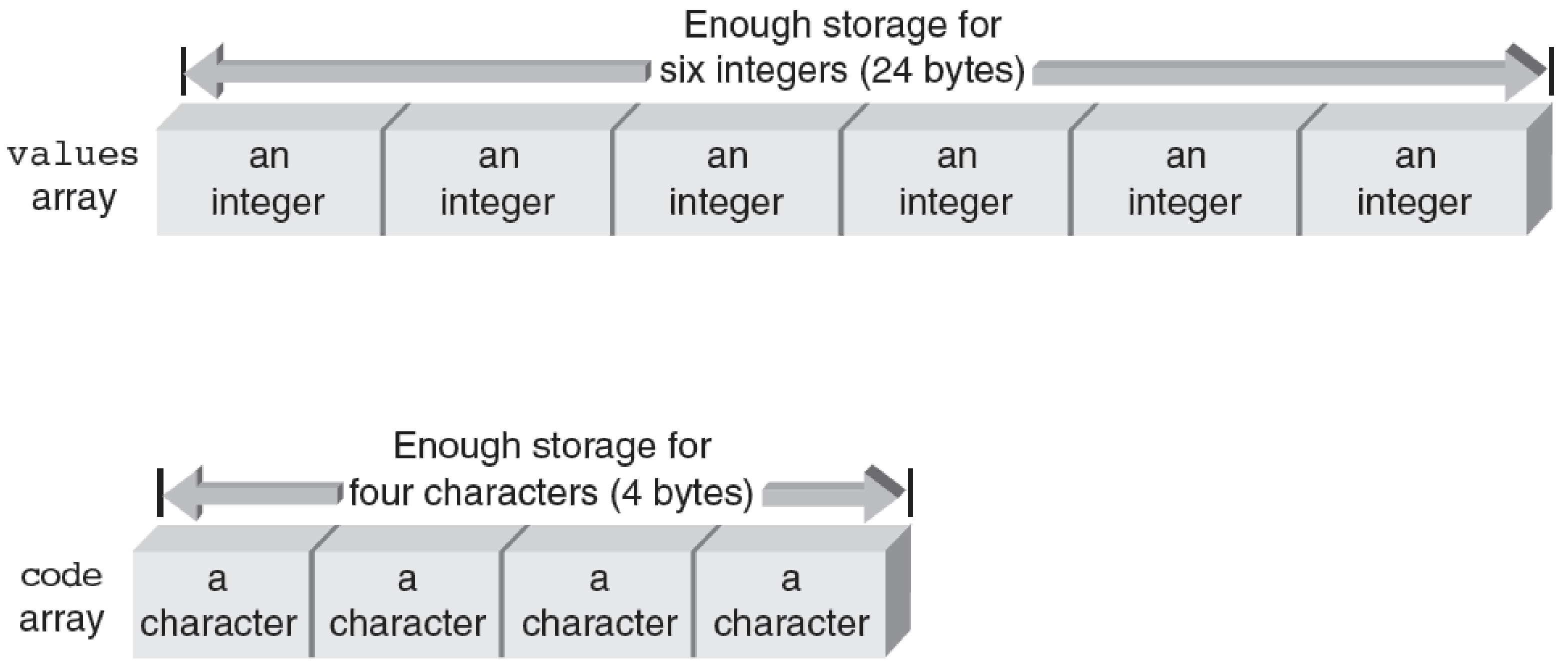


Figure 7.1 The `values` and `code` arrays in memory

One-Dimensional Arrays (cont'd.)

- **Index (subscript** value**):** position of individual element in an array
- Accessing of array elements: done by giving array name and element’s index
  - `grade[0]` refers to first grade stored in `grade` array
- Subscripted variables can be used anywhere that scalar variables are valid:

```
grade[0] = 95.75;
grade[1] = grade[0] - 11.0;
```

One-Dimensional Arrays (cont'd.)

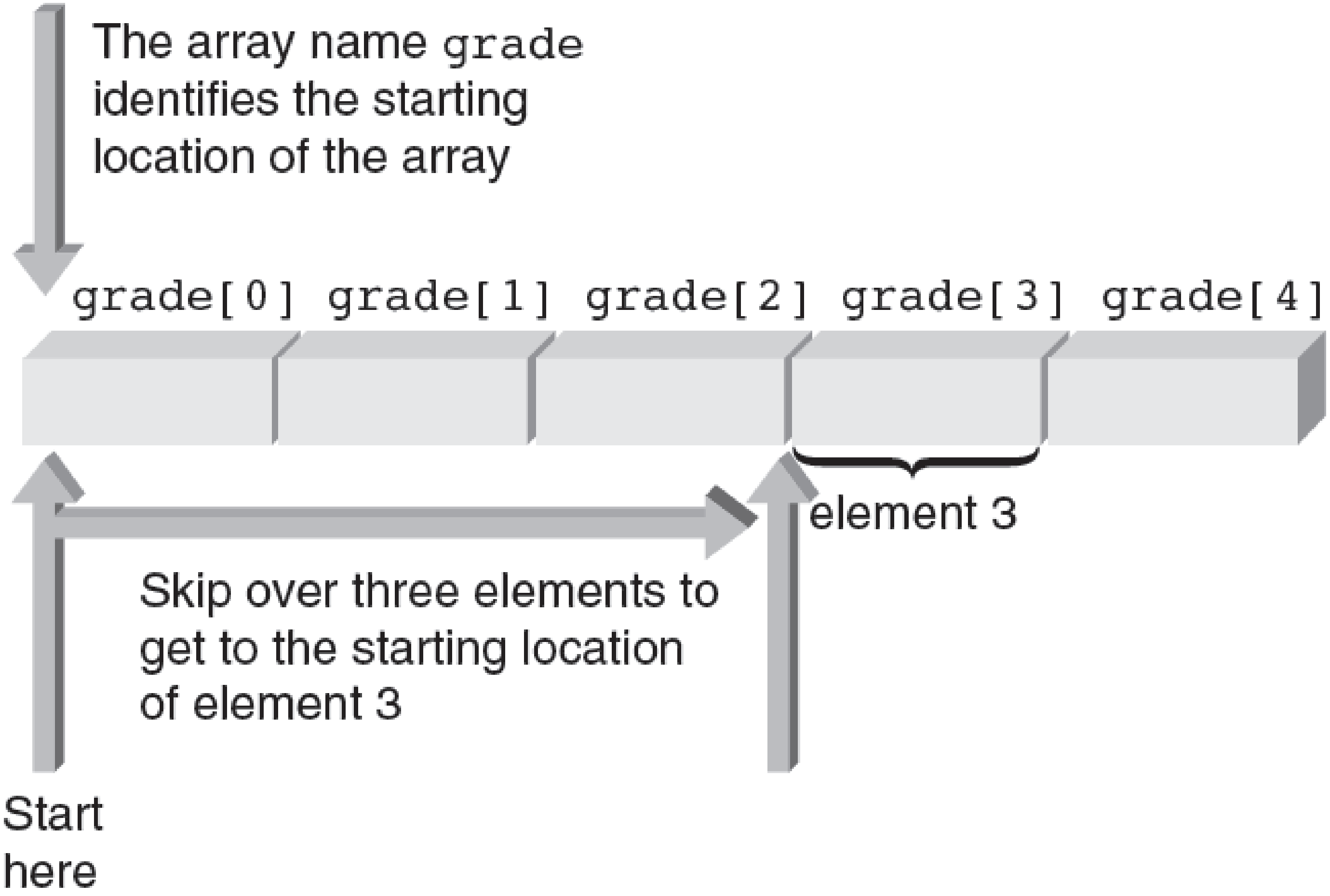

The array name grade identifies the starting location of the array
grade[0]
grade[1]
grade[2]
grade[3]
grade[4]
element 3
Skip over three elements to get to the starting location of element 3
Start here

Figure 7.3 Accessing an array element—element 3

One-Dimensional Arrays (cont'd.)

- **Subscripts**: do not have to be integers
  - Any expression that evaluates to an integer may be used as a subscript
  - Subscript must be within the declared range
- Examples of valid subscripted variables (assumes `i` and `j` are `int` variables):

```
grade[i]
grade[2*i]
grade[j-i]
```

Input and Output of Array Values

- Individual array elements can be assigned values interactively using a `cin` stream object

```
cin >> grade[0];
cin >> grade[1] >> grade[2] >> grade[3];
cin >> grade[4] >> prices[6];
```

- Instead, a `for` loop can be used

```
const int NUMELS = 5;
for (int i = 0; i < NUMELS; i++)
{
 cout << "Enter a grade: ";
 cin >> grade[i];
}
```

Input and Output of Array Values (cont'd.)

- **Bounds checking**: C++ does not check if value of an index is within declared bounds
- If an out-of-bounds index is used, C++ will not provide notification
  - Program will attempt to access out-of-bounds element, causing program error or crash
  - Using symbolic constants helps avoid this problem

Input and Output of Array Values (cont'd.)

- Using `cout` to display subscripted variables:
  - Example 1

    ```
    cout << prices[5];
    ```

  - Example 2

    ```
    cout << "The value of element " << i << " is
     " << grade[i];
    ```

  - Example 3

    ```
    const int NUMELS = 20;
    for (int k = 5; k < NUMELS; k++)
      cout << k << " " << amount[k];
    ```

Input and Output of Array Values (cont'd.)

- Program example of array I/O (Program 7.1):

```
#include <iostream>
using namespace std;
int main()
{
   const int NUMELS = 5;
   int i, grade[NUMELS];
   for (i = 0; i < NUMELS; i++) // Enter the grades
   {
     cout << "Enter a grade: ";
     cin >> grade[i];
   }
   cout << endl;
   for (i = 0; i < NUMELS; i++) // Print the grades
     cout << "grade [" << i << "] is " << grade[i] <<
     endl;
   return 0;
}
```

Input and Output of Array Values (cont'd.)

- Sample run using Program 7.1:

```
Enter a grade: 85
Enter a grade: 90
Enter a grade: 78
Enter a grade: 75
Enter a grade: 92

grade[0] is 85
grade[1] is 90
grade[2] is 78
grade[3] is 75
grade[4] is 92
```

Array Initialization

- Array elements can be initialized within declaration statements
  - Initializing elements must be included in braces
  - Example:

```
const int NUMGALS = 20;
int gallons[NUMGALS] =
{19, 16, 14, 19, 20, 18, // initializing values
 12, 10, 22, 15, 18, 17, // can extend across
 16, 14, 23, 19, 15, 18, // multiple lines
 21, 5};
```

Array Initialization (cont'd.)

- Size of array may be omitted when initializing values are included in declaration statement
- Example: the following are equivalent

```
    const int NUMCODES = 6;
    char code[6] = {'s', 'a', 'm', 'p', 'l', 'e'};


    char code[ ] = {'s', 'a', 'm', 'p', 'l', 'e'};
```

- Both declarations set aside six character locations for an array named `code`

Array Initialization (cont'd.)

- Simplified method for initializing character arrays

```
char codes[ ] = “sample”;
                    //no braces or commas
```

- This statement uses the string “`sample`” to initialize the `code` array
  - The array is comprised of seven characters
  - The first six characters are the letters:

    `s`, `a`, `m`, `p`, `l`, `e`
  - The last character (the escape sequence `\0`) is called the **null character**

Array Initialization (cont'd.)

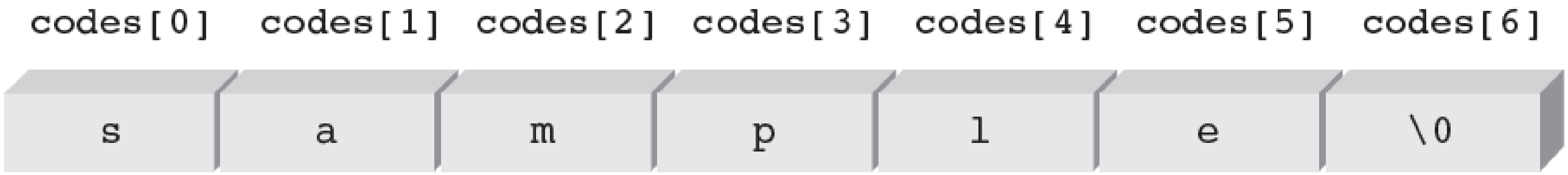


Figure 7.4 Terminating a string with the `\0` character

Arrays as Arguments

- Array elements are passed to a called function in same manner as individual scalar variables
  - Example:

    ```
    findMax(grades[2], grades[6]);
    ```
- Passing a complete array to a function provides access to the actual array, not a copy
  - Making copies of large arrays is wasteful of storage

Arrays as Arguments (cont'd.)

- Examples of function calls that pass arrays:

```
int nums[5];          // an array of five integers
char keys[256];       // an array of 256 characters
double units[500], grades[500];// two arrays of
                               // 500 doubles
```

- The following function calls can then be made:

```
      findMax(nums);
      findCharacter(keys);
      calcTotal(nums, units, grades);
```

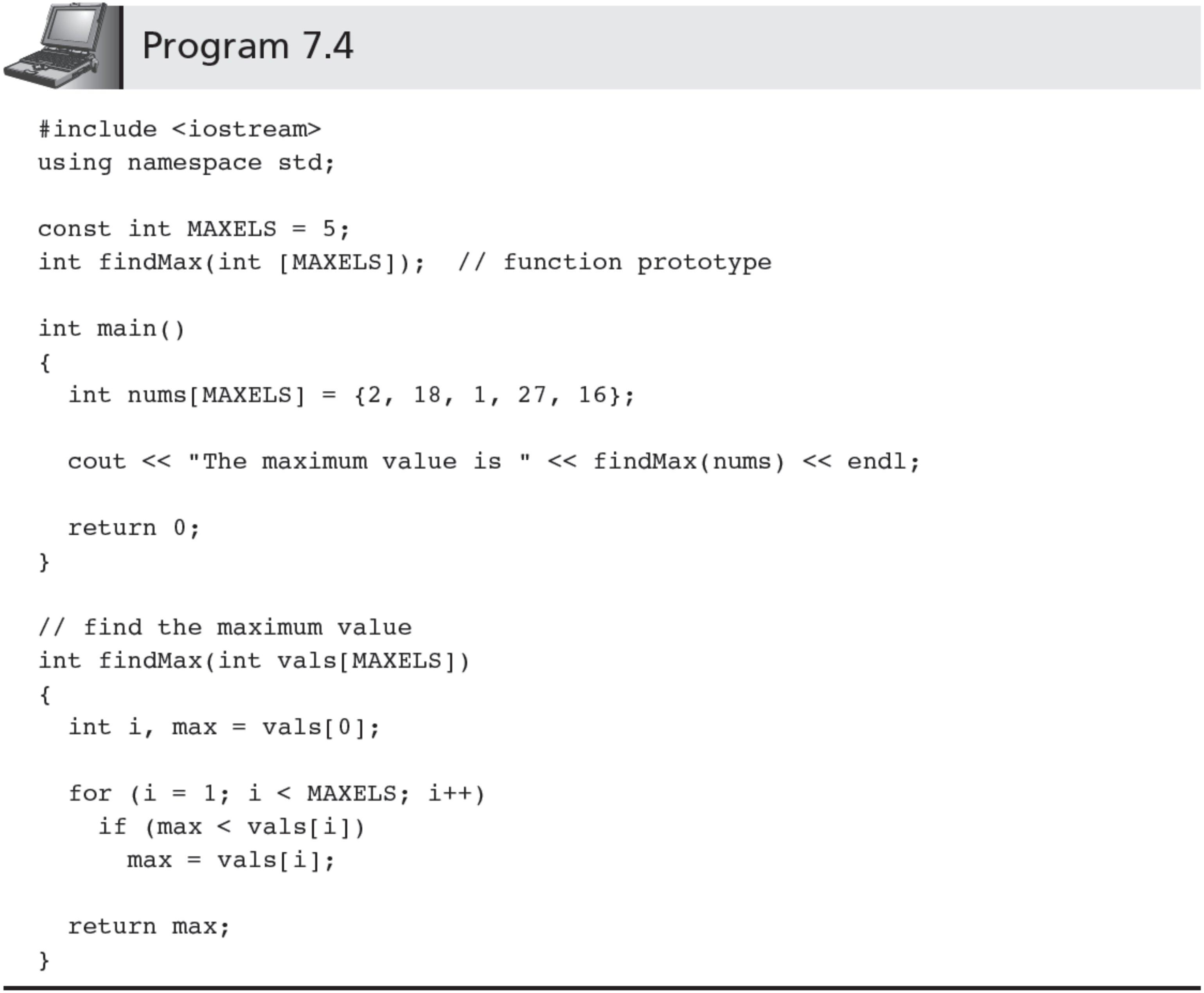
Arrays as Arguments (cont'd.)

- Suitable receiving side function header lines:

```
int findMax(int vals[5])
char findCharacter(char inKeys[256])
void calcTotal(int arr1[5],
               double arr2[500],
 double arr3[500])
```

Arrays as Arguments (cont'd.)

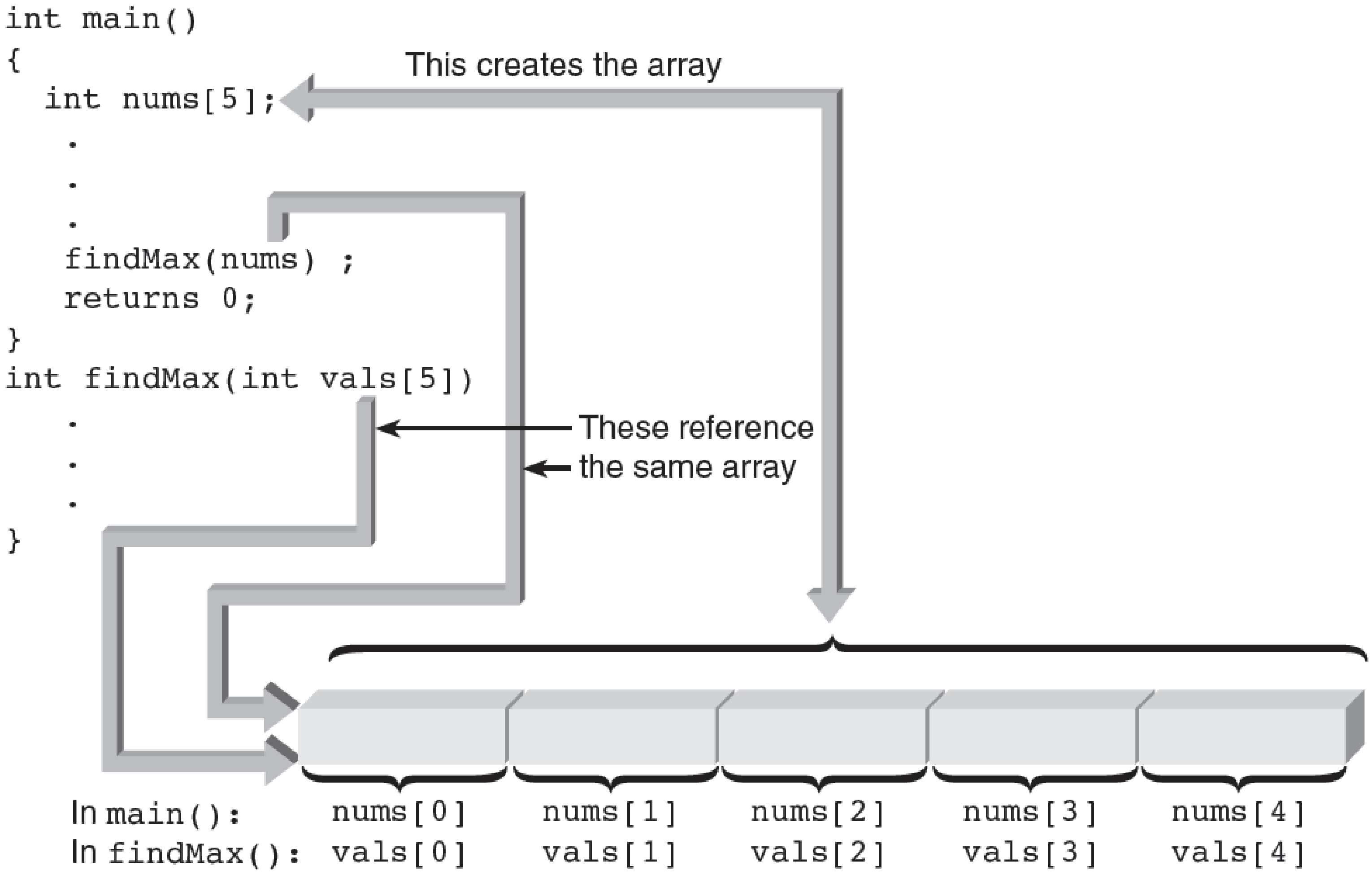
- Example of passing arrays as arguments (Program 7.4):
  - Constant `MAXELS` is declared globally
  - Prototype for *`findMax()`* uses constant *`MAXELS`* to declare that *`findMax()`* expects an array of five integers as an argument
  - As shown in Figure 7.5, only one array is created in Program 7.4
    - In `main()`, the array is known as `nums`
    - In `findMax()`, it is known as `vals`



Program 7.4

```
#include <iostream>
using namespace std;

const int MAXELS = 5;
int findMax(int [MAXELS]);  // function prototype

int main()
{
  int nums[MAXELS] = {2, 18, 1, 27, 16};

  cout << "The maximum value is " << findMax(nums) << endl;

  return 0;
}

// find the maximum value
int findMax(int vals[MAXELS])
{
  int i, max = vals[0];

  for (i = 1; i < MAXELS; i++)
    if (max < vals[i])
      max = vals[i];

  return max;
}
```

Arrays as Arguments (cont'd.)



```
int main()
{
  int nums[5];
    .
    .
    .
    findMax(nums) ;
    returns 0;
}
int findMax(int vals[5])
    .
    .
    .
}
```

This creates the array
These reference the same array
In main(): nums[0] nums[1] nums[2] nums[3] nums[4]
In findMax(): vals[0] vals[1] vals[2] vals[3] vals[4]

Figure 7.5 Only one array is created

Two-Dimensional Arrays

- Two-dimensional array (table): consists of both rows and columns of elements
- Example: two-dimensional array of integers

8	16	9	52
3	15	27	6
14	25	2	10

- Array declaration: names the array `val` and reserves storage for it

```
int val[3][4];
```

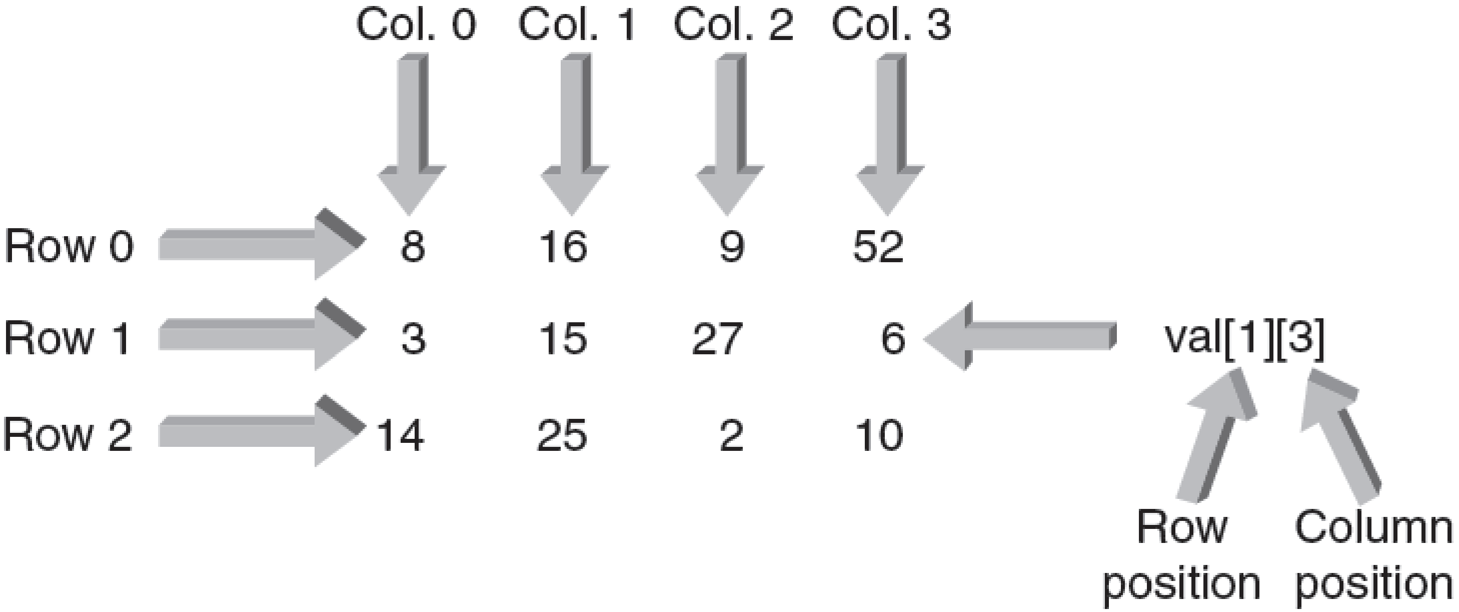
Two-Dimensional Arrays (cont'd.)

- Locating array elements (Figure 7.7)
  - `val[1][3]` uniquely identifies element in row 1, column 3
- Examples using elements of `val` array:

```
price = val[2][3];
val[0][0] = 62;
newnum = 4 * (val[1][0] - 5);
sumRow = val[0][0] + val[0][1] + val[0][2]
  + val[0][3];
```

  - The last statement adds the elements in row 0 and sum is stored in `sumRow`

Two-Dimensional Arrays (cont'd.)



	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	8	16	9	52
Row 1	3	15	27	6
Row 2	14	25	2	10

Figure 7.7 Each array element is identified by its row and column position

Two-Dimensional Arrays (cont'd.)

- Initialization: can be done within declaration statements (as with single-dimension arrays)
- Example:

```
int val[3][4] = { {8,16,9,52},
                  {3,15,27,6},
                  {14,25,2,10} };
```

  - First set of internal braces contains values for row 0, second set for row 1, and third set for row 2
  - Commas in initialization braces are required; inner braces can be omitted

Two-Dimensional Arrays (cont'd.)

- Processing two-dimensional arrays: nested `for` loops typically used
  - Easy to cycle through each array element
    - A pass through outer loop corresponds to a row
    - A pass through inner loop corresponds to a column
  - Nested `for` loop in Program 7.7 used to multiply each `val` element by 10 and display results
- Output of Program 7.7

```
Display of multiplied elements
   80  160   90  520
   30  150  270   60
  140  250   20  100
```

Two-Dimensional Arrays (cont'd.)

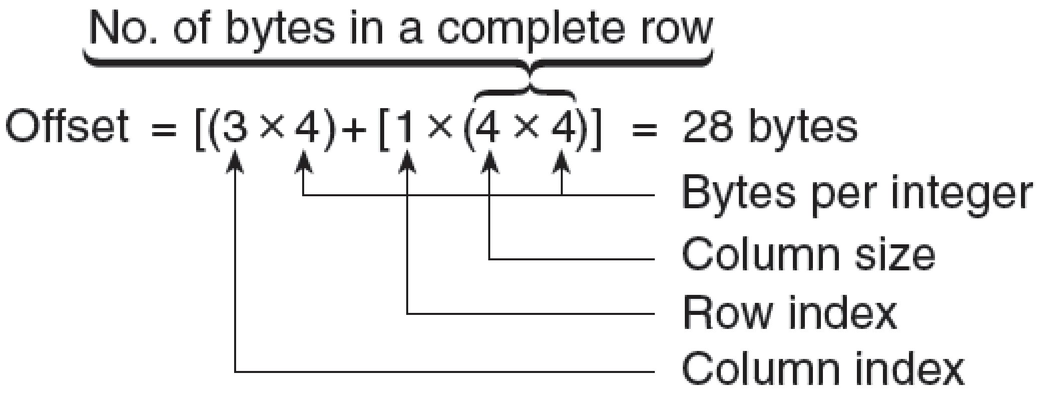
- Prototypes for functions that pass two-dimensional arrays can omit the row size of the array
  - Example (Program 7.8):

    ```
    display (int nums[ ][4]);
    ```

  - Row size is optional, but column size is required
    - The element `val[1][3]` is located 28 bytes from the start of the array (assuming 4 bytes for an int)

Two-Dimensional Arrays (cont'd.)

- Determining offset of an array
  - Computer uses row index, column index, and column size to determine offset



$$\text{Offset} = [(3 \times 4) + [1 \times \overbrace{(4 \times 4)}^{\text{No. of bytes in a complete row}}] = 28 \text{ bytes}$$

Two-Dimensional Arrays (cont'd.)

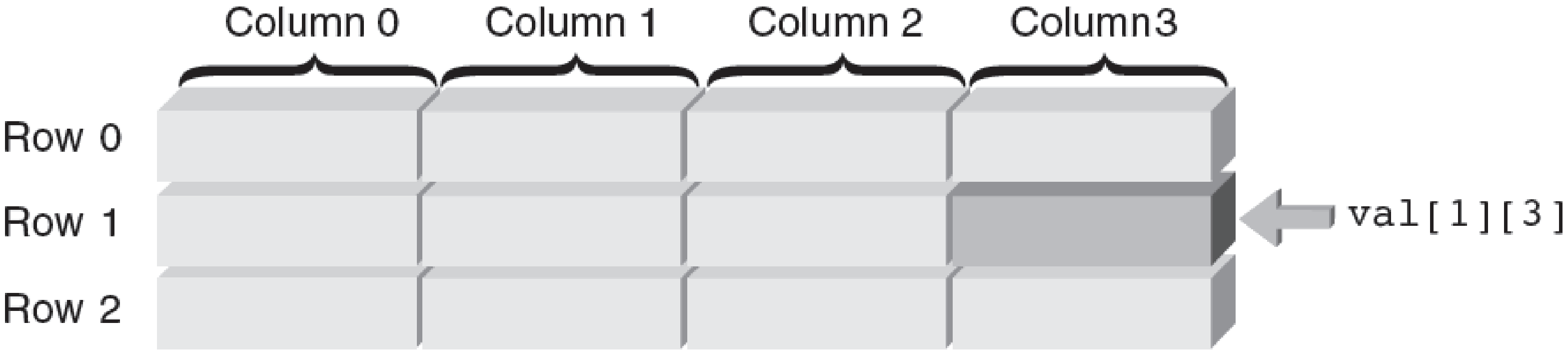


Figure 7.9 Storage of the `val` array

Larger Dimensional Arrays

- Arrays with more than two dimensions allowed in C++ but not commonly used
- Example: `int response[4][10][6]`
  - First element is `response[0][0][0]`
  - Last element is `response[3][9][5]`
- A three-dimensional array can be viewed as a book of data tables (Figure 7.10)
  - First subscript (rank) is page number of table
  - Second subscript is row in table
  - Third subscript is desired column

Larger Dimensional Arrays (cont'd.)

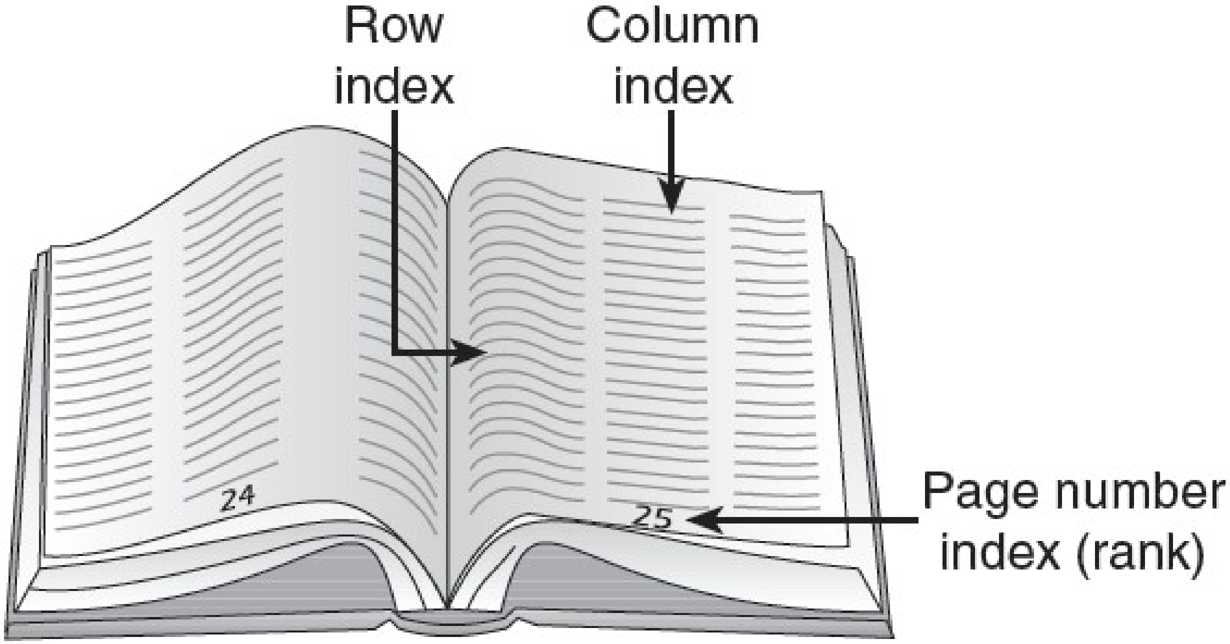


Figure 7.10 Representation of a three-dimensional array

Common Programming Errors

- Forgetting to declare an array
  - Results in a compiler error message equivalent to “invalid indirection” each time a subscripted variable is encountered within a program
- Using a subscript that references a nonexistent array element
  - For example, declaring array to be of size 20 and using a subscript value of 25
  - Not detected by most C++ compilers and will probably cause a runtime error

Common Programming Errors (cont'd.)

- Not using a large enough counter value in a `for` loop counter to cycle through all array elements
- Forgetting to initialize array elements
  - Don’t assume compiler does this

Summary

- One-dimensional array: a data structure that stores a list of values of same data type
  - Must specify data type and array size
  - Example:

    `int num[100];` creates an array of 100 integers
- Array elements are stored in contiguous locations in memory and referenced using the array name and a subscript
  - Example: `num[22]`

Summary (cont'd.)

- Two-dimensional array is declared by listing both a row and column size with data type and name of array
- Arrays may be initialized when they are declared
  - – For two-dimensional arrays, you list the initial values, in a row-by-row manner, within braces and separating them with commas
- Arrays are passed to a function by passing name of array as an argument

Chapter Supplement: Searching and Sorting Methods

- Most programmers encounter the need to both sort and search a list of data items at some time in their programming careers

Search Algorithms

- Linear (sequential) search
  - Each item in the list is examined in the order in which it occurs until the desired item is found or the end of the list is reached
  - List doesn’t have to be in sorted order to perform the search
- Binary search
  - Starting with an ordered list, the desired item is first compared with the element in the middle of the list
  - If item is not found, you continue the search on either the first or second half of the list