In this exercise we study a simple implementation of observer pattern.

Excercise 11 (Observer pattern, 4p)

Modify the class OverflowCounter of execise 10 so that it informs a single observer, when an overflow has occurred. The Observer is an interface class that any class can implement, if it wants to be informed about an overflow. Usually this class is a class that uses the counter. The advantage of this pattern is that when the OverflowCounter class has been implemented and tested, it does not need any modification even when we use it from any kind of "counter user objects". Only thing we need to do to get our counter user class to be informed is to implement the Observer interface in the counter user class.

To make this work and see how it works in practice you need the following modifications to the OverflowCounter class:

- Add a data member that points to the Observer interface class.
  ```
  Observer* obs;
  ```
- Add function SetObserver, that sets the pointer member to point to any object that implements the Observer interface.
  ```
  void SetObserver(Observer *)
  ```
- Add **private** function Notify, that informs the observer by calling the function HandleLimitReached() of the observer.

Only one function is needed in the interface of the observer. The function is called HandleLimitReached. This function is used in a way to pass the message "Limit has been reached" from the OverflowCounter to the observer. This member function of observer is called from the member function Notify of the OverflowCounter.

```
class Observer {
public:
	virtual void HandleLimitReached() = 0;
};
```

To test the new version of OverflowCounter, write an example class CounterUser, that uses a limited counter. It has an OverflowCounter as a data member and it implements the observer interface. To keep the test class as simple as possible only the following member functions are necessary:

- Constructor where the limited counter is initialized with a limit value (5 for example) and where CounterUser object is set as the observer of the OverflowCounter.
- IncrementBy, where counter is incremented n times, where n is passed as a parameter.
- HandleLimitReached() that displays that the limit has been reached.

Use the following test program where CounterUser object is declared, member function IncrementBy is called (with parameter value 12). When you run the program, you should see the message "Limit reached" twice from the first counter user, because limit is 5 and number of increments is 12) and once from the second.

```
void main(){
	CounterUser cu(5);
	cu.IncrementBy(12); //OUTPUT: two times "Limit has been reached"
	CounterUser cu2(9);
	cu2.IncrementBy(9);
	cout << "Just passing time" << endl;
	cu2.IncrementBy(1); //OUTPUT: "Limit has been reached"
}
```

Extra exercise (Advanced version 4p)

Amend your Counter class by adding a member function to set Observer. The implementations should use a vector to store pointers to counter observers and when limit is reached or overflow occurs the counter passes a pointer to itself as a parameter to the observer. The observer then prints notification and the current counter value.

```
class Counter {
public:
	virtual void inc() = 0;
	virtual void dec() = 0;
	virtual void SetObserver(CounterObserver *) = 0;
	virtual operator int() = 0;
	virtual ~Counter() {};
};

class CounterObserver {
public:
	virtual void HandleLimitReached(Counter *ctr) = 0;
};
```

Write a test program that creates two counters of both types and three observers. Two observers to monitor a single counter each and one that monitors both.

Test your objects with UseCounter function from exercise 10. Write a set of tests to verify that your counters and observers work correctly.