C++ 中的 map 容器

内容提要：在标准模板库（STL）中提供了很多的容器,它们是存储对象的对象。本文主要介绍 STL 中的关联容器——map 容器，内容包括 map：：begin、map：：clear、map：：count、map：：empty、map：：end 等 27 种函数。本容器是以模板的形式设计的，能应用于广泛的数据类型。

关键字：begin、clear、count 、empty、end、find

引言：map 是一个容器，它用于储存数据并且能从一个数据集合中取出数据。它的数据组成包含两项，一个是它的数据值，一个是用于排序的关键字。其中关键字是惟一的，它用于将数据自动排序。而每个元素的数据值与关键字无关，可以直接改变。

正文：

1 map 容器

map 容器简介：

map 是一个容器，它用于储存数据并且能从一个数据集合中取出数据。它的数据组成包含两项，一个是它的数据值，一个是用于排序的关键字。其中关键字是惟一的，它用于将数据自动排序。而每个元素的数据值与关键字无关，可以直接改变。

需加载的头文件：

```
#include<map>
using namespace std;
```

模板原型：

```
template <
    class Key,
    class Type,
    class Traits = less<Key>,
    class Allocator=allocator<pair <const Key, Type> >
>
```

说明：

表 1 为 map 的参数说明。

表 1 map 的参数说明

参数	含义
Key	存储在 map 容器中的关键字的数据类型
Type	储存在 map 容器中的数据值的数据类型
Traits	它是一个能提供比较两个元素的关键字来决定它们在 map 容器中的相对位置。它是可选的，它的默认值是 less<key>
Allocator	它代表存储管理设备。它是可选的，它的默认值为 allocator<pair <const Key, Type> >

map 容器有以下的特点：

（1）它是一个相关联的容器，它的大小可以改变，它能根据关键字来提高读取数据能力。

（2）它提供一个双向的定位器来读写取数据。

（3）它已经根据关键字和一个比较函数来排好序。

（4）它的每一个元素的关键字都是惟一的。

（5）它是一个模板，它能提供一个一般且独立的数据类型。

成员变量：

map 的成员变量说明如表 2 所示。

表 2　map 的成员变量说明

成员变量	功能说明
Allocator_type	对象分配器
Const_iterator	提供一个双向的定位器，它能读取 map 中的一个常元
Const_pointer	它能提供到一个常元的指针
Const_reference	一个常元地址
Const_reverse_iterator	提供一个双向的定位器，使得能够在 map 容器中读取任意一个常值元素
Difference_type	它提供 map 容器中由定位器所指定的范围内的元素的个数
Iterator	提供一个双向入口定位器，使得能够在 map 中读取或者修改元素
key_compare	它是提供一个元素间的关键字的次序关系的函数
key_type	它描述每一个元素的关键字
mapped_type	它表示存储在 map 容器中的数据类型
pointer	提供一个指向 map 中的某元素的指针
reference	提供在 map 容器中的一个常元的地址
reverse_iterator	在反向的 map 容器中提供一个双向的入口定位器，使得能够读取或者修改元素
size_type	map 容器中元素个数
Value_type	它提供一个能根据关键字来比较两个元素的相对位置的函数

下面介绍 map 的成员函数。

1.1　map::begin

功能：

返回第一个元素的定位器（iterator）的地址。

语法：

const_iterator begin（）　const;

iterator begin（）;

说明：

当返回的第一个元素的地址值为一个常值定位器（_iterator），则 map 不会被修改。

当返回的第一个元素的地址值为一个定位器（iterator），则 map 可被修改。

函数返回值：

返回一个指向第一个元素的双向定位器地址。

示例：

```
/*************************************************
程序编号：1
程序功能说明：用 begin 来定位到 ctr 的开始位置，并打印出该元素。
*************************************************/
#include <map>
#include <iostream>
int main（）
{      using namespace std;
    map <int,char> ctr;
    map <int,char>::  iterator cp;
    ctr.insert（pair <int,char>（1,'a'））;
    ctr.insert（pair <int,char>（2,'b'））;
    cp=ctr.begin（）; //定位到 ctr 的开始位置
    cout<<"The first element is："<<cp->second<<endl;        //打印出第一个元素
```

```
    return 0;
}
```

运行结果：

运行结果如图 1 所示。

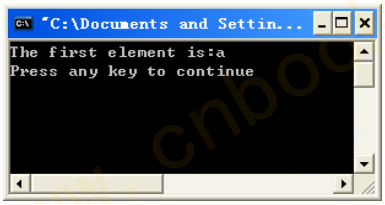

图 1　程序运行结果

1.2　map::clear

功能：

将一个 map 容器的全部元素删除。

语法：

void clear（）；

说明：

clear 会删除 map 容器的全部元素。

函数返回值：

无。

示例：

```
/*************************************************
程序编号：2
程序功能说明：先创建一个 map 容器，再用 clear 函数清空，最后打印是否为空的信息。
*************************************************/
#include <map>
#include <iostream>
int main（）
{
    using namespace std;
    map <int,char> ctr;
    ctr.insert（pair <int,char>（1,'a'））；
    ctr.insert（pair <int,char>（2,'b'））；
    ctr.insert（pair <int,char>（3,'c'））；
    ctr.clear（）；                    //清空 map 容器
    if（ctr.empty（））                //map 容器为空时
        cout<<"The container is empty"<<endl;
    else                               //map 容器为非空时
        cout<<"The container is not empty"<<endl;
    return 0;
 }
```

运行结果：

运行结果如图 2 所示。

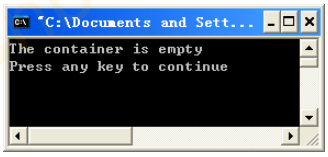

图 2　程序运行结果

1.3 map::count

功能：

返回对应某个关键字的元素的个数。

语法：

```
size_type count（
    const Key& _Key
）  const;
```

说明：

_Key 是要进行匹配的关键字的值。

该函数的返回值的范围是[low_bound（_Key）,upper_bound（_Key）]。

因为 map 容器的关键字是惟一的，故它只能取 0 或者 1。

函数返回值：

当 map 容器包含了关键字为_Key 的这个元素时，返回 1，否则返回 0。

示例：

```
/*************************************************
程序编号：3
程序功能说明：先创建一个 map 容器，再用 count 函数来求出关键字为 1 的元素的个数。
*************************************************/
#include <map>
#include <iostream>
int main（）
{
    using namespace std;
    map <int,char> ctr;
    ctr.insert（pair <int,char>（1,'a'））;
    ctr.insert（pair <int,char>（2,'b'））;
    ctr.insert（pair <int,char>（1,'c'））;
    int i;
    i=ctr.count（1）;        //求出关键字为 1 的元素的个数
    if（i==0）
        cout<<"There is no such key!"<<endl;
    else
        cout<<"The number of key is：  "<<i<<endl;
    return 0;
}
```

运行结果：

运行结果如图 3 所示。

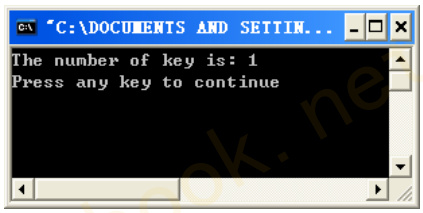

图 3　程序运行结果

1.4 map::empty

功能：

测试一个 map 容器是否为空。

语法：

bool empty（） const;

说明：

empty 函数用于测试一个 map 容器是否为空。

函数返回值：

当容器 map 为空时，返回 true，否则返回 false。

示例：

```
/************************************************
程序编号：4
程序功能说明：创建一个 map 容器，打印是否为空的信息。
************************************************/
#include <map>
#include <iostream>

int main（）
{
    using namespace std;
    map <int,char> ctr;
    if（ctr.empty（））                    //map 容器为空时
        cout<<"The container is empty"<<endl;
    else                                   //map 容器为非空时
        cout<<"The container is not empty"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 4 所示。

图 4　程序运行结果

1.5　map::end

功能：

返回最后一个元素后面的定位器（iterator）的地址。

语法：

const_iterator end（） const;

iterator end（）;

说明：

end 函数用于测试一个定位器是否已达到它的尾部。

函数返回值：

返回一个指向最后一个元素后面的双向定位器地址。当 map 容器为空时，结果没定义。

示例：

```
/************************************************
程序编号：5
```

程序功能说明：先初始化一个 map，再打印其中的内容，最后显示出最后一个元素的值。
**********/

```
#include <map>
#include <iostream>
using namespace std;
int print（map <int,int> c）      //用于打印一个 map
{
     map <int,int>:: const_iterator cp;
    for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
     cout<<cp->second<<" ";
    return 0;
}
int main（）
{
    map <int,int> ctr;
    map <int,int>:: const_iterator cp;
    int i;
    for（i=0;i<3;i++）ctr.insert（pair <int,int>（i,i））;
//给 ctr 赋值
    cout<<"The ctr is:  ";
    print（ctr）;                  //调用子程序来打印 ctr 的内容
    cout<<endl<<"The last element is:  ";
    cp=ctr.end（）;                //让 cp 指向最后一个元素的位置
    cp--;
    cout<<cp->second<<endl;  //显示最后一个元素的值
    return 0;
}
```

运行结果：

运行结果如图 5 所示。

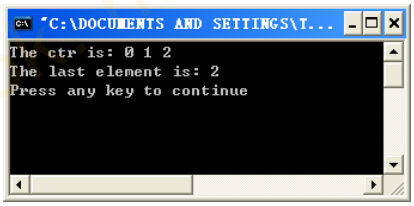

图 5　程序运行结果

1.6　map::equal_range

功能：

返回一对定位器，它们分别指向第一个大于或等于给定的关键字的元素和第一个比给定的关键字大的元素。

语法：

```
pair <const_iterator, const_iterator> equal_range （
    const Key& _Key
） const;
pair <iterator, iterator> equal_range （
```

const Key& _Key
） const;

说明：

_Key 是一个用于排序的关键字。

函数返回值：

返回一对定位器。

要从第一个定位器中取得数据，可用 pr.first。

从第二个定位器中取得数据，则用 pr.second。

示例：

```
/*************************************************
程序编号：6
程序功能说明：先初始化一个 map，再打印其中的内容，最后打印出关键字>=2 或>2 的元素。
*************************************************/
#include <map>
#include <iostream>
using namespace std;

int print_one_item（map <int,int>：：const_iterator cp）        //用于打印 map 的一个元素
{
     cout<<"（"<<cp->first<<" , "<<cp->second<<"） ";
     return 0;
}
int print（map <int,int> c）      //用于打印一个 map
{
      map <int,int>：：const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
      print_one_item（cp）;          //调用子程序来打印一个元素
        return 0;
}

int main（）
{
    map <int,int> ctr;
    pair <map <int,int>：：const_iterator, map <int,int>：：const_iterator> p;
    int i;
    for（i=0;i<=3;i++）ctr.insert（pair <int,int>（i,i））;
//给 ctr 赋值
    cout<<"The ctr is：  ";
    print（ctr）;                    //调用子程序来打印 ctr 的内容
    cout<<endl;
    p=ctr.equal_range（2）;
    if（p.first!=ctr.end（））
    {
        cout<<"The first element which key >= 2 is：  ";
        print_one_item（p.first）; //调用子程序来打印一项
```

```
        cout<<endl;
    }
    if（p.second!=ctr.end（））
    {
        cout<<"The first element which key > 2 is：  ";
        print_one_item（p.second）;
        cout<<endl;
    }
    return 0;
}
```

运行结果：

运行结果如图 6 所示。

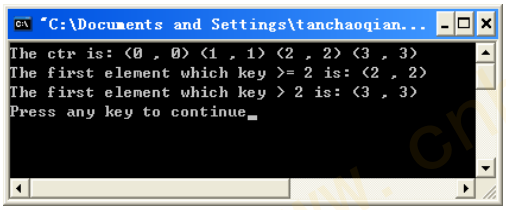

图 6　程序运行结果

1.7　map::erase

功能：

将一个或一定范围的元素删除。

语法：

```
iterator erase（
    iterator _Where
）;
iterator erase（
    iterator _First,
    iterator _Last
）;
size_type erase（
    const key_type& _Key
）;
```

说明：

表 3 为 erase 的参数说明。

表 3　erase 的参数说明

参数	含义
_Where	表示要删除的元素的位置
_First	第一个被删除的元素的位置
_Last	第一个不被删除的元素的位置
_Key	从 map 容器中删除的元素的关键字的值

因为没有重新分配空间，故只有对应于被删除元素的 iterator 和 reference 失效。

注意：它不会抛出任何的 exception。

函数返回值：

前两个函数返回一个指向第一个没被删除的元素的双向定位器，如果不存在这样的元素，则返回 map 容器的末尾。

第三个函数返回被删除的元素的个数。

示例：

```
/*************************************************
程序编号：7
程序功能说明：用 erase 函数将 ctr 的第二个元素删除。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5

int print_one_item（map <int,int>:: const_iterator cp）        //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}

int print（map <int,int> c）          //用于打印一个 map
{
     map <int,int>:: const_iterator cp;
     for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
  print_one_item（cp）;             //调用子程序来打印一个元素
    return 0;
}

int main（）
{
    map <int,int> ctr;
    map <int,int>:: iterator cp;

    int i;
    for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;       //下面先给 ctr 赋值
    cout<<"ctr is：";
    print（ctr）;                  //调用子程序，把 ctr 打印出来
    cout<<endl;

    cout<<"After erase the second element   ctr is：";
    cp=ctr.begin（）;
    cp++;
    ctr.erase（cp）;               //擦除 ctr 的 cp 位置的元素
    print（ctr）;                  //调用子程序，把 ctr 打印出来
    cout<<endl;
    return 0;
}
```

运行结果：

运行结果如图 7 所示。

图 7　程序运行结果

1.8　map::find

功能：

求出与给定的关键字相等的元素的定位器。

语法：

```
iterator find（
    const Key& _Key
）;
const_iterator find（
    const Key& _Key
）  const;
```

说明：

_Key 是要进行搜索的关键字的值。

该函数会返回一个指向关键字为_Key 的元素的定位器。

当返回的第一个元素的地址值为一个常值定位器（_iterator），则 map 可通过它不会被修改。

当返回的第一个元素的地址值为一个定位器（iterator），则 map 可通过它被修改。

函数返回值：

找到该元素时，返回一个指向关键字为_Key 的元素的定位器，否则返回一个指向 map 容器的结束的定位器。

示例：

```
/*************************************************
程序编号：8
程序功能说明：用将 key=2 的元素打印出来。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5
int print_one_item（map <int,int>:: const_iterator cp）        //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}
int print（map <int,int> c）      //用于打印一个 map 容器
{
     map <int,int>:: const_iterator cp;
     for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
  print_one_item（cp）;  //调用子程序来打印一个元素
```

```
        return 0;
}
int main（）
{
    map <int,int> ctr;
    map <int,int>:: iterator cp;
    int i;
    for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//下面先给 ctr 赋值
    cout<<"ctr is: ";
    print（ctr）;                    //调用子程序，把 ctr 打印出来
    cout<<endl;
    cp=ctr.find（2）;                //查找 key=2 的元素
    if（cp!=ctr.end（））            //找到时，打印出来
    {
        cout<<"The element whose key = 2 is: ";
        print_one_item（cp）;
        cout<<endl;
    }
    else                             //找不到时
        cout<<"There is no element whose key = 2"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 8 所示。

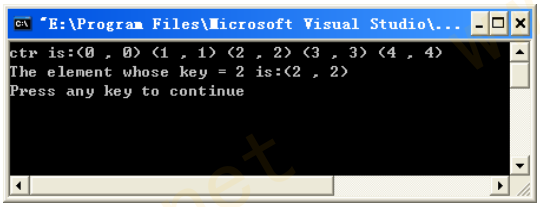

图 8　程序运行结果

1.9　map::get_allocator

功能：

返回一个构造该 map 容器的 allocator 的一个副本。

语法：

Allocator get_allocator（）　const;

说明：

容器 map 的 allocator 指明一个类的存储管理。默认的 allocator 能提供 STL 容器高效的运行。

函数返回值：

该容器的 allocator。

示例：

```
/************************************************
程序编号：9
程序功能说明：用 ctr 的 allocator 来创建 ctr2。
************************************************/
```

```
#include <map>
#include <iostream>
using namespace std;
int main（）
{
    map <int,char> ctr;
    ctr.insert（pair <int,char>（1,'a'））;
    map <int,char> ctr2（less<int>（）,ctr.get_allocator（））;
    cout<<"ctr2's size is：  "<<ctr2.size<<endl;
    return 0;
    }
```

运行结果：

运行结果如图 9 所示。

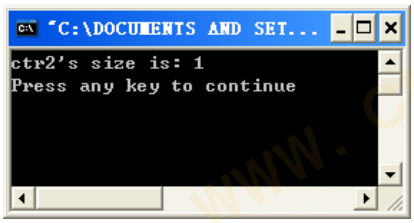

图 9　程序运行结果

1.10　map::insert

功能：

将一个元素或者一定数量的元素插入到 map 的特定位置中去。

语法：

```
pair <iterator, bool> insert（
    const value_type& _Val
）;
iterator insert（
    iterator _Where,
    const value_type& _Val
）;
template<class InputIterator>
    void insert（
        InputIterator _First,
        InputIterator _Last
    ）;
```

说明：

表 4 为参数的说明。

表 4　insert 的参数说明

参数	含义
_Where	第一个被插入到 map 的元素的位置
_Val	插入的参数的值
_First	第一个被插入的位置
_Last	第一个不被插入的位置

如果能在_Where 后面迅速地插入，那么只要很短的时间。

第三个成员函数将范围为[_First,_Last]中的元素插入。

函数返回值：

第一个函数返回一对值，当插入成功时，bool=true，当要插入的元素的关键字与已有的参数的值相同，则 bool=false，而 iterator 指向插入的位置或者已存在的元素的位置。

第二个函数返回指向插入位置的定位器。

示例：

```
/*************************************************
程序编号：10
程序功能说明：利用 insert 函数给 ctr 赋值。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5
int print_one_item（map <int,int>:: const_iterator cp）
//用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}
int print（map <int,int> c）
//用于打印一个 map
{
     map <int,int>:: const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
      print_one_item（cp）;
//调用子程序打印一个元素
       return 0;
}
int main（）
{
   map <int,int> ctr;
   map <int,int>:: iterator cp;
     int i;
   for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//下面先给 ctr 赋值
   cout<<"ctr is：";
   print（ctr）;
//调用子程序，把 ctr 打印出来
   cout<<endl;

       return 0;
}
```

运行结果：

运行结果如图 10 所示。

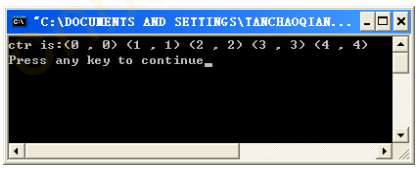

图 10　程序运行结果

1.11 map::key_comp

功能：

取得一个比较对象的副本以对 map 容器中的元素进行排序。

语法：

key_compare key_comp（） const;

说明：

存储对象定义了一个成员函数：

bool operator（const Key& _Left, const Key& _Right）;

当_Left 与_Right 在次序比较中不同时，返回 true，否则返回 false。

函数返回值：

取得一个对 map 容器中的元素进行排序的比较对象。

示例：

```
/*************************************************
程序编号：11
程序功能说明：先取得一个 key_compare 对象，再用此对象比较 1 和 2。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
int main（）
{
    map <int,int> ctr;
    map <int, int, less<int> >:: key_compare kc = ctr.key_comp（） ;
    if（kc（ 1, 2 ））
    {
        cout<<"kc（1,2） is true"<<endl;
    }
    else
    {
        cout<<"kc（1,2） is false"<<endl;
    }
        return 0;
 }
```

运行结果：

运行结果如图 11 所示。

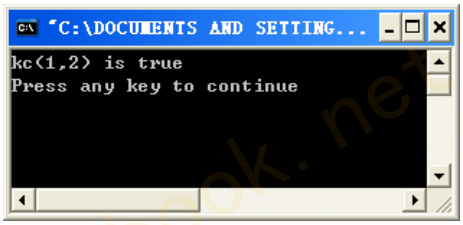

图 11 程序运行结果

1.12 map::lower_bound

功能：

求出指向第一个关键字的值是大于等于一个给定值的元素的定位器。

语法：

```
iterator lower_bound（
    const Key& _Key
）;
const_iterator lower_bound（
    const Key& _Key
）  const;
```

说明：

_Key 是一个用于排序的关键字。

当返回值为一个 const _iterator，则 map 容器不会被修改。

当返回值为一个 iterator，则 map 容器可被修改。

函数返回值：

返回一个指向第一个关键字的值是大于等于一个给定值的元素的定位器，或者返回指向 map 容器的结束的定位器。

示例：

```
/*************************************************
程序编号：12
程序功能说明：先初始化一个 map，再打印其中的内容，最后将关键字比 2 大的元素打印出来。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
int print_one_item（map <int,int>：：const_iterator cp）        //用于打印 map 的一个元素
{
     cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
     return 0;
}
int print（map <int,int> c）      //用于打印一个 map
{
      map <int,int>：：const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
 print_one_item（cp）;          //调用子程序来打印一个元素
        return 0;
}
int main（）
{
    map <int,int> ctr;
    map <int,int>：：const_iterator cp;
    int i;
    for（i=0;i<=3;i++）ctr.insert（pair <int,int>（i,i））;
//给 ctr 赋值
    cout<<"The ctr is：  ";
    print（ctr）;                //调用子程序来打印 ctr 的内容
    cout<<endl;
    cp=ctr.lower_bound（2）;
```

```
    if（cp!=ctr.end（））
    {
        cout<<"The first element which key >= 2 is：  ";
        print_one_item（cp）；      //调用子程序来打印一项
        cout<<endl;
    }
    return 0;
 }
```

运行结果：

运行结果如图 12 所示。

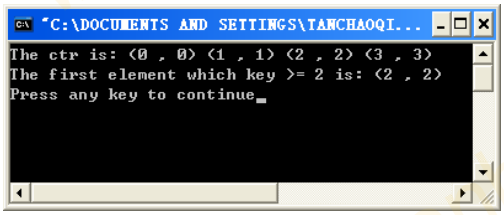

图 12　程序运行结果

1.13　map::map

功能：

map 的构造函数。

语法：

```
map（）；
explicit map（
    const Traits& _Comp
）；
explicit map（
    const Traits& _Comp,
    const Allocator& _Al
）；
map（
    const _map& _Right
）；
template<class InputIterator>
    map（
        InputIterator _First,
        InputIterator _Last
    ）；
template<class InputIterator>
    map（
        InputIterator _First,
        InputIterator _Last,
        const Traits& _Comp
    ）；
template<class InputIterator>
    map（
        InputIterator _First,
```

```
        InputIterator _Last,
        const Traits& _Comp,
        const Allocator& _Al
    );
```

说明：

表 5 为 map 函数的参数说明。

表 5　map 函数的参数说明

参数	含义
_Al	一个分配器类
_Comp	一个用于比较的函数，它的默认值为 hash_compare
_Right	一个 map 的拷贝
_First	被拷贝的第一个元素的位置
_Last	被拷贝的最后一个元素的位置

所有的构造函数都存储一个分配器和初始化 map 容器。

所有的构造函数都存储一个 Traits 类型的函数对象，它用于对 map 容器的元素进行排序。

头三个构造函数创建一个空的初始 map 容器。

第四个构造函数创建一个_Right 容器的副本。

后三个构造函数拷贝在范围_First~_Last 内的一个 map 容器。

函数返回值：

无。

示例：

```
/*************************************************
程序编号：13
程序功能说明：先定义一个 ctr1 的 map 容器，再构造一个以 ctr1 相同的容器。
*************************************************/
#include <map>
#include <iostream>
using namespace std;

int print_one_item（map <int,int>:: const_iterator cp）        //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"） ";
    return 0;
}

int print（map <int,int> c）     //用于打印一个 map
{
    map <int,int>:: const_iterator cp;
    for（cp=c.begin（）;cp!=c.end（）;cp++）
                                //让 cp 从 c 的开始到结束打印 cp 对应的值
print_one_item（cp）;           //调用子程序来打印一个元素
       return 0;
}

int main（）
{
```

```
    map <int,int> ctr1;       //创建一个有两个为字母为'a'的 map
     int i;
     for（i=0;i<=3;i++）ctr1.insert（pair <int,int>（i,i））;
//给 ctr1 赋值
map <int,int> ctr2（ctr1）;
                              //创建一个以 ctr1 相同的 map 容器
     cout<<"The ctr2 is：  ";
     print（ctr2）;           //打印 ctr2 的内容
     cout<<endl;
     return 0;
}
```

运行结果：

运行结果如图 13 所示。

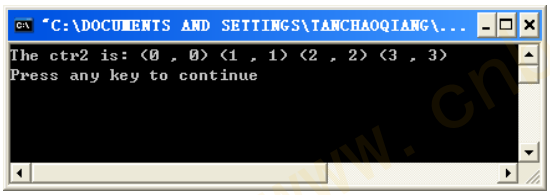

图 13　程序运行结果

1.14　map：：max_size

功能：

计算 map 容器的最大长度。

语法：

size_type max_size（）const;

说明：

max_size 会返回 map 容器的最长度。

函数返回值：

返回 map 容器可能的最大长度。

示例：

```
/*************************************************
程序编号：14
程序功能说明：求出一个 map 容器的可能最大的长度。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5

int main（）
{
     map <int,int> ctr;
     map <int,int>：：size_type st;
     int i;
     for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//先给 ctr 赋值
```

```
    cout<<"the max_size of ctr is：";
    st=ctr.max_size（）;
    cout<<st<<endl;
    return 0;
}
```

运行结果：

运行结果如图 14 所示。

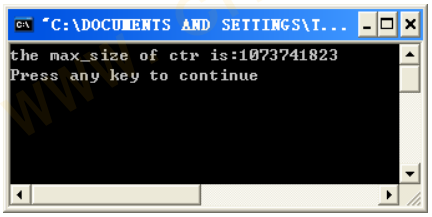

图 14　程序运行结果

1.15　map：：operator[]

功能：

将一个给定的值插入到 map 容器中去。

语法：

```
Type& operator[]（
    const Key& _Key
）;
```

说明：

_Key 是被插入的元素的关键字的值。

注意：当 map 容器中找不到_Key 的值时，则插入到 map 容器中，且值是一个默认值。

函数返回值：

返回一个_Key 位置的元素的地址（reference）。

示例：

```
/*************************************************
程序编号：15
程序功能说明：用 operator[]在 map 容器中插入两个元素。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
int print_one_item（map <int,int>：：const_iterator cp）          //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}
int print（map <int,int> c）      //用于打印一个 map
{
     map <int,int>：：const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
    print_one_item（cp）;      //调用子程序来打印一个元素
```

```
        return 0;
}
int main（）
{
    map <int,int> ctr;
    int i;
    for（i=0;i<=3;i++）ctr.insert（pair <int,int>（i,i））;
//给 ctr 赋值
    cout<<"The ctr is：  ";
    print（ctr）;                    //调用子程序来打印 ctr 的内容
    cout<<endl;
    ctr[1]=10;
    ctr[7];
    cout<<"The ctr now is：  ";
    print（ctr）;                    //调用子程序来打印 ctr 的内容
    cout<<endl;
    return 0;
}
```

运行结果：

运行结果如图 15 所示。

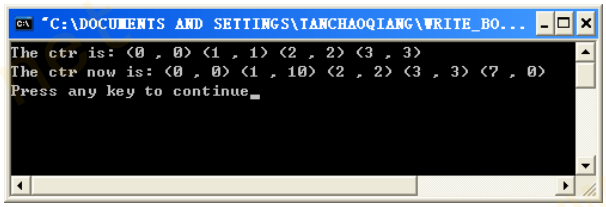

图 15　程序运行结果

1.16　map::operator!=

功能：

测试该 map 容器的左边与右边是否相同。

语法：

```
bool operator!=（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）;
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器相等，当且仅当它们的元素个数相等且同一个位置上的值相等。

函数返回值：

当_Left 和_Right 不同时，返回 True，否则返回 False。

示例：

```
/************************************************
程序编号：16
程序功能说明：比较两个 map 容器是否相等。
************************************************/
#include <map>
```

```
#include <iostream>
int main（）
{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
    if（ctr1!=ctr2）          //当 ctr1 与 ct2 不同时
        cout<<"They are not equal"<<endl;
    else                      //当 ctr1 与 ctr2 相同时
        cout<<"They are equal"<<endl;

    return 0;
}
```

运行结果：

运行结果如图 16 所示。

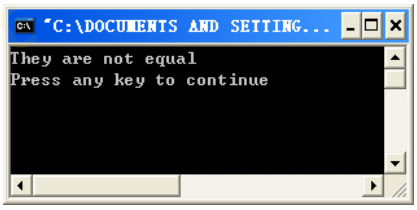

图 16　程序运行结果

1.17　map::operator<

功能：

测试该 map 容器的左边是否小于右边。

语法：

```
bool operator==（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）;
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器的大小比较是基于第一个不相同的元素的大小比较。

函数返回值：

当_Left<_Right 时，返回 True，否则返回 False。

示例：

```
/************************************************
程序编号：17
程序功能说明：比较 ctr1 与 ctr2 的大小。
************************************************/
#include <map>
```

```
#include <iostream>
int main（）
{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）        //下面给 ctr1 和 ctr2 赋值
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
    if（ctr1<ctr2）
        cout<<"ctr1<ctr2"<<endl;
    else
        cout<<"ctr1>=ctr2"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 17 所示。

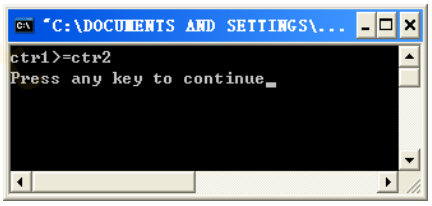

图 17　程序运行结果

1.18　map::operator<=

功能：

测试左边的 map 容器是否小于或等于右边。

语法：

```
bool operator<（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）;
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器的大小比较是基于第一个不相同的元素的大小比较。

函数返回值：

当_Left<=_Right 时，返回 True,否则返回 False。

示例：

```
/************************************************
程序编号：18
程序功能说明：比较 ctr1 与 ctr2 的大小。
************************************************/
#include <map>
#include <iostream>
```

```
int main（）

{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
    if（ctr1<=ctr2）
        cout<<"ctr1<=ctr2"<<endl;
    else
        cout<<"ctr1>ctr2"<<endl;
    return 0;
 }
```

运行结果：

运行结果如图 18 所示。

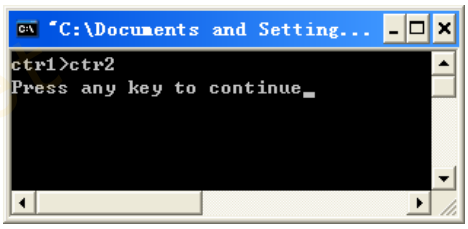

图 18　程序运行结果

1.19　map::operator==

功能：

测试左边的 map 容器与右边是否相同。

语法：

```
bool operator==（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）;
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器相等，当且仅当它们的元素个数相等且同一个位置上的值相等。

函数返回值：

当_Left 和_Right 相同时，返回 True，否则返回 False。

示例：

```
/************************************************
程序编号：19
程序功能说明：比较两个数是否相等。
************************************************/
#include <map>
```

```
#include <iostream>

int main（）
{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
  if（ctr1==ctr2）
//当 ctr1 与 ctr2 相同时
        cout<<"They are equal"<<endl;
  else
//当 ctr1 与 ct2 不同时
        cout<<"They are not equal"<<endl;
       return 0;
    }
```

运行结果：

运行结果如图 19 所示。

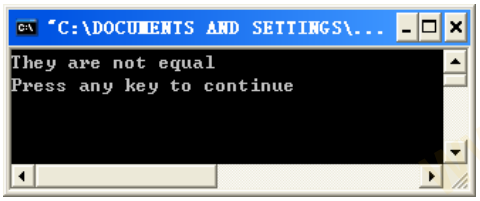

图 19　程序运行结果

1.20　map::operator>

功能：

测试左边的 map 容器是否大于右边。

语法：

```
bool operator>（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器的大小比较是基于第一个不相同的元素的大小比较。

函数返回值：

当_Left>_Right 时，返回 True，否则返回 False。

示例：

```
/************************************************
程序编号：20
程序功能说明：比较 ctr1 与 ctr2 的大小。
```

```
**************************************************/
#include <map>
#include <iostream>

int main（）
{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
    if（ctr1>ctr2）
        cout<<"ctr1>ctr2"<<endl;
    else
        cout<<"ctr1<=ctr2"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 20 所示。

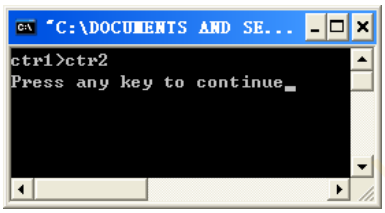

图 20　程序运行结果

1.21　map::operator>=

功能：

测试左边的 map 容器是否大于或等于右边。

语法：

```
bool operator>=（
    const map <Key, Type, Traits, Allocator>& _Left,
    const map <Key, Type, Traits, Allocator>& _Right
）;
```

说明：

_Left 和_Right 是待比较的两个 map 容器。

两个 map 容器的大小比较是基于第一个不相同的元素的大小比较。

函数返回值：

当_Left>=_Right 时，返回 True，否则返回 False。

示例：

```
/**************************************************
```

程序编号：21

程序功能说明：比较 ctr1 与 ctr2 的大小。

```
**************************************************/
#include <map>
#include <iostream>

int main（）
{
    using namespace std;
    map <int,char> ctr1,ctr2;
    int i;
    for（i=0;i<3;i++）
    {
        ctr1.insert（pair <int,char>（i,'a'+i））;
        ctr2.insert（pair <int,char>（i,'A'+i））;
    }
    if（ctr1>=ctr2）
        cout<<"ctr1>=ctr2"<<endl;
    else
        cout<<"ctr1<ctr2"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 21 所示。

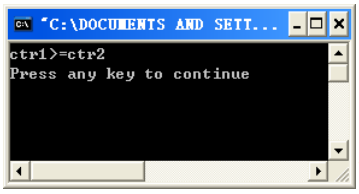

图 21　程序运行结果

1.22　map::rbegin

功能：

返回一个指向反向 map 容器的第一个元素的定位器。

语法：

```
const_reverse_iterator rbegin（）const;
reverse_iterator rbegin（）;
```

说明：

rbegin 与反向 map 容器一起使用，它的作用与 map 容器中的 begin 一样。

当返回值为一个 const _reverse _iterator，则 map 容器不会被修改。

当返回值为一个 reverse _iterator，则 map 容器可被修改。

函数返回值：

返回一个指向反向 map 容器的第一个元素的反向双向定位器。

示例：

```
/**************************************************
程序编号：22
程序功能说明：打印出正向和反向的 map 容器。
**************************************************/
```

```
#include <map>
#include <iostream>
using namespace std;
#define len 5

int print_one_item（map <int,int>:: const_iterator cp）
//用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}

int print（map <int,int> c）
//用于打印一个 map
{
     map <int,int>:: const_iterator cp;
     for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
print_one_item（cp）;
//调用子程序来打印一个元素
       return 0;
}

int main（）
{
    map <int,int> ctr;
    map <int,int>:: iterator cp;
    map <int,int>:: reverse_iterator rcp;
    int i;
    for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//下面先给 ctr 赋值
    cout<<"ctr is：";
    print（ctr）;
//调用子程序，把 ctr 打印出来
    cout<<endl;
        cout<<"Its reverse is：";
    for（rcp=ctr.rbegin（）;rcp!=ctr.rend（）;rcp++）
//打印出反向 map 容器
         cout<<"（"<<rcp->first<<" , "<<rcp->second<<"）  ";
    cout<<endl;
    return 0;
}
```

运行结果：

运行结果如图 22 所示。

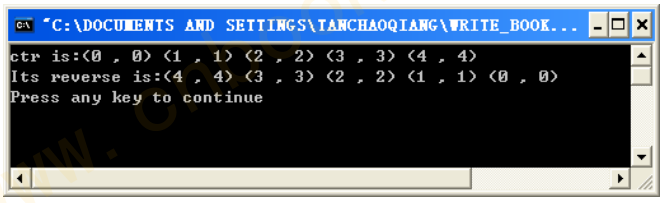

图 22　程序运行结果

1.23 map::rend

功能：

返回一个指向反向 map 容器的最后元素后面的定位器。

语法：

const_reverse_iterator rend（） const;

reverse_iterator rend（）;

说明：

rend 与反向 map 容器一起使用，它的作用与 map 容器中的 end 一样。

当返回值为一个 const _reverse _iterator，则 map 不会被修改。

当返回值为一个 reverse _iterator，则 map 可被修改。

函数返回值：

返回一个指向反向 map 容器中的最后一个元素后面的反向双向定位器。

示例：

```
/*************************************************
程序编号：23
程序功能说明：打印出正向和反向的 map 容器。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5

int print_one_item（map <int,int>:: const_iterator cp）        //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"） ";
    return 0;
}
int print（map <int,int> c）     //用于打印一个 map
{
     map <int,int>:: const_iterator cp;
     for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
 print_one_item（cp）;          //调用子程序来打印一个元素
       return 0;
}
int main（）
{
    map <int,int> ctr;
    map <int,int>:: iterator cp;
    map <int,int>:: reverse_iterator rcp;
    int i;
for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//下面先给 ctr 赋值
    cout<<"ctr is：";
    print（ctr）;                  //调用子程序，把 ctr 打印出来
    cout<<endl;
```

```
        cout<<"Its reverse is： ";
    for（rcp=ctr.rbegin（）;rcp!=ctr.rend（）;rcp++）          //打印出反向 map 容器
        cout<<"（"<<rcp->first<<" , "<<rcp->second<<"）  ";
    cout<<endl;
    return 0;
 }
```

运行结果：

运行结果如图 23 所示。

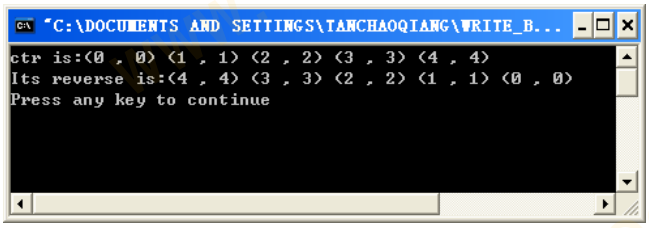

图 23　程序运行结果

1.24　map：：size

功能：

计算 map 容器的大小。

语法：

size_type size（）　const;

说明：

size 函数会计算出 map 容器的长度。

函数返回值：

当前 map 容器的长度。

示例：

```
/*************************************************
程序编号：24
程序功能说明：求出 map 容器的长度。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
#define len 5

int main（）
{
    map <int,int> ctr;
    int i;
 for（i=0;i<len;i++）ctr.insert（pair <int,int>（i,i））;
//下面先给 ctr 赋值
    cout<<"The current map's length is：  ";
    cout<<ctr.size（）<<endl;
    return 0;
}
```

运行结果：

运行结果如图 24 所示。

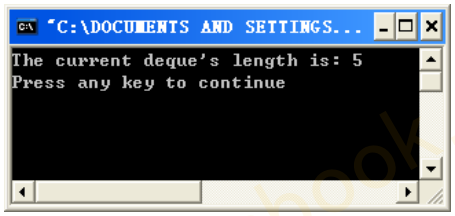

图 24　程序运行结果

1.25　map::swap

功能：

交换两个 map 容器的元素。

语法：

```
void swap（
    map& _Right
）;
```

说明：

_Right 是与目标容器交换元素的 map 容器。

函数返回值：

无。

示例：

```
/************************************************
程序编号：25
程序功能说明：让 ctr1 与 ctr2 交换。
************************************************/
#include <map>
#include <iostream>
using namespace std;
int print_one_item（map <int,int>:: const_iterator cp）        //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}
int print（map <int,int> c）     //用于打印一个 map
{
     map <int,int>:: const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
print_one_item（cp）;          //调用子程序来打印一个元素
       return 0;
}
int main（）
{
    map <int,int> ctr1, ctr2;
    map <int,int>:: const_iterator cp;
        int i;
```

```
    for（i=0;i<3;i++）          //下面先给 ctr1 和 ctr2 赋值
    {
        ctr1.insert（pair <int,int>（i,i））;
        ctr2.insert（pair <int,int>（i,i+10））;
    }

    cout<<"Before exchange with ctr2 the ctr1 is：";
    print（ctr1）;              //调用子程序，把 ctr2 打印出来
    cout<<endl;

    cout<<"After exchange with ctr2 the ctr1 is：";
    ctr1.swap（ctr2）;          //让 ctr1 的内容与 ctr2 交换
    print（ctr1）;              //调用子程序，把 ctr1 打印出来
    cout<<endl;
    return 0;
}
```

运行结果：

运行结果如图 25 所示。

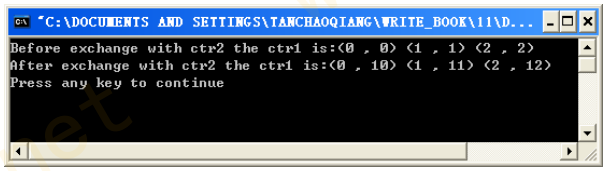

图 25　程序运行结果

1.26　map::upper_bound

功能：

求出指向第一个关键字的值是大于一个给定值的元素的定位器。

语法：

```
iterator upper_bound（
    const Key& _Key
）;
const_iterator upper_bound（
    const Key& _Key
）  const;
```

说明：

_Key 是一个用于排序的关键字。

当返回值为一个 const _iterator，则 map 容器不会被修改。

当返回值为一个 iterator，则 map 容器可被修改。

函数返回值：

返回一个指向第一个关键字的值是大于一个给定值的元素的定位器，或者返回指向 map 容器的结束的定位器。

示例：

```
/*************************************************
```

程序编号：26

程序功能说明：先初始化一个 map，再打印其中的内容，最后将关键字比 2 大的第一个元素打印出来。

```
**************************************************/
#include <map>
#include <iostream>
using namespace std;
int print_one_item（map <int,int>:: const_iterator cp）  //用于打印 map 的一个元素
{
    cout<<"（"<<cp->first<<" , "<<cp->second<<"）  ";
    return 0;
}
int print（map <int,int> c）
//用于打印一个 map
{
     map <int,int>:: const_iterator cp;
for（cp=c.begin（）;cp!=c.end（）;cp++）
//让 cp 从 c 的开始到结束打印 cp 对应的值
  print_one_item（cp）;
//调用子程序来打印一个元素
       return 0;
}
int main（）
{
    map <int,int> ctr;
    map <int,int>:: const_iterator cp;
    int i;
    for（i=0;i<=3;i++）ctr.insert（pair <int,int>（i,i））;
//给 ctr 赋值
    cout<<"The ctr is:  ";
    print（ctr）;
//调用子程序来打印 ctr 的内容
    cout<<endl;
    cp=ctr.upper_bound（2）;
    if（cp!=ctr.end（））
    {
        cout<<"The first element which key > 2 is:  ";
        print_one_item（cp）;
//调用子程序来打印一项
        cout<<endl;
    }
    return 0;
}
```

运行结果：

运行结果如图 26 所示。

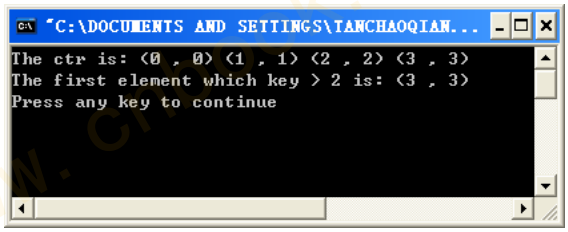

图 26　程序运行结果

1.27 map::value_comp

功能：

返回一个能确定元素的次序的函数。

语法：

value_compare value_comp（） const;

说明：

对于容器 m，当 e1（k1,d1）和 e2（k2,d2）是它的两个元素，则：

m.value_comp（e1,e2）=m..key_com（k1,k2）

函数返回值：

返回一个能确定元素的次序的函数。

示例：

```
/*************************************************
程序编号：27
程序功能说明：先取得一个 key_value 函数，再用此函数来比较元素 1 和元素 2。
*************************************************/
#include <map>
#include <iostream>
using namespace std;
int main（）
{
    map <int,int> ctr;
    map <int, int, less<int> >:: value_compare vc = ctr.value_comp（） ;
    map <int,int>:: iterator cp1,cp2,cp3;
    pair< map<int,int>:: iterator, bool > cpr1, cpr2;
    int i;
    for（i=0;i<3;i++）
//给 ctr 赋值
        ctr.insert（pair <int,int>（i,i））;
    cpr1.first=ctr.begin（）;
    cpr2.first=ctr.begin（）;
    cpr2.first++;
    if（vc（*cpr1.first,*cpr2.first））
        cout<<"element 1 is precedes element 2"<<endl;
    else
        cout<<"element 1 does not precede element 2"<<endl;
    return 0;
}
```

运行结果：

运行结果如图 27 所示。

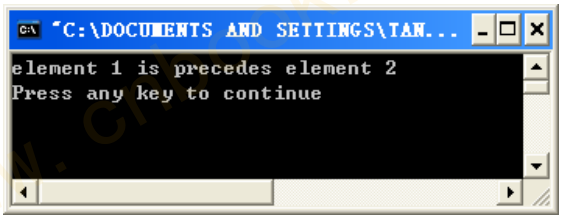

图 27 程序运行结果