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

using namespace cfl;
using namespace cfl::Data;

class YieldShapeHW: public cfl::IFunction
{
public:
  YieldShapeHW(double dLambda, double dInitialTime)
  {
    PRECONDITION(dLambda >=0);
    m_dLambda = dLambda;
    m_dInitialTime = dInitialTime;
  }

  double operator()(double dT) const
  {
    PRECONDITION(belongs(dT));
    double dF = m_dLambda*(dT - m_dInitialTime);
    if (dF< cfl::c_dEps) {
      return 1.;
    }
    double dYield = (1.-std::exp(-dF))/dF;
    return dYield;
  }

  bool belongs(double dT) const {
    return (dT >= m_dInitialTime);
  }
private:
  double m_dInitialTime, m_dLambda;
};

cfl::Function prb::
yieldShapeHullWhite(double dLambda, double dInitialTime)
{
  return cfl::Function(new YieldShapeHW(dLambda, dInitialTime));
}
```

```cpp
#include "Examples/Examples.hpp"
#include "cfl/Interp.hpp"

using namespace cfl;

cfl::Function prb::
discountLogLinearInterp(const std::vector<double> & rTimes,
                        const std::vector<double> & rDiscount,
                        double dInitialTime)
{
  PRECONDITION(rTimes.size() == rDiscount.size());
  PRECONDITION(rTimes.size()>0);
  PRECONDITION(rTimes.front()>dInitialTime);
  PRECONDITION(std::equal(rTimes.begin()+1, rTimes.end(), rTimes.begin(),
                          std::greater<double>()));

  //times for interpolation = initial time + discount times
  std::vector<double> uTimes(rTimes);
  uTimes.insert(uTimes.begin(),dInitialTime);

  //logarithm of discount factors
  std::vector<double> uLogDiscount(uTimes.size());
  uLogDiscount.front() = 0.;
  std::transform(rDiscount.begin(), rDiscount.end(), uLogDiscount.begin()+1,
                 [](double dX){ return std::log(dX); });

  //linear interpolation of the logarithm of discount factors
  cfl::Interp uLinear = NInterp::linear();
  Function uLogDiscountFunction =
    uLinear.interpolate(uTimes.begin(), uTimes.end(), uLogDiscount.begin());

  return cfl::exp(uLogDiscountFunction);
}
```

```cpp
#include <numeric>
#include "Examples/Examples.hpp"

using namespace cfl;

class Yield: public std::function<double(double, double)>
{
public:
  Yield(double dInitialTime)
    :m_dInitialTime(dInitialTime)
  {}
  double operator()(double dTime, double dDiscount) const
  {
    PRECONDITION(dTime > m_dInitialTime+cfl::c_dEps);
    double dYield = - std::log(dDiscount)/(dTime - m_dInitialTime);
    return dYield;
  }
private:
  double m_dInitialTime;
};

cfl::Function prb::
discountFitHullWhite(const std::vector<double> & rTimes,
                     const std::vector<double> & rDiscount,
                     double dLambda, double dInitialTime)
{
  PRECONDITION(rTimes.size() == rDiscount.size());
  PRECONDITION(rTimes.size()>0);
  PRECONDITION(rTimes.front()>dInitialTime);
  PRECONDITION(std::equal(rTimes.begin()+1, rTimes.end(), rTimes.begin(),
                          std::greater<double>()));

  std::vector<double> uYield(rTimes.size());
  std::transform(rTimes.begin(), rTimes.end(), rDiscount.begin(),
                 uYield.begin(), Yield(dInitialTime));

  Function uYieldShape = yieldShapeHullWhite(dLambda, dInitialTime);

  std::vector<double> uA(rTimes.size());
  std::transform(rTimes.begin(), rTimes.end(), uA.begin(),uYieldShape);

  double dCov =
    std::inner_product(uYield.begin(), uYield.end(), uA.begin(), 0.);
  double dVar = std::inner_product(uA.begin(), uA.end(), uA.begin(),0.);

  double dA = dCov/dVar;

  return cfl::Data::discount(dA*uYieldShape,dInitialTime);
}
```