C++ Program Design -- STL - Overview

Junjie Cao @ DLUT

Summer 2016

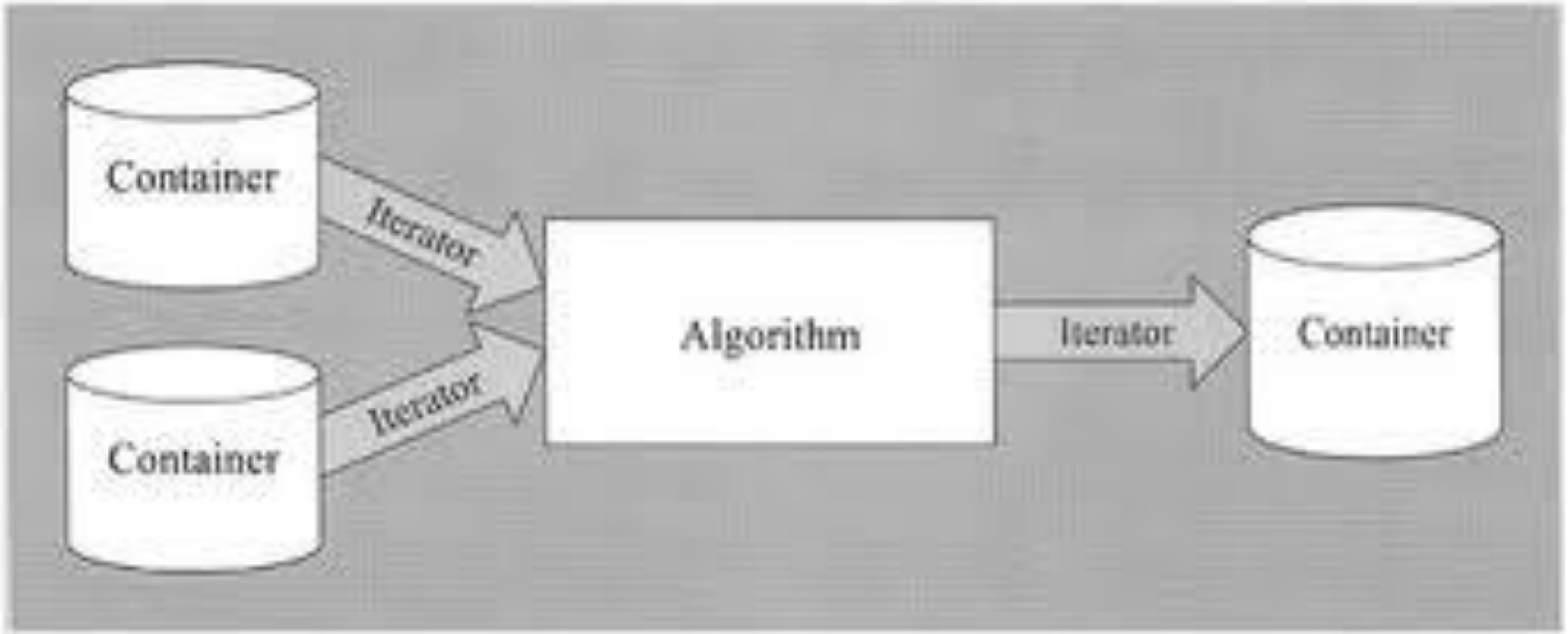
http://jjcao.github.io/cPlusPlus

Content

- C++ is about **efficient** programming with **abstractions**
- STL is a good example, let our programs Succinct, Abstract & efficient
  - Bookkeeping **details**
  - Take care of **memory**
  - Worry about **actual problem** we need to solve.

The standard library

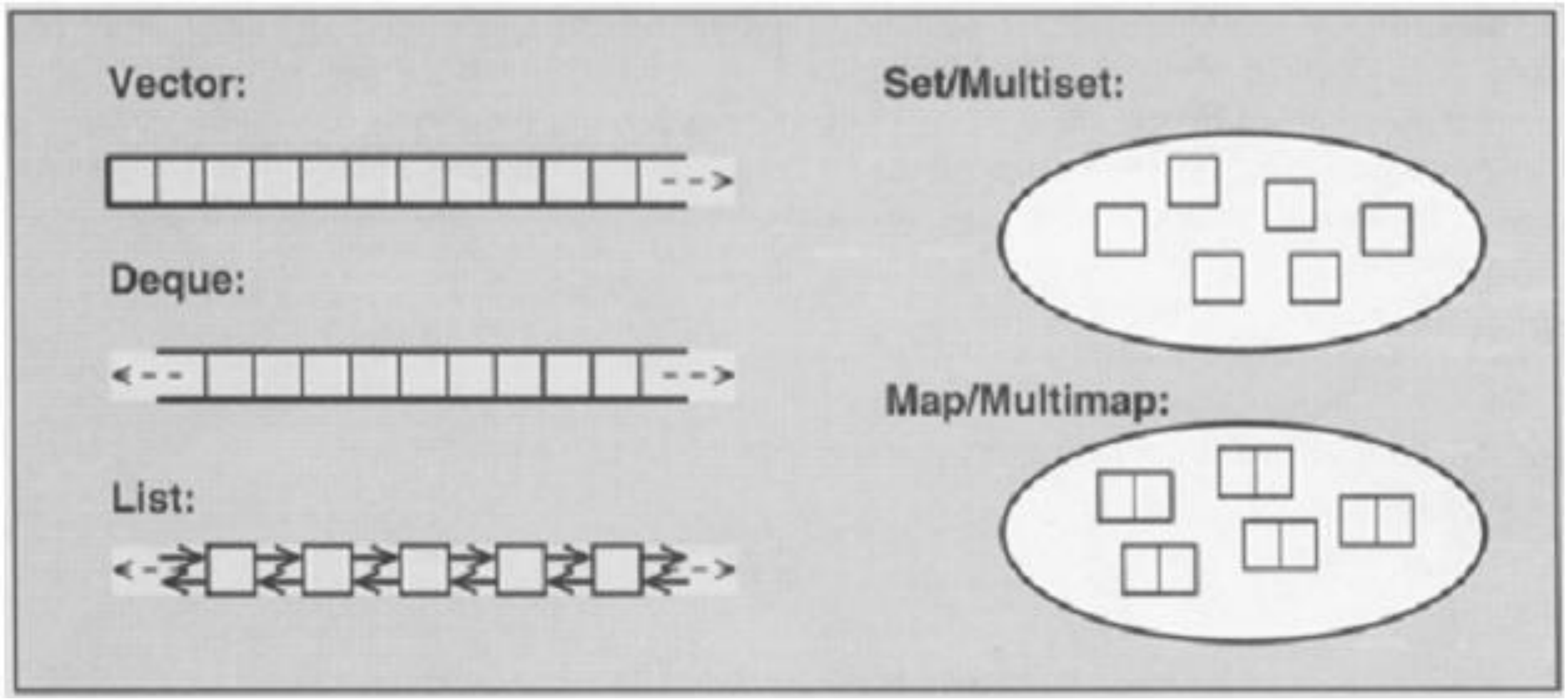
- Gentle introduction of STL:
  - **container classes**
  - **generic algorithms**



- OOP vs Generic Programming

Iterator: common interface for any arbitrary container type

STL containers overview



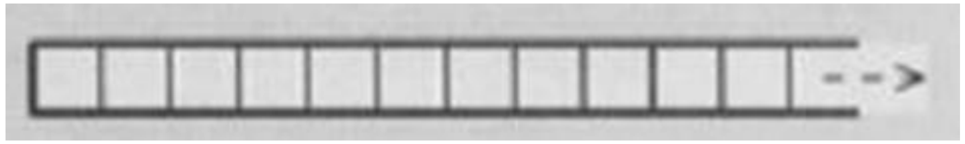
Sequence Containers

- **Sequence** containers are container classes that maintain the ordering of elements in the container.
- 6 sequence containers in STL:
  - std::vector, std::deque, std::array,
  - std::list,
  - std::forward_list, and std::basic_string
- **Associative** containers are containers that automatically sort their inputs when those inputs are inserted into the container.
- By default, associative containers compare elements using operator<.
  - std::set, std::multiset
  - std::map, std::multimap

- Note: ordering here is for the ordering of input

vector

a dynamic array

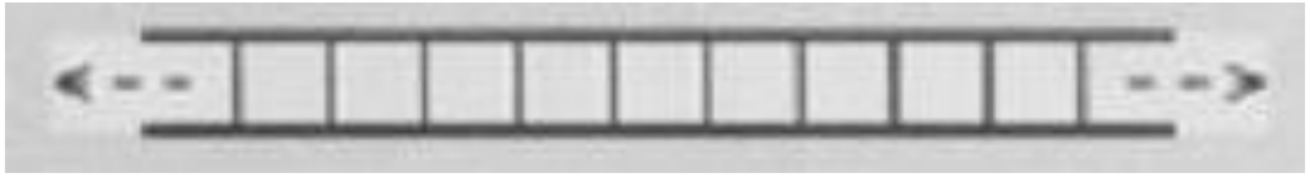


```
vector<int> vect;
for (int nCount=0; nCount < 6; nCount++)
    vect.push_back(10 - nCount); // insert at end of array

for (int nIndex=0; nIndex < vect.size(); nIndex++)
    cout << vect[nIndex] << ” ”;
```

The deque class (pronounced “deck”)

a double-ended queue class, implemented as a dynamic array that can g row from both ends.



```
deque<int> deq;
for (int nCount=0; nCount < 3; nCount++)
{
    deq.push_back(nCount); // insert at end of array
    deq.push_front(10 - nCount); // insert at front of array
}

for (int nIndex=0; nIndex < deq.size(); nIndex++)
    cout << deq[nIndex] << " ";
```

vector

- a special type of sequence container called a doubly linked list where e ach element in the container contains pointers that point at the next and previous elements in the list.
- no random access provided

```
list<char> coll; //list container for character elements
for (char c='a'; c<= ' z '; ++c) {// append elements from 'a' to 'z'
        coll.push_back(c);
}

while (! coll.empty()) {/* print all elements while there are elements*/
    cout << coll.front() << ' ';
    coll.pop_front(); // remove the first element
}
cout << endl;
```

STL iterators overview

- An **Iterator** is an object that can traverse (iterate over) a container class without the user having to know how the container is implemented.

```
vector<int>::const_iterator it;
it = vect.begin();
while (it != vect.end()){
    cout << *it << " ";
    ++it;
}
```

- `list<int>::const_iterator it; // declare an iterator`
- `it = li.begin(); // assign it to the start of the list`
- `set<int>::const_iterator it; // declare an iterator`
- `it = myset.begin();`

Iterating through a map

- use first() as the key, and second() as the value.

```
map<int, string> mymap;
mymap.insert(make_pair(4, "apple"));
mymap.insert(make_pair(2, "orange"));
mymap.insert(make_pair(1, "banana"));
map<int, string>::const_iterator it; // declare an iterator
it = mymap.begin(); // assign it to the start of the vector
while (it != mymap.end()) // while it hasn't reach the end
{
    cout << it->first << "=" << it->second << " "; // print the value
of the element it points to
    ++it; // and iterate to the next element
}
```

1=banana 2=orange 4=apple

STL algorithms overview

- In addition to container classes and iterators, STL also provides a number of generic algorithms for working with the elements of the container classes.
- These allow you to do things like search, sort, insert, reorder, remove, and copy elements of the container class.

min_element and max_element

```
#include <algorithm>
int main(){
    list<int> li;
    for (int nCount=0; nCount < 6; nCount++)
        li.push_back(nCount);

    list<int>::const_iterator it; // declare an iterator
    it = min_element(li.begin(), li.end());
        cout << *it << ” ”;

    it = max_element(li.begin(), li.end());
        cout << *it << ” ”;
    cout << endl;
}
```

find (and list::insert)

```
list<int> li;
for (int nCount=0; nCount < 6; nCount++)
    li.push_back(nCount);

list<int>::iterator it; // declare an iterator
it = find(li.begin(), li.end(), 3); // find the value 3 in the list
li.insert(it, 8); // use list::insert to insert the value 8 before it

for (it = li.begin(); it != li.end(); it++) // for loop with iterators
    cout << *it << " ";

cout << endl;
```

```
0 1 2 8 3 4 5
```

sort and reverse

```
vector<int> vect;          vect.push_back(7);
vect.push_back(-3);        vect.push_back(6);
vect.push_back(2);         vect.push_back(-5);
vect.push_back(0);         vect.push_back(4);

sort(vect.begin(), vect.end()); // sort the list

vector<int>::const_iterator it; // declare an iterator
for (it = vect.begin(); it != vect.end(); it++) // for loop with iterators
    cout << *it << " ";
cout << endl;

reverse(vect.begin(), vect.end()); // reverse the list
...
```

```
-5 -3 0 2 4 6 7
7 6 4 2 0 -3 -5
```