**C++STL**

**#include <algorithm>**

**#include <vector> //向量容器**

//创建vector对象v

vector<int> v;

vector<double> v(10);

vector<double> v(10,8.6);

//尾部元素扩张

v.push\_back(2);

v[0]=2; //下标方式访问 vector 元素

//用迭代器访问 vector 元素

vector<int>::iterator it;//定义迭代器变量

for(it=v.begin();it!=v.end();it++) {

//输出迭代器上的元素值

cout<<\*it<<" ";

}

//在最前面插入新元素，元素值为 8

v.insert(v.begin(),8);

//在第 2 个元素前插入新元素 1

v.insert(v.begin()+2,1);

//在向量末尾追加新元素 3

v.insert(v.end(),3);

//删除 2 个元素，从 0 开始计数

v.erase(v.begin()+2);

//删除迭代器第 1 到第 5 区间的所有元素

v.erase(v.begin()+1,v.begin()+5);

//反向排列向量的从首到尾间的元素

reverse(v.begin(),v.end());

//排序，升序排列

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

v.size();//返回vextor大小

//返回向量是否为空，非空返回逻辑0，否则返回1

v.empty();

**#include <string> //基本字符系列容器**

string s;

s="hello,C++STL";

// string 对象尾部追加字符串

s=s+'a'; //在 string 对象的尾部添加一个字符（char）'',或字符串""

s.append("abc");

//把一个字符插入到迭代器位置之前

s.insert(it+1,'p');

//删除第 3 个元素，元素位置从 0 开始计数

s.erase(it+3);

s.length();//返回长度

//从第 3 个开始，将连续的 3 个字符替换为“good”

//即将“abc”替换为“good”

s.replace(3,3,"good");

//查找第一个字符‘c’，返回下标值,查不到则返回 4294967295

s.find('c');

//用 compare()方法与其他字符串相比较。如果它比对方大，则返回 1；

//如果它比对方小，则返回-1；如果它与对方相同（相等），则返回 0

s.compare("cat");

reverse(s.begin(),s.end());

//将字符串分离成子串，分隔符为空格

sscanf("abc 123 pc","%s %s %s",sa,sb,sc);

//将字符串分离成数字，分隔符为“,”和“$”

//当用到数字的时候，跟 scanf 一样，它要传指针地址

int x,y,z;

sscanf("4,5$6","%d,%d$%d",&x,&y,&z);

//将数值转换为 string 的一种方法：(C 方法)

char b[10];

string a;

sprintf(b,"%d",1975);

a=b;

**#include <set>//集合容器 平衡二叉检索树**

set<int> s;

multiset<int> s;

s.insert(1);

s.erase(6);

s.size();

it=s.find(6);

//反向遍历集合中的元素

set<int>::reverse\_iterator rit;//定义反向迭代器

for(rit=s.rbegin();rit!=s.rend();rit++)

{

cout<<\*rit<<" ";

}

//自定义比较函数 myComp，重载“()”操作符

struct myComp {

bool operator()(const int &a,const int &b){

if(a!=b){

return a>b;

}

else{

return a>b;

}

}

};

set<int,myComp> s;

struct Info {

string name;

float score;

//重载“<”操作符，自定义排序规则

//按 score 由大到小排列。如果要由小到大排列，使用“>”号即可。

bool operator < (const Info &a) const {

return a.score<score;

}

};

set<Info> s;

**#include <stack>//堆栈容器**

//定义堆栈 s，其元素类型是整型

stack<int> s;

//元素入栈

s.push(1);

while(s.empty()!=true)//堆栈非空

{

cout<<s.top()<<" ";//读取栈顶元素

s.pop();//出栈（即删除栈顶元素）

}

**#include <list>//双向链表容器**

list<int> l;

//在链表尾部插入新元素，链表自动扩张

l.push\_back(2);

//在链表头部插入新元素，链表自动扩张

l.push\_front(8);

//删除值等于 1 的所有元素

l.remove(1);

//删除首元素

l.pop\_front();

//删除尾元素

l.pop\_back();

//采用 find()查找算法在链表中查找

it=find( l.begin(), l.end(),5 );

//使用 sort()方法对链表排序，是升序排列

l.sort();

//剔除连续重复元素（只保留一个）

l.unique();

**#include <map> //映照容器**

map<string,float> m;

//插入元素，按键值的由小到大放入黑白树中

m["Jack"]=98.5;

m["Bomi"]=96.0;

m["Kate"]=97.5;

cout<<(\*it).first<<" : "<<(\*it).second<<endl;

**#include <queue>//队列容器**

//定义队列，元素类型是整型

queue<int> q;

//入队，即插入元素

q.push(1);

//队首元素出队（删除队首元素）

q.pop();

//读取队首元素

cout<<q.front()<<endl;

//读取队尾元素

cout<<q.back()<<endl;

q.size();

q.empty();

**#include <queue>//优先队列容器**

//定义优先队列，元素类型为整型

priority\_queue<int> pq;

//入队，插入新元素

pq.push(1);

//读取当前队首元素

cout<<pq.top()<<" ";

//出队，删除队首元素

pq.pop();

#include <deque>//双端队列容器

deque<int> d;

//使用 push\_back()方法从尾部插入元素，会不断扩张队列

d.push\_back(2);

//从头部插入元素，不会增加新元素，只将原有的元素覆盖

d.push\_front(10);

//中间插入元素,不会增加新元素，只将原有的元素覆盖

d.insert(d.begin()+1,88);

//从头部删除元素

d.pop\_front();

//从尾部删除元素

d.pop\_back();

//从中间删除元素，erase 的参数是迭代器位置

d.erase(d.begin()+1);

//清空元素

d.clear();

**#include <bitset>//位集合容器**

bitset<100000> b;//必须要指定bitset容器的大小,大小一经定义，就不能修改了

//采用下标法给元素赋值

b[1]=1;

b[6]=1;

//采用 set()方法，一次性将元素设置为 1

b.set();

//直接向输出流输出全部元素

cout<<b<<endl;

//bitset方法

b.any() //b 中是否存在置为 1 的二进制位？

b.none() //b 中不存在置为 1 的二进制位吗？

b.count() //b 中置为 1 的二进制位的个数

b.size() //b 中二进制位的个数

b[pos] 访问 //b 中在 pos 处的二进制位

b.test(pos) //b 中在 pos 处的二进制位是否为 1？

b.set() 把 //b 中所有二进制位都置为 1

b.set(pos) //把 b 中在 pos 处的二进制位置为 1

b.reset() //把 b 中所有二进制位都置为 0

b.reset(pos) //把 b 中在 pos 处的二进制位置为 0

b.flip() //把 b 中所有二进制位逐位取反

b.flip(pos) //把 b 中在 pos 处的二进制位取反

b.to\_ulong() //用 b 中同样的二进制位返回一个 unsigned long 值

os << b //把 b 中的位集输出到 os 流

**其他函数:**

assign() //对Vector中的元素赋值

at() //返回指定位置的元素

back() //返回最末一个元素

begin() //返回第一个元素的迭代器

capacity() //返回vector所能容纳的元素数量(在不重新分配内存的情况下）

clear() //清空所有元素

empty() //判断Vector是否为空（返回true时为空）

end() //返回最末元素的迭代器(实指向最末元素的下一个位置)

erase() //删除指定元素

front() //返回第一个元素

get\_allocator() //返回vector的内存分配器

insert() //插入元素到Vector中

max\_size() //返回Vector所能容纳元素的最大数量（上限）

pop\_back() //移除最后一个元素

push\_back() //在Vector最后添加一个元素

rbegin() //返回Vector尾部的逆迭代器

rend() //返回Vector起始的逆迭代器

reserve() //设置Vector最小的元素容纳数量

resize() //改变Vector元素数量的大小

size() //返回Vector元素数量的大小

swap() //交换两个Vector

list:

assign() //给list赋值

back() //返回最后一个元素

begin() //返回指向第一个元素的迭代器

clear() //删除所有元素

empty() //如果list是空的则返回true

end() //返回末尾的迭代器

erase() //删除一个元素

front() //返回第一个元素

get\_allocator() //返回list的配置器

insert() //插入一个元素到list中

max\_size() //返回list能容纳的最大元素数量

merge() //合并两个list

pop\_back() //删除最后一个元素

pop\_front() //删除第一个元素

push\_back() //在list的末尾添加一个元素

push\_front() //在list的头部添加一个元素

rbegin() //返回指向第一个元素的逆向迭代器

remove() //从list删除元素

remove\_if() //按指定条件删除元素

rend() //指向list末尾的逆向迭代器

resize() //改变list的大小

reverse() //把list的元素倒转

size() //返回list中的元素个数

sort() //给list排序

splice() //合并两个list

swap() //交换两个list

unique() //删除list中重复的元素

stack:

empty() //堆栈为空则返回真

pop() //移除栈顶元素

push() //在栈顶增加元素

size() //返回栈中元素数目

top() //返回栈顶元素

queue:

back() //返回最后一个元素

empty() //如果队列空则返回真

front() //返回第一个元素

pop() //删除第一个元素

push() //在末尾加入一个元素

size() //返回队列中元素的个数

lower\_bound()//找到大于等于某值的第一次出现。

upper\_bound()//找到大于某值的第一次出现。

equal\_range()//找到（在不破坏顺序的前提下）可插入给定值的最大范围。

binary\_search()//在有序序列中确定给定元素是否存在。

next\_permutation()//按字典序的下一个排列。

prev\_permutation()//按字典序的前一个排列。

unique()//删除相邻的重复元素。

remove()//删除具有给定值的元素。

count()//在序列中统计某个值出现的次数。

find\_if()//在序列中找出符合某谓词的第一个元素。

find\_end()//在序列中找出一子序列的最后一次出现的位置

memset(arr, 0, sizeof(arr));//元素置零

getline(cin,str);//带空格整行输入str//保留两位小数点

#include<iomanip>

cout<<setiosflags(ios::fixed)<<setprecision(2)<<\*it;