程序设计实习

郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431
刘家瑛 微博 http://weibo.com/pkuliujiaying

标准模板库STL

函数对象

函数对象

- 若一个类重载了运算符“()”，
  则该类的对象就成为函数对象

```
class CMyAverage  {  //函数对象类
public:
      double operator() ( int a1, int a2, int a3 ) {
            return (double)(a1 + a2+a3) / 3;
      }
};
 CMyAverage average;  //函数对象
 cout << average(3,2,3); //  average.operator()(3, 2, 3)
```

输出 2.66667

函数对象的应用

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 等价于class
```

函数对象的应用

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;
  }
```

调用accumulate时， 和__binary_op对应的实参可以是个函数或函数对象

函数对象的应用示例

```
#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;
}
```

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

```
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;
}
```

STL中的函数对象类模板

以下模板可以用来生成函数对象。

equal_to

greater

less

…….

头文件： <functional>

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;
        }
};
```

greater 的应用

list 有两个sort成员函数

- void sort();

  将list中的元素按 “<” 规定的比较方法升序排列。

- template <class Compare>

  void sort (Compare op);

  将list中的元素按 op 规定的比较方法升序排列。即要比较x,y大小时，看 op(x,y)的返回值，为true则认为 x小于y

greater 的应用

```
#include <list>
#include <iostream>
using namespace std;
class MyLess {
    public:
      bool operator()( const int & c1, const int & c2 )
      {
            return (c1 % 10) < (c2 % 10);
       }
};
template <class T>
void Print(T first,T last) {
          for(  ; first != last ; ++ first )  cout << * first << ",";
}
```

```
int main()
{     const int SIZE = 5;
      int a[SIZE] = {5,21,14,2,3};
      list<int> lst(a,a+SIZE);
      lst.sort(MyLess());
      Print( lst.begin(),lst.end());
      cout << endl;
      lst.sort(greater<int>()); //greater<int>()是个对象
      Print( lst.begin(),lst.end());
      cout << endl;
      return 0;
}
```

输出：

21,2,3,14,5,

21,14,5,3,2,

在STL中使用自定义的“大”，“小”关系

关联容器和STL中许多算法，都是可以用函数或函数对象自定义比较器的。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

例题:写出MyMax模板

```
#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,a+5,MyLess())
              << endl;
    cout << * MyMax(a,a+5,MyCompare)
              << endl;
    return 0;
}
```

输出:
19
12

例题:写出MyMax模板

```
template <class T, class Pred>
T MyMax( T first, T last, Pred myless)
{
  T tmpMax = first;
  for(; first != last; ++ first)
       if( myless( * tmpMax,* first))
            tmpMax = first;
  return tmpMax;
};
```

In-Video Quiz

如果A是一个类，lst是list<A>类的对象，则若要使lst.sort(greater<A>())能够工作，必须重载哪个运算符？

A) >

B) <

C) ==

D) 亲，根本不需要重载任何运算符的哟