Object-Oriented Programming: Standard Template Library

Lectured by Ming-Te Chi 紀明德

First Semester, 2022

Computer Science Department
National Chengchi University

Slides credited from 李蔡彥 and 廖峻鋒

Standard Template Library (STL): Introduction

- STL: Platform-independent class library
  - Generic programming: do not depend on the underlying container.
  - Since ANSI/ISO C++ 98 standard.
- Pros:
  - generic type
  - reusable components
  - bug free (memory management)
- Cons:
  - slow compilation
  - large code
  - possibly slow execution

Three Key Components of STL

- Containers
  - vector, deque, list, set, multiset, map, multimap, stack, queue, priority_queue, etc.
- Iterators
  - input, output, forward, bidirectional, random access, etc.
- Algorithms
  - for_each, find, copy, fill, partition, random_shuffle, rotate, reverse, unique, swap, transform, etc.

Containers (I)

`vector`	similar to dynamic array.
`list`	a linked list, which is more efficient than an array for insertion/deletion
`stack`	a sequence which obeys last-in, first-out (LIFO) semantics
`queue`	s sequence which obeys first-in, first-out (FIFO) semantics
`deque`	a queue which allows insertion and removal of elements from either the head or tail. Similar to a `vector`, but is more efficient to insertion and remove at the ends.

Vector Example 1

```
#include <vector>
using namespace std;
int main(){
    vector<int> vec;    // 宣告一個裝 int 的 vector
                        // 現在 vec 是空的
    vec.push_back(10);
    vec.push_back(20);
    vec.push_back(30);  // vec 是 [10, 20, 30]

    int length = vec.size();        // length = 3
    for(int i=0 ; i<length ; i++){
        cout << vec[i] << endl;     // 輸出 10, 20, 30
    }
}
```

Vector Example 2

```
int main(){
    vector<int> vec;

    for(int i=0 ; i<5 ; i++){
        vec.push_back(i * 10);       // [0, 10, 20, 30, 40]
    }

    vec.pop_back();      // 移除 40
    vec.pop_back();      // 移除 30

    for(int i=0 ; i<vec.size() ; i++){  // vec.size() = 3
         cout << vec[i] << endl;        // 輸出 0, 10, 20
    }
}
```

List Example

```
int main() {
    list <int> gqlist;
    for (int i = 0; i < 5; ++i) {
        gqlist.push_back(i * 2);
    }
    cout << "\nList (gqlist) is : ";
    showlist(gqlist);
    cout << "\ngqlist.front() : " << gqlist.front();
    cout << "\ngqlist.back() : " << gqlist.back();
    cout << "\ngqlist.pop_front() : ";
    gqlist.pop_front();
    showlist(gqlist);
    return 0;
}
```

```
List (gqlist) is: 0 2 4 6 8
gqlist.front(): 0
gqlist.back(): 18
gqlist.pop_front(): 2 4 6 8
```

Queue Example

```
#include <queue>
using namespace std;
int main(){
    queue<int> q;   // an empty queue
    q.push(10);
    q.push(20);
    q.push(30);     // [10, 20, 30]

    cout << q.front() << endl;  // 10
    cout << q.back() << endl;   // 30

    q.pop();                    // [20, 30]
    cout << q.size() << endl;   // 2
}
```

Stack Example

```
#include <stack>
using namespace std;
int main(){
    stack<int> s;

    s.push(10);     //   | 30 |
    s.push(20);     //   | 20 |    疊三個盤子
    s.push(30);     //   |_10_|    10 在最下面

    for(int i=0 ; i<s.size() ; i++){    // s.size() = 3
        cout << s.top() << endl;
        s.pop();
    }                                   // 輸出 30, 20, 10
}
```

Containers (II)

`priority_queue`	a queue which dequeues elements according to their priority, rather than the order of insertion.
`map`	a sorted associative array which maps a key to a value. Work somewhat like a hash table, but doesn’t use hashing
`multimap`	similar to `map`, but allows duplicated keys
`set`	similar to `map`, but only stores the key.
`multiset`	similar to `set`, but allows duplicate keys.

Set Example

```
#include <set>
using namespace std;
int main(){
    set<int> mySet;
    mySet.insert(20);   // mySet = {20}
    mySet.insert(10);   // mySet = {10, 20}
    mySet.insert(30);   // mySet = {10, 20, 30}

    cout << mySet.count(20) << endl;    // 存在 -> 1
    cout << mySet.count(100) << endl;   // 不存在 -> 0

    mySet.erase(20);                    // mySet = {10, 30}
    cout << mySet.count(20) << endl;    // 0
}
```

Map Example

```
#include <map>
using namespace std;

int main(){
    map<string, int> m;     // 從 string 對應到 int

    m["one"] = 1;         // "one" -> 1
    m["two"] = 2;         // "two" -> 2
    m["three"] = 3;       // "three" -> 3

    cout << m["one"] << endl;       // 1
    cout << m["three"] << endl;     // 3
    cout << m["ten"] << endl;       // 0 (無對應值)
}
```

Iterators (I)

- Iterators provide a common interface to iterating over each element in a container class and allow for shared algorithms.

```
Example:
for(vector<int>::iterator iter=ages.begin();
   iter!=ages.end(); iter++)
   cout << *iter << '\n';
```

- The same code works if ages was a `set`, `map`, `deque`, or `list`.
- Types of iterators:
  - Forward
  - Bi-directional
  - Random-access

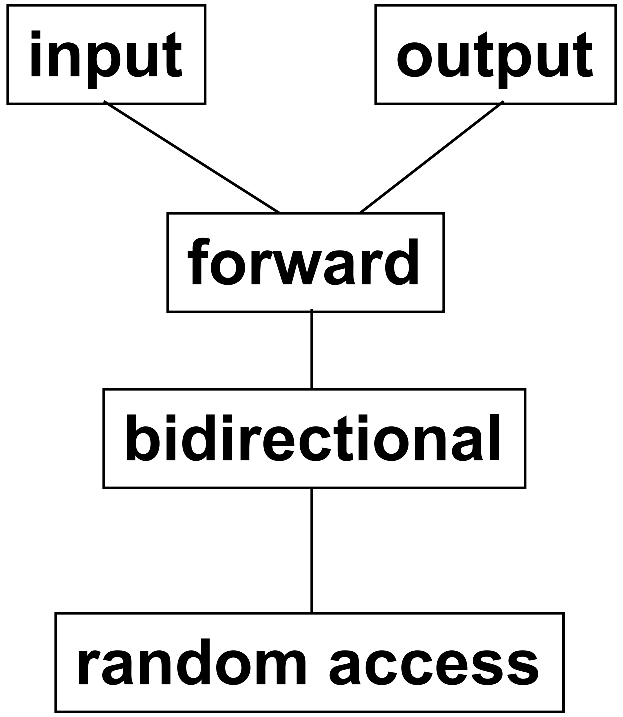
auto (c++11)

```
for(vector<int>::iterator iter=ages.begin();
    iter!=ages.end(); iter++)
    cout << *iter << '\n';
```

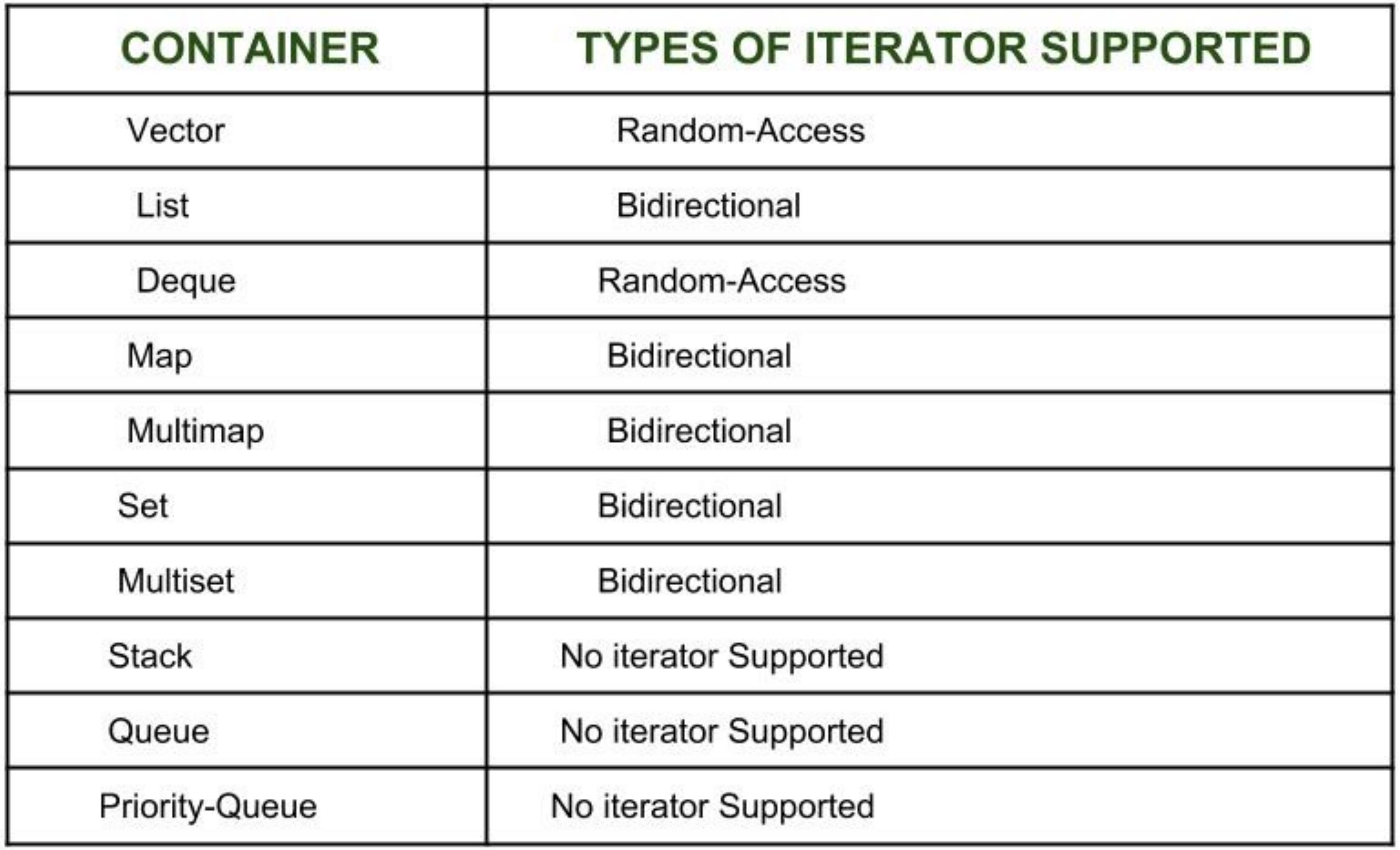
```
for(auto iter=ages.begin(); iter!=ages.end(); iter++)
    cout << *iter << '\n';
```

Iterators (II)

- Each of these iterators support
  - operator++: goes to the next element
  - operator*: return the current element
- Bidirectional also support
  - operator--
- Random access also support
  - operator+= and -=



Supported Iterators in Containers



CONTAINER	TYPES OF ITERATOR SUPPORTED
Vector	Random-Access
List	Bidirectional
Deque	Random-Access
Map	Bidirectional
Multimap	Bidirectional
Set	Bidirectional
Multiset	Bidirectional
Stack	No iterator Supported
Queue	No iterator Supported
Priority-Queue	No iterator Supported

Benefits of Iterators (1)

- Convenience in programming: code usability

```
int main() {
  vector<int> v = { 1, 2, 3 };
  vector<int>::iterator i; // Declaring an iterator
  int j;
  for (j = 0; j < 3; ++j) // without using iterators
     cout << v[j] << " ";
  for (i = v.begin(); i != v.end(); ++i) // using iterators
     cout << *i << " ";
  v.push_back(4); // adding an element
  for (i = v.begin(); i != v.end(); ++i) // same code
     cout << *i << " ";
  return 0;
}
```

```
1 2 3
1 2 3
1 2 3 4
```

Benefits of Iterators (2)

- Dynamic processing of the container
  - No need to handle element shifting

```
int main() {
  vector<int> v = { 1, 2, 3 };
  vector<int>::iterator i;
  int j;
  // Inserting element using iterators
  for (i = v.begin(); i != v.end(); ++i)
    if (i == v.begin())
        i = v.insert(i, 5); // inserting 5 at the beginning
  // v contains 5 1 2 3
  for (i = v.begin(); i != v.end(); ++i)
        cout << *i << " ";
}
```

```
5 1 2 3
```

Algorithms (I)

`for_each()`	execute an operation on each element of a collection
`find()`	find first occurrence of a particular value in a collection
`count()`	counts the number of occurrences of a particular value in a collection
`swap()`	swaps two elements in a collection
`replace()`	replaces all occurrences of a particular value with another
`remove()`	removes all occurrences of a particular value from a collection
`unique()`	removes duplicates from a collection
`reverse()`	reverses the elements in a collection

Algorithms (II)

`random_shuffle()`	randomly shuffles elements in a collection.
`sort()`	sorts a collection
`binary_search()`	find an element in a sorted collection
`set_operations()`	creates a collection based upon the set union, intersection, or difference of two collections.
`min_element()`	returns the minimum value in a collection
`max_element()`	returns the maximum value in a collection
`next_permutation()`	returns the next permutation of the elements in a collection.

STL Example 1 (1/3)

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
int main() {
  const int SIZE = 6;
  int a[SIZE] = {1, 2, 3, 4, 5, 6};
  vector<int> v(a, a+SIZE); //
  ostream_iterator<int> output(cout, " ");
  cout << "Vector v contains: ";
  copy (v.begin(), v.end(), output);
  // reverse iterator
  vector<int>::reverse_iterator rit;
  for (rit=v.rbegin(); rit!=v.rend(); rit++)
        cout << *rit << “ “;
```

STL Example 1 (2/3)

```
cout << "\nFirst element of v: " << v.front();
cout << "\nLast element of v: " << v.back();
v[0] = 7;          // set first element to 7
v.at(2) = 10;      // set element at position 2 to 10
v.insert( v.begin() + 1, 22); // insert 22 as 2nd element
cout << "\nContents of vector v after changes: ";
copy (v.begin(), v.end(), output);
try {
    v.at(100) = 777; // access element out of range
} catch (out_of_range e) {
    cout << "\nException: " << e.what();
}
```

STL Example 1 (3/3)

```
v.erase(v.begin());
cout << "\nContents of vector v after erase: ";
copy(v.begin(), v.end(), output);
v.erase(v.begin(), v.end());
cout << "\nAfter erase, vector v "
     << (v.empty()? "is ": "is not ") << "empty";
v.insert(v.begin(), a, a+SIZE);
cout << "\nContents of vector v before clear: ";
copy(v.begin(), v.end(), output);
v.clear(); // clear calls erase to empty a collection
cout << "\nAfter clear, vector v "
     << (v.empty()? "is ": "is not ") << "empty" << endl;
return 0;
```

STL Example 2

```
int x[10]={9,8,7,6,5,4,3,2,1};
int y[10]={23,25,21,26,24,29,20,27,28,22};
int z[10];
char str[] = "QAZJSKEDC";
sort(str, str+strlen(str));
sort(x, x+5);
sort(y, y+10);
merge(x, x+5, y, y+5, z);
int sum = accumulate(x, x+3, 0);
iter_swap(x+3, y+5);
int *pm = max_element(z, z+10);
if (binary_search(str, str+n, 'S'))
     cout<< "found" << endl;
```

STL Example 3

```
int arr[] = {3, 1, 4, 2, 5};
vector<int> vec(arr, arr+5);    // vec = [3, 1, 4, 2, 5]
sort(vec.begin(), vec.begin() + 3); // 排序前三個
sort(vec.begin(), vec.end()); // 全部排序
// 反轉 -> [5, 4, 3, 2, 1]
reverse(vec.begin(), vec.end());
// 找找看 3 有沒有在裡面，找不到就會回傳 vec.end()
vector<int>::iterator it;
it = find(vec.begin(), vec.end(), 3);
if(it != vec.end()){
    cout << "found 3" << endl;
} else {
    cout << "not found 3" << endl;
}
```

What C++ Lacks

- Graphical User-Interface (GUI) Classes: a number of GUI class libraries have been developed, such as
  - Microsoft Foundation Classes (MFC), Borland's Object Window Library (OWL).
  - Cross-platform: Qt, wxWidgets.
- Networking Classes: a major weakness in the age of the Internet.
- Database Classes: C++ does not provide cross-platform interface to databases.

- Thread Classes: (C++11)
  - Exceptions: Pthreads, CMA Threads, etc.

Conclusions

- Make use of C++ extensions to C.
- C++ is a good and widely accepted general-purpose object-oriented language at the present time.

- class
- polymorphism
- template
- iostream
- Exception
- STL: container, iterator, and algorithm
- Boost library

Learn more

- Learn more about C++:
  - **To learn more about C++ syntax:** *C++ primer*, 5th edition, Stanley Lippman.
  - **To use C++ features effectively:** *Effective Modern C++*, Scott Meyers
  - **To learn more about software design:** *C++ Software Design: Design Principles and Patterns for High-Quality Software*, by Klaus Iglberger
  - **To learn more about the history of C++:** *The design and evolution of C++,* by Bjarne Stroustrup.

- Continuing classes:
  - Software Engineering (selective)
  - Computer Science Project (required)
  - Compiler
  - Object-oriented design