C++ Standard Template Library

LyuJiuyang

2021.3.6

Contents

什么是STL?

STL（Standard Template Library），即标准模板库，是一个具有工业强度的，高效的C++程序库。 STL 借助模板实现了常用的**数据结构及算法**，并且做到了数据结构和算法的分离。

STL包括**容器（Container）**、**迭代器（Iterator）**、**算法（Algorithm）**、仿函数（Functor）、适配器（Adaptor）、分配器（Allocator）六大组件，在算法竞赛里有广泛的应用。

什么是STL?

简单来说，就是有人把经常用到的算法和数据结构打包好了，直接就能用。不需要我们再造轮子了。

> “ 造轮子（Reinventing_the_wheel）指的是重复发明已有的算法，或者重复编写现成优化过的代码。造轮子通常耗时耗力，同时效果还没有别人好。但若是为了学习或者练习，造轮子则是必要的。 ”

请注意

1. 《数据结构》《C++语言基础》等基础课程中一般是**禁止使用**STL的。
2. 我们经常使用的是STL中的部分容器、容器适配器、迭代器和部分算法。

容器概览

序列式容器(Sequence container)

- array (C++标准库)
- **vector** 动态数组
- list 双向链表
- forward_list 单向链表
- **deque** 双向队列

序列式指其中的元素都可序(ordered)，不是指有序。

关联式容器(Associative container)

- **set** (内含一个 RB-tree)
- **map** (内含一个 RB-tree)
- multiset (内含一个 RB-tree)
- multimap (内含一个 RB-tree)
- unordered_set (内含一个 hashtable)
- unordered_map (内含一个 hashtable)

容器适配器(Container adapter)

- **stack** (内含一个 deque)
- **queue** (内含一个 deque)
- **priority-queue** (内含一个 heap)

算法概览

- sort: 排序（快速排序+堆排序）
- unique：去重
- binary_search, lower_bound, upper_bound：二分查找
- next_permutation：全排列
- nth_element：第n大元素
- reverse：翻转容器
- merge：合并容器

SGI-STL算法图示 C++ Reference Algorithm

容器的创建

```
#include <containerName>
containerName <typeName, ...> name
```

其中参数的个数根据具体的容器会变。

```
vector <int> v;
stack <double> s;
map <string, int> mp;
```

```
map <int, pair<int, int> > m;   // 这里两个>不能连起来，否则会被识别成 >>
```

所有容器共有的方式

`begin()`：返回指向开头元素的迭代器。

`end()`：返回指向末尾的**下一个元素**的迭代器。

`size()`：返回容器内的元素个数。

`empty()`：返回容器是否为空。

`swap()`：交换两个容器。

`clear()`：清空容器。

Vector

可以理解成长度可以变化的数组。

`push_back()`：插入操作(末尾)

`pop_back()`：删除操作(末尾)

`erase()`：清除某范围 `[first, last)` 元素，或删除某个位置上的元素

`clear()`：清除所有元素

示例

Vector的其他访问方式

用法	含义
v[i]	直接以下标方式访问容器中的元素
v.front()	返回首元素
v.back()	返回尾元素

```
for(int i = 0; i < v.size(); ++ i)
        cout << v[i] << endl;
```

List

双向链表，不常用。

链表只能用迭代器访问。迭代器会在后面说到。

```
int main () {
    for (int i=1; i<=5; ++i) mylist.push_back(i);
    for (list<int>::iterator it = mylist.begin(); it != mylist.end(); ++it)
        std::cout << ' ' << *it;
}
```

Stack

栈，遵循**后进先出**原则，类似洗碗摆放。

Queue

队列，遵循**先进先出**原则，类似超市收银台结账。

栈和队列会在第6节课详细讲解。

访问

`push(x)`：尾端进

`pop()`：首端出

`top()` / `front()`：（栈和队列分别的）返回首端元素

`empty()`：容器是否为空

访问

queue和stack没有迭代器，每次只能访问队首元素。 如果要遍历则需要像这样使用：

```
int main() {
    queue <int> q;
    q.push(1); q.push(2); q.push(3);

    while( !q.empty() ) {
        cout << q.front() << " ";
        q.pop();
    }
}
```

Priority_queue

优先队列。

普通队列在尾部追加元素，从头部删除，优先队列则是优先级最高的最先删除。

优先队列的**基本使用方法和队列是一样的**。

下面我们通过一个例子来学习。（情节虚构，如有雷同不胜荣幸）

校队最近因为财政紧张，只能保留5个名额（←我瞎编的）。黄老师给定每个人的能力值，请问各个时刻校队的人员变化?

这个时候我们想，排个序不就完了？(sort，复杂度$O(nlogn)$)

id	name	power	id	name	power
0	zls	100000000	3	jdl	1900
1	夏教	2200	4	ljy	1000
2	cuccenter	1900			

但是，之后发生了：

- yxt（1300）加入新生队（sort一次或枚举找一遍）
- yys（1400）入队 （又一次）
- wqs（1500）入队 （再一次）
- 此处省略372个字
- ……
  如果每个时刻都排序，总的复杂度会相当高。

这个时候应该使用优先队列。

```cpp
struct node {
    string name;
    int val;
    node(string _n, int _v) : name(move(_n)), val(_v) {}
    bool operator < (const node & a) const { // 一般不重载大于号
        return val>a.val;
    }
};
priority_queue<node> q;
int main() {
    string n; int v;
    while (cin >> n >> v) {
        q.push(node(n, v));
        cout << "+" << n << " " << v << endl;
        if (q.size() > 5) {
            cout << "-" << q.top().name << " " << q.top().val << endl;
            q.pop();
        }
    }
}
```

几种构造方法

```
priority_queue <int,vector<int>,less<int> > p;
priority_queue <int,vector<int>,greater<int> > q;
```

```
struct node {
    int x,y;
    bool operator<(const node &a) const {
        return x>a.x;
    }
    // friend bool operator<(const node &a, const node &b) {
    //     return a.x > b.x;
    // }
};
priority_queue <node> q;
```

几种构造方法

```
struct cmp {
    operator bool()(int x, int y) {
        return  x > y;
    }
};
priority_queue<int, vector<int>, cmp> q;
```

Set

集合。Set会自动实现内部元素的排序和去重。

“ 确定性、互异性、无序性为集合的三个特性。——《初中数学》 ”

和数学上不同，STL里的Set是自动有序的。这样比较的时候才能体现出“无序性”。

插入删除查找都是$O(log_2 n)$, n是set的大小。

Set的访问

Set只能通过迭代器访问。

```
int main() {
    set<int> s;
    s.insert(1);
    s.insert(2);
    s.insert(1);
    auto it = s.begin();
    cout << *(++ it) << " " << *( -- it ) << endl; // 2 1
}
```

Tips

std::set::lower_bound & std::lower_bound

```
set< int > s;
s.lower_bound( num );   // O(logn)
```

```
set< int > s;
lower_bound( s.begin(), s.end(), num ); // O(logn + n)
```

Stackoverflow

那要是不想互异怎么办？

```
multiset <int> ms;
```

那要是不想有序怎么办？

```
#include <unordered_set>
unordered_set <int> ms;
```

unordered_set 内部用散列（哈希表）实现，不需要排序的时候会更快。

Map

映射，或者称为键值对。类似Python里的字典dict。

```
map < keytype, valuetype > name;
```

map可以实现从任意基本类型到任意基本类型的映射，但是keytype不能是数组类型。

map会自动按照keytype的字典序从小到大进行排序。

插入、查找、删除的时间复杂度都是$O(log_2 n)$, n是map的大小。

Map 访问

可以通过键或者迭代器访问。

```
int main() {
    map< string ,int > age;
    age["San Zhang"] = 12;
    age["Wang Xiao"] = 23;
    age["Ann Xia"] = 128;
    cout << age["Ann Xia"] << endl;
    for( map<string, int>::iterator it = age.begin(); it != age.end(); ++ it )
        cout << it->first << " " << it->second << endl;
}
```

那要是键值对不唯一怎么办?

```
multimap <string, int> mp;
// 注意此时也不能通过键访问或添加元素
```

那要是不想有序怎么办?

```
#include <unordered_map>
unordered_map <string, int> mp;
```

unordered_map 内部用散列（哈希表）实现，不需要排序的时候查找会更快。

Bitset

严格意义上bitset不属于STL。

bitset的结构类似于bool数组，由于内存地址是按字节即 byte 寻址，而非比特 bit，一个 bool 类型的变量，虽然只能表示 0/1, 但是也占了 1 byte 的内存。

bitset 就是通过固定的优化，使得一个字节的八个比特能分别储存 8 位的 0/1。

Bitset使用

定义：

```
bitset<length> name;
```

```
int main()
{
    bitset<8> bit(25);
    cout << bit[0] << endl; // 1
    cout << bit << endl;    // 00011001
}
```

迭代器

迭代器可以看成一个数据指针。主要支持两个运算符：自增 (++) 和解引用（单目 * 运算符）

```
container::iterator // 前面的部分和定义的时候一样
```

```
for (vector<int>::iterator iter = v.begin(); iter != v.end(); iter++)
    cout << *iter << endl;
for (map<string, int>::iterator it = age.begin(); it != age.end(); it++)
    cout << it->first << " " << it->second << endl;
```

注意这里是 `!=` 而非 `<`，迭代器不能比较大小。

容器可以用迭代器访问，容器适配器（stack、queue、priority_queue）不可以。

成员函数表格

- C++03 起存在的函数
- C++11 起存在的函数
- C++17 起存在的函数
- C++20 起存在的函数

		顺序容器					关联容器				无序关联容器				容器适配器		
	头文件	array	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unordered_set>		<unordered_map>		<stack>	<queue>	
	容器	array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(构造函数)	(隐式)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(析构函数)	(隐式)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap	~stack	~queue	~priority_queue
	operator=	(隐式)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
	assign		assign	assign	assign	assign											
迭代器	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin			
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin			
	end	end	end	end	end	end	end	end	end	end	end	end	end	end			
	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend			
	rbegin	rbegin	rbegin	rbegin		rbegin	rbegin	rbegin	rbegin	rbegin							
	crbegin	crbegin	crbegin	crbegin		crbegin	crbegin	crbegin	crbegin	crbegin							
	rend	rend	rend	rend		rend	rend	rend	rend	rend							
	crend	crend	crend	crend		crend	crend	crend	crend	crend							
元素访问	at	at	at	at					at				at				
	operator[]	operator[]	operator[]	operator[]					operator[]				operator[]				
	data	data	data														
	front	front	front	front	front	front										front	top
	back	back	back	back		back									top	back	
容量	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty
	size	size	size	size		size	size	size	size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size			
	resize		resize	resize	resize	resize											
	capacity		capacity								bucket_count	bucket_count	bucket_count	bucket_count			
	reserve		reserve								reserve	reserve	reserve	reserve			
	shrink_to_fit		shrink_to_fit	shrink_to_fit													
修改器	clear		clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear			
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert	insert	insert	insert	insert			
	insert_or_assign								insert_or_assign				insert_or_assign				
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace			
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint			
	try_emplace								try_emplace				try_emplace				
	erase		erase	erase	erase_after	erase	erase	erase	erase	erase	erase	erase	erase	erase			
	push_front			push_front	push_front	push_front											
	emplace_front			emplace_front	emplace_front	emplace_front											
	pop_front			pop_front	pop_front	pop_front										pop	pop
	push_back		push_back	push_back		push_back									push	push	push
	emplace_back		emplace_back	emplace_back		emplace_back									emplace	emplace	emplace
	pop_back		pop_back	pop_back		pop_back									pop		
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap
	merge				merge	merge	merge	merge	merge	merge	merge	merge	merge	merge			
	extract						extract	extract	extract	extract	extract	extract	extract	extract			
链表操作	splice				splice_after	splice											
	remove				remove	remove											
	remove_if				remove_if	remove_if											
	reverse				reverse	reverse											
	unique				unique	unique											
	sort				sort	sort											
查找	count						count	count	count	count	count	count	count	count			
	find						find	find	find	find	find	find	find	find			
	contains						contains	contains	contains	contains	contains	contains	contains	contains			
	lower_bound						lower_bound	lower_bound	lower_bound	lower_bound							
	upper_bound						upper_bound	upper_bound	upper_bound	upper_bound							
	equal_range						equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range			
观察器	key_comp						key_comp	key_comp	key_comp	key_comp							
	value_comp						value_comp	value_comp	value_comp	value_comp							
	hash_function										hash_function	hash_function	hash_function	hash_function			
	key_eq										key_eq	key_eq	key_eq	key_eq			
分配器	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator			
	容器	array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
		顺序容器					关联容器				无序关联容器				容器适配器		

auto

C++11 之后，允许使用 `auto iter = v.begin()` 简化上述代码。

```
for (auto iter = v.begin(); iter != v.end(); iter++)
    cout << *iter << endl;
```

还可以使用以下代码进行遍历：

```
for (auto i: v ) cout << i << endl; // 无法修改只能读取
for (auto &i: v ) i += 2;
```

Tips

Dev C++默认使用的版本是C++98，需要进行简单配置才能使用上页的代码。

VS2019、Clion2020默认使用的是C++11，可以直接运行。

目前ICPC官方规则支持C++17，请注意C++20的部分语法无法使用。

本节课除了auto部分所有代码可在C++98上实现。

例题 CF958D1

给定2e5个算式，计算每个算式结果出现的次数，每个算式结果范围在0~1e9之间。

从小到大输出每个结果和对应的次数。
（题意有简化和修改）

例题 NOIP2012普及组

分解质因数。输出一个数字的全部因数和对应指数。

例题 CF469A

给定A能通过的关卡数和B能通过的关卡数，问两个人一起从第一关开始闯关，问是否可以通过全部关卡？

```
4          // 总关卡数
3 1 2 3    // A能通过的关卡数，无序且重复
4 2 2      // B能通过的关卡数，无序且重复
```

例题 ICPC2019(南昌)网络赛

T(1-10)组数据，初始时有n(4e7)张牌(按顺序摆放)，每一次操作你将顶端的牌拿出，然后按顺序将上面的m(1-10)张牌放到底部。
求出队顺序。

Time: 6000ms Memory:524288K

一些闲话

Q：STL有什么缺点嘛?

A：“慢”。极端情况下会被卡时间，用错了数据结构也会卡时间，乱deep copy也会卡时间，vector也比array慢。不过这个慢是相对的，大部分情况下比你手动实现快得多...这个知道就行，能用STL还是用吧。

Q：记不住那么多函数怎么办?

A：只要记住有哪种结构就可以了，具体的可以现查。多练就能掌握。

Q：STL后续用的多吗?

A：非常多，应用场景非常广泛，由于篇幅限制我们无法详细说明。

参考文献及推荐阅读

- STL - CUC ACM Wiki. by Cuccenter, YanhuiJessica, LyuLumos
  前校队哥姐的精心之作，自学专用。
- STL 容器 - OI Wiki 比较详细的解释，内容有一定难度和深度。
- C++ Reference 官方文档，当字典查。
- STL源码剖析 挺好的一本书。