



# Vectors in C++

*Mohamed Saied*

## 7.10 Introduction to C++ Standard Library Class Template `vector`

- C++ Standard Library class template `vector` is similar to class template `array`, but also supports dynamic resizing.
- Except for the features that modify a `vector`, the other features shown in Fig. 7.25 also work for `arrays`.
- Standard class template `vector` is defined in header `<vector>` (line 5) and belongs to namespace `std`.

---

```
1 // Fig. 7.25: fig07_25.cpp
2 // Demonstrating C++ Standard Library class template vector.
3 #include <iostream>
4 #include <iomanip>
5 #include <vector>
6 #include <stdexcept>
7 using namespace std;
8
9 void outputVector( const vector< int > & ); // display the vector
10 void inputVector( vector< int > & ); // input values into the vector
11
12 int main()
13 {
14     vector< int > integers1( 7 ); // 7-element vector< int >
15     vector< int > integers2( 10 ); // 10-element vector< int >
16
17     // print integers1 size and contents
18     cout << "Size of vector integers1 is " << integers1.size()
19         << "\nvector after initialization:" << endl;
20     outputVector( integers1 );
21 }
```

---

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

---

```
22 // print integers2 size and contents
23 cout << "\nSize of vector integers2 is " << integers2.size()
24     << "\nvector after initialization:" << endl;
25 outputVector( integers2 );
26
27 // input and print integers1 and integers2
28 cout << "\nEnter 17 integers:" << endl;
29 inputVector( integers1 );
30 inputVector( integers2 );
31
32 cout << "\nAfter input, the vectors contain:\n"
33     << "integers1:" << endl;
34 outputVector( integers1 );
35 cout << "integers2:" << endl;
36 outputVector( integers2 );
37
38 // use inequality (!=) operator with vector objects
39 cout << "\nEvaluating: integers1 != integers2" << endl;
40
41 if ( integers1 != integers2 )
42     cout << "integers1 and integers2 are not equal" << endl;
43
```

---

**Fig. 7.25** | Demonstrating C++ Standard Library class template vector. (Part 2 of 7.)

---

```
44 // create vector integers3 using integers1 as an
45 // initializer; print size and contents
46 vector< int > integers3( integers1 ); // copy constructor
47
48 cout << "\nSize of vector integers3 is " << integers3.size()
49     << "\nvector after initialization:" << endl;
50 outputVector( integers3 );
51
52 // use overloaded assignment (=) operator
53 cout << "\nAssigning integers2 to integers1:" << endl;
54 integers1 = integers2; // assign integers2 to integers1
55
56 cout << "integers1:" << endl;
57 outputVector( integers1 );
58 cout << "integers2:" << endl;
59 outputVector( integers2 );
60
61 // use equality (==) operator with vector objects
62 cout << "\nEvaluating: integers1 == integers2" << endl;
63
64 if ( integers1 == integers2 )
65     cout << "integers1 and integers2 are equal" << endl;
66
```

---

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

---

```
67 // use square brackets to use the value at location 5 as an rvalue
68 cout << "\nintegers1[5] is " << integers1[ 5 ];
69
70 // use square brackets to create lvalue
71 cout << "\n\nAssigning 1000 to integers1[5]" << endl;
72 integers1[ 5 ] = 1000;
73 cout << "integers1:" << endl;
74 outputVector( integers1 );
75
76 // attempt to use out-of-range subscript
77 try
78 {
79     cout << "\nAttempt to display integers1.at( 15 )" << endl;
80     cout << integers1.at( 15 ) << endl; // ERROR: out of range
81 } // end try
82 catch ( out_of_range &ex )
83 {
84     cerr << "An exception occurred: " << ex.what() << endl;
85 } // end catch
```

---

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

---

```
86
87     // changing the size of a vector
88     cout << "\nCurrent integers3 size is: " << integers3.size() << endl;
89     integers3.push_back( 1000 ); // add 1000 to the end of the vector
90     cout << "New integers3 size is: " << integers3.size() << endl;
91     cout << "integers3 now contains: ";
92     outputVector( integers3 );
93 } // end main
94
95 // output vector contents
96 void outputVector( const vector< int > &array )
97 {
98     for ( int item : items )
99         cout << item << " ";
100
101     cout << endl;
102 } // end function outputVector
103
104 // input vector contents
105 void inputVector( vector< int > &array )
106 {
107     for ( int &item : items )
108         cin >> item;
109 } // 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 `arrays`, 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 `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 `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`.
- Line 92 then displays `integers3`'s new contents.

## 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 `vectors` (and other C++ Standard Library data structures).
- At the time of this writing, list initializers were not yet supported for `vectors` in Visual C++.