C++语言程序设计

贺利坚 主讲

STL基本算法(续)

排序和搜索算法

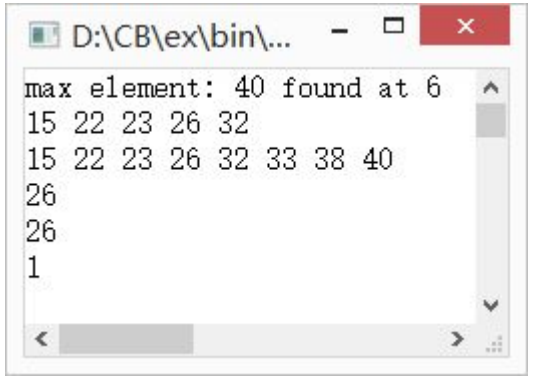
- 排序和搜索算法
  - 对序列进行排序
  - 对两个有序序列进行合并
  - 对有序序列进行搜索
  - 有序序列的集合操作
  - 堆算法
- 例：

```
template <class RandomAccessIterator , class UnaryPredicate>
void sort(RandomAccessIterator first, RandomAccessIterator last, UnaryPredicate comp);
```

//以函数对象comp为“<”，对 [first, last)区间内的数据进行排序

算法示例

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <iterator>
#include <vector>
using namespace std;
```

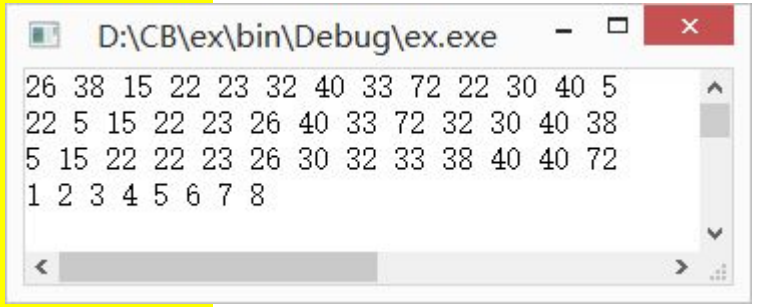



```
int main()//略去了输出中的换行cout<<endl;
{
   int iarray[8] = { 26, 38, 15, 22, 23, 32, 40, 33 };
   // 查找并输出第一个最大值元素及其位置
   vector<int>::iterator p = max_element(ivector.begin(), ivector.end());
   int n = p  - ivector.begin();
   cout << "max element: " << *p << " found at " << n << endl;
   //局部排序并复制到别处
   vector<int> ivector1(5);
   partial_sort_copy(ivector.begin(), ivector.begin()+6, ivector1.begin(), ivector1.end());
   copy(ivector1.begin(), ivector1.end(), ostream_iterator<int>(cout, " "));
   //排序，缺省为递增。
   sort(ivector.begin(), ivector.end());
   copy(ivector.begin(), ivector.end(), ostream_iterator<int>(cout, " "));
    // 返回小于等于24和大于等于24的元素的位置
   cout << *lower_bound(ivector.begin(), ivector.end(), 24) << endl;
   cout << *upper_bound(ivector.begin(), ivector.end(), 24) << endl;
   //对于有序区间，可以用二分查找方法寻找某个元素
   cout << binary_search(ivector.begin(), ivector.end(), 33) << endl;
   return 0;
}
```

算法示例(续)

```
int main()
{
    int iarray[8] = { 26, 38, 15, 22, 23, 32, 40, 33 };
    int iarray1[8] = { 72, 22, 30, 40, 5};
    vector<int> ivector(iarray, iarray + 8);
    vector<int> ivector1(iarray1, iarray1 + 5);
    //合并两个序列将[first, middle) 和[middle, end)中的元素，并将结果放到ivector2中
    vector<int> ivector2(13);
    merge(ivector.begin(), ivector.end(), ivector1.begin(), ivector1.end(), ivector2.begin());
    //将小于*(ivector.begin()+5)的元素放置在该元素之左,其余置于该元素之右
    nth_element(ivector2.begin(), ivector2.begin() + 5, ivector2.end()
    //排序，并保持原来相对位置
    stable_sort(ivector2.begin(), ivector2.end());
    //合并两个有序序列，然后就地替换
    int iarray3[8] = { 1, 3, 5, 7, 2, 4, 6, 8 };
    vector<int> ivector3(iarray3, iarray3 + 8);
    inplace_merge(ivector3.begin(), ivector3.begin() + 4, ivector3.end());
    return 0;
}
```

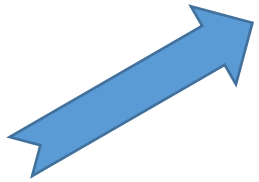
```
#include <iostream>
#include <algorithm>
#include <functional>
#include <iterator>
#include <vector>
using namespace std;
```

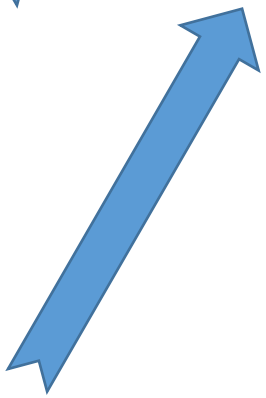


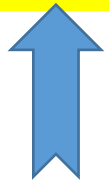
排序算法应用

```
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
struct Person
{
    string name;
    int age;
    string favoriteColor;
};
```

```
const unsigned numberOfPeople = 3;
vector<Person> people(numberOfPeople);
sort(people.begin(), people.end(), sortByName);
copy(people.begin(), people.end(), ostream_iterator<Person>(cout, " "));
cout << endl;
```





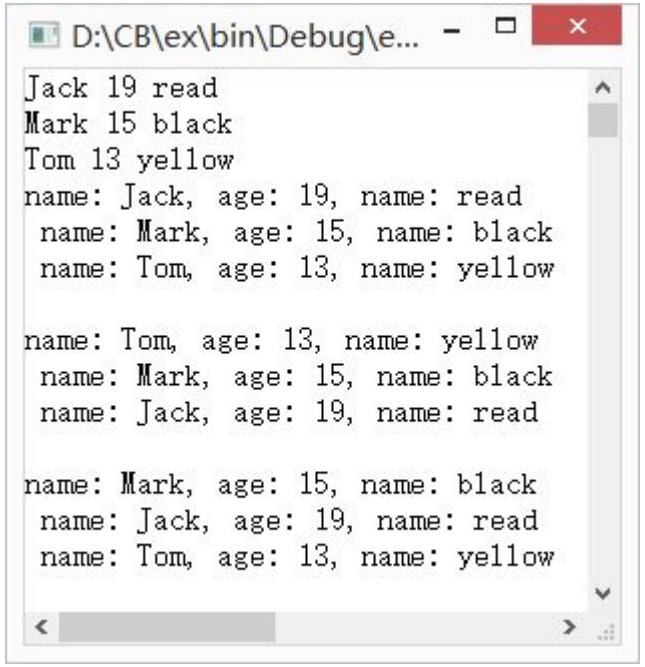


```
ostream &operator<<(ostream &out, Person person)
{
    out<<"name: "<<person.name;
    out<<", age: "<<person.age;
    out<<", name: "<<person.favoriteColor;
    out<<endl;
    return out;
}
```

```
bool sortByName(const Person &lhs, const Person &rhs)
{
    return lhs.name < rhs.name;
}
bool sortByAge(const Person &lhs, const Person &rhs)
{
    return lhs.age < rhs.age;
}
bool sortByColor(const Person &lhs, const Person &rhs)
{
    return lhs.favoriteColor < rhs.favoriteColor;
}
```

排序算法应用(续)

```
struct Person
{
    string name;
    int age;
    string favoriteColor;
};
```




```
int main(){
    vector<Person> people(numberOfPeople);
    for (vector<Person>::size_type i = 0; i != numberOfPeople; ++i)    {
        cin >> people[i].name;
        cin >> people[i].age;
        cin >> people[i].favoriteColor;
    }
    cout << endl;
    // Sort by name
    sort(people.begin(), people.end(), sortByName);
    copy(people.begin(), people.end(), ostream_iterator<Person>(cout, " "));
    cout << endl;
    // Sory by age
    sort(people.begin(), people.end(), sortByAge);
    copy(people.begin(), people.end(), ostream_iterator<Person>(cout, " "));
    cout << endl;
    // Sort by color
    sort(people.begin(), people.end(), sortByColor);
    copy(people.begin(), people.end(), ostream_iterator<Person>(cout, " "));
    cout << endl;
    return 0;
}
```

STL数值算法

- 数值算法
  - 求序列中元素的“和”、部分“和”、相邻元素的“差”或两序列的内积
  - 求“和”的“+”、求“差”的“-”以及求内积的“+”和“·”都可由函数对象指定
- 例：

```
template<class InputIterator, class OutputIterator, class BinaryFunction>
OutputIterator partial_sum(InputIterator first, InputIterator last, OutputIterator result, BinaryFunction op);
```

//对[first, last)内的元素求部分“和”——所谓部分“和”，是一个长度与输入序列相同的序列，其第n项为输入序列前n个元素的“和”

//以函数对象op为“+”运算符

//结果通过result输出

//返回的迭代器指向输出序列最后一个元素的下一个元素

- #include<numeric>

数值算法示例

```
#include <iostream>
#include <numeric>
#include <functional>
#include <iterator>
#include <vector>
using namespace std;
```

```
int main()
{
    int iarray[] = { 1, 2, 3, 4, 5 };
    vector<int> ivector(iarray, iarray + sizeof(iarray) / sizeof(int));

    //元素的累计
    cout << accumulate(ivector.begin(), ivector.end(), 0) << endl;
    //向量的内积
    cout << inner_product(ivector.begin(), ivector.end(), ivector.begin(), 10) << endl;
    //向量容器中元素局部求和
    partial_sum(ivector.begin(), ivector.end(), ostream_iterator<int>(cout," "));
    cout << endl;
    //向量容器中相邻元素的差值
    adjacent_difference(ivector.begin(), ivector.end(), ostream_iterator<int>(cout," "));
    cout << endl;
    return 0;
}
```