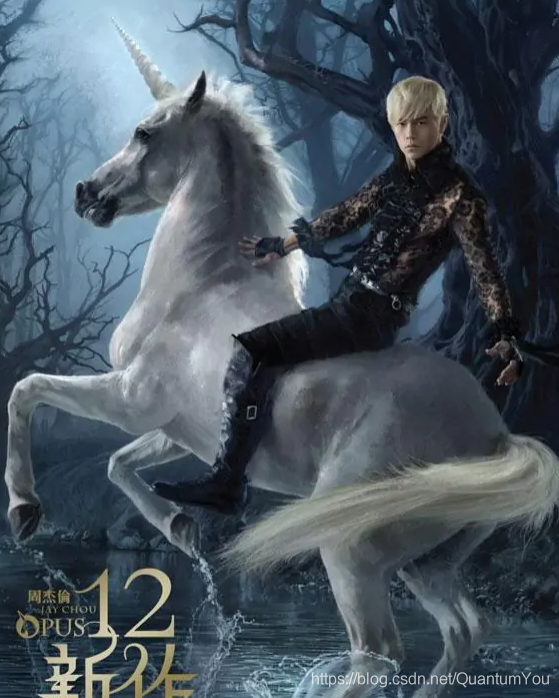
* 没有退路可言、不要给自己找接口、以及**顾虑太多**
* 缺乏目的、目标、方法、效率再勤奋也只是一种时间的牺牲



中古世纪的城市里 我想就走到这

海鸥不再眷恋大海 可以飞更远

远方传来风笛 我只在意有你的消息

城堡为爱守着秘密 而我为你守着回忆  
 —— 《明明就》 周杰伦

@[toc]

# list容器

## list基本概念

**功能** ： 将数据进行链式存储

**链表**（list）： 是一种物理存储单元上非连续的存储结构，数据元素的逻辑顺序是通过链表中的指针链接实现的

* 链表的组成：链表由一系列**结点**组成
* 结点的组成：一个是存储数据元素的**数据域**，另一个是存储下一个结点地址的**指针域**
* STL中的链表是一个双向循环链表

由于链表的存储方式并不是连续的内存空间，因此链表list中的迭代器只支持前移和后移，属于**双向迭代器**

**list的优点**：

* 采用动态存储分配，不会造成内存浪费和溢出
* 链表执行插入和删除操作十分方便，修改指针即可，不需要移动大量元素

**list的缺点**：：

* 链表灵活，但是空间(指针域) 和 时间（遍历）额外耗费较大
* List有一个重要的性质，插入操作和删除操作都不会造成原有list迭代器的失效，这在vector是不成立的。

总结：STL中**List和vector是两个最常被使用的容器**，各有优缺点

## list构造函数

**功能描述：**

* 创建list容器

**函数原型：**

* list<T> lst; //list采用采用模板类实现,对象的默认构造形式：
* list(beg,end); //构造函数将[beg, end)区间中的元素拷贝给本身。
* list(n,elem); //构造函数将n个elem拷贝给本身。
* list(const list &lst); //拷贝构造函数。

**示例：**

#include <list>  
  
void printList(const list<int>& L) {  
  
 for (list<int>::const\_iterator it = L.begin(); it != L.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 list<int>L1;  
 L1.push\_back(10);  
 L1.push\_back(20);  
 L1.push\_back(30);  
 L1.push\_back(40);  
  
 printList(L1);  
  
 list<int>L2(L1.begin(),L1.end());  
 printList(L2);  
  
 list<int>L3(L2);  
 printList(L3);  
  
 list<int>L4(10, 1000);  
 printList(L4);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：list构造方式同其他几个STL常用容器，熟练掌握即可

## list 赋值和交换

**功能描述：**

* 给list容器进行赋值，以及交换list容器

**函数原型：**

* assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。
* assign(n, elem); //将n个elem拷贝赋值给本身。
* list& operator=(const list &lst); //重载等号操作符
* swap(lst); //将lst与本身的元素互换。

**示例：**

#include <list>  
  
void printList(const list<int>& L) {  
  
 for (list<int>::const\_iterator it = L.begin(); it != L.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//赋值和交换  
void test01()  
{  
 list<int>L1;  
 L1.push\_back(10);  
 L1.push\_back(20);  
 L1.push\_back(30);  
 L1.push\_back(40);  
 printList(L1);  
  
 //赋值  
 list<int>L2;  
 L2 = L1;  
 printList(L2);  
  
 list<int>L3;  
 L3.assign(L2.begin(), L2.end());  
 printList(L3);  
  
 list<int>L4;  
 L4.assign(10, 100);  
 printList(L4);  
  
}  
  
//交换  
void test02()  
{  
  
 list<int>L1;  
 L1.push\_back(10);  
 L1.push\_back(20);  
 L1.push\_back(30);  
 L1.push\_back(40);  
  
 list<int>L2;  
 L2.assign(10, 100);  
  
 cout << "交换前： " << endl;  
 printList(L1);  
 printList(L2);  
  
 cout << endl;  
  
 L1.swap(L2);  
  
 cout << "交换后： " << endl;  
 printList(L1);  
 printList(L2);  
  
}  
  
int main() {  
  
 //test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：list赋值和交换操作能够灵活运用即可

## list 大小操作

**功能描述：**

* 对list容器的大小进行操作

**函数原型：**

* size(); //返回容器中元素的个数
* empty(); //判断容器是否为空
* resize(num); //重新指定容器的长度为num，若容器变长，则以默认值填充新位置。
* //如果容器变短，则末尾超出容器长度的元素被删除。
* resize(num, elem); //重新指定容器的长度为num，若容器变长，则以elem值填充新位置。
* //如果容器变短，则末尾超出容器长度的元素被删除。

**示例：**

#include <list>  
  
void printList(const list<int>& L) {  
  
 for (list<int>::const\_iterator it = L.begin(); it != L.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//大小操作  
void test01()  
{  
 list<int>L1;  
 L1.push\_back(10);  
 L1.push\_back(20);  
 L1.push\_back(30);  
 L1.push\_back(40);  
  
 if (L1.empty())  
 {  
 cout << "L1为空" << endl;  
 }  
 else  
 {  
 cout << "L1不为空" << endl;  
 cout << "L1的大小为： " << L1.size() << endl;  
 }  
  
 //重新指定大小  
 L1.resize(10);  
 printList(L1);  
  
 L1.resize(2);  
 printList(L1);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 判断是否为空 --- empty
* 返回元素个数 --- size
* 重新指定个数 --- resize

## list 插入和删除

**功能描述：**

* 对list容器进行数据的插入和删除

**函数原型：**

* push\_back(elem);//在容器尾部加入一个元素
* pop\_back();//删除容器中最后一个元素
* push\_front(elem);//在容器开头插入一个元素
* pop\_front();//从容器开头移除第一个元素
* insert(pos,elem);//在pos位置插elem元素的拷贝，返回新数据的位置。
* insert(pos,n,elem);//在pos位置插入n个elem数据，无返回值。
* insert(pos,beg,end);//在pos位置插入[beg,end)区间的数据，无返回值。
* clear();//移除容器的所有数据
* erase(beg,end);//删除[beg,end)区间的数据，返回下一个数据的位置。
* erase(pos);//删除pos位置的数据，返回下一个数据的位置。
* remove(elem);//删除容器中所有与elem值匹配的元素。

**示例：**

#include <list>  
  
void printList(const list<int>& L) {  
  
 for (list<int>::const\_iterator it = L.begin(); it != L.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//插入和删除  
void test01()  
{  
 list<int> L;  
 //尾插  
 L.push\_back(10);  
 L.push\_back(20);  
 L.push\_back(30);  
 //头插  
 L.push\_front(100);  
 L.push\_front(200);  
 L.push\_front(300);  
  
 printList(L);  
  
 //尾删  
 L.pop\_back();  
 printList(L);  
  
 //头删  
 L.pop\_front();  
 printList(L);  
  
 //插入  
 list<int>::iterator it = L.begin();  
 L.insert(++it, 1000);  
 printList(L);  
  
 //删除  
 it = L.begin();  
 L.erase(++it);  
 printList(L);  
  
 //移除  
 L.push\_back(10000);  
 L.push\_back(10000);  
 L.push\_back(10000);  
 printList(L);  
 L.remove(10000);  
 printList(L);  
   
 //清空  
 L.clear();  
 printList(L);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 尾插 --- push\_back
* 尾删 --- pop\_back
* 头插 --- push\_front
* 头删 --- pop\_front
* 插入 --- insert （参数为迭代器）
* 删除 --- erase (参数为迭代器)
* 移除 --- remove
* 清空 --- clear

## list 数据存取

**功能描述：**

* 对list容器中数据进行存取

**函数原型：**

* front(); //返回第一个元素。
* back(); //返回最后一个元素。

**示例：**

#include <list>  
  
//数据存取  
void test01()  
{  
 list<int>L1;  
 L1.push\_back(10);  
 L1.push\_back(20);  
 L1.push\_back(30);  
 L1.push\_back(40);  
  
   
 //cout << L1.at(0) << endl;//错误 不支持at访问数据  
 //cout << L1[0] << endl; //错误 不支持[]方式访问数据  
 cout << "第一个元素为： " << L1.front() << endl;  
 cout << "最后一个元素为： " << L1.back() << endl;  
  
 //list容器的迭代器是双向迭代器，不支持随机访问  
 list<int>::iterator it = L1.begin();  
 //it = it + 1;//错误，不可以跳跃访问，即使是+1  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* list容器中不可以通过[]或者at方式访问数据
* 返回第一个元素 --- front
* 返回最后一个元素 --- back

**注意**：

* List 的存取没有 at() 以及 [] ，原因在于 list 不是顺序存取
* **迭代器**是不支持随机访问的

## list 反转和排序

**功能描述：**

* 将容器中的元素反转，以及将容器中的数据进行排序

**函数原型：**

* reverse(); //反转链表
* sort(); //链表排序

**示例：**

void printList(const list<int>& L) {  
  
 for (list<int>::const\_iterator it = L.begin(); it != L.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
bool myCompare(int val1 , int val2)  
{  
 return val1 > val2;  
}  
  
//反转和排序  
void test01()  
{  
 list<int> L;  
 L.push\_back(90);  
 L.push\_back(30);  
 L.push\_back(20);  
 L.push\_back(70);  
 printList(L);  
  
 //反转容器的元素  
 L.reverse();  
 printList(L);  
  
 //排序  
 L.sort(); //默认的排序规则 从小到大  
 printList(L);  
  
 L.sort(myCompare); //指定规则，从大到小  
 printList(L);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 反转 --- reverse
* 排序 --- sort （成员函数）

**总结**：

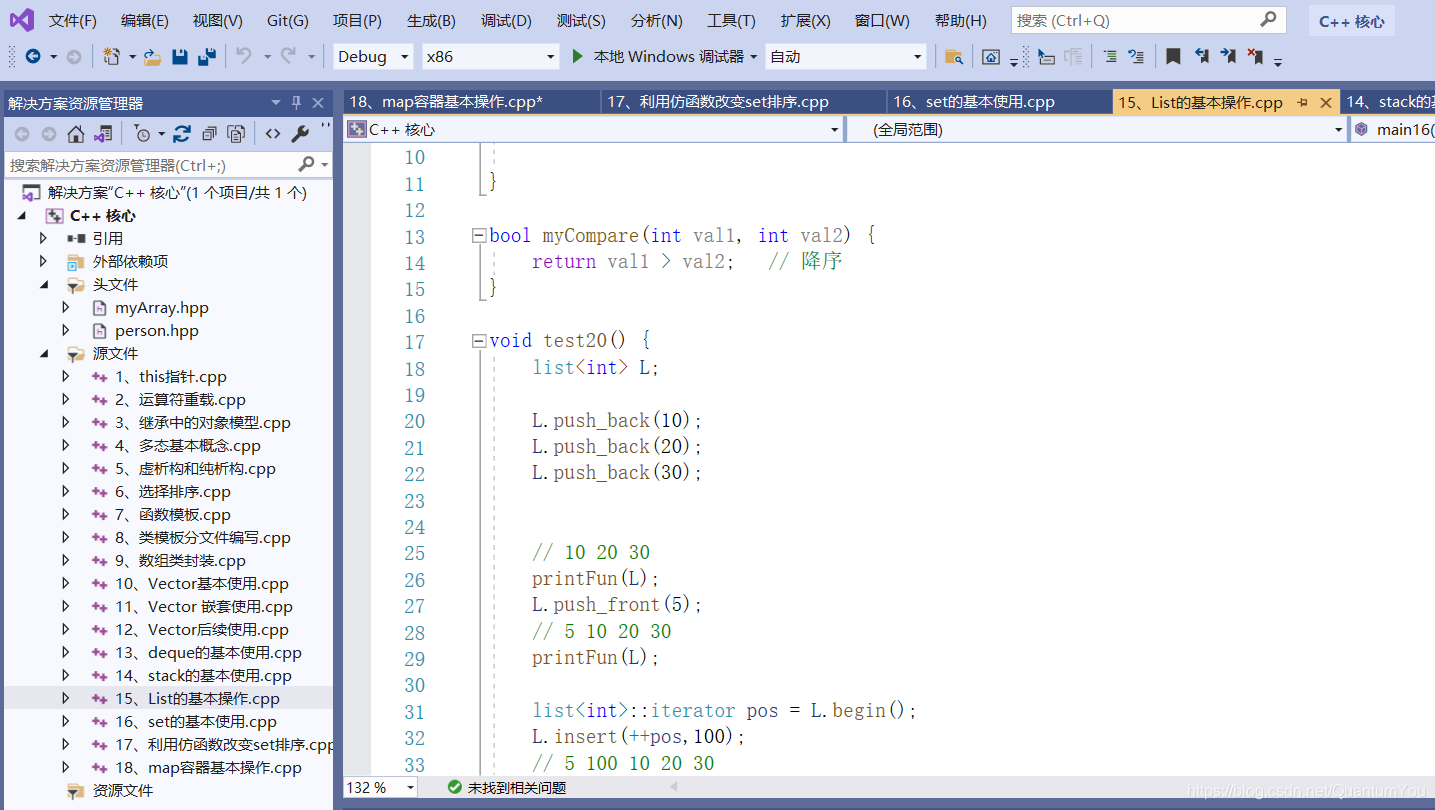
* 对于自定义数据类型，必须要指定排序规则，否则编译器不知道如何进行排序
* 高级排序只是在排序规则上再进行一次逻辑规则制定，并不复

**Tips**:  
验证迭代器是否支持随机访问的

list<int>::iterator it = L1.begin() ;  
it++ ; //支持双向  
it-- ;  
it = it +1 ; // 不支持随机访问 因为可以如此，则可以 it = it +5 ;

* 所有不支持随机访问迭代器的容器，不可以使用标准算法（全局函数）,不支持随机访问送代器的容器，内部会提供对应一些算法
* 可以使用成员函数  
  例如下述：

sort(L1) // 错误，全局函数  
L1.sort() // 成员变量，局部函数



# set/ multiset 容器

## set基本概念

**简介：**

* 所有元素都会在插入时自动被排序

**本质：**

* set/multiset属于**关联式容器**，底层结构是用**二叉树**实现。

**set和multiset区别**：

* set不允许容器中有重复的元素
* multiset允许容器中有重复的元素

## set构造和赋值

功能描述：创建set容器以及赋值

构造：

* set<T> st; //默认构造函数：
* set(const set &st); //拷贝构造函数

赋值：

* set& operator=(const set &st); //重载等号操作符

**示例：**

#include <set>  
  
void printSet(set<int> & s)  
{  
 for (set<int>::iterator it = s.begin(); it != s.end(); it++)  
 {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//构造和赋值  
void test01()  
{  
 set<int> s1;  
  
 s1.insert(10);  
 s1.insert(30);  
 s1.insert(20);  
 s1.insert(40);  
 printSet(s1);  
  
 //拷贝构造  
 set<int>s2(s1);  
 printSet(s2);  
  
 //赋值  
 set<int>s3;  
 s3 = s2;  
 printSet(s3);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* set容器插入数据时用insert
* set容器插入数据的数据会自动排序

## set大小和交换

**功能描述：**

* 统计set容器大小以及交换set容器

**函数原型：**

* size(); //返回容器中元素的数目
* empty(); //判断容器是否为空
* swap(st); //交换两个集合容器

**示例：**

#include <set>  
  
void printSet(set<int> & s)  
{  
 for (set<int>::iterator it = s.begin(); it != s.end(); it++)  
 {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//大小  
void test01()  
{  
  
 set<int> s1;  
   
 s1.insert(10);  
 s1.insert(30);  
 s1.insert(20);  
 s1.insert(40);  
  
 if (s1.empty())  
 {  
 cout << "s1为空" << endl;  
 }  
 else  
 {  
 cout << "s1不为空" << endl;  
 cout << "s1的大小为： " << s1.size() << endl;  
 }  
  
}  
  
//交换  
void test02()  
{  
 set<int> s1;  
  
 s1.insert(10);  
 s1.insert(30);  
 s1.insert(20);  
 s1.insert(40);  
  
 set<int> s2;  
  
 s2.insert(100);  
 s2.insert(300);  
 s2.insert(200);  
 s2.insert(400);  
  
 cout << "交换前" << endl;  
 printSet(s1);  
 printSet(s2);  
 cout << endl;  
  
 cout << "交换后" << endl;  
 s1.swap(s2);  
 printSet(s1);  
 printSet(s2);  
}  
  
int main() {  
  
 //test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 统计大小 --- size
* 判断是否为空 --- empty
* 交换容器 --- swap

## set插入和删除

**功能描述：**

* set容器进行插入数据和删除数据

**函数原型：**

* 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。

**示例：**

#include <set>  
  
void printSet(set<int> & s)  
{  
 for (set<int>::iterator it = s.begin(); it != s.end(); it++)  
 {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//插入和删除  
void test01()  
{  
 set<int> s1;  
 //插入  
 s1.insert(10);  
 s1.insert(30);  
 s1.insert(20);  
 s1.insert(40);  
 printSet(s1);  
  
 //删除  
 s1.erase(s1.begin());  
 printSet(s1);  
  
 s1.erase(30);  
 printSet(s1);  
  
 //清空  
 //s1.erase(s1.begin(), s1.end());  
 s1.clear();  
 printSet(s1);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 插入 --- insert
* 删除 --- erase
* 清空 --- clear

## set查找和统计

**功能描述：**

* 对set容器进行查找数据以及统计数据

**函数原型：**

* find(key); //查找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); //统计key的元素个数

**示例：**

#include <set>  
  
//查找和统计  
void test01()  
{  
 set<int> s1;  
 //插入  
 s1.insert(10);  
 s1.insert(30);  
 s1.insert(20);  
 s1.insert(40);  
   
 //查找  
 set<int>::iterator pos = s1.find(30);  
  
 if (pos != s1.end())  
 {  
 cout << "找到了元素 ： " << \*pos << endl;  
 }  
 else  
 {  
 cout << "未找到元素" << endl;  
 }  
  
 //统计  
 int num = s1.count(30);  
 cout << "num = " << num << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 查找 --- find （返回的是迭代器）
* 统计 --- count （对于set，结果为0或者1）

## set和multiset区别

**学习目标：**

* 掌握set和multiset的区别

**区别：**

* set不可以插入重复数据，而multiset可以
* set插入数据的同时会返回插入结果，表示插入是否成功
* multiset不会检测数据，因此可以插入重复数据
* 注意，原因在于set 插入内置pair对组，<iterator,bool> ,而 multisrt 里面只有iterator

**示例：**

#include <set>  
  
//set和multiset区别  
void test01()  
{  
 set<int> s;  
 pair<set<int>::iterator, bool> ret = s.insert(10);  
 if (ret.second) {  
 cout << "第一次插入成功!" << endl;  
 }  
 else {  
 cout << "第一次插入失败!" << endl;  
 }  
  
 ret = s.insert(10);  
 if (ret.second) {  
 cout << "第二次插入成功!" << endl;  
 }  
 else {  
 cout << "第二次插入失败!" << endl;  
 }  
   
 //multiset  
 multiset<int> ms;  
 ms.insert(10);  
 ms.insert(10);  
  
 for (multiset<int>::iterator it = ms.begin(); it != ms.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 如果不允许插入重复数据可以利用set
* 如果需要插入重复数据利用multiset

## pair对组创建

**功能描述：**

* 成对出现的数据，利用对组可以返回两个数据

**两种创建方式：**

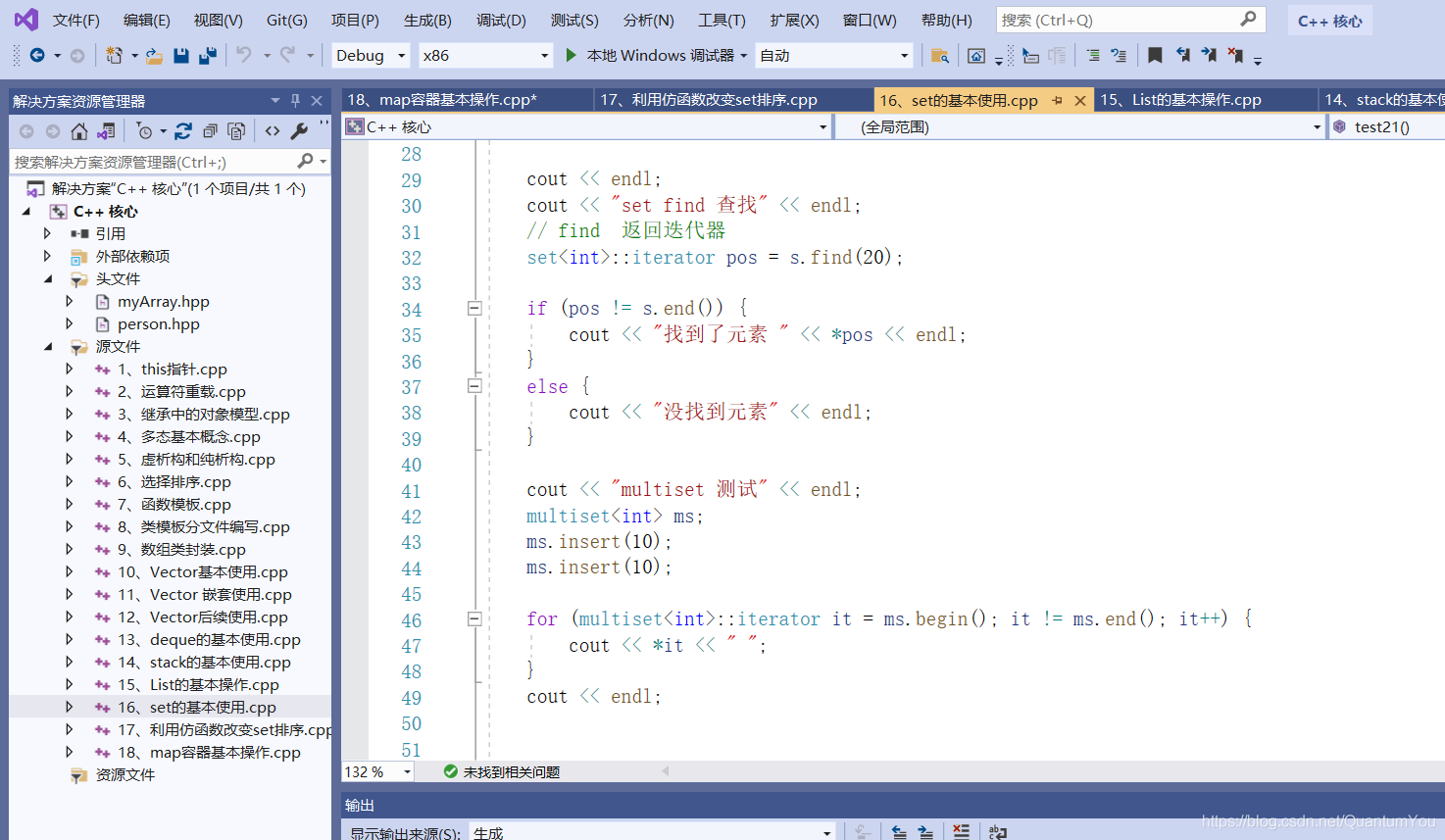
* pair<type, type> p ( value1, value2 );
* pair<type, type> p = make\_pair( value1, value2 );

**示例：**

#include <string>  
  
//对组创建  
void test01()  
{  
 pair<string, int> p(string("Tom"), 20);  
 cout << "姓名： " << p.first << " 年龄： " << p.second << endl;  
  
 pair<string, int> p2 = make\_pair("Jerry", 10);  
 cout << "姓名： " << p2.first << " 年龄： " << p2.second << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

两种方式都可以创建对组，记住一种即可



## set容器排序

学习目标：

* set容器默认排序规则为从小到大，掌握如何改变排序规则

主要技术点：

* 利用仿函数，可以改变排序规则

**示例一** set存放内置数据类型

#include <set>  
  
class MyCompare   
{  
public:  
 bool operator()(int v1, int v2) {  
 return v1 > v2;  
 }  
};  
void test01()   
{   
 set<int> s1;  
 s1.insert(10);  
 s1.insert(40);  
 s1.insert(20);  
 s1.insert(30);  
 s1.insert(50);  
  
 //默认从小到大  
 for (set<int>::iterator it = s1.begin(); it != s1.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
  
 //指定排序规则  
 set<int,MyCompare> s2;  
 s2.insert(10);  
 s2.insert(40);  
 s2.insert(20);  
 s2.insert(30);  
 s2.insert(50);  
  
 for (set<int, MyCompare>::iterator it = s2.begin(); it != s2.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：利用仿函数可以指定set容器的排序规则

**示例二** set存放自定义数据类型

#include <set>  
#include <string>  
  
class Person  
{  
public:  
 Person(string name, int age)  
 {  
 this->m\_Name = name;  
 this->m\_Age = age;  
 }  
  
 string m\_Name;  
 int m\_Age;  
  
};  
class comparePerson  
{  
public:  
 bool operator()(const Person& p1, const Person &p2)  
 {  
 //按照年龄进行排序 降序  
 return p1.m\_Age > p2.m\_Age;  
 }  
};  
  
void test01()  
{  
 set<Person, comparePerson> s;  
  
 Person p1("刘备", 23);  
 Person p2("关羽", 27);  
 Person p3("张飞", 25);  
 Person p4("赵云", 21);  
  
 s.insert(p1);  
 s.insert(p2);  
 s.insert(p3);  
 s.insert(p4);  
  
 for (set<Person, comparePerson>::iterator it = s.begin(); it != s.end(); it++)  
 {  
 cout << "姓名： " << it->m\_Name << " 年龄： " << it->m\_Age << endl;  
 }  
}  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

对于自定义数据类型，set必须指定排序规则才可以插入数据

# map/ multimap容器

## map基本概念

**简介：**

* map中所有元素都是pair
* pair中第一个元素为key（键值），起到索引作用，第二个元素为value（实值）
* 所有元素都会根据元素的键值自动排序

**本质：**

* map/multimap属于**关联式容器**，底层结构是用二叉树实现。

**优点：**

* 可以根据key值快速找到value值

map和multimap**区别**：

* map不允许容器中有重复key值元素
* multimap允许容器中有重复key值元素

## map构造和赋值

**功能描述：**

* 对map容器进行构造和赋值操作

**函数原型：**

**构造：**

* map<T1, T2> mp; //map默认构造函数:
* map(const map &mp); //拷贝构造函数

**赋值：**

* map& operator=(const map &mp); //重载等号操作符

**示例：**

#include <map>  
  
void printMap(map<int,int>&m)  
{  
 for (map<int, int>::iterator it = m.begin(); it != m.end(); it++)  
 {  
 cout << "key = " << it->first << " value = " << it->second << endl;  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 map<int,int>m; //默认构造  
 m.insert(pair<int, int>(1, 10));  
 m.insert(pair<int, int>(2, 20));  
 m.insert(pair<int, int>(3, 30));  
 printMap(m);  
  
 map<int, int>m2(m); //拷贝构造  
 printMap(m2);  
  
 map<int, int>m3;  
 m3 = m2; //赋值  
 printMap(m3);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：map中所有元素都是成对出现，插入数据时候要使用对组

## map大小和交换

**功能描述：**

* 统计map容器大小以及交换map容器

函数原型：

* size(); //返回容器中元素的数目
* empty(); //判断容器是否为空
* swap(st); //交换两个集合容器

**示例：**

#include <map>  
  
void printMap(map<int,int>&m)  
{  
 for (map<int, int>::iterator it = m.begin(); it != m.end(); it++)  
 {  
 cout << "key = " << it->first << " value = " << it->second << endl;  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 map<int, int>m;  
 m.insert(pair<int, int>(1, 10));  
 m.insert(pair<int, int>(2, 20));  
 m.insert(pair<int, int>(3, 30));  
  
 if (m.empty())  
 {  
 cout << "m为空" << endl;  
 }  
 else  
 {  
 cout << "m不为空" << endl;  
 cout << "m的大小为： " << m.size() << endl;  
 }  
}  
  
  
//交换  
void test02()  
{  
 map<int, int>m;  
 m.insert(pair<int, int>(1, 10));  
 m.insert(pair<int, int>(2, 20));  
 m.insert(pair<int, int>(3, 30));  
  
 map<int, int>m2;  
 m2.insert(pair<int, int>(4, 100));  
 m2.insert(pair<int, int>(5, 200));  
 m2.insert(pair<int, int>(6, 300));  
  
 cout << "交换前" << endl;  
 printMap(m);  
 printMap(m2);  
  
 cout << "交换后" << endl;  
 m.swap(m2);  
 printMap(m);  
 printMap(m2);  
}  
  
int main() {  
  
 test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 统计大小 --- size
* 判断是否为空 --- empty
* 交换容器 --- swap

## map插入和删除

**功能描述：**

* map容器进行插入数据和删除数据

**函数原型：**

* 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(key); //删除容器中值为key的元素。

**示例：**

#include <map>  
  
void printMap(map<int,int>&m)  
{  
 for (map<int, int>::iterator it = m.begin(); it != m.end(); it++)  
 {  
 cout << "key = " << it->first << " value = " << it->second << endl;  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 //插入  
 map<int, int> m;  
 //第一种插入方式  
 m.insert(pair<int, int>(1, 10));  
 //第二种插入方式  
 m.insert(make\_pair(2, 20));  
 //第三种插入方式  
 m.insert(map<int, int>::value\_type(3, 30));  
 //第四种插入方式  
 m[4] = 40;   
 printMap(m);  
  
 //删除  
 m.erase(m.begin());  
 printMap(m);  
  
 m.erase(3);  
 printMap(m);  
  
 //清空  
 m.erase(m.begin(),m.end());  
 m.clear();  
 printMap(m);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* map插入方式很多，记住其一即 (有四种插入方式，只需要掌握一)

s.insert(pair<int,int>(1,20))

* 插入 --- insert
* 删除 --- erase
* 清空 --- clear

## map查找和统计

**功能描述：**

* 对map容器进行查找数据以及统计数据

**函数原型：**

* find(key); //查找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); //统计key的元素个数

**示例：**

#include <map>  
  
//查找和统计  
void test01()  
{  
 map<int, int>m;   
 m.insert(pair<int, int>(1, 10));  
 m.insert(pair<int, int>(2, 20));  
 m.insert(pair<int, int>(3, 30));  
  
 //查找  
 map<int, int>::iterator pos = m.find(3);  
  
 if (pos != m.end())  
 {  
 cout << "找到了元素 key = " << (\*pos).first << " value = " << (\*pos).second << endl;  
 }  
 else  
 {  
 cout << "未找到元素" << endl;  
 }  
  
 //统计  
 int num = m.count(3);  
 cout << "num = " << num << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 查找 --- find （返回的是迭代器）
* 统计 --- count （对于map，结果为0或者1）

## map容器排序

**学习目标：**

* map容器默认排序规则为 按照key值进行 从小到大排序，掌握如何改变排序规则

**主要技术点:**

* 利用仿函数，可以改变排序规则

**示例：**

#include <map>  
  
class MyCompare {  
public:  
 bool operator()(int v1, int v2) {  
 return v1 > v2;  
 }  
};  
  
void test01()   
{  
 //默认从小到大排序  
 //利用仿函数实现从大到小排序  
 map<int, int, MyCompare> m;  
  
 m.insert(make\_pair(1, 10));  
 m.insert(make\_pair(2, 20));  
 m.insert(make\_pair(3, 30));  
 m.insert(make\_pair(4, 40));  
 m.insert(make\_pair(5, 50));  
  
 for (map<int, int, MyCompare>::iterator it = m.begin(); it != m.end(); it++) {  
 cout << "key:" << it->first << " value:" << it->second << endl;  
 }  
}  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 利用仿函数可以指定map容器的排序规则
* 对于自定义数据类型，map必须要指定排序规则,同set容器

