一．Set容器

之间已经讲过，set容器与map容器都是关联型容器，set容器不像map容器那样，有键值和对应的实值，set容器的元素的键值就是实值。因此set容器是不允许有相同的键值的。

由于set容器的键值就是对应的实值，且关联型容器在数据插入时已经将键值和实值绑定了，因此，不能随意更改set的元素值，修改set的元素值相当于修改键值，这会破坏set的排序结构。因此，无法通过set的迭代器修改set元素值，即set的iterator是const\_iterator

二．Multiset容器

Multiset特性和用法和set完全一样，差别在于multiset允许键值重复。

三．Set容器常用操作

1. set构造函数

set<T> st;//set默认构造函数：

mulitset<T> mst; //multiset默认构造函数:

set(const set &st);//拷贝构造函数

1. set赋值操作

set& operator=(const set &st);//重载等号操作符

swap(st);//交换两个集合容器

1. set大小操作

set& operator=(const set &st);//重载等号操作符

swap(st);//交换两个集合容器

1. set插入和删除操作

insert(elem);//在容器中插入元素。

clear();//清除所有元素

erase(pos);//删除pos迭代器所指的元素，返回下一个元素的迭代器。

erase(beg, end);//删除区间[beg,end)的所有元素 ，返回下一个元素的迭代器。

erase(elem);//删除容器中值为elem的元素。

插入返回Pair，bool意味着是否插入成功

Pair<set<int>::iterator, bool> ret = s1.insert(10);

对组(pair)将一对值组合成一个值，这一对值可以具有不同的数据类型，两个值可以分别用pair的两个公有属性first和second访问。

Pair.first

Pair.second

如何构造pair呢？两种方法：

第一种：

Pair<string, int> p(string(“Tom”), 100);

第二种：

Pair<string, int> p2 = make\_pair(“Jerry”, 200);

1. set查找操作

find(key);//查找键key是否存在,若存在，返回该键的元素的迭代器；若不存在，返回set.end();

count(key);//查找键key的元素个数

lower\_bound(keyElem);//返回第一个key>=keyElem元素的迭代器。

upper\_bound(keyElem);//返回第一个key>keyElem元素的迭代器。

equal\_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器。

1. set的排序规则

set容器的元素在插入后就已经和键值绑定了，顺序是无法修改的，因此要想修改set的排序规则，就必须在元素插入前修改。即set容器初始化时可以设置排序规则。

set容器在初始化set时，第二个模板参数可以设置排序规则，如下所示：

set<int, myCompare> s1;

myCompare是一个仿函数，该仿函数也是一个类。为什么不能是函数呢？

因为set是类模板，类模板的参数只能放数据类型，而函数不是数据类型，因此只能用仿函数（类），

Set容器排序默认升序，想要降序，在插入之前指定排序规则：

用仿函数myCompare重载()运算符，指定排序规则。

Class myCompare  
{  
public:

//重载()运算符，要加const

Bool operator() (int v1, int v2) const

{  
 return v1 > v2;

}

}

Set<int, myCompare> s1;

自定义排序规则

当Set中插入自定义数据类型，需自定义排序规则

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<set>

using namespace std;

class Person

{

public:

Person(string name, int age)

{

this->m\_Name = name;

this->m\_Age = age;

}

string m\_Name;

int m\_Age;

};

class myCompare

{

public:

bool operator() (const Person& p1,const Person& p2) const

{

if (p1.m\_Age > p2.m\_Age)

{

return true;

}

return false;

}

};

void test()

{

set<Person, myCompare> s1;

Person p1("w", 10);

Person p2("a", 20);

Person p3("n", 30);

Person p4("g", 40);

Person p5("x", 50);

Person p6("u", 60);

s1.insert(p1);

s1.insert(p2);

s1.insert(p3);

s1.insert(p4);

s1.insert(p5);

s1.insert(p6);

for (set<Person, myCompare>::iterator it = s1.begin(); it != s1.end(); ++it)

{

cout << "姓名： " << (\*it).m\_Name << " 年龄：" << (\*it).m\_Age << endl;

}

}

int main()

{

test();

return 0;

}