# vector

1、resize函数，第一个参数指定元素数量，第二个元素指定初始值。这个函数只会初始化一次vector的元素，例如：

vector<int> stus;

stus.resize(2,1); //此时stus的两个元素为1

stus.resize(3,2); //此时stus的前两个元素为1，第三个元素为2

如果想要覆盖stus原有元素值，则可使用algorithm里的fill函数，例如：

stus.fill(stus.begin(), stus.end(), 2);

resize 函数会改变size大小，例如：

vector<int> stus;

stus.reize(2, 0);

stus.push\_back(1); //1为下标2的元素，即第三个元素

2、empty 函数，判断vector是否为空

3、clear 函数，清空vector

4、erase函数，使用iterator遍历删除某一元素时，调用erase(iterator)后，iterator自动后移到被删除元素的下一个位置。若要想遍历正确，则有以下两种方法：

正向遍历：

vector<int>::iterator it = vec.begin();

for(;it != vec.end();)

{

if(\*it == 5)

//删除指定元素，返回指向删除元素的下一个元素的位置的迭代器

it = vec.erase(it);

else

//迭代器指向下一个元素位置

++it;

}

逆向遍历：

vector<int>::reverse\_iterator it = vec.rbegin();

for(; it != vec.rend();)

{

if(\*it == 5)

vec.erase((++it).base());

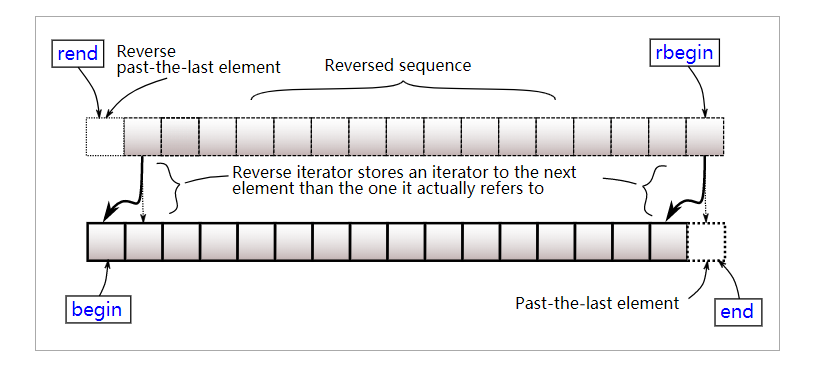
else

++it

}

注意rbegin，rend及其base的位置，如下图。

rbegin.base() – 1 与 rbegin 指向同一个元素。



# map

map内部实现是红黑树（一种二叉查找树），所以对map进行插入、删除后都会自动排序（默认按键值排序），若Key为自定义类型，则自定义类型需要重载operator< 函数。

1、自定义map的排序方式

map类的定义：

template < class Key, class T, class Compare = less<Key>, class Allocator = allocator<pair<const Key, T> > > class map;

重点看第三个参数class Compare = less<Key>，less是一个函数对象（即实现operator()的重载的类或结构体）。下面是less的实现：

template <class T> struct less : binary\_function <T, T, bool> {

bool operator() (const T& x, const T& y) const

{return x<y;}

};

与less相对的还有greater：

template <class T> struct greater: binary\_function <T, T, bool> {

bool operator() (const T& x, const T& y) const

{return x>y;}

};

可以知道map默认使用less函数对象对Key进行排序，即按键值从小到大排序。我们可以在声明map对象时指定函数对象，下面是使用自定义的函数对象对map进行排序。

1. 自定义键值排序的函数对象

struct CmpByKeyLength { //按键值字符串长度排序

bool operator()(const string& k1, const string& k2) {

return k1.length() < k2.length();

}

};

map<string, int, CmpByKeyLength> OrderMap; //指定排序使用的函数对象

接下来再对map进行插入、删除就会自动按照键值字符串长度排序了。

1. 若要希望map按值排序，则借助vector<pair<Key, Value>>对象。将map的键值对放入vector中，再定义一个比较函数传给sort函数，即可对vector排序，相当于对map的键值对按值大小进行排序。如下：

typedef pair<string, int> PAIR;

bool CmpByValue(const PAIR& lhs, const PAIR& rhs) {

return lhs.second < rhs.second;

}

vector<PAIR> OrderVec(OrderMap.begin(), OrderMap.end());

sort(OrderVec.begin(), OrderVec.end(), CmpByValue);

这种情况可以使用hash\_map或C++11的unordered\_map提高执行效率。

# unordered\_map（C++11）

unordered\_map内部使用hash表实现，其定义：

template <class Key,

class Ty,

class Hash = std::hash<Key>,

class Pred = std::equal\_to<Key>,

class Alloc = std::allocator<std::pair<const Key, Ty>>>

class unordered\_map;

1. 用法

* 头文件：#include <unordered\_map>
* 取键值

unordered\_map<Key,T>::iterator it; it->first; it->second;

* 成员函数

begin/cbegin end/cend （c开头表示返回const属性的迭代器）

empty 检查容器是否为空

size 返回容纳的元素数

max\_size 返回可容纳的最大元素数

clear 清空内容

insert 插入元素或节点（C++17）

insert\_or\_assign 插入元素，或若关键已存在则赋值给当前元素

# set

底层使用一种平衡查找二叉树——红黑树实现，插入、删除自动排序（默认从小到大）。set 中元素不重复，与之相对的是multiset。支持集合的并、交、差操作。

# 常用的函数

1、merge：合并两个有序序列。

vector<int> v1;

vector<int> v2;

vector<int> v3;

sort(v1.begin(), v1.end());

sort(v2.begin(), v2.end());

merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());//把v1，v2合并到v3中

merge 与 sort 类似可以自定义比较函数，若比较对象没有<或>则需要自定义merge时需要的比较操作（函数或函数对象），并传给最后一个参数