第零讲
标准模板库
Lecture 0
Standard Template Library

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

语言掌握的层次
Mastery Level of Programming Language

- 学习一门编程语言大概分为如下三个层次：

  The mastery of a programming language is ranked into three levels:

  - 掌握该语言的语法，如命名约定，关键字，控制结构等

    Mastery of the syntax of the language, for instance, the naming convention, key words, control structures, etc.

  - 掌握该语言提供的辅助函数和库

    Mastery of some utility functions and libraries.

  - 可以基于软件工程的思想，利用该语言实现一定规模的系统

    Ability of implementation of systems based on the practice of software engineering

语言掌握的层次
Mastery Level of Programming Language

- C++的标准模板库是C++语言标准或规范的重要组成部分之一，其提供了利用C++语言实现复杂系统的基本性支持；对模板库的掌握是在第二个层次上衡量熟稔运用C++的重要参考。

  The standard Template Library (STL) is the most significant component of C++ standard. STL which builds upon some features of C++ language provides basic support for implementing complex systems. The mastery of STL is also an important indicator for the mastery of C++ language.

- 标准模板库初衷是利用C++语言的一些特性，如类，模板，实现一些常用数据结构及基于这些数据结构的算法，为程序设计提供基本的支持，避免重复造轮子。

  The motivation for designing STL is by utilizing some features of C++, such as class, template, etc., to offer some basic data structures and the corresponding algorithms to cut the cost on re-inventing the wheels.

标准模板库
Standard Template Library

- C++的标准模板库最初由在惠普实验室工作的Alex Stepanov和Meng Lee于1990年实现，其在1994年被接收为C++标准。

  In 1990, Alex Stepanov and Meng Lee of Hewlett Packard Laboratories extended C++ with a library of class and function templates which has come to be known as the STL. And in 1994, STL was adopted as part of ANSI/ISO Standard C++.

- 标准模板库主要有三部分构成，容器，算法，与迭代器（另一种说法是容器，迭代器，分配器）。

  STL had three basic components: Containers, Algorithms, Iterators (Another viewpoint is Container, Iterator, and Allocator).

标准模板库
Standard Template Library

- 容器（Containers）
  - 存储数据集合的通用类模板

    Generic class templates for storing collection of data.
- 算法（Algorithms）
  - 操作容器类的通用函数模板

    Generic function templates for operating on containers.
- 迭代器（Iterators）
  - 以通用方式使用容器的智能指针，通用算法通过迭代器操作通用容器

    Generalized ‘smart’ pointers that facilitate use of containers. They provide an interface that is needed for STL algorithms to operate on STL containers.

标准模板库
Standard Template Library

- 通常，使用容器类有如下好处：
  - STL 提供种类丰富的容器
  - STL 标明了各个类型的容器的时间和存储复杂性
  - STL 容器的大小可以自动增长和缩减
  - STL 提供了用于处理容器的内置算法
  - STL 提供了迭代器，使容器和算法灵活高效。
  - STL 是可扩展的，这意味着用户可以添加新容器和新算法，同时满足：
    - 标准的STL 算法可以处理 STL 容器以及用户定义的容器
    - 用户自定义的算法可以处理 STL 容器以及用户定义的容器

标准模板库
Standard Template Library

- The benefit of using STL:
  - STL offers an assortment of containers
  - STL publicizes the time and storage complexity of its containers
  - STL containers grow and shrink in size automatically
  - STL provides built-in algorithms for processing containers
  - STL provides iterators that make the containers and algorithms flexible and efficient.
  - STL is extensible which means that users can add new containers and new algorithms such that:
    - STL algorithms can process STL containers as well as user defined containers
    - User defined algorithms can process STL containers as well user defined containers

STL容器
STL Containers

- 容器可视为能保存任何对象的数据结构

  Data structures that hold anything (other objects).

- 常见的容器分为序列容器，如向量，链表等，和关联容器，如集合，映射等

  The commonly-used containers are categorized into sequence containers, such as vector, set, etc., and associative containers, such as set, map, etc.

- 可以基于这些常见的容器类继承，构建更加复杂的容器

  More complicated container could be implemented by inheriting from the standard containers.

向量
vector

- 向量类似于C/C++语言的数组，因此有如下功能与特性：

  Vector resembles the array in C/C++ language, and is endowed with the characteristics and functionalities as below:

  - 类似于数组的随机访问

    random access ([] operator) like an array

  - 以常量时间在末尾添加或删除元素；以线性时间在中间插入或删除元素

    add / remove an element in constant time at the last of vector; add / remove an element in linear time in the middle of vector

  - 元素大小自动推定，数组大小自动管理

    automatic deduction of element size and automatic memory management

向量
vector

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> v;
    v.push_back(10), v.push_back(20);
    cout<<v[0]<< " " <<v[1]<<endl;
    cout << v.size() << " " << v.capacity() << endl;
    v.pop_back();
    cout << v.size() << " " << v.capacity() << endl;
}
```

```
/* Output:
10
10 20
2 2
10
1 2
*/
```

字符串
string

- String类似于C语言中的字符串数组，C++中以抽象的方式提供了字符串类的统一实现

  string resembles the char * array in C, and the C++ standard library provides a common implementation of a string class abstraction named string.

- 示例（example）：

```
string s = "hello";
string s1 = s.substr(0, 3); // "hel"
s+= " world "; // "hello world"
cout << s.size() << endl; // 12
```

链表
list

- 标准库链表是元素非连续存储的序列容器，有如下功能与特性：

  List in the STL is a sequence container with elements stored in a non-contiguous manner, and is endowed with the characteristics and functionalities as below:

  - 由于非连续存储，因此不能随机索引，需要以线性时间访问特定位置元素

    Non-contiguous storage prohibits random access like an array, and incurs a linear time access to element in the specific position

  - 以常量时间在头部或末尾添加或删除元素；以线性时间在中间插入或删除元素

    add / remove an element in constant time at the head or tail of the list; add / remove an element in linear time in the middle of list;

  - 元素大小自动推定，链表大小自动管理

    automatic deduction of element size and automatic memory management

链表
list

```
#include <iostream>
#include <list>
using namespace std;

auto print = [](list<int>& l){ for (auto& e : l) cout << e << "\t";};
int main()
{
    list <int> la, lb;
    la.push_front(3), la.push_back(1), lb.push_back(4), lb.push_front(2);
    la.sort(), lb.sort(), la.merge(lb);
    print(la);
}
```

```
/* Output:
1       2       3       4*/
```

栈和队列
stack and queue

- 其它的序列容器还有栈（stack）和队列（queue），示例如下：

```
#include <iostream>
#include <stack>

using namespace std;

int main()
{
        stack<int> s;
        s.push(8), s.push(5), s.push(6);
        cout << s.top() << endl;
        s.pop();
        cout << s.top() << endl;
}
```

```
/* Output:
6
5
*/
```

```
#include <iostream>
#include <queue>

using namespace std;

int main()
{
        queue<int> q;
        q.push(8), q.push(5), q.push(6);
        cout << q.front() << "\t" << q.back() << endl;
        q.pop();
        cout << q.front() << "\t" << q.back() << endl;
}
```

```
/* Output:
8       6
5       6
*/
```

关联容器
Associative Containers

- STL中关联容器的实现可能是通过平衡二分查找树（查找时间复杂度为$O(n\log n)$）来实现的，虽然具体实现对用户透明，但这些实现决定了元素在集合中位置是排序的（即独立于插入时的时机）；由于位置对用户透明，这决定了关联式容器中的元素访问是通过值，而不是通过元素在容器中的位置来访问。

  The associative containers might be implemented as a balanced binary-search trees (with insertion searching complexity $O(n\log n)$). Although the concrete implementation is transparent for the end-user, it induces the constraint that elements in the container follow a strict order (independent of the insertion time). Meantime, the transparence of the position requires that elements in associative containers are referenced by keys or values instead of their absolute positions in the container.

- 集合是关联容器的一种，其与数学中的集合的概念一致，集合中的元素不允许有重复

  Set is one of the associative containers that STL provides. Same as the concept in mathematics, duplicate elements in the container are not allowed.

集合

set

```
#include <string.h>
#include <iostream>
#include <set>
using namespace std;

template<typename T> void print(set<const char*, T>& l) { for (auto& e : l) cout << e << "\t";};

int main()
{
    auto g = [](const char* s1, const char* s2) { return strlen(s1) <= strlen(s2); };
    const char* str[6] = {"This", "is", "a", "Test", "of", "string"};
    set<const char*, less_equal<const char*>> s(str, str + 6);
    print<less_equal<const char*>>(s), cout << endl;
    set<const char*, decltype(g)> s1(str, str + 6, g);
    print<decltype(g)>(s1);
}
```

```
/* Output:
This    is      a       Test    of      string
a       of      is      Test    This    string
*/
```

映射
map

```
#include <iostream>
#include <map>

using namespace std;

int main()
{
        map<string, int> grade;
        grade["Mark"] = 95, grade["Edward"] = 87;
        grade["Louise"] = 66, grade["Allen"] = 76;

        for(auto& [key, value] : grade)
       cout<< "Name: " << key << "\tMark: "<< value << endl;
}
```

```
/* Output:
Name: Edward    Mark: 87
Name: Louise    Mark: 66
Name: Mark      Mark: 95
*/
```

迭代器
Iterator

- 迭代器提供了以一种通用方式访问序列容器（向量、链表）和关联容器（集合、映射）中元素的方法。

  Provide a general way for accessing each element in sequential (vector, list) or associative (map, set) containers.

- 迭代器扩展了指针的概念，可以认为与指针概念兼容；如令iter为迭代器，则：

  Iterator extends the concept of pointer, so it is compatible with the semantics of pointer. For example, let iter be an instance of an iterator, then:

  - ++ iter（或iter ++）表示将指针指向下一个元素

    ++ iter (or iter ++) means to advances the iterator to the next element

  - * iter表示访问指针指向的元素的值

    * iter returns the value of the element addressed by the iterator.

迭代器
Iterator

- 每个容器都提供了两个成员函数begin()与end()，

  Each container provide a begin() and end() member functions.

  - 其中begin()返回指向容器中第一个元素的地址

    begin() returns an iterator that addresses the first element of the container.

  - end()返回最后一个元素的下一个位置，类似于哨兵位置

    end() returns an iterator that addresses 1 past the last element, resembling a pivot position.

- 常见的迭代范式如下（The paradigm of iteration is as below）：

```
for(iter = container.begin(); iter != container.end(); iter ++ ) {
      // do something with the element
}
```

映射的迭代器
Iterator of Map

- 需要单独指出的是，许多容器的迭代器均指向该容器存储的单个元素，但映射的迭代器会指向一个二元组对象，包含该位置的键与值。

  Notably, the iterator for map points to a pair of objects at specific positions instead of just a single elements as iterators for other containers. The first object of the pair is the key, and the second object is the corresponding value.

- 其中pair的定义如下：

```
struct pair<T1, T2>
{
    T1 first;
    T2 second;
};
```

```
#include <iostream>
#include <map>

using namespace std;

int main()
{
    map<string, int> grade;
    grade["Mark"] = 95, grade["Edward"] = 87;
    grade["Louise"] = 66, grade["Allen"] = 76;

    for(auto it = grade.begin(); it != grade.end(); it ++)
        cout<< "Name: " << it->first << "\tMark: "<< it->second << endl;
}
```

算法
Algorithms

- STL为常见使用场景提供了一些通用算法，一些算法可能作用特定类型的容器，而另一些算法则对所有容器均适用。

  STL provides some algorithms catering to common scenarios. Some algorithms apply to specific containers, others are generic enough for all.

- STL算法大概分为以下四类：

  The algorithms of STL is divided into four categories:

  - 非修改算法（Nonmodifying algorithms）
  - 修改算法（Modifying algorithms）
  - 数字算法（Numeric algorithms）
  - 堆算法（Heap algorithms）

算法
Algorithms

- 非修改算法顾名思义，其不改变容器中元素的值，如下表所示：
- Non-modifying algorithms just as its name implies, it doesn’t modify the content of its elements:

adjacent_find	find_end	max_element
binary_search	find_first_of	min
count	find_if	min_element
count_if	for_each	mismatch
equal	includes	search
equal_range	lower_bound	search_n
find	max	upper_bound

算法
Algorithms

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main ()
{
    vector<int> myvector ({10, 20, 30, 40});
    auto it = find (myvector.begin(), myvector.end(), 30);
    if (it != myvector.end())
        cout << "The position of 30 in the array is " << distance(myvector.begin(), it) << endl;
    else cout << "Cannot find 30 in the array" << endl;
    auto min_e = *min_element(myvector.begin(), myvector.end());
    cout << "The minimum element in the array is " << min_e << endl;
    return 0;
}
```

算法
Algorithms

- 修改性算法或者改变容器中元素的值，或者改变容器中元素的当前顺序，如下表所示：

  Modifying algorithms either modify the content of its elements, or change the current order of the elements:

copy	prev_permutation	rotate_copy
copy_backward	random_shuffle	set_difference
fill	remove	set_intersection
fill_n	remove_copy	set_symmetric_difference
generate	remove_copy_if	set_union
generate_n	remove_if	sort
inplace_merge	replace	stable_partition
iter_swap	replace_copy	stable_sort
merge	replace_copy_if	swap
next_permutation	replace_if	swap_ranges
nth_element	reverse	transform
partial_sort	reverse_copy	unique
partial_sort_copy	rotate	unique_copy
partition		

算法
Algorithms

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
int main ()
{
   auto print = [](const int& e) { cout << e << "\t"; };
   vector<int> myvector ({2, 1, 4, 6});
   transform(myvector.begin(), myvector.end(), myvector.begin(), [](int& e){ return e * e;});
   for_each(myvector.begin(), myvector.end(), print);
   cout << endl;
   sort(myvector.begin(), myvector.end(), greater<int>());
   for_each(myvector.begin(), myvector.end(), print);
   return 0;
}
```

算法
Algorithms

- 数值算法于容器中的元素上进行一些数值运算，STL提供了如下算法：

  Numeric algorithms perform numeric calculations on container elements:

accumulate	inner_product
adjacent_difference	partial_sum

算法
Algorithms

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
#include <numeric>
using namespace std;
int main ()
{
   vector<int> v({2, 1, 4, 6}), v2;
   auto sum = accumulate(v.begin(), v.end(), 0);
   cout << "sum of the array: " <<sum << endl;
   v2.resize(v.size()), fill_n(v2.begin(), v.size(), 1);
   auto sum2 = inner_product(v.begin(), v.end(), v2.begin(), 0);
   cout << "inner-product of v and v2: " << sum2 << endl;
   return 0;
}
```

其它
Miscellaneous

- 关于STL的其他方面，如函数对象，函子，分配器等，这里不作过多描述，在后面学习的过程中接触到再讲解。

  For other miscellaneous topics such as function object, functors, allocator, etc., they will be respectively dissected on necessary later.

- 注意，有些算法的对象是迭代器时，某些操作会返回一个新的迭代器对象，操作完成之后，当前迭代器会失效，对其的操作会引发异常。

  Notably, when some algorithms applied to iterators, these algorithms might invalidate the current iterator and return a new one. Any operation to the old iterator may raise an exception when the algorithms finish.

习题
Problems

- 进行网络检索，了解STL的各个组成部分都在哪些头文件中

  To understand the header files that include different components of STL