15 Standard Library Containers and Iterators

Objectives

In this chapter you'll:

- Be introduced to the Standard Library containers, iterators and algorithms.
- Use the `vector`, `list` and `deque` sequence containers.
- Use the `set`, `multiset`, `map` and `multimap` associative containers.
- Use the `stack`, `queue` and `priority_queue` container adapters.
- Use iterators to access container elements.
- Use the `copy` algorithm and `ostream_iterator`s to output a container.
- Understand how to use the `bitset` "near container" to manipulate a collection of bit flags.

Self-Review Exercises

15.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Pointer-based code is complex and error prone—the slightest omissions or oversights can lead to serious memory-access violations and memory-leak errors that the compiler will warn you about.

ANS: False. The compiler does not warn about these kinds of execution-time errors.

b) `deque`s offer rapid insertions and deletions at front or back and direct access to any element.

ANS: True.

c) `list`s are singly linked lists and offer rapid insertion and deletion anywhere.

ANS: False. They are doubly linked lists.

d) `multimap`s offer one-to-many mapping with duplicates allowed and rapid key-based lookup.

ANS: True.

e) Associative containers are nonlinear data structures that typically can locate elements stored in the containers quickly.

ANS: True.

f) The container member function `cbegin` returns an `iterator` that refers to the container's first element.

ANS: False. It returns a `const_iterator`.

g) The `++` operation on an iterator moves it to the container's next element.

ANS: True.

h) The `*` (dereferencing) operator when applied to a `const` iterator returns a `const` reference to the container element, allowing the use of non-`const` member functions.

ANS: False. Disallowing the use of non-`const` member functions.

i) Using `iterator`s where appropriate is another example of the principle of least privilege.

ANS: False. Using `const_iterator`s where appropriate is another example of the principle of least privilege.

j) Many algorithms operate on sequences of elements defined by iterators pointing to the first element of the sequence and to the last element.

ANS: False. Many algorithms operate on sequences of elements defined by iterators pointing to the first element of the sequence and to one element past the last element.

k) Function `capacity` returns the number of elements that can be stored in the `vector` before the `vector` needs to dynamically resize itself to accommodate more elements.

ANS: True.

l) One of the most common uses of a deque is to maintain a first-in, first-out queue of elements. In fact, a deque is the default underlying implementation for the queue adaptor.

ANS: True.

m) `push_front` is available only for class `list`.

ANS: False. It's also available for class `deque`.

n) Insertions and deletions can be made only at the front and back of a `map`.

ANS: False. Insertions and deletions can be made anywhere in a `map`.

o) Class `queue` enables insertions at the front of the underlying data structure and deletions from the back (commonly referred to as a first-in, first-out data structure).

ANS: False. Insertions may occur only at the back and deletions may occur only at the front.

15.2 Fill in the blanks in each of the following statements:

a) The three key components of the "STL" portion of the Standard Library are ________, ________ and ________.

ANS: containers, iterators and algorithms.

b) Built-in arrays can be manipulated by Standard Library algorithms, using ________ as iterators.

ANS: pointers.

c) The Standard Library container adapter most closely associated with the last-in, first-out (LIFO) insertion-and-removal discipline is the ________.

ANS: `stack`.

d) The sequence containers and ________ containers are collectively referred to as the first-class containers.

ANS: associative.

e) A(n) ________ constructor initializes the container to be a copy of an existing container of the same type.

ANS: copy.

f) The ________ container member function returns `true` if there are no elements in the container; otherwise, it returns `false`.

ANS: `empty`.

g) The ________ container member function (C++11) moves the elements of one container into another—this avoids the overhead of copying each element of the argument container.

ANS: move version of `operator=`.

h) The container member function ________ is overloaded to return either an `iterator` or a `const_iterator` that refers to the first element of the container.

ANS: `begin`.

i) Operations performed on a `const_iterator` return _______ to prevent modification to elements of the container being manipulated.

ANS: `const` references.

j) The sequence containers are `array`, `vector`, `deque`, ________ and ________.

ANS: `list` and `forward_list`.

k) Choose the ________ container for the best random-access performance in a container that can grow.

ANS: `vector`.

l) Function `push_back`, which is available in sequence containers other than ________, adds an element to the end of the container.

ANS: `array`.

m) As with `cbegin` and `cend`, C++11 includes `vector` member function `crbegin` and `crend` which return ________ that represent the starting and ending points when iterating through a container in reverse.

ANS: `const_reverse_iterators`.

n) A unary ________ function takes a single argument, performs a comparison using that argument and returns a `bool` value indicating the result.

ANS: predicate.

o) The primary difference between the ordered and unordered associative containers is ________.

ANS: the unordered ones do not maintain their keys in sorted order.

p) The primary difference between a `multimap` and a `map` is ________.

ANS: a `multimap` allows duplicate keys with associated values to be stored and a map allows only unique keys with associated values.

q) C++11 introduces class template `tuple`, which is similar to `pair`, but can ________.

ANS: hold any number of items of various types.

r) The `map` associative container performs fast storage and retrieval of unique keys and associated values. Duplicate keys are not allowed—a single value can be associated with each key. This is called a(n) ________ mapping.

ANS: one-to-one.

s) Class ________ provides functionality that enables insertions in sorted order into the underlying data structure and deletions from the front of the underlying data structure.

ANS: `priority_queue`.

15.3 Write a statement or expression that performs each of the following `bitset` tasks:

a) Write a declaration that creates `bitset` `flags` of size `size`, in which every bit is initially `0`.

ANS: `bitset<size> flags;`

b) Write a statement that sets bit `bitNumber` of `bitset` `flags` "off."

ANS: `flags.reset(bitNumber);`

c) Write a statement that returns a reference to the bit `bitNumber` of `bitset` `flags`.

ANS: `flags[bitNumber];`

d) Write an expression that returns the number of bits that are set in `bitset` `flags`.

ANS: `flags.count()`

e) Write an expression that returns `true` if all of the bits are set in `bitset` `flags`.

ANS: `flags.all()`

f) Write an expression that compares `bitsets` `flags` and `otherFlags` for inequality.

ANS: `flags != otherFlags`

g) Write an expression that shifts the bits in `bitset` `flags` left by `n` positions.

ANS: `flags <<= n;`

Exercises

NOTE: Solutions to the programming exercises are located in the **`ch15solutions`** *folder.*

15.4 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Many of the Standard Library algorithms can be applied to various containers independently of the underlying container implementation.

ANS: True.

b) `arrays` are fixed in size and offer direct access to any element.

ANS: True.

c) `forward_lists` are singly linked lists, that offer rapid insertion and deletion only at the front and the back.

ANS: False. They offer rapid insertion and deletion anywhere.

d) `sets` offer rapid lookup and duplicates are allowed.

ANS: False. No duplicates are allowed.

e) In a `priority_queue`, the lowest-priority element is always the first element out.

ANS: False. The highest priority element is always the first element out.

f) The sequence containers represent non-linear data structures.

ANS: False. They represent linear data structures.

g) As of C++11, there is now a non-member function version of `swap` that swaps the contents of its two arguments (which must be of different container types) using move operations rather than copy operations.

ANS: False. The two arguments must be of the same container type.

h) Container member function `erase` removes all elements from the container.

ANS: False. It removes one or more elements from the container

i) An object of type `iterator` refers to a container element that *can* be modified.

ANS: True.

j) We use `const` versions of the iterators for traversing read-only containers.
ANS: True.
k) For input iterators and output iterators, it's common to save the iterator then use the saved value later.
ANS: False. For input and output iterators, it's not possible to save the iterator then use the saved value later.
l) Class templates `array`, `vector` and `deque` are based on built-in arrays.
ANS: True.
m) Attempting to dereference an iterator positioned outside its container is a compilation error. In particular, the iterator returned by `end` should not be dereferenced or incremented.
ANS: False. It's a run-time logic error.
n) Insertions and deletions in the middle of a `deque` are optimized to minimize the number of elements copied, so it's more efficient than a `vector` but less efficient than a `list` for this kind of modification.
ANS: True.
o) Container `set` does *not* allow duplicates.
ANS: True.
p) Class `stack` (from header `<stack>`) enables insertions into and deletions from the underlying data structure at one end (commonly referred to as a last-in, first-out data structure).
ANS: True.
q) Function `empty` is available in all containers except the `deque`.
ANS: False. Function `empty` is available in all containers

15.5 Fill in the blanks in each of the following statements:
a) The three styles of container classes are first-class containers, ________ and near containers.
ANS: container adapters.
b) Containers are divided into four major categories—sequence containers, ordered associative containers, ________ and container adapters.
ANS: unordered associative containers.
c) The Standard Library container adapter most closely associated with the first-in, first-out (FIFO) insertion-and-removal discipline is the ________.
ANS: `queue`.
d) Built-in arrays, `bitset`s and `valarray`s are all ________ containers.
ANS: near.
e) A(n) ________ constructor (new in C++11) moves the contents of an existing container of the same type into a new container, without the overhead of copying each element of the argument container.
ANS: move.
f) The ________ container member function returns the number of elements currently in the container.
ANS: `size`
g) The ________ container member function returns `true` if the contents of the first container are not equal to the contents of the second; otherwise, returns `false`.
ANS: `operator!=`
h) We use iterators with sequences—these can be input sequences or output sequences, or they can be ________.
ANS: in containers.

i) The Standard Library algorithms operate on container elements only indirectly via ________.

ANS: iterators.

j) Applications with frequent insertions and deletions in the middle and/or at the extremes of a container normally use a(n) ________ .

ANS: `list`.

k) Function ________ is available in *every* first-class container (except `forward_list`) and it returns the number of elements currently stored in the container.

ANS: `size`.

l) It can be wasteful to double a `vector`'s size when more space is needed. For example, a full `vector` of 1,000,000 elements resizes to accommodate 2,000,000 elements when a new element is added, leaving 999,999 unused elements. You can use ________ and ________ to control space usage better.

ANS: `resize`, `reserve`.

m) As of C++11, you can ask a `vector` or `deque` to return unneeded memory to the system by calling member function ________.

ANS: `shrink_to_fit`

n) The associative containers provide direct access to store and retrieve elements via keys (often called search keys). The ordered associative containers are `multiset`, `set`, ________ and ________ .

ANS: `multimap` and `map`.

o) Classes ________ and ________ provide operations for manipulating sets of values where the values are the keys—there is *not* a separate value associated with each key.

ANS: `multiset`, `set`.

p) We use C++11's `auto` keyword to ________ .

ANS: infer the variable's type from its initializer.

q) A `multimap` is implemented to efficiently locate all values paired with a given ________.

ANS: key.

r) The Standard Library container adapters are `stack`, `queue` and ________.

ANS: `priority_queue`.

Discussion Questions

15.6 Why is it expensive to insert (or delete) an element in the middle of a `vector`?

ANS: The entire portion of the `vector` after the insertion (or deletion) point must be moved, because `vector` elements occupy contiguous cells in memory.

15.7 Containers that support random-access iterators can be used with most but not all Standard Library algorithms. What is the exception?

ANS: If an algorithm modifies a container's size, the algorithm can't be used on built-in arrays or `array` objects.

15.8 Why would you use operator `*` to dereference an iterator?

ANS: So that you can use the element to which it points.

15.9 Why is insertion at the back of a `vector` efficient?

ANS: The `vector` simply grows, if necessary, to accommodate the new item.

15.10 When would you use a `deque` in preference to a `vector`?

ANS: Applications that require frequent insertions and deletions at both ends of a container normally use a `deque` rather than a `vector`. Although we can insert and delete elements at the front and back of both a `vector` and a `deque`, class `deque` is more efficient than `vector` for doing insertions and deletions at the front.

15.11 Describe what happens when you insert an element in a vector whose memory is exhausted.

ANS: The `vector` allocates a larger contiguous area of memory, copies the original elements into the new memory and deallocates the old memory.

15.12 When would you prefer a `list` to a `deque`?

ANS: Class `deque` is implemented for efficient insertion and deletion operations at its front and back, much like a list, but a list is also capable of efficient insertions and deletions in the `middle` of the `list`.

15.13 What happens when the map subscript is a key that's not in the map?

ANS: When the `map` subscript is a key that's already in the `map`, the operator returns a reference to the associated value. When the subscript is a key that's not in the `map`, the operator inserts the key in the `map` and returns a reference that can be used to associate a value with that key.

15.14 Use C++11 list initializers to initialize the `vector names` with the `strings "Suzanne"`, `"James"`, `"Maria"` and `"Juan"`. Show both common syntaxes.

ANS:
```
vector< string > names{"Suzanne", "James", "Maria", "Juan"};
```

15.15 What happens when you erase a container element that contains a pointer to a dynamically allocated object?

ANS: Erasing an element that contains a pointer to a dynamically allocated object does not delete that object—this can lead to a memory leak. If the element is a `unique_ptr`, the memory would be deleted. If the element is a `shared_ptr`, the reference count to the dynamically allocated object would be decremented and the memory would be deleted only if the reference count reached `0`.

15.16 Describe the `multiset` ordered associative container.

ANS: The `multiset` ordered associative container (from header `<set>`) provides fast storage and retrieval of keys and allows duplicate keys. The elements' ordering is determined by a so-called comparator function object. For example, in an integer `multiset`, elements can be sorted in ascending order by ordering the keys with comparator function object `less<int>`.

15.17 How might a `multimap` ordered associative container be used in a credit-card transaction processing system?

ANS: In a credit-card transaction-processing system, one credit-card account can have many associated transactions; in a university, one student can take many courses, and one professor can teach many students; in the military, one rank (like "private") has many people.

15.18 Write a statement that creates and initializes a multimap of strings and ints with three key–value pairs.

ANS:
```
multimap<string, int, less<string>> pairs{
   {"Sam", 2}, {"Judy", 9}, {"Jerry", 7}};
```

15.19 Explain the `push`, `pop` and `top` operations of a `stack`.

ANS: `push` inserts an element at the top of the `stack`. `pop` removes the top element of the `stack`. `top` gets a reference to the top element of the `stack`.

15.20 Explain the `push`, `pop`, `front` and `back` operations of a `queue`.

ANS: `push` inserts an element at the back of the `queue`. `pop` removes the element at the front of the `queue`. `front` gets a reference to the first element in the `queue`. `back` gets a reference to the last element in the `queue`.

15.21 How does inserting an item in a `priority_queue` differ from inserting an item in virtually any other container?

ANS: `push` inserts an item at the appropriate location based on priority order of the `priority_queue`.