```
   // 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; // assign integers2 to integers1

   cout << "integers1:" << endl;
   outputVector( integers1 );
   cout << "integers2:" << endl;
   outputVector( integers2 );

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

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

```

Fig. 7.25 | Demonstrating C++ Standard Library class template vector. (Part 3 of 7.)

```
   // use square brackets to use the value at location 5 as an 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
   try
   {
      cout << "\nAttempt to display integers1.at( 15 )" << endl;
      cout << integers1.at( 15 ) << endl; // ERROR: out of range
   } // end try
   catch ( out_of_range &ex )
   {
      cerr << "An exception occurred: " << ex.what() << endl;
   } // end catch
```

Fig. 7.25 | Demonstrating C++ Standard Library class template `vector`. (Part 4 of 7.)

```cpp

   // changing the size of a vector
   cout << "\nCurrent integers3 size is: " << integers3.size() << endl;
   integers3.push_back( 1000 ); // add 1000 to the end of the vector
   cout << "New integers3 size is: " << integers3.size() << endl;
   cout << "integers3 now contains: ";
   outputVector( integers3 );
} // end main

// output vector contents
void outputVector( const vector< int > &array )
{
   for ( int item : items )
      cout << item << " ";

   cout << endl;
} // end function outputVector

// input vector contents
void inputVector( vector< int > &array )
{
   for ( int &item : items )
      cin >> item;
} // end function inputVector
```

Fig. 7.25 | Demonstrating C++ Standard Library class template vector. (Part 5 of 7.)

```
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
```

Fig. 7.25 | Demonstrating C++ Standard Library class template `vector`. (Part 6 of 7.)

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

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 display integers1.at( 15 )
An exception occurred: invalid vector<T> subscript

Current integers3 size is: 7
New integers3 size is: 8
integers3 now contains: 1 2 3 4 5 6 7 1000
```

Fig. 7.25 | Demonstrating C++ Standard Library class template `vector`. (Part 7 of 7.)

7.10 Introduction to C++ Standard Library Class Template vector (cont.)

- By default, all the elements of a vector object are set to 0.
- vectors can be defined to store most data types.
- vector member function size obtain the number of elements in the vector.
- As with class template array, you can also do this using a counter-controlled loop and the subscript ([]) operator.

7.10 Introduction to C++ Standard Library Class Template `vector` (cont.)

- You can use the assignment (=) operator with `vector` objects.
- As is the case with `array`s, C++ is not required to perform bounds checking when `vector` elements are accessed with square brackets.
- Standard class template `vector` provides bounds checking in its member function `at` (as does class template `array`).

7.10 Introduction to C++ Standard Library Class Template `vector` (cont.)

- An exception indicates a problem that occurs while a program executes.
- The name “exception” suggests that the problem occurs infrequently—if the “rule” is that a statement normally executes correctly, then the problem represents the “exception to the rule.”
- Exception handling enables you to create fault-tolerant programs that can resolve (or handle) exceptions.
- When a function detects a problem, such as an invalid `array` subscript or an invalid argument, it throws an exception—that is, an exception occurs.

7.10 Introduction to C++ Standard Library Class Template `vector` (cont.)

- To handle an exception, place any code that might throw an exception in a `try` statement.
- The `try` block contains the code that might throw an exception, and the `catch` block contains the code that handles the exception if one occurs.
- You can have many `catch` blocks to handle different types of exceptions that might be thrown in the corresponding `try` block.
- The `vector` member function `at` provides bounds checking and throws an exception if its argument is an invalid subscript.
- By default, this causes a C++ program to terminate.

7.10 Introduction to C++ Standard Library Class Template vector (cont.)

Changing the Size of a vector

- One of the key differences between a vector and an array is that a vector can dynamically grow to accommodate more elements.
- To demonstrate this, line 88 shows the current size of integers3, line 89 calls the vector’s push_back member function to add a new element containing 1000 to the end of the vector and line 90 shows the new size of integers3

7.10 Introduction to C++ Standard Library Class Template `vector` (cont.)

C++11: List Initializing a `vector`

- Many of the `array` examples in this chapter used list initializers to specify the initial `array` element values.
- C++11 also allows this for `vector`s (and other C++ Standard Library data structures).
- At the time of this writing, list initializers were not yet supported for `vector`s in Visual C++.