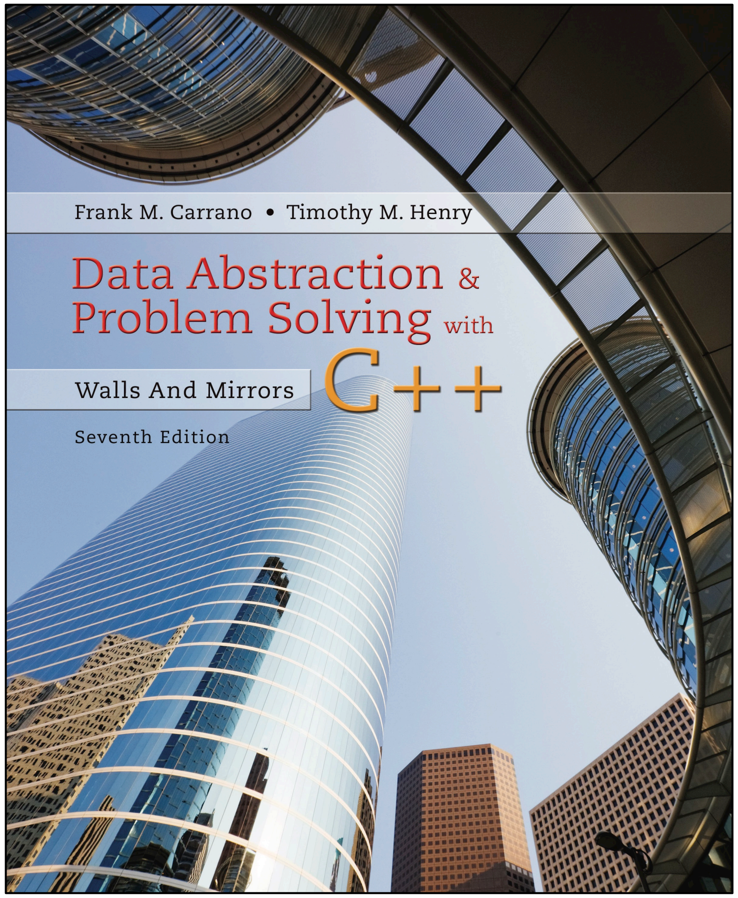
Data Abstraction & Problem Solving with C++: Walls and Mirrors

Seventh Edition


Frank M. Carrano • Timothy M. Henry
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
  - Container adapters
  - Sequence containers
  - Associative containers
- Operations common to all STL containers
  - Constructor, destructor
  - **`operator`** `=`
  - **`bool`** `empty()`
  - **`unit`** `size()`

STL Containers (3 of 6)

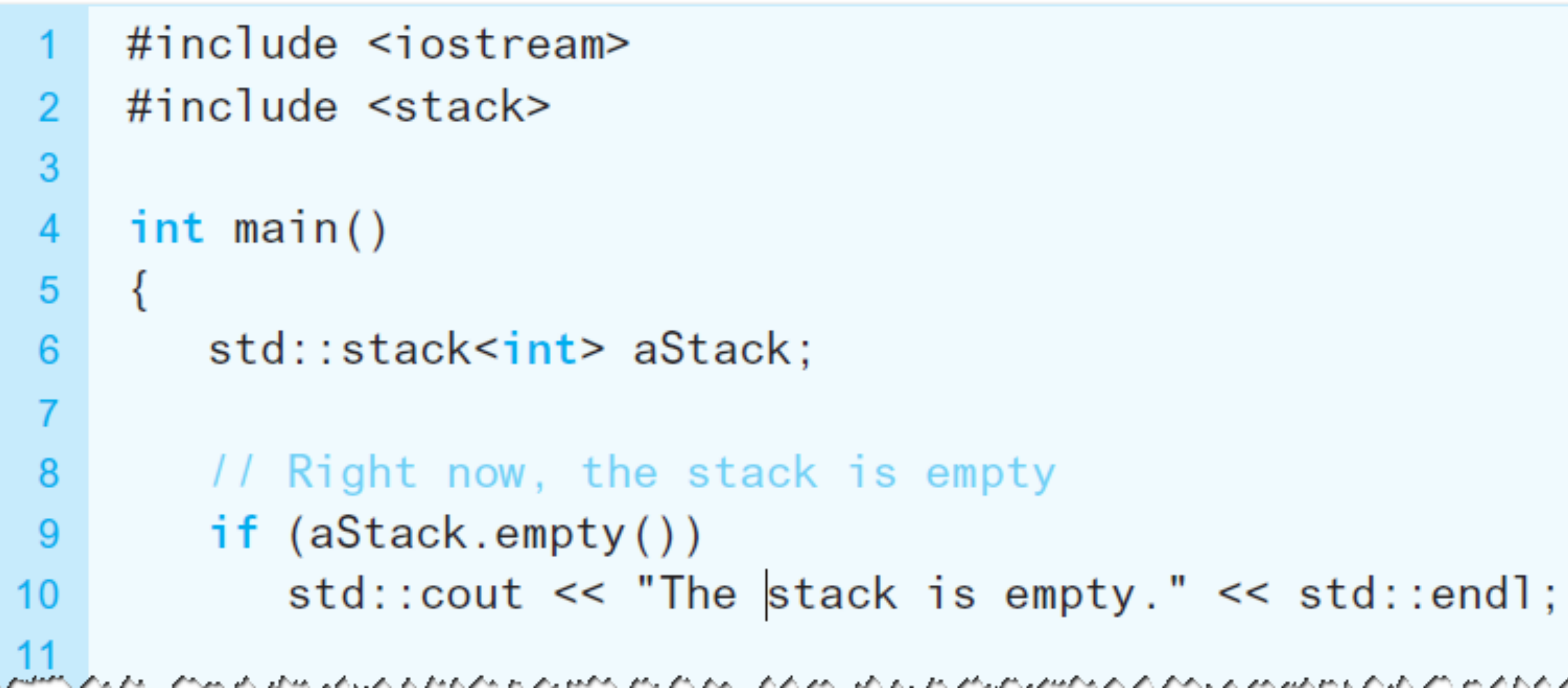
- STL **`stack`** operations
  - `value_type& top()`
  - **`void`** `push(value_type& item)`
  - **`void`** `pop()`
- STL **`queue`** operations
  - `value_type& front()`
  - `value_type& back()`
  - **`void`** `push(value_type& item)`
  - **`void`** `pop()`

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**



```
#include <iostream>
#include <stack>

int main()
{
    std::stack<int> aStack;

    // Right now, the stack is empty
    if (aStack.empty())
        std::cout << "The stack is empty." << std::endl;

```

STL Containers (6 of 6)

Listing C8-1 (continued)

```
    for (int j = 0; j < 5; j++)
       aStack.push(j); // Places items on top of stack

    while (!aStack.empty())
    {
       std::cout << aStack.top() << " ";
       aStack.pop();
    }  // end while

    return 0;
 }  // end main
```

Output

```
The stack is empty.
 4 3 2 1 0
```

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**

```
#include <iostream>
#include <string>
#include <list>

int main()
{
   std::list<string> groceryList; // Create an empty list
   std::list<string>::iterator myPosition = groceryList.begin();

   groceryList.insert(myPosition, "apples");
   groceryList.insert(myPosition, "bread");
   groceryList.insert(myPosition, "juice");
   groceryList.insert(myPosition, "carrots");

   std::cout << "Number of items on my grocery list: "
             << groceryList.size() << std::endl;

```

Sequence Containers (8 of 8)

Listing C8-2 [Continued]

```cpp

    groceryList.sort();

    std::cout << "Items are:" << std::endl;
    for (auto groceryItem : groceryList)
    {
        std::cout << groceryItem << std::endl;
    }   // end for
}   // 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

```
#include <iostream>
#include <string>
#include <unordered_map>

// Create a type since this is a long name to use (optional)
typedef std::unordered_map<std::string, int> StringKeyMap;

// Create a dummyMap object so we can get its hash function
StringKeyMap dummyMap;

// Capture the hash function for use in program
StringKeyMap::hasher myHashFunction = dummyMap.hash_function();

```

Associative Containers (6 of 6)

Listing C8-3 [Continued]

```cpp

int main()
{
    std::cout << "Hashing a String: " << myHashFunction("Hashing a String:")
              << std::endl;
    std::cout << "Smashing a String: " << myHashFunction ("Smashing a String:")
              << std::endl;
    return 0;
}  // end main
```

Output

```
Hashing a String: 2084157801917477989
Smashing a String: 1404877508690385080З
```

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)`