## myAlgorithm.cpp

#include <iostream>   
#include <string>   
#include <algorithm>   
#include "myAlgorithm.h"   
   
/\*   
（1）简单查找算法   
 （1） find(beg, end, val) //返回迭代器，若找到指向指定元素迭代器   
 （2） find\_if(beg, end, unaryPred) //返回第一个满足unaryPred的元素 否则尾后迭代器   
 （3） count(beg, end, val) //返回一共有多少个   
 （4） count\_if(beg, end, val) //满足条件的一共有多少个   
 （5） search(beg1, end1, beg2, end2) //子序列2在序列1中所处的位置   
 （6） find\_first\_of(beg1, end1, beg2, end2) //返回第二个序列任意元素在第一个范围内出现的位置   
 （7） find(beg1, end1, beg2, end2) //和search相反，返回最后一个出现的子序列的位置   
   
（2）其他只读算法   
 （1）for\_each(beg, end, unaryOp) //对每个元素使用可调用对象   
 （2）equal(beg1, end1, beg2) //如果输入序列每个元素都和beg2开始的序列相等，则返回true   
   
（3）二分搜索算法   
 （1）lower\_bound(beg, end, val) //返回指向第一个小于等于val的迭代器   
 （2）upper\_bound(beg, end, val) //返回指向第一个大于val的迭代器   
 （3）equal\_range(beg, end, val) //返回以上一个pair包含以上两个函数返回参数   
   
（4）写容器算法   
 （1）暂不总结；   
   
（5）划分算法   
 （1）is\_partitioned(beg, end, unaryPrey) //若满足谓词的在前，不满足在后，则返回true, 空也是true   
 （2）partitioned\_copy(beg, end, dest1, dest2, unaryPred)   
 //将满足谓词的元素放在dest1中，将不满足拷贝在dest2中，返回一个pair   
 //,first指向dest1的末尾，second指向dest2的末尾   
   
（6）排序算法   
 （1）sort(beg, end) //给容器排序   
 （2）stable\_sort(beg, end, comp) //稳定排序   
 （3）is\_sorted(beg, end) //返回bool，表示是否有序   
 （4）is\_sorted\_until(beg, end) //返回最长有序子序列的尾后迭代器   
 （5）remove(beg, end, val) //删除元素val，返回指向删除最后一个元素的迭代器   
 （6）remove\_if(beg, end, unaryPred) //删除满足谓词的元素，返回指向删除最后一个元素的迭代器   
 （7）unique(beg, end, val) //重排元素，对于重复元素重新排在最大不重复子序列的尾后迭代器，一般sort之后   
 //使用该元素将重复元素放在最大不重复子序列的后面，然后可以使用erase删除   
 （8）reverse(beg, end) //翻转序列   
   
（7）最大最小值算法   
 （1）min(val1, val2)   
 （2）max(val1, val2)   
 （3）min\_element(beg, end) //返回指向最小值的迭代器   
 （4）max\_element(beg, end) //返回指向最大值的迭代器   
 （5）minmax\_element(beg, end) //返回pair 指向（min, max）的两个迭代器   
   
（8）数值算法   
 （1）accumulate(beg, end, init) //求容器和，init设定为初值，返回和   
\*/   
   
   
void myAlgorithmTest()   
{   
 std::cout << "----------------------------this is class Algorithm demo------------------------" << std::endl;   
   
}

## myArray.cpp

#include <iostream>  
#include <string>  
#include <array>  
#include "myArray.h"  
/\*  
（1）构造：std::array<int, 5> myArr = {2, 4, 6};  
（2）通用操作：size(), maxsize(), empty(), operator[](), at(), front(), back(), swap(),   
（3）特有操作：data(), fill()  
\*/  
void myArrayTest()  
{  
 std::cout << "--------this is class array demo----------" << std::endl;  
 std::array<int, 10 >myArray = { 3, 5 };  
 for (auto tmp : myArray)  
 {  
 std::cout << tmp << std::ends;  
  
 }  
 std::cout << '\n';  
 //没啥好讲的这一节：  
  
 auto p = myArray.data(); //返回数组第一个元素的指针  
 std::cout << \*p << std::endl;  
  
}

## myDeque.cpp

#include <iostream>  
#include <string>  
#include <deque>  
#include "myDeque.h"  
  
/\*  
 //特点：支持随机访问，可以在内部进行插入和删除操作，在两端插入删除性能最好，  
 （1）创建：  
 （2）通用操作：push\_back(), pop\_back(), insert(), erase(),   
 clear(), swap(), empty(), back(), front(), at(), []  
 （3）特有操作：push\_front(), pop\_front()  
\*/  
void myDequeTest()  
{  
 std::cout << "-----this is class deque demo--" << std::endl;  
 std::deque<int >myDeque = { 3, 5, 7 };  
 for (auto tmp : myDeque)  
 {  
 std::cout << tmp << std::ends;  
 }  
 std::cout << '\n';  
  
 myDeque.push\_back(9); //在队列尾部添加元素  
 myDeque.push\_front(1); //在队列头部添加元素  
 std::cout << "after pushing, myDeque is: \n";  
 for (auto tmp : myDeque)  
 {  
 std::cout << tmp << std::ends;  
 }  
 std::cout << '\n';  
  
 /\*  
 insert()版本：  
 （1）insert(p, t) //在迭代器p之前创建一个值为t，返回指向新添加的元素的迭代器  
 （2）insert(p, b, e) //将迭代器[b, e）指定的元素插入到p所指位置，返回第一个插入元素的迭代器  
 //关于迭代器确定范围都是左闭右开！！！！  
 \*/  
 auto ret = myDeque.insert(myDeque.begin() + 1, 2);  
 std::cout << "after inserting, myDeque is: \n";  
 for (auto tmp : myDeque)  
 {  
 std::cout << tmp << std::ends;  
 }  
 std::cout << '\n';  
 std::cout << "返回迭代指向的元素为：";  
 std::cout << \*ret << std::endl;  
  
 auto ret1 = myDeque.erase(ret);  
 std::cout << "after earsing, myDeque is: \n";  
 for (auto tmp : myDeque)  
 {  
 std::cout << tmp << std::ends;  
 }  
 std::cout << '\n';  
 std::cout << "返回迭代指向的元素为：";  
 std::cout << \*ret1 << std::endl;  
  
}

## myForward\_list.cpp

#include <iostream>  
#include <string>  
#include <forward\_list>  
#include <iterator> //其中的advance函数，可以移动迭代器移动指定长度；  
#include "myForward\_list.h" //单向链表  
#include "myList.h"  
/\*  
 （1）构造：  
 （2）通用函数：empty(), front(）, swap(), clear(),   
 push\_front(), pop\_front(), reverse(),   
 （3）特有函数：inert\_after(), erase\_after(), before\_begin(),   
 remove(), remove\_if(), unique(), sort(), merge(),   
  
\*/  
  
void myForward\_listTest()  
{  
 std::cout << "----this is class forward\_list demo---" << std::endl;  
 std::forward\_list<int> myForward\_list = { 1, 3, 7, 5, 5 };  
 std::cout << "the front of myForward\_list is : " << myForward\_list.front() << std::endl; //front数  
  
 myForward\_list.push\_front(0);  
 for (auto tmp : myForward\_list)  
 std::cout << tmp << ' ';  
 std::cout << '\n';  
 /\*  
 lst.insert\_after(p, t) //在迭代器p之后的位置插入元素t，返回指向插入元素的迭代器  
 lst.insert\_after(p, b, e) //在迭代器p之后插入范围为[b, e）的元素，返回最后一个插入链表的迭代器  
 \*/  
 auto iter = myForward\_list.before\_begin();  
 std::advance(iter, 1);   
 //另外还可以使用iterator中的advance函数对迭代器进行偏移  
 myForward\_list.insert\_after(myForward\_list.before\_begin(), 9);   
 //list 和forward\_list虽然不支持+,-操作，但是支持++， （--）操作  
 std::cout << "after inserting , myForward\_list is : " << std::endl;  
 for (auto iter = myForward\_list.begin(); iter != myForward\_list.end(); ++iter)   
 {  
 std::cout << \*iter << std::ends;  
 }  
 std::cout << '\n';  
  
 myForward\_list.erase\_after(myForward\_list.before\_begin());   
 //在迭代器p之后的位置插入元素t，返回指向插入元素的迭代器  
 std::cout << "after erasing , myForward\_list is : " << std::endl;   
 for (auto iter = myForward\_list.begin(); iter != myForward\_list.end(); ++iter)  
 {  
 std::cout << \*iter << std::ends;  
 }  
 std::cout << '\n';  
   
 //myForward\_list.remove(9); //删除某一特定值元素  
 //myForward\_list.remove\_if(is\_odd); //按照传入谓词来删除某一元素  
  
 myForward\_list.unique(); //踢出重复数据  
 std::cout << "after unique , myForward\_list is : " << std::endl;  
 for (auto iter = myForward\_list.begin(); iter != myForward\_list.end(); ++iter)  
 {  
 std::cout << \*iter << std::ends;  
 }  
 std::cout << '\n';  
  
 myForward\_list.sort(); //对链表数据进行排序  
 std::cout << "after sorting , myForward\_list is : " << std::endl;   
 for (auto iter = myForward\_list.begin(); iter != myForward\_list.end(); ++iter)  
 {  
 std::cout << \*iter << std::ends;  
 }  
 std::cout << '\n';  
  
  
}

## myList.cpp

#include <iostream>  
#include <list>  
#include <string>  
#include "myList.h"  
  
/\*  
 (1)构造：  
 (2)通用函数：push\_back(), pop\_back(), empty(), clear(), swap(), insert(), erase(), reverse()  
 (3)特有操作：push\_front(), pop\_front(), merge(), remove(), remove\_if()  
\*/  
  
bool is\_odd(const int x) //充当一元谓词  
{  
 return (x % 2 == 1);  
}  
void myListTest()  
{  
 std::cout << "------this is class List demo----" << std::endl;  
 std::list<int >myList{ 3, 5 };  
 myList.push\_back(7);  
 myList.push\_front(1); //双向链表支持在两端的快速插入和删除  
 std::cout << "after push, myList is : " << std::endl;  
 for (auto tmp : myList)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
  
 //插入操作和其他的一样，在此不再赘述  
 auto ret = myList.insert(myList.end(), 9);   
 std::cout << "after inserting, myList is : " << std::endl;  
 for (auto tmp : myList)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
  
 std::list<int >myList1 = { 2, 4, 6, 8 };  
 myList.merge(myList1); //合并两个有序链表,且元素无重复，返回合并后的排序链表，若二者其一就会出错  
 std::cout << "after merging, myList is : " << std::endl;  
 for (auto tmp : myList)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
  
 myList.remove(8); //删除所有满足等于8的元素  
 std::cout << "after removing, myList is : " << std::endl;  
 for (auto tmp : myList)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
  
 //remove\_if() //当满足条件时，删除  
 myList.remove\_if(is\_odd);  
 std::cout << "after remove\_if(is\_odd), myList is : " << std::endl;  
 for (auto tmp : myList)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
}

## myMap.cpp

#include <iostream>  
#include <string>  
#include <map>  
#include <iterator>  
#include <stdio.h>  
#include "myMap.h"  
/\*  
 （1）构造：std::map<string, int>, std::pair<string, int>, std::make\_pair(v1, v2)  
 （2）特殊操作：first, second, insert(), erase(), count(), find(), operator[]();   
\*/  
void myMapTest()  
{  
 std::cout << "---------this is class map demo---------" << std::endl;  
 std::map<std::string, int>myMap;  
   
 auto myPair1 = std::pair<std::string, int>("hello", 1);  
 auto myPair2 = std::make\_pair("world", 10);  
  
 /\*  
 //insert返回一个pair，first是一个迭代器指向具有给定关键字的值，second是一个bool量，  
 若是true则表示插入成功，  
 //若为false则表示插入失败，说明已经存在，则该语句什么也不做，只能手动的++  
 m.insert(e); //插入pair对象  
 m.insert(beg, end); //将范围内的元素插入  
 m.insert(iter, e); //unknow  
 \*/  
 auto ret1 = myMap.insert(myPair1);   
 if (ret1.second)  
 std::cout << "insert successfully\n"; //插入pair类型  
 printf("%s---->%d\n", ret1.first->first.c\_str(), ret1.first->second);  
 auto ret2 = myMap.insert(myPair2);  
 if (ret2.second)  
 std::cout << "insert successfully\n";  
 printf("%s---->%d\n", ret2.first->first.c\_str(), ret2.first->second);  
  
 for (auto iter = myMap.begin(); iter != myMap.end(); ++iter)  
 {  
 printf("%s---->%d\n", iter->first.c\_str(), iter->second);  
 }  
 printf("\n");  
  
 ++myMap["word"]; //使用下标运算符，若不存在，则创建新的键值对(word, 0)  
 std::cout << "after operator[], myMap is : \n";  
 for (auto iter = myMap.begin(); iter != myMap.end(); ++iter)  
 {  
 printf("%s---->%d\n", iter->first.c\_str(), iter->second);  
 }  
 /\*  
 m.erase(k); //删除关键字为k的元素  
 m.erase(p); //删除迭代器p指向的元素  
 m.erase(b, e); //删除范围内的元素  
 \*/  
 myMap.erase("word");  
 std::cout << "after erasing, myMap is : \n";  
 for (auto iter = myMap.begin(); iter != myMap.end(); ++iter)  
 {  
 printf("%s---->%d\n", iter->first.c\_str(), iter->second);  
 }  
  
 unsigned cnt = myMap.count("hello"); //返回关键字的多了  
 printf("hello occurred %d times\n", cnt);  
 std::cout << "find keyword hello : \n";  
 auto ret3 = myMap.find("hello"); //返回关键字的迭代器  
 printf("%s---->%d\n", ret3->first.c\_str(), ret3->second);  
  
  
}

## mySet.cpp

#include <iostream>  
#include <stdio.h>  
#include <string>  
#include <set>  
  
#include "mySet.h"  
  
/\*  
 //所有元素都是按照字典序自动排序，set只有键值，键值就是市值  
 （1）构造：  
 （2）通用操作：empty(), insert(), erase(), size(), swap(), find(), count(),   
 （3）特有操作：equal\_range(), lower\_bound(), upper\_bound()  
\*/  
  
void mySetTest()  
{  
 std::cout << "----this is class set demo-------" << std::endl;  
 std::set<std::string >mySet = {"a", "b", "c"};  
  
 /\*  
 //insert返回一个pair，first是一个迭代器指向具有给定关键字的值，second是一个bool量，  
 若是true则表示插入成功，  
 //若为false则表示插入失败，说明已经存在，则该语句什么也不做  
 m.insert(e); //插入pair对象  
 m.insert(beg, end); //将范围内的元素插入  
 m.insert(iter, e); //unknow  
 \*/  
  
 bool flag = mySet.empty();  
 auto ret = mySet.insert("d"); //插入键值  
 std::cout << "after inserting, mySet is : \n";  
 if (ret.second)  
 {  
 for (auto tmp : mySet)  
 std::cout << tmp << std::ends;  
 };  
 std::cout << "\n";  
 auto ret1 = mySet.erase("d"); //删除元素  
 for (auto tmp : mySet)  
 std::cout << tmp << std::ends;  
 std::cout << "\n";  
  
 auto ret2 = mySet.find("a");  
 if (ret2 == mySet.end())  
 std::cout << "dont find it \n";  
 else  
 std::cout << "find it \n";  
  
 unsigned int cnt = mySet.count("a");  
 printf("a occurred %d times", cnt);  
  
 std::set<std::string>::iterator iter\_beg = mySet.lower\_bound("a");  
 std::set<std::string>::iterator iter\_end = mySet.upper\_bound("a");   
 //给定关键字的范围[lower\_bound, upper\_bound)  
  
}

## myStackAndQueue.cpp

#include <iostream>  
#include <string>  
#include <stack>  
#include <queue>  
#include <stdio.h>  
  
#include "myStackAndQueue.h"  
  
/\*  
 （1）构造：std::atack<int >myStack;  
 （2）通用操作：empty(), size(),  
 （3）特有操作：pop(), push(),top(),  
\*/  
void myStackTest()  
{  
 std::cout << "----------this is class stack demo---------------" << std::endl;  
 std::stack<int >myStack;  
  
 myStack.push(1);  
 myStack.push(2);  
 myStack.push(3); //元素进栈  
   
  
 int top\_num = myStack.top(); //栈顶元素  
 std::cout << "the num of myStack is: " << top\_num << std::endl;  
 myStack.pop(); //弹出栈顶元素  
 int top\_num1 = myStack.top();  
 std::cout << "the num of myStack is: " << top\_num1 << std::endl;  
   
}  
/\*  
 （1）构造： std::queue<int> myQueue;  
 （2）通用操作：empty(), size(), front(), back(),   
 （3）特有操作：push(), pop(),   
\*/  
void myQueueTest()  
{  
 std::cout << "---------------this is class queue demo-----" << std::endl;  
 std::queue<int > myQueue;  
 myQueue.push(1);  
 myQueue.push(2);  
 myQueue.push(3); //入队列  
 int front\_num = myQueue.front(); //队列头元素  
 int back\_num = myQueue.back(); //队列尾元素  
 printf("the front and back num of the myQueue : %d and %d \n", front\_num, back\_num);  
  
 myQueue.pop(); //弹出元素不返回元素  
 myQueue.pop();  
 int front\_num1 = myQueue.front();  
 int back\_num1 = myQueue.back();  
 printf("the front and back num of the myQueue : %d and %d \n", front\_num1, back\_num1);  
  
  
  
  
  
}

## myString.cpp

/\*  
实现string常见的操作：  
（1）构造、赋值:  
（2）基本操作：size(), empty(), push\_back(), pop\_back(), insert(), erase(), clear();  
（3）string特有操作：substr(), append(), replace(), find()系列函数, compare()  
\*/  
#include <iostream>  
#include <string>  
#include <string.h>  
#include "myString.h"  
  
  
void myStringTest()  
{  
 std::cout << "---------this is class string demo--------------------------" << std::endl;  
 std::string str1 = "hello world";  
 std::cout << "the length of str1：" << str1.size() << std::endl;  
 str1.push\_back('!'); //在尾部添加字符  
 std::cout << "after pushing, str1 is: " << str1 << std::endl;  
   
 auto ret = str1.insert(str1.begin() + 5, '@');   
 //接受迭代器版本返回的是指向插入元素的迭代器  
 std::cout << "after inserting, str1 is : " << str1 << std::endl;  
 std::cout << "插入返回的迭代器所指向的值为：" << \*ret << std::endl;  
  
 auto ret1 = str1.insert(6, "#"); //参数为下标的，插入的是字符串，  
 //返回的是插入之后str1的引用,记住字符串下标是从0开始的  
 std::cout << "after inserting, str1 is : " << str1 << std::endl;  
 std::cout << "插入返回的值为：" << ret1 << std::endl;  
  
 auto ret2 = str1.erase(str1.begin() + 5); //接受迭代器版本的，返回删除元素之后的迭代器  
 std::cout << "after earsing , str1 is: " << str1 << std::endl;  
 std::cout << "返回的迭代器指向元素为：" << \*ret2 << std::endl;  
  
 auto ret3 = str1.erase(5, 1); //接受下标参数的，,接受删除长度，返回删除元素之后的引用  
 std::cout << "after earsing , str1 is: " << str1 << std::endl;  
 std::cout << "返回的迭代器指向元素为：" << ret3 << std::endl;  
 //string s.substr(pos, n) //n若缺失，到结尾  
 std::string str2 = str1.substr(0, 5); //从0开始，长度为5的子串，若5缺失，则默认到末尾  
 std::cout << "after substr, str2 is :" << str2 << std::endl;  
  
 str1.append("!!!"); //在尾部添加子串  
 std::cout << "after appending, str1 is :" << str1 << std::endl;  
  
 //string &s.replace(range, args) //将range范围内的元素，替换为args，可以不一样长  
 //举例几种常见的：  
 str1.replace(5, 1, "###"); //(pos, len, str)形式  
 std::cout << "after replace, str1 is : " << str1 << std::endl;  
  
 str1.replace(str1.begin() + 5, str1.begin() + 7, "%%%"); //(iterator1, iterator2, string)形式  
 std::cout << "after replace, str1 is : " << str1 << std::endl;  
  
 //s.compare() ，目前就记住这一个就行，多了也记不住  
 str1.compare(str2);  
  
 /\*  
 find系列函数：  
 (1) s.find(args) //查找s中第一次出现的位置,并返回子串在主串中第一个字符的下标  
 (2) s.rfind(args) //查找最后一个...  
 (3) s.find\_first\_of(args) //在s中查找args中任何一个字符的第一次出现的位置，  
 (4) s.find\_last\_of(args) //在s中查找args中任何一个字符最后一次出现的位置。  
 (5) s.find\_first\_not\_of(args) //在s中查找第一个不在args中的字符  
 (6) s.find\_last\_not\_of(args) //查找最后一个不在args中的字符  
 \*/  
 auto pos = str1.find(str2);  
 std::cout << "str2 is at the pos : " << pos << " of str1" << std::endl;  
 std::string numbers("01234556789");  
 std::string name("r2d2");  
 auto pos1 = name.find\_first\_of(numbers);  
 std::cout << "number first at the pos : " << pos1 << " of name" << std::endl;  
  
 if (str1.compare(str2))   
 std::cout << "str1 > str2" << std::endl;  
}

## myVector.cpp

#include <iostream>  
#include <string>  
#include <vector>  
#include "myVector.h"  
#include "myVector.h"  
/\*  
实现string常见的操作：  
 （1）构造、赋值:  
 （2）基本操作：size(), empty(), push\_back(), pop\_back(), insert(), erase(), clear(), swap();  
\*/  
  
void myVectorTest()  
{  
 std::cout << "-----------this is class vector demo------" << std::endl;  
 std::vector<int> myVec = { 1, 3, 5, 7, 9 };  
 std::vector<int> myVec1 = { 4, 6, 8 };  
 std::cout << "the length of myVec：" << myVec.size() << std::endl;  
 myVec.push\_back(0);   
 //在尾部添加字符  
 std::cout << "after pushing, myVec is: " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 /\*  
 insert()版本：  
 （1）insert(p, t) //在迭代器p之前创建一个值为t，返回指向新添加的元素的迭代器  
 （2）insert(p, b, e) //将迭代器[b, e）指定的元素插入到p所指位置，返回第一个插入元素的迭代器  
 （3）insert(p, il) //将列表中的元素插入，返回第一个插入元素的迭代器  
 //关于迭代器确定范围都是左闭右开！！！！  
 \*/  
 auto ret = myVec.insert(myVec.begin() + 1, 2); //插入单个元素，返回该元素迭代器  
 std::cout << "after inserting , myVec is : " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << "返回的迭代器值为：" << \*ret << std::endl;  
  
 auto ret1 = myVec.insert(myVec.begin() + 1, myVec1.begin(), myVec1.end());   
 //插入系列元素，返回第一个插入元素迭代器  
 std::cout << "after inserting , myVec is : " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << "返回的迭代器值为：" << \*ret1 << std::endl;  
 /\*  
 insert()版本：  
 （1）erase(p) //删除迭代器p所指元素，返回下一个元素的迭代器  
 （2）erase(b, e) //删除迭代器[b, e) 范围内的元素；  
 //关于迭代器确定范围都是左闭右开！！！！  
 \*/  
 auto ret2 = myVec.erase(myVec.begin() + 1);   
 //删除单个迭代器指向的元素，  
 std::cout << "after earsing , myVec is : " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << "返回的迭代器值为：" << \*ret2 << std::endl;  
  
 auto ret3 = myVec.erase(myVec.begin() + 1, myVec.begin() + 4);   
 //返回迭代器对指向的范围内的元素  
 std::cout << "after earsing , myVec is : " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << "返回的迭代器值为：" << \*ret3 << std::endl;  
  
 std::swap(myVec, myVec1);   
 //交换两个容器的值，其实实质上并不交换  
 std::cout << "after swapping , myVec is : " << std::endl;  
 for (auto tmp : myVec)  
 {  
 std::cout << tmp << ' ';  
 }  
 std::cout << '\n';  
  
}

## STL.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Copyright: wuyong  
Author: wuyong  
Date: 2018-05-16  
Description:本例程提供了C++的STL常用数据结构及其算法的使用范例，为面试笔试编程题提供便利  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
#include <iostream>   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*顺序容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include <string> //和vector是一样的，支持快速随机访问，在尾部之外的其他的位置插入都很慢  
#include <vector> //可变大小数组，支持快速随机访问，在尾部之外的其它位置插入或者删除元素可能很慢  
  
#include <list> //双向链表，只支持双向顺序访问，在list中的任意位置插入和删除都很快  
  
//forward\_list单向链表设计目标是达到与手写单向链表相当的性能。  
#include <forward\_list> //单向链表，只支持单向顺序访问，在链表任意位置插入和删除都很快,是c++11新加的标准  
  
//在queue的中间位置插入或者删除元素代价都很高  
#include <deque> //双端队列，支持快速随机访问（肯定是顺序存储式队列），从头尾位置处插入和删除速度很快  
  
//与内置数组相比，更加安全和方便；  
#include <array> //固定数组大小，支持快速，不能添加或者删除元素，是c++11新加的标准，支持对象赋值或者拷贝操作  
  
//是一种容器适配器实现的栈结构  
#include <stack> //栈结构，支持栈顶的快速进栈出栈操作，在栈的其他部位不可操作。  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*顺序容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*关联容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include <map>   
#include <set>  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*关联容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*无序容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*无序容器\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*----------------------------------------容器所共同支持的操作--------------------------------------------  
1）类型别名  
 1. iterator/const\_iterator/size\_type/difference\_type/reference/const\_reference  
  
2）构造函数  
 1. C c //调用默认构造函数，无参  
 2. C c1(c2) //调用复制构造函数，有参，可合成  
 3. C c(b, e) //调用构造函数，带参，迭代器b,e指向的容器范围进行初始化构造  
 4. C c{a, b, c, ......} //列表初始化构造，带参构造  
  
3）赋值与swap  
 c1 = c2  
 c1 = {a, b, c, ......}  
 a.swap(b) //成员函数版的交换函数  
 swap(a, b) //非成员函数版的交换函数  
  
4）大小  
 1. c.size() //求容器大小，  
 2. c.max\_size() //容器最多可保存的数据  
 3. c.empty() //返回容器是否为空  
  
5）添加删除元素  
 1. c.insert(args) //插入元素  
 2. c.emplace(inits) //使用inits构造c中的一个元素  
 3. c.erase(args) //删除元素  
  
6）获取迭代器  
 1. c.bengin() c.end()  
 2. c.cbegin(), c.cend()  
 反向迭代器的成员  
 reverse\_iterator  
 const\_reverse\_interator  
 3. c.rbegin(), c.rend()  
 4. c.crbegin(), c.crend()  
-----------------------------------------------------------------------------------------------------------\*/  
  
  
/\*-------------------------------------------选择顺序容器的准则：-------------------------------------------  
  
 1）若要求支持随机访问,vector queue  
 2）程序要求在中间插入或者删除元素，则选择使用list 或者forward\_list  
 3）如果程序只会在头部或者尾部插入删除数据，则选择使用queue  
 4）如果不知道选择哪种容器，则在程序中只使用list和vector容器，并且只使用迭代器而不使用下标操作  
-----------------------------------------------------------------------------------------------------------\*/  
  
  
/\*--------------------------------------------容器的定义及初始化--------------------------------------------  
C c 默认构造函数，若c是个array，则执行默认初始化，若是vector等则是空容器  
C c1(c2) c1初始化为c2的拷贝，必须是同种类型，且保存的相同的元素类型，若是array，两者还必须是相同的大小  
C c1 = c2 同上  
  
C c{a, b, c,...} 对于array，列表中的元素必须少于array的大小  
C c = {a, b, c,...}  
  
C c(b, e) c初始化为迭代器b,e指定范围中元素的拷贝  
  
//只有顺序容器才支持的操作：  
C seq(n) seq包含n个元素,这些元素进行了值初始化  
C seq(n, t) seq包含n个初始化为值t的元素  
  
//关于array类型：  
array<int, 40>   
-----------------------------------------------------------------------------------------------------------\*/  
  
  
/\*--------------------------------------------赋值和swap()--------------------------------------------  
c1 = c2 c2向c1拷贝  
  
c = {a, b, c,...} 列表赋值， 而array不适用，因为array没有定义隐式转换的构造函数？  
swap(c1, c2) 交换两个容器的元素  
c1.wap(c2) 同上  
  
//assign 操作不适合关联容器和array  
seq.assign(b, e) 使用迭代器b，e替换容器中的元素  
seq.assign(il) 使用列表进行复制  
seq.assign(n, t) 使用n个值为t的元素。  
  
//使用swap() 交换两个容器的值，不对元素进行拷贝工作，所以速度很快，除array以外  
 统一使用非成员swap(）是一个很好的选择  
vector<string> svec1(10);  
vector<string> svec2(20);   
swap(svec1, svec2);   
-----------------------------------------------------------------------------------------------------------\*/  
  
/\*------------------------------------------------容器大小操作-----------------------------------------------  
> < == : 比和string字符串的比较一样的，只有容器的元素定义了关系运算符，才可以比较容器之间的大小  
size() 返回容器中元素的个数  
empty() 容器是否为空  
max\_size() 该类型容器最大容纳的元素的个数  
 //forward\_list不支持size()操作，原因肯定是因为要去手写单向链表一致  
-----------------------------------------------------------------------------------------------------------\*/  
  
/\*------------------------------------------------顺序容器操作-----------------------------------------------  
  
//forward\_list 有自己专门的insert和emplace,不支持push\_back和emplace\_back, vector/string不支持push\_front  
 以及emplace\_front,虽然有些容器支持,但是对于insert(begin,...)没有限制  
  
c.push\_back(t) 在c的尾部创建t或者args创建的元素，返回void  
c.emplace\_back(args)   
  
c.push\_front(t) 在c的开头创建t或者args创建的元素，返回void  
c.emplace\_front(t)  
  
c.insert(p, n, t) 在迭代器p之前插入n个值为t的元素，返回新添加的第一个元素的迭代器，若n为0，则返回p  
c.insert(p, b, e) 将迭代器b, e指向的元素之前插入到p所指向新添加元素之前，返回新添加第一个元素的迭代器  
//insert 是按顺序向后插入的，比如{0, 1, 2}的begin插入{3，4，5}是{3, 4, 5, 0, 1, 2}  
//向一个vector/string/deque中插入元素会使所有指向容器的迭代器、引用、指针失效。  
//当我们使用一个对象来初始化容器时，或者将元素插入容器中，实际上放入容器的是其对象值的一个拷贝，两者并无关联  
  
//vector和string不支持push\_front()操作，而list、forward\_list、deque支持push\_front()操作;push\_front是一种倒序的结果  
//emplace\_back emplace emplace\_front 是在内存空间里直接构造对象，而不是拷贝。 emplace\_back(args)  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
访问元素，除了forward\_list 每一个容器都提供了c.front(),以及c.back()成员，用以返回容器的首尾元素的引用  
at和下标操作只适合string vector deque array 如果越界，则会爆出out\_of \_range错误  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
以下操作不适合array，因为这些操作会改变容器的大小，不适于array  
c.pop\_back() //删除c中的尾元素,若c为空则函数行为未定义  
c.pop\_front() //删除队头元素，若c为空，则函数未定义  
  
c.erase(p) //删除迭代器所指的元素，返回一个指向被删除元素之后的元素的迭代器，若p指向最后一个元素，则返回  
 //尾后迭代器，  
c.erase(p, e) //删除迭代器b和e所指定范围内的元素，返回一个指向最后一个被删除元素之后的元素的迭代器，若e本身也是  
 //最后一个元素，那么也返回尾后迭代器,  
  
c.clear() //删除所有的元素，  
  
//PS：删除deque除首尾之外的所有元素都会值得所有迭代器，引用或者指针失效，指向vector以及string的删除点之后位置的  
迭代器、引用、指针失效。  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
forward\_list特有的操作：forward\_list<int >lst  
lst.before\_begin()  
lst.cbefore\_begin() 返回首前迭代器  
  
lst.insert\_after(p, t) 在迭代器p之后插入为值t的对象  
lst.insert\_after(p, n, t) n个值为t的对象 返回最后一个插入的元素的迭代器，  
lst.insert\_after(p, b, e) n个值为t的对象 返回最后一个插入的元素的迭代器，  
lst.insert\_after(p, il) n个值为t的对象 返回最后一个插入的元素的迭代器，  
emplace\_after(p, args) 在p之后创建元素  
  
lst.erase\_after(p) 删除p之后的元素,返回一个被删除元素之后的元素的迭代器  
lst.erase\_after(b, e) 删除[p, e)的元素,返回最后一个被删除元素之后的元素的迭代器  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
//resize() 不适合array  
c.resize() 调整c的大小为c个元素，若不足，则补足  
c.resize(n, t) 略  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
c.capacity() 返回在不扩张内存的情况下可以容纳多少元素。  
c.reserve() 分配至少能容纳n个元素的内存空间  
c.size() 容器中有多少元素  
-----------------------------------------------------------------------------------------------------------\*/  
  
/\*------------------------------------------------额外的string构造方法-----------------------------------------------  
string s(cp, n) s是cp指向数组中前n个字符的拷贝，此数组至少应该包含n个字符  
string s(s2, pos2) s是s2从下标pos2开始的拷贝，  
string s(s2, pos2, len2) s是s2从下标pos2开始，长度为len2的拷贝  
  
s.substr(pos = 0, n = s.size() - pos) //返回一个string 包含从pos开始的n个字符的拷贝  
string s("hello world");  
string s2 = s.substr(0, 5); //s2 = hello  
string s3 = s.substr(6); //s3 = world  
string s4 = s.substr(6, 11); //s4 = world  
string s5 = s.substr(12); //抛出一个out\_of\_range  
//除了普通的insert()和erase()操作，string还有以下的重载版本，都是在pos之前插入或者删除，字符串下标是从0开始的  
//string还提供了两个额外的成员append(),replace()，append()是在string末尾加入的一种形式，而replace是调用erase()和  
//insert()的简写形式  
  
s.insert(pos, args) //在pos之前插入args指定的字符，pos是个下标或者是一个迭代器，接受下标的版本返回一个指向s的  
 引用,接受迭代器的版本返回指向第一个插入字符的迭代器  
s.erase(pos, len) //删除从pos处开始的len个字符，如果len省略则删除pos开始的所有字符，返回一个指向s的引用  
s.replace(range, args) //删除range内的元素，替换为args的元素，args可以是一个下标加一个长度，或者一对迭代器，返回  
 返回一个指向s的引用  
s.append(args) //将args加入到s的尾部，返回一个指向s的引用  
  
s.assign(args) //将s中的字符替换为args字符，返回一个引用  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
string的搜索函数：  
(1) s.find(args) //查找s中第一次出现的位置,并返回子串在主串中第一个字符的下标  
(2) s.rfind(args) //查找最后一个...  
(3) s.find\_first\_of(args) //在s中查找args中任何一个字符的第一次出现的位置，  
(4) s.find\_last\_of(args) //在s中查找args中任何一个字符最后一次出现的位置。  
(5) s.find\_first\_not\_of(args) //在s中查找第一个不在args中的字符  
(6) s.find\_last\_not\_of(args) //查找最后一个不在args中的字符  
  
(1) args必须是以下的形式：  
（c, pos） 从s中位置pos开始查找字符c，pos默认0。  
（s2, pos） 从s中位置pos开始查找字符串s2，pos默认0。  
（cp, pos） 从s中位置pos开始查找cp指向的以空字符结尾的C风格字符串，pos默认为0  
（cp, pos, n） 从s中位置pos开始查找指针cp指向的数组的前n个字符。pos和n无默认值  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
compare函数：这与C标准库，提供的strcmp很相似；  
（1）s.compare(s2) //比较s和s2  
（2）s.compare(pos1, n1, s2) //将pos1开始的长度为n1的字符串和s2进行比较  
（3）s.compare(pos1, n1, s2, pos2, n2)   
（4) s.compare(cp) //比较s与cp指向的以空格结尾的字符数组  
-----------------------------------------------------------------------------------------------------------\*/  
  
/\*------------------------------------------------容器适配器-------------------------------------------------  
//除了string vector list forword\_list deque array等顺序容器之外，还定义了queue stack priority\_queue等适配器  
（1）stack //栈，先入后出结构，  
 （1）pop()  
 （2）top()  
 （3）push()  
 （4）empty()  
（2）queue //队列，不是双端队列，但是是基于deque实现的，priority\_queue是基于vector实现的  
 （1）q.pop()  
 （2）q.top()  
 （3）q.front()  
 （4）q.back()  
 （5）q.top()  
-----------------------------------------------------------------------------------------------------------\*/  
/\*----------------------------------------------关联容器--------------------------------------------------  
//关联容器支持高效的关键字查找和访问，主要有map和set两种。再冠以multi以及unorder就一共有八种关联容器  
关联容器：有三个比较关键的类型key\_value mapped\_type value\_type //  
有两个数据成员first、second 两个，first 是关键字，second是值，进行下标运算的时候，若元素不在容器内，那么容器创建新键值对，  
并且值为0；需要手动加加；  
而set的find()函数，找到了返回该元素迭代器，找不到返回尾后迭代器；  
//关联容器的关键值一定要有比较运算符，因为插入容器的元素默认是按字典序排序的；  
//关联容器迭代器：map的关键值是const 类型 而set的迭代器就是const类型，且迭代器支持++操作  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
pair类型:  
 (1)pair<t1, t2>p()  
 (2)pair<t1, t2>p = {v1, v2};  
 (3)make\_pair(v1, v2)  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
(1)插入操作：  
 c.insert(v) 对于map set 当v不在容器中才执行插入操作  
 c.insert(b, e)   
 c.insert(il)   
//插入单个元素返回一个pair类型，pair的second成员是一个bool值，返回是否插入成功则返回true，first无论何时都指向value例如  
 std::map<std::string, int>word\_count;  
 std::string tmp;  
 while (std::cin >> tmp)  
 {  
 auto ret = word\_count.insert({ tmp, 1 });  
 if (!ret.second)  
 //++word\_count[tmp];  
 ++ret.first->second;  
 }  
 auto begin = word\_count.begin();  
 while (begin != word\_count.end())  
 {  
 std::cout << begin->first << ":" << begin->second << std::endl;  
 ++begin;  
 }  
(2)而向multimap以及multiset中插入元素，犯规一个指向新值的迭代器，没有second，因为总会新插入一个迭代器  
(3)map的下标操作：  
 （1）c[k] //下标操作会返回一个mapped\_type 而解引用操作会返回一个value\_type  
 （2）c.at(k)   
(4)find()操作  
 例：if(word\_count.find("foobar") == word\_count.end())  
 但是如果是一个multimap或者multiset的时候，由于具有相同关键字的连续存储，则需要先使用count获取数量，在使用  
 find()函数获取第一个元素，然后在使用循环挨个访问；  
 可以使用面向迭代器的解决方法：  
 lower\_bound() upper\_bound()来解决，前者指向第一个匹配的关键字，后者最后匹配的关键字，也可以只用equal\_range来实现  
 若存在返回一个pair，first指向第一个，second指向最后一个。若不存在，则都返回指向可以插入的位置  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
/\*------------------------------------------------额外的string构造方法--------------------------------------  
给出string vector array list forward\_list deque queue stack map multimap set multiset tuple bitset所支持的操作以及范例  
-----------------------------------------------------------------------------------------------------------\*/  
#include "myString.h"  
#include "myVector.h"  
#include "myList.h"  
#include "myForward\_list.h"  
#include "myDeque.h"  
#include "myArray.h"  
#include "myMap.h"  
#include "mySet.h"  
#include "myAlgorithm.h"  
#include "myStackAndQueue.h"  
#include "tupleAndBitset.h"  
  
int main()  
{  
 myStringTest();  
 myVectorTest();  
 myListTest();  
 myForward\_listTest();  
 myDequeTest();  
 myArrayTest();  
 myMapTest();  
 myMapTest();  
 mySetTest();  
 myStackTest();  
 myQueueTest();  
 tupleAndBitsetTest();  
 myAlgorithmTest();  
 system("pause");  
 return 0;  
}