Help

```
#include  "bs1d_std.h"


static int BBS(double s,NumFunc_1*p,double t,
    double r,double divid,double sigma,int N, double *ptpr
    ice,double *ptdelta)
    {
       int i,j,dummy;
       double u,d,h,pu,pd,a1,b1,stock,upperstock,
    delta,tmp,d_square;
       double *P,*iv;

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

       /*Up and Down factors*/
       h=t/(double)N;
       a1= exp(h*(r-divid));
       b1= SQR(a1)*(exp(SQR(sigma)*h)-1.);
       tmp=SQR(a1)+b1+1.;
       u = (tmp+sqrt(SQR(tmp)-4.*SQR(a1)))/(2.*a1
    );
       d= 1./u;
     d_square=d*d;

       /*Risk-Neutral Probability*/
       pu=(a1-d)/(u-d);
       pd=1.-pu;
       pu*=exp(-r*h);
       pd*=exp(-r*h);

       /*Intrisic value initialisation*/
       upperstock=s;
```

```
    for (i=0;i<N;i++)
        upperstock*=u;
    stock=upperstock;
    for (i=0;i<2*N+1;i++)
        {
        iv[i]=(p->Compute)(p->Par,stock);
        stock*=d;
        }

    /*Backward Resolution*/
    stock=upperstock*d;

    /*LastTime Step*/
    for (j=0;j<N;j++)
        {
        if  ((p->Compute)==&Call)
            dummy=Call_BlackScholes_73(stock,
p->Par[0].Val.V_PDOUBLE,h,r,divid,sigma,P+j,&delt
a);
        else
            if  ((p->Compute)==&Put)
                dummy=Put_BlackScholes_73(sto
ck,p->Par[0].Val.V_PDOUBLE,h,r,divid,sigma,P+j,&
delta);
            stock*=d_square;
      P[j]=MAX(iv[1+2*j],P[j]);
        }

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

        /*Delta*/
        *ptdelta=(P[0]-P[1])/(s*u-s*d);

        /*First time step*/
        P[0] = pu*P[0]+ pd*P[1];
        P[0]= MAX(iv[N],P[0]);
```

```
            /*Price*/
            *ptprice=P[0];

            free(P);
            free(iv);

            return OK;
    }

static int BBSR(double s,NumFunc_1*p,double t,
    double r,double divid,double sigma,int N, double *ptpr
    ice,double *ptdelta)
    {
    int dummy;
    double price1,delta1,price2,delta2;

    dummy=BBS(s,p,t,r,divid,sigma,N,&price1,&delt
    a1);
    dummy=BBS(s,p,t,r,divid,sigma,N/2,&price2,&de
    lta2);

    /*Price*/
    *ptprice=2.*price1-price2;

    /*Delta*/
    *ptdelta=2.*delta1-delta2;

    return OK;
    }

int CALC(TR_BBSR)(void *Opt,void *Mod,Pricing
    Method *Met)
    {
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;
    double r,divid;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
```

```
    return BBSR(ptMod->S0.Val.V_PDOUBLE,ptOpt->
    PayOff.Val.V_NUMFUNC_1,ptOpt->Maturity.Val.V_DATE-
    ptMod->T.Val.V_DATE,r,divid,ptMod->Sigma.Val.V_PD
    OUBLE,Met->Par[0].Val.V_INT,&(Met->Res[0].Val.V_
    DOUBLE),&(Met->Res[1].Val.V_DOUBLE));
    }


int CHK_OPT(TR_BBSR)(void *Opt, void *Mod)
    {
    if ( (strcmp( ((Option*)Opt)->Name,"PutAmer")
    ==0) || (strcmp( ((Option*)Opt)->Name,"
    CallAmer")==0) )
        return OK;

    return WRONG;
    }

static int MET(Init)(PricingMethod *Met)
    {
    static int first=1;

    if (first)
        {
        Met->Par[0].Val.V_INT2=100;

        first=0;
        }

    return OK;
    }

PricingMethod MET(TR_BBSR)=
    {
    "TR_BBSR",
        {{"StepNumber",INT2,100,ALLOW},{" ",END,0
    ,FORBID}},
        CALC(TR_BBSR),
        {{"Price",DOUBLE,100,FORBID},{"Delta",
    DOUBLE,100,FORBID} ,{" ",END,0,FORBID}},
        CHK_OPT(TR_BBSR),
```

```
        CHK_tree,
        MET(Init)
    };
```

References