FE545 Final Project

Trading Strategy with Design Patterns

By Wan Li

Project Objective

The purpose of the project is to create different trading strategies to check whether they are effective in reality and to design different patterns to help implement the whole processes more flexible and always apply to object oriented principle design.

Expected Results

1. **Create Positive Return from the strategies**
2. **Open for extension, close for modification for further use.**

Financial Specification

1. Indicator

Stochastic RSI: it's an oscillator that calculates a value between 0 and 1 which is then plotted as a line. The indicator is normally used for identifying overbought and oversold conditions. The calculation formula is as below:

```
StochRSI = (RSI - Lowest Low RSI) / (Highest High RSI - Lowest Low RSI)
```

```
              100
RSI = 100 - --------
             1 + RS

RS = Average Gain / Average Loss
```

Average Gain = EMA(Gain, 14), Average Loss = EMA(Loss, 14)

```
EMA: {Close - EMA(previous day)} x multiplier + EMA(previous day).
```

```
Multiplier: (2 / (Time periods + 1) )
```

Application:

Stochastic RSI < 0.2, indicate oversold signal, buy action is suggested.

Stochastic RSI > 0.8, indicate oversell signal, sell action is suggested.

SMA:

A simple moving average (SMA) is a simple, or arithmetic, moving average that is calculated by adding the closing price of the security for a number of time periods and then dividing this total by the number of time periods. It's used to measure the price trend movement. Short-term averages respond quickly to changes in the price of the underling, while long-term averages are slow to react.

SMA = Sum(price)/ Period

Application:

When short term SMA line across the long term SMA line, an uptrend signal indicates, buy action is suggested. When short term SMA line across the long term SMA line, a downtrend signal indicates, sell action is suggested.

2. **Strategy**
   Two strategies are applied in the project;
   One is the positive strategy, namely moderate strategy since it's not too aggressive, it's firstly based on the SMA signal for long-term trend track, and once an uptrend forms or downtrend forms, it begins to search StochasticRSI signal for short-term arbitrage. Strategy also sets 5% as stop loss to control potential risks;

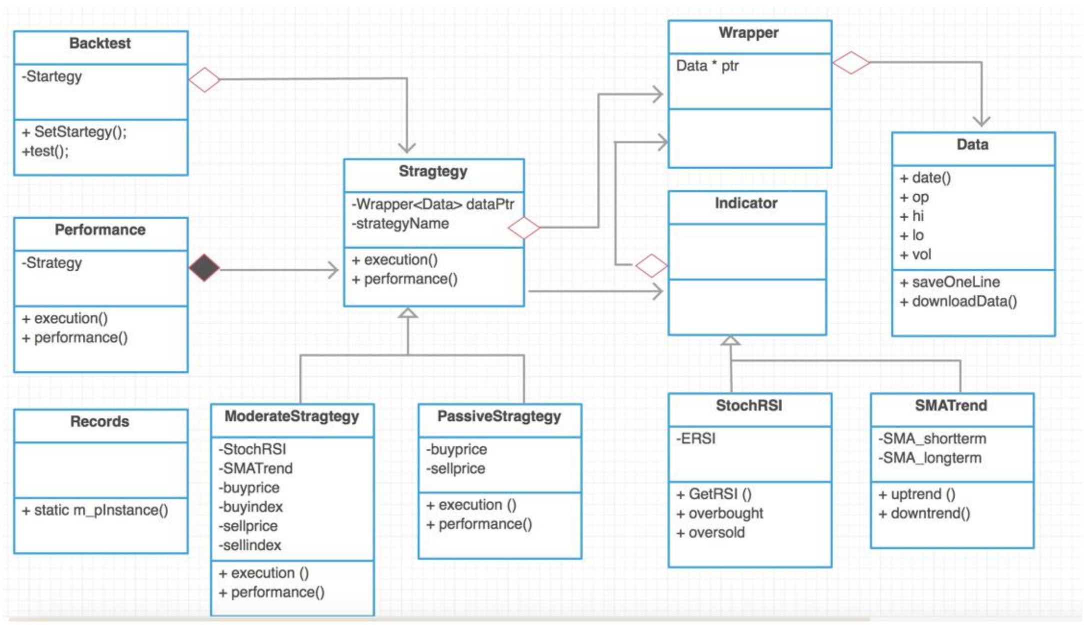
   Second one is the oppositely passive strategy, it follows the principle of efficient market and arbitrage-free theory, so nothing need to be done, just like buying ETF or index randomly and earn the market gain.

3. **Performance**

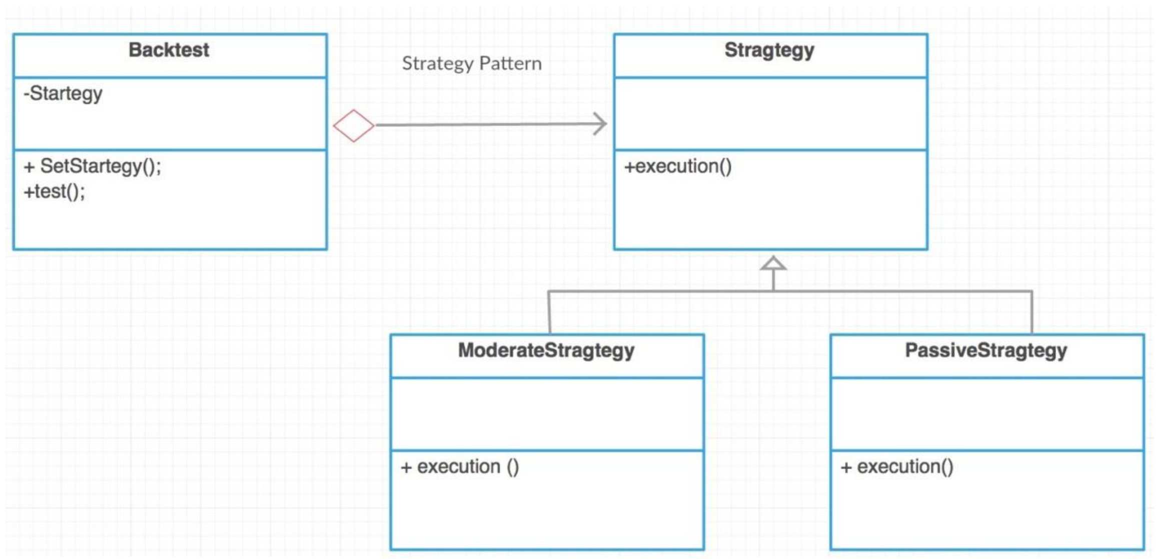
   Finally an overall performance will be evaluated;
   For the first strategy, it will value the trading times and corresponding return;
   For the second strategy, it will value the whole time return.

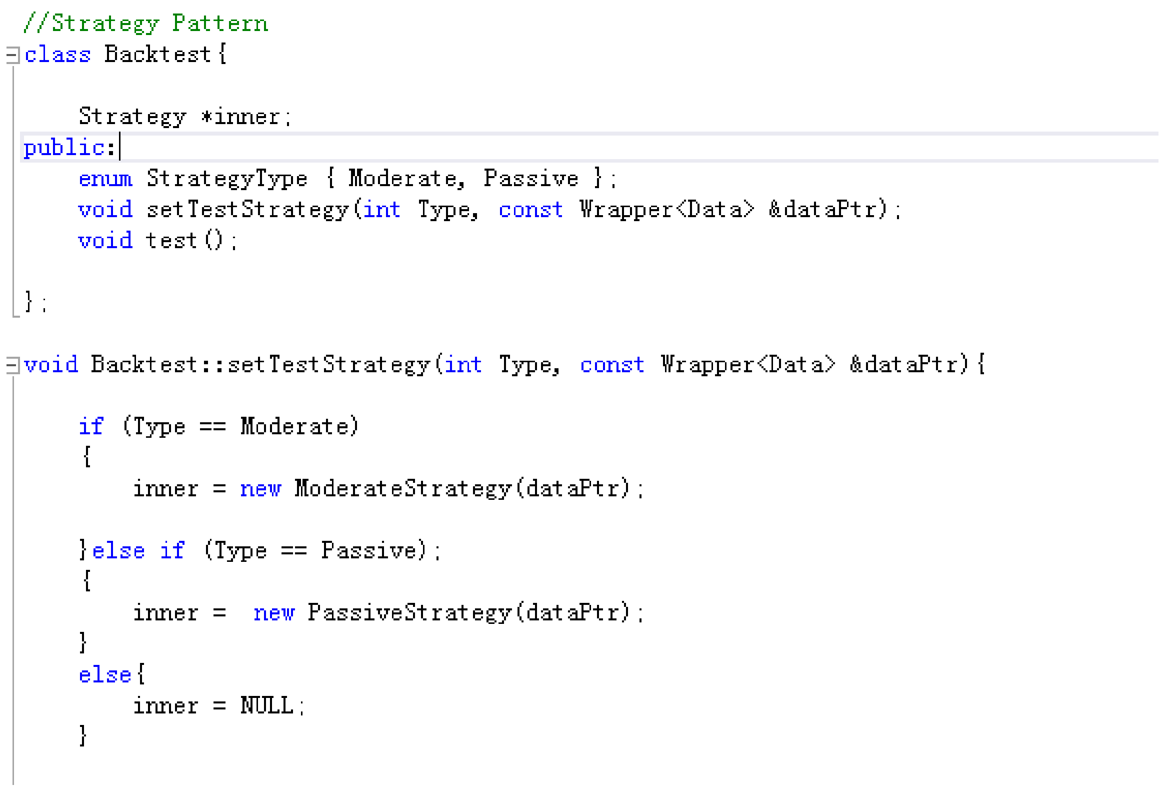
Technical Specification

The overall diagram

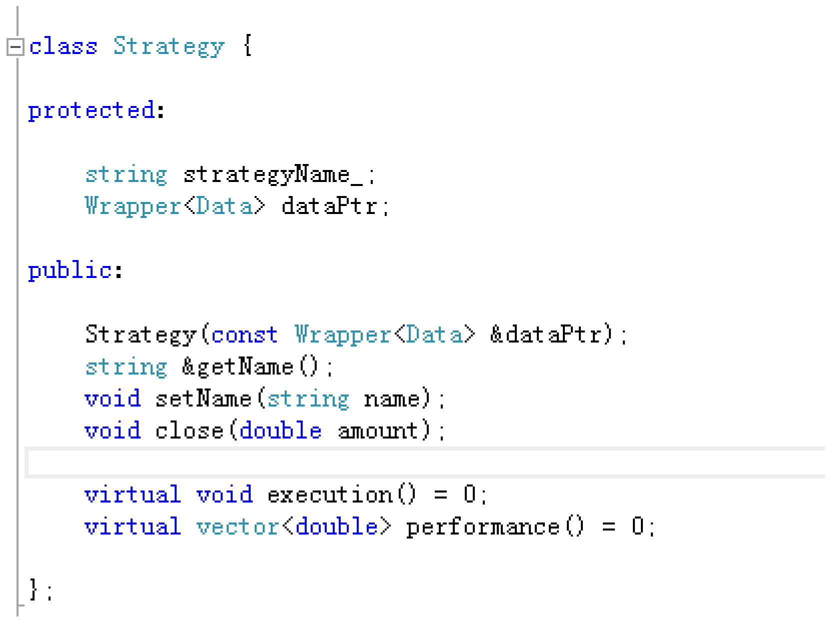


1. **Strategy Design Pattern**



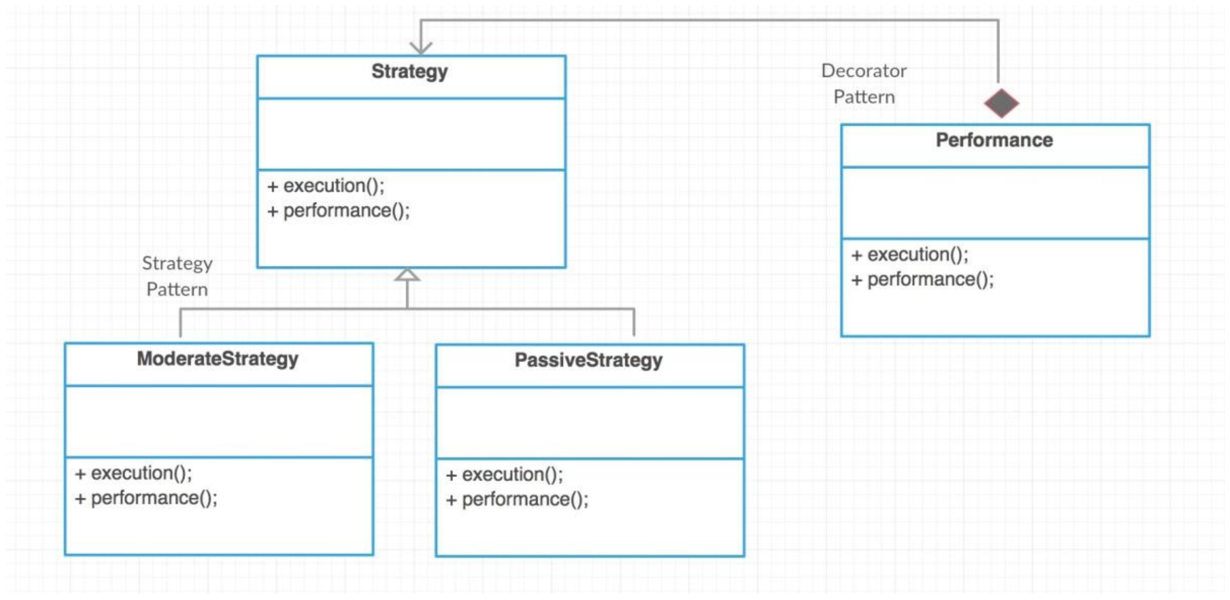


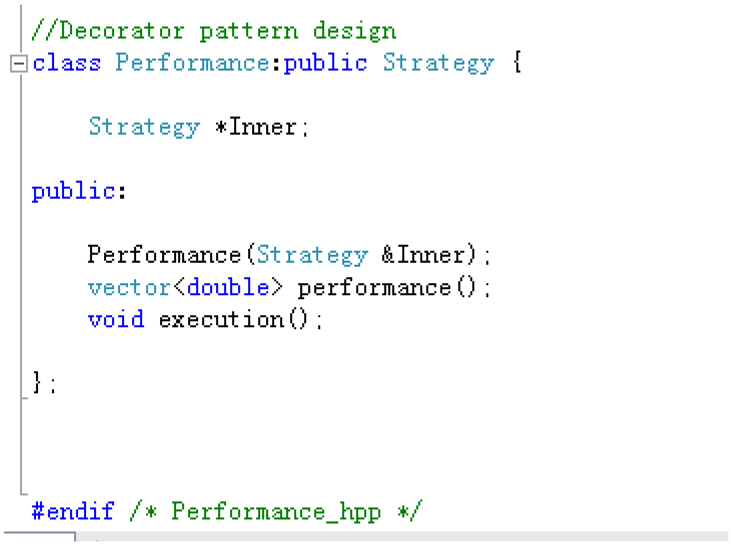
```cpp
//Strategy Pattern
class Backtest{

    Strategy *inner;
public:
    enum StrategyType { Moderate, Passive };
    void setTestStrategy(int Type, const Wrapper<Data> &dataPtr);
    void test();

};

void Backtest::setTestStrategy(int Type, const Wrapper<Data> &dataPtr){

    if (Type == Moderate)
    {
        inner = new ModerateStrategy(dataPtr);

    }else if (Type == Passive);
    {
        inner =  new PassiveStrategy(dataPtr);
    }
    else{
        inner = NULL;
    }
```



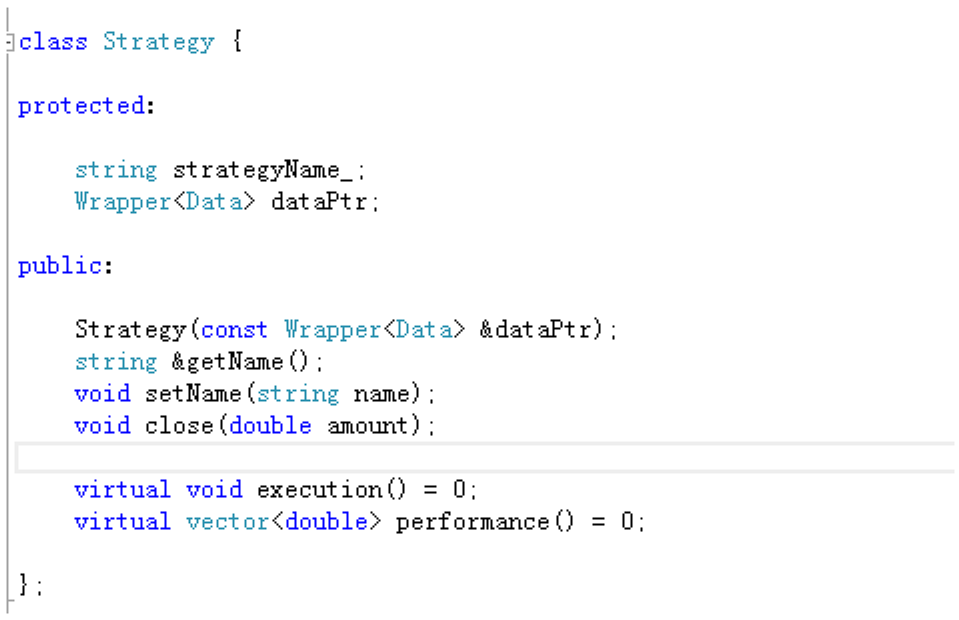
```cpp
class Strategy {

protected:

    string strategyName_;
    Wrapper<Data> dataPtr;

public:

    Strategy(const Wrapper<Data> &dataPtr);
    string &getName();
    void setName(string name);
    void close(double amount);

    virtual void execution() = 0;
    virtual vector<double> performance() = 0;

};
```

2. **Decorator Design Pattern**

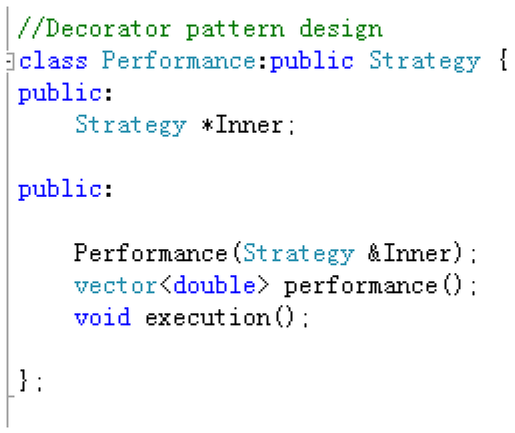




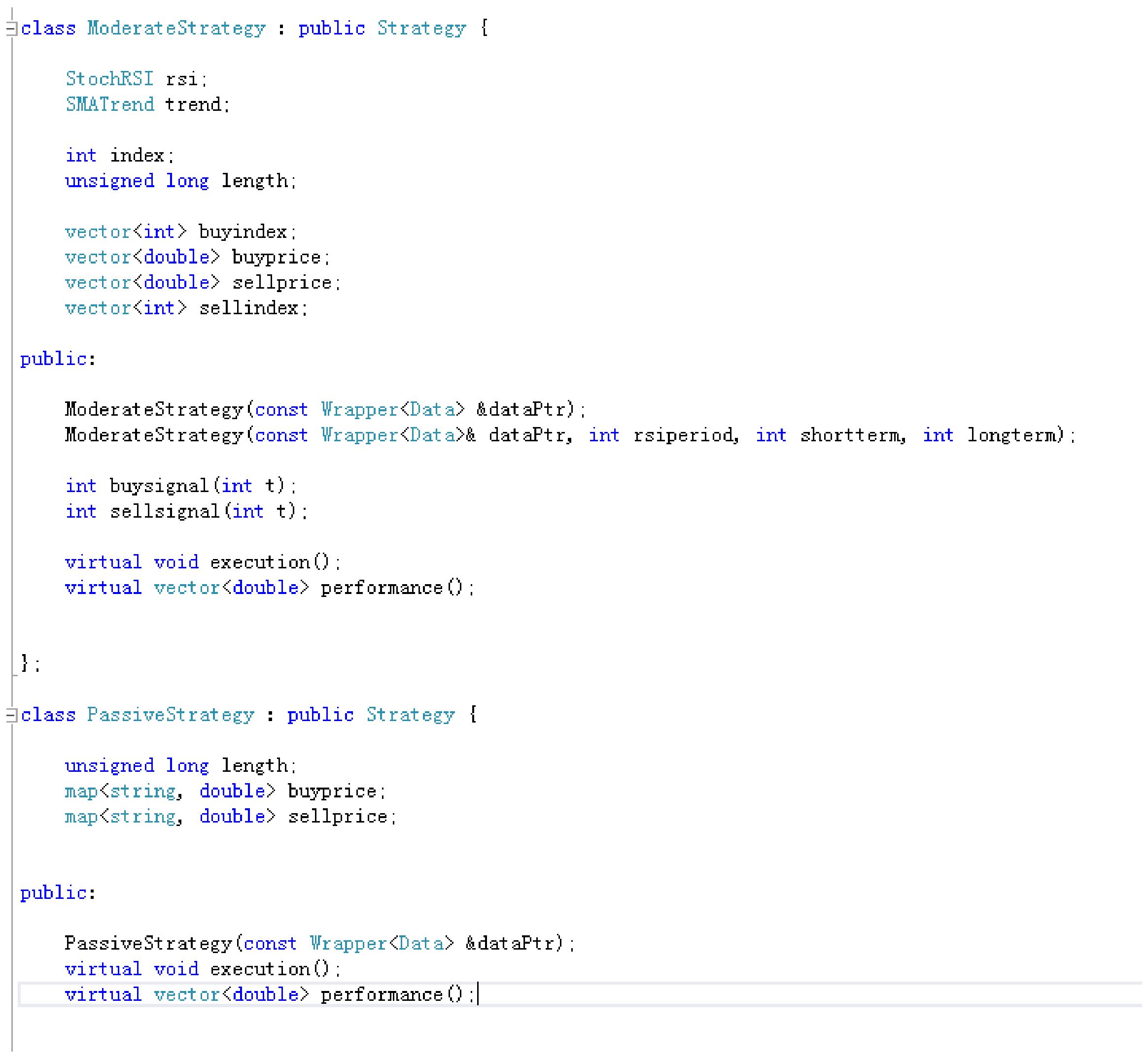
```
//Decorator pattern design
class Performance:public Strategy {

    Strategy *Inner;

public:

    Performance(Strategy &Inner);
    vector<double> performance();
    void execution();

};


#endif /* Performance_hpp */
```



```cpp
class Strategy {

protected:

    string strategyName_;
    Wrapper<Data> dataPtr;

public:

    Strategy(const Wrapper<Data> &dataPtr);
    string &getName();
    void setName(string name);
    void close(double amount);

    virtual void execution() = 0;
    virtual vector<double> performance() = 0;

};
```

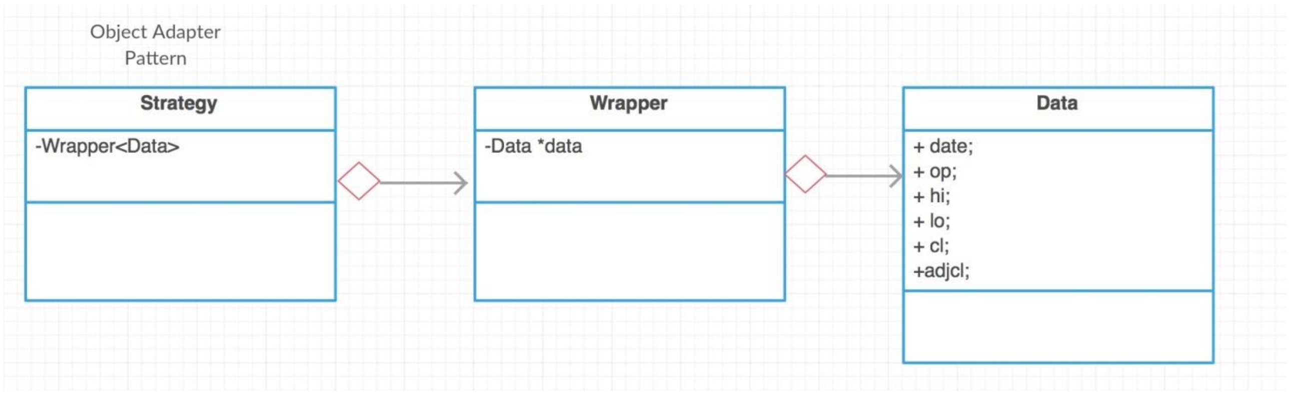


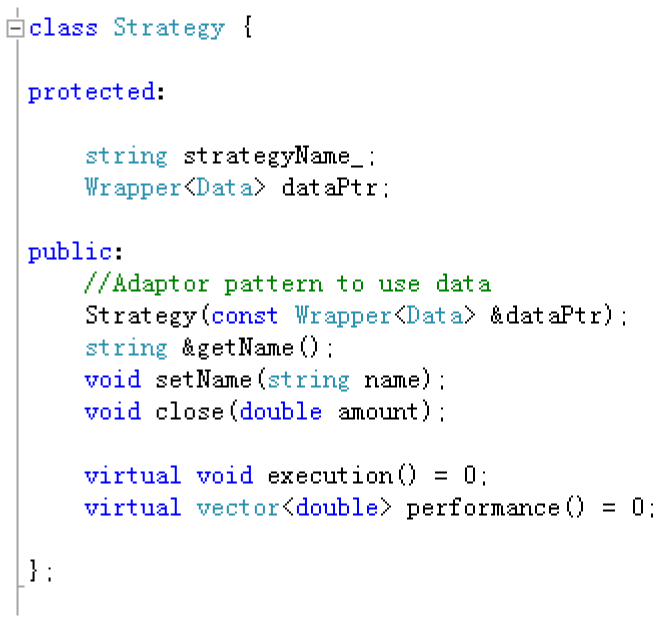
```cpp
//Decorator pattern design
class Performance:public Strategy {
public:
    Strategy *Inner;

public:

    Performance(Strategy &Inner);
    vector<double> performance();
    void execution();

};
```



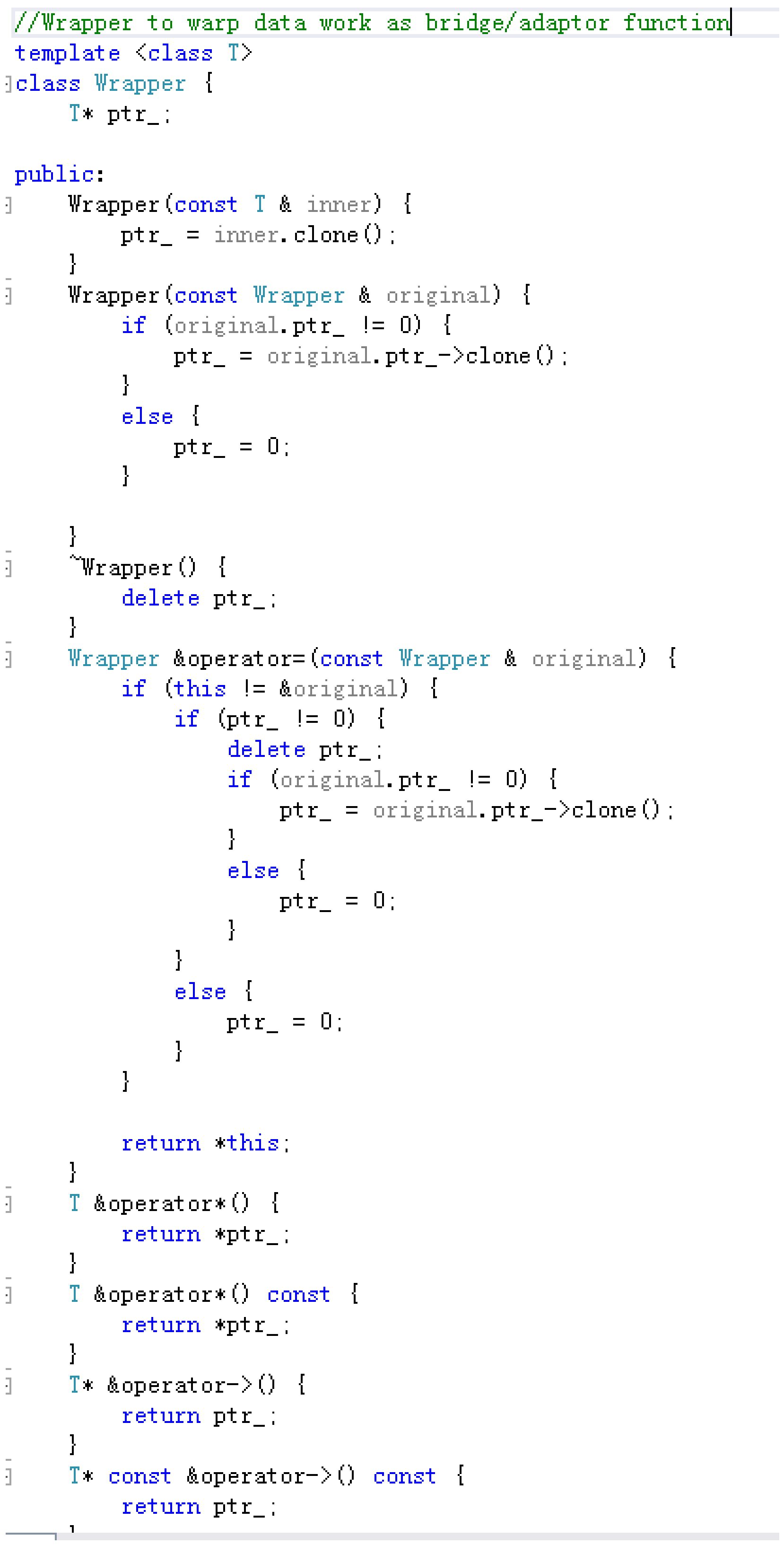
```
class ModerateStrategy : public Strategy {

    StochRSI rsi;
    SMATrend trend;

    int index;
    unsigned long length;

    vector<int> buyindex;
    vector<double> buyprice;
    vector<double> sellprice;
    vector<int> sellindex;

public:

    ModerateStrategy(const Wrapper<Data> &dataPtr);
    ModerateStrategy(const Wrapper<Data>& dataPtr, int rsiperiod, int shortterm, int longterm);

    int buysignal(int t);
    int sellsignal(int t);

    virtual void execution();
    virtual vector<double> performance();


};

class PassiveStrategy : public Strategy {

    unsigned long length;
    map<string, double> buyprice;
    map<string, double> sellprice;


public:

    PassiveStrategy(const Wrapper<Data> &dataPtr);
    virtual void execution();
    virtual vector<double> performance();
```

3. Adaptor Pattern

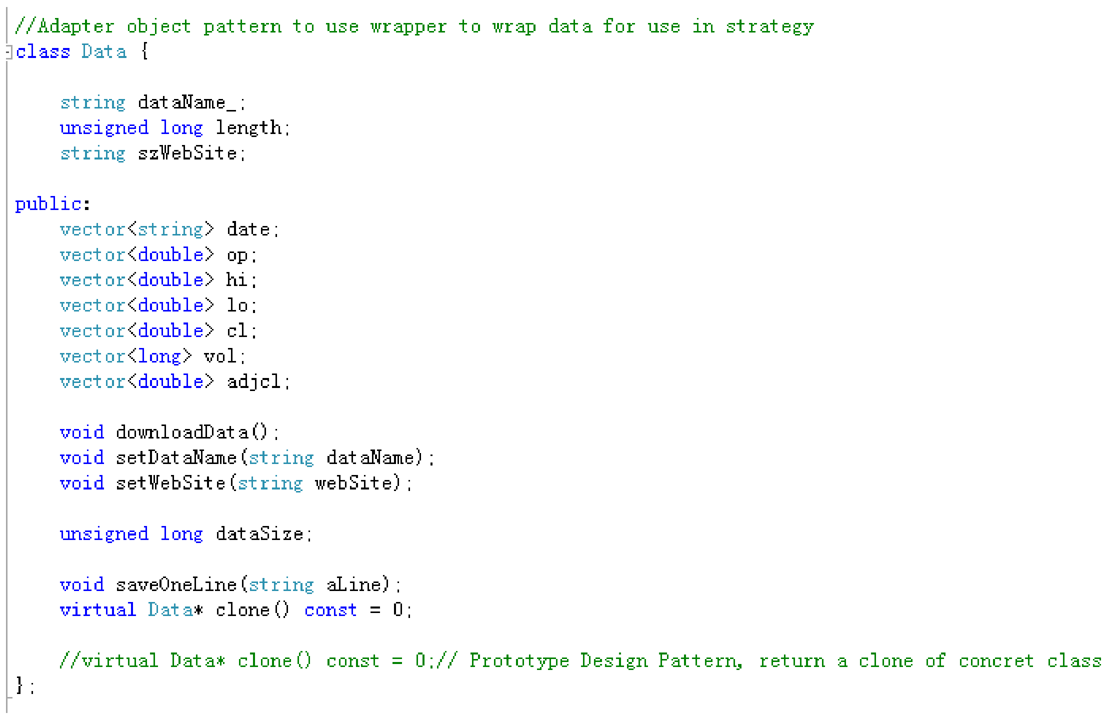




```cpp
class Strategy {

protected:

    string strategyName_;
    Wrapper<Data> dataPtr;

public:
    //Adaptor pattern to use data
    Strategy(const Wrapper<Data> &dataPtr);
    string &getName();
    void setName(string name);
    void close(double amount);

    virtual void execution() = 0;
    virtual vector<double> performance() = 0;

};
```

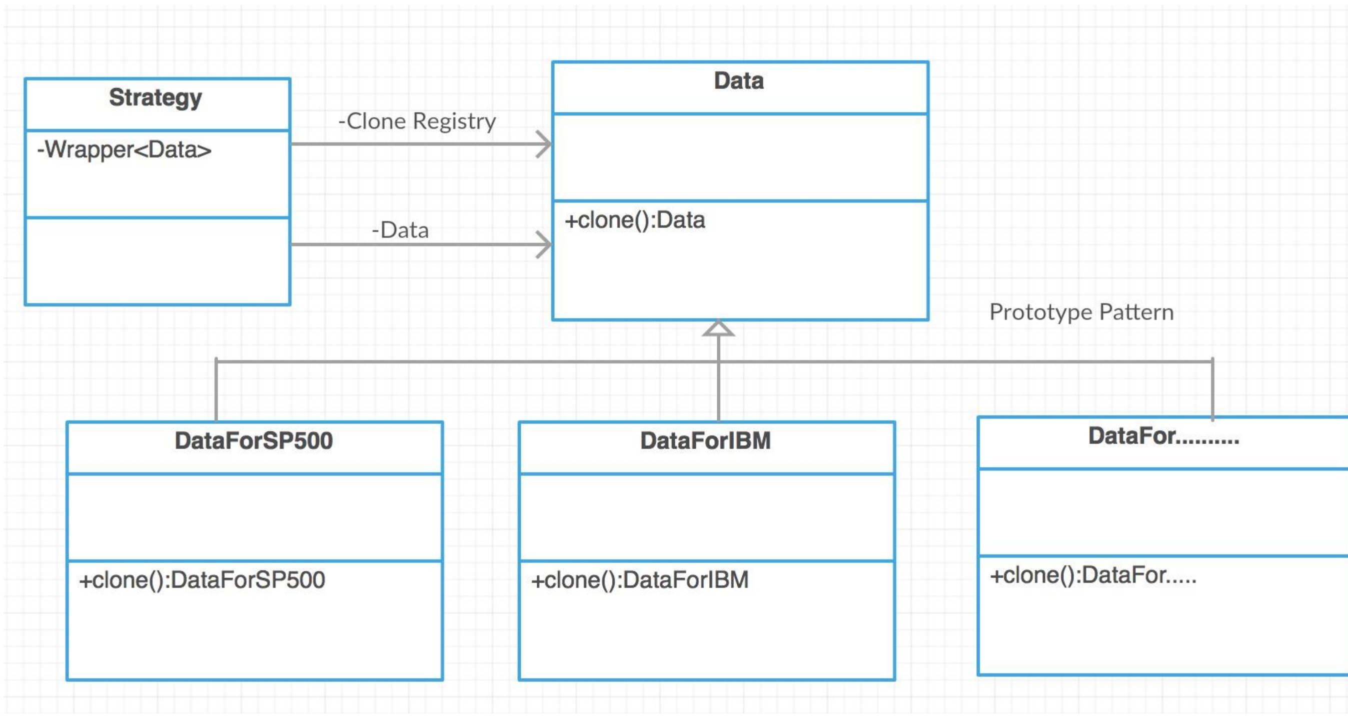


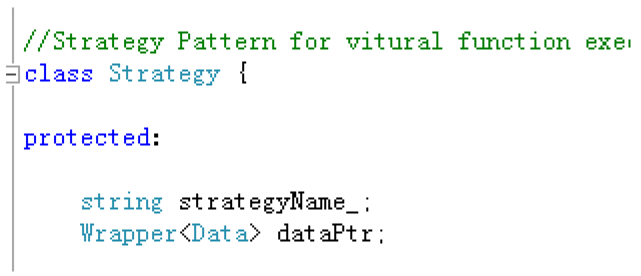
```cpp
//Wrapper to warp data work as bridge/adaptor function
template <class T>
class Wrapper {
    T* ptr_;

public:
    Wrapper(const T & inner) {
        ptr_ = inner.clone();
    }
    Wrapper(const Wrapper & original) {
        if (original.ptr_ != 0) {
            ptr_ = original.ptr_->clone();
        }
        else {
            ptr_ = 0;
        }

    }
    ~Wrapper() {
        delete ptr_;
    }
    Wrapper &operator=(const Wrapper & original) {
        if (this != &original) {
            if (ptr_ != 0) {
                delete ptr_;
                if (original.ptr_ != 0) {
                    ptr_ = original.ptr_->clone();
                }
                else {
                    ptr_ = 0;
                }
            }
            else {
                ptr_ = 0;
            }
        }

        return *this;
    }
    T &operator*() {
        return *ptr_;
    }
    T &operator*() const {
        return *ptr_;
    }
    T* &operator->() {
        return ptr_;
    }
    T* const &operator->() const {
        return ptr_;
    }
```



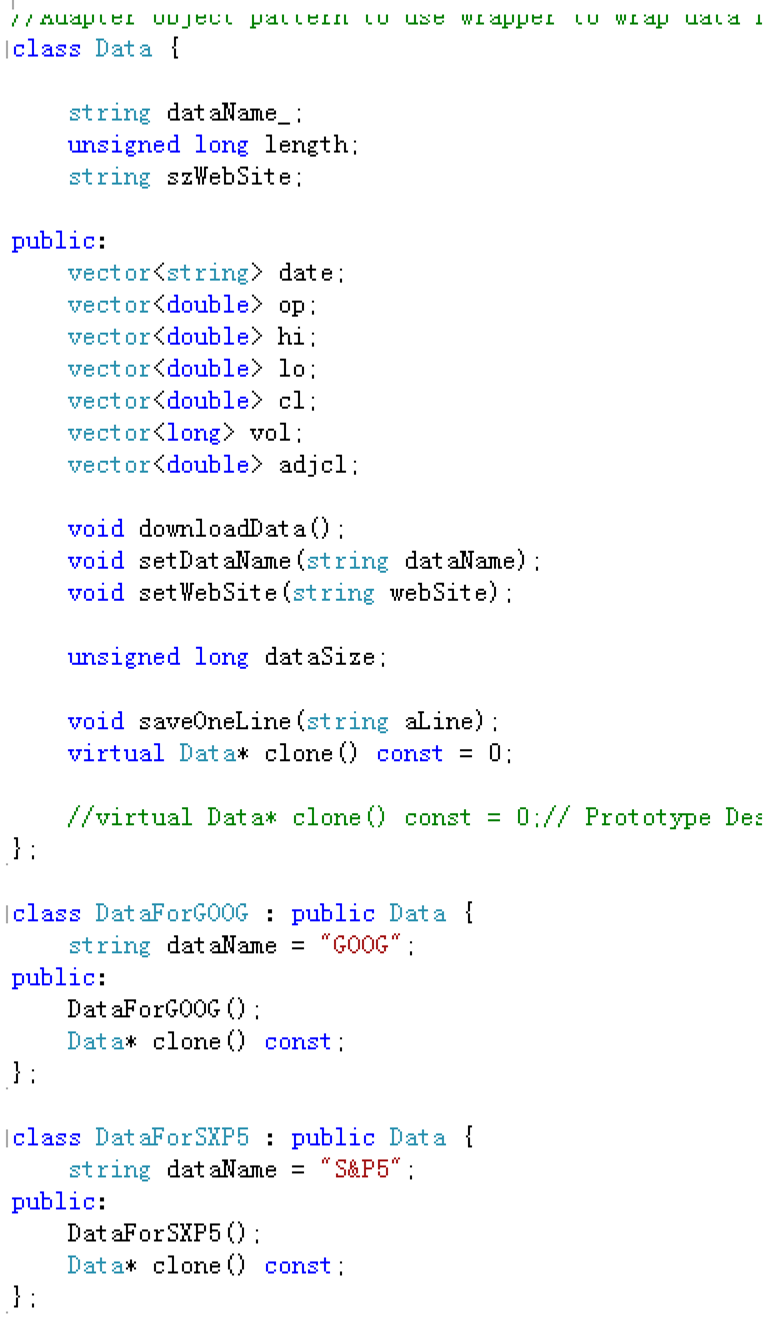
```
//Adapter object pattern to use wrapper to wrap data for use in strategy
class Data {

    string dataName_;
    unsigned long length;
    string szWebSite;

public:
    vector<string> date;
    vector<double> op;
    vector<double> hi;
    vector<double> lo;
    vector<double> cl;
    vector<long> vol;
    vector<double> adjcl;

    void downloadData();
    void setDataName(string dataName);
    void setWebSite(string webSite);

    unsigned long dataSize;

    void saveOneLine(string aLine);
    virtual Data* clone() const = 0;

    //virtual Data* clone() const = 0;// Prototype Design Pattern, return a clone of concret class
};
```

4. Prototype Pattern



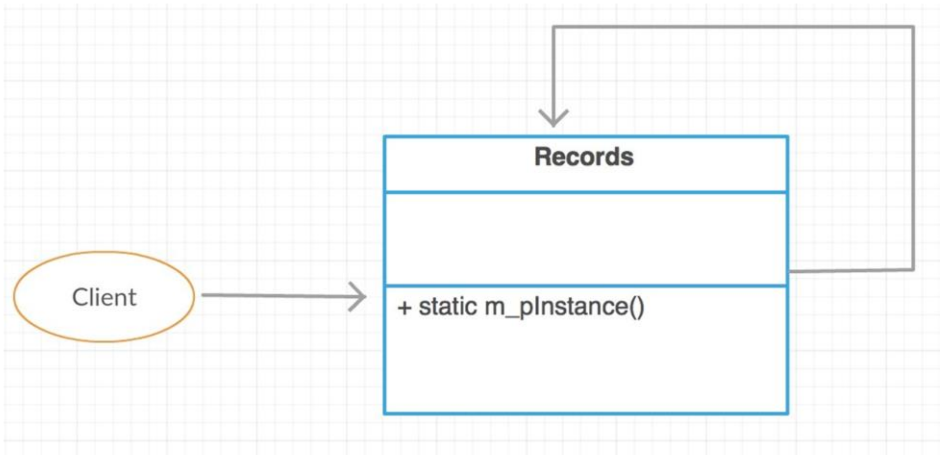


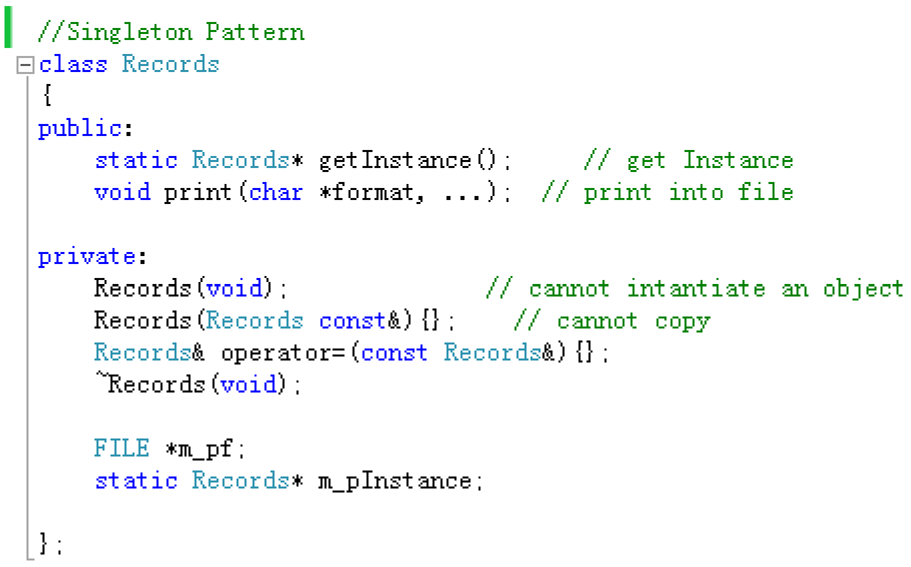
```
//Strategy Pattern for vitural function exe
class Strategy {

protected:

     string strategyName_;
     Wrapper<Data> dataPtr;
```



```
//Adapter object pattern to use wrapper to wrap data i
class Data {

     string dataName_;
     unsigned long length;
     string szWebSite;

public:
     vector<string> date;
     vector<double> op;
     vector<double> hi;
     vector<double> lo;
     vector<double> cl;
     vector<long> vol;
     vector<double> adjcl;

     void downloadData();
     void setDataName(string dataName);
     void setWebSite(string webSite);

     unsigned long dataSize;

     void saveOneLine(string aLine);
     virtual Data* clone() const = 0;

     //virtual Data* clone() const = 0;// Prototype Des
};

class DataForGOOG : public Data {
     string dataName = "GOOG";
public:
     DataForGOOG();
     Data* clone() const;
};

class DataForSXP5 : public Data {
     string dataName = "S&P5";
public:
     DataForSXP5();
     Data* clone() const;
};
```

5. Singleton Pattern



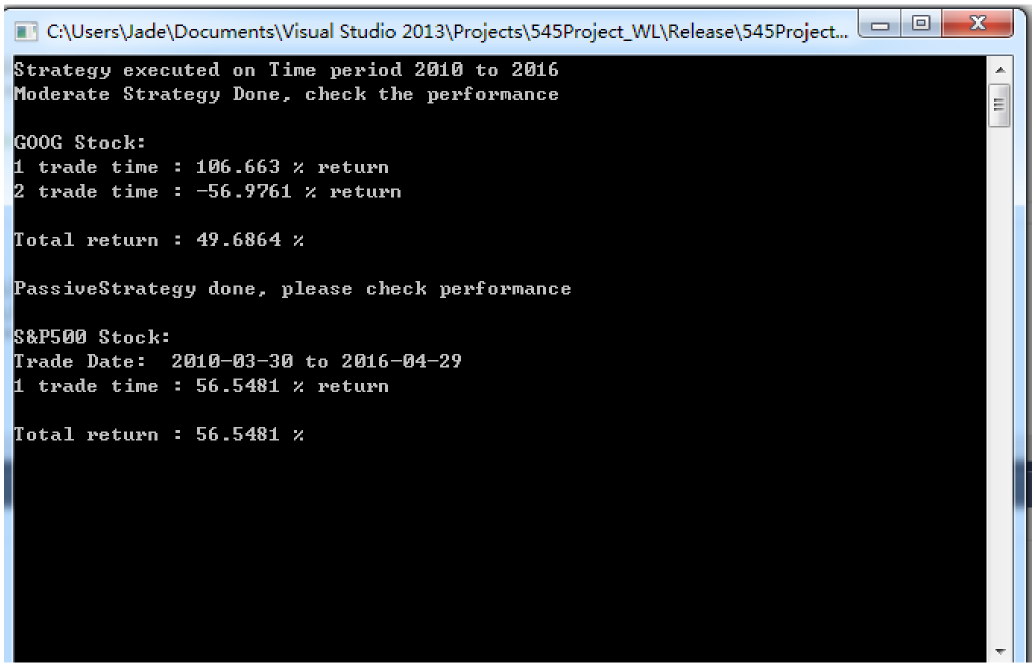


```
//Singleton Pattern
class Records
{
public:
    static Records* getInstance();      // get Instance
    void print(char *format, ...);  // print into file

private:
    Records(void);                 // cannot intantiate an object
    Records(Records const&){};     // cannot copy
    Records& operator=(const Records&){};
    ~Records(void);

    FILE *m_pf;
    static Records* m_pInstance;

};
```

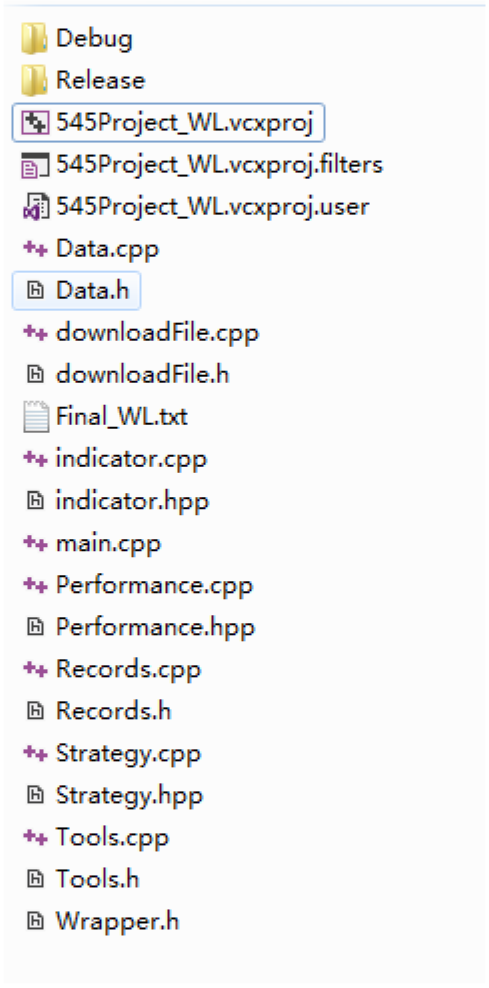
Output Results

Executed screenshot results as blow

```
int main() {

    //For Moderate Strategy
     cout << "Strategy executed on Time period 2010 to 2016 " << endl;
     DataForGOOG GOOG;
     ModerateStrategy  ms(GOOG);
     Performance *m = new Performance(ms);
     m->execution();
     cout << "GOOG Stock:" << endl;
     vector<double> j;
     j = m->performance();

     //Created file to record moderate results
     Records::getInstance()->print("Moderate Startegy");
     for (int i = 0; i < j.size(); ++i){
     Records::getInstance()->print("the trade %i ", i + 1);
     Records::getInstance()->print("time of the return is  %g  ", j[i]);
     }
     cout << endl;


     cout << endl;
     //For Passive Strategy
     DataForSXP5 SP5;
     PassiveStrategy ps(SP5);
     Performance *p = new Performance(ps);
     p->execution();
     cout << "S&P500 Stock:" << endl;
     vector<double> k;
     k = p->performance();

     //Create file to record passive strategy results
     Records::getInstance()->print("\n\n");
     Records::getInstance()->print("Passive Startegy ");
     for (int i = 0; i < k.size(); ++i){

         Records::getInstance()->print("the trade %i ", i + 1);
         Records::getInstance()->print("time of the return is  %g  ", k[i]);
     }
     cout << endl;
     int a;
     cin >> a;

}
```


C:\Users\Jade\Documents\Visual Studio 2013\Projects\545Project_WL\Release\545Project...

```
Strategy executed on Time period 2010 to 2016
Moderate Strategy Done, check the performance

GOOG Stock:
1 trade time : 106.663 % return
2 trade time : -56.9761 % return

Total return : 49.6864 %

PassiveStrategy done, please check performance

S&P500 Stock:
Trade Date:  2010-03-30 to 2016-04-29
1 trade time : 56.5481 % return

Total return : 56.5481 %
```



```
Debug
Release
545Project_WL.vcxproj
545Project_WL.vcxproj.filters
545Project_WL.vcxproj.user
Data.cpp
Data.h
downloadFile.cpp
downloadFile.h
Final_WL.txt
indicator.cpp
indicator.hpp
main.cpp
Performance.cpp
Performance.hpp
Records.cpp
Records.h
Strategy.cpp
Strategy.hpp
Tools.cpp
Tools.h
Wrapper.h
```

Conclusion & Improvement

1) For the strategy: since strategy sets a stop loss, not too much loss happens, but from the results we can see for the moderate strategy, the trade signal is very limited within the five years, it’s still not very practical in use. We can improve logic on further execution. But from the results, passive strategy has a great return which at some point demonstrates a good market performance and the efficiency and non-arbitrage theory.

2) For the pattern: since the data download field is the prototype pattern, it’s still not very convenient to switch strategy test cross tickers and date, it can be improved in further ; Also for the strategy pattern, the back test class has not been really implemented, it can also be implemented practically.