

知识点1【list存放自定义数据 如果删除某个节点 必须重载 ==运算符】

知识点2【list容器 对自定义数据排序 重载<运算符】

知识点3【list 对自定义数据排序 指定排序规则】

知识点4【vector 对自定义数据 排序指定排序规则】 1-2

1、方法一 普通函数 实现排序规则

2、方法二 仿函数 指定排序规则

知识点5【仿函数 重载函数调用运算符()】

知识点6【lambda表达式】

知识点7【set容器】 2-1

1、set容器的特点：

知识点8【set容器的API】

1、set容器的插入删除

2、set容器的查找

3、set容器的lower_bound和upper_bound

案例2-2

4、对组

5、更改set容器的默认排序规则（推荐使用仿函数）

6、set容器存放自定义数据的时候 重载<运算符

运行结果：

7、set容器存放自定义数据的时候 指定排序规则。

知识点9【multiset允许插入重复的键值】

1、set容器不允许插入相同的key值

知识点1 【list存放自定义数据 如果删除某个节点 必须重载 ==运算符】

```
1 #include <iostream>
2 #include <list>
3 #include<algorithm>
4 #include<string>
5 using namespace std;
6 class Person
7 {
8 public:
9     string name;
10    int age;
11 public:
12    Person(string name,int age)
13    {
14        this->name = name;
15        this->age = age;
16    }
17
18    //成员函数重载 ==运算符
19    bool operator==(const Person &ob)
20    {
21        if(this->name == ob.name && this->age == ob.age)
22            return true;
23        return false;
24    }
25 };
26 void printListPerson(list<Person> &L)
27 {
28     cout<<"-----"<<endl;
29     for(list<Person>::iterator it=L.begin(); it!= L.end();it++)
30     {
31         cout<<(*it).name<<" "<<(*it).age<<endl;
32     }
33 }
34 void test01()
```

```
35 {
36     //存放自定义数据
37     list<Person> L;
38     L.push_back(Person("德玛西亚",48));
39     L.push_back(Person("提莫", 28));
40     L.push_back(Person("狗头", 18));
41     L.push_back(Person("牛头", 19));
42
43     printListPerson(L);
44
45     //删除狗头
46     Person tmp("狗头", 18);
47     //重载==运算符
48     L.remove(tmp);
49     printListPerson(L);
50 }
51 int main(int argc, char *argv[])
52 {
53     test01();
54     return 0;
55 }
56
```

运行结果：

德玛西亚 48

提莫 28

狗头 18

牛头 19

德玛西亚 48

提莫 28

牛头 19

知识点2 【list容器 对自定义数据排序 重载<运算符】

```
1 //成员函数重载 <运算符
2 bool operator<(const Person &ob)
3 {
4     if(this->age < ob.age)
```

```
5  return true;
6
7  return false;
8  }
```

```
1  void test02()
2  {
3      //存放自定义数据
4      list<Person> L;
5      L.push_back(Person("德玛西亚",48));
6      L.push_back(Person("提莫", 28));
7      L.push_back(Person("狗头", 18));
8      L.push_back(Person("牛头", 19));
9      printListPerson(L);
10
11     //对于自定义数据 我们可以重载<运算符
12     L.sort();
13
14     printListPerson(L);
15 }
```

运行结果：

德玛西亚 48

提莫 28

狗头 18

牛头 19

狗头 18

牛头 19

提莫 28

德玛西亚 48

知识点3 【list 对自定义数据排序 指定排序规则】

```
1 bool myComparePerson(const Person &ob1, const Person &ob2)
2 {
3     return ob1.age < ob2.age;
4 }
```

```
5 void test03()
6 {
7     //存放自定义数据
8     list<Person> L;
9     L.push_back(Person("德玛西亚",48));
10    L.push_back(Person("提莫", 28));
11    L.push_back(Person("狗头", 18));
12    L.push_back(Person("牛头", 19));
13    printListPerson(L);
14
15    //对于自定义数据 我们可以重载<运算符
16    L.sort(myComparePerson);
17
18    printListPerson(L);
19 }
```

运行结果：

德玛西亚 48

提莫 28

狗头 18

牛头 19

狗头 18

牛头 19

提莫 28

德玛西亚 48

知识点4 【vector 对自定义数据 排序指定排序规则】 1-2

1、方法一 普通函数 实现排序规则

```
1 bool myComparePerson(const Person &ob1, const Person &ob2)
```



```
2 {  
3     return ob1.age > ob2.age;  
4 }
```

```
1 void test04()  
2 {  
3     vector<Person> v;  
4     v.push_back(Person("德玛西亚",48));  
5     v.push_back(Person("提莫", 28));  
6     v.push_back(Person("狗头", 18));  
7     v.push_back(Person("牛头", 19));  
8     PrintVectorPerson(v);  
9  
10    //默认比较方式从小-->大  
11    //vector存放自定义数据 可指定排序规则 (普通函数)  
12    sort(v.begin(),v.end(),myComparePerson);  
13  
14    PrintVectorPerson(v);  
15 }
```

运行结果:

德玛西亚 48

提莫 28

狗头 18

牛头 19

德玛西亚 48

提莫 28

牛头 19

狗头 18

2、方法二 仿函数 指定排序规则

```
1 //仿函数指定排序规则
2 class myComparePerson2
3 {
4 public:
5     bool operator()(Person &ob1, Person &ob2)
6     {
7         return ob1.age < ob2.age;
8     }
9 };
```

```
1 void test04()
2 {
3     vector<Person> v;
4     v.push_back(Person("德玛西亚",48));
5     v.push_back(Person("提莫", 28));
6     v.push_back(Person("狗头", 18));
7     v.push_back(Person("牛头", 19));
8     PrintVectorPerson(v);
9
10    //默认比较方式从小-->大
11    //vector存放自定义数据 可指定排序规则 (普通函数)
12    //sort(v.begin(),v.end(), myComparePerson);
13
14    //vector存放自定义数据 可指定排序规则 (仿函数)
15    sort(v.begin(),v.end(), myComparePerson2() );
16
17    PrintVectorPerson(v);
18 }
```

运行结果：

德玛西亚	48
提莫	28
狗头	18
牛头	19

狗头	18
牛头	19
提莫	28
德玛西亚	48

知识点5 【仿函数 重载函数调用运算符()】

```
1 class MyAdd{  
2 public:
```

```

3 //本质就是成员函数 函数名叫operator()
4 int operator()(int a, int b)
5 {
6     cout<<"调用了operator() int int"<<endl;
7     return a+b;
8 }
9 int operator()(int a, int b, int c)
10 {
11     cout<<"调用了operator()int int int "<<endl;
12     return a+b+c;
13 }
14
15 };
16 void test05()
17 {
18     MyAdd ob1;
19     cout<<ob1.operator()(10,20)<<endl;
20     //编译器优化
21     //严格意义: ob1是对象 和()结合 调用operator()成员函数
22     //ob1(100,200) ob1本质是对象不是函数名 只是形式像函数调用
23     //叫做仿函数
24     cout<<ob1(100,200)<<endl;
25     cout<<ob1(100,200,300)<<endl;
26
27     cout<<MyAdd()(1000,2000)<<endl;
28
29 }

```

运行结果：


```
调用了operator() int int
30
调用了operator() int int
300
调用了operator() int int int
600
调用了operator() int int
3000
```

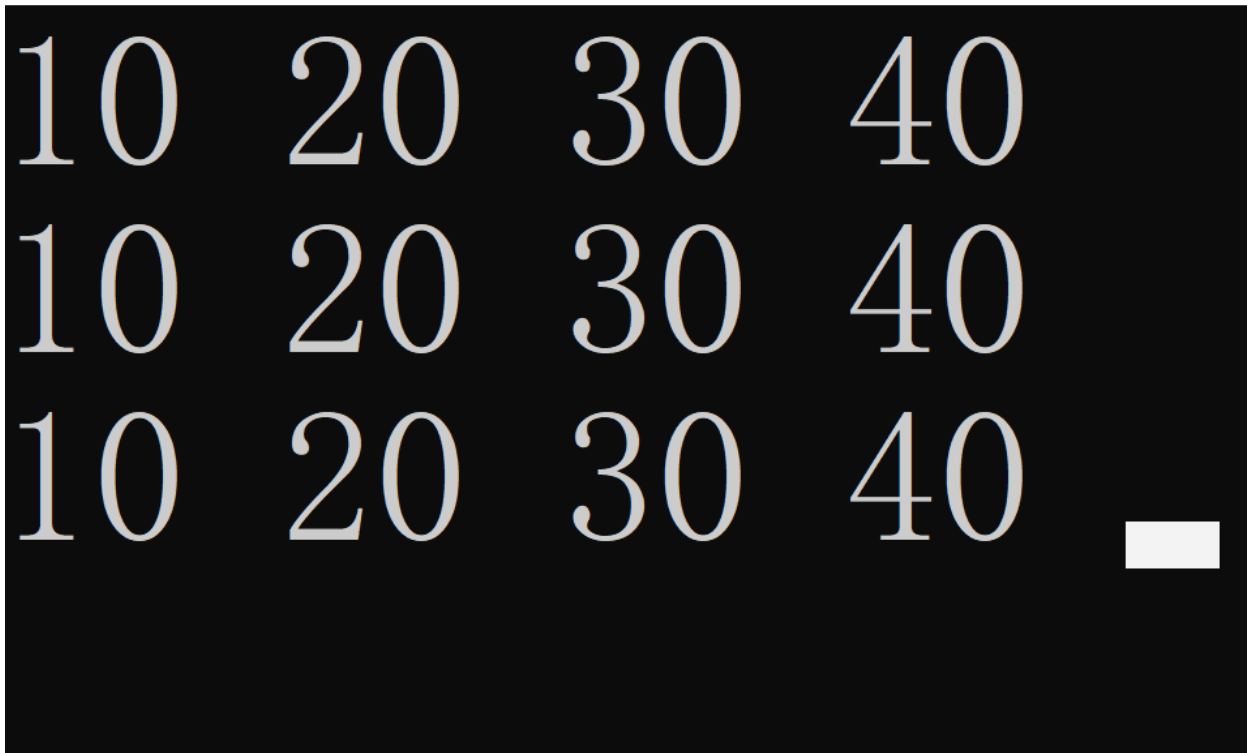
知识点6 【lambda表达式】

```
1 void myPrintVectorInt(vector<int> &v)
2 {
3     for(vector<int>::iterator it=v.begin();it!=v.end();it++)
4     {
5         cout<<*it<<" ";
6     }
7     cout<<endl;
8 }
9 //指定打印方式
10 void myPrint01(int val)
11 {
12     cout<<val<<" ";
13 }
14 void test06()
15 {
16     vector<int> v;
17     v.push_back(10);
18     v.push_back(20);
19     v.push_back(30);
20     v.push_back(40);
21
22     //方法1:访问v容器 普通函数
23     myPrintVectorInt(v);
```

```
24
25 //方法2:访问v容器 系统算法for_each
26 for_each(v.begin(),v.end(), myPrint01);
27 cout<<endl;
28
29 //方法3:访问v容器 lambda表达式
30 //[]表示函数名 ()参数列表 {}函数体
31 for_each(v.begin(),v.end(), [](int val){
32     cout<<val<<" ";
33 } );
34 }
```

运行结果:

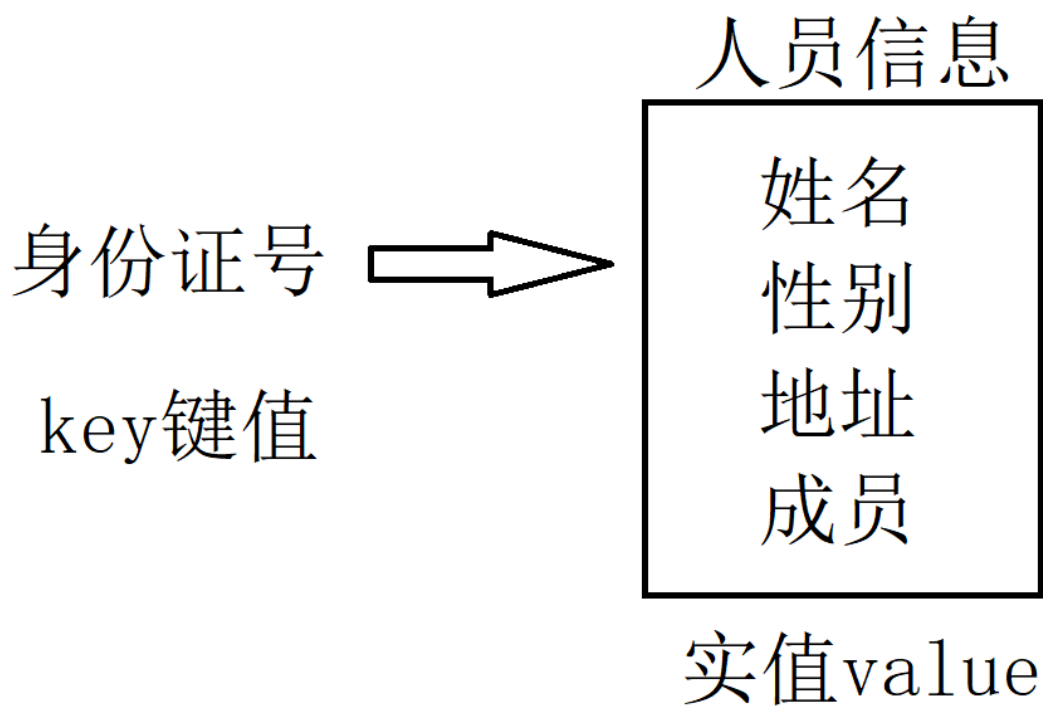
 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe



知识点7 【set容器】 2-1

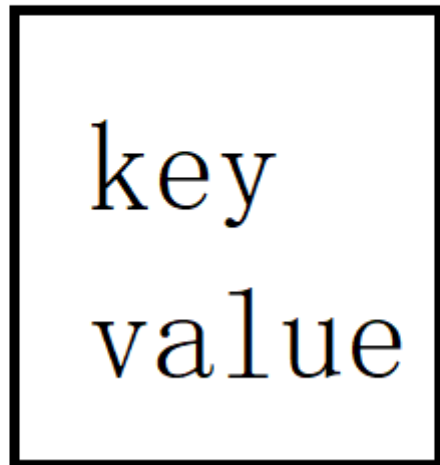
1、set容器的特点:

所有元素都会根据元素的键值自动被排序。



set的元素即是键值又是实值。

set容器的元素



set容器的键值 不允许相同。

set容器提供的是只读迭代器（不允许用户修改元素的内容）

知识点8 【set容器的API】

1、set容器的插入删除

```
1 /*
2 3.7.2.1 set构造函数
3 set<T> st; //set默认构造函数:
4 multiset<T> mst; //multiset默认构造函数:
```



```

5  set(const set &st); //拷贝构造函数
6  3.7.2.2 set赋值操作
7  set& operator=(const set &st); //重载等号操作符
8  swap(st); //交换两个集合容器
9  3.7.2.3 set大小操作
10 size(); //返回容器中元素的数目
11 empty(); //判断容器是否为空
12 3.7.2.4 set插入和删除操作
13 insert(elem); //在容器中插入元素。
14 clear(); //清除所有元素
15 erase(pos); //删除pos迭代器所指的元素，返回下一个元素的迭代器。
16 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。
17 erase(elem); //删除容器中值为elem的元素
18 */
19 void test01()
20 {
21     set<int> s;
22     //set容器自动根据键值排序
23     s.insert(30);
24     s.insert(10);
25     s.insert(20);
26     s.insert(50);
27     s.insert(40);
28
29     for_each(s.begin(), s.end(), [](int val){
30         cout<<val<<" ";
31     }); //10 20 30 40 50
32     cout<<endl;
33
34     //set容器提供的是只读迭代器const_iterator
35     //用户不可以修改set容器的元素
36     set<int>::const_iterator it=s.begin();
37     //*it = 100; //err
38     cout<<"size = "<<s.size()<<endl;
39
40     //删除起始位置的元素
41     s.erase(s.begin());
42     for_each(s.begin(), s.end(), [](int val){
43         cout<<val<<" ";
44     }); //20 30 40 50

```

```

45     cout<<endl;
46
47     //根据元素删除
48     s.erase(40);
49     for_each(s.begin(),s.end(),[](int val){
50         cout<<val<<" ";
51     }); //20 30 50
52     cout<<endl;
53 }
54

```

2、set容器的查找

```

1  /*
2  3.7.2.5 set查找操作
3  find(key); //查找键key是否存在,若存在, 返回该键的元素的迭代器;
4  若不存在, 返回set.end();
5  count(key); //查找键key的元素个数
6  */
7  void test02()
8  {
9      set<int> s;
10     //set容器自动根据键值排序
11     s.insert(30);
12     s.insert(10);
13     s.insert(20);
14     s.insert(50);
15     s.insert(40);
16
17     //find(key); //查找键key是否存在,若存在, 返回该键的元素的迭代器;
18     //若不存在, 返回set.end();
19     set<int>::const_iterator ret;
20     ret = s.find(20);
21     if(ret == s.end())
22     {
23         cout<<"没有找到"<<endl;
24     }
25     else
26     {
27         cout<<"找到"<<*ret<<endl;
28     }

```

```
29
30 //count(key); //查找键key的元素个数
31 //set容器的键值 是不重复的 那么count(key)只能是1或0
32 cout<<s.count(20)<<endl;//1
33 cout<<s.count(200)<<endl;//0
34 }
```

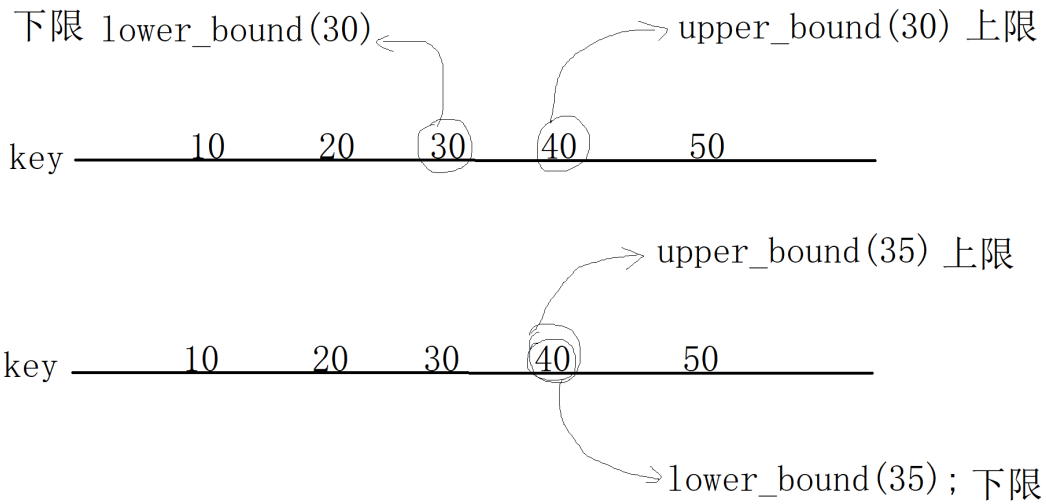
运行结果：



3、set容器的lower_bound和upper_bound

- 1 lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。
- 2 upper_bound(keyElem); //返回第一个key>keyElem元素的迭代器。

```
1 lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。
2 upper_bound(keyElem); //返回第一个key>keyElem元素的迭代器。
```



案例2-2

```
1 /*
2 lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。
3 upper_bound(keyElem); //返回第一个key>keyElem元素的迭代器。
4 equal_range(keyElem); //返回容器中key与keyElem相等的上下限的两个迭代器
5 */
6 void test03()
7 {
8     set<int> s;
9     //set容器自动根据键值排序
10    s.insert(30);
11    s.insert(10);
12    s.insert(20);
13    s.insert(50);
14    s.insert(40);
15
16    //下限:lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。
17    set<int>::const_iterator lower_ret;
18    lower_ret = s.lower_bound(30);
19    if(lower_ret == s.end())
20    {
21        cout<<"没有找到30的下限"<<endl;
22    }
23    else
24    {
```

```

25  cout<<"找到30的下限:"<<*lower_ret<<endl;
26  }
27
28  //upper_bound(keyElem);//返回第一个key>keyElem元素的迭代器
29  set<int>::const_iterator up_ret;
30  up_ret = s.upper_bound(30);
31  if(up_ret == s.end())
32  {
33  cout<<"没有找到30的上限"<<endl;
34  }
35  else
36  {
37  cout<<"找到30的上限:"<<*up_ret<<endl;
38  }
39
40  //equal_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器
41  //equal_range 返回的是对组
42  //first 对组中第一个值 second对组中第二个值
43  pair<set<int>::const_iterator , set<int>::const_iterator> pair_ret;
44  pair_ret = s.equal_range(30);
45  //下限lower_bound
46  if(pair_ret.first == s.end())
47  {
48  cout<<"下限未找到"<<endl;
49  }
50  else
51  {
52  cout<<"下限找到:"<< *(pair_ret.first) <<endl;
53  }
54  //上限upper_bound
55  if(pair_ret.second == s.end())
56  {
57  cout<<"上限未找到"<<endl;
58  }
59  else
60  {
61  cout<<"上限找到:"<< *(pair_ret.second) <<endl;
62  }
63  }

```

运行结果：

找到30的下限:30
找到30的上限:40
下限找到:30
上限找到:40

4、对组

```
1 void test04()  
2 {  
3     //9527--星爷 10086--移动 10010--联通  
4     //定义对组方式1:  
5     pair<int,string> pair1(9527,"星爷");  
6     cout<<"编号:"<<pair1.first<<","人物:"<<pair1.second<<endl;  
7  
8     //定义对组方式1:(推荐)  
9     pair<int,string> pair2 = make_pair(10086,"移动");  
10    cout<<"编号:"<<pair2.first<<","人物:"<<pair2.second<<endl;  
11 }
```

运行结果:

编号:9527, 人物:星爷
编号:10086, 人物:移动

5、更改set容器的默认排序规则 (推荐使用仿函数)

```
1 class MyGreater
```

```

2 {
3 public:
4     bool operator()(int val1, int val2)
5     {
6         return val1>val2;
7     }
8 };
9
10 void test05()
11 {
12     //默认从小-->大排
13     //改成从大-->小排
14     //set<int,排序规则> s;
15     set<int,MyGreater> s;
16
17     s.insert(30);
18     s.insert(10);
19     s.insert(20);
20     s.insert(50);
21     s.insert(40);
22
23     for_each(s.begin(),s.end(),[](int val){cout<<val<<" ";});
24     cout<<endl;
25
26 }
27 int main(int argc, char *argv[])
28 {
29     test05();
30     return 0;
31 }

```

运行结果：

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

50 40 30 20 10

6、set容器存放自定义数据的时候 重载<运算符

```

1  class Person
2  {
3  public:
4      string name;
5      int age;
6      Person(string name, int age)
7      {
8          this->name =name;
9          this->age = age;
10     }
11
12     //方法一：重载<运算符 小-->大
13     bool operator<(const Person &ob)const
14     {
15         return this->age < ob.age;
16     }
17 };
18 void test06()
19 {
20     set<Person> s;
21     s.insert(Person("德玛西亚", 19));
22     s.insert(Person("小法", 18));
23     s.insert(Person("小炮", 21));
24     s.insert(Person("风男", 29));
25
26     for_each(s.begin(),s.end(),[](Person val){
27         cout<<"name="<<val.name<<" , age="<<val.age<<endl;
28     });
29 }

```

运行结果：

name=小法, age=18
name=德玛西亚, age=19
name=小炮, age=21
name=风男, age=29

7、set容器存放自定义数据的时候 指定排序规则。

```
1 class Person
2 {
3 public:
4     string name;
5     int age;
6     Person(string name, int age)
7     {
8         this->name =name;
9         this->age = age;
10    }
11    #if 0
12        //方法一：重载<运算符 小-->大
13        bool operator<(const Person &ob) const
14        {
15            //单纯的告诉编译器 该函数为常函数不会修改成员数据
16            //常对象 调用 常函数
17            return this->age < ob.age;
18        }
19    #endif
20 };
21
22 class myGreaterPerson
23 {
24 public:
25     bool operator()(const Person &ob1, const Person &ob2)
26     {
27         return ob1.age > ob2.age;
```

```

28  }
29  };
30  void test06()
31  {
32      //方法二：指定排序规则
33      set<Person, myGreaterPerson> s;
34      s.insert(Person("德玛西亚", 19));
35      s.insert(Person("小法", 18));
36      s.insert(Person("小炮", 21));
37      s.insert(Person("风男", 29));
38
39      for_each(s.begin(), s.end(), [](Person val){
40          cout<<"name="<<val.name<<"", age="<<val.age<<endl;
41      });
42  }

```

运行结果：

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

```

name=风男, age=29
name=小炮, age=21
name=德玛西亚, age=19
name=小法, age=18

```

知识点9 【multiset允许插入重复的键值】

1、set容器不允许插入相同的key值

```

1  void test07()
2  {
3      set<int> s;
4      pair<set<int>::const_iterator, bool> ret1;
5      pair<set<int>::const_iterator, bool> ret2;
6
7      ret1 = s.insert(10);
8      if(ret1.second == true)
9      {

```

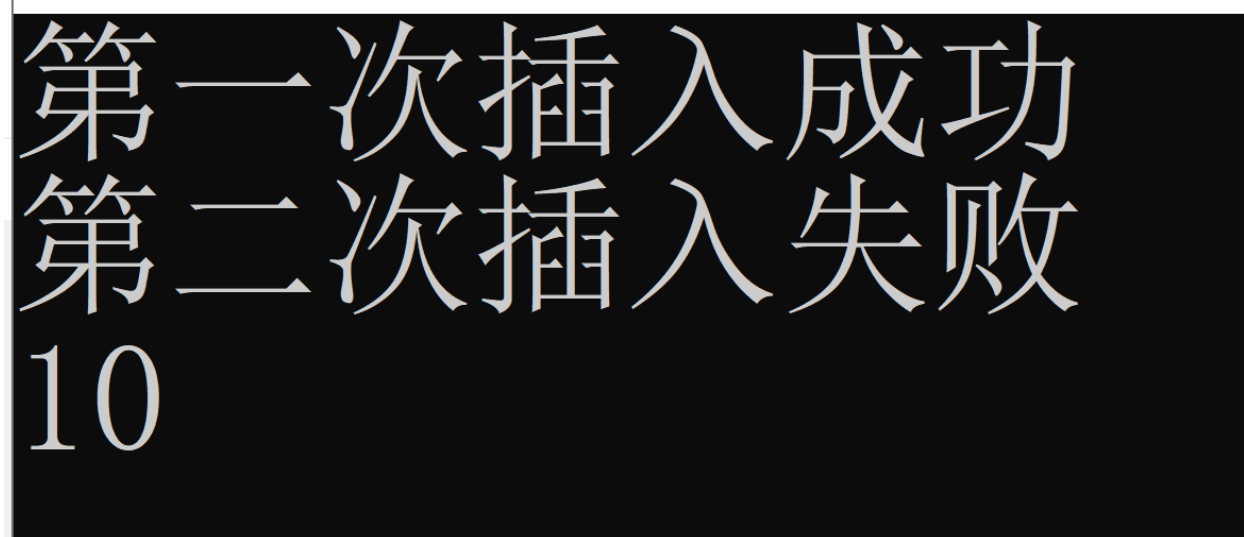
```

10  cout<<"第一次插入成功"<<endl;
11  }
12  else
13  {
14  cout<<"第一次插入失败"<<endl;
15  }
16  ret2 = s.insert(10);
17  if(ret2.second == true)
18  {
19  cout<<"第二次插入成功"<<endl;
20  }
21  else
22  {
23  cout<<"第二次插入失败"<<endl;
24  }
25
26  for_each(s.begin(),s.end(),[](int val){
27  cout<<val<<" "<<endl;
28  });
29  cout<<endl;
30  }

```

运行结果：

C:\Qt\Qt5.6.0\tools\qtcreator\bin\qtcreator_process_sub.exe



第一次插入成功
第二次插入失败
10

2、multiset可以插入重复的键值

```

1  void test07()
2  {
3  multiset<int> s;
4
5  s.insert(10);

```

```
6  s.insert(10);  
7  
8  for_each(s.begin(),s.end(),[](int val){  
9      cout<<val<<" "<<endl;  
10  });  
11  cout<<endl;  
12 }
```

运行结果:

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreat

10

10

