

C++语言程序设计

贺利坚 主讲

关联容器

关联容器的特点

► 关联容器的特点

- 每个关联容器都有一个键(key)
- 每个元素按键的取值升序排列
(对于一个关联容器，使用迭代器s在区间[s.begin(), s.end())内遍历，访问到的序列总是升序)

► 优势

- 可以根据键值，高效地查找元素

► 要求：键的类型必须能够用<比较

- int、double等基本数据类型
- 其他重载了<运算符的类型

► 例：学生表的存储

- `class Student{...};`
- `map<string, Student> students;`

学号	姓名	性别	党员	奖学金	出生日期	班级	照片
201151002	张晓明	男	是	3000	1993.5.22	水 1201	
201151005	苏畅	男	否		1994.1.3	水 1201	
201152050	李晓梅	女	否	1000	1993.12.10	土 1202	
201152055	王立志	男	否		1993.10.8	土 1202	
201153060	赵新	男	是	2000	1994.2.25	土 1203	
201153062	王萧	女	是	1000	1993.8.12	土 1203	

简单关联容器——集合(set)

► 数学中的集合

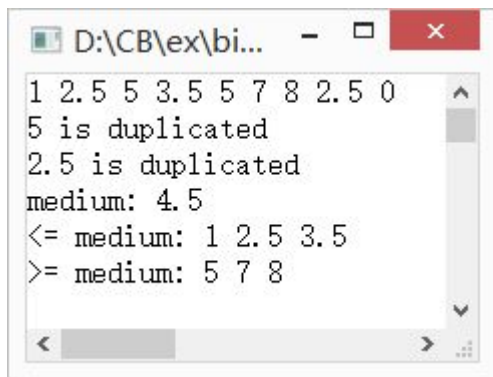
- 一些同质元素的共同体
- 不含重复的元素
- 不讲究顺序
- 有限集合/无限集合

► STL中的set

- 存储一组数据
- 无重复
- 有序
- 有限集合

► 有序的理由

- 高效查找



```
D:\CB\ex\bi... - □ ×
1 2.5 5 3.5 5 7 8 2.5 0
5 is duplicated
2.5 is duplicated
medium: 4.5
<= medium: 1 2.5 3.5
>= medium: 5 7 8
```

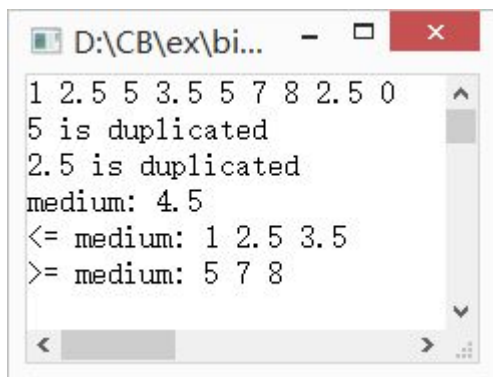
```
#include <set>
#include <iterator>
#include <utility>
#include <iostream>
using namespace std;
```

```
int main()
{
    set<double> s;
    pair<set<double>::iterator, bool> r;
    //输入集合中元素
    set<double>::iterator iter1 = s.begin();
    set<double>::iterator iter2 = s.end();
    double medium = (*iter1 + *(--iter2)) / 2;
    cout << "medium: " << medium << endl;
    //输出小于或等于中值的元素
    cout << "<= medium: ";
    copy(s.begin(), s.upper_bound(medium), ostream_iterator<double>(cout, " "));
    cout << endl;
    //输出大于或等于中值的元素
    cout << ">= medium: ";
    copy(s.lower_bound(medium), s.end(), ostream_iterator<double>(cout, " "));
    cout << endl;
    return 0;
}
```

```
while (true)
{
    double v;
    cin >> v;
    if (v == 0) break;
    r = s.insert(v);
    if (!r.second)
        cout << v << "duplicated\n";
}
```

关联容器的操作：以set为例

- ▶ 插入：insert(...)
- ▶ 删除：erase(...)
- ▶ 查找：find(k)
- ▶ 定界
 - lower_bound(k); //返回第一个不小于k的元素的迭代器
 - upper_bound(k); //返回第一个大于k的元素的迭代器
 - equal_range(k);
- ▶ 计数：count(k);



```
D:\CB\ex\bi... - □ ×
1 2.5 5 3.5 5 7 8 2.5 0
5 is duplicated
2.5 is duplicated
medium: 4.5
<= medium: 1 2.5 3.5
>= medium: 5 7 8
```

```
int main()
{
    set<double> s;
    pair<set<double>::iterator, bool> r;
    //输入集合中元素
    set<double>::iterator iter1 = s.begin();
    set<double>::iterator iter2 = s.end();
    double medium = (*iter1 + *(--iter2)) / 2;
    cout << "medium: " << medium << endl;
    //输出小于或等于中值的元素
    cout << "<= medium: ";
    copy(s.begin(), s.upper_bound(medium),
    cout << endl;
    //输出大于或等于中值的元素
    cout << ">= medium: ";
    copy(s.lower_bound(medium), s.end(),
    cout << endl;
    return 0;
}
```

```
#include <set>
#include <iterator>
#include <utility>
#include <iostream>
using namespace std;
```

```
while (true)
{
    double v;
    cin >> v;
    if (v == 0) break;
    r = s.insert(v);
    if (!r.second)
        cout << v << " duplicated\n";
}
```

```
template <class T1, class T2>
struct pair
{
    T1 first;
    T2 second;
    pair();
    pair(const T1& x, const T2& y);
    template <class U, class V>
    pair (const pair<U,V> &p);
}
```

映射(map)

► 映射与集合相同点

- 一组无重复有序数据

► 映射与集合主要区别

- 集合的元素类型是键本身
- 映射的元素类型是由键和附加数据所构成的二元组。

```
map<string, int> courses;
```

► 在映射中按照键查找一个元素时，除了能确定它的存在性外，还可以得到相应的附加数据。

► map的元素类型—— pair

```
template <class T1, class T2>
```

```
struct pair{
```

```
    T1 first;
```

```
    T2 second;
```

```
    ...
```

```
}
```

```
#include <iostream>
```

```
#include <map>
```

```
#include <string>
```

```
#include <utility>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    map<string, int> courses;
```

```
    map<string, int>::iterator iter;
```

```
    string name;
```

```
    courses.insert(make_pair("CSAPP", 3));
```

```
    courses.insert(make_pair("C++", 2));
```

```
    courses.insert(make_pair("CSARCH", 4));
```

```
    courses.insert(make_pair("COMPILER", 4));
```

```
    courses.insert(make_pair("OS", 5));
```

```
    int n = 3;
```

```
    int sum = 0;
```

```
//选3门课并计算学分
```

```
    cout << "Total credit: " << sum << endl;
```

```
    return 0;
```

```
}
```

```
while (n > 0){
```

```
    cin >> name;
```

```
    iter = courses.find(name);
```

```
    if (iter == courses.end()) {
```

```
        cout << name << " is not available\n";
```

```
    }
```

```
    else {
```

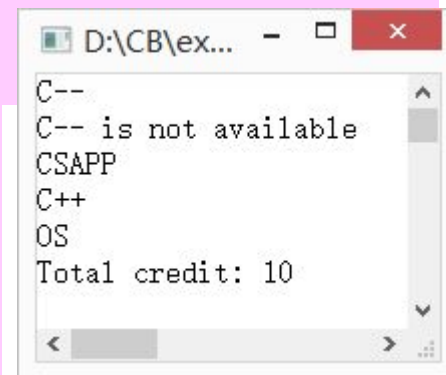
```
        sum += iter->second;
```

```
        courses.erase(iter);
```

```
        n--;
```

```
    }
```

```
}
```



例：有五门课程，每门都有相应学分，从中选择三门，输出学分总和。

关联容器分类

► 简单关联容器

□ 容器只有一个类型参数(set<T>)

□ 容器的元素就是键本身

► 二元关联容器

□ 容器有两个类型参数，分别表示键和附加数据的类型

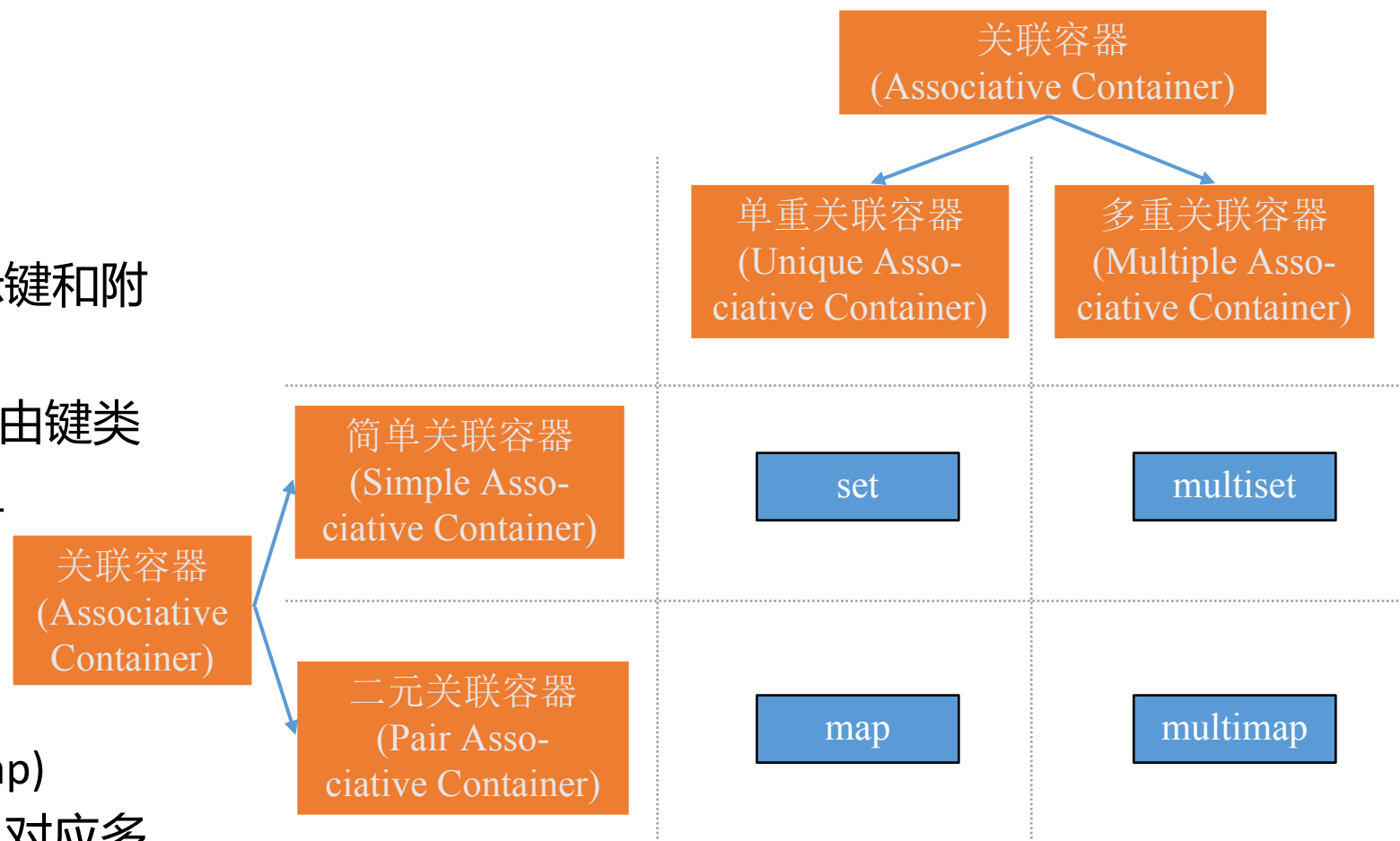
□ 容器的元素类型是pair<K,V>，即由键类型和元素类型复合而成的二元组

► 单重关联容器(set和map)

□ 键值是唯一的，一个键值只能对应一个元素

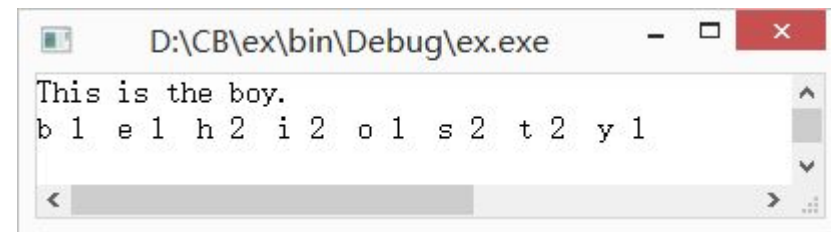
► 多重关联容器(multiset和multimap)

□ 键值是不唯一的，一个键值可以对应多个元素



例：统计一句话中每个字母出现的次数

```
#include <iostream>
#include <map>
#include <cctype>
using namespace std;
int main(){
    map<char, int> s;
    char c;
    do {
        cin >> c;
        if (isalpha(c)) {
            c = tolower(c);
            s[c]++;
        }
    }
    while (c != '.');
    for (map<char, int>::iterator iter = s.begin(); iter != s.end(); ++iter)
        cout << iter->first << " " << iter->second << " ";
    cout << endl;
    return 0;
}
```

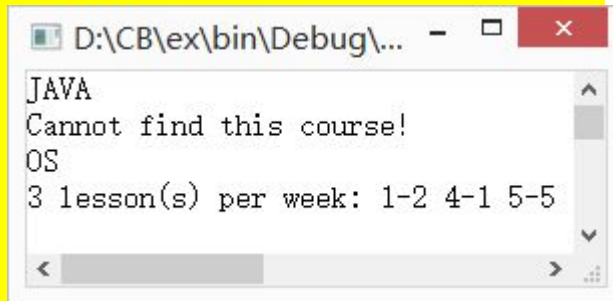


多重集合(multiset)与多重映射(multimap)

- ▶ 多重集合是允许有重复元素的集合
- ▶ 多重映射是允许一个键对应多个附加数据的映射。

```
#include <iostream>
#include <map>
#include <utility>
#include <string>
using namespace std;
```

```
int main()
{
    multimap<string, string> courses;
    typedef multimap<string, string>::iterator CourseIter;
    courses.insert(make_pair("C++", "2-6"));
    courses.insert(make_pair("COMPILER", "3-1"));
    courses.insert(make_pair("COMPILER", "5-2"));
    courses.insert(make_pair("OS", "1-2"));
    courses.insert(make_pair("OS", "4-1"));
    courses.insert(make_pair("OS", "5-5"));
    //输入课程名，输出每周上课次数与时间
    return 0;
}
```



```
string name;
int count;
do
{
    cin >> name;
    count = courses.count(name);
    if (count == 0)
        cout << "Cannot find this course!\n";
}
while (count == 0);

cout << count << " lesson(s) per week: ";
pair<CourseIter, CourseIter> range;
range = courses.equal_range(name);
for (CourseIter iter = range.first;
     iter != range.second;
     ++iter)
    cout << iter->second << " ";
cout << endl;
```