Help

```
#include <stdlib.h>
#include  "
href../../../mod/bs1d/bs1d_lim/bs1d_lim_h_src.pdfbs1d_lim.h"
#include "
href../../../common/error_msg_h_src.pdferror_msg.h"

static int Ritchken_95_DownIn(int am, double s, NumFunc_1 *p, double rebate, dou
{
  int i, j, npoints, eta0;
  double h, pu, pm, pd, z, u, d, stock, upperstock, eta, lambda, price, delta;
  double *P, *iv;

  /*Price, intrisic value arrays*/
  npoints = 2 * N + 1;
  P = malloc(npoints * sizeof(double));
  if (P == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  iv = malloc(npoints * sizeof(double));
  if (iv == NULL)
    return MEMORY_ALLOCATION_FAILURE;
  /*Up and Down factors*/

  h = t / (double) N;
  eta = log(s / l) / (sigma * sqrt(h));
  eta0 = (int) floor(eta);

  lambda = eta / (double)eta0;
  if (eta0 > N)
    {
      eta0 = N;
      lambda = 1.22474;
    }
  u = exp(lambda * sigma * sqrt(h));
  d = 1. / u;

  /*Discounted Probability*/
  z = (r - divid) - SQR(sigma) / 2.;
  pu = (1. / (2.*SQR(lambda)) + z * sqrt(h) / (2.*lambda * sigma));
  pm = (1. - 1. / SQR(lambda));
```

```
pd = (1. - pu - pm);
pu *= exp(-r * h);
pm *= exp(-r * h);
pd *= exp(-r * h);

/*Intrisic value initialisation and terminal values*/
upperstock = s;
for (i = 0; i < N; i++)
  upperstock *= u;

stock = upperstock;
for (i = 0; i < N + eta0; i++)
  {
    iv[i] = (p->Compute)(p->Par, stock);
    P[i] = rebate;
    stock *= d;
  }

npoints = N + eta0;
if ((p->Compute) == &Call)
  pnl_cf_call_bs(l, p->Par[0].Val.V_PDOUBLE, 0., r, divid, sigma, &price, &del
else
  pnl_cf_put_bs(l, p->Par[0].Val.V_PDOUBLE, 0., r, divid, sigma, &price, &delt
P[npoints] = price;

/*Backward Resolution*/
for (i = 1; i <= N - eta0; i++)
  {
    npoints -= 1;
    for (j = 0; j < npoints; j++)
      {
        P[j] = pu * P[j] + pm * P[j + 1] + pd * P[j + 2];
        if (am)
          P[j] = MAX(iv[j + i], P[j]);
      }
    if ((p->Compute) == &Call)
      pnl_cf_call_bs(l, p->Par[0].Val.V_PDOUBLE, (double)i * h, r, divid, sigm
    else
      pnl_cf_put_bs(l, p->Par[0].Val.V_PDOUBLE, (double)i * h, r, divid, sigma
    P[npoints] = price;
  }
```

```
  npoints++;

  for (i = N - eta0 + 1; i < N; i++)
    {
      npoints -= 2;
      for (j = 0; j < npoints; j++)
        {
          P[j] = pu * P[j] + pm * P[j + 1] + pd * P[j + 2];
          if (am)
            P[j] = MAX(iv[j + i], P[j]);
        }
    }

  /*Delta*/
  *ptdelta = (P[0] - P[2]) / (s * u - s * d);
  /*First time step*/
  P[0] = pu * P[0] + pm * P[1] + pd * P[2];
  if (am)
    P[0] = MAX(iv[N], P[0]);
  /*Price*/
  *ptprice = P[0];

  free(P);
  free(iv);

  return OK;
}

int CALC(TR_Ritchken_DownIn)(void *Opt, void *Mod, PricingMethod *Met)
{
  TYPEOPT *ptOpt = (TYPEOPT *)Opt;
  TYPEMOD *ptMod = (TYPEMOD *)Mod;
  double r, divid, limit, rebate;

  r = log(1. + ptMod->R.Val.V_DOUBLE / 100.);
  divid = log(1. + ptMod->Divid.Val.V_DOUBLE / 100.);
  limit = ((ptOpt->Limit.Val.V_NUMFUNC_1)->Compute)((ptOpt->Limit.Val.V_NUMFUNC_
  rebate = ((ptOpt->Rebate.Val.V_NUMFUNC_1)->Compute)((ptOpt->Rebate.Val.V_NUMFU

  return Ritchken_95_DownIn(ptOpt->EuOrAm.Val.V_BOOL,
                            ptMod->S0.Val.V_PDOUBLE, ptOpt->PayOff.Val.V_NUMFUNC
```

```
                            limit, ptOpt->Maturity.Val.V_DATE - ptMod->T.Val.V_D
                            r, divid, ptMod->Sigma.Val.V_PDOUBLE, Met->Par[0].Va
                            &(Met->Res[0].Val.V_DOUBLE), &(Met->Res[1].Val.V_DOU
}

static int CHK_OPT(TR_Ritchken_DownIn)(void *Opt, void *Mod)
{
  Option *ptOpt = (Option *)Opt;
  TYPEOPT *opt = (TYPEOPT *)(ptOpt->TypeOpt);

  if ((opt->OutOrIn).Val.V_BOOL == IN)
    if ((opt->DownOrUp).Val.V_BOOL == DOWN)
      if ((opt->Parisian).Val.V_BOOL == FALSE)
        return OK;

  return  WRONG;
}

static int MET(Init)(PricingMethod *Met, Option *Opt)
{
  if (Met->init == 0)
    {
      Met->init = 1;
      Met->Par[0].Val.V_INT2 = 100;
    }

  return OK;
}

PricingMethod MET(TR_Ritchken_DownIn) =
{
  "TR_Ritchken_DownIn",
  {{"StepNumber", INT2, {100}, ALLOW}, {" ", PREMIA_NULLTYPE, {0}, FORBID}},
  CALC(TR_Ritchken_DownIn),
  {{"Price", DOUBLE, {100}, FORBID}, {"Delta", DOUBLE, {100}, FORBID} , {" ", PR
  CHK_OPT(TR_Ritchken_DownIn),
  CHK_tree,
  MET(Init)
};
```