策略模式定义了算法家族，分别封装起来，让它们之间可以相互替换，此模式让算法的变化，不会影响道使用算法的客户。

```
//
//  T2_20190109.hpp
//  DesignPattern
//
//  Created by shadot on 2019/1/9.
//  Copyright © 2019 shadot. All rights reserved.
//

//策略模式

#ifndef T2_20190109_hpp
#define T2_20190109_hpp

#include <iostream>

using namespace std;

//定义算法的基类公共接口，子类需重新实现
class Strategy{
public:
    Strategy(){}

    virtual void AlgorithmInterface() = 0;
};

//具体算法A的实现类
class ConcreteStrategyA : public Strategy{
public:
    ConcreteStrategyA():Strategy(){}

    void AlgorithmInterface(){
        cout << "算法A:Add的实现！" << endl;
    }
};

//具体算法B的实现类
class ConcreteStrategyB : public Strategy{
public:
    ConcreteStrategyB():Strategy(){}

    void AlgorithmInterface(){
```

```
42          cout << "算法B:Sub的实现! " << endl;
43      }
44  };
45
46  //具体算法A的实现类
47  class ConcreteStrategyC : public Strategy{
48  public:
49      ConcreteStrategyC():Strategy(){}
50
51      void AlgorithmInterface(){
52          cout << "算法C:Mul的实现! " << endl;
53      }
54  };
55
56  //具体算法D的实现类
57  class ConcreteStrategyD : public Strategy{
58  public:
59      ConcreteStrategyD():Strategy(){}
60
61      void AlgorithmInterface(){
62          cout << "算法D:Div的实现! " << endl;
63      }
64  };
65
66  //其他算法。。。
67
68  //用来维护对算法对象Strategy的使用
69  class Context
70  {
71  public:
72      enum EMOperate {Add, Sub, Mul, Div};
73
74      //初始化时传入算法标示，内部创建具体的算法
75      Context(EMOperate op){
76          switch (op) {
77              case Add:   m_pObj = new ConcreteStrategyA();   break;
78              case Sub:   m_pObj = new ConcreteStrategyB();   break;
79              case Mul:   m_pObj = new ConcreteStrategyC();   break;
80              case Div:   m_pObj = new ConcreteStrategyD();   break;
81
82              default:    break;
83          }
84      }
85
86      //调用具体的算法
87      void ContextInterface()
```

```
88      {
89          m_pObj->AlgorithmInterface();
90      }
91
92      Strategy* m_pObj;
93  };
94
95  #endif /* T2_20190109_hpp */
96
97  int main(int argc, const char * argv[]) {
98
99          Context context1(Context::Add);
100         context1.ContextInterface();
101
102         Context context2(Context::Sub);
103         context2.ContextInterface();
104
105         Context context3(Context::Mul);
106         context3.ContextInterface();
107
108         Context context4(Context::Div);
109         context4.ContextInterface();
110
111     return 0;
112 }
113
```