# 7.7 Sorting and Searching arrays (cont.)

- Lines 29 and 34 demonstrate use binary_search to determine whether a value is in the array.
- The sequence of values must be sorted in ascending order first—binary_search does *not* verify this for you.
- The function's first two arguments represent the range of elements to search and the third is the *search key*—the value to locate in the array.
- The function returns a bool indicating whether the value was found.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
1  // Fig. 7.18: fig07_18.cpp
2  // Sorting and searching arrays.
3  #include <iostream>
4  #include <iomanip>
5  #include <array>
6  #include <string>
7  #include <algorithm> // contains sort and binary_search
8  using namespace std;
9
10 int main()
11 {
12    const size_t arraySize = 7; // size of array colors
13    array< string, arraySize > colors = { "red", "orange", "yellow",
14       "green", "blue", "indigo", "violet" };
15
16    // output original array
17    cout << "Unsorted array:\n";
18    for ( string color : colors )
19       cout << color << " ";
20
21    sort( colors.begin(), colors.end() ); // sort contents of colors
22
```

**Fig. 7.18** | Sorting and searching arrays. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
23     // output sorted array
24     cout << "\nSorted array:\n";
25     for ( string item : colors )
26        cout << item << " ";
27
28     // search for "indigo" in colors
29     bool found = binary_search( colors.begin(), colors.end(), "indigo" );
30     cout << "\n\n\"indigo\" " << ( found ? "was" : "was not" )
31        << " found in colors" << endl;
32
33     // search for "cyan" in colors
34     found = binary_search( colors.begin(), colors.end(), "cyan" );
35     cout << "\"cyan\" " << ( found ? "was" : "was not" )
36        << " found in colors" << endl;
37  } // end main
```

```
Unsorted array:
red orange yellow green blue indigo violet
Sorted array:
blue green indigo orange red violet yellow

"indigo" was found in colors
"cyan" was not found in colors
```

Fig. 7.18 | Sorting and searching arrays. (Part 2 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

# 7.8 Multidimensional Arrays

- You can use `array`s with two dimensions (i.e., subscripts) to represent tables of values consisting of information arranged in rows and columns.
- To identify a particular table element, we must specify two subscripts—by convention, the first identifies the element's *row* and the second identifies the element's *column*.
- Often called two-dimensional arrays or 2-D arrays.
- Arrays with two or more dimensions are known as multidimensional arrays.
- Figure 7.20 illustrates a two-dimensional array, `a`.
  - The array contains three rows and four columns, so it's said to be a 3-by-4 array.
  - In general, an array with *m rows and n columns is called an m*-by-*n* array*.*

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

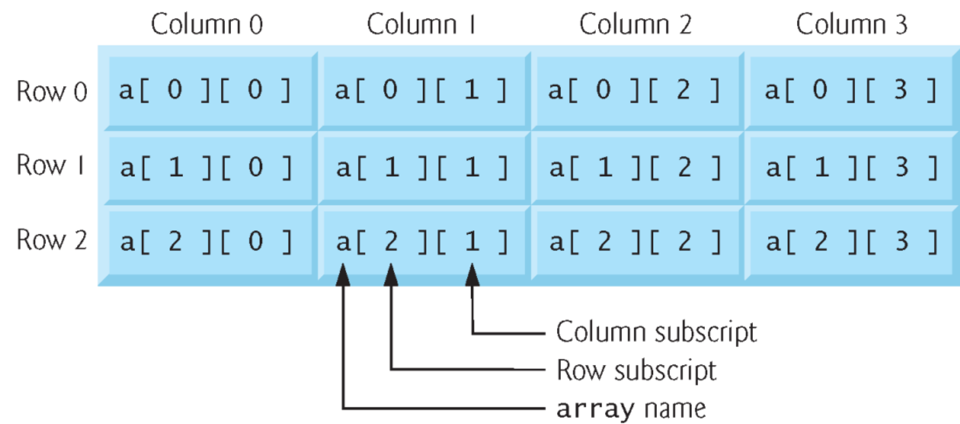


**Fig. 7.19** | Two-dimensional `array` with three rows and four columns.

©1992-2014 by Pearson Education, Inc. All
Rights Reserved.

## Common Programming Error 7.5

Referencing a two-dimensional `array` element `a[x][y]` incorrectly as `a[x, y]` is an error. Actually, `a[x, y]` is treated as `a[y]`, because C++ evaluates the expression `x, y` (containing a comma operator) simply as `y` (the last of the comma-separated expressions).

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

# 7.8 Multidimensional arrays (cont.)

- Figure 7.20 demonstrates initializing two-dimensional `array`s in declarations.
- In each `array`, the type of its elements is specified as

```
array< int, columns >
```

- indicating that each `array` contains as its elements three-element `array`s of `int` values—the constant `columns` has the value 3.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
1   // Fig. 7.20: fig07_20.cpp
2   // Initializing multidimensional arrays.
3   #include <iostream>
4   #include <array>
5   using namespace std;
6
7   const size_t rows = 2;
8   const size_t columns = 3;
9   void printArray( const array< array< int, columns >, rows> & );
10
11  int main()
12  {
13     array< array< int, columns >, rows > array1 = { 1, 2, 3, 4, 5, 6 };
14     array< array< int, columns >, rows > array2 = { 1, 2, 3, 4, 5 };
15
16     cout << "Values in array1 by row are:" << endl;
17     printArray( array1 );
18
19     cout << "\nValues in array2 by row are:" << endl;
20     printArray( array2 );
21  } // end main
22
```

**Fig. 7.20** | Initializing multidimensional arrays. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
23  // output array with two rows and three columns
24  void printArray( const array< array< int, columns >, rows> & a )
25  {
26     // loop through array's rows
27     for ( auto const &row : a )
28     {
29        // loop through columns of current row
30        for ( auto const &element : row )
31           cout << element << ' ';
32
33        cout << endl; // start new line of output
34     } // end outer for
35  } // end function printArray
```

```
Values in array1 by row are:
1 2 3
4 5 6

Values in array2 by row are:
1 2 3
4 5 0
```

**Fig. 7.20** | Initializing multidimensional arrays. (Part 2 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

# 7.8 Multidimensional `array`s (cont.)

## *Nested Range-Based `for` Statements*

- To process the elements of a two-dimensional `array`, we use a nested loop in which the *outer* loop iterates through the *rows* and the *inner* loop iterates through the *columns* of a given row.
- The C++11 auto keyword tells the compiler to infer (determine) a variable's data type based on the variable's initializer value.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

# 7.8 Multidimensional arrays (cont.)

## *Nested Counter-Controlled `for` Statements*

- We could have implemented the nested loop with counter-controlled repetition as follows:

```
for ( size_t row = 0; row < a.size(); ++row )
{
   for ( size_t column = 0; column < a[ row ].size(); ++column )
      cout << a[ row ][ column ] << ' ';
   cout << endl;
} // end outer for
```

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

# 7.9 Case Study: Class `GradeBook` Using a Two-Dimensional `array`

- In most semesters, students take several exams.
- Professors are likely to want to analyze grades across the entire semester, both for a single student and for the class as a whole.
- Figure 7.21 shows the output that summarizes 10 students grades on three exams.
- We store the grades as a two-dimensional `array` in an object of the next version of class `GradeBook` Figures 7.22–7.23.
- Each row of the array represents a single student's grades for the entire course, and each column represents all the grades the students earned for one particular exam.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
Welcome to the grade book for
CS101 Introduction to C++ Programming!

The grades are:

            Test 1  Test 2  Test 3  Average
Student  1      87      96      70    84.33
Student  2      68      87      90    81.67
Student  3      94     100      90    94.67
Student  4     100      81      82    87.67
Student  5      83      65      85    77.67
Student  6      78      87      65    76.67
Student  7      85      75      83    81.00
Student  8      91      94     100    95.00
Student  9      76      72      84    77.33
Student 10      87      93      73    84.33
```

**Fig. 7.21** | Output of `GradeBook` that uses two-dimensional `arrays`. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Overall grade distribution:
  0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: ***********
90-99: *******
  100: ***
```

**Fig. 7.21** | Output of `GradeBook` that uses two-dimensional `arrays`. (Part 2 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
1  // Fig. 7.22: GradeBook.h
2  // Definition of class GradeBook that uses a
3  // two-dimensional array to store test grades.
4  // Member functions are defined in GradeBook.cpp
5  #include <array>
6  #include <string>
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // constants
13    static const size_t students = 10; // number of students
14    static const size_t tests = 3; // number of tests
15
16    // constructor initializes course name and array of grades
17    GradeBook( const std::string &,
18       std::array< std::array< int, tests >, students > & );
19
```

**Fig. 7.22** | Definition of class GradeBook that uses a two-dimensional array to store test grades. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.