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

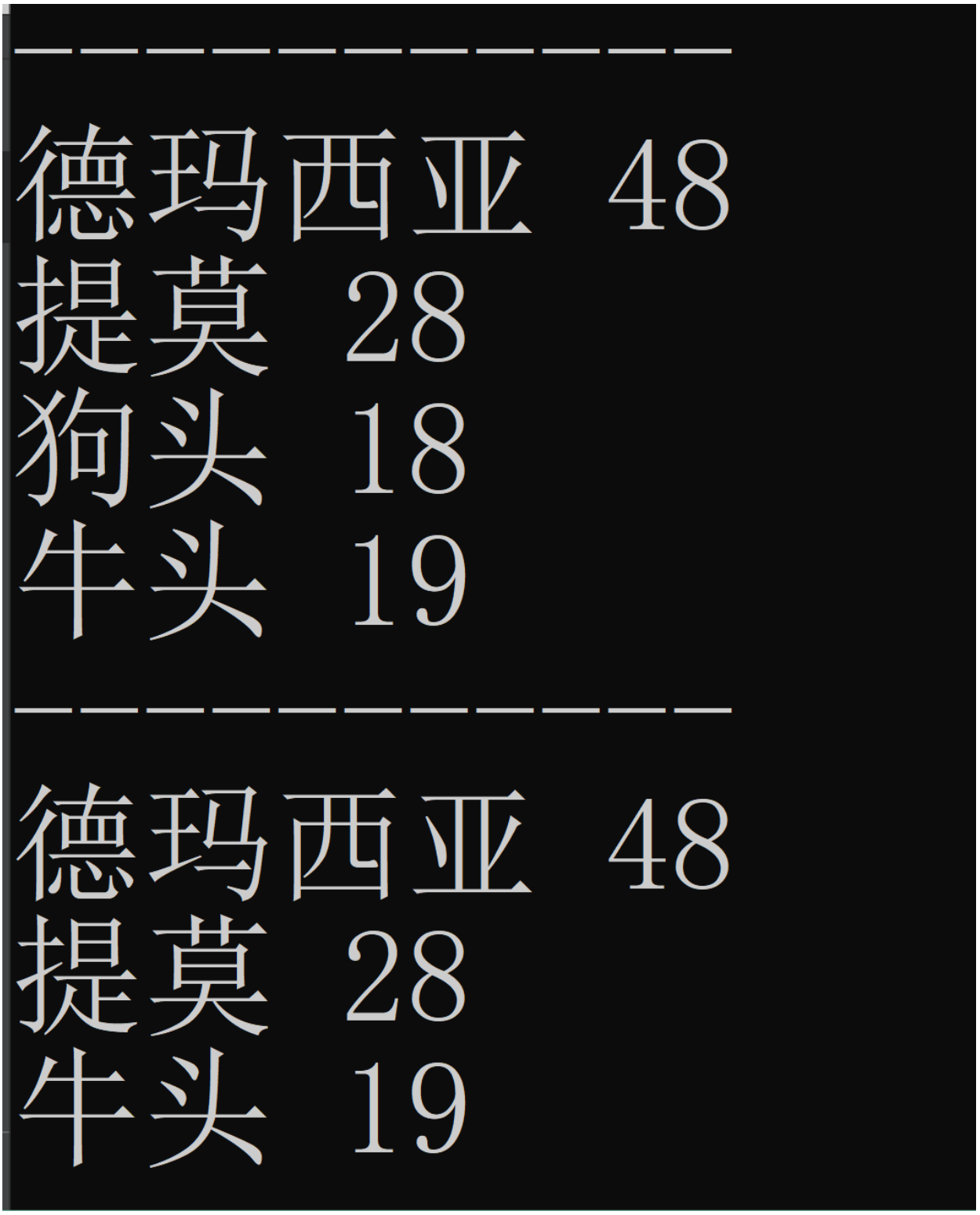
2、multiset可以插入重复的键值

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】

```
#include <iostream>
#include <list>
#include<algorithm>
#include<string>
using namespace std;
class Person
{
public:
 string name;
  int age;
public:
  Person(string name,int age)
  {
  this->name = name;
  this->age = age;
  }

  //成员函数重载 ==运算符
  bool operator==(const Person &ob)
  {
  if(this->name == ob.name && this->age == ob.age)
  return true;
  return false;
  }
};
void printListPerson(list<Person> &L)
{
  cout<<"-----------"<<endl;
  for(list<Person>::iterator it=L.begin(); it!= L.end();it++)
  {
  cout<<(*it).name<<" "<<(*it).age<<endl;
  }
}
void test01()
```

```
{
 //存放自定义数据
 list<Person> L;
 L.push_back(Person("德玛西亚",48));
 L.push_back(Person("提莫", 28));
 L.push_back(Person("狗头", 18));
 L.push_back(Person("牛头", 19));

 printListPerson(L);

 //删除狗头
 Person tmp("狗头", 18);
 //重载==运算符
 L.remove(tmp);
 printListPerson(L);
}
int main(int argc, char *argv[])
{
 test01();
 return 0;
}

```

运行结果：



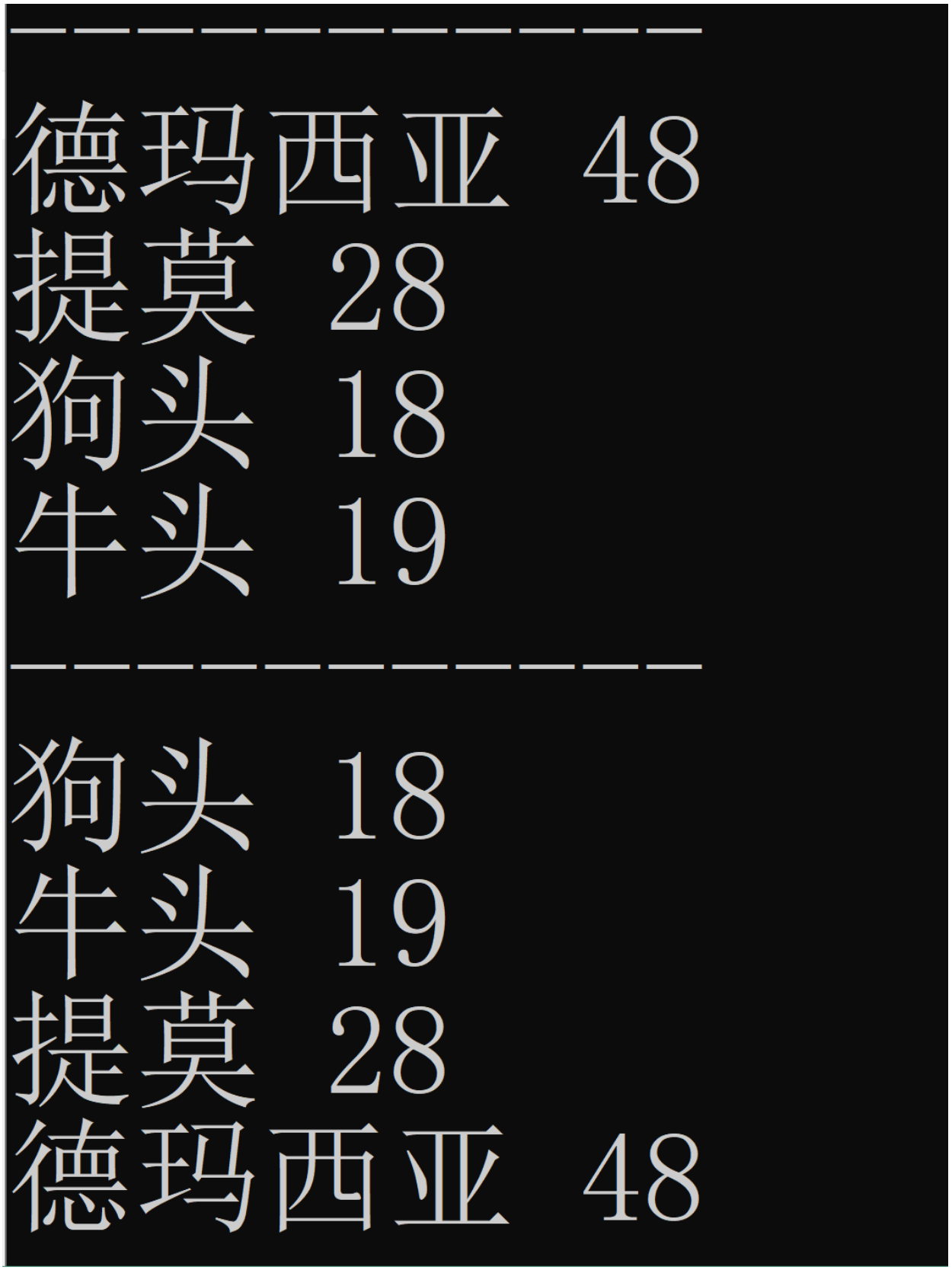
知识点2【list容器 对自定义数据排序 重载<运算符】

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

```
 return true;

 return false;
 }
```

```
void test02()
{
 //存放自定义数据
 list<Person> L;
 L.push_back(Person("德玛西亚",48));
 L.push_back(Person("提莫", 28));
 L.push_back(Person("狗头", 18));
 L.push_back(Person("牛头", 19));
 printListPerson(L);

 //对于自定义数据 我们可以重载<运算符
 L.sort();

 printListPerson(L);
}
```

运行结果：

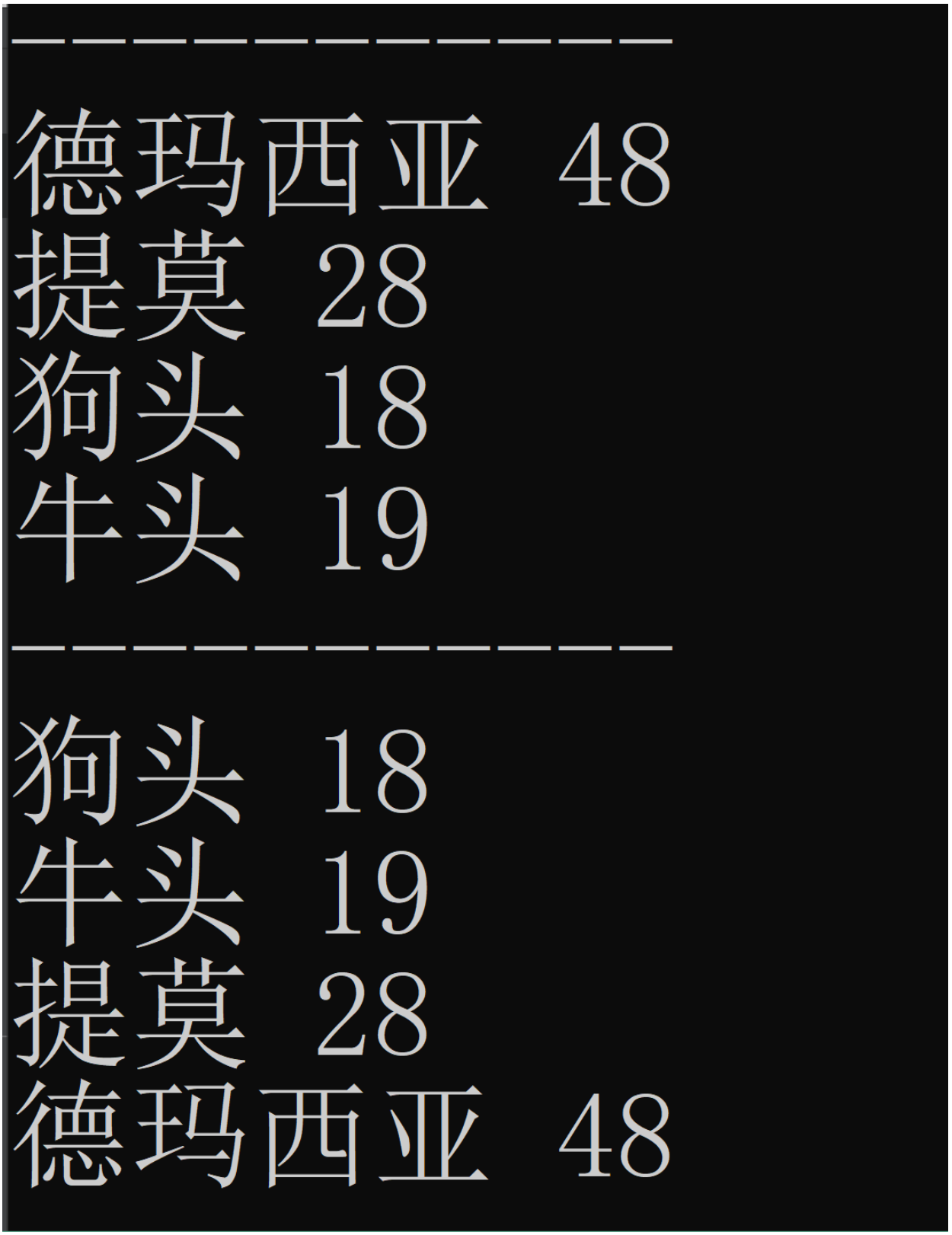


知识点3【list 对自定义数据排序 指定排序规则】

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

```
void test03()
{
 //存放自定义数据
 list<Person> L;
 L.push_back(Person("德玛西亚",48));
  L.push_back(Person("提莫", 28));
  L.push_back(Person("狗头", 18));
  L.push_back(Person("牛头", 19));
  printListPerson(L);

  //对于自定义数据 我们可以重载<运算符
  L.sort(myComparePerson);

  printListPerson(L);
}
```

运行结果：



知识点4【vector 对自定义数据 排序指定排序规则】1-2

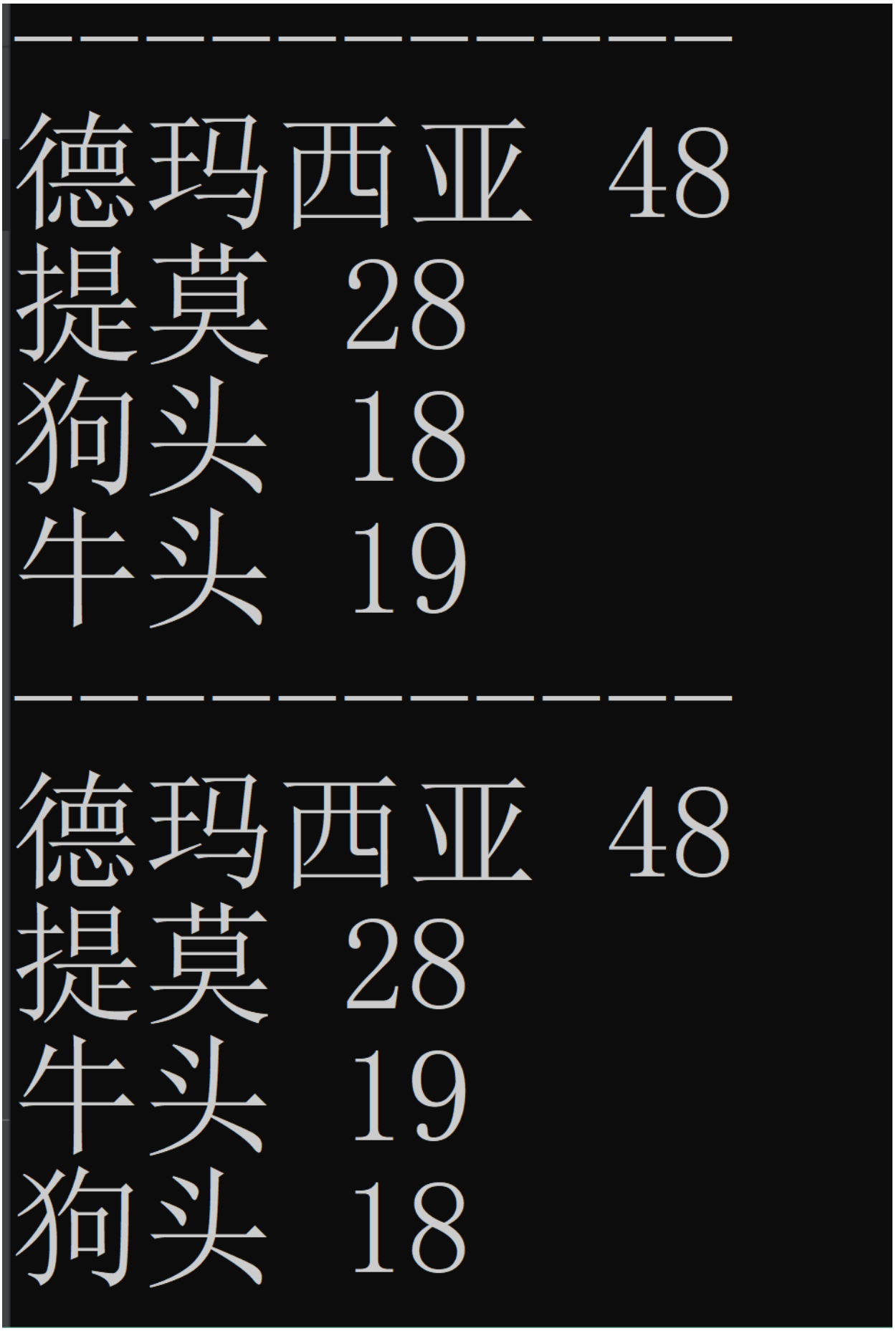
1、方法一 普通函数 实现排序规则

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

```
{
 return ob1.age > ob2.age;
}
```

```
void test04()
{
 vector<Person> v;
 v.push_back(Person("德玛西亚",48));
 v.push_back(Person("提莫", 28));
 v.push_back(Person("狗头", 18));
 v.push_back(Person("牛头", 19));
 PrintVectorPerson(v);

 //默认比较方式从小-->大
 //vector存放自定义数据 可指定排序规则 (普通函数)
 sort(v.begin(),v.end(),myComparePerson);

 PrintVectorPerson(v);
}
```

运行结果：

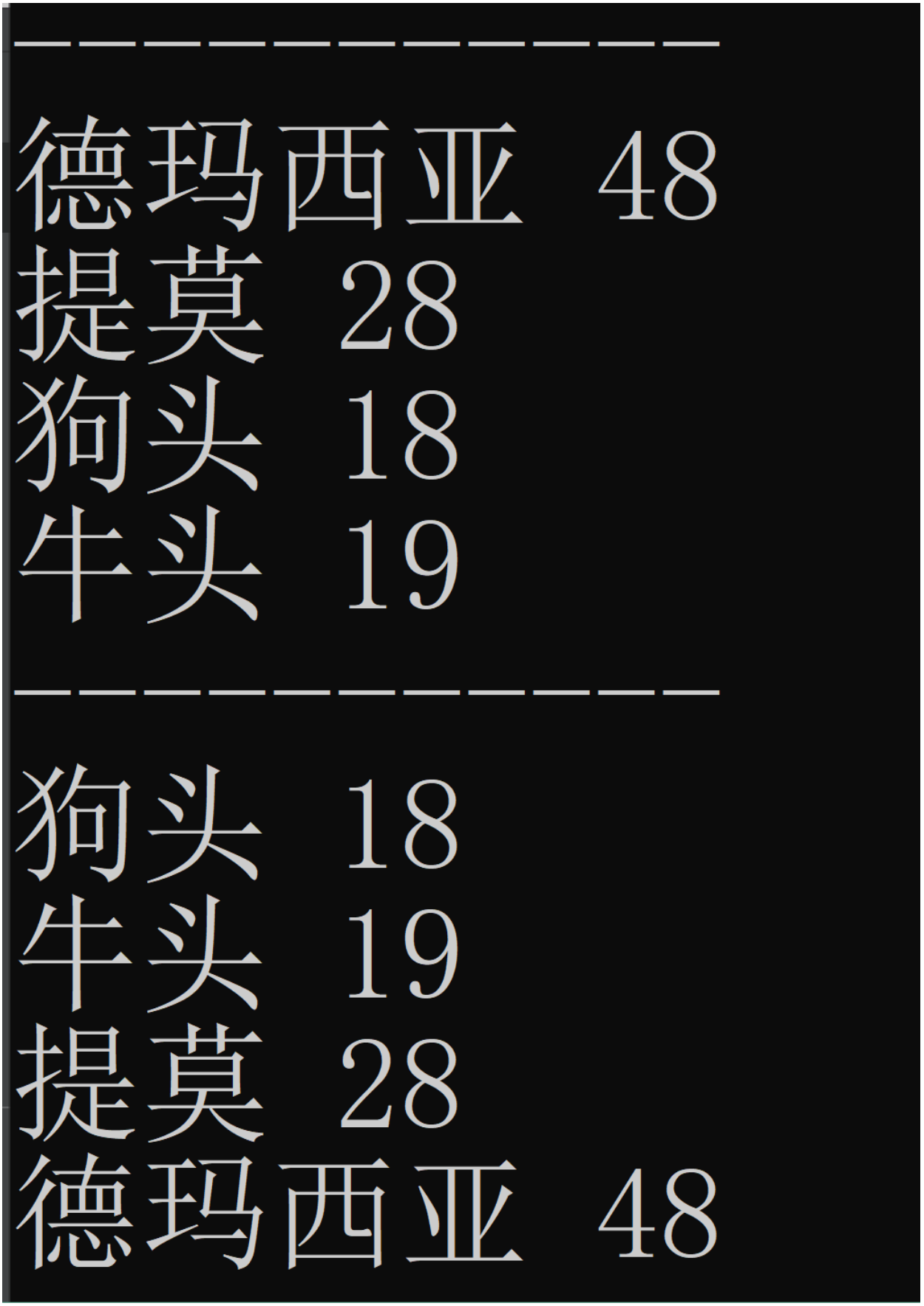


2、方法二 仿函数 指定排序规则

```
//仿函数指定排序规则
class myComparePerson2
{
public:
 bool operator()(Person &ob1, Person &ob2)
 {
 return ob1.age < ob2.age;
 }
};
```

```
void test04()
{
 vector<Person> v;
 v.push_back(Person("德玛西亚",48));
 v.push_back(Person("提莫", 28));
 v.push_back(Person("狗头", 18));
 v.push_back(Person("牛头", 19));
 PrintVectorPerson(v);

 //默认比较方式从小-->大
 //vector存放自定义数据 可指定排序规则 (普通函数)
 //sort(v.begin(),v.end(), myComparePerson);

 //vector存放自定义数据 可指定排序规则 (仿函数)
 sort(v.begin(),v.end(), myComparePerson2() );

 PrintVectorPerson(v);
}
```

运行结果：

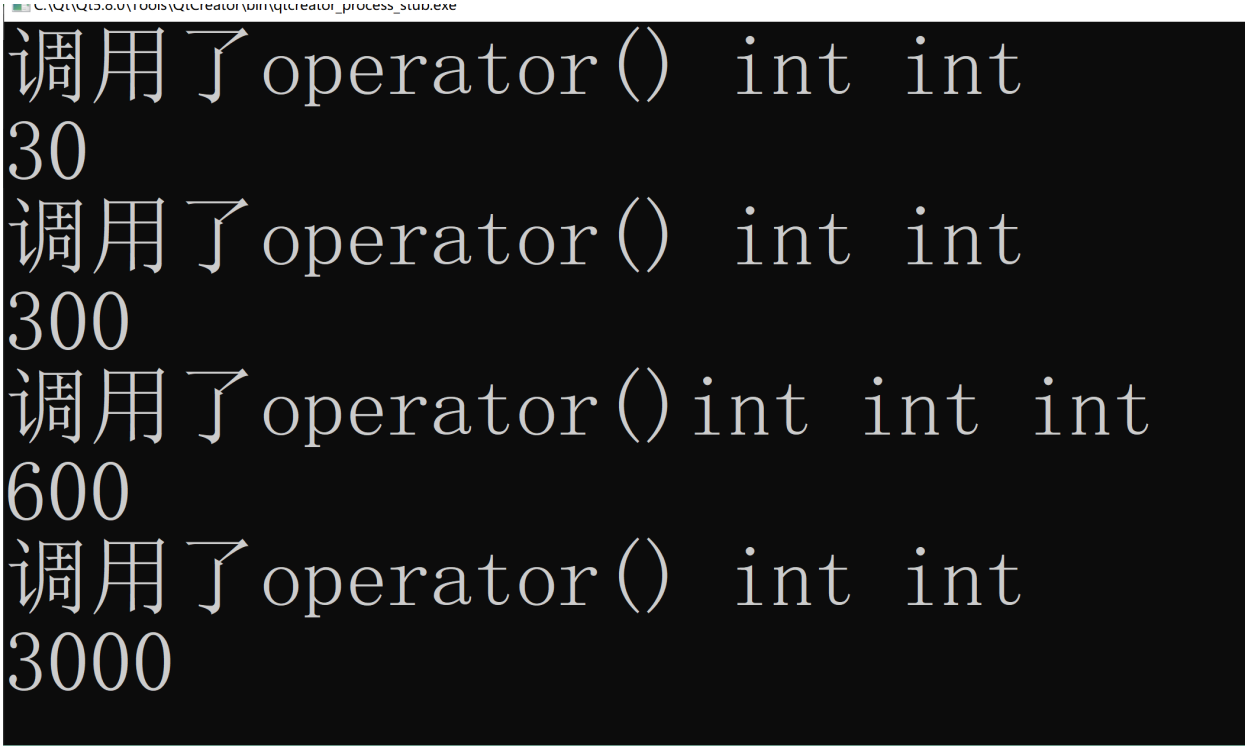


知识点5【仿函数 重载函数调用运算符()】

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

```
//本质就是成员函数 函数名叫operator()
int operator()(int a, int b)
{
cout<<"调用了operator() int int"<<endl;
return a+b;
}
int operator()(int a, int b, int c)
 {
 cout<<"调用了operator()int int int "<<endl;
 return a+b+c;
 }

};
void test05()
{
 MyAdd ob1;
 cout<<ob1.operator ()(10,20)<<endl;
 //编译器优化
 //严格意义：ob1是对象 和（）结合 调用operator ()成员函数
 //ob1(100,200) ob1本质是对象不是函数名 只是形式像函数调用
 //叫做仿函数
 cout<<ob1(100,200)<<endl;
 cout<<ob1(100,200,300)<<endl;

 cout<<MyAdd()(1000,2000)<<endl;

}
```

运行结果：

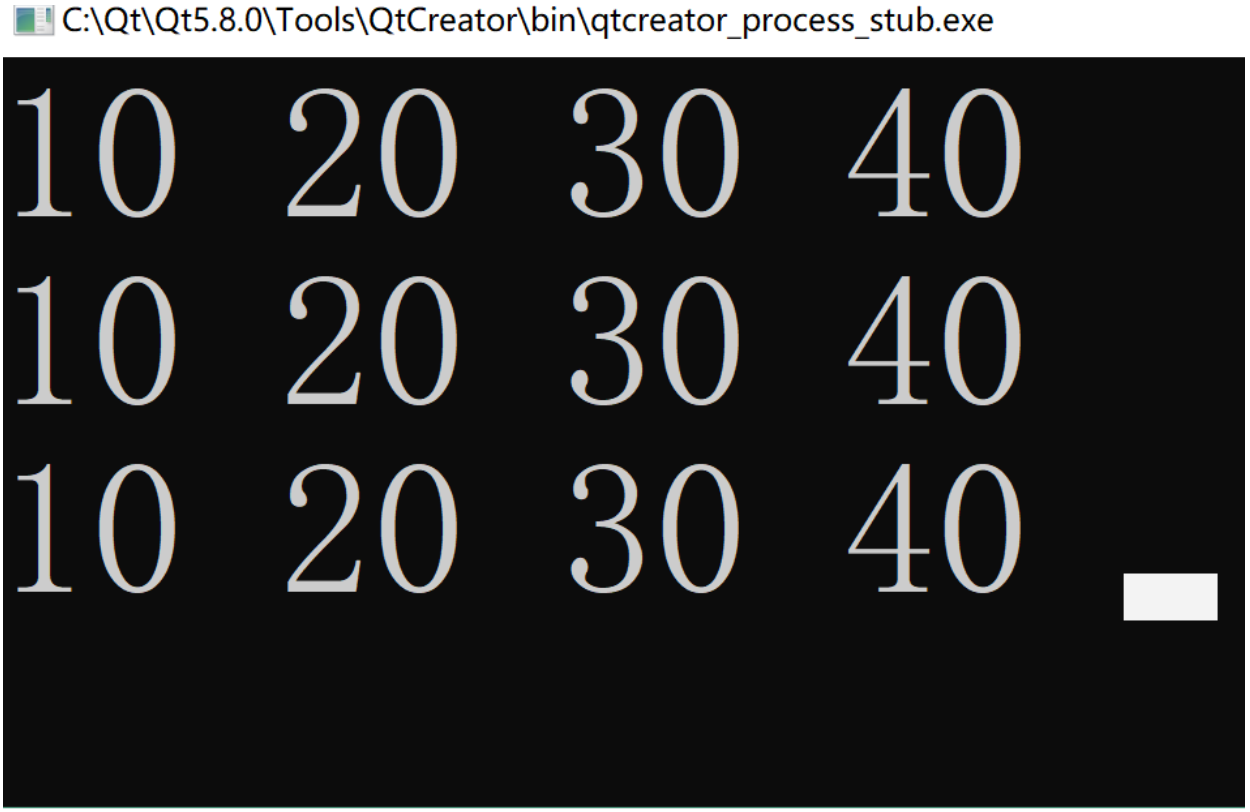


知识点6【lambda表达式】

```
void myPrintVectorInt(vector<int> &v)
{
 for(vector<int>::iterator it=v.begin();it!=v.end();it++)
 {
 cout<<*it<<" ";
 }
 cout<<endl;
}
//指定打印方式
void myPrint01(int val)
{
 cout<<val<<" ";
}
void test06()
{
 vector<int> v;
 v.push_back(10);
 v.push_back(20);
 v.push_back(30);
 v.push_back(40);

 //方法1:访问v容器 普通函数
 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 }
```

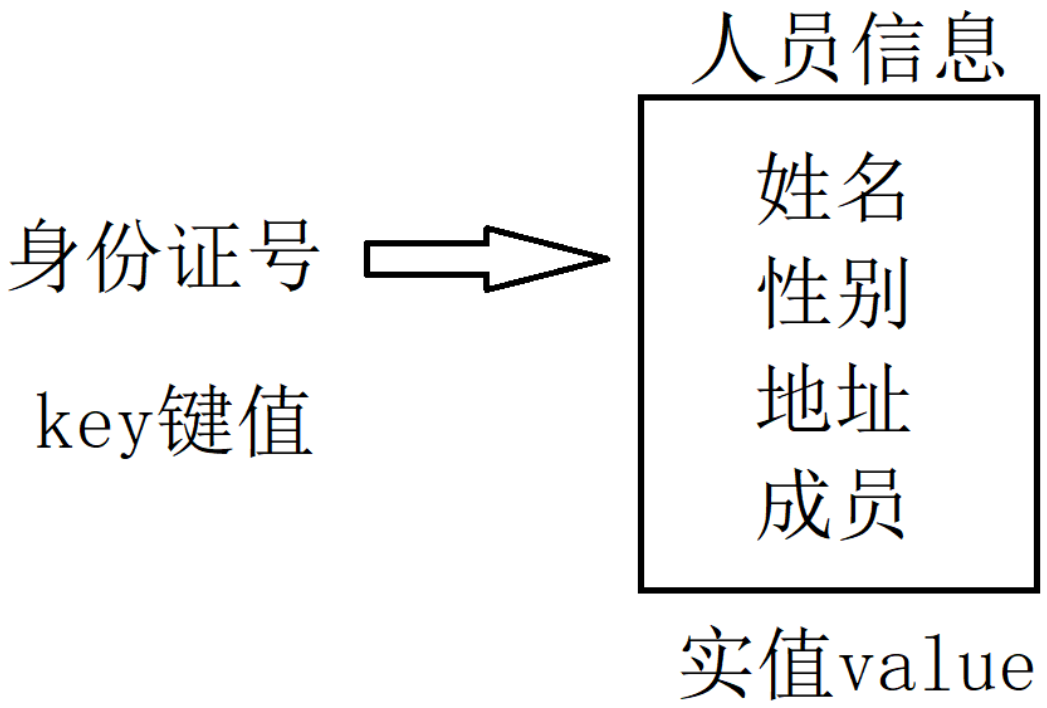
运行结果：



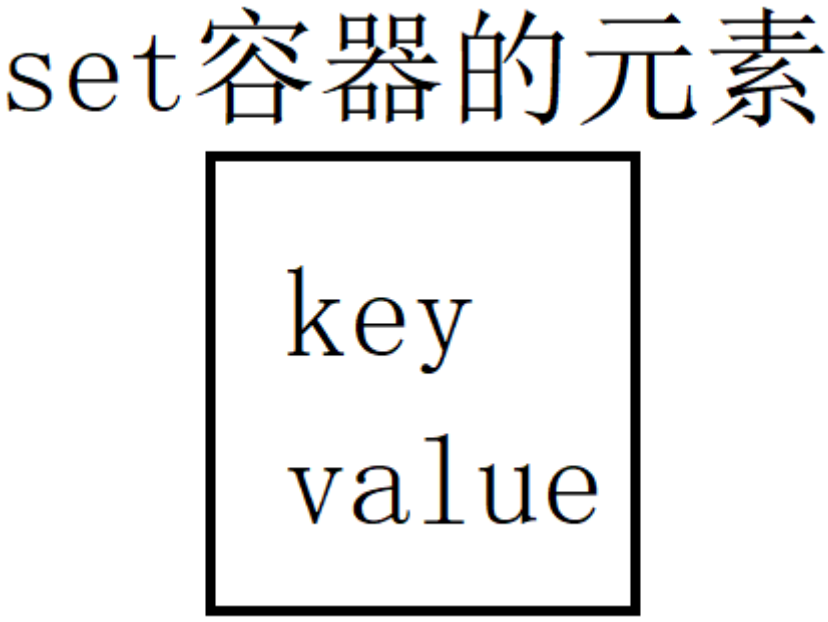
知识点7【set容器】2-1

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

```
/*
3.7.2.1 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
3.7.2.2 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
3.7.2.3 set大小操作
 size();//返回容器中元素的数目
 empty();//判断容器是否为空
 3.7.2.4 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
 */
 void test01()
 {
  set<int> s;
  //set容器自动根据键值排序
  s.insert(30);
  s.insert(10);
  s.insert(20);
  s.insert(50);
  s.insert(40);

  for_each(s.begin(),s.end(),[](int val){
  cout<<val<<" ";
  });//10 20 30 40 50
  cout<<endl;

  //set容器提供的是只读迭代器const_iterator
  //用户不可以修改set容器的元素
  set<int>::const_iterator it=s.begin();
  //*it = 100;//err
  cout<<"size = "<<s.size()<<endl;

  //删除起始位置的元素
  s.erase(s.begin());
  for_each(s.begin(),s.end(),[](int val){
  cout<<val<<" ";
  });//20 30 40 50
```

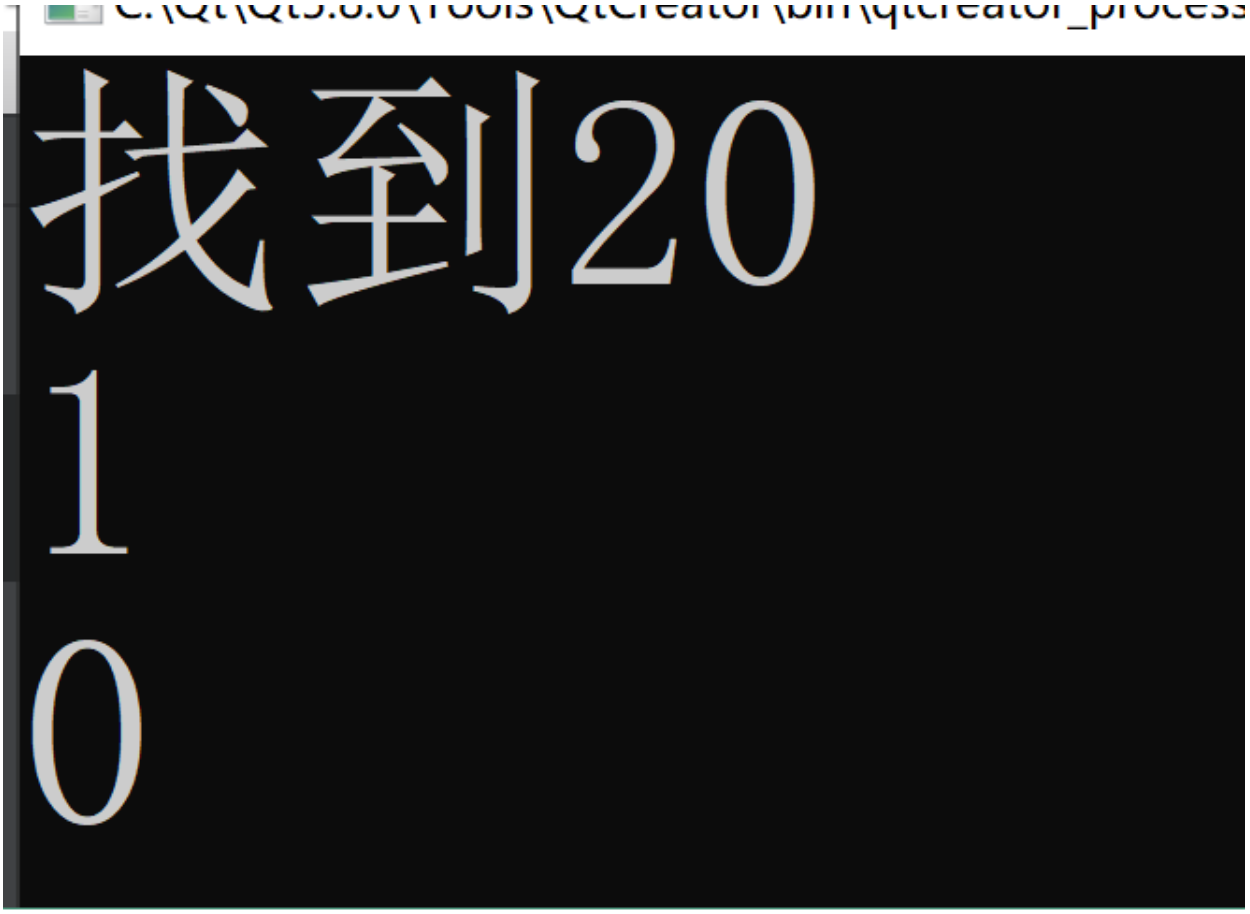
```
 cout<<endl;

 //根据元素删除
 s.erase(40);
 for_each(s.begin(),s.end(),[](int val){
 cout<<val<<" ";
 });//20 30 50
 cout<<endl;
}

```

2、set容器的查找

```
/*
3.7.2.5 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
*/
void test02()
{
 set<int> s;
 //set容器自动根据键值排序
 s.insert(30);
 s.insert(10);
 s.insert(20);
 s.insert(50);
 s.insert(40);

 //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();
 set<int>::const_iterator ret;
 ret = s.find(20);
 if(ret == s.end())
 {
 cout<<"没有找到"<<endl;
 }
 else
 {
 cout<<"找到"<<*ret<<endl;
 }
```

```
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 }
```

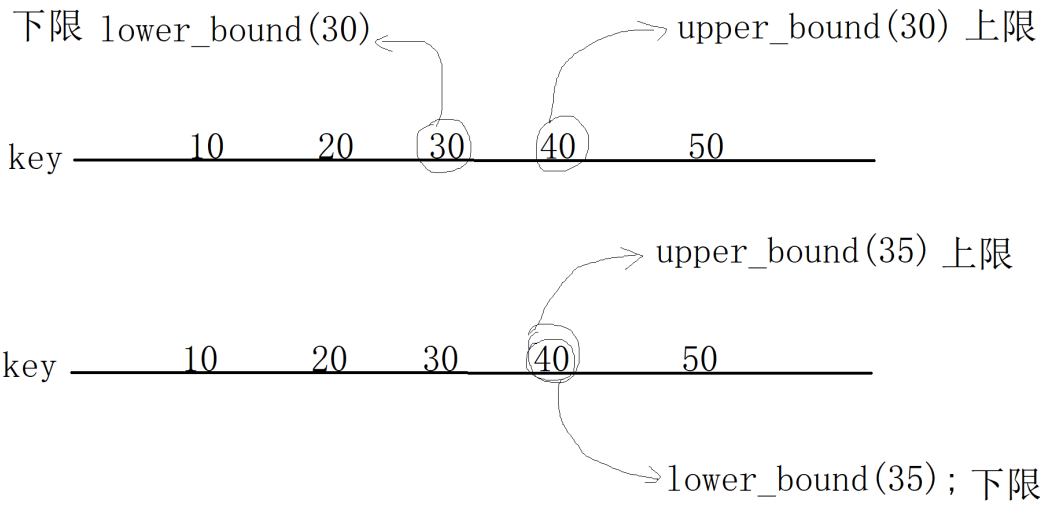
运行结果：



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。
```

```
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。
```

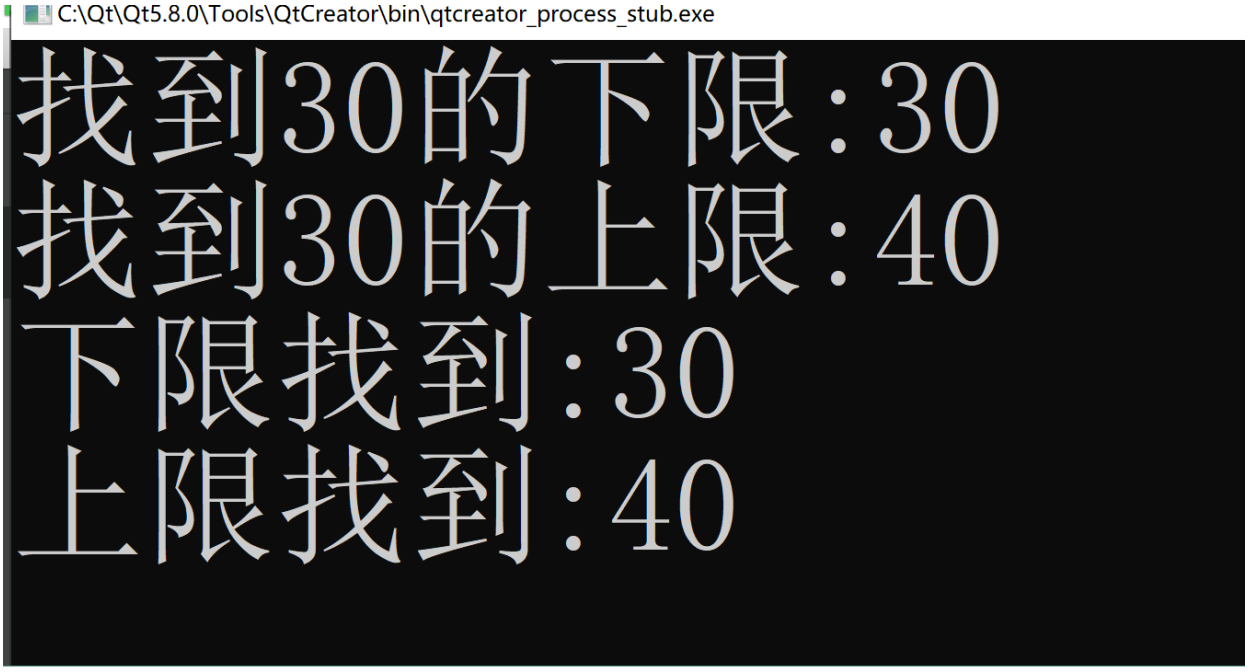


案例2-2

```
/*
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
*/
void test03()
{
 set<int> s;
 //set容器自动根据键值排序
  s.insert(30);
  s.insert(10);
  s.insert(20);
  s.insert(50);
  s.insert(40);

  //下限:lower_bound(keyElem);//返回第一个key>=keyElem元素的迭代器。
  set<int>::const_iterator lower_ret;
  lower_ret = s.lower_bound(30);
  if(lower_ret == s.end())
  {
  cout<<"没有找到30的下限"<<endl;
  }
  else
  {
```

```
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 }
```

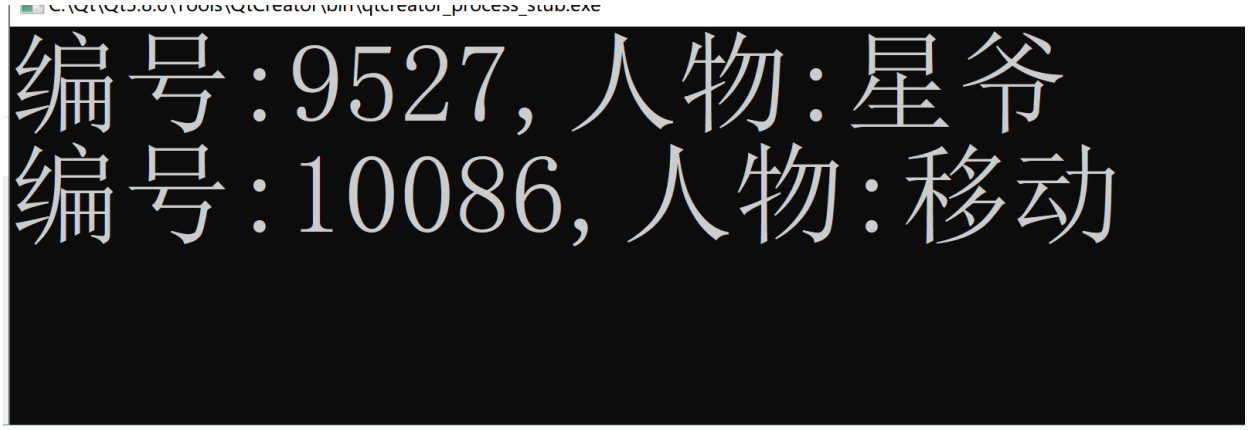
运行结果：



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:

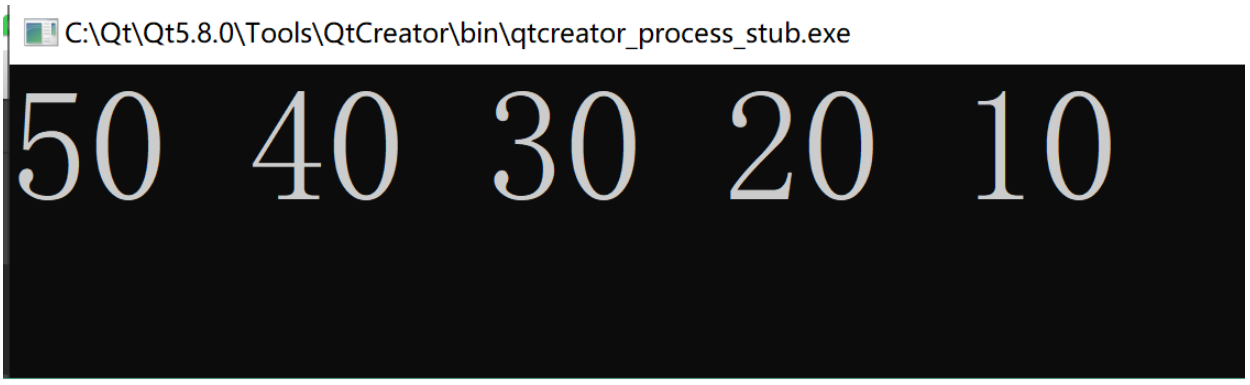


5、更改set容器的默认排序规则（推荐使用仿函数）

```
1  class MyGreater
```

```
{
public:
 bool operator()(int val1, int val2)
 {
 return val1>val2;
 }
};

void test05()
{
 //默认从小-->大排
 //改成从大-->小排
 //set<int,排序规则> s;
 set<int,MyGreater> s;

 s.insert(30);
 s.insert(10);
 s.insert(20);
 s.insert(50);
 s.insert(40);

 for_each(s.begin(),s.end(),[](int val){cout<<val<<" ";});
 cout<<endl;

}
int main(int argc, char *argv[])
{
 test05();
 return 0;
}
```

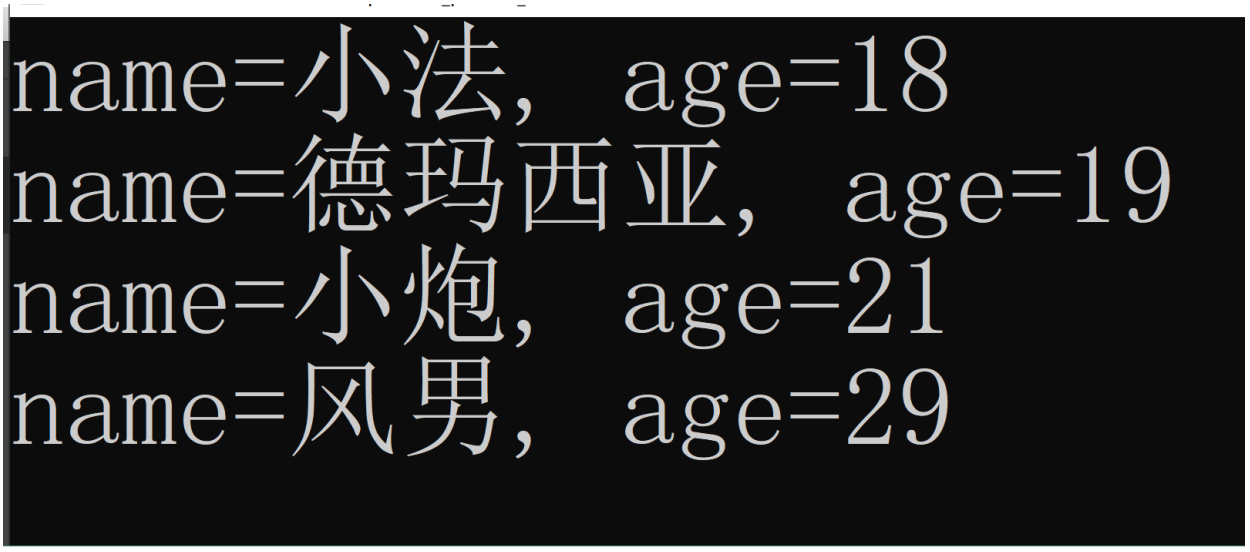
运行结果：



6、set容器存放自定义数据的时候 重载<运算符

```cpp
class Person
{
public:
 string name;
 int age;
 Person(string name, int age)
 {
 this->name =name;
 this->age = age;
 }

 //方法一：重载<运算符 小-->大
 bool operator<(const Person &ob)const
 {
 return this->age < ob.age;
 }
};
void test06()
{
 set<Person> s;
 s.insert(Person("德玛西亚", 19));
 s.insert(Person("小法", 18));
 s.insert(Person("小炮", 21));
 s.insert(Person("风男", 29));

 for_each(s.begin(),s.end(),[](Person val){
 cout<<"name="<<val.name<<", age="<<val.age<<endl;
 });
}
```

运行结果：

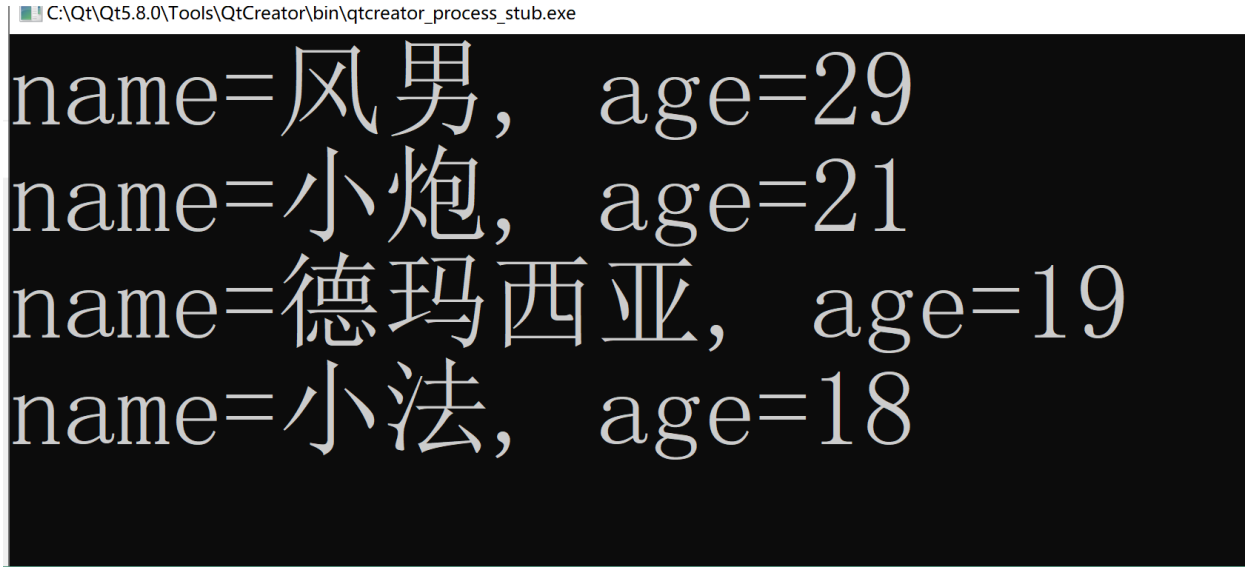


7、set容器存放自定义数据的时候 指定排序规则。

```
class Person
{
public:
 string name;
 int age;
 Person(string name, int age)
 {
 this->name =name;
 this->age = age;
  }
#if 0
 //方法一: 重载<运算符 小-->大
 bool operator<(const Person &ob) const
 {
 //单纯的告诉编译器 该函数为常函数不会修改成员数据
 //常对象 调用 常函数
 return this->age < ob.age;
  }
#endif
};

class myGreaterPerson
{
public:
 bool operator()(const Person &ob1, const Person &ob2)
 {
 return ob1.age > ob2.age;
```

```
 }
};
void test06()
{
 //方法二：指定排序规则
 set<Person, myGreaterPerson> s;
 s.insert(Person("德玛西亚", 19));
 s.insert(Person("小法", 18));
 s.insert(Person("小炮", 21));
 s.insert(Person("风男", 29));

 for_each(s.begin(),s.end(),[](Person val){
 cout<<"name="<<val.name<<", age="<<val.age<<endl;
 });
}
```

运行结果：



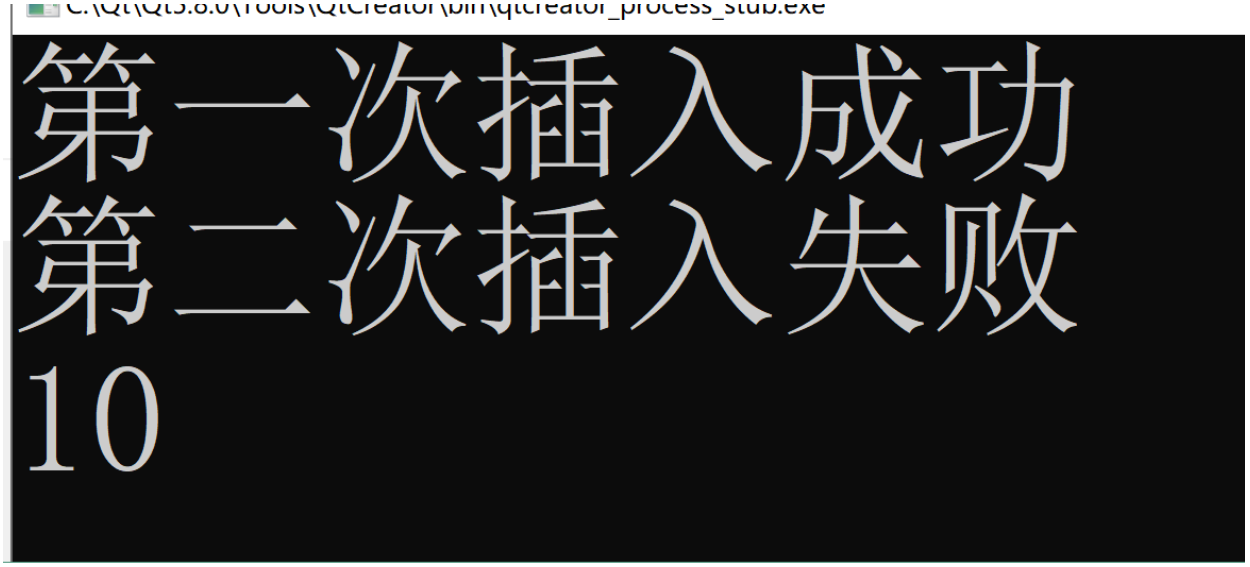
知识点9【multiset允许插入重复的键值】

1、set容器不允许插入相同的key值

```
void test07()
{
 set<int> s;
 pair<set<int>::const_iterator, bool> ret1;
 pair<set<int>::const_iterator, bool> ret2;

 ret1 = s.insert(10);
 if(ret1.second == true)
 {
```

```
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：



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：