```cpp
#include "Examples/Examples.hpp"
#include "Examples/ExamplesFunctions.hpp"

using namespace cfl;

class IndUpDownOut: public IResetValues
{
public:
  IndUpDownOut(double dLowerBarrier, double dUpperBarrier,
               const AssetModel & rModel)
    :m_dLowerBarrier(dLowerBarrier), m_dUpperBarrier(dUpperBarrier),
     m_rModel(rModel)
  {
    ASSERT(dLowerBarrier < dUpperBarrier);
  }

  Slice resetValues(unsigned iTime, double dBeforeReset) const
  {
    return dBeforeReset*indicator(m_rModel.spot(iTime), m_dLowerBarrier)*
      indicator(m_dUpperBarrier, m_rModel.spot(iTime));
  }
private:
  double m_dLowerBarrier;
  double m_dUpperBarrier;
  const AssetModel & m_rModel;
};

cfl::PathDependent prb::
indUpDownOut(double dLowerBarrier, double dUpperBarrier,
             const std::vector<unsigned> & rBarrierIndexes,
             const AssetModel & rModel)
{
  double dInitialValue = 1.;
  return PathDependent(new IndUpDownOut(dLowerBarrier, dUpperBarrier, rModel),
                       rBarrierIndexes, dInitialValue);
}

cfl::MultiFunction prb::
barrierUpOrDownAndOut_path(double dNotional, double dLowerBarrier,
                           double dUpperBarrier,
                           const std::vector<double> & rBarrierTimes,
                           AssetModel & rModel)
{
  PRECONDITION(rModel.initialTime() < rBarrierTimes.front());
  PRECONDITION(dLowerBarrier < dUpperBarrier);

  std::vector<double> uEventTimes(rBarrierTimes);
  uEventTimes.insert(uEventTimes.begin(), rModel.initialTime());
  rModel.assignEventTimes(uEventTimes);

  std::vector<unsigned> uResetIndexes(rBarrierTimes.size(), 1);
  std::transform(uResetIndexes.begin(), uResetIndexes.end()-1,
                 uResetIndexes.begin()+1,
                 [](unsigned iX){ return iX+1; });
  int iInd = rModel.addState(indUpDownOut(dLowerBarrier, dUpperBarrier,
                                          uResetIndexes, rModel));

  int iTime = rModel.eventTimes().size()-1;
  Slice uOption = dNotional*rModel.state(iTime, iInd);
  uOption.rollback(0);
  return interpolate(uOption, iInd);
}
```

```cpp
#include "Examples/Examples.hpp"
#include "Examples/ExamplesFunctions.hpp"

using namespace cfl;

class HistAverage: public IResetValues
{
public:
  HistAverage(const std::vector<unsigned> & rAverIndexes,
              const AssetModel & rModel)
    :m_uAverIndexes(rAverIndexes), m_rModel(rModel)
  {}

  Slice resetValues(unsigned iTime, double dBeforeReset) const
  {
    PRECONDITION(std::binary_search(m_uAverIndexes.begin(),
                                    m_uAverIndexes.end(), iTime));
    //number of averaging times including the current one
    int iTimes = std::lower_bound(m_uAverIndexes.begin(),
                                  m_uAverIndexes.end(), iTime) -
      m_uAverIndexes.begin() + 1;
    return ((iTimes-1)*dBeforeReset  + m_rModel.spot(iTime))/iTimes;
  }

private:
  std::vector<unsigned> m_uAverIndexes;
  const AssetModel & m_rModel;
};

cfl::PathDependent prb::
histAverage(const std::vector<unsigned> & rAverIndexes,
            const AssetModel & rModel)
{
  double dInitialValue = 0.;
  return PathDependent(new HistAverage(rAverIndexes, rModel),
                       rAverIndexes, dInitialValue);
}

cfl::MultiFunction prb::
asianCall(const std::vector<double> & rAverTimes, double dStrike,
          double dMaturity, AssetModel & rModel)
{
  PRECONDITION(rModel.initialTime() < rAverTimes.front());
  PRECONDITION(rAverTimes.back() < dMaturity);

  std::vector<double> uEventTimes(1, rModel.initialTime());
  uEventTimes.insert(uEventTimes.end(), rAverTimes.begin(),
                     rAverTimes.end());
  uEventTimes.push_back(dMaturity);
  rModel.assignEventTimes(uEventTimes);

  std::vector<unsigned> uResetIndexes(rAverTimes.size(), 1);
  std::transform(uResetIndexes.begin(), uResetIndexes.end()-1,
                 uResetIndexes.begin()+1,
                 [](unsigned iX){ return iX+1; });
  int iHistAverage = rModel.addState(histAverage(uResetIndexes, rModel));

  int iTime = rModel.eventTimes().size()-1;
  Slice uOption = max(rModel.state(iTime, iHistAverage)-dStrike, 0.);
  uOption.rollback(0);
  return interpolate(uOption, iHistAverage);
}
```

```cpp
#include "Examples/Examples.hpp"
#include "Examples/ExamplesFunctions.hpp"

using namespace cfl;

class SavingsAccountNextTime: public IResetValues
{
public:
  SavingsAccountNextTime(double dPeriod, const InterestRateModel & rModel)
    :m_dPeriod(dPeriod), m_rModel(rModel)
  {};

  Slice resetValues(unsigned iTime, double dBeforeReset) const
  {
    return dBeforeReset/
      m_rModel.discount(iTime, m_rModel.eventTimes()[iTime]+m_dPeriod);
  }
private:
  double m_dPeriod;
  const InterestRateModel & m_rModel;
};

cfl::PathDependent prb::
savingsAccountNextTime(double dPeriod, double dNotional,
                       const std::vector<unsigned> & rResetTimes,
                       const cfl::InterestRateModel & rModel)
{
  return PathDependent(new SavingsAccountNextTime(dPeriod, rModel),
                       rResetTimes, dNotional);
}


cfl::MultiFunction prb::
savingsAccount(double dPeriod, unsigned iNumberOfPeriods,
               double dNotional, InterestRateModel & rModel)
{
  std::vector<double> uEventTimes(iNumberOfPeriods+1, rModel.initialTime());
  std::transform(uEventTimes.begin(), uEventTimes.end()-1,
                 uEventTimes.begin()+1,
                 [dPeriod](double dX){ return dX+dPeriod; });
  rModel.assignEventTimes(uEventTimes);

  std::vector<unsigned> uResetIndexes(uEventTimes.size(), 0);
  std::transform(uResetIndexes.begin(), uResetIndexes.end()-1,
                 uResetIndexes.begin()+1,
                 [](unsigned iX){ return iX+1; });
  int iState = rModel.addState(savingsAccountNextTime(dPeriod, dNotional,
                                                      uResetIndexes, rModel));
  int iTime = uEventTimes.size()-1;
  //the value of the savings account at maturity
  Slice uSavingsAccount = rModel.state(iTime, iState)
    * rModel.discount(iTime, rModel.eventTimes()[iTime]+dPeriod);
  uSavingsAccount.rollback(0);

  return interpolate(uSavingsAccount, iState);
}
```

```cpp
#include "Examples/Examples.hpp"
#include "Examples/ExamplesFunctions.hpp"

using namespace cfl;


cfl::MultiFunction prb::
putOnSavingsAccount(const Data::CashFlow & rAccount,
                    InterestRateModel & rModel)
{
  std::vector<double>
    uEventTimes(rAccount.numberOfPayments, rModel.initialTime());
  std::transform(uEventTimes.begin(), uEventTimes.end()-1,
                 uEventTimes.begin()+1,
                 [&rAccount](double dX){ return dX+rAccount.period; });
  rModel.assignEventTimes(uEventTimes);

  std::vector<unsigned> uResetIndexes(uEventTimes.size(), 0);
  std::transform(uResetIndexes.begin(), uResetIndexes.end()-1,
                 uResetIndexes.begin()+1,
                 [](unsigned iX){ return iX+1; });
  int iState =
    rModel.addState(savingsAccountNextTime(rAccount.period, rAccount.notional,
                                           uResetIndexes, rModel));
  //last payment time before maturity
  int iTime = uEventTimes.size()-1;
  //value of savings account at maturity
  Slice uSavingsAccount = rModel.state(iTime, iState);
  //value of fixed payment at maturity
  double dFixedPayment = rAccount.notional;
  for (unsigned iI=0; iI<rAccount.numberOfPayments; iI++) {
    dFixedPayment += (dFixedPayment*rAccount.rate*rAccount.period);
  }
  double dTime = rModel.eventTimes()[iTime];
  Slice uOption = max(dFixedPayment - uSavingsAccount, 0.)
    * rModel.discount(iTime, dTime + rAccount.period);
  uOption.rollback(0);
  return interpolate(uOption, iState);
}
```