16. Standard *Template Library (STL)*

**Standard Template Library**

- Promotes Reuse
- Saves time and effort
- Better quality
- part of ANSI/ISO C++ standard
STL headers

• Defined in `std` namespace
• Headers are of form `<headername>` without the `.h`
  • Containers:
    – `<vector>`, `<list>`, `<deque>`, `<queue>`, `<stack>`, `<map>
    (also `multimap`), `<set>` (also `multiset`), `<bitset`
  • Iterators:
    – `<iterator>`
  • Algorithms:
    – `<algorithm>`
  • Strings:
    – `<string>`

STL Containers

• Sequence Containers
  – `vector` insert/delete at back, access any elem
  – `deque` insert/delete front/back, access any elem
    • Double ended queue (deck)
  – `list` doubly linked list, insert/delete any where

• Associative Containers
  – `set` lookup, no duplicates
  – `multiset` lookup, duplicates OK
  – `map` one-to-one mapping, no duplicates, key-based lookup
  – `multimap` one-to-many, duplicates OK, key-based lookup

• Container Adapters
  – `stack` Last-in-first-out
  – `queue` First-in-first-out
  – `priority_queue` highest priority element is first out
Container Types

• First-class Containers:
  – Sequence containers and Associative containers

• Near Containers (Almost containers)
  – arrays, string, bitset, valarray

Which one to choose?

• Standard Containers are interchangeable

• Each container is efficient for a certain usage
  – Key based look up: map
  – List operations: list
  – Add/remove elements at end: deque, stack, queue
  – Most common usage: vector
Common Operations and Members

• Common Member Types
  - value_type, allocator_type, size_type,
    difference_type, iterator, const_iterator,
    reverse_iterator, const_reverse_iterator, reference,
    const_reference, [for associative only: key_type,
    mapped_type, key_compare]

• Common Operations
  - Iterators
    • begin(), end(), rbegin(), rend()
  - Elements access
    • front(), back(), [not for list: [] (unchecked
      subscripting), at()]
  - Stack and Queue operations
    • push_back(), pop_back(), [list and deque only:
      push_front(), pop_front()]
  - List Operations
    • insert(), erase(), clear()
Iterators

- Iterators used to point to an element of a container
- Container’s
  - `begin()` returns an iterator pointing to the first element
  - `end()` returns an iterator pointing to the end (next to last)
- Use instance of iterator to point to element you want to modify
- Use instance of `const_iterator` to point to element you do not want to modify
- `*iter` refers to the pointed element
- `++iter` advances the iterator to point to the next element

Category of Iterators and Operations

- **Input**
  - `++itr`
  - `itr++`
  - `*itr (rvalue)`
  - `itr1 = itr2`
  - `itr1 == itr2`
  - `itr1 != itr2`

- **Output**
  - `++itr`
  - `itr++`
  - `*itr(lvalue)`

- **Forward**
  - `itr += i`
  - `itr -= i`
  - `itr [i]`
  - `itr1 <= itr2`
  - `itr1 >= itr2`

- **Bidirectional**
  - `--itr`
  - `itr--`

- **Random Access**
  - `itr += i`
  - `itr -= i`
  - `itr [i]`
  - `itr1 <= itr2`
  - `itr1 >= itr2`

*not all operations listed*
Iterators Aid Generic Programming

Iterators decouple Algorithms from Containers

![Diagram of Algorithm, Iterator, and Container]

Algorithms in STL

• Algorithms are used to manipulate containers
• Loosely coupled to the containers
• Works on Iterators returned by Containers
• Returns Iterator to element/result
• Examples: sort, search, delete, find, insert, etc.
STL Algorithms

- Mutating-sequence algorithms:
  - copy, copy_backward, fill, fill_n, generate, generate_n, iter_swap,
    partition, random_shuffle, remove, remove_copy, remove_copy_if,
    remove_if, replace, replace_copy, replace_copy_if, replace_if,
    reverse, reverse_copy, rotate, rotate_copy, stable_partition, swap,
    swap_ranges, transform, unique, unique_copy

- Non-mutating-sequence algorithms:
  - adjacent_find, count, count_if, equal, find, for_each, mismatch,
    search, search_n

- Numerical algorithms
  - accumulate, inner_product, partial_sum, adjacent_difference

Lab Work: Details provided on-line.