Tutorial Link https://course.testpad.chitkarauniversity.edu.in/tutorials/C++ Standard Template Library (STL): containers/5b5d3a6899e4fb423ae507bc

TUTORIAL

C++ Standard Template Library (STL): containers

C++98/03 has 11 containers: vector, stack, list, queue, deque, priority_queue, map, multimap, set, multiset and bitset. C++11 added forward_list, unordered_map, unordered_multimap, unordered_set and unordered_multiset, and moves bitset from container category to its own separate category. string is not really part of STL, but implemented many STL concepts.

The STL container are template class, which can be instantiated with a type. The actual type has to be copy constructable (having copy constructor) and assignable (overload = operator).

Sequence Container

A sequence container requires that elements are arranged in a strict linear order.

- `vector`: direct-access class-representation of dynamic array. Elements can be added and removed from the end in constant time. But insertion and removal not from the end require linear time. It supports direct-access, as well as bidirectional transversal.
- `deque`: (pronounced "deck") double-ended queue, allow elements to be added and removed from both the front and the rear, at constant time. The `deque` implementation in STL is similar to `vector`, which allows direct access. It is more complex than `vector`, as it allows constant-time insertion and removal from the front and rear; whereas vector only for rear.
- `list`: double-linked list that can be transverse in both direction. It support constant-time insertion and removal, but does not allow

direct (random) access with no indexing operator.
- `forward_list` (C++11): single-linked list that support forward transversal only. It is simpler than `list`.
- `queue`: allow elements to be added at the rear and removed from the front at constant time. STL's `queue` is very restrictive that it does not support direct access nor transversal through the elements.
- `priority_queue`: Similar to `queue`, but move the larger element to the front of the queue.
- `stack`: LIFO (last-in-first-out) queue, elements can be added and removed from the top-of-stack in a last-in-first-out manner. In STL, `stack` is an adapter class over the `vector` class. It is more restrictive than vector. Elements can only be accessed, added and removed from one-end (top-of-stack). It does not support direct access, nor transversal through all the elements.
- `array` (C++11): `array` is NOT a STL container because it has a fixed size and does not support operations like insert. But you can apply STL algorithms on `array` container.

Associative Containers

Associative container stores key-value pair or name-value pairs (i.e., associate a key with a value, and use a key to find the value). Associative container (or associative array) is actually similar to an array or vector, where a numerical index key is associated with a value, and you use the numerical key to find a value. Example of key-value pair are person-phone number(s), id-name, etc.

STL supports the following associative containers:

- `set`: the key is the same as the value. It does not allow duplicate values. STL `set` is sorted and reversible.
- `multiset`: allow duplicate values.
- `map`: key-value pair, where keys are unique. One key is associated with one value. STL `map` is sorted and reversible. It needs two types to instantiate: `map<`*`key-type, value-type`*`)`. To represent each key-value, STL provides a template class called `pair<class K, class V>`. You can instantiate the template class with specific key-type and value-type, e.g., `pair<const int, string>`.

- `multimap`: one key could be associated with multiple values.
- C++11 added 4 unordered associative containers: `unordered_set`, `unordered_multiset`, `unordered_map`, and `unordered_multimap`. These unordered associative containers are based on hash table (efficient in insertion, removal and search, but requires more storage spaces); whereas the ordered counterparts are based on tree.

First-class Containers, Adapters and Near Containers

The containers can also be classified as:

1. First-class Containers: all sequence containers and associative containers are collectively known as first-class container.
2. Adapters: constrained first-class containers such as `stack` and `queue`.
3. Near Containers: Do not support all the first-class container operations. For example, the built-in array (pointer-like), `bitsets` (for maintaining a set of flags), `valarray` (support array computation), `string` (stores only character type).

Container's Functions

All containers provides these functions:

- Default Constructor: to construct an empty container. Other constructors are provided for specific purposes.
- Copy Constructor:
- Destructor:
- `empty()`: returns true if the container is empty.
- `size()`: returns the size (number of elements).
- Assignment Operator (`=`)
- Comparison Operators (`==`, `!=`, `<`, `<=`, `>`, `>=`).
- `swap()`: exchanges the contents of two containers.

In addition, the first-class containers support these functions:

- `begin`, `end`, `cbegin`, `cend`: Returns the begin and end `iterator`, or the `const` version.
- `rbegin`, `rend`, `crbegin`, `crend`: Returns the `reverse_iterator`.

- `clear()`: Removes all elements.
- `erase()`: Removes one element given an iterator, or a range of elements given [begin, end) iterators.
- `max_size()`: Returns the maximum number of elements that the container can hold.

Container Header

- `<vector>`
- `<list>`
- `<deque>`
- `<queue>`: `queue` and `priority_queue`
- `<stack>`
- `<map>`: `map` and `multimap`
- `<set>`: `set` and `multiset`
- `<valarray>`
- `<bitset>`
- `<array>` (C++11)
- `<forward_list>` (C++11)
- `<unordered_map>` (C++11)
- `<unordered_set>` (C++11)