set：关联容器；按照数学概念理解，set就是集合的意思，集合中的元素最多只出现一次，并且set中得元素已经有序，按照从小到大的顺序排列好了。于map类似，内部采用红黑树实现，查找、插入和删除的时间复杂度为O(logN)。内存不连续，每个节点的内存都是单独申请。

特点：

1、set和map的插入和删除效率比其他容器高；
2、每次insert后，以前保存的迭代器不会失效；
3、当数据元素增多时，查找和插入速度变化不大；
4、set中的元素不允许重复，无法修改；

头文件：

```
#include <set>
```

基本接口：

```
set.size();                          // 返回set中元素的个数
set.max_size();                      // 返回set中能够存储的元素的最大数量
set.empty();                         // 判断容器是否为空
set.count(const value_type& val);    // 计算容器中val元素出现的次数

set.begin();     // 返回一个指向第一个元素的迭代器
set.end();       // 返回一个指向最后一个元素的下一个位置的迭代器，不指向任何有效元素
set.find(const value_type& val);     // 在容器中查找元素，找到时返回指向该元素的迭代器，找不到时返回的迭代器指向set.end();

//删除容器中元素，仅指向被删除元素的迭代器失效，其他迭代器全部有效；所以可以使用erase(it++)进行安全删除操作；
iterator  erase (const_iterator position);                  // 返回值为被删除元素下一个位置的迭代器
iterator  erase (const_iterator first, const_iterator last);// ditto
size_type erase (const value_type& val);                    // 返回值为删除元素的个数
```

```
//use iterator to delete all element safely
for (set<int>::iterator iter = set.begin(); iter != set.end();){
 set.erase(iter++);
}
```

//删除容器中得所有元素，释放容器中对应元素占用的内存(set中存放的数据为指针类型时，并不会释放指针指向的空间，只有当存放的数据为类或者结构体类型时，clear的时候才会释放内存)
set.clear();

// 析构函数，删除所有元素并释放内存(set中存放的数据为指针类型时，并不会释放指针指向的空间，只有当存放的数据为类或者结构体类型时，析构的时候才会释放内存)
~set();

```
#include <iostream>
#include <set>

using namespace std;

typedef struct Test
{
public:
 Test(int key){
  m_key = key;
  }
  ~Test(){
  cout << "destruct, key: " << m_key << endl;
  }

public:
  int m_key;
} Test;

struct setCmp{
  bool operator()(const Test &a, const Test &b) {
  return a.m_key > b.m_key;
  }
};

struct setCmp2{
  bool operator()(const Test *a, const Test *b) {
  return a->m_key > b->m_key;
  }
};

void main(void)
```

```
{
 {
 cout << "enter cs work space..." << endl;
 set<Test, setCmp> cs;
 for (int i = 20; i < 22; i++){
 Test t(i);
 cs.insert(t);
 }
 cout << "leave cs work space..." << endl;
 }

 {
 cout << "enter css work space..." << endl;
 set<Test*, setCmp2> css;
 for (int i = 30; i < 32; i++){
 Test *t = new Test(i);
 css.insert(t);
 // test whether ~set() will delete t
 }
 cout << "leave css work space..." << endl;
 }

 {
 cout << "enter csss work space..." << endl;
 set<Test*, setCmp2> csss;
 for (int i = 40; i < 42; i++){
 Test *t = new Test(i);
 csss.insert(t);
 // test whether clear() will delete t
 }
 csss.clear();
 cout << "csss size = " << csss.size() << endl;
 cout << "leave css work space..." << endl;
 }

 set<int> s;
 for (int i = 0; i < 10; i++){
 s.insert(i);
 }
 for (set<int>::iterator it = s.begin(); it != s.end();){
```

```
73    cout << " " << *it << " ";
74    s.erase(it++);
75    }
76    cout << endl;
77    cout << "size: " << s.size() << endl;
78
79    cout << "exit main fun" << endl;
80
81    system("pause");
82  }
```

```
enter cs work space...
destruct, key: 20
destruct, key: 21
leave cs work space...
destruct, key: 20
destruct, key: 21
enter css work space...
destruct, key: 30
destruct, key: 31
leave css work space...
enter csss work space...
csss size = 0
leave css work space...
 0  1  2  3  4  5  6  7  8  9
size: 0
exit main fun
```

由上面的测试程序可知：

~set()析构函数和set.clear()函数：

1、当成员类型为类时，会调用类的析构函数，释放内存；

2、当成员类型为类指针时，只会从set中删除元素，不会调用类的析构函数，不释放内存；