Cheatography

C++ STL Cheat Sheet

by Zoupers via cheatography.com/129178/cs/30896/

迭代器基本操作

++p, p++, *p	前向迭代器
--p, p--	双向迭代器
p+=i, p-=i, p+i, p-i, p[i]	随机访问迭代器

迭代器分正向和反向，对反向进行++，迭代器指向前一个元素

Array容器

std::array<char,50>addr{"hello"}	初始化
begin()/end()	返回正向迭代器
cbegin()/cend()	返回常量正向迭代器
rbegin()/rend()	返回反向迭代器
crbegin()/crend()	返回常量反向迭代器
values[i], values.at(i), get<n>(*)	访问单个元素
values.size()	返回元素个数-(size_t)
values.empty()	判断容器是否有元素

该容器内存储的所有元素一定会位于连续且相邻的内存中，无法扩展或者收缩

Vector容器

std::vector<double> values(N, V);	初始化
begin()/end(),rbegin()/rend()	同Array
size(),empty()	同Array
capacity()	返回当前容量
front()	返回第一个元素的引用
resize()	改变实际元素个数
back()	返回最后元素的引用
reserve(N)	增加容量到N
push_back(v),emplace_back(v)	在尾部添加一个元素

Vector容器 (cont)

insert(p,v), insert-(p,n,v)	在p前插入1/n个元素
insert(p,first,last)	插入元素[first,last)
insert(p,initlist)	插入初始化列表,{}括着
pop_back()	删除最后元素
erase(p), erase(-beg,end)	删除p处或[-beg,end)
remove(v)	删除和v相等元素
clear()	删除所有元素
shrink_to_fit()	将容量缩减至合适

容量指不分配新内存当前能保存的最多元素个数

Deque容器

std:deque<int> d(N,v)	初始化
emplace_front(v)	直接在头部生成元素

其他方法基本同vector容器，除了reserve()-和capacity()方法

List容器（双向链表容器）

splice()	将一个list容器中的元素插入到另一个容器的指定位置
remove-_if()	删除容器中满足条件的元素
unique()	删除相邻的重复元素，只保留一个
merge()	合并两个有序list容器，合并后仍然有序
sort()	更改容器中元素的位置，将他们进行排序
reverse()	反转容器中元素的顺序

不支持随机访问，其余方法同deque

Map容器

map<string,int> myMap{{"k1",1},{"k2",2}}	初始化
count(k)	查找键k个数
find(k)	返回迭代器

默认选择std::less<T>排序规则，升序排列
另外还有multimap，和map类似，但是可以同时存储多个键相同的键值对，但是multimap未提供at()成员方法，也没有重载[]运算符

Multimap容器

和map类似，但是可以同时存储多个键相同的键值对，但是multimap未提供at()成员方法，也没有重载[]运算符

Set容器

set<string> myset	初始化

默认采用less<T>规则，升序排列；基本方法同map类似.
另外还有multiset容器，不过可以储存多个相同的值.

unordered_map容器

load_factor()	返回当前负载因子
rehash(n)	加那个底层使用桶数量设为n

其他方法类似map函数
另外还有unordered_multimap，同unordered_map类似，但是可以存储多个键相等的键值对

unordered_set容器

unordered_set<int> uset;	初始化
insert(v)/emplace(v)	添加元素
erase(v)	删除指定元素

不能修改元素，无法随机访问
另外还有unordered_multiset，同unordered_set类似，但是可以同时存储多个值相同的元素

unordered_multiset容器

同unordered_set类似，但是可以同时存储多个值相同的元素

stack容器适配器

stack<int, list<int>> my_stack;	初始化
empty(),size()	同array
top()	返回栈顶，为空则报错
push(v)/emplace(v)	将v压入栈顶
pop()	弹出栈顶
swap(other_stack)	和另一个栈互换

默认封装了deque<T>，互换栈时需要存储元素类型和底层基础容器

Queue容器适配器

std::queue<int> values;	初始化
values.push(elem)	以移动方法在尾部添加元素
values.emplace(elem)	直接在queue尾部添加一个元素
values.pop()	删除queue中的第一个元素
values.front()	返回第一个元素
values.back()	返回最后一个元素

和stack一样，queue也没有迭代器，访问元素的唯一方式是不断的移除访问过的元素。

priority_queue容器适配器

priority_queue<int> values;	初始化，默认大根堆
values.push(elem)	添加元素
values.emplace(elem)	添加元素
values.pop()	移除第一个元素
values.top()	返回第一个元素

priority_queue也没有迭代器，默认的比较函数是less，可以使用greater初始化为小根堆

迭代器适配器

reverse_iterator	反向迭代器，又称逆向迭代器
inserter或者insert_iterator	安插型迭代器
istream_iterator/ostream_iterator	流迭代器
istreambuf_iterator/ostreambuf_iterator	流缓冲迭代器
move_iterator	移动迭代器

以上4中迭代器仅供参考，想要详细了解请自行查阅，另外还有迭代器辅助函数等。
这里提供一个参考网站，https://c.biancheng.net/view/7255.html

常用算法

sort(first, last)	对[first, last)范围内元素排序
stable_sort(first, last)	对[first, last)内元素进行稳定排序
partial_sort(first,middle,last)	对[first,last)内元素选出middle-first个元素放在[first,middle)区间内
partial_sor_copy(first,last,result_first,result_last)	筛选出result_last-result_first个元素排序并存储到[result_first,result_last)
is_sorted(first,last)	检测[first,last)范围内元素是否排好序，默认升序
is_sorted_until(first,last)	
nth_element(first,nth,last)	nth左侧都比nth小，右侧都比nth大
merge(first1,last1,first2,last2,result)	有序合并有序序列至result
inplace_merge(first,middle,last)	
find(first,last,val)	在[first,last)中找val第一次出现位置

常用算法 (cont)

find_if(first,last,pred)	可以允许自定义规则
find_if_not(first,last,pred)	
find_end(first1,last1,first2,last2)	找1中寻找2最后一次出现位置
find_first_of(first1,last1,first2,last2)	找到1序列中2序列任意元素出现的首个位置
adjacent_find(first,last)	找到2个连续相等的元素
search(first1,last1,first2,last2)	在1中寻找2第一次出现位置
search_n(first,last,n,val)	寻找val在连续出现n次的位置

STL算法太多了...熟能生巧