Arrays

CS 1: Problem Solving & Program Design Using C++

Objectives

- Start simple with one-dimensional arrays
- Then, really get started with array initialization
- Look at arrays as arguments
- Get a little more fancy with two-dimensional arrays
- Look at more common programming errors

One-Dimensional Arrays

- ONE-DIMENSION 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 (2)

- 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

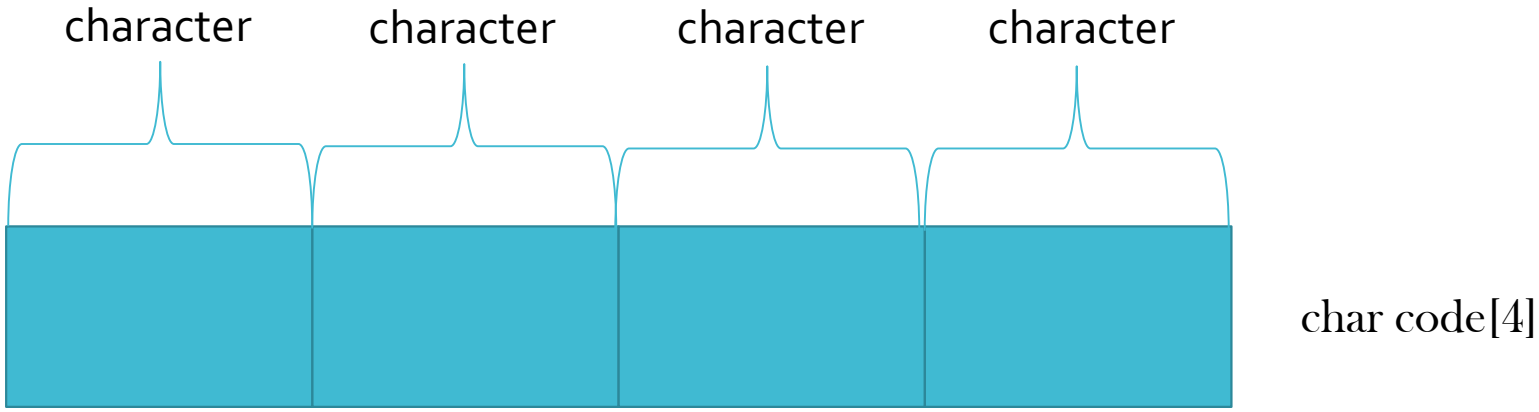
Example of One-Dimensional Array Declaration Statements

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

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

const int NUMELS = 6;
double prices[NUMELS];
```

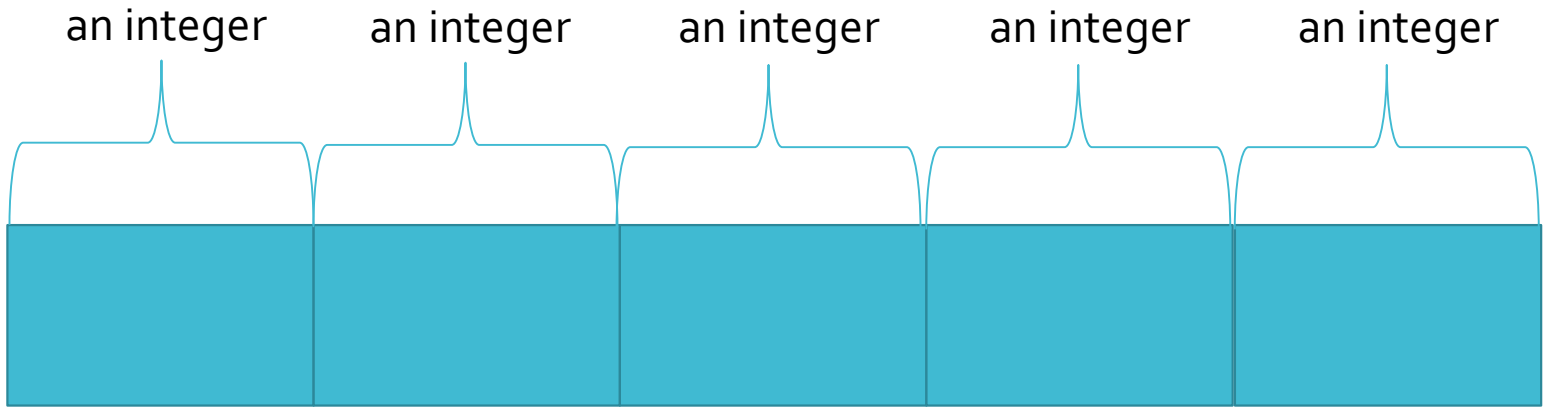
About One-Dimensional Array Declaration Statements

- 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 Pictorially



Enough array storage for four characters (four bytes)



Enough array storage for five integers (twenty bytes)

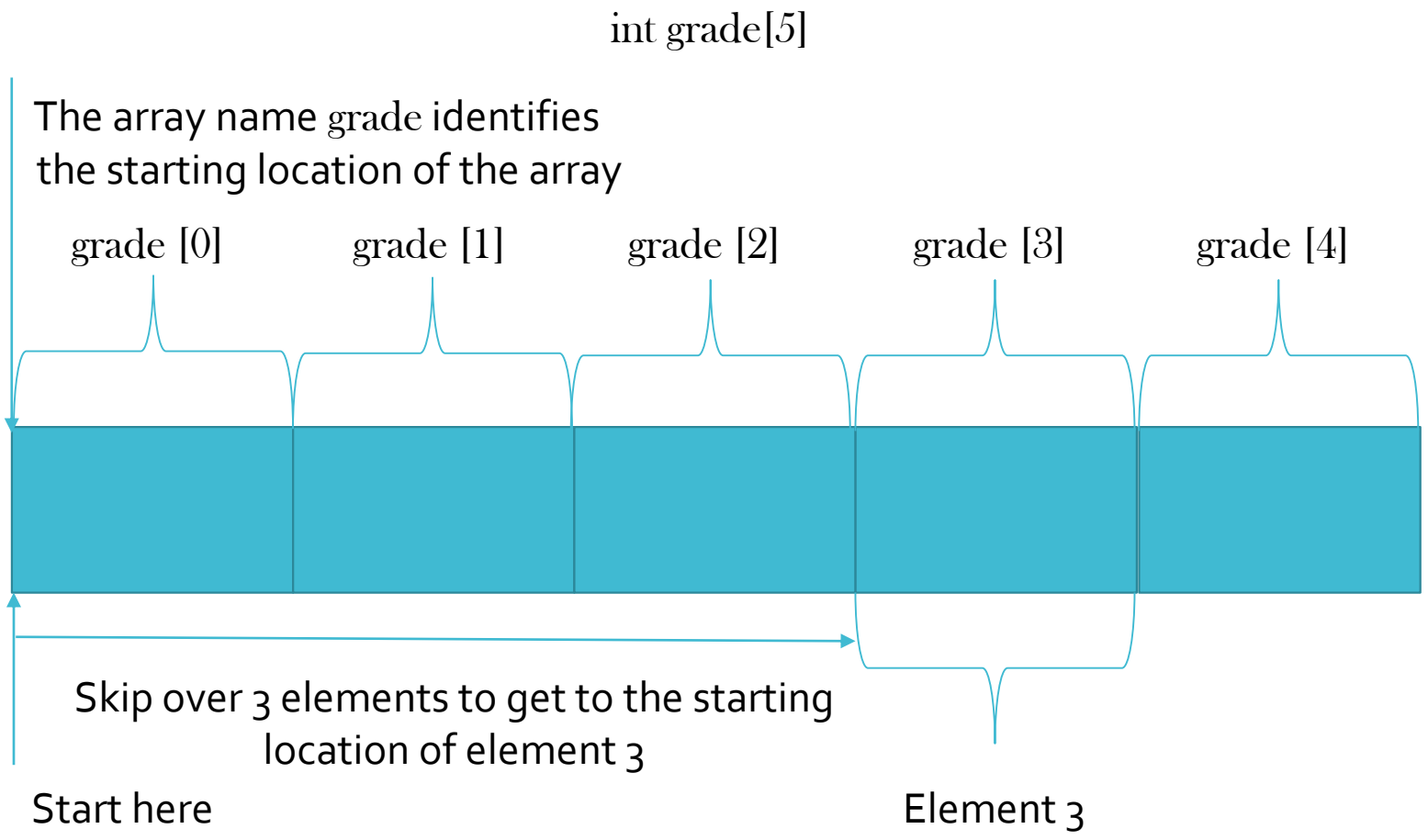
int grade[5]

Indexes and Accessing Array Elements

- 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;
```

Pictorially Accessing an Individual Array Element



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];
}
```

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

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];
}
```

Example of Array Input/Output (I/O)

```
#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];
        }
```

Example of Array Input/Output (I/O) (2)

```
        cout << endl;
        for (i = 0; i < NUMELS; i++) // Print the grades
        {
                cout << "grade [" << i << "] is " << grade[i] << endl;
        }
        return 0;
}
```

Sample Run of Example of Array Input/Output (I/O)

```
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, // may extend across
         16, 14, 23, 19, 15, 18, // multiple lines
         21, 5};
```

Array Initialization (2)

- 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 6 character locations for an array named code

Array Initialization (3)

- Simplified method for initializing character arrays

```
char code[ ] = "sample"; //no braces or commas
```

- This statement uses the string "sample" to initialize the code array
- The array is comprised of 7 characters
- The first 6 characters are the letters:

  s, a, m, p, l, e

- The last character (the escape sequence \0) is called the Null character

Visual Example of "sample"

code[0]	code[1]	code[2]	code[3]	code[4]	code[5]	code[6]
s	a	m	p	l	e	\0

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 (2)

- 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 (3)

- 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 Example

```
#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;
}
```

Arrays as Arguments Example (2)

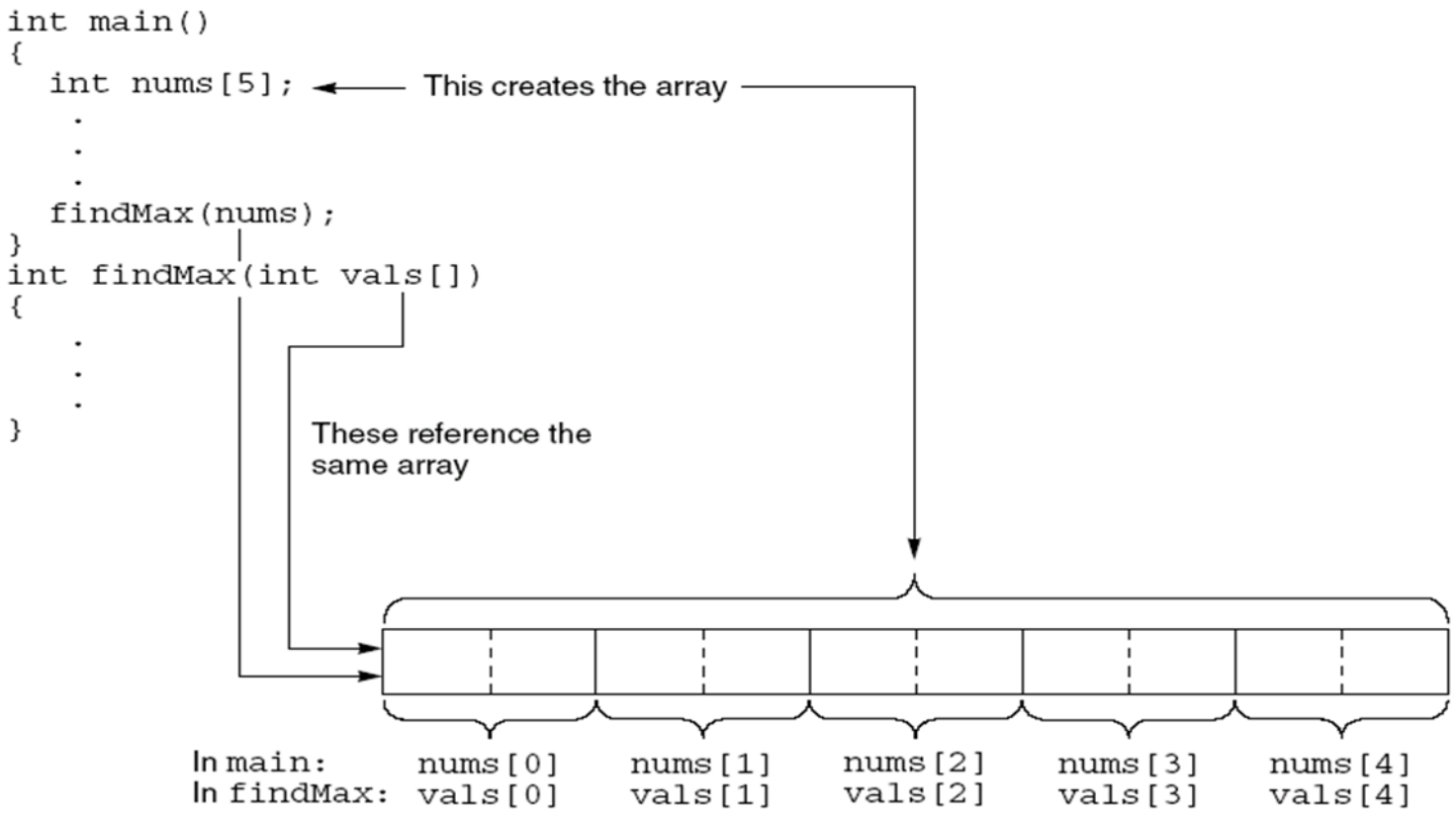
```
// find the maximum value
int findMax(int vals[MAXELS])
{
        int i, max = vals[0];
        for (i = 1; i < MAXELS; i++)
        {
                if (max < vals[i])
                {
                        max = vals[i];
                }
        }
        return max;
}
```

About the Arrays as Arguments Example

- Constant MAXELS is declared globally
- Prototype for findMax() uses constant MAXELS to declare that findMax() expects an array of five integers as an argument
- Only one array is created
  - In main(), the array is known as nums
  - In findMax(), it is known as vals

Visual of Arrays as Arguments Example

FIGURE 8.7 *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 (2)

- Locating array elements
  - 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

Visual of Two-Dimensional Array

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

val[1][3]

Two-Dimensional Array 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

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

Prototypes for Functions that Pass Two-Dimensional Arrays

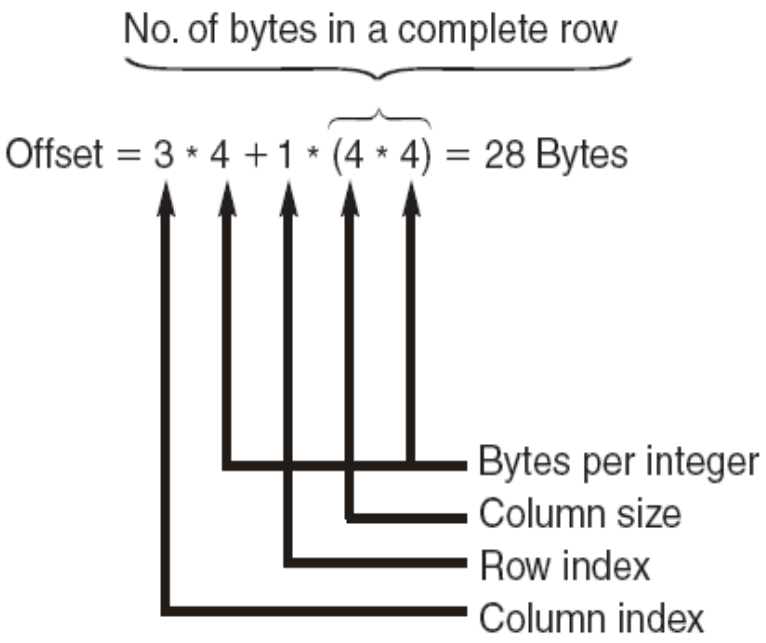
- Can omit the row size of the array
- Example:

```
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)

Determining Offset of An Array

- Computer uses row index, column index and column size to determine offset

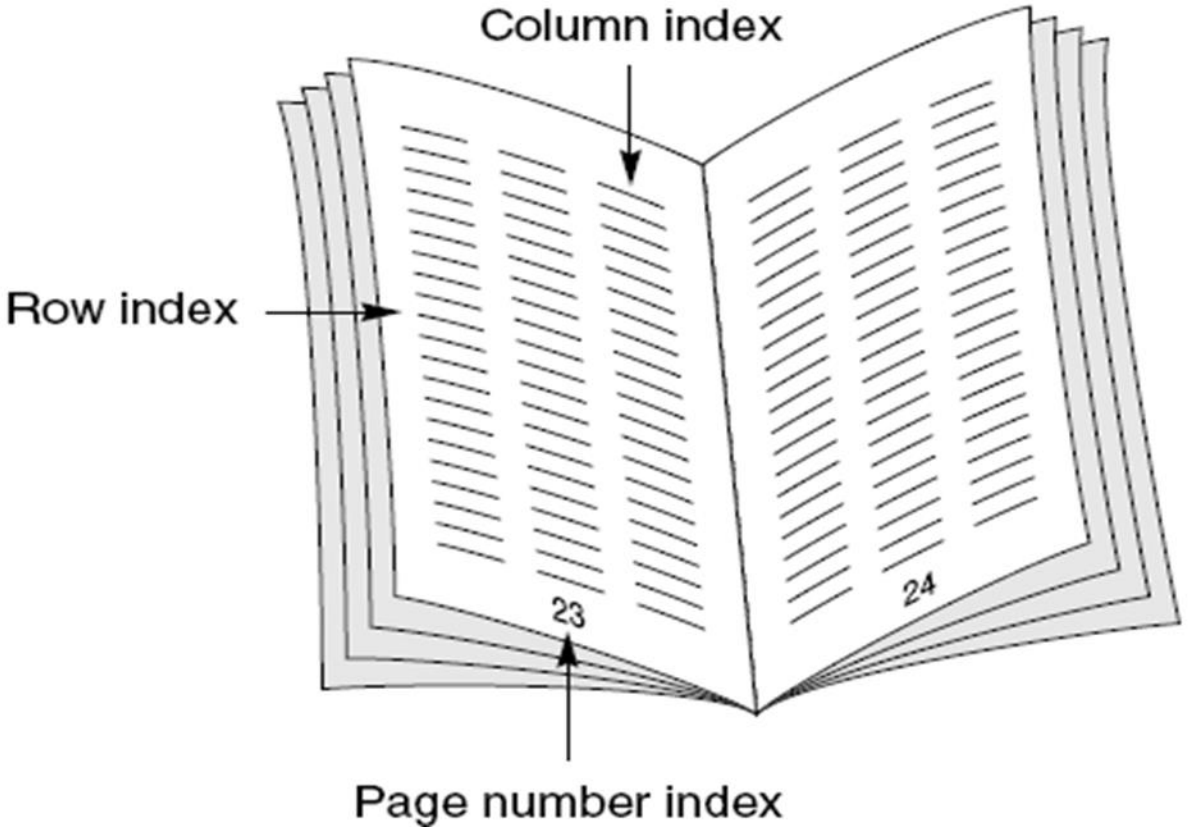


Large-Dimension Arrays

- 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
  - First subscript (rank) is page number of table
  - Second subscript is row in table
  - Third subscript is desired column

Large-Dimension Arrays (2)

FIGURE 8.12 *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 (2)

- 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

- Single-dimensional array: a data structure that stores a list of values of same data type
  - Must specify data type and array size
  - 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
  - For example, num[22]

Summary (2)

- 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