7

Arrays and Vectors

东南大学软件学院

OBJECTIVES

In this chapter you'll learn:

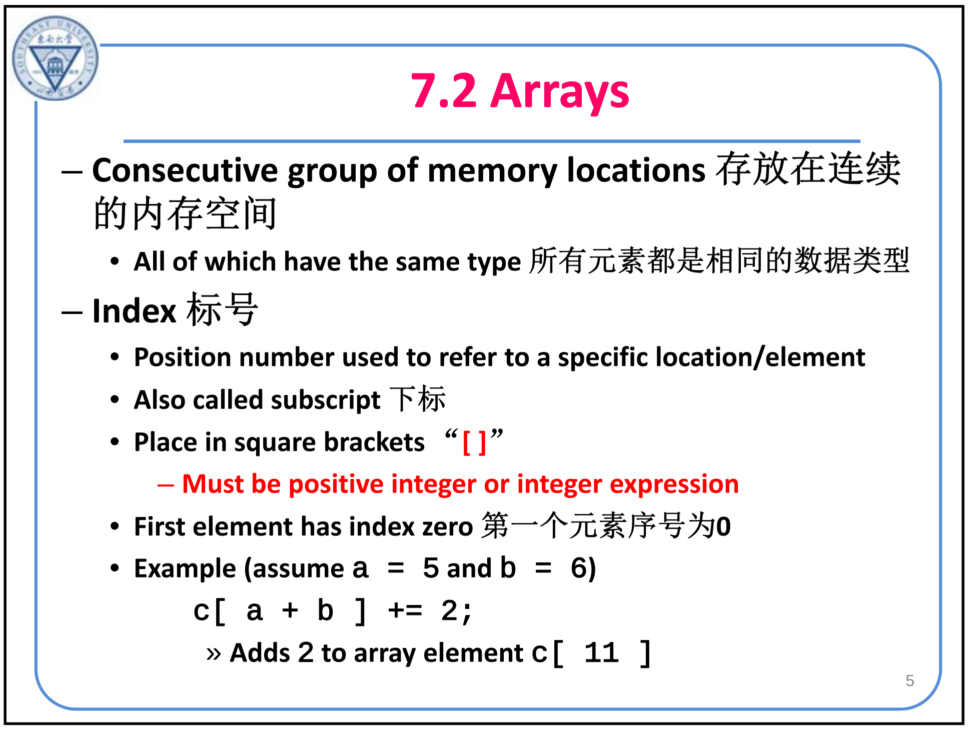
- To use the array data structure to represent a set of related data items.
- To declare arrays, initialize arrays and refer to the individual elements of arrays.
- To use arrays to store存储, sort排序 and search查找 lists and tables of values.
- Basic searching and sorting techniques.
- To pass arrays to functions.数组作为参数进行函数调用
- To declare and manipulate multidimensional arrays.多维数组
- To use C++ Standard Library class template `vector`.

7.1 Introduction

- Arrays
  - Data structures containing related data items of same type
  - Always remain the same size once created
    - Are “static” entities
  - Character arrays can also represent strings
  - C-style pointer-based arrays vs. vectors (object-based)
    - Vectors are safer and more versatile

7.3 Declaring Arrays

- Declaring an array 数组声明
  - Arrays occupy space in memory
  - Programmer specifies type and number of elements
    - `int c[ 12 ];//c is an array of 12 ints`
  - Array’s size must be an integer constant greater than zero
  - Arrays can be declared to contain values of any non-reference data type
  - Multiple arrays of the same type can be declared in a single declaration
    - `int c[ 12 ], b[ 5 ];`



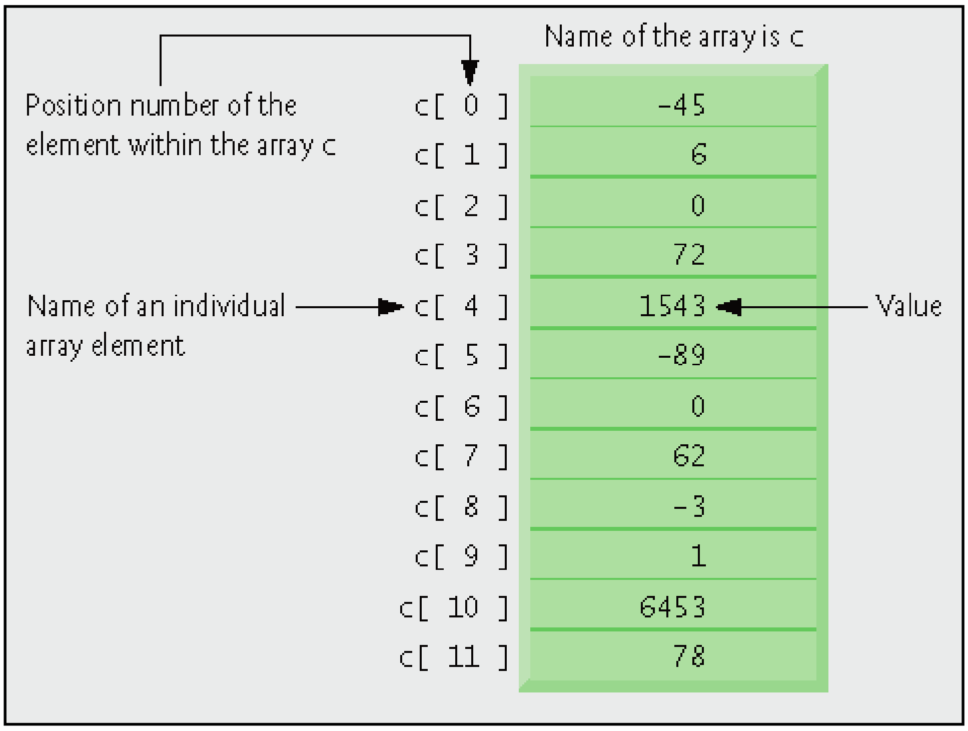
7.2 Arrays

– Consecutive group of memory locations 存放在连续的内存空间
  - All of which have the same type 所有元素都是相同的数据类型
– Index 标号
  - Position number used to refer to a specific location/element
  - Also called subscript 下标
  - Place in square brackets “[ ]”
    – Must be positive integer or integer expression
  - First element has index zero 第一个元素序号为0
  - Example (assume `a = 5` and `b = 6`)

```
c[ a + b ] += 2;
```

» Adds 2 to array element `c[ 11 ]`

5



7.2 Arrays (Cont.)

- Examine array `c` in Fig. 7.1
  - `c` is the array *name*
  - `c` has 12 *elements* (`c[0]`, `c[1]`, … `c[11]`)
    - The *value* of `c[0]` is `–45`
- Brackets used to enclose an array subscript are actually an operator in C++

7

Operators	Associativity	Type
`::`	left to right	scope resolution
`()` `[]`	left to right	parens/brackets
`++` `--` `static_cast`< *type* >(*operand*)	left to right	unary (postfix)
`++` `--` `+` `-` `!`	right to left	unary (prefix)
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<<` `>>`	left to right	insertion/extraction
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`=` `+=` `-=` `*=` `/=` `%=`	right to left	assignment
`,`	left to right	comma

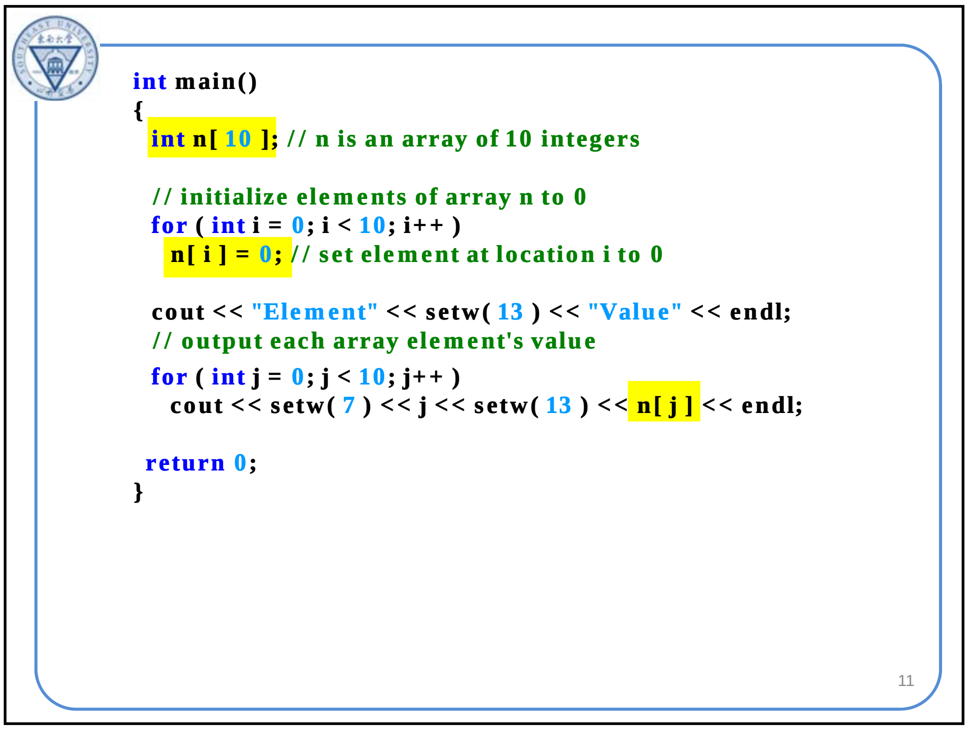
8

Common Programming Error 7.1

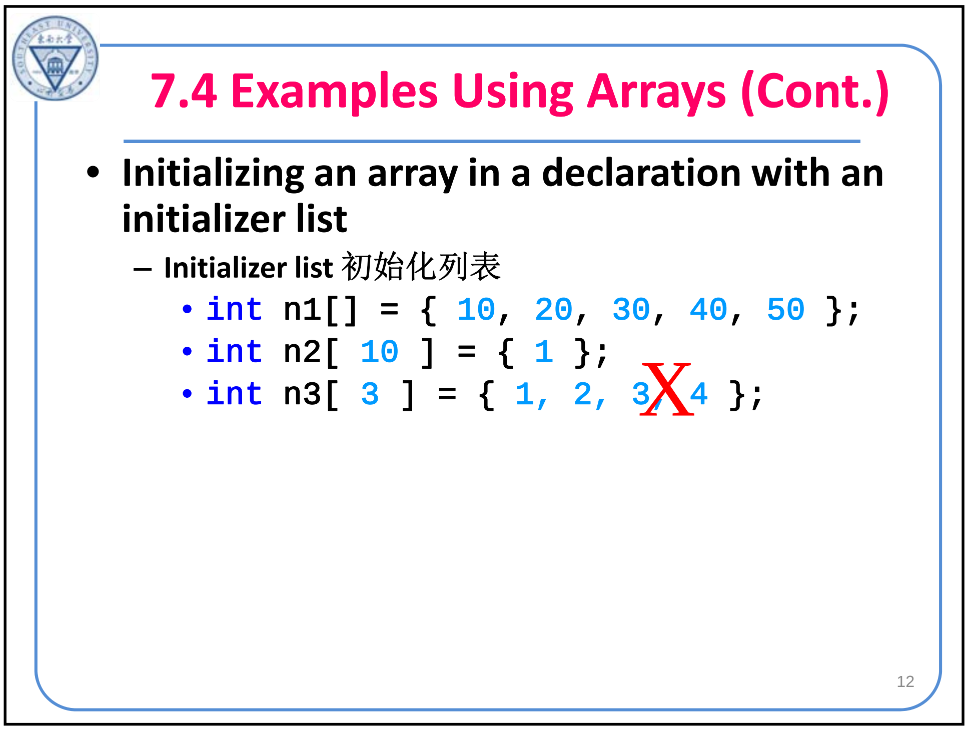
It is important to note the difference between the "seventh element of the array" and "array element 7." Array subscripts begin at 0 (e.g., `c[ 6 ]` or `c[ 7 ]`).

7.4 Examples Using Arrays

- Using a loop to initialize the array's elements
  - Declare array, specify number of elements
  - Use repetition statement to loop for each element
    - Use body of repetition statement to initialize each individual array element



```
int main()
{
  int n[ 10 ]; // n is an array of 10 integers

  // initialize elements of array n to 0
  for ( int i = 0; i < 10; i++ )
    n[ i ] = 0; // set element at location i to 0

  cout << "Element" << setw( 13 ) << "Value" << endl;
  // output each array element's value
  for ( int j = 0; j < 10; j++ )
    cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;

 return 0;
}
```



7.4 Examples Using Arrays (Cont.)

- **Initializing an array in a declaration with an initializer list**
  - Initializer list 初始化列表
    - `int n1[] = { 10, 20, 30, 40, 50 };`
    - `int n2[ 10 ] = { 1 };`
    - `int n3[ 3 ] = { 1, 2, 3X4 };`

7.4 Examples Using Arrays (Cont.)

- Specifying an array’s size with a constant variable常数变量and setting array elements with calculations
  - Initialize elements of 10-element array to even integers
  - Use repetition statement 循环语句 that calculates value for current element, initializes array element using calculated value

```
int main()
{
  // constant variable can be used to specify array size
  const int arraySize = 10;  定义时必须初始化

  int s[ arraySize ];   此处只能使用常整形变量

  for ( int i = 0; i < arraySize; i++ ) // set the values
    s[ i ] = 2 + 2 * i;

  cout << "Element" << setw( 13 ) << "Value" << endl;

  // output contents of array s in tabular format
  for ( int j = 0; j < arraySize; j++ )
       cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

  return 0;                 提高程序的可读性和可扩展性
}
```

7.4 Examples Using Arrays (Cont.)

- Summing the elements of an array
  - Array elements can represent a series of values
    - We can sum these values
    - Use repetition statement to loop through each element
      - Add element value to a total

```
int main()
{
     const int arraySize = 10;
     int a[ arraySize ] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
     int total = 0;
     for ( int i = 0; i < arraySize; i++ )
       total += a[ i ];
     cout << "Total of array elements: " << total << endl;
     return 0;
}
```

7.4 Examples Using Arrays (Cont.)

- Using bar charts to display array data graphically (程序图7.9)
  - Present data in graphical manner
    - E.g., bar chart
  - Examine the distribution of grades
  - Nested `for` statement used to output bars

7.4 Examples Using Arrays (Cont.)

- Using the elements of an array as counters
  - Use a series of counter variables to summarize data
    int frequency1, frequency2, frequency3, …, frequency6;
    使用switch语句结构
  - Counter variables make up an array
  - Store frequency values int frequency[6];
  - P198，图6.9程序的数组版本

应该如何做？

17

```
17 int main()
18 {
19    const int arraySize = 7;
20    int frequency[ arraySize ] = { 0 };
21
22    srand( time( 0 ) );
23
24    // roll die 6,000,000 times; use die value as frequency index
25    for ( int roll = 1; roll <= 6000000; roll++ )
26       frequency[ 1 + rand() % 6 ]++;
27
28    cout << "Face" << setw( 13 ) << "Frequency" << endl;
29
30    // output each array element's value
31    for ( int face = 1; face < arraySize; face++ )
32       cout << setw( 4 ) << face << setw( 13 ) << frequency[ face ]
33          << endl;
34
35    return 0;
36 }
```

以骰子的值作为下标，
frequency[0]没有使用

如果19行的arraySize误写为6，结果会怎样？

18

7.4 Examples Using Arrays (Cont.)

- C++ has no array bounds checking 无边界检测
  - Does not prevent the computer from referring to an element that does not exist
    - Could lead to serious execution-time errors运行时错误

7.4 Examples Using Arrays (Cont.)

- Using arrays to summarize survey results
  - 40 students rate the quality of food
    - 1-10 rating scale: 1 means awful, 10 means excellent
  - Place 40 responses in an array of integers
  - Summarize results
  - Each element of the array used as a counter for one of the survey responses

```
int main()
{
  // define array sizes
  const int responseSize = 40; // size of array responses
  const int frequencySize = 11; // size of array frequency

  // place survey responses in array responses
  const int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
    10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
    5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

  // initialize frequency counters to 0
  int frequency[ frequencySize ] = { 0 };

 // for each answer, select responses element and use that value
 // as frequency subscript to determine element to increment
  for ( int answer = 0; answer < responseSize; answer++ )
    frequency[ responses[ answer ] ]++;

  cout << "Rating" << setw( 17 ) << "Frequency" << endl;
```

Software Engineering Observation 7.2

The `const` qualifier should be used to enforce the principle of least privilege. Using the principle of least privilege to properly design software can greatly reduce debugging time and improper side effects and can make a program easier to modify and maintain.

最小特权原则：规定代码应该只被赋予完成它的设计任务所需要的权限，无需更多的权限。

7.4 Examples Using Arrays (Cont.)

- Using character arrays to store and manipulate strings
  - Arrays may be of any type, including `char`s
    - We can store character strings in `char` arrays
  - Can be initialized using a string literal
    - Example

      ```
      char string1[] = "Hi";
      ```
    - Equivalent to

      ```
      char string1[] = { 'H', 'i', '\0' };
      ```
  - Array contains each character **plus a special string-termination character called the null character (`'\0'`)**

7.4 Examples Using Arrays (Cont.)

- Using character arrays to store and manipulate strings (Cont.)
  - Can also input a string directly into a character array from the keyboard using

    ```
    cin >> string1;
    ```
    - `cin >>` may read more characters than the array can store 数组大小应足够大，确保能放下输入的字符串
  - A character array representing a null-terminated string can be output with `cout << string1;`
    字符数组必须包含字符串终止符才可用这种输出方式

```
int main()
{
  char string1[ 20 ];
  char string2[] = "string  literal";字符串可以包含空格

  // read string from user into array string1
  cout << "Enter the string \"hello there\": ";
  cin >> string1;   cin输入字符以白字符作为分隔，不可以包含空格

  cout << "string1 is: " << string1 << "\nstring2 is: " << string2;

  cout << "\nstring1 with spaces between characters is:\n";

  for ( int i = 0; string1[ i ] != '\0'; i++ )
     cout << string1[ i ] << ' ';

  cin >> string1; // reads "there"
  cout << "\nstring1 is: " << string1 << endl;

  return 0;
}
```

```
Enter the string "hello there": hello there
string1 is: hello
string2 is: string literal
string1 with spaces between characters is:
h e l l o
string1 is: there
```

输入

无需输入there

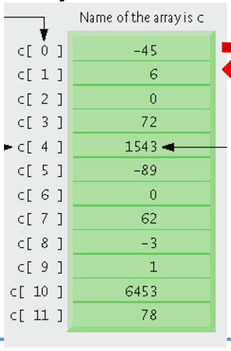
cin.sync(); //用于清空输入缓存区未读取的信息

7.4 Examples Using Arrays (Cont.)

- `static` local arrays and automatic local arrays
  - A `static` local variable in a function
    - Exists for the duration of the program
    - But is visible only in the function body
  - A `static` local array
    - Exists for the duration of the program
    - Is initialized when its declaration is first encountered
      - All elements are initialized to zero if not explicitly initialized

7.5 Passing Arrays to Functions (Cont.)

- Functions that take arrays as arguments
  - Function parameter list must specify array parameter
    - `void modArray( int b[], int arraySize );`
    - `void modArray( int b[3], int arraySize );`
    - Compiler only cares about the address of the first element



从c[0]到c[1]只需要知道存放**int**型数需要多大的内存空间

7.5 Passing Arrays to Functions

- To pass an array argument to a function
  - Specify array name without brackets

```
int myarray[ 24 ];
modifyArray(myarray,24);
```

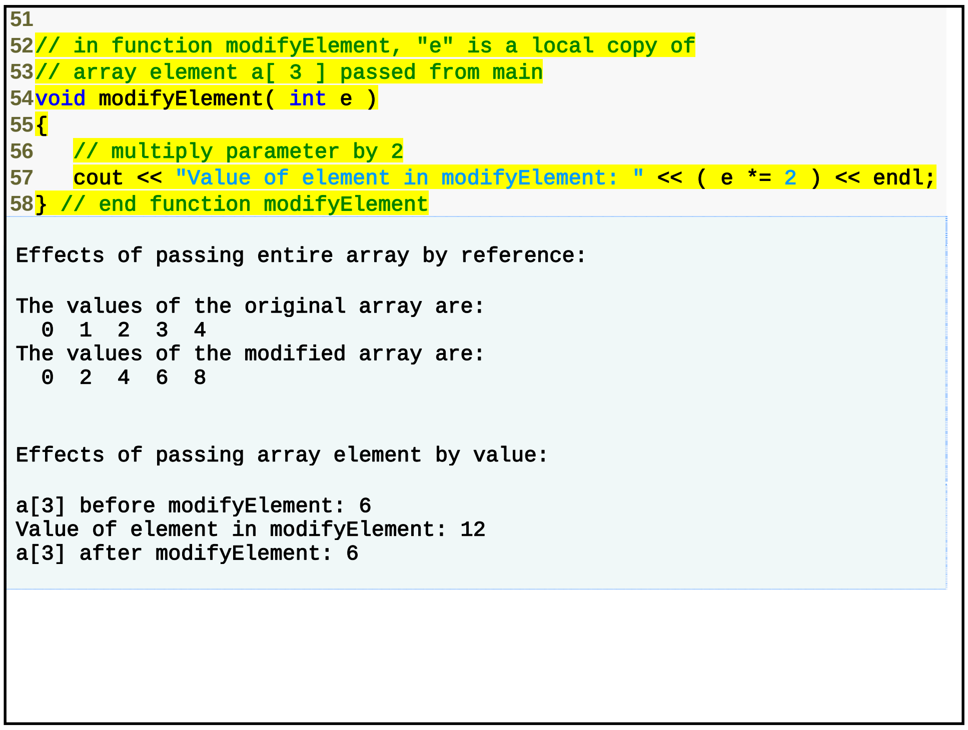
  - Array size is normally passed as another argument so the function can process the specific number of elements in the array

7.5 Passing Arrays to Functions (Cont.)

- Arrays are passed by reference 以传引用方式
  - Function call actually passes starting address of array
    - So function knows where array is located in memory
  - Caller gives called function direct access to caller’s data
    - Called function can manipulate this data

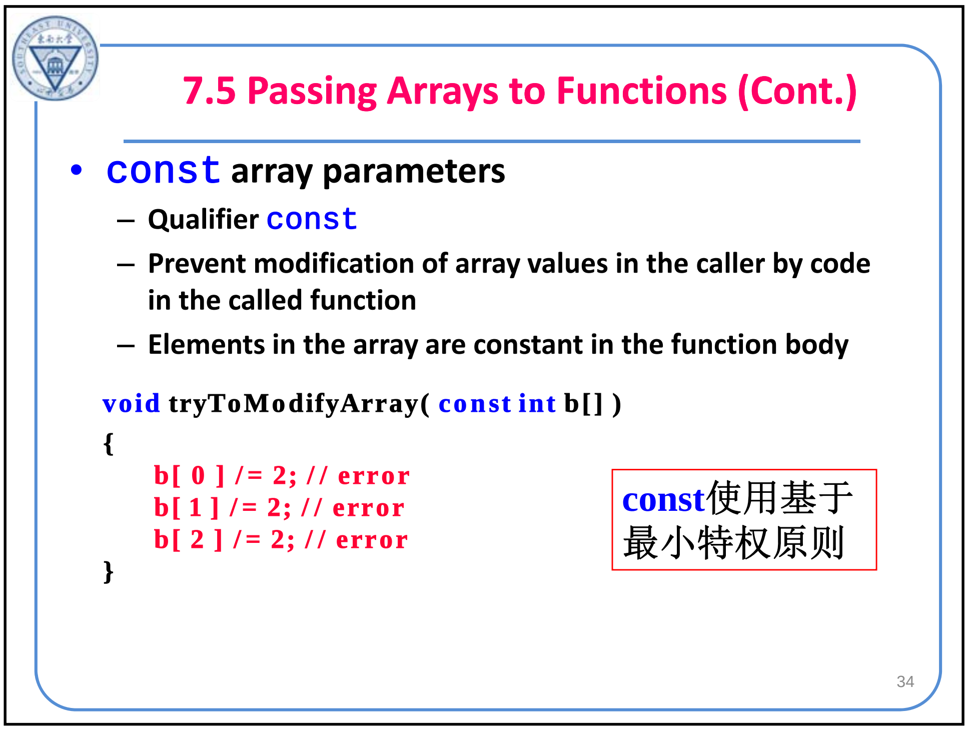
```
// Fig. 7.14: fig07_14.cpp
// Passing arrays and individual array elements to functions.

void modifyArray( int [], int ); // appears strange
void modifyElement( int );

int main()
{
   const int arraySize = 5; // size of array a
   int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a

   cout << "Effects of passing entire array by reference:"
      << "\n\nThe values of the original array are:\n";

   // output original array elements
   for ( int i = 0; i < arraySize; i++ )
      cout << setw( 3 ) << a[ i ];

   cout << endl;

   // pass array a to modifyArray by reference
   modifyArray( a, arraySize );
   cout << "The values of the modified array are:\n";
```

```

   // output modified array elements
   for ( int j = 0; j < arraySize; j++ )
      cout << setw( 3 ) << a[ j ];

   cout << "\n\n\nEffects of passing array element by value:"
      << "\n\na[3] before modifyElement: " << a[ 3 ] << endl;

   modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
   cout << "a[3] after modifyElement: " << a[ 3 ] << endl;

   return 0; // indicates successful termination
} // end main

// in function modifyArray, "b" points to the original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
   // multiply each array element by 2
   for ( int k = 0; k < sizeOfArray; k++ )
      b[ k ] *= 2;
} // end function modifyArray
```



```cpp

// in function modifyElement, "e" is a local copy of
// array element a[ 3 ] passed from main
void modifyElement( int e )
{
   // multiply parameter by 2
   cout << "Value of element in modifyElement: " << ( e *= 2 ) << endl;
} // end function modifyElement
```

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

a[3] before modifyElement: 6
Value of element in modifyElement: 12
a[3] after modifyElement: 6
```



7.5 Passing Arrays to Functions (Cont.)

- `const` array parameters
  - Qualifier `const`
  - Prevent modification of array values in the caller by code in the called function
  - Elements in the array are constant in the function body

```cpp
void tryToModifyArray( const int b[] )
{
    b[ 0 ] /= 2; // error
    b[ 1 ] /= 2; // error
    b[ 2 ] /= 2; // error
}
```

const使用基于最小特权原则

7.6 Case Study: Class GradeBook Using an Array to Store Grades

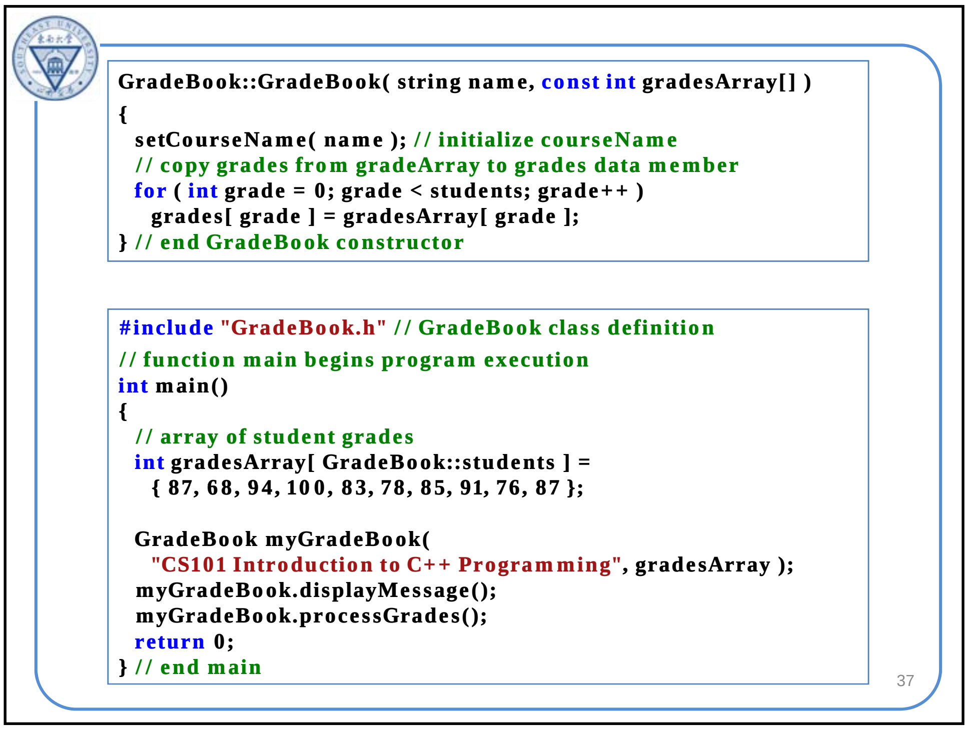
- 例：统计学生成绩（最高、最低、均分），并画出成绩分布情况
- Class GradeBook
  - Represent a grade book that stores and analyzes grades
  - Can now store grades in an array
- static data members 静态数据成员
  - Also called class variables (类变量)
  - Variables for which each object of a class does not have a separate copy
    - One copy is shared among all objects of the class

```
class GradeBook
   {
   public:
     const static int students = 10;
     GradeBook( string, const int [] );
      ......
   private:
     string courseName;
     int grades[ students ];
 };
```

➢ GradeBook类的所有对象共享students静态类变

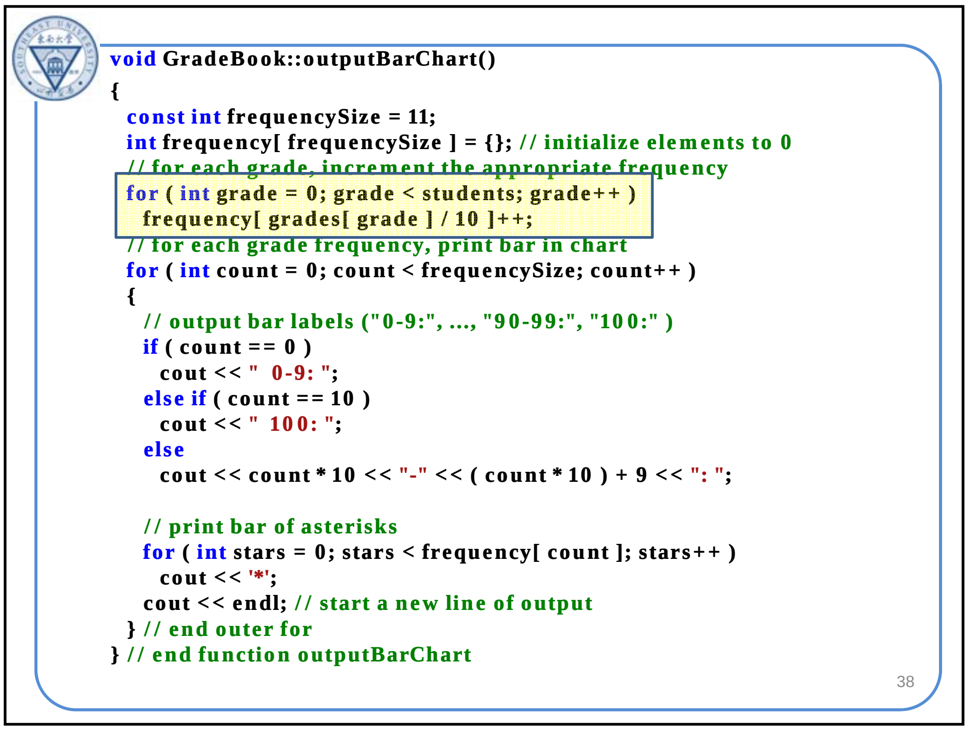
➢ 即使没有定义GradeBook类的对象，也可以用GradeBook::students来读取其中的值。
例如，客户段代码中可以写如下语句：

```
int gradesArray[ GradeBook::students ] =
{ 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
```



```
GradeBook::GradeBook( string name, const int gradesArray[] )
{
  setCourseName( name ); // initialize courseName
  // copy grades from gradeArray to grades data member
  for ( int grade = 0; grade < students; grade++ )
    grades[ grade ] = gradesArray[ grade ];
} // end GradeBook constructor
```

```
#include "GradeBook.h" // GradeBook class definition
// function main begins program execution
int main()
{
  // array of student grades
  int gradesArray[ GradeBook::students ] =
    { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };

  GradeBook myGradeBook(
    "CS101 Introduction to C++ Programming", gradesArray );
  myGradeBook.displayMessage();
  myGradeBook.processGrades();
  return 0;
} // end main
```



```
void GradeBook::outputBarChart()
{
  const int frequencySize = 11;
  int frequency[ frequencySize ] = {}; // initialize elements to 0
  // for each grade, increment the appropriate frequency
  for ( int grade = 0; grade < students; grade++ )
    frequency[ grades[ grade ] / 10 ]++;
  // for each grade frequency, print bar in chart
  for ( int count = 0; count < frequencySize; count++ )
  {
    // output bar labels ("0-9:", ..., "90-99:", "100:" )
    if ( count == 0 )
      cout << "  0-9: ";
    else if ( count == 10 )
      cout << "  100: ";
    else
      cout << count * 10 << "-" << ( count * 10 ) + 9 << ": ";

    // print bar of asterisks
    for ( int stars = 0; stars < frequency[ count ]; stars++ )
      cout << '*';
    cout << endl; // start a new line of output
  } // end outer for
} // end function outputBarChart
```

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

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100

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

7.7 Searching Arrays with Linear Search

- Arrays may store large amounts of data
  - May need to determine if certain key value is located in an array
- Linear search 线性查找
  - Compares each element of an array with a search key
  - On average, program must compare the search key with half the elements of the array
  - To determine that value is not in array, program must compare the search key to every element in the array
  - Works well for small or unsorted arrays

```
int linearSearch( const int [], int, int ); // prototype

int main()
{
  ......
  int element = linearSearch( a, searchKey, arraySize );
  if ( element != -1 )
       cout << "Found value in element " << element << endl;
  else
       cout << "Value not found" << endl;
  return 0;
} // end main

int linearSearch( const int array[], int key, int sizeOfArray )
{
     for ( int j = 0; j < sizeOfArray; j++ )
       if ( array[ j ] == key ) // if found,
          return j;
     return -1;
} // end function linearSearch
```

7.8 Sorting Arrays with Insertion Sort

- Sorting data 排序
  - One of the most important computing applications
- Insertion sort 插入排序
  - Simple but inefficient（程序简单但效率较低）

```
         34  56   4  10  77  51  93  30  5  52
Iter 1:  34 [56]  4  10  77  51  93  30  5  52
Iter 2:  34  56 [4]  10  77  51  93  30  5  52
Iter 3:   4  34  56 [10] 77  51  93  30  5  52
```

```
for ( int next = 1; next < arraySize; next++ )
  {
    insert = data[ next ];
    int moveItem = next;

    while ( ( moveItem > 0 ) && ( data[ moveItem - 1 ] > insert ) )
    {
      data[ moveItem ] = data[ moveItem - 1 ];
      moveItem--;
    } // end while
    data[ moveItem ] = insert;
  } // end for
```

insert = data[next]

next=3 →

4 34 56 $\boxed{10}$ 77 51 93 30 5 52

← moveItem

Unsorted array:

```
34 56 4 10 77 51 93 30 5 52
34 56 4 10 77 51 93 30 5 52
4 34 56 10 77 51 93 30 5 52
4 10 34 56 77 51 93 30 5 52
4 10 34 56 77 51 93 30 5 52
4 10 34 51 56 77 93 30 5 52
4 10 34 51 56 77 93 30 5 52
4 10 30 34 51 56 77 93 5 52
4 5 10 30 34 51 56 77 93 52
4 5 10 30 34 51 52 56 77 93
```

7.9 Multidimensional Arrays 多维数组

- Multidimensional arrays with two dimensions
  - Called two dimensional or 2-D arrays
  - Represent tables of values with rows and columns
  - Elements referenced with two subscripts (`[ x ][ y ]`)
  - In general, an array with *m* rows and *n* columns is called an *m*-by-*n* array
- Multidimensional arrays can have more than two dimensions 多维数组可以超过二维

45

7.9 Multidimensional Arrays (Cont.)

- Declaring and initializing two-dimensional arrays
  - Declaring two-dimensional array
    - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`

      1 and 2 initialize `b[ 0 ][ 0 ]` and `b[ 0 ][ 1 ]`

      3 and 4 initialize `b[ 1 ][ 0 ]` and `b[ 1 ][ 1 ]`
    - `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`
      - Row `0` contains values `1` and `0` (implicitly initialized to zero)
      - Row `1` contains values `3` and `4`
  - Multi-dimensional array
    - `int array[3][4][5];`

46

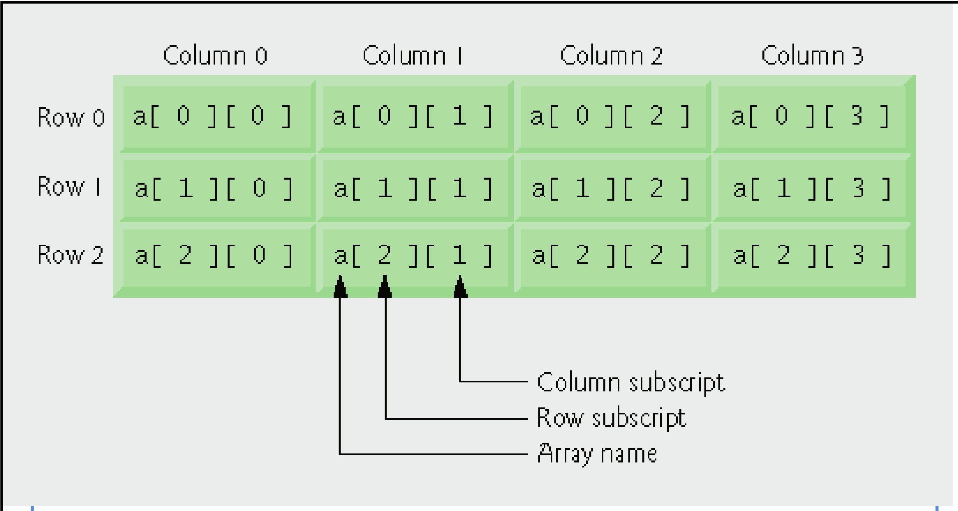
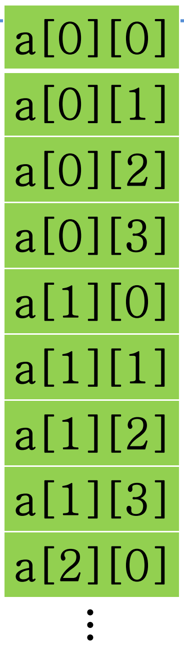


Fig.7.21 | Two-dimensional array with three rows and four columns.



7.9 Multidimensional Array (Cont.)

- Multidimensional-array manipulations
  - Commonly performed with `for` statements
    - Example
      - Modify all elements in a row

```
for ( int col = 0; col < 4; col++ )
   a[ 2 ][ col ] = 0;
```

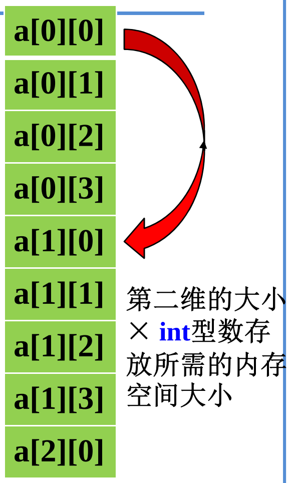
    - Example
      - Total all elements

```
total = 0;
for ( row = 0; row < 3; row++ )
   for ( col = 0; col < 4; col++ )
      total += a[ row ][ col ];
```

49

7.9 Multidimensional Arrays (Cont.)

- Multidimensional array parameter多维数组形参
  - Size of first dimension is not required 第一维大小不需要
    - As with a one-dimensional array
  - Size of subsequent dimensions are required 第二维大小必须提供
    - Compiler must know how many elements to skip to move to the second element in the first dimension



```
void printArray( const int a[][ 3 ] );
```

50

```cpp
void printArray( const int [][ 3 ] ); // prototype

int main()
{
 int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
 int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
 int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

 cout << "Values in array1 by row are:" << endl;
 printArray( array1 );

 cout << "\nValues in array2 by row are:" << endl;
 printArray( array2 );

 cout << "\nValues in array3 by row are:" << endl;
 printArray( array3 );
 return 0; // indicates successful termination
} // end main
```

```cpp
void printArray( const int a[][ 3 ] )
{
 // loop through array's rows
 for ( int i = 0; i < 2; i++ )
 {
   // loop through columns of current row
   for ( int j = 0; j < 3; j++ )
      cout << a[ i ][ j ] << ' ';

   cout << endl;
 }
}
```

交换两行代码位置，
输出结果是什么？

注意在这个程序里二维数
组的大小必须事先给定

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

7.10 Case Study: Class GradeBook Using a Two-Dimensional Array

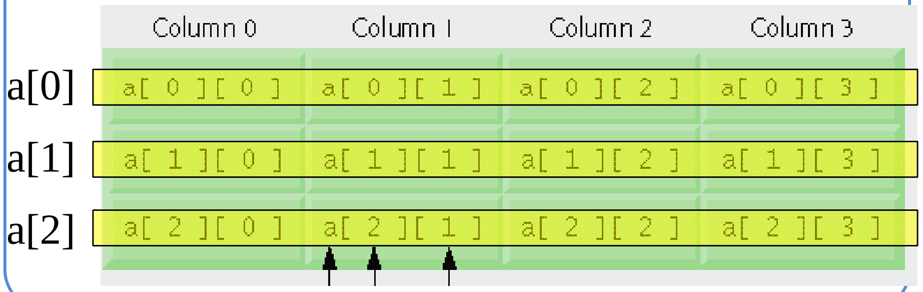
- Class GradeBook
  - One-dimensional array
    - Store student grades on a single exam
  - Two-dimensional array
    - Store multiple grades for a single student and multiple students for the class as a whole
      - Each row represents a student's grades
      - Each column represents all the grades the students earned for one particular exam

```
//fig 7.23
class GradeBook
{
   public:
     const static int students = 10; // number of students
     const static int tests = 3; // number of tests
     GradeBook( string, const int [][ tests ] );
     ......
     double getAverage( const int [], const int );
         private:
     string courseName;
     int grades[ students ][ tests ];
};
```

```
void GradeBook::outputBarChart()
{
 cout << "\nOverall grade distribution:" << endl;
 const int frequencySize = 11;
 int frequency[ frequencySize ] = { 0 };
 for ( int student = 0; student < students; student++ )
     for ( int test = 0; test < tests; test++ )
         ++frequency[ grades[ student ][ test ] / 10 ];
  ……
}
```

55

```
void GradeBook::outputGrades()
{
 ……
 for ( int student  = 0; student< students; student++ )
 {
   ……
   for ( int test = 0; test < tests; test++ )
          cout << setw( 8 ) << grades[ student ][ test ];
   double average = getAverage( grades[ student ], tests );
   cout << setw( 9 ) << setprecision( 2 ) << fixed << average << endl;
 } // end outer for
} // end function outputGrades
```



	Column 0	Column 1	Column 2	Column 3
a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

7.11 Introduction to C++ Standard Library Class Template `vector`

- C-style pointer-based arrays
  - Have great potential for errors and several shortcomings
    - C++ does not check whether subscripts fall outside the range of the array
    - Two arrays cannot be meaningfully compared with equality or relational operators
    - One array cannot be assigned to another using the assignment operators

57

7.11 Introduction to C++ Standard Library Class Template `vector` (Cont.)

- Class template `vector`
  - Available to anyone building applications with C++
  - Can be defined to store any data type
    - Specified between angle brackets in `vector<` *type* `>`
    - All elements in a `vector` are set to `0` by default
  - Member function `size` obtains size of array
    - Number of elements as a value of type `size_t`
  - `vector` objects can be compared using equality and relational operators
  - Assignment operator can be used for assigning `vector`s

58

7.11 Introduction to C++ Standard Library Class Template `vector` (Cont.)

- `vector` member function `at`
  - Provides access to individual elements
  - Performs bounds checking 边界检测
    - Throws an exception when specified index is invalid
    - Accessing with square brackets does not perform bounds checking

59

```
#include <vector>
#include <iostream>
#include <iomanip>
int main( )
{
  using namespace std;
  vector <int> v1;

  v1.push_back( 10 );
  v1.push_back( 20 );
  const int &i = v1.at( 0 );
  int &j = v1.at( 1 );
  cout << "The first element is " << i << endl;
  cout << "The second element is " << j << endl;
  cout<<"Please input an integer number...";
  int num;
  cin>>num;
  v1.resize(num);
  for (size_t i=0;i<v1.size();i++)
           cout<<setw(4)<<i<<setw(10)<<v1.at(i)<<endl;
}
```

60

```cpp
// Fig. 7.26: fig07_26.cpp
// Demonstrating C++ Standard Library class template vector.

#include <vector>
using std::vector;

void outputVector( const vector< int > & ); // display the vector
void inputVector( vector< int > & ); // input values into the vector

int main()
{
   vector< int > integers1( 7 ); // 7-element vector< int >
   vector< int > integers2( 10 ); // 10-element vector< int >

   // print integers1 size and contents
   cout << "Size of vector integers1 is " << integers1.size()
      << "\nvector after initialization:" << endl;
   outputVector( integers1 );

   // print integers2 size and contents
   cout << "\nSize of vector integers2 is " << integers2.size()
      << "\nvector after initialization:" << endl;
   outputVector( integers2 );
```

```cpp

   // use inequality (!=) operator with vector objects
   cout << "\nEvaluating: integers1 != integers2" << endl;

   if ( integers1 != integers2 )
      cout << "integers1 and integers2 are not equal" << endl;

   // create vector integers3 using integers1 as an
   // initializer; print size and contents
   vector< int > integers3( integers1 ); // copy constructor

   cout << "\nSize of vector integers3 is " << integers3.size()
      << "\nvector after initialization:" << endl;
   outputVector( integers3 );

   // use overloaded assignment (=) operator
   cout << "\nAssigning integers2 to integers1:" << endl;
   integers1 = integers2; // integers1 is larger than integers2

   // use equality (==) operator with vector objects
   cout << "\nEvaluating: integers1 == integers2" << endl;

   if ( integers1 == integers2 )
      cout << "integers1 and integers2 are equal" << endl;

   // use square brackets to create rvalue
   cout << "\nintegers1[5] is " << integers1[ 5 ];

   // use square brackets to create lvalue
   cout << "\n\nAssigning 1000 to integers1[5]" << endl;
   integers1[ 5 ] = 1000;
   cout << "integers1:" << endl;
   outputVector( integers1 );

   // attempt to use out-of-range subscript
   cout << "\nAttempt to assign 1000 to integers1.at( 15 )" << endl;
   integers1.at( 15 ) = 1000; // ERROR: out of range
   return 0;
} // end main
```

```cpp

// output vector contents
void outputVector( const vector< int > &array )
{
   size_t i; // declare control variable

   for ( i = 0; i < array.size(); i++ )
   {
      cout << setw( 12 ) << array[ i ];

      if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
         cout << endl;
   } // end for

   if ( i % 4 != 0 )
      cout << endl;
} // end function outputVector

// input vector contents
void inputVector( vector< int > &array )
{
   for ( size_t i = 0; i < array.size(); i++ )
      cin >> array[ i ];
} // end function inputVector
```

```
Size of vector integers1 is 7
vector after initialization:
           0           0           0           0
           0           0           0

Size of vector integers2 is 10
vector after initialization:
           0           0           0           0
           0           0           0           0
           0           0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the vectors contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2
integers1 and integers2 are not equal

Size of vector integers3 is 7
vector after initialization:
           1           2           3           4
           5           6           7
```

(continued at top of next slide)

(continued from bottom of previous slide)

```
Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

Attempt to assign 1000 to integers1.at( 15 )

abnormal program termination
```