```cpp
#include "Homework4/Homework4.hpp"

using namespace cfl;
using namespace std;

cfl::MultiFunction prb::
    collar(const Data::CashFlow &rCap, double dFloorRate,
           InterestRateModel &rModel)
{
  PRECONDITION(dFloorRate < rCap.rate);

  std::vector<double> uEventTimes(rCap.numberOfPayments);
  uEventTimes.front() = rModel.initialTime();
  std::transform(uEventTimes.begin(), uEventTimes.end() - 1,
                 uEventTimes.begin() + 1,
                 [&rCap](double dX) { return dX + rCap.period; });
  rModel.assignEventTimes(uEventTimes);

  //last minus one payment time
  int iTime = uEventTimes.size() - 1;
  double dCapFactor = 1. + rCap.rate * rCap.period;
  double dFloorFactor = 1. + dFloorRate * rCap.period;
  Slice uDiscount =
      rModel.discount(iTime, rModel.eventTimes()[iTime] + rCap.period);
  //value of cap payment at the next payment time
  Slice uCap = max(1. - uDiscount * dCapFactor, 0.);
  //value of floor payment at the next payment time
  Slice uFloor = max(uDiscount * dFloorFactor - 1., 0.);
  //we multiply on notional at the end
  Slice uOption = uCap - uFloor;

  while (iTime > 0)
  {
    //uOption is the value of future payments
    iTime--;
    uOption.rollback(iTime);
    uDiscount =
        rModel.discount(iTime, rModel.eventTimes()[iTime] + rCap.period);
    uCap = max(1. - uDiscount * dCapFactor, 0.);
    uFloor = max(uDiscount * dFloorFactor - 1., 0.);
    uOption += (uCap - uFloor);
  }
  uOption *= rCap.notional;

  return interpolate(uOption);
}
```

```cpp
#include "Homework4/Homework4.hpp"

using namespace cfl;
using namespace std;

namespace NAmerSwaption
{
  cfl::Slice
  couponBond(unsigned iTime, const Data::CashFlow &rBond,
             const InterestRateModel &rModel)
  {
    Slice uCashFlow = rModel.cash(iTime, 0.);
    double dTime = rModel.eventTimes()[iTime];
    for (unsigned iI = 0; iI < rBond.numberOfPayments; iI++)
    {
      dTime += rBond.period;
      uCashFlow += rModel.discount(iTime, dTime);
    }
    uCashFlow *= (rBond.rate * rBond.period);
    uCashFlow += rModel.discount(iTime, dTime);
    uCashFlow *= rBond.notional;
    return uCashFlow;
  }

  cfl::Slice swap(unsigned iTime, const Data::Swap &rSwap,
                  const InterestRateModel &rModel)
  {
    //assume first that we receive fixed and pay float
    Slice uSwap = couponBond(iTime, rSwap, rModel) - rSwap.notional;
    if (!rSwap.payFloat)
    { //if we pay fixed
      uSwap *= -1;
    }
    return uSwap;
  }
}

using namespace NAmerSwaption;

cfl::MultiFunction prb::
    americanSwaption(const Data::Swap &rSwap,
                     const std::vector<double> &rExerciseTimes,
                     InterestRateModel &rModel)
{
  PRECONDITION(rModel.initialTime() < rExerciseTimes.front());
  PRECONDITION(std::is_sorted(rExerciseTimes.begin(), rExerciseTimes.end(),
                              std::less_equal<double>()));

  std::vector<double> uEventTimes(rExerciseTimes.size() + 1);
  uEventTimes.front() = rModel.initialTime();
  copy(rExerciseTimes.begin(), rExerciseTimes.end(), uEventTimes.begin() + 1);
  rModel.assignEventTimes(uEventTimes);

  int iTime = uEventTimes.size() - 1;
  Slice uOption = rModel.cash(iTime, 0);
  while (iTime > 0)
  {
    //uOption is the value to continue
    uOption = max(swap(iTime, rSwap, rModel), uOption);
    iTime--;
    uOption.rollback(iTime);
  }
  return interpolate(uOption);
}
```

```cpp
#include "Homework4/Homework4.hpp"

using namespace cfl;
using namespace std;

cfl::MultiFunction prb::
    putCallBond(const Data::CashFlow &rBond,
                double dRedemptionPrice, double dRepurchasePrice,
                InterestRateModel &rModel)
{
  PRECONDITION(rBond.numberOfPayments > 0);

  std::vector<double> uEventTimes(rBond.numberOfPayments);
  uEventTimes.front() = rModel.initialTime();
  std::transform(uEventTimes.begin(), uEventTimes.end() - 1,
                 uEventTimes.begin() + 1,
                 [&rBond](double dX) { return dX + rBond.period; });
  rModel.assignEventTimes(uEventTimes);

  int iTime = uEventTimes.size() - 1; //last minus one coupon time
  double dCoupon = rBond.rate * rBond.period;
  Slice uDiscount =
      rModel.discount(iTime, rModel.eventTimes()[iTime] + rBond.period);
  Slice uBond = uDiscount * (1. + dCoupon);

  while (iTime > 0)
  {
    //uBond is the value to continue
    uBond = min(uBond, dRepurchasePrice);
    uBond = max(uBond, dRedemptionPrice);
    uBond += dCoupon;
    iTime--;
    uBond.rollback(iTime);
  }

  uBond *= rBond.notional;
  return interpolate(uBond);
}
```

```cpp
#include "Homework4/Homework4.hpp"

using namespace cfl;
using namespace std;

cfl::MultiFunction prb::
americanPutOnFutures(double dNotional, double dBondMaturity,
                     double dFuturesMaturity,  unsigned iFuturesTimes,
                     double dStrike, InterestRateModel & rModel)
{
  PRECONDITION(dBondMaturity > dFuturesMaturity);

  double dPeriod = (dFuturesMaturity - rModel.initialTime())/(iFuturesTimes);
  std::vector<double> uEventTimes(iFuturesTimes + 1);
  uEventTimes.front() = rModel.initialTime();
  std::transform(uEventTimes.begin(), uEventTimes.end()-1,
                 uEventTimes.begin()+1,
                 [dPeriod](double dX){ return dX+dPeriod; });
  ASSERT(std::abs(uEventTimes.back()- dFuturesMaturity) < cfl::EPS);
  rModel.assignEventTimes(uEventTimes);

  int iTime = rModel.eventTimes().size()-1;
  Slice uFutures = dNotional*rModel.discount(iTime, dBondMaturity);
  Slice uPut = rModel.cash(iTime,0.);
  while (iTime > 0) {
    //uPut is value to continue
    uPut = max(uPut, dStrike - uFutures);
    iTime--;
    uFutures.rollback(iTime);
    uFutures /= rModel.discount(iTime, rModel.eventTimes()[iTime] + dPeriod);
    uPut.rollback(iTime);
  }

  return interpolate(uPut);
}
```