#ifndef \_\_BinaryTree\_Paolo\_Bowen\_\_BinaryTree\_\_  
#define \_\_BinaryTree\_Paolo\_Bowen\_\_BinaryTree\_\_  
  
#include <iostream>  
#include <cmath>  
  
using namespace std;  
  
  
struct derivative  
{  
    double r; // interest rate  
    double So; // current stock price, spot pric  
    double X; // strike price  
    double T; // time to maturity  
    double vol; // volatility  
      
    derivative(double interest, double spot, double strike, double time, double volatility) // constructor  
    {  
        r = interest;  
        So = spot;  
        X = strike;  
        T = time;  
        vol = volatility;  
    };  
      
    ~derivative() {}; // destructor  
      
};  
  
  
struct node  
{  
    double stock; //stock price  
    double option; // option price  
    double time; // time to maturity at this node  
};  
  
  
#endif /\* defined(\_\_BinaryTree\_Paolo\_Bowen\_\_BinaryTree\_\_) \*/  
  
  
  
  
  
  
========================================================  
  
  
//  
//  main.cpp  
//  BinaryTree Paolo Bowen  
//  
//  Created by Bowen Gong on 15/04/2013.  
//  Copyright (c) 2013 Bowen Gong. All rights reserved.  
//  
  
#include <iostream>  
#include "BinaryTree.h"  
  
  
  
int main()  
{  
    // specify the derivatives and levels on the tree  
    int tot\_level = 25;  
    int max\_level = tot\_level -1;  
    derivative D(0.05, 100, 100, 1, 0.3);  
      
    // construct the dynamic 2D array  
      
    node\*\* level\_control;  
      
    level\_control = new node\*[tot\_level]; // total levels array  
      
    int n = tot\_level ;  
    for (int i=0; i<n; i++) // construct each level array  
    {  
        size\_t m = pow(2, i);  
        level\_control[i] = new node[m];  
    }  
      
      
    double u ; // uptick  
    double d ; // downtick  
    double dt = D.T / tot\_level;   
      
    u = exp(D.vol\*sqrt(dt)); // uptick, matching volatility  
    d = exp(-D.vol\*sqrt(dt)); // downtick, matching volatility  
      
    double p = (exp(D.r\*dt)-d) / (u-d); // probability  
    double anti\_p = 1-p;  
  
      
    // forward fill stock prices and time to maturity  
      
      // fill the first column  
    level\_control[0][0].stock = D.So; // initial stock price  
    level\_control[0][0].time = D.T; // initial time to maturity  
      
      
      // fill the rest of the columns  
    for (int i =1; i<tot\_level; i++)  
    {  
        size\_t k = pow(2, i) ; // # of elements in current column + 1  
        for (int j=0; j<k; j= j+2)  
        {  
            int tempj= j/2; // int division  
            level\_control[i][j].stock = level\_control[i-1][tempj].stock \* u; // uptick  
            level\_control[i][j+1].stock = level\_control[i-1][tempj].stock \* d; // downtick  
          
            level\_control[i][j].time = D.T - dt\*i; // time to maturity at each level  
            level\_control[i][j+1].time = D.T - dt\*i;  
        }  
    }  
      
      
    // backward fill option prices  
      
      
      //fill the last column  
    size\_t temp = pow(2, max\_level);  
    double strike = D.X;  
      
    for (int j=0; j<temp; j=j+2)  
    {  
        level\_control[max\_level][j].option = level\_control[max\_level][j].stock - strike;  
        level\_control[max\_level][j+1].option = level\_control[max\_level][j+1].stock - strike;  
    }  
      
    for (int i = max\_level-1; i> -1; i--)  
    {  
        size\_t k = pow(2, i);  
        for (int j=0; j<k; j++)  
        {  
            double fu = level\_control[i+1][2\*j].option; // access the uptick option price  
            double fd = level\_control[i+1][2\*j+1].option; // access the downtick option price  
              
            level\_control[i][j].option = exp(-D.r\*dt)\*(p\*fu+anti\_p\*fd); // option price calculation  
            // f = exp(-r\*dt)\*(p\*fu+(1-p)\*fd);  
            // p = (exp(r\*dt) -d) / (u-d)  
        }  
    }  
      
    // query the binary tree  
      
      
    int quit = 1;  
      
    while(quit)  
    {  
        cout<<"there are "<<tot\_level<<" of levels including the first spot\n please enter how many upticks:"<<endl;  
        int upticks;  
        cin>> upticks;  
          
        cout<<"please enter how many downticks:"<<endl;  
        int downticks;  
        cin>>downticks;  
          
        cout<<"stock:"<<level\_control[upticks+downticks-1][downticks].stock<<endl;  
        cout<<"option:"<<level\_control[upticks+downticks-1][downticks].option<<endl;  
        cout<<"time to maturity:"<<level\_control[upticks+downticks-1][downticks].time<<endl;  
          
        cout<<"enter 0 to quit, or any other number to continue"<<endl;  
        cin >>quit;  
    }  
      
          
      
    // destruct dynamic array  
      
    for (int i=0; i<n; i++)  
    {  
        delete [] level\_control[i] ;  
    }  
      
    delete [] level\_control;  
      
      
       
    return 0;  
}

///////////////////////////////////////////////////////////////////////