```cpp
#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>

#include <cassert>
#include <cmath>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics.hpp>
#include <boost/shared_ptr.hpp>

#include "Path.h"
#include "TimeGrid.h"
#include "IContract.h"
#include "IModel.h"
#include "BlackScholes.h"
#include "EuropeanOption.h"
#include "Payoff.h"

//Path createOnePath(TimeGrid& timeGrid)
//{
//    Path path(timeGrid);
//    return path;
//}
//
bool doubleEqual(double a, double b, int effectiveOrder)
{
    const int aint = a * std::pow(10, effectiveOrder);
    const int bint = b * std::pow(10, effectiveOrder);
    return aint == bint;
}

double discount(const double payoff, const double discountFactor)
{
    return payoff * discountFactor;
}


int main()
{

    // for unit tests
    CPPUNIT_NS::TestResult controller;

    CPPUNIT_NS::TestResultCollector result;
    controller.addListener(&result);

    CPPUNIT_NS::BriefTestProgressListener progress;
    controller.addListener(&progress);

    CPPUNIT_NS::TestRunner runner;
    runner.addTest(CppUnit::TestFactoryRegistry::getRegistry().makeTest());
    runner.run(controller);

    CPPUNIT_NS::CompilerOutputter outputter(&result, CPPUNIT_NS::stdCOut());
    outputter.write();



    // for combination tests

    const double strike = 100.0;
    const double maturity = 1.0;
    const double spot = 100.0;
    const double volatility = 0.2;
    const double interestRate = 0.06;
    const std::size_t numberOfPaths = 100;
    const std::size_t timesteps = 10;
    const double drift = interestRate - 0.5 * volatility * volatility;

    mctr::TimeGrid timeGrid(timesteps);
    std::cout << timeGrid(1) << std::endl;
    boost::shared_ptr<mctr::IModel> model(new mctr::BlackScholes(drift, volatility))
;
    boost::shared_ptr<mctr::IContract> europeanCall(
        new mctr::EuropeanOption(strike, maturity, mctr::Payoff::call));

    double price = 0.0;
    {
        using namespace boost::accumulators;
        // accumulator is used to store each discounted payoffs
        accumulator_set<double, stats<tag::mean, tag::variance> > accumulator;

        // create one path
        for (std::size_t i = 0; i < numberOfPaths; ++i) {
            boost::shared_ptr<mctr::Path> path = model->createOnePath(timeGrid);
            double payoff = europeanCall->calculatePayoff(path);
            const double discountFactor = std::exp( - interestRate * maturity);

            double discountedPayoff = discount(payoff, discountFactor);
            accumulator(discountedPayoff);
        }

    price = mean(accumulator);


    }
    //price = 9.3846;

    assert(doubleEqual(price, 9.3846, 5));

    return 0;
}
```