

知识点1【map容器】

map容器的特点：

知识点2【map容器的API】

案例1：map容器的插入、遍历

案例2：map和vector容器配合使用（重要）

知识点3【multimap容器】

知识点4【谓词】

案例：一元谓词

案例2：二元谓词

知识点5【内建函数对象】

知识点6【函数适配器】（了解）

案例：bind2nd 或bind1st区别

知识点7【取反适配器】（了解）

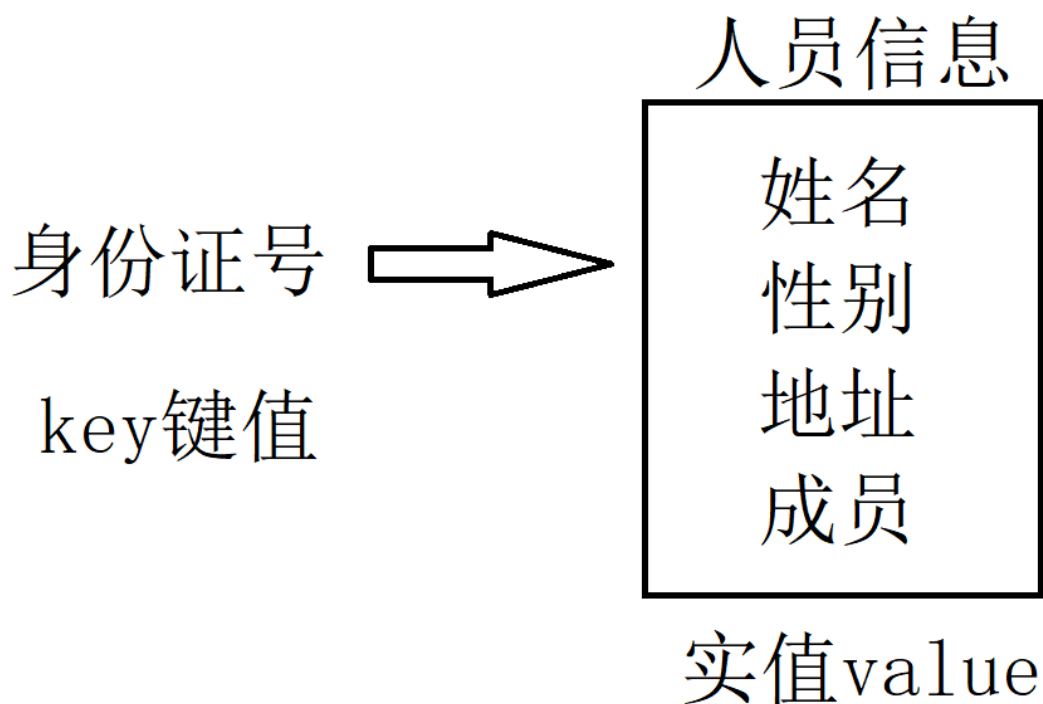
案例2：二元取反not2

知识点8【成员函数适配器】

知识点1【map容器】

map容器的特点：

- 1、所有元素都会根据元素的键值自动排序
 - 2、所有的元素都是pair,同时拥有键值和实值。
- 第一个参数：键值 第二个参数：实值



3、map容器的键值唯一 不可变 实值 可变。

知识点2【map容器的API】

案例1：map容器的插入、遍历

```
1 #include <iostream>
2 #include<map>
3 #include<string>
4 #include<algorithm>
5 using namespace std;
6 /*
7 3.8.2 map/multimap常用API
8 3.8.2.1 map构造函数
9 map<T1, T2> mapTT;//map默认构造函数:
10 map(const map &mp);//拷贝构造函数
11 3.8.2.2 map赋值操作
12 map& operator=(const map &mp);//重载等号操作符
13 swap(mp);//交换两个集合容器
14 3.8.2.3 map大小操作
15 size();//返回容器中元素的数目
16 empty();//判断容器是否为空
17 3.8.2.4 map插入数据元素操作
18 map.insert(...); //往容器插入元素, 返回pair<iterator,bool>
19 map<int, string> mapStu;
20 // 第一种 通过pair的方式插入对象
```

```

21 mapStu.insert(pair<int, string>(3, "小张"));
22 // 第二种 通过pair的方式插入对象
23 mapStu.inset(make_pair(-1, "校长"));
24 // 第三种 通过value_type的方式插入对象
25 mapStu.insert(map<int, string>::value_type(1, "小李"));
26 // 第四种 通过数组的方式插入值
27 mapStu[3] = "小刘";
28 mapStu[5] = "小王";
29 3.8.2.5 map删除操作
30 clear();//删除所有元素
31 erase(pos);//删除pos迭代器所指的元素，返回下一个元素的迭代器。
32 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。
33 erase(keyElem);//删除容器中key为keyElem的对组。
34 3.8.2.6 map查找操作
35 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map.end();
36 count(keyElem);//返回容器中key为keyElem的对组个数。对map来说，要么是0，要么是1。对multimap来说，值可能大于1。
37 lower_bound(keyElem);//返回第一个key>=keyElem元素的迭代器。
38 upper_bound(keyElem);//返回第一个key>keyElem元素的迭代器。
39 equal_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器。
40 */
41 void myPrintMap01(map<int,string> &m)
42 {
43     for(map<int,string>::const_iterator it=m.begin();it!=m.end();it++)
44     {
45         /*it == <int,string>
46         cout<<"key="<<(*it).first<< ", value="<<(*it).second<<endl;
47     }
48 }
49 void test01()
50 {
51     //存放9527--"星爷" 10086-"移动" 10010--"联通" 10000--"电信"
52     //int为键值的类型 string为实值的类型
53     map<int,string> m;
54
55     //第1种:
56     m.insert(pair<int,string>(9527,"星爷"));
57     //第2种: (推荐)
58     m.insert(make_pair(10086,"移动"));
59     //第3种:

```

```

60  m.insert(map<int,string>::value_type(10010,"联通"));
61  //第4种：读map容器数据的时候 推荐
62  m[10000] ="电信";//m.insert(make_pair(10000,"电信"));
63
64  //m[10] 寻找key为10的实值
65  //但是：如果容器中没有key为10 使用m[10] 会创建一个key为10实值为空的 对组
66  //如果容器中有key为10 那么m[10]代表key=10的实值
67  //建议：10是存在的
68  cout<<m[10]<<endl;
69
70  myPrintMap01(m);
71  #if 0
72  for_each(m.begin(),m.end(),[](pair<int,string> val){
73      cout<<"key="<<val.first<<" , value="<<val.second<<endl;
74  });
75  #endif
76
77  //只想查看key==9527 的实值 （保证key是存在的）
78  cout<<m[9527]<<endl;//"星爷"
79  cout<<m[10010]<<endl;//"联通"
80
81  //如果不能确定key值是否存在
82  map<int,string>::const_iterator ret;
83  ret = m.find(10086);
84  if(ret == m.end())
85  {
86      cout<<"未找到相关节点"<<endl;
87  }
88  else
89  {
90      //ret代表的是key=10086的对组的迭代器
91      cout<<"找到相关节点:key="<<\
92      (*ret).first<<" , value="<<(*ret).second <<endl;
93  }
94
95  }
96  int main(int argc, char *argv[])
97  {
98      test01();
99      return 0;

```

```
100 }  
101
```

运行结果：

```
key=10, value=  
key=9527, value=星爷  
key=10000, value=电信  
key=10010, value=联通  
key=10086, value=移动  
星爷  
联通  
找到相关节点:key=10086, value=移动
```

案例2：map和vector容器配合使用（重要）

LOL职业联赛：有4个战队 随机抽签出场 请打印出场顺序

```
1  #include<numeric>//极少的算法  
2  #include<stdlib.h>  
3  #include<time.h>  
4  void test02()  
5  {  
6      //设置种子  
7      srand(time(NULL));  
8  
9      //战队容器（战队编号,战队名称）  
10     map<int,string> m;  
11     m.insert(make_pair(1,"RNG"));  
12     m.insert(make_pair(2,"IG"));  
13     m.insert(make_pair(3,"WE"));  
14     m.insert(make_pair(4,"EDG"));  
15  
16     //使用vector存放战队编号  
17     vector<int> v;  
18     v.push_back(1);  
19     v.push_back(2);  
20     v.push_back(3);  
21     v.push_back(4);  
22
```

```

23 //随机抽签(洗牌) 打乱容器的元素顺序
24 random_shuffle(v.begin(),v.end());
25
26 //随机出场
27 for_each(v.begin(),v.end(), [&](int val){
28 //val就是m容器中的key值
29 cout<<m[val]<<"战队出场了"<<endl;
30 } );
31
32
33 }

```

运行结果：

WE战队出场了
EDG战队出场了
RNG战队出场了
IG战队出场了

知识点3 【multimap容器】

multimap允许key相同，map不允许key相同

```

1 void test03()
2 {
3 //战队容器（战队编号,战队名称）
4 map<int,string> m;
5 m.insert(make_pair(1,"RNG"));
6 m.insert(make_pair(1,"IG"));
7

```

```

8  cout<<m.count(1)<<endl;//2
9
10  multimap<int,string> m1;
11  m1.insert(make_pair(1,"RNG"));
12  m1.insert(make_pair(1,"IG"));
13  cout<<m1.count(1)<<endl;//2
14  }

```

案例：5名员工 加入3个部门

```

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 void createVectorPerson(vector<Person> &v)
13 {
14     v.push_back(Person("员工A", 21));
15     v.push_back(Person("员工B", 23));
16     v.push_back(Person("员工C", 21));
17     v.push_back(Person("员工D", 22));
18     v.push_back(Person("员工E", 21));
19 }
20 #include<vector>
21
22 void PersonByGroup(vector<Person> &v, multimap<int, Person> &m)
23 {
24     //逐个将员工分配到 各个部门内
25     for(vector<Person>::iterator it=v.begin();it!=v.end();it++)
26     {
27         /*it == Person
28         cout<<"请输入"<<(*it).name<<"将要加入的部门:1 2 3"<<endl;
29         int operate = 0;
30         cin>>operate;
31         if(operate >=1 && operate<=3)
32         {

```

```

33  m.insert(make_pair(operate, *it));
34  }
35
36  }
37  }
38
39  void showPersonList(multimap<int, Person> &m, int op)
40  {
41      switch (op) {
42          case 1:
43              cout<<"研发部:"<<endl;
44              break;
45          case 2:
46              cout<<"测试部:"<<endl;
47              break;
48          case 3:
49              cout<<"人事部:"<<endl;
50              break;
51      }
52
53      //注意: m中存放的数据《部门号、员工信息》
54      //统计相同部门号的元素个数
55      int n = m.count(op);
56      cout<<"部门的人数个数: "<<n<<endl;
57      //由于 multimap将key自动排序 重复 相同key一定挨在一起
58      multimap<int, Person>::const_iterator ret;
59      ret = m.find(op);
60      if(ret != m.end())//寻找到key
61      {
62          for(int i=0;i<n;i++,ret++)
63          {
64              //(*ret) == <int, Person>
65              //(*ret).second == Person
66              cout<<"name = "<<(*ret).second.name<<",age = "<<\
67              (*ret).second.age<<endl;
68          }
69      }
70
71  }
72  void test04()

```



```
73 {  
74     //使用vector容器将员工信息存储起来  
75     vector<Person> v;  
76  
77     //创建5名员工  
78     createVectorPerson(v);  
79  
80     //将员工分配到不同的部门（员工<部门号、员工信息>）  
81     multimap<int, Person> m; //存放员工<部门号、员工信息>  
82     PersonByGroup(v, m);  
83  
84     //按部门查看员工信息  
85     showPersonList(m, 1);  
86     showPersonList(m, 2);  
87     showPersonList(m, 3);  
88 }
```

运行结果：

```
请输入员工A将要加入的部门:1 2 3  
1  
请输入员工B将要加入的部门:1 2 3  
1  
请输入员工C将要加入的部门:1 2 3  
2  
请输入员工D将要加入的部门:1 2 3  
2  
请输入员工E将要加入的部门:1 2 3  
3
```

```
name = 员工A, age = 21
name = 员工B, age = 23
测试部：
部门的人数个数： 2
name = 员工C, age = 21
name = 员工D, age = 22
人事部：
部门的人数个数： 1
name = 员工E, age = 21
```

知识点4【谓词】

返回值类型为`bool`的普通函数或仿函数 就叫谓词。

如果普通函数或仿函数 有一个参数 就叫一元谓词。

如果普通函数或仿函数 有二个参数 就叫二元谓词。

案例：一元谓词

```
1 //普通函数作为一元谓词
2 bool greaterThan20(int val)
3 {
4     return val>20;
5 }
6 //仿函数作为一元谓词
7 class MyGreaterThan20
8 {
9     public:
10    bool operator()(int val)
11    {
12        return val>20;
13    }
14 };
15 void test01()
```

```

16 {
17     vector<int> v;
18     v.push_back(10);
19     v.push_back(20);
20     v.push_back(30);
21     v.push_back(40);
22     v.push_back(50);
23
24     for_each(v.begin(),v.end(),[](int val){cout<<val<<" ";});
25     cout<<endl;
26     //需求：找出第一个大于20的数
27     vector<int>::iterator ret;
28     //普通函数完成
29     //ret = find_if(v.begin(),v.end(), greaterThan20);
30     ret = find_if(v.begin(),v.end(), MyGreaterThan20());
31     if(ret != v.end())
32     {
33         cout<<"第一个大于20的数为:"<<*ret<<endl;
34     }
35
36 }

```

运行结果：

10 20 30 40 50
第一个大于20的数为:30

案例2：二元谓词

```

1 //普通函数作为二元谓词
2 bool myGreater(int v1,int v2)
3 {
4     //为啥从大到小 排序
5     return v1>v2;
6 }
7 //仿函数作为二元谓词
8 class MyGreater


```

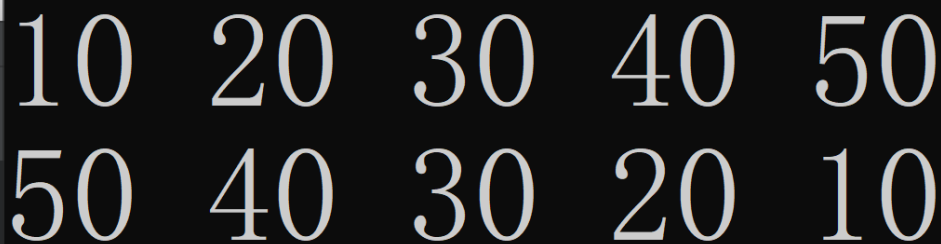
```

9 {
10 public:
11     bool operator()(int v1,int v2)
12     {
13         return v1>v2;
14     }
15 };
16
17 void test02()
18 {
19     vector<int> v;
20     v.push_back(10);
21     v.push_back(20);
22     v.push_back(30);
23     v.push_back(40);
24     v.push_back(50);
25
26     for_each(v.begin(),v.end(),[](int val){cout<<val<<" ";});
27     cout<<endl;
28
29     //从大-->小排序
30     //sort(v.begin(),v.end(), myGreater);
31     sort(v.begin(),v.end(), MyGreater());
32     for_each(v.begin(),v.end(),[](int val){cout<<val<<" ";});
33     cout<<endl;
34 }

```

运行结果：

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



```

10 20 30 40 50
50 40 30 20 10

```

知识点5 【内建函数对象】

6 个算数类函数对象,除了 `negate` 是一元运算, 其他都是二元运算。

```
template<class T> T plus<T> //加法仿函数
template<class T> T minus<T> //减法仿函数
template<class T> T multiplies<T> //乘法仿函数
template<class T> T divides<T> //除法仿函数
template<class T> T modulus<T> //取模仿函数
template<class T> T negate<T> //取反仿函数
```

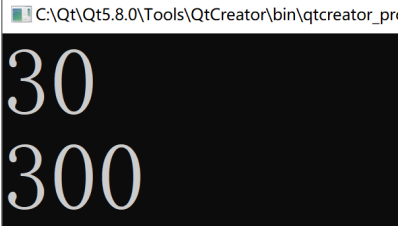
6 个关系运算类函数对象,每一种都是二元运算。

```
template<class T> bool equal_to<T> //等于
template<class T> bool not_equal_to<T> //不等于
template<class T> bool greater<T> //大于
template<class T> bool greater_equal<T> //大于等于
template<class T> bool less<T> //小于
template<class T> bool less_equal<T> //小于等于
```

逻辑运算类运算函数, `not` 为一元运算, 其余为二元运算。

```
template<class T> bool logical_and<T> //逻辑与
template<class T> bool logical_or<T> //逻辑或
template<class T> bool logical_not<T> //逻辑非
```

```
void test03()
{
    plus<int> p;
    cout<<p(10,20)<<endl;
    cout<<plus<int>()(100,200)<<endl;
}
```



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

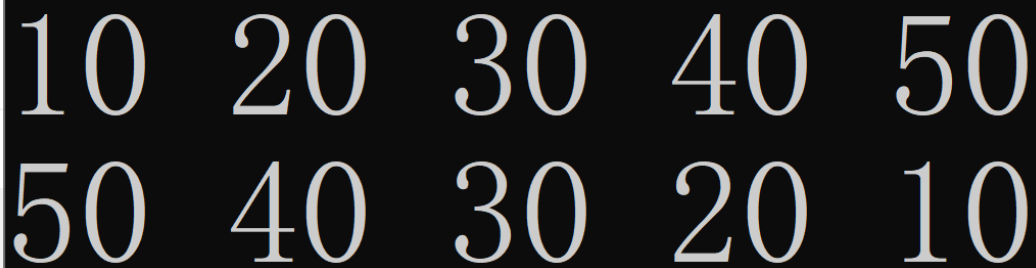
30
300

```
1 void test04()
2 {
3     vector<int> v;
4     v.push_back(10);
5     v.push_back(20);
6     v.push_back(30);
7     v.push_back(40);
8     v.push_back(50);
```

```
9
10  for_each(v.begin(),v.end(),[](int val){cout<<val<<" ";});
11  cout<<endl;
12
13  //使用内建函数对象 改变排序规则
14  sort(v.begin(),v.end(), greater<int>() );
15
16  for_each(v.begin(),v.end(),[](int val){cout<<val<<" ";});
17  cout<<endl;
18 }
```

运行结果：

C:\Qt\Qt5.5.0\tools\qtcreator\bin\qtcreator_process_stable.exe



```
10 20 30 40 50
50 40 30 20 10
```

知识点6【函数适配器】（了解）

绿联

泰雨行动

USB3.0设计, 1GB
插上即用【电脑转

扩展函数的参数接口（假如函数有一个参数 再扩展一个接口 据可以传递两个参数）

```
1 //val 是for_each提供 tmp
2 //适配器2: 公共继承binary_function
3 //适配器3: 参数的萃取
4 //适配器4: 对operator()进行const修饰
5 class MyPrint:public binary_function<int,int, void>
6 {
7 public:
8     void operator()(int val,int tmp) const
9     {
10         cout<<val+tmp<<" ";
11     }
12 };
13
14 void test05()
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  v.push_back(50);
22
23  //适配器1: bind2nd 或bind1st 绑定参数
24  for_each(v.begin(), v.end(), bind2nd(MyPrint(),1000) );
25  cout<<endl;
26  }

```

运行结果:

1010 1020 1030 1040 1050

案例: bind2nd 或bind1st区别

bind2nd: 讲外界数据 绑定到第二个参数

bind1st: 讲外界数据 绑定到第一个参数

```

1  void test05()
2  {
3      vector<int> v;
4      v.push_back(10);
5      v.push_back(20);
6      v.push_back(30);
7      v.push_back(40);
8      v.push_back(50);
9
10     //适配器1: bind2nd 或bind1st 绑定参数
11     cout<<"bind2nd"<<endl;
12     for_each(v.begin(), v.end(), bind2nd(MyPrint(),1000) );
13     cout<<endl;
14
15     cout<<"-----"<<endl;
16     cout<<"bind1st"<<endl;
17     for_each(v.begin(), v.end(), bind1st(MyPrint(),1000) );
18     cout<<endl;
19  }

```

运行结果:

bind2nd

val=10, tmp = 1000

val=20, tmp = 1000

val=30, tmp = 1000

val=40, tmp = 1000

val=50, tmp = 1000

bind1st

val=1000, tmp = 10

val=1000, tmp = 20

val=1000, tmp = 30

val=1000, tmp = 40

val=1000, tmp = 50

知识点7 【取反适配器】（了解）

not1一元取反

not2二元取反

```

1 //取反适配器2:public unary_function
2 //取反适配器3:参数萃取
3 //取反适配器4:const修饰operator()
4 class MyGreaterThanOr3:public unary_function<int,bool>
5 {
6 public:
7     //一元谓词
8     bool operator()(int val)const
9     {
10         return val>3;
11     }
12 };
13 void test06()
14 {
15     vector<int> v;
16     v.push_back(1);
17     v.push_back(2);
18     v.push_back(3);
19     v.push_back(4);
20     v.push_back(5);
21     //找出第一个大于3的数
22     vector<int>::iterator ret;
23     ret = find_if(v.begin(),v.end(), MyGreaterThanOr3() );
24     if(ret != v.end())
25     {
26         cout<<"*ret = "<<*ret<<endl;//4
27     }
28
29     //找出第一个小于3的数
30     //取反适配器1:not1修饰
31     ret = find_if(v.begin(),v.end(), not1(MyGreaterThanOr3()) );
32     if(ret != v.end())
33     {
34         cout<<"*ret = "<<*ret<<endl;//4
35     }
36 }

```

运行结果：

```
*ret = 4
*ret = 1
```

注意:

binary_function 二元继承

unary_function 一元继承

案例2：二元取反not2


```
1  class MyGreaterInt:public binary_function<int,int,bool>
2  {
3  public:
4      bool operator()(int v1,int v2)const
5      {
6          return v1>v2;
7      }
8  };
9  void test07()
10 {
11     vector<int> v;
12     v.push_back(2);
13     v.push_back(1);
14     v.push_back(5);
15     v.push_back(3);
16     v.push_back(4);
17
18     for_each(v.begin(),v.end(), [](int v){cout<<v<<" ";});
19     cout<<endl;
20
21     //默认小--->大
22     //sort(v.begin(),v.end());
```

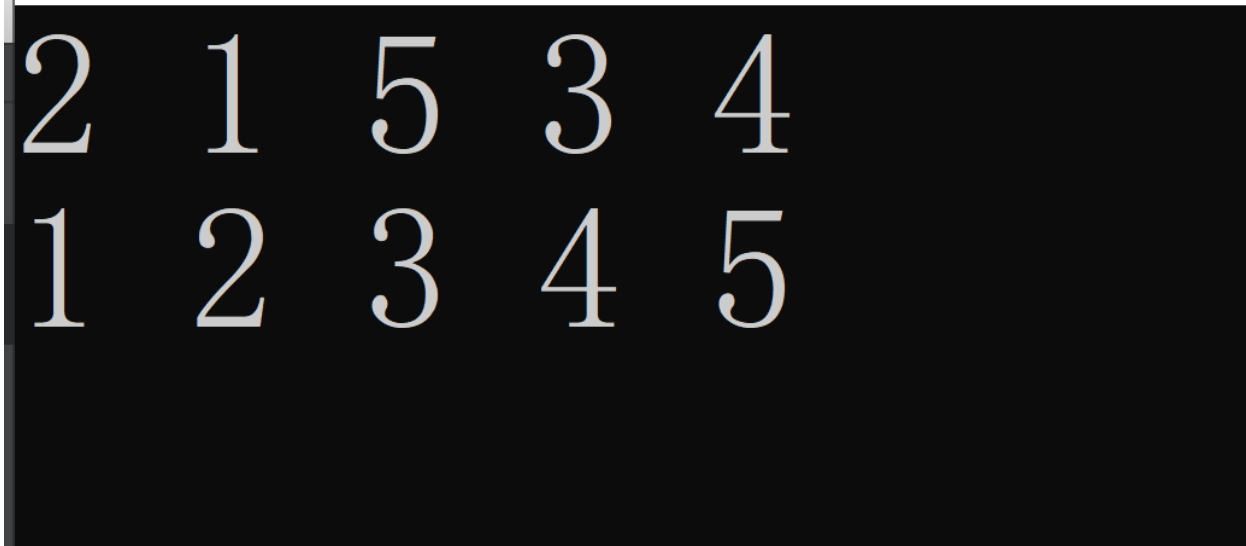
```

23 //更改排序规则大-->小
24 //sort(v.begin(),v.end(), MyGreaterInt());
25 //使用not2对MyGreaterInt()取反 小--->大
26 //sort(v.begin(),v.end(), not2(MyGreaterInt()));
27 //使用not2对内建函数取反
28 sort(v.begin(),v.end(), not2(greater<int>()));
29 //sort(v.begin(),v.end(), less<int>());
30 for_each(v.begin(),v.end(), [](int v){cout<<v<<" ";});
31 cout<<endl;
32 }

```

运行结果：

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



知识点8 【成员函数适配器】

```

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    void showPerson()
12    {
13        cout<<"name = "<<this->name<<",age="<<this->age<<endl;

```

```

14  }
15  };
16  void myPrintPerson(Person &ob)
17  {
18      cout<<"name = "<<ob.name<<",age="<<ob.age<<endl;
19  }
20  void test08()
21  {
22      vector<Person> v;
23      v.push_back(Person("德玛西亚",18));
24      v.push_back(Person("狗头",28));
25      v.push_back(Person("牛头",19));
26      v.push_back(Person("小法",38));
27
28      //遍历 myPrintPerson普通函数
29      //for_each(v.begin(),v.end(), myPrintPerson );
30      //遍历 Person成员函数
31      //利用 mem_fun_ref 将Person内部成员函数适配
32      for_each(v.begin(),v.end(), mem_fun_ref(&Person::showPerson) );
33  }

```

运行结果：

```

name = 德玛西亚, age=18
name = 狗头, age=28
name = 牛头, age=19
name = 小法, age=38

```