STL

```
// ConsoleApplication1.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>
#include "afx.h"
#include "afxtempl.h"
#include <string>
#include <vector>
#include <map>
#include <list>
#include "algorithm" //包含了stl的排序算法
#include <queue>
using namespace std;

bool compare(int x, int y) {
    return x > y;
}
int main()
{

//std::string, 前面加了using namespace std，这里就不用加std::了

    string ss1 = "12-45";
    string ss2 = "67890";
    int pos = ss1.find('-');
    string left2 = ss1.substr(0, pos);
    string right2 = ss1.substr(pos + 1, 2);
    cout <<"left="<< left2 << endl;
    cout <<"right="<< right2 << endl;
    //字符倒叙的2种方法
    //方法1，自己遍历,从右往左,迭代器定义
    string svert;
    for (string::reverse_iterator it = ss1.rbegin(); it != ss1.rend(); it++)
    {
        svert += *it;
    }
    cout << svert<<endl;
    //另外一种遍历方式
    for (int j = 0; j < ss2.size(); j++) {
        cout << ss2[j];
    }
    cout << endl;
    //方法2，使用通用的reverse函数
    reverse(ss1.begin(), ss1.end()); //直接调用reverse
    //另一个例子，普通数组逆序
    cout << ss1<<endl;
    int a[5] = { 1,2,3,4.5 };
    reverse(a, a + 2);
    for (int i = 0; i < 5; i++) {
        printf("%d\n", a[i]);
    }
    //也可以用index遍历字符串
    string::iterator it2 = ss1.begin();
```

```
    for (int i = 0; i < ss1.length(); i++) {
        cout << it2[i] << endl;
    }

    //vector 数组

    vector<int> tv = { 1,2,3,4,5 };  //初始化很方便
    vector<int>::iterator itv; //正向迭代器，从左往右
    vector<int>::reverse_iterator ritv;  //反向迭代器，从右往左

    itv = tv.begin();
    cout << *itv; //结果为1，表明begin()指向第一个元素
    itv = tv.end();

    try { cout << *itv; }//这里会抛异常，说明end()指向了最后一个元素的后面,即无效元素}
    catch (exception e) { cout << "can't asign value to .end()"; }

    //再看看reverse_iterator的情况
    ritv = tv.rbegin();
    cout << *ritv; //这里指向5
    ritv = tv.rend();

    try {cout << *ritv;}//同样这里会抛异常
    catch (exception e) { cout <<"cant asign value to .rend()" ; }

    //因此，vector的遍历方法为
    for (itv = tv.begin(); itv != tv.end(); itv++)
    {
        //do your work
    }

    //第一个元素有一个专门的函数 front(),就不用迭代器了，最后一个元素是back()
    cout << tv.front() << " " << tv.back() << endl;
    //插入操作,在迭代器之前插入
    itv = tv.begin();
    tv.insert(itv, 9); //因为itv指向第一个元素，所以这里的意思就是放在最前面
    tv.insert(tv.end(), 10); //因为end指向最后一个元素的后面，所以这里来实际上是放在最后
    //还有两个专门的命令放在最后和删除最后一个元素
    tv.push_back(11);
    tv.pop_back();
    //奇葩的是，为什么不设计两个对应的放在最前面/删除最前面的命令??
    //删除一个元素，用erase
    tv.erase(tv.begin());
    //但不能用 tv.erase(tv.end()), 因为end指向的是最后一个元素的后面，即无效的， 所以删除最后一个元素用 pop_back()， 或者使用反向迭代器的第一个元素erase(tv.rbegin())

    //删除第3个元素怎么操作？
    tv.erase(tv.begin() + 2);
    //读写元素怎么操作？??
    tv[0] = 1;
    int val = tv[5];
    //所以遍历的另一种方法是
    for(int i = 0; i < tv.size(); i++)
    {
        cout << tv[i]; //这看起来才像数组,哈哈哈哈哈，也是最常用的。 只有删除元素才需要用到迭代器
    }
    //问题，如果使用迭代器遍历，怎么知道当前遍历到的是第几个元素??
```

```
    for (itv = tv.begin(); itv != tv.end(); itv++) {
        //当前是第几个元素呢？ 使用迭代器左减法
        cout <<"index "<< itv - tv.begin()<<endl;
    }

    //排序
    sort(tv.begin(), tv.end(),greater<int>()); //默认升序，这里改成降序，如果是降序用
    for (int i = 0; i < tv.size(); i++) {
        cout << tv[i]<<",";
    }
    cout << endl;

    //如果想按自己的规则排序怎么办? 自己写比较大小函数

    sort(tv.begin(), tv.end(), compare); //只要元素实现了小于号<的重载，那么就可以自动排序，否在就自己写compare函数，定义小于<这个比较函数，compare是前面自己写的函数
    //但我在compare里面变成了return a>b 而不是return a<b，所以就实现了按照从大到小的顺序排列
    //很重要的一点是，这里sort前两个函数分别是 tv.begin()和tv.end()，而.end()指向无效数据，这说明什么？？说明这里的参数是个前闭后开的，假如
    sort(tv.begin(), tv.begin() + 3); //这里的意思就是把 0，1，2三个元素重新按照升序排列了，并不包括第3个元素。

    //list
    list<int> list2;
    list2.push_front(1);
    list2.push_back(2);
    list2.pop_back(); //删除元素
    list2.pop_front(); //删除元素
    if(!list2.empty()) list2.back(); //最后一个元素
    if(!list2.empty()) list2.front(); //第一个元素
    //遍历呢？ 用 迭代器 begin/end/rbegin/rend
    list<int>::iterator it = list2.begin();
    while (it != list2.end()) {
        cout << *it;
        it++;
    }

    //插入呢?
    list2.insert(it, 3);
    list2.push_back(2);
    list2.push_front(3);

    //查找并删除（值删除)
    list2.remove(1); //把所有为1的元素删除,remove()函数删除链表中所有值为val的元素
    //删除指定位置,用迭代器
    if(it!=list2.end()) list2.erase(it);
    //逆序
    list2.reverse();
    //排序
    list2.sort(compare); //按照自己的规则doif排序，默认为升序，注意是升序，如果想降序，自己加函数*-1即可,这里如果是数字，不用加doif，doif用于自定义的排序

    for (list<int>::iterator it = list2.begin(); it != list2.end(); it++)
    {
        cout << *it;
    }
```

```
//Map, Map是key和value的集合，内部使用哈希算法，所以存取时间非常快
    map<int, string> mapstl;
    map<int, string>::iterator imapstl;
    mapstl[5] = "eric";
    mapstl[2] = "dongbao";
    mapstl[1990] = "lisa";
    mapstl[400] = "magi";
    mapstl[9999] = "eric";
    //判断一个key是否在map里面
    if (mapstl.find(1991) == mapstl.end()) { //如果找不到则返回最后尾巴的迭代器指针
        printf("not found");
    }
    //remove a element
    mapstl.erase(100);
    mapstl.erase(849585);

    //遍历,遍历，遍历，因为map有两个元素，这两个元素凑成一个pair，所以用->firt ,->second来表示key和value
    for (imapstl = mapstl.begin(); imapstl != mapstl.end(); imapstl++) {
        cout << imapstl->first <<" "<<imapstl->second<<endl;
    }

    //map什么时候用，举个栗子， 一个字符串里面，只有一个字符是不重复的，请找出这个字符，可以这样写

    map<char,int> cntmap;
    string str = "11223345566660000";
    for (int i = 0; i < str.size(); i++) {
        if (cntmap.find(str[i]) == cntmap.end()) {
            cntmap[str[i]] += 1;
        }
        else {
            cntmap[str[i]] = 1;
        }
    }
    //到这里，发现map没有提供按照value查找的函数，是不是很坑，还要重新再遍历一遍。
    //另外，map有一个函数叫count，千万不要以为是统计value的个数， 不是！！！，是统计key的个数，而一般的map，key是唯一的，所以只能返回0或1，很坑的函数，不要用。


    //有序队列，不用自己排队了，一边插入一边排序，效率很高,是非常强大的一个数据结构，默认top表示最大值

    priority_queue<int> pqueue;
    pqueue.push(5);
    pqueue.push(4);
    pqueue.push(1);
    pqueue.push(2);

    int v=pqueue.top();
    cout << v << endl; //输出5


    //priority_queue<int, greater<int> > q;  //如果top要求返回最小的，则定义的时候加第2个参数 greater<int>
    //priority_queue<int,less<int> > q;
```

```
    priority_queue<int,vector<int>, greater<int>> pqueue2;
    pqueue2.push(3);
    pqueue2.push(1);
    pqueue2.push(2);
    cout << pqueue2.top(); //这里输出1


}

// 运行程序: Ctrl + F5 或调试 >“开始执行(不调试)”菜单
// 调试程序: F5 或调试 >“开始调试”菜单

// 入门使用技巧:
//   1. 使用解决方案资源管理器窗口添加/管理文件
//   2. 使用团队资源管理器窗口连接到源代码管理
//   3. 使用输出窗口查看生成输出和其他消息
//   4. 使用错误列表窗口查看错误
//   5. 转到“项目”>“添加新项”以创建新的代码文件，或转到“项目”>“添加现有项”以将现有代码文件添加到项目
//   6. 将来，若要再次打开此项目，请转到“文件”>“打开”>“项目”并选择 .sln 文件
```