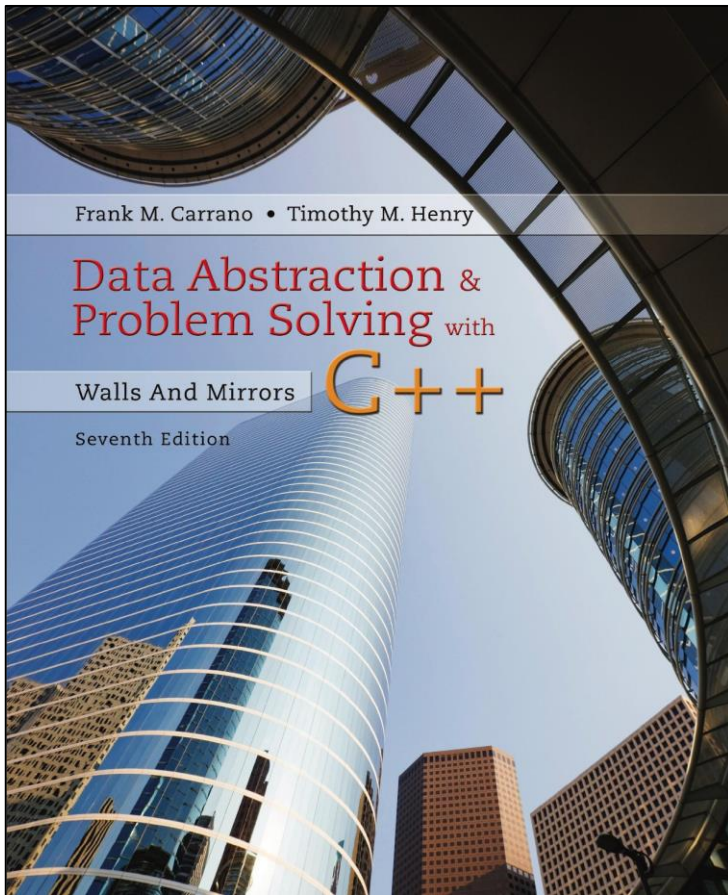


Data Abstraction & Problem Solving with C++: Walls and Mirrors

Seventh Edition



C++ Interlude 8

The Standard Template Library

STL Containers (1 of 6)

- C++ has a library of container classes
 - In form of class templates
 - Defined as Standard Template Library (STL)
- So why does this text develop ADTs?
 - Learn how to develop ADTs not found in STL
 - STL not part of class hierarchy
 - Working in a language without STL

STL Containers (2 of 6)

- Types of containers using STL
 - Sequence containers
 - Associative containers
 - Container adapters
- Operations common to all STL containers
 - Constructor, destructor
 - **operator =**
 - **bool** empty()
 - **unit** size()

Sequence Containers (1 of 8)

- STL **array** Operations
 - value_type& front()
 - value_type& back()
 - value_type& at(size_type n)
 - **void fill(const value_type& val)**
 - iterator begin()
 - iterator end()
 - reverse_iterator rbegin()
 - reverse_iterator rend()

Sequence Containers (2 of 8)

- Operations common to STL sequence containers
 - `value_type& front()`
 - `value_type& back()`
 - **`void push_back(value_type& item)`**
 - **`void pop_back(value_type& item)`**
 - **`void resize(uint newSize)`**
 - **`void clear()`**

Sequence Containers (3 of 8)

- **void insert(uint position, value_type& item)**
- **void insert(iterator itPosition, value_type& item)**
- **void erase(uint position)**
- **void erase(iterator itPosition)**

Sequence Containers (4 of 8)

- `iterator begin()`
- `iterator end()`
- `reverse_iterator rbegin()`
- `reverse_iterator rend()`

Sequence Containers (5 of 8)

- Additional STL **vector** Operation
 - `value_type& at(size_type n)`
- Additional STL **deque** Operations
 - `value_type& at(size_type n)`
 - **void** `push_front(value_type& item)`
 - **void** `pop_front(value_type& item)`

Sequence Containers (6 of 8)

- Additional STL **list** and **forward_list** Operations
 - **void** push_front(value_type& item)
 - **void** pop_front(value_type& item)
 - **void** remove(value_type& val)
 - **void** sort()
 - **void** merge(list<value_type>& rhs)
 - **void** slice(iterator position, list<value_type>& rhs)
 - **void** reverse()

Sequence Containers (7 of 8)

Listing C8-2 Example of using the STL list

```
1  #include <iostream>
2  #include <string>
3  #include <list>
4
5  int main()
6  {
7      std::list<string> groceryList; // Create an empty list
8      std::list<string>::iterator myPosition = groceryList.begin();
9
10     groceryList.insert(myPosition, "apples");
11     groceryList.insert(myPosition, "bread");
12     groceryList.insert(myPosition, "juice");
13     groceryList.insert(myPosition, "carrots");
14
15     std::cout << "Number of items on my grocery list: "
16               << groceryList.size() << std::endl;
17 }
```

Sequence Containers (8 of 8)

Listing C8-2 [Continued]

```
17  
18     groceryList.sort();  
19  
20     std::cout << "Items are:" << std::endl;  
21     for (auto groceryItem : groceryList)  
22     {  
23         std::cout << groceryItem << std::endl;  
24     } // end for  
25 } // end main
```

Output

```
Number of items on my grocery list: 4  
Items are:  
apples  
bread  
carrots  
juice
```

Associative Containers (1 of 6)

- Operations Common to the STL **set** and **multiset**
 - **void** clear()
 - **void** insert(value_type& item)
 - **void** erase(value_type& item)
 - **void** erase(iterator& position)
 - iterator find(value_type& item)
 - **uint** count(value_type& item)

Associative Containers (2 of 6)

- iterator `lower_bound(value_type& item)`
- iterator `upper_bound(value_type& item)`
- iterator `begin()`
- iterator `end()`
- reverse_iterator `rbegin()`
- reverse_iterator `rend()`

Associative Containers (3 of 6)

- Operations Common to the STL **map** and **multimap**
 - **void** clear()
 - **void** insert(pair_type& item)
 - **uint** erase(key_type& item)
 - void erase(iterator& position)
 - iterator find(key_type& item)
 - **uint** count(key_type& item)

Associative Containers (4 of 6)

- iterator `lower_bound(key_type& item)`
- iterator `upper_bound(key_type& item)`
- iterator `begin()`
- iterator `end()`
- reverse_iterator `rbegin()`
- reverse_iterator `rend()`

Associative Containers (5 of 6)

Listing C8-3 Alternative definition of a hashing function

```
1  #include <iostream>
2  #include <string>
3  #include <unordered_map>
4
5  // Create a type since this is a long name to use (optional)
6  typedef std::unordered_map<std::string, int> StringKeyMap;
7
8  // Create a dummyMap object so we can get its hash function
9  StringKeyMap dummyMap;
10
11 // Capture the hash function for use in program
12 StringKeyMap::hasher myHashFunction = dummyMap.hash_function();
13
```


Associative Containers (6 of 6)

Listing C8-3 [Continued]

```
13
14  int main()
15  {
16      std::cout << "Hashing a String: " << myHashFunction("Hashing a String:")
17              << std::endl;
18      std::cout << "Smashing a String: " << myHashFunction ("Smashing a String:")
19              << std::endl;
20      return 0;
21  } // end main
```

Output

```
Hashing a String: 2084157801917477989
Smashing a String: 14048775086903850803
```

STL Containers (3 of 6)

- STL **stack** operations
 - value_type& top()
 - **void** push(value_type& item)
 - **void** pop()
- STL **queue** operations
 - value_type& front()
 - value_type& back()
 - **void** push(value_type& item)
 - **void** pop()

STL Containers (4 of 6)

- STL **priority_queue** operations
 - `value_type& top()`
 - **void** `push(value_type& item)`
 - **void** `pop()`

STL Containers (5 of 6)

Listing C8-1 Example use of the STL stack

```
1  #include <iostream>
2  #include <stack>
3
4  int main()
5  {
6      std::stack<int> aStack;
7
8      // Right now, the stack is empty
9      if (aStack.empty())
10         std::cout << "The stack is empty." << std::endl;
11
```

STL Containers (6 of 6)

Listing C8-1 (continued)

```
11
12     for (int j = 0; j < 5; j++)
13         aStack.push(j); // Places items on top of stack
14
15     while (!aStack.empty())
16     {
17         std::cout << aStack.top() << " ";
18         aStack.pop();
19     } // end while
20
21     return 0;
22 } // end main
```

Output

The stack is empty.
4 3 2 1 0

STL Algorithms (1 of 7)

- STL Search and compare Algorithms
 - **void** for_each(iterator start, iterator end, Function fun)
 - iterator find(iterator start, iterator end, value_type& val)
 - iterator find_if(iterator start, iterator end, PredFunction fun)
 - **uint** count(iterator start, iterator end, value_type& val)

STL Algorithms (2 of 7)

- **uint** count_if(iterator start, iterator end, PredFunction fun)
- **bool** equal(iterator start1, iterator end1, iterator start2)
- **value_type&** min(value_type& item1, value_type& item2)

STL Algorithms (3 of 7)

- `value_type& min_element(iterator start, iterator end)`
- `value_type& max(value_type& item1, value_type& item2)`
- `value_type& max_element(iterator start, iterator end)`

STL Algorithms (4 of 7)

- STL sequence modification algorithms
 - `iterator copy(iterator start1, iterator end1, iterator start2)`
 - `iterator copy_backward(iterator start1, iterator end1, iterator start2)`
 - **`void swap(value_type& item1, value_type& item2)`**

STL Algorithms (5 of 7)

- `iterator transform(iterator start1, iterator end1, iterator start2, UnaryOperator op)`
- `iterator transform(iterator start1, iterator end1, iterator operand2, iterator start2, BinaryOperator bop)`
- **`void fill(iterator start1, iterator end1, value_type& val)`**

STL Algorithms (6 of 7)

- STL sorting and heap algorithms
 - **void** sort(iterator start, iterator end)
 - **void** stable_sort(iterator start, iterator end)
 - iterator partition(iterator start, iterator end, PredFunction fun)
 - iterator partition_stable(iterator start, iterator end, PredFunction fun)
 - **void** nth_element(iterator start, iterator nth, iterator end)

STL Algorithms (7 of 7)

- **void** make_heap(iterator start, iterator end)
- **void** push_heap(iterator start, iterator end)
- **void** pop_heap(iterator start, iterator end)
- **void** sort_heap(iterator start, iterator end)

Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

Summary

- Sequence containers: linear data structures
 - vector, list, deque
- Associative containers: non-linear containers that can locate elements quickly
 - set, multiset, map, multimap
- Container adapters: constrained versions of sequence containers
 - stack, queue, priority_queue

Summary

STL Container	Header File	Applications
vector	<vector>	For direct access to any element, and quick insertion and deletion at the end of the vector.
deque	<deque>	For direct access to any element, quick insertion and deletion at the front and end of the deque.
list	<list>	For rapid insertion and deletion anywhere.
set	<set>	For direct lookup, no duplicated elements.
multiset	<set>	Same as set except that duplicated elements allowed.
map	<map>	Key/value pair mapping, no duplicates allowed, and quick lookup using the key.
multimap	<map>	Same as map, except that keys may be duplicated
stack	<stack>	Last-in/first-out container.
queue	<queue>	First-in/first-out container.
priority_queue	<queue>	The highest-priority element is removed first.

Summary

STL Container	Type of Iterators Supported
vector	random access iterators
deque	random access iterators
list	bidirectional iterators
set	bidirectional iterators
multiset	bidirectional iterators
map	bidirectional iterators
multimap	bidirectional iterators
stack	no iterator support
queue	no iterator support
priority_queue	no iterator support