新标准C++程序设计

北京大学信息学院　郭 炜

GWPL@PKU.EDU.CN

标准模板库(二)

STL中三个基本的概念：

容器：可容纳各种数据类型的数据结构。

迭代器：可依次存取容器中元素的东西

算法：用来操作容器中的元素的函数模板。例如，STL用sort()来对一个vector中的数据进行排序，用find()来搜索一个list中的对象。函数本身与他们操作的数据的结构和类型无关，因此他们可以在从简单数组到高度复杂容器的任何数据结构上使用。

➢容器分为三大类：

1) 顺序容器

vector, deque, list

2) 关联容器

set, multiset, map, multimap

注意：前2者合称为第一类容器

3) 容器适配器

stack, queue, priority_queue

4.2 list 容器

- 在任何位置插入删除都是常数时间，不支持随机存取。

除了具有所有顺序容器都有的成员函数以外，还支持8个成员函数：

push_front: 在前面插入

pop_front: 删除前面的元素

sort: 排序 (list 不支持 STL 的算法 sort)

remove: 删除和指定值相等的所有元素

unique: 删除所有和前一个元素相同的元素

merge: 合并两个链表，并清空被合并的那个

reverse: 颠倒链表

splice:　在指定位置前面插入另一链表中的一个或多个元素,并在另一链表中删除被插入的元素

```
#include <list>
#include <iostream>
#include <algorithm>
using namespace std;
 class A {
      private:
            int n;
      public:
            A( int n_ ) { n = n_; }
            friend bool operator<( const A & a1, const A & a2);
            friend bool operator==( const A & a1, const A & a2);
            friend ostream & operator <<(  ostream & o, const A & a);
};
```

```
bool operator<( const A & a1, const A & a2) {
    return a1.n < a2.n;
}
bool operator==( const A & a1, const A & a2) {
    return a1.n == a2.n;
}
ostream & operator <<(  ostream & o, const A & a) {
    o << a.n;
    return o;
}
```

```
template <class T>
void PrintList(const list<T> & lst) {
    int tmp = lst.size();
    if( tmp > 0  ) {
        typename list<T>::const_iterator i;
        i = lst.begin();
        for( i = lst.begin();i != lst.end(); i ++)
                cout << * i << ",";
    }
}
// typename用来说明 list<T>::const_iterator是个类型
//在vs中不写也可以
```

```
int main()   {
    list<A>  lst1,lst2;
    lst1.push_back(1);lst1.push_back(3);
    lst1.push_back(2);lst1.push_back(4);  lst1.push_back(2);
    lst2.push_back(10);lst2.push_front(20);
    lst2.push_back(30);lst2.push_back(30);
    lst2.push_back(30);lst2.push_front(40);   lst2.push_back(40);
    cout << "1) "; PrintList( lst1); cout << endl;
    cout << "2) "; PrintList( lst2); cout << endl;
    lst2.sort();
    cout << "3) "; PrintList( lst2); cout << endl;
    lst2.pop_front();
    cout << "4) "; PrintList( lst2); cout << endl;
```

```
lst1.remove(2); //删除所有和A(2)相等的元素
cout << "5) "; PrintList( lst1); cout << endl;
lst2.unique();  //删除所有和前一个元素相等的元素
cout << "6) "; PrintList( lst2); cout << endl;
lst1.merge (lst2);  //合并 lst2到lst1并清空lst2
cout << "7) "; PrintList( lst1); cout << endl;
cout << "8) "; PrintList( lst2); cout << endl;
lst1.reverse();
cout << "9) "; PrintList( lst1); cout << endl;
lst2.push_back (100);lst2.push_back (200);
lst2.push_back (300);lst2.push_back (400);
```

```
    list<A>::iterator p1,p2,p3;
    p1 = find(lst1.begin(),lst1.end(),3);
    p2 = find(lst2.begin(),lst2.end(),200);
    p3 = find(lst2.begin(),lst2.end(),400);
    lst1.splice(p1,lst2,p2, p3);  //将[p2,p3)插入p1之前，
                                  //并从lst2中删除[p2,p3)
    cout << "11) "; PrintList( lst1); cout << endl;
    cout << "12) "; PrintList( lst2); cout << endl;
    return 0;
}
```

输出：

1) 1,3,2,4,2,

2) 40,20,10,30,30,30,40,

3) 10,20,30,30,30,40,40,

4) 20,30,30,30,40,40,

5) 1,3,4,

6) 20,30,40,

7) 1,3,4,20,30,40,

8)

9) 40,30,20,4,3,1,

11) 40,30,20,4,200,300,3,1,

12) 100,400,

4.3 deque 容器

- 所有适用于 vector的操作都适用于 deque。

  deque还有 push_front（将元素插入到前面） 和 pop_front(删除最前面的元素）操作。

5 函数对象

- 是个对象，但是用起来看上去象函数调用，实际上也执行了函数调用。

```
class CMyAverage  {
public:
    double operator()( int a1, int a2, int a3 ) { //重载 () 运算符
        return (double)(a1 + a2+a3) / 3;
    }
};
 CMyAverage average;  //函数对象
 cout << average(3,2,3); // average.operator(3,2,3) 用起来看上去象函数调用 输出 2.66667
```

函数对象的应用：

STL里有以下模板：

```
template<class InIt, class T, class Pred>
T accumulate(InIt first, InIt last, T val, Pred pr);
```

- pr 就是个函数对象。

  对[first,last)中的每个迭代器 I,

  执行 **val = pr(val,* I)** ,返回最终的val。

- Pr也可以是个函数。

Dev C++ 中的 Accumulate 源代码1:

```
template<typename _InputIterator, typename _Tp>
  _Tp  accumulate(_InputIterator __first, _InputIterator __last,
                  _Tp __init)
  {

    for ( ; __first != __last; ++__first)
          __init = __init + *__first;
    return __init;
  }

// typename 等效于class
```

Dev C++ 中的 Accumulate 源代码2:

```
template<typename _InputIterator, typename _Tp, typename
_BinaryOperation>
   _Tp    accumulate(_InputIterator __first, _InputIterator __last,
          _Tp __init,     _BinaryOperation __binary_op)
   {
         for ( ; __first != __last; ++__first)
                __init = __binary_op(__init, *__first);
             return __init;
   }
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <functional>
using namespace std;
int sumSquares( int total, int value)
{    return total + value * value;   }

template <class T>
void PrintInterval(T first, T last)
{ //输出区间[first,last)中的元素
        for( ; first != last; ++ first)
                cout << * first << " ";
        cout << endl;
}
```

```
template<class T>
class SumPowers
{
        private:
                int power;
        public:
                SumPowers(int p):power(p) { }
                const T  operator() ( const T & total,
                                      const T & value)
                { //计算 value的power次方，加到total上
                        T v = value;
                        for( int i = 0;i < power - 1; ++ i)
                                v = v * value;
                        return total + v;
                }
};
```

```
int main()
{
        const int SIZE = 10;
        int a1[] = { 1,2,3,4,5,6,7,8,9,10 };
        vector<int> v(a1,a1+SIZE);
        cout << "1) "; PrintInterval(v.begin(),v.end());
        int result = accumulate(v.begin(),v.end(),0,SumSquares);
        cout << "2) 平方和： " << result << endl;
        result =
          accumulate(v.begin(),v.end(),0,SumPowers<int>(3));
        cout << "3) 立方和： " << result << endl;
        result =
          accumulate(v.begin(),v.end(),0,SumPowers<int>(4));
        cout << "4) 4次方和： " << result;
        return 0;
}
```

```
int result = accumulate(v.begin(),v.end(),0,SumSquares);
```

实例化出：

```
int accumulate(vector<int>::iterator
first,vector<int>::iterator last,
        int init,int ( * op)( int,int))
{
        for ( ; first != last; ++first)
                init = op(init, *first);
    return init;
}
```

```
accumulate(v.begin(),v.end(),0,SumPowers<int>(3));
```

实例化出：

```
int accumulate(vector<int>::iterator
first,vector<int>::iterator last,
        int init, SumPowers<int> op)
{
        for ( ; first != last; ++first)
                init = op(init, *first);
    return init;
}
```

输出：

1) 1 2 3 4 5 6 7 8 9 10
2) 平方和：385
3) 立方和：3025
4) 4次方和：25333

STL 的<functional> 里还有以下函数对象类模板：

equal_to

greater

less …….

这些模板可以用来生成函数对象。

greater 函数对象类模板

```
template<class T>
struct greater : public binary_function<T, T, bool> {
    bool operator()(const T& x, const T& y) const {
        return x > y;
    }
};
```

binary_function定义：

```
template<class Arg1, class Arg2, class Result>
struct binary_function {
    typedef Arg1 first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Result result_type;    };
```

greater 的应用：

- list 有两个sort函数，前面例子中看到的是不带参数的sort函数，它将list中的元素按 < 规定的比较方法 升序排列。
- list还有另一个sort函数：

  void **sort**(T op);

  可以用 op来比较大小，即 op(x,y) 为true则认为x应该排在前面。

```
#include <list>
#include <iostream>
#include <iterator>
using namespace std;
class MyLess {
public:
        bool    operator()( const int & c1, const int & c2 )
        {
                return (c1 % 10) < (c2 % 10);
        }
};
```

```
int main()
{    const int SIZE = 5;
     int a[SIZE] = {5,21,14,2,3};
     list<int> lst(a,a+SIZE);
     lst.sort(MyLess());
     ostream_iterator<int> output(cout,",");
     copy( lst.begin(),lst.end(),output); cout << endl;
     lst.sort(greater<int>()); //greater<int>()是个对象
                               //本句进行降序排序
     copy( lst.begin(),lst.end(),output); cout << endl;

     return 0;
}
```

输出：

```
21,2,3,14,5,
21,14,5,3,2,
```

```
#include <iostream>
#include <iterator>
using namespace std;
class MyLess
{
  public:
     bool operator() (int a1,int a2)
     {
       if( ( a1 % 10 ) < (a2%10) )
           return true;
       else
           return false;
      }
};
bool MyCompare(int a1,int a2)
{
   if( ( a1 % 10 ) < (a2%10) )
       return false;
   else
        return true;  }
```

```
int main()
{
    int a[] = {35,7,13,19,12};
    cout << MyMax(a,5,MyLess()) << endl;
    cout << MyMax(a,5,MyCompare) << endl;
    return 0;
}
```

输出:

```
19
12
```

要求写出MyMax

```
template <class T, class Pred>
T MyMax( T * p, int n, Pred myless)
{
  T tmpmax = p[0];
  for( int i = 1;i < n;i ++ )
       if( myless(tmpmax,p[i]))
           tmpmax = p[i];
  return tmpmax;
};
```

引入函数对象后，STL中的“大”，“小”关系

关联容器和STL中许多算法，都是可以自定义比较器的。在自定义了比较器op的情况下，以下三种说法是等价的：

1) x小于y
2) op(x,y)返回值为true
3) y大于x

6 关联容器

set, multiset, map, multimap

- 内部元素有序排列，新元素插入的位置取决于它的值，查找速度快。
- 除了各容器都有的函数外，还支持以下成员函数：

find: 查找某个值
lower_bound : 查找某个下界
upper_bound : 查找某个上界
equal_range : 同时查找上界和下界
count :计算等于某个值的元素个数
insert: 用以插入一个元素或一个区间

预备知识： pair 模板 stl_pair.h里源代码:

```
template<class _T1, class _T2>
  struct pair
  {
    typedef _T1 first_type;
    typedef _T2 second_type;
    _T1 first;
    _T2 second;
    pair(): first(), second() { }
    pair(const _T1& __a, const _T2& __b)
    : first(__a), second(__b) { }
    template<class _U1, class _U2>
      pair(const pair<_U1, _U2>& __p)
          : first(__p.first), second(__p.second) { }
  };
```

pair模板可以用于生成 key-value对

第三个个构造函数： pair<int,int> p(pair<double,double>(5.5,4.6));

6.1 multiset

定义：

```
template<class Key, class Pred = less<Key>, class A = allocator<Key> >
class multiset { …… };
```

- 第二个参数 Pred 是个函数对象类。

  Pred的对象决定了multiset 中的元素，“一个比另一个小”是怎么定义的。比较两个元素x,y的大小的做法，就是生成一个 Pred 对象，假定为 p,若表达式p(x,y) 返回值为true,则 x比y小。

  Pred的缺省类型是 less<Key>。

- less 模板的定义：

```
template<class T>
struct less : public binary_function<T, T, bool>
{ bool operator()(const T& x, const T& y) { return x < y ; } const;   };
    //less模板是靠 < 来比较大小的
```

multiset的成员函数

iterator find(const T & val);在容器中查找值为val的元素，返回其迭代器。如果找不到，返回**end()**。

iterator insert(const T & val); 将val插入到容器中并返回其迭代器。

void insert(iterator first,iterator last); 将区间[first,last)插入容器。

int count(const T & val); 统计有多少个元素的值和val相等。

iterator lower_bound(const T & val); 查找一个最大的位置 it，使得[begin(),it) 中所有的元素都比 val 小。

iterator upper_bound(const T & val); 查找一个最小的位置 it，使得[it,end()) 中所有的元素都比 val 大。

pair<iterator,iterator>
equal_range(const T & val);同时求得lower_bound和upper_bound。

iterator erase(iterator it);　删除it指向的元素，返回其后面的元素的迭代器(Visual studio 2010上如此，但是在C++标准和Dev C++中，返回值不是这样)。

multiset 的用法：

```
class A{
    .....
};
multiset <A> a;
```

就等效于

```
multiset<A, less<A>> a;
```

- 由于less模板是用 < 进行比较的，所以,这都要求 A 的对象能用 < 比较，即适当重载了 <

//出错的例子：

```
#include <set>
using namespace std;
class A {  };
main()  {
    multiset<A> a;
    a.insert( A());  //error
}
```

//编译出错是因为，插入元素时，**multiset**会将被插入元素和已有元素进行比较，以决定新元素的存放位置。本例中缺省地就是用**less<A>**函数对象进行比较，然而**less<A>**函数对象进行比较时，前提是**A**对象能用 < 进行比较。但本例中没有适当重载 <

- 从 begin() 到 end()遍历一个 multiset对象，就是从小到大遍历各个元素。

例子程序：

```
#include <set>
#include <iostream>
using namespace std;
class MyLess;
```

```
class A {
    private:      int n;
    public:
        A(int n_ ) { n = n_; }
     friend bool operator< ( const A & a1, const A & a2 )
    {               return a1.n < a2.n;  }
    friend ostream &  operator<< ( ostream & o, const A & a2 )
    {               o << a2.n;    return o;   }
    friend class MyLess;
};
```

```
class MyLess
{
public:
    bool operator()( const A & a1, const A & a2) {
        return ( a1.n % 10 ) < (a2.n % 10);
    }
};
typedef multiset<A> MSET1;
typedef multiset<A,MyLess> MSET2;
// MSET2 里，元素的排序规则与 MSET1不同，
//假设 le 是一个 MyLess对象，a1和a2是MSET2对象
//里的元素，那么， le(a1,a2) == true 就说明 a1比a2小
```

```
int main()  {
    const int SIZE = 5;
    A a[SIZE] = { 4,22,19,8,33 };
    ostream_iterator<A> output(cout,",");
    MSET1 m1;          m1.insert(a,a+SIZE);   m1.insert(22);
    cout << "1) " << m1.count(22) << endl;
    MSET1::const_iterator p;
    cout << "2) ";
    for( p = m1.begin();p != m1.end(); p ++ )
        cout << * p << "," ;
    cout << endl;
```

```
MSET2 m2;
m2.insert(a,a+SIZE);
cout << "3) " ;
copy(m2.begin(),m2.end(),output);
cout << endl;
MSET1::iterator pp =  m1.find(19);
if( pp != m1.end() ) //找到
      cout << "found" << endl;
```

```
    cout << "4) " ;
    copy(m1.begin(),m1.end(),output);
    cout <<  endl;
    cout << "6) ";
    cout << * m1.lower_bound(22) << ",";
    cout << * m1.upper_bound(22)<< endl;
    pair<MSET1::iterator, MSET1::iterator>   pr;
    pr = m1.equal_range(22);
    cout << "7) " << * pr.first << "," << * pr.second;
}
```

输出：

1) 2

2) 4,8,19,22,22,33,

3) 22,33,4,8,19,

found

4) 4,8,19,22,22,33,

6) 22,33

7) 22,33

lower_bound

查找[begin,end)中的,最大的位置 FwdIt,使得[begin,FwdIt) 中所有的元素都比 val 小

upper_bound

查找[begin,end)中的,最小的位置 FwdIt,使得[FwdIt,end) 中所有的元素都比 val 大

equal_range

返回值是一个pair, 假设为 p, 则

p.first 就是 lower_bound

p.second 就是 upper_bound

6.2 set

```
template<class Key, class Pred = less<Key>, class A =
    allocator<Key> >

  class set { … }
```

插入set中已有的元素时，忽略插入。