观察者模式是一种行为设计模式，它用于在对象之间建立一对多的依赖关系，当一个对象的状态发生变化时，所有依赖它的对象都会得到通知并自动更新。下面是一个简单的观察者模式的实现示例：

首先，我们需要定义一个主题（Subject）类，用于管理观察者的注册、移除和通知。

cpp

#include <iostream>

#include <vector>

class Observer;

class Subject { //被观测类

public:

void attach(Observer\* observer) { //添加观察者

observers.push\_back(observer);

}

void detach(Observer\* observer) { //移除观察者

for (auto it = observers.begin(); it != observers.end(); ++it) {

if (\*it == observer) {

observers.erase(it);

break;

}

}

}

void notify() { //通知观察者

for (auto observer : observers) {

observer->update() ; //进行更新

}

}

private:

std::vector<Observer\*> observers; //观察者成员变量

};

然后，我们定义一个观察者（Observer）类，它具有一个更新（update）方法，用于接收主题的通知。

cpp

class Observer {

public:

virtual void update() = 0; //抽象观察者类

};

接下来，我们定义一个具体的观察者类，它实现了观察者接口，并实现了具体的更新方法。

cpp

class ConcreteObserver : public Observer { //多态派生类

public:

void update() override {

std::cout << "Received notification from subject" << std::endl; //实现update方法

}

};

最后，我们可以使用这些类来实现观察者模式的功能。

cpp

int main() {

Subject subject;

ConcreteObserver observer1; //添加观察者1

subject.attach(&observer1);

ConcreteObserver observer2; //添加观察者2

subject.attach(&observer2);

subject.notify(); //通知观察者进行更新

subject.detach(&observer1); //移除观察者1

subject.notify(); //再次通知观察者

return 0;

}

在上面的示例中，我们创建了一个主题对象subject，并创建了两个观察者对象observer1和observer2。然后，我们将观察者对象注册到主题对象中。当调用subject.notify()时，主题对象会通知所有注册的观察者对象，并调用它们的更新方法。最后，我们将observer1从主题对象中移除，并再次调用subject.notify()。

运行上面的代码，你会看到输出结果：

Received notification from subject

Received notification from subject

Received notification from subject

这表明观察者模式成功地将主题对象的状态变化通知给了观察者对象。