1. 适配器

我们都知道，笔记本电脑的转接头适配器能够扩展缺少的接口，STL中的适配器也是一样的效果，它能够将某些类型扩展接口，适配到其他类型中。即扩展参数的个数，如将一元函数扩展为二元函数…

需要引入头文件

#include<functional>

二．仿函数 适配器扩展参数到二元

（一）应用场景：本为一元函数，需要将用户输入的num适配到原有的MyPoint()仿函数中，因此用适配器将MyPoint()仿函数适配为二元仿函数。

（二）步骤

第一步： 绑定数据 bind2nd，意思是绑定数据num到第二个参数start上; 绑定数据bind1st，将数据num绑定到第一个参数v上

第二步：仿函数的类继承 binary\_function<参数类型1， 参数类型2，返回值类型>

第三步：加const限定 operator()，不允许修改

Class MyPoint : public binary\_function<int, int, void>

{

Public:

Void operator() (int v, int start) const

{  
 cout << v + start << endl;

}

}

Void test()

{  
 vector<int> v;

For(int I = 0 ; I < 10; ++i)

{   
 v.push\_back(i);

}

Int num;

Cin >> num;

For\_each(v.begin(), v.end(), bind2nd(MyPoint(), num) );

}

三．取反适配器

not1对一元函数对象取反，not2对二元函数对象取反

（一）步骤

继承 unary\_function<参数类型1，返回值类型>

限制const

（二）场景：

在vector中寻找大于5的第一个数字，后更改需求，寻找小于5的第一个数字，将该一元仿函数转换成相反的功能。Not1适配器，实现相反功能

Class GreaterThanFive : public unary\_function<int, bool>

{

Public:

Bool operator() (int v) const

{  
 return v > 5;

}  
}

Void test()

{

Vector<int> v;

For(int I = 0; I < 10; ++i)

{

v.push\_back(i);

}

Vector<int> iterator pos = find\_if(v.begin(), v.end(), not1(GreaterThanFive()) );

If( pos != v.end())

{  
 cout << “找到小于5的数字为：” << \*pos << endl;

}

Else

{  
 cout <<”未找到” << endl;

}

}

四．普通函数指针适配器 ptr\_fun

普通函数指针不像前面说的仿函数是个类能够继承，我们需要将普通函数指针适配为函数对象（仿函数），用到ptr\_fun，ptr\_fun(普通函数指针)适配成了函数对象，也能够向前面一样扩展接口参数。但不需要继承，以及限制const

Void MyPrint(int v, int start)

{  
 cout << v +start <<endl;

}

Void test()

{  
 vector<int> v;

For(int I = 0; I < 10; ++i)

{  
 v.push\_back(i);

}

Int num;

Cin >> num;

For\_each(v.begin(), v.end(), bind2nd(ptr\_fun(MyPoint), num ) );

}

五．成员函数适配器 mem\_fun\_ref / mem\_fun

针对类中的内容，我们之前对类中的内容进行排序时，设置一个全局回调函数，调用回调函数。因为是对同一个类的内容进行处理，那么用类中的成员函数不是更好么？

因此，可以用成员函数适配器mem\_fun\_ref将类中的成员函数适配成回调函数。

两种形式：

如：当vector<Person> v; vector中存储的是Person对象时，用mem\_fun\_ref适配器。 mem\_fun\_ref( &Person::showPerson) 将Person类中的showPerson()成员函数配置成回调函数,注意括号内为Person类的showPerson的成员函数的地址。

当vector<Person \*> v; vector中存储的是Person的对象指针时，使用mem\_fun适配器。Mem\_fun(&Person::showPerson));

Class Person

{  
public:

Person(string name, int age)

{  
this->m\_Name = name;

This->m\_Age = age;

}

Void showPerson()

{  
 cout << “姓名：” << m\_Name <<” 年龄：“ << m\_Age <<endl;

}

String m\_Name;

Int m\_Age;

}

Void test()

{  
vector<Person> v;

Person p1(“1”, 1);

Person p2(“2”, 2);

v.push\_back(p1);

v.push\_back(p2);

for\_each(v.begin(), v.end(), mem\_fun\_ref(&Person::showPerson));  
}