```cpp
//
//  main.cpp
//  DesignPattern
//
//  Created by shadot on 2019/1/8.
//  Copyright © 2019 shadot. All rights reserved.
//

#include <iostream>
using namespace std;

//运算的抽象基类
class Operation{
public:
    Operation(){
        m_numberX = 0.0;
        m_numberY = 0.0;
    }

    void Setxy(double x, double y){
        m_numberX = x;
        m_numberY = y;
    }

    virtual double GetResult() = 0;

protected:
    double m_numberX;
    double m_numberY;
};

//加减乘除具体实现类
class OperationAdd : public Operation{
public:
    OperationAdd():Operation(){}

    double GetResult(){
        return m_numberX + m_numberY;
    }
};

class OperationSub : public Operation{
```

```cpp
public:
    OperationSub():Operation(){}

    double GetResult(){
        return m_numberX - m_numberY;
    }
};

class OperationMul : public Operation{
public:
    OperationMul():Operation(){}

    double GetResult(){
        return m_numberX * m_numberY;
    }
};

class OperationDiv : public Operation{
public:
    OperationDiv():Operation(){}

    double GetResult(){
        if (m_numberY == 0)
            return 0;
        return m_numberX / m_numberY;
    }
};

//简单运算工厂类
class OperationFactory
{
public:
    enum EMOperate {Add, Sub, Mul, Div};

    OperationFactory(){}

    Operation* CreateOperate(EMOperate operate){
        Operation* pOper = nullptr;

        switch (operate) {
            case Add:
                pOper = new OperationAdd(); break;
            case Sub:
                pOper = new OperationSub(); break;
            case Mul:
                pOper = new OperationMul(); break;
            case Div:
                pOper = new OperationDiv(); break;
            default: break;
        }
        return pOper;
    }
};

int main(int argc, const char * argv[]) {

    Operation* pOper =                        OperationFactory().CreateOperate(OperationFa
    pOper->Setxy(1, 2);
    auto result = pOper->GetResult();

    return 0;
}

```

简单运算工厂负责如何去实例化对象，以后如果需要再次进行扩展，只是在基于Operation类进行派生，然后重新实现GetResult()方法，再在工厂类switch中添加分支即可