

Object-oriented programming

Week 9
Standard Template Library (STL)

7/2015

Agenda

- ☐ Introduction to STL
 - ☐ Sequence containers
 - ☐ Associative containers
 - ☐ Ordered sets
 - ☐ Container adaptors
 - ☐ Other special containers
 - ☐ **iterator**
-

Standard library of C++

<algorithm>	<ios>	<map>	<stack>
<bitset>	<iosfwd>	<memory>	<stdexcept>
<complex>	<iostream>	<new>	<streambuf>
<deque>	<istream>	<numeric>	<string>
<exception>	<iterator>	<ostream>	<typeinfo>
<fstream>	<limits>	<queue>	<utility>
<functional>	<list>	<set>	<valarray>
<iomanip>	<locale>	<sstream>	<vector>

STL: Standard Template Library

- ❑ Standard Template Library (STL) provides containers (i.e. data structures), algorithms and iterators to develop applications on C++
- ❑ STL was introduced by Alexander Stepanov for generic programming.
- ❑ The concepts of STL are developed independently from C++

STL (cont)

- ❑ Components in STL are not OOP but are generic programming
- ❑ Most containers are designed and implemented based on templates to handle different kinds of data types.
- ❑ Simple, powerful and efficient.
- ❑ The 2 most popular containers of STL are **vector** and **string**

Main components of STL

STL consists of 3 main components:

- ❑ Containers: data structures have been defined based on templates.
- ❑ Iterator: a pointer. It is used to access elements of a container.
- ❑ Algorithm: consists of popular algorithms, such as sorting, searching and others to deal with data...

STL containers

Containers can be grouped as

- ☐ Sequence containers
- ☐ Associative containers
- ☐ Ordered sets
- ☐ Container adapters
- ☐ Others

Sequence containers

- ❑ Those containers store elements by using a sequence
- ❑ Sequence containers:
 - **vector**
 - **deque**
 - **list**

Sequence containers: **vector**

- ❑ Using dynamic allocated array, allowing instant access to any element in the sequence.
- ❑ Adding or deleting the last element fast.
- ❑ Having out-of-range checking.

Sequence containers: **deque**

- ❑ Similar to **vector**: using a dynamic allocated array to handle the elements.
- ❑ Adding or deleting elements at 2 ends quickly (a little bit slower than **vector** because of handling both ends.)

cuu duong than cong . com

Sequence containers: **list**

- ❑ Using doubly-linked list to maintain the elements.
- ❑ There is no instant access to all the elements in the list like **vector**.
- ❑ Adding or deleting any element: fast!

cuu duong than cong . com

Associative containers

- ❑ Associative containers have key/value pairs:
 - Get the values via keys.
 - Elements sorted by keys.
 - Often implemented as a balance binary tree.
- ❑ There are two associative containers
 - **Map**
 - **Multimap**

Associative containers

- ❑ **map** allows users to access elements via keys of any data type. Map is a generalization of accessing elements via index **int** of **vector**.
- ❑ **multimap** is similar to **map** but it allows 1 key to map more than 1 element.

Ordered sets

- Sometimes they are classified as associative containers. They have the following characteristics:
 - Store elements in order
 - Often implemented by using balanced binary tree.
 - However, they don't have set operations (e.g. union...)

Ordered sets

□ **set**

- keep the elements in order when they are added.

- a set of unique objects.

□ **multiset** is similar to **set** but they allow duplicate objects.

Container adapters

- ❑ Those containers are built based on existing containers. They are different in the ways of accessing their elements.
- ❑ Because of applying different ways of accessing elements, those containers don't have **iterator**.

Container adapters

- ❑ **stack** only allows to access elements as LIFO (Last In, First Out).
- ❑ **queue**: FIFO (First In, First Out).
- ❑ **priority_queue** always return the top priority element.

cuu duong than cong . com

Other containers

- ❑ Those containers are implemented to represent a certain kind of data structure or have special functionality... com
- ❑ **string**: similar to **vector<char>** but it has special and useful methods/functions for operation on strings.ong . com

Other containers (cont.)

☐ **bitset**

- Data structure for storing bits effectively
- Special methods/functions for bits (AND, OR...)

☐ **valarray** is a special and efficient implementation of array. However, it doesn't have all the standard methods as other containers.

Member functions/methods of STL

□ All containers have:

- default copy constructor, destructor
- empty
- max_size, size
- Operators: = < <= > >= == !=
- swap

□ Only in sequence, associative containers and ordered sets

- begin, end
- rbegin, rend
- erase, clear

iterator

- ❑ **iterator** is similar to a pointer
 - Point to a element in a container
- ❑ Operators of an **iterator**
 - * dereference the element
 - ++ go to the next element
 - **begin()** returns the iterator of the first element
 - **end()** returns the iterator of the last element of the container.

Types of `iterator`

- ❑ **Input:** read the elements of a container, supports `++`, `+=` (increasing only).
E.g.: `istream_iterator`
 - ❑ **Output:** write the elements to a container, supports `++`, `+=` (increasing only).
E.g.: `ostream_iterator`
 - ❑ **Forward:** e.g. `hash_set<T>` iterator
 - Combination input iterator and output iterator
 - Multi-pass
-

Types of `iterator` (cont.)

- ❑ **Bi-directional:** similar to forward but can do (`--`, `--=`)

E.g.: `list<T>` iterator

- ❑ **Random access:** similar to bi-directional but can access to any element

E.g.: `vector<T>` iterator

Operators on `iterator`

- ❑ Input iterator: `++`, `=*p`, `->`, `==`, `!=`
- ❑ Output iterator: `++`, `*p=`, `p=p1`
- ❑ Forward iterator: for input và output iterator
- ❑ Bidirectional iterator: operators for forward and `--`
- ❑ Random access: operator for bidirectional and `+`, `+=`, `-`, `-=`, `>`, `>=`, `<`, `<=`, `[]`

Container supports the following **iterator**

- ☐ Sequence containers
 - **vector**: random access
 - **deque**: random access
 - **list**: bidirectional
- ☐ Associative containers: bidirectional
- ☐ Orderd sets: bidirectional
- ☐ Container adapters: don't have **iterator**
- ☐ **Bitset** and **valarray**: don't have **iterator**