```cpp
#include <numeric>
#include "Homework1/Homework1.hpp"
#include "cfl/Macros.hpp"
#include <cmath>

using namespace cfl;
using namespace prb;

Function prb::volatilityVar(const Function &rVar, double dInitialTime)
{
  std::function<double(double)> uVol =
    [dInitialTime, rVar](double dT) {
      PRECONDITION(dT >= dInitialTime);
      double dX = std::max<double>(dT - dInitialTime, cfl::EPS);
      return std::sqrt(rVar(dInitialTime + dX) / dX);
    };

  std::function<bool(double)> uVolBelongs =
    [rVar](double dT) {
      return rVar.belongs(dT);
    };

  return Function(uVol, uVolBelongs);
}
```

```cpp
#include <numeric>
#include "Homework1/Homework1.hpp"
#include "cfl/Macros.hpp"
#include <cmath>

using namespace cfl;
using namespace std;

cfl::Function prb::carryBlack(double dTheta, double dLambda,
                              double dSigma, double dInitialTime)
{
  PRECONDITION((dLambda>=0) && (dSigma>=0));

  std::function<double(double)> uY =
    [dTheta, dLambda, dSigma, dInitialTime](double dT) {
      double dX = std::max(dLambda*(dT-dInitialTime), cfl::EPS);
      double dB = (1.-std::exp(-dX))/dX;
      double dC = (1. - std::exp(-2.*dX))/(2.*dX);
      double dY = dTheta*dB + 0.5*dSigma*dSigma*dC;
      return dY;
    };

  return Function(uY, dInitialTime);
}
```

```cpp
#include <numeric>
#include "Homework1/Homework1.hpp"
#include "cfl/Macros.hpp"
#include <cmath>

using namespace cfl;
using namespace std;

cfl::Function prb::yieldSvensson(double dC0, double dC1, double dC2, double dC3,
                                 double dLambda1, double dLambda2, double dInitialTime)
{
  PRECONDITION(std::min(dLambda1, dLambda2) >= 0);
  std::function<double(double)> uY =
    [dC0, dC1, dC2, dC3, dLambda1, dLambda2, dInitialTime](double dT) {
      double dF1 = max<double>(dLambda1 * (dT - dInitialTime),cfl::EPS);

      double dF2 = max<double>(dLambda2 * (dT - dInitialTime),cfl::EPS);

      double  dY = dC0 + dC1 * (1. - std::exp(-dF1)) / dF1 + dC2 * (1. - (1. + dF1) * std::exp(-d
F1)) / dF1;
      dY += dC3 * (1. - (1. + dF2) * std::exp(-dF2)) / dF2;
      return dY;
    };

  return Function(uY, dInitialTime);
}
```

```cpp
#include "Homework1/Homework1.hpp"
#include "cfl/Macros.hpp"
#include <cmath>

using namespace cfl;
using namespace std;

cfl::Function prb::
forwardCouponBond(double dRate, double dPeriod, double dMaturity,
                  const cfl::Function &rDiscount,
                  double dInitialTime, bool bClean)
{
  PRECONDITION(dMaturity > dInitialTime);

  std::function<double(double)> uF =
    [dRate, dPeriod, dMaturity, rDiscount, bClean](double dT) {
      double dPayTime = dMaturity;
      double dSum = 0;
      while (dPayTime > dT)
        {
          dSum += rDiscount(dPayTime);
          dPayTime -= dPeriod;
        }
      double dPayment = dRate * dPeriod;
      dSum *= dPayment;
      dSum += rDiscount(dMaturity);
      double dF = dSum / rDiscount(dT);
      if (bClean)
        {
          dF -= dRate * (dT - dPayTime);
        }
      return dF;
    };

  return Function(uF, dInitialTime, dMaturity);
}
```