set是STL中一种标准关联容器（vector,list,string,deque都是序列容器，而set，multiset，map，multimap是标准关联容器），它底层使用平衡的搜索树——红黑树实现，插入删除操作时仅仅需要指针操作节点即可完成，不涉及到内存移动和拷贝，所以效率比较高。set，顾名思义是“集合”的意思，在set中元素都是唯一的，而且默认情况下会对元素自动进行升序排列，支持集合的交(set\_intersection),差(set\_difference) 并(set\_union)，对称差(set\_symmetric\_difference) 等一些集合上的操作，如果需要集合中的元素允许重复那么可以使用multiset

STL中算法包含在<Algorithm>头文件中,集合的算法也包含在该头文件中.

集合的并:set\_union

集合的交:set\_intersection

集合的差:set\_differnce

#include<set>

#include<algorithm>

#include<iterator>

#include<iostream>

using namespace std;

int main()

{

set<int>eg1;

//插入

eg1.insert(1);

eg1.insert(100);

eg1.insert(5);

eg1.insert(1); //元素1因为已经存在所以set中不会再次插入1

eg1.insert(10);

eg1.insert(9);

//遍历set，可以发现元素是有序的

set<int>::iterator set\_iter=eg1.begin();

cout<<"Set named eg1:"<<endl;

for(;set\_iter!=eg1.end();set\_iter++) cout<<\*set\_iter<<" ";

cout<<endl;

//使用size()函数可以获得当前元素个数

cout<<"Now there are "<<eg1.size()<<" elements in the set eg1"<<endl;

if(eg1.find(200)==eg1.end()) //find()函数可以查找元素是否存在

cout<<"200 isn't in the set eg1"<<endl;

set<int>eg2;

for(int i=6;i<15;i++)

eg2.insert(i);

cout<<"Set named eg2:"<<endl;

for(set\_iter=eg2.begin();set\_iter!=eg2.end();set\_iter++)

cout<<\*set\_iter<<" ";

cout<<endl;

//获得两个set的并

set<int>eg3;

cout<<"Union:";

set\_union(eg1.begin(),eg1.end(),eg2.begin(),eg2.end(),insert\_iterator<set<int> >(eg3,eg3.begin())); //注意第五个参数的形式

copy(eg3.begin(),eg3.end(),ostream\_iterator<int>(cout," "));

cout<<endl;

//获得两个set的交，注意进行集合操作之前接收结果的set要调用clear()函数清空一下

eg3.clear();

set\_intersection(eg1.begin(),eg1.end(),eg2.begin(),eg2.end(),insert\_iterator<set<int> >(eg3,eg3.begin()));

cout<<"Intersection:";

copy(eg3.begin(),eg3.end(),ostream\_iterator<int>(cout," "));

cout<<endl;

//获得两个set的差

eg3.clear();

set\_difference(eg1.begin(),eg1.end(),eg2.begin(),eg2.end(),insert\_iterator<set<int> >(eg3,eg3.begin()));

cout<<"Difference:";

copy(eg3.begin(),eg3.end(),ostream\_iterator<int>(cout," "));

cout<<endl;

//获得两个set的对称差，也就是假设两个集合分别为A和B那么对称差为AUB-A∩B

eg3.clear();

set\_symmetric\_difference(eg1.begin(),eg1.end(),eg2.begin(),eg2.end(),insert\_iterator<set<int> >(eg3,eg3.begin()));

copy(eg3.begin(),eg3.end(),ostream\_iterator<int>(cout," "));

cout<<endl;

return 0;

}

set会对元素进行排序，那么问题也就出现了排序的规则是怎样的呢？上面的示例代码我们发现对int型的元素可以自动判断大小顺序，但是对char\*就不会自动用strcmp进行判断了，更别说是用户自定义的类型了，事实上set的标准形式是set<Key, Compare, Alloc>，

|  |  |  |
| --- | --- | --- |
| **参数** | **描述** | **默认值** |
| Key | 集合的关键字和值的**类型** |  |
| Compare | 关键字比较函数，它的参数类型key参数指定的类型，如果第一个参数小于第二个参数则返回true，否则返回false | less<Key> |
| Alloc | set的分配器，用于内部内存管理 | alloc |

下面给出一个关键字类型为char\*的示例代码

#include<iostream>  
#include<iterator>  
#include<set>  
using namespace std;  
struct ltstr  
{  
bool operator() (const char\* s1, const char\* s2) const  
{  
   return strcmp(s1, s2) < 0;  
}  
};

int main()  
{  
const int N = 6;  
const char\* a[N] = {"isomer", "ephemeral", "prosaic",   
   "nugatory", "artichoke", "serif"};  
const char\* b[N] = {"flat", "this", "artichoke",  
   "frigate", "prosaic", "isomer"};

set<const char\*,ltstr> A(a, a + N);  
set<const char\*,ltstr> B(b, b + N);  
set<const char\*,ltstr> C;

cout << "Set A: ";  
//copy(A.begin(), A.end(), ostream\_iterator<const char\*>(cout, " "));  
set<const char\*,ltstr>::iterator itr;  
for(itr=A.begin();itr!=A.end();itr++) cout<<\*itr<<" ";  
cout << endl;  
cout << "Set B: ";  
copy(B.begin(), B.end(), ostream\_iterator<const char\*>(cout, " "));     
cout << endl;

cout << "Union: ";  
set\_union(A.begin(), A.end(), B.begin(), B.end(),  
    ostream\_iterator<const char\*>(cout, " "),  
    ltstr());     
cout << endl;

cout << "Intersection: ";  
set\_intersection(A.begin(), A.end(), B.begin(),B.end(),ostream\_iterator<const char\*>(cout," "),ltstr());  
cout<<endl;  
set\_difference(A.begin(), A.end(), B.begin(), B.end(),inserter(C, C.begin()),ltstr());  
cout << "Set C (difference of A and B): ";  
copy(C.begin(), C.end(), ostream\_iterator<const char\*>(cout, " "));  
cout <<endl;  
return 0;  
}

其中的ltstr也可以这样定义  
class ltstr  
{  
        public:  
        bool operator() (const char\* s1,const char\*s2)const  
        {  
                return strcmp(s1,s2)<0;  
        }  
};

更加通用的应用方式那就是数据类型也是由用户自定义的类来替代，比较的函数自定义，甚至可以加上二级比较，比如首先按照总分数排序，对于分数相同的按照id排序，下面是示例代码

#include<set>  
#include<iostream>  
using namespace std;  
struct  
{  
                int id;  
                int score;  
                string name;  
};  
struct compare  
{  
        bool operator()(const Entity& e1,const Entity& e2)const   {  
                if(e1.score<e2.score) return true;  
                else  
                        if(e1.score==e2.score)  
                                if(e1.id<e2.id) return true;

                return false;  
        }  
};

int main()  
{  
        set<Entity,compare>s\_test;  
        Entity a,b,c;  
        a.id=123;a.score=90;a.name="bill";  
        b.id=121;b.score=85;b.name="mary";  
        c.id=130;c.score=85;c.name="jerry";  
        s\_test.insert(a);s\_test.insert(b);s\_test.insert(c);  
        set<Entity,compare>::iterator itr;  
        cout<<"Score List(ordered by score):\n";  
        for(itr=s\_test.begin();itr!=s\_test.end();itr++)  
                cout<<itr->id<<"---"<<itr->name<<"---"<<itr->score<<endl;  
        return 0;  
}