Financial analysis and investment using language programming C++.

Dr Michel Zaki Guirguis **Date:13/05/2021**

94, Terpsichoris road
Palaio-Faliro
Post Code:17562
Athens
Greece
Tel:0030-210-9841550
Mobile: 0030-6982044429
E-mail: guirguismichel@gmail.com

Abstract

C++ is very useful language programming in modeling investment exercises, financial derivatives and their related Greeks symbols. The book covers BA and BS, MA and MS in Risk Management, Business Administration, Financial Services, International Business and Financial Derivatives and BS in Computational Finance. Most importantly, the book includes a range of materials to help the student, the practitioners and the investors to reinforce their learning skills. Each section offers a solid explanation of basic concepts, followed by examples for the reader to work through. The market will be very responsive for our book especially that a lot of international students experience problem with their English and their numerical skills. It will cover the basic needs of postgraduate students and those who are interesting in investment. The market in the next five years will become very complicated and would require the use of sophisticated risk management techniques and technological solutions in order to hedge market, operational, and credit risk. Before starting programming, please read Chandan Sengupta (2007), Financial Modeling using C++. Wiley Finance. Good luck in your future career as derivatives and investment programmer in the major investment banks such as JP Morgan, Merrill Lunch, Morgan Stanley, Deutsche Bank, Goldman Sachs, and Bank of America. Please e-mail me if you have any questions or if you would like to suggest investment exercises. My e-mail is guirguismichel@gmail.com

Thank you for your participation.

Introduction

C++ is very useful language programming in modeling investment exercises, financial derivatives and their related Greeks symbols. First of all, you should download the software as part of an integrated development environment, (IDE). A possible public domain is www. Bloodshed.net. You could download the software, (IDE) version 4.9.9.2.

You can start C++ by creating a source code in the text editor. Then, press execute and selects the C++ compiler to compile the code. Finally, press the debug or run button to get the output in a DOS window or console. You first have to do the compiling and then to run the console. During the compiling, the source code is translated into object code. Then, through the linker the object code is combined with other files into an executable program. The most common software is to use the integrated development environment as part of a C++ compiler and a debugger to help you fix the errors or improve the code. Sometimes, you will get errors that are the reason of lacking to insert the correct header. For example, if you do not include the header *#include <math.h>* or # include <*cmath*>, the compiler will not recognize the mathematical functions such as exp, log and sqrt. It is very important to select the correct headers. For example, <cmath> means standard mathematical library.

If you do not have the right headers, then, you will not be able to do compiling and debug. In other words, you will not be able to get the output in the DOS window or console. You will get error messages as the computer cannot recognize the formulas.

The comments, variables, functions, and mathematical formulas have to be inserted to be able to get the output. We use the symbol // to show a single line comment. The lines that include statements should end with *semicolon ;* in C++ . We use numerical variables with fractional parts. In this case, the term *double* is inserted in front of each variable. Once you define a variable with capital or small letters, you can not change it with different layout as you move to the calculation part. Consistency is very important. Other types of data are integers, real numbers, characters, strings and Boolean values. Pay particular attention to the lower and upper case letters. For example, integer for interest rate is written as variable *intRate*. Another example that can be used is the integer value for the amount. It is written as *intAmount.*

Very important headers that we use in derivatives and investment are the following:

```
# include <iostream>
# include <cmath> Or  # include <math.h>
#include < cstdlib>
using namespace std;
```

The name of the header has to be enclosed between the symbols < >. You should use the header < *cstdlib*> for functions that will be used from the standard library,(STL). The header < *iostream*> is very important as it includes the input/output, (I/O) library. You should also use blank line for presentation purposes. The execution of the program starts with the statement *int main()*

Then, the code should be enclosed between two braces {}.

The main heading to start C++ is as follows:

```
# include <iostream>
using namespace std;

int main()

 {

/* Identify your given variables using double for numbers with fractions or int for
integers numbers. Identify the variables that will be calculated. The symbols /* and */
are used for multiline comments.
*/

// Insert the mathematical formula or formulas.

*/ Insert the output functions cout and the cin statement if needed. The cout statement
    is a prompt followed by the cin statement. For example,
   cout <<”Enter interest rate:”<<endl;
   cin>> interest rate;

You should used the extraction operator, >> and quotation marks for the cout
function. This is the basic layout that I have used to convert most of the mathematical
problems to C++.
*/

system("PAUSE");
  return 0;
}
```

The data types that are incorporated into the C++ language are *char, wchar-t, int, float, double, bool, and void*. The available modifiers are *signed, unsigned, long and short.*

The *cin* function tells the computer to use the input value from the keyboard and place it in the variable length that has been identified in the variable section.

The function *cout* is used to produce outputs. The symbol << is known as the output operator and the statement or message should be enclosed between quotation marks.

Messages should be enclosed between a pair of double quotation marks. You could use an escape character *\n* to tell the program to start a new line.

The statement *system("PAUSE");* tells the computer in DOS environment to display the message and wait until you press any key to close the console or the DOS window.

Finally the statement *returns 0;* tells the computer to end the program.

Calculations of European call and put prices by applying Black and Scholes model, (BSM).

```
/* Calculation of Euro call and put prices by applying Black and Scholes model,
BSM. price of stock = 70, strike price = 70, interest rate = 0.08, dividend yield = 0.03,
life  to maturity = 0.5, and volatility = 0.2.*/

#include <iostream>
# include <math.h>
using namespace std;

// Calculations of the European call and put prices.

double Call(double S, double K, double r, double q, double T,
  double sig);
double Put(double S, double K, double r, double q, double T,
  double sig);

// Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
    return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
      + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}
```

```
// Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S*exp(-q*T)*N(d1) - K*exp(-r*T)*N(d2) ;
}

// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*N(-d2) - S*exp(-q*T)*N(-d1) ;
}

// Identification or declaration of the variables.

int main()
{
  double S = 70;        // price of stock
  double K = 70;        // strike price
  double r = 0.08;       // interest rate
  double q = 0.03;       // dividend yield
  double T = 0.5;       //  life to maturity
  double sig = 0.20;     // volatility


/*Calculation of call and put prices in addition of showing the numerical values of the
variables.*/

double callPrice, putPrice;

callPrice = Call (S,K,r,q,T,sig);
putPrice = Put (S,K,r,q,T,sig);

std::cout<<"Price of stock: " <<S<<std::endl;
std::cout<<"Strike price: " <<K<<std::endl;
std::cout<<"Interest rate: " <<r<<std::endl;
std::cout<<"Dividend yield: " <<q<<std::endl;
std::cout<<"Life to maturity: " <<T<<std::endl;
std::cout<<"Volatility: " <<sig<<std::endl;
cout<<"Call price: " <<callPrice << endl;
cout<<"Put price:" <<putPrice << endl;

system("PAUSE");

 return 0;

}
```

<u>Output</u>

After compiling and debugging, the DOS window or console will open and display the following results:

```
Price of stock: 70
Strike price: 70
Interest rate: 0.08
Dividend yield: 0.03
Life to maturity: 0.5
Volatility: 0.2
Call price: 4.75036
Put price: 3.04778

Press any key to continue….
```

Calculations of European call and put prices by applying Black and Scholes model, (BSM). It is the same example as above. The difference is in calculating the cumulative normal distribution and inserting it in the mathematical formulas for call and put price. In addition, I have included other layout for call and put option calculations.

```
/* Calculations of European call and put prices by applying Black and Scholes model,
(BSM). price of stock = 70, strike price = 70, interest rate = 0.08, dividend yield =
0.03, life to maturity = 0.5, and volatility = 0.2.*/

#include <iostream>
# include <math.h>
using namespace std;

// Calculations of the European call and put prices.

double Call(double S, double K, double r, double q, double T,
  double sig);
double Put(double S, double K, double r, double q, double T,
  double sig);

// Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}

// Mathematical formulas to calculate d1, d2 and the European call price.

double Call(
  double S,        // price of stock
  double K,        // strike price
  double r,        // interest rate
  double q,        // dividend yield
  double T,        // life to maturity
  double sig       // volatility
  )
{
  double d1, d2;
  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S*exp(-q*T)*norm_cdf(d1) - K*exp(-r*T)*norm_cdf(d2) ;

}

// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(
  double S,        // price of stock
  double K,        // strike price
  double r,        // interest rate
  double q,        // dividend yield
  double T,        // life to maturity
  double sig       // volatility
  )

{
  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*norm_cdf(-d2) - S*exp(-q*T)*norm_cdf(-d1) ;

}

// Identification or declaration of the variables.

int main()
{
  double S = 70;        // price of stock
  double K = 70;        // strike price
  double r = 0.08;       // interest rate
  double q = 0.03;       // dividend yield
  double T = 0.50;       //  life to maturity
  double sig = 0.20;     // volatility

/*Calculation of call and put prices in addition of showing the numerical values of the
variables.*/

double callPrice, putPrice;

callPrice = Call (S,K,r,q,T,sig);
putPrice = Put (S,K,r,q,T,sig);

std::cout<<"Price of stock: " <<S<<std::endl;
```

```
std::cout<<"Strike price: " <<K<<std::endl;
std::cout<<"Interest rate: " <<r<<std::endl;
std::cout<<"Dividend yield: " <<q<<std::endl;
std::cout<<"Life to maturity: " <<T<<std::endl;
std::cout<<"Volatility: " <<sig<<std::endl;
cout << "Call price: " << callPrice << endl;
cout << "Put price: " << putPrice << endl;

system("PAUSE");

 return 0;

}
```

<u>Output</u>

After compiling and debugging, the DOS window or console will open and display the following results:

```
Price of stock: 70
Strike price: 70
Interest rate: 0.08
Dividend yield: 0.03
Life to maturity: 0.5
Volatility: 0.2
Call price: 4.75
Put price: 3.05

Press any key to continue….
```

Calculations of European call and put prices and their related Greeks by applying Black and Scholes model, (BSM).

```
/*Calculations of Euro call and put prices by applying Black and Scholes model,
BSM. price of stock = 70, strike price = 70, interest rate = 0.08, dividend yield = 0.03,
life to maturity = 0.5, and volatility = 0.2.*/

#include <iostream>
# include <math.h>
using namespace std;

// Calculation of the European call and put prices.

double Call(double S, double K, double r, double q, double T,
  double sig);
double Put(double S, double K, double r, double q, double T,
  double sig);
double CallDelta(double S, double K, double r, double q,
  double T, double sig);
double PutDelta(double S, double K, double r, double q,
  double T, double sig);
double CallTheta(double S, double K, double r, double q,
  double T, double sig);
double PutTheta(double S, double K, double r, double q,
  double T, double sig);
double Gamma(double S, double K, double r, double q,
  double T, double sig);
double Vega(double S, double K, double r, double q,
  double T, double sig);
double CallRho(double S, double K, double r, double q,
  double T, double sig);
double PutRho(double S, double K, double r, double q,
  double T, double sig);

// Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
    return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
      + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}

// Mathematical formulas to calculate d1, d2 and the European Call price.

double Call(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S*exp(-q*T)*N(d1) - K*exp(-r*T)*N(d2) ;
}

// Mathematical formulas to calculate d1, d2 and the CallDelta price.

double CallDelta(double S, double K, double r, double q,
  double T, double sig){

double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return exp(-q*T)*N(d1);
}
```

```
// Mathematical formulas to calculate d1, d2 and the CallTheta price.

double CallTheta(double S, double K, double r, double q,
  double T, double sig){

double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return -(S*NP(d1)*sig*exp(-q*T))/(2*sqrt(T))
    + (q*S*N(d1)*exp(-q*T)) - (r*K*exp(-r*T)*N(d2));
}

// Mathematical formulas to calculate d1, d2 and the CallRho price.

double CallRho(double S, double K, double r, double q,
  double T, double sig){

double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*T*exp(-r*T)*N(d2);
}


// Mathematical formulas to calculate d1, d2 and the Gamma price.

double Gamma(double S, double K, double r, double q,
  double T, double sig){

double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return (NP(d1)*exp(-q*T))/(S*sig*sqrt(T));
}
```

```
// Mathematical formulas to calculate d1, d2 and the Vega price.

double Vega(double S, double K, double r, double q,
  double T, double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S*sqrt(T)*NP(d1)*exp(-q*T);
}

// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*N(-d2) - S*exp(-q*T)*N(-d1) ;
}

// Mathematical formulas to calculate d1, d2 and the European PutDelta.

double PutDelta(double S, double K, double r, double q,
  double T, double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return exp(-q*T)*(N(d1)-1);
}

// Mathematical formulas to calculate d1, d2 and the European PutTheta.

double PutTheta(double S, double K, double r, double q,
  double T, double sig){


double d1, d2;
```

```
  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return -(S*NP(d1)*sig*exp(-q*T))/(2*sqrt(T))
    - (q*S*N(-d1)*exp(-q*T)) + (r*K*exp(-r*T)*N(-d2));
}
// Mathematical formulas to calculate d1, d2 and the European PutRho.

double PutRho(double S, double K, double r, double q,
  double T, double sig){

double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return -K*T*exp(-r*T)*N(-d2);
}

// Identification or declaration of the variables.

int main()
{
  double S = 70;        // price of stock
  double K = 70;        // strike price
  double r = 0.08;       // interest rate
  double q = 0.03;       // dividend yield
  double T = 0.5;       //  life to maturity
  double sig = 0.20;     // volatility


/*Calculation of call and put prices in addition of showing the numerical values of the
variables.*/

double callPrice, putPrice, callDelta, putDelta, callTheta, putTheta, callRho, putRho,
gamma, vega;

callPrice = Call (S,K,r,q,T,sig);
putPrice = Put (S,K,r,q,T,sig);
callDelta = CallDelta (S, K, r, q, T, sig) ;
putDelta =  PutDelta(S, K, r, q, T, sig);
callTheta = CallTheta(S, K, r, q, T, sig) ;
putTheta = PutTheta(S, K, r, q, T, sig) ;
callRho =  CallRho(S, K, r, q, T, sig) ;
putRho =   PutRho(S, K, r, q, T, sig) ;
gamma = Gamma(S, K, r, q, T, sig) ;
vega = Vega(S, K, r, q, T, sig) ;
```

```
std::cout<<"Price of stock: " <<S<<std::endl;
std::cout<<"Strike price: " <<K<<std::endl;
std::cout<<"Interest rate: " <<r<<std::endl;
std::cout<<"Dividend yield: " <<q<<std::endl;
std::cout<<"Life to maturity: " <<T<<std::endl;
std::cout<<"Volatility: " <<sig<<std::endl;
cout<<"Call price: " <<callPrice << endl;
cout<<"Put price:" <<putPrice << endl;
cout << "Call delta: "<<callDelta <<endl;
cout << "Put delta: "<<putDelta <<endl;
cout << "Call theta: "<<callTheta <<endl;
cout << "Put theta: "<<putTheta <<endl;
cout <<" Callrho:"<<callRho <<endl;
cout <<"  Putrho: "<<putRho <<endl;
cout << "Gamma: "<<gamma <<endl;
cout << "Vega: "<<vega <<endl;

system("PAUSE");

 return 0;

}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Price of stock: 70
Strike price: 70
Interest rate: 0.08
Dividend yield: 0.03
Life to maturity: 0.5
Volatility: 0.2
Call price: 4.75
Put price: 3.05
Call delta: 0.589
Put delta: -0.396 or -0.40
Call theta: -5.45
Put theta: -2.14
Call rho: 18.23
Put rho: -15.39
Gamma: 0.0385 or 0.039
Vega: 18.87
Press any key to continue….
```

Calculation of the mean of stock price returns and inserting the function cin to allow us to write in the console or DOS window the numerical values of stock price returns.

```
#include <iostream>
using namespace std;
int main()
{
double stockPrice[6];
stockPrice[1] = 10;
stockPrice[2] = 12;
stockPrice[3] = 13;
stockPrice[4] = 14;
stockPrice[5] = 15;
stockPrice[6] = 16;
double average;

// Insert the mathematical formula.

average = (stockPrice[1] + stockPrice[2] + stockPrice[3] + stockPrice[4] +
stockPrice[5] + stockPrice[6])/6;

// Output functions.

cout<<" The mean: "<<average<<endl;
cout<< "Enter stockPrice" <<endl;
cin>> stockPrice[1];
cin>> stockPrice[2];
cin>> stockPrice[3];
cin>> stockPrice[4];
cin>> stockPrice[5];
cin>> stockPrice[6];
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
The mean : 13.3333
Enter StockPrice  (This command allows you to insert the stock price returns.
You input 10. Then, press Enter.
12  Press Enter.
13  Press Enter
14  Press Enter
15  Press Enter
Press any key to continue….
```

Repetition of the previous example by replacing the function double with int. Calculation of the mean of stock price returns and inserting the function cin to allow us to write in the console or DOS window the numerical values of stock price returns.

I would like to make clear that we use the function double to include numbers with fractional parts. In contrast, we use the function int which is integers to include whole numbers. I repeat the above example, to show the difference. In Financial investment, we use the function double to include the decimals.

```
#include <iostream>
using namespace std;
int main()
{
int stockPrice[6];
stockPrice[1] = 10;
stockPrice[2] = 12;
stockPrice[3] = 13;
stockPrice[4] = 14;
stockPrice[5] = 15;
stockPrice[6] = 16;
int average;

// Insert the mathematical formula.

average = (stockPrice[1] + stockPrice[2] + stockPrice[3] + stockPrice[4] +
stockPrice[5] + stockPrice[6])/6;

// Output functions.

cout<<" The mean: "<<average<<endl;
cout<< "Enter stockPrice:" <<endl;
cin>> stockPrice[0];
cin>> stockPrice[1];
cin>> stockPrice[2];
cin>> stockPrice[3];
cin>> stockPrice[4];
cin>> stockPrice[5];
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
The mean : 13
Enter StockPrice  (This command allows you to insert the stock price returns.
You input 6. Then press Enter.
12
13
14
15
Press any key to continue….
```

Calculations of the mean and the harmonic mean of share price returns, call option returns, put option returns and index option returns.

```
#include <iostream>
using namespace std;
int main()
{
double SharePriceReturns[6];
SharePriceReturns[1] = 0.08;
SharePriceReturns[2] = 0.05;
SharePriceReturns[3] = 0.04;
SharePriceReturns[4] = 0.07;
SharePriceReturns[5] = 0.02;
SharePriceReturns[6] = 0.03;

double CallOptionPriceReturns[6];
CallOptionPriceReturns[1] = 0.01;
CallOptionPriceReturns[2] = 0.02;
CallOptionPriceReturns[3] = 0.04;
CallOptionPriceReturns[4] = 0.05;
CallOptionPriceReturns[5] = 0.06;
CallOptionPriceReturns[6] = 0.06;

double PutOptionPriceReturns[6];
PutOptionPriceReturns[1] = 0.03;
PutOptionPriceReturns[2] = 0.07;
PutOptionPriceReturns[3] = 0.08;
PutOptionPriceReturns[4] = 0.09;
PutOptionPriceReturns[5] = 0.1;
PutOptionPriceReturns[6] = 0.02;

double IndexOptionPriceReturns[6];
IndexOptionPriceReturns[1] = 0.12;
IndexOptionPriceReturns[2] = 0.23;
IndexOptionPriceReturns[3] = 0.34;
IndexOptionPriceReturns[4] = 0.15;
IndexOptionPriceReturns[5] = 0.2;
IndexOptionPriceReturns[6] = 0.11;

double averageSharePriceReturns;
double averageCallOptionPriceReturns;
double averagePutOptionPriceReturns;
double averageIndexOptionPriceReturns;
double harmonicaverageSharePriceReturns;
double harmonicaverageCallOptionPriceReturns;
double harmonicaveragePutOptionPriceReturns;
double harmonicaverageIndexOptionPriceReturns;
```

```
averageSharePriceReturns = (SharePriceReturns[1] + SharePriceReturns[2] +
SharePriceReturns[3] + SharePriceReturns[4] + SharePriceReturns[5] +
SharePriceReturns[6])/6;

averageCallOptionPriceReturns = (CallOptionPriceReturns[1] +
CallOptionPriceReturns[2] + CallOptionPriceReturns[3] + CallOptionPriceReturns[4]
+ CallOptionPriceReturns[5] + CallOptionPriceReturns[6])/6;

averagePutOptionPriceReturns = (PutOptionPriceReturns[1] +
PutOptionPriceReturns[2] + PutOptionPriceReturns[3] + PutOptionPriceReturns[4] +
PutOptionPriceReturns[5] + PutOptionPriceReturns[6])/6;

averageIndexOptionPriceReturns = (IndexOptionPriceReturns[1] +
IndexOptionPriceReturns[2] + IndexOptionPriceReturns[3] +
IndexOptionPriceReturns[4] + IndexOptionPriceReturns[5] +
IndexOptionPriceReturns[6])/6;

harmonicaverageSharePriceReturns = (1/((1/SharePriceReturns[1] +
1/SharePriceReturns[2] + 1/SharePriceReturns[3] + 1/SharePriceReturns[4] +
1/SharePriceReturns[5] + 1/SharePriceReturns[6])/6));

harmonicaverageCallOptionPriceReturns = (1/((1/CallOptionPriceReturns[1] +
1/CallOptionPriceReturns[2] + 1/CallOptionPriceReturns[3] +
1/CallOptionPriceReturns[4] + 1/CallOptionPriceReturns[5] +
1/CallOptionPriceReturns[6])/6));

harmonicaveragePutOptionPriceReturns = (1/((1/PutOptionPriceReturns[1] +
1/PutOptionPriceReturns[2] + 1/PutOptionPriceReturns[3] +
1/PutOptionPriceReturns[4] + 1/PutOptionPriceReturns[5] +
1/PutOptionPriceReturns[6])/6));

harmonicaverageIndexOptionPriceReturns = (1/((1/IndexOptionPriceReturns[1] +
1/IndexOptionPriceReturns[2] + 1/IndexOptionPriceReturns[3] +
1/IndexOptionPriceReturns[4] + 1/IndexOptionPriceReturns[5] +
1/IndexOptionPriceReturns[6])/6));

cout<<" The mean SharePriceReturns: "<<averageSharePriceReturns<<endl;
cout<<" The mean CallOptionPriceReturns:
"<<averageCallOptionPriceReturns<<endl;
cout<<" The mean PutOptionPriceReturns: "<<averagePutOptionPriceReturns<<endl;
cout<<" The mean IndexOptionPriceReturns:
"<<averageIndexOptionPriceReturns<<endl;

cout<<" The harmonic mean SharePriceReturns:
"<<harmonicaverageSharePriceReturns<<endl;
cout<<" The harmonic mean CallOptionPriceReturns:
"<<harmonicaverageCallOptionPriceReturns<<endl;
cout<<" The harmonic mean PutOptionPriceReturns:
"<<harmonicaveragePutOptionPriceReturns<<endl;
```

```
cout<<" The harmonic mean IndexOptionPriceReturns:
"<<harmonicaverageIndexOptionPriceReturns<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
The mean SharePriceReturns: 0.048
The mean CallOptionPriceReturns:0.04
The mean PutOptionPriceReturns: 0.065
The mean IndexOptionPriceReturns: 0.19
The harmonic mean SharePriceReturns: 0.039
The harmonic mean CallOptionPriceReturns: 0.026
The harmonic mean PutOptionPriceReturns: 0.046
The harmonic mean IndexOptionPriceReturns: 0.16

Press any key to continue….
```

Exercise of how to calculate the payoff of buying a call, a put and the net result or position of your investment position expressed in pounds.

```
#include <iostream>
using namespace std;

int main()
{
double SharePrice[6];
SharePrice[1] = 105;
SharePrice[2] = 110;
SharePrice[3] = 120;
SharePrice[4] = 130;
SharePrice[5] = 140;
SharePrice[6] = 150;

double ExercisePriceCall[6];
ExercisePriceCall[1] = 90;
ExercisePriceCall[2] = 90;
ExercisePriceCall[3] = 90;
ExercisePriceCall[4] = 90;
ExercisePriceCall[5] = 90;
ExercisePriceCall[6] = 90;

double ExercisePricePut[6];
ExercisePricePut[1] = 190;
ExercisePricePut[2] = 190;
ExercisePricePut[3] = 190;
ExercisePricePut[4] = 190;
ExercisePricePut[5] = 190;
ExercisePricePut[6] = 190;

double PremiumofCall [1];
PremiumofCall [1] = 14;

double PremiumofPut [1];
PremiumofPut [1] = 9;

double TotalNumberofContracts [1];
TotalNumberofContracts [1] = 10;

double totalNumberOfShares =100;

double payoffBuyaCall1;
double payoffBuyaCall2;
double payoffBuyaCall3;
double payoffBuyaCall4;
double payoffBuyaCall5;
double payoffBuyaCall6;
double payoffBuyaPut1;
double payoffBuyaPut2;
double payoffBuyaPut3;
double payoffBuyaPut4;
double payoffBuyaPut5;
double payoffBuyaPut6;

double NetResult1;
double NetResult2;
double NetResult3;
double NetResult4;
double NetResult5;
double NetResult6;

payoffBuyaCall1 = (SharePrice[1]-
(ExercisePriceCall[1]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;
payoffBuyaCall2 = ( SharePrice[2]-
(ExercisePriceCall[2]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;
payoffBuyaCall3 = ( SharePrice[3]-
(ExercisePriceCall[3]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;
payoffBuyaCall4 = ( SharePrice[4]-
(ExercisePriceCall[4]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;
payoffBuyaCall5 = ( SharePrice[5]-
(ExercisePriceCall[5]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;
payoffBuyaCall6 = ( SharePrice[6]-
(ExercisePriceCall[6]+PremiumofCall[1]))*TotalNumberofContracts[1]*totalNumber
OfShares;


payoffBuyaPut1 = ( ExercisePricePut[1]-SharePrice[1]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;
payoffBuyaPut2 = ( ExercisePricePut[2]-SharePrice[2]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;
payoffBuyaPut3 = ( ExercisePricePut[3]-SharePrice[3]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;
payoffBuyaPut4 = ( ExercisePricePut[4]-SharePrice[4]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;
payoffBuyaPut5 = ( ExercisePricePut[5]-SharePrice[5]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;
payoffBuyaPut6 = ( ExercisePricePut[6]-SharePrice[6]-
PremiumofPut[1])*TotalNumberofContracts[1]*totalNumberOfShares;

NetResult1 = payoffBuyaCall1 - payoffBuyaPut1;
NetResult2 = payoffBuyaCall2 - payoffBuyaPut2;
NetResult3 = payoffBuyaCall3 - payoffBuyaPut3;
```

```
NetResult4 = payoffBuyaCall4 - payoffBuyaPut4;
NetResult5 = payoffBuyaCall5 - payoffBuyaPut5;
NetResult6 = payoffBuyaCall6 - payoffBuyaPut6;

cout<<" Payoff of buy a call1: "<<payoffBuyaCall1<<endl;
cout<<" Payoff of buy a call2: "<<payoffBuyaCall2<<endl;
cout<<" Payoff of buy a call3: "<<payoffBuyaCall3<<endl;
cout<<" Payoff of buy a call4: "<<payoffBuyaCall4<<endl;
cout<<" Payoff of buy a call5: "<<payoffBuyaCall5<<endl;
cout<<" Payoff of buy a call6: "<<payoffBuyaCall6<<endl;

cout<<" Payoff of buy a Put1: "<<payoffBuyaPut1<<endl;
cout<<" Payoff of buy a Put2: "<<payoffBuyaPut2<<endl;
cout<<" Payoff of buy a Put3: "<<payoffBuyaPut3<<endl;
cout<<" Payoff of buy a Put4: "<<payoffBuyaPut4<<endl;
cout<<" Payoff of buy a Put5: "<<payoffBuyaPut5<<endl;
cout<<" Payoff of buy a Put6: "<<payoffBuyaPut6<<endl;


cout<<" NetResult1:  "<<NetResult1<<endl;
cout<<" NetResult2:  "<<NetResult2<<endl;
cout<<" NetResult3:  "<<NetResult3<<endl;
cout<<" NetResult4:  "<<NetResult4<<endl;
cout<<" NetResult5:  "<<NetResult5<<endl;
cout<<" NetResult6:  "<<NetResult6<<endl;


system ("PAUSE");
return 0;
}
```

<u>Output</u>

The amounts are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Payoff of buy a call1 :  1000
Payoff of buy a call2 :  6000
Payoff of buy a call3 :  16000
Payoff of buy a call4 :  26000
Payoff of buy a call5 :  36000
Payoff of buya call6 :  46000

Payoff of buy a put1 :  76000
Payoff of buy a put2 :  71000
Payoff of buy a put3 :  61000
Payoff of buy a put4 :  51000
Payoff of buy a put5 :  41000
Payoff of buy a put6 :  31000
```

```
NetResult1 :   -75000
NetResult2 :   -65000
NetResult3 :   -45000
NetResult4 :   -25000
NetResult5 :    -5000
NetResult6 :     15000


Press any key to continue….
```

Exercise.

Please calculate the range based on different share prices.

Answer.

```
#include <iostream>
using namespace std;

int main()
{
double SharePrice[6];
SharePrice[1] = 20;
SharePrice[2] = 30;
SharePrice[3] = 40;
SharePrice[4] = 50;
SharePrice[5] = 60;
SharePrice[6] = 70;

double RangeSharePrice;

// Insert the mathematical formula.

RangeSharePrice = (SharePrice[6]-SharePrice[1]);

// Output function.

cout<<" The Range: "<<RangeSharePrice<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
The Range: 50

Press any key to continue …
```

Calculate the log returns and the average using actual share price returns expressed in pounds.

```
#include <iostream>
#include<cmath>

using namespace std;

int main()
{

//Identify the variables.

double dailyret2;
double dailyret3;
double dailyret4;
double dailyret5;
double average;

// Actual share prices.

double sharePrices[5];
sharePrices[1] = 14.23;
sharePrices[2] = 15.67;
sharePrices[3] = 12.13;
sharePrices[4] = 11.45;
sharePrices[5] = 10.11;

/* Insert the mathematical formulas for ln returns. In C++ use the log function.
In Excel use the ln function.*/

    dailyret2 =  log(sharePrices[2]/sharePrices[1]);
    dailyret3 =  log(sharePrices[3]/ sharePrices[2]);
    dailyret4 =  log(sharePrices[4]/ sharePrices[3]);
    dailyret5 =  log (sharePrices[5]/ sharePrices[4]);

// Average calculation.

average = (dailyret2 + dailyret3+ dailyret4 + dailyret5)/4;

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"dailyret2:"<<dailyret2<<endl;
cout<<"dailyret3:"<<dailyret3<<endl;
cout<<"dailyret4:"<<dailyret4<<endl;
cout<<"dailyret5:"<<dailyret5<<endl;
cout<< "Average:"<<average<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
dailyret2:  0.10
dailyret3: -0.26
dailyret4: -0.06
dailyret5: -0.12
Average: -0.09
Press any key to continue …
```

Calculation of daily and annual volatility of percentage returns measured by the sample standard deviation.

```
/* Calculation of daily volatility of percentage returns measured by the sample
standard deviation. Then, convert it to annual by multiplying the standard deviation
by the square root of 250.*/

#include <iostream>
#include<math.h>
using namespace std;

int main()
{

// Identify the variables.

double sum ;
double average;
double sumDiffSqr;
double DailystdDev;
double AnnualstdDev;


double sharePrices[5] = {0.1423,0.1567,0.1213,0.1145,0.1011};

/* The following expression inserts a variable x that tells the computer
that the first observation 0 is 0.1423 and x is less than 5 to include all observations.*/

for (int x = 0; x < 5; x++)

//We insert the formula for summation.

sum += sharePrices[x];

// We insert the formula for average.
average = sum/5;

/* We calculate the sumDiffSqr by inserting the expression for that
tells the computer the numbers that will be included in the calculation. We calculate
the difference between each number and the average. Then, we square it and we sum
it.*/

sumDiffSqr = 0;
for (int x = 0; x < 5; x++)
{
        sumDiffSqr = sumDiffSqr + pow((sharePrices[x]-average),2);
        }
/* Insert the formula of the sample standard deviation by taking into account the
formual n-1.*/
DailystdDev = sqrt(sumDiffSqr/4);

//We use 250 trading days per year instead of 365 days.

AnnualstdDev = DailystdDev * sqrt(250);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Daily sample standard deviation:"<<DailystdDev<<endl;
cout<<"Annual standard deviation:"<<AnnualstdDev<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Daily sample standard deviation: 0.0222
Annual standard deviation: 0.3514
Press any key to continue …
```

Calculations of the sum, the average, the sum difference squares and the sample standard deviation of index price returns expressed as percentages from an array format. Please remember that we use the formula n-1. We have 5 observations. Thus, 5 – 1 = 4.

```
#include <iostream>
#include<math.h>
using namespace std;

int main()
{
// Calculations of the sum and average function of index prices from an array format.

// Identify the variables.

double sum ;
double average;
double stdDev;
double sumDiffSqr;

/* Identify the array of index prices. We start with a bracket that includes the five
observations followed by the numbers.*/

double arrayIndexPriceReturns[5] = {90.54,80.32,82.98,87.12,97.78};

/* The following expression inserts a variable x that tells the computer
that the first observation 0 is 90.54 and x is less than 5 to include all observations.*/

for (int x = 0; x < 5; x++)

//We insert the formula for summation.

sum += arrayIndexPriceReturns[x];

// We insert the formula for average.
average = sum/5;

/* We calculate the sumDiffSqr by inserting the expression for that
tells the computer the numbers that will be included in the calculation. We calculate
the difference between each number and the average. Then, we square it and we sum
it.*/

sumDiffSqr = 0;
for (int x = 0; x < 5; x++)
{
      sumDiffSqr = sumDiffSqr + pow((arrayIndexPriceReturns[x]-average),2);
      }
/* Insert the formula of the sample standard deviation by taking into account the
formual n-1.*/
stdDev = sqrt(sumDiffSqr/4);


// Output functions and conclusion.

cout<< "Sum:"<<sum<<endl;
cout<< "Average:"<<average<<endl;
cout<<"Standard deviation:"<<stdDev<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Sum: 438.74
Average: 87.748
Standard deviation: 6.83264

Press any key to continue …
```

<u>Calculations of the sum, the average, the sum difference squares and the population standard deviation of index price returns expressed as percentages from an array format. In this case, we use n = 5.</u>

```
#include <iostream>
#include<math.h>
using namespace std;

int main()
{
// Calculations of the sum and average function of index prices from an array format.

// Identify the variables.

double sum ;
double average;
double sumDiffSqr;
double stdDev;

/* Identify the array of index prices. We start with a bracket that includes the five
observations followed by the numbers.*/

double arrayIndexPriceReturns[5] = {90.54,80.32,82.98,87.12,97.78};

/* The following expression inserts a variable x that tells the computer
that the first observation 0 is 90.54 and x is less than 5 to include all observations.*/

for (int x = 0; x < 5; x++)

//We insert the formula for summation.

sum += arrayIndexPriceReturns[x];

// We insert the formula for average.
average = sum/5;

/* We calculate the sumDiffSqr by inserting the expression for that
 tells the computer the numbers that will be included in the calculation.*/

sumDiffSqr = 0;
for (int x = 0; x < 5; x++)
{
        sumDiffSqr = sumDiffSqr + pow((arrayIndexPriceReturns[x]-average),2);
        }
// Insert the formula of the population standard deviation.
stdDev = sqrt(sumDiffSqr/5);


// Output functions and conclusion.
/* By the function cin we tell the program to allow us to insert the
index price returns in the console by pressing Enter.*/

cout<< "Enter arrayIndexPriceReturns" <<endl;
cin>> arrayIndexPriceReturns[1];
cin>> arrayIndexPriceReturns[2];
cin>> arrayIndexPriceReturns[3];
cin>> arrayIndexPriceReturns[4];
cin>> arrayIndexPriceReturns[5];
cout<< "Sum:"<<sum<<endl;
cout<< "Average:"<<average<<endl;
cout<<"Standard deviation:"<<stdDev<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter arrayIndexPriceReturns
90.54 Press Enter
80.32 Press Enter
82.98 Press Enter
87.12 Press Enter
97.78 Press Enter

Sum: 438.74
Average: 87.748
Standard deviation: 6.1113

Press any key to continue …
```

Calculations of the sum, average, sum difference squares and population standard deviation of call option returns from an array format. In this case, we use n = 5.

```
#include <iostream>
#include<math.h>
using namespace std;

int main()
{

/* Calculations of the sum, average, sum difference square and standard deviation
functions of call option returns from an array format.*/

// Identify the variables.

double sum ;
double average;
double stdDev;
double sumDiffSqr;

/* Identify the array of call option retunrs. We start with a bracket that includes the
five observations followed by the numbers.*/

double arraycallreturn[5] = {3.56,4.78,5.65,6.43,8.56};

/* The following expression inserts a varoable x that tells the computer
that the first observation 0 is 3.56 and x is less than 5 to include all observations.*/

for (int x = 0; x < 5; x++)

//We insert the formula for summation.
sum += arraycallreturn[x];

// We insert the formula for average.

average = sum/5;

/* We calculate the sumDiffSqr by inserting the expression for that
 tells the computer the numbers that will be included in the calculation.*/

sumDiffSqr = 0;

for (int x = 0; x < 5; x++)
{
        sumDiffSqr = sumDiffSqr + pow((arraycallreturn[x]-average),2);
        }
/* Insert the formula of the sample standard deviation by taking into account the
 formual n-1.*/
stdDev = sqrt(sumDiffSqr/5);

/* By the function cin we tell the program to allow us to insert the
call option retunrs in the console by pressing Enter.*/

cout<< "Enter arraycallreturn" <<endl;
cin>> arraycallreturn[1];
cin>> arraycallreturn[2];
cin>> arraycallreturn[3];
cin>> arraycallreturn[4];
cin>> arraycallreturn[5];

//Output functions.

cout<< "Sum:"<<sum<<endl;
cout<< "Average:"<<average<<endl;
cout<<"Standard deviation:"<<stdDev<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter arraycallreturn
3.56 Press Enter
4.78 Press Enter
5.65 Press Enter
6.43 Press Enter
8.56 Press Enter

Sum: 28.98
Average: 5.796
Standard deviation: 1.67898

Press any key to continue …
```

<u>Calculations of the sum, the average, the sum difference squares and the population standard deviation of call option returns from an array format. In this case, we use n = 5. Demonstration of the high and low level of call option returns.</u>

```
#include <iostream>
#include<math.h>
using namespace std;

int main()
{

/* Calculations of the sum, average, sum difference square and standard deviation
functions of call option returns. Demonstration of the high and low level of call option
returns. Insert observations as integer in the sum equation.*/

// from an array format.
// Identify the variables.

double high = 10.032 ;
double low  = 1.56 ;
double sum ;
int observations = 5 ;
double average;
double sumDiffSqr;
double stdDev;

/*Identify the array of call option retunrs. We start with a bracket that includes the
five observations followed by the numbers.*/

double arraycallreturn[5] = {3.56,4.78,5.65,6.43,8.56};

/*The following expression inserts a varoable x that tells the computer
that the first observation 0 is 3.56 and x is less than 5 to include all observations.*/

for (int x = 0; x < 5; x++)

//We insert the formula for summation.

sum += arraycallreturn[x];

// We insert the formula for average.

average = sum/observations;

/* We calculate the sumDiffSqr by inserting the expression for that
 tells the computer the numbers that will be included in the calculation.*/

sumDiffSqr = 0;
for (int x = 0; x < 5; x++)
{
        sumDiffSqr = sumDiffSqr + pow((arraycallreturn[x]-average),2);
        }
/* Insert the formula of the sample standard deviation by taking into account the
formual n-1.*/

stdDev = sqrt(sumDiffSqr/5);

/* By the function cin we tell the program to allow us to insert the
call option retunrs in the console by pressing Enter.*/

cout<< "Enter arraycallreturn" <<endl;
cin>> arraycallreturn[1];
cin>> arraycallreturn[2];
cin>> arraycallreturn[3];
cin>> arraycallreturn[4];
cin>> arraycallreturn[5];

//Output functions.

cout<<"High call option return:"<<high<<endl;
cout<<"Low call option return:"<<low<<endl;
cout<< "Sum:"<<sum<<endl;
cout<< "Average:"<<average<<endl;
cout<<"Standard deviation:"<<stdDev<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter arraycallreturn
3.56 Press Enter
4.78 Press Enter
5.65 Press Enter
6.43 Press Enter
8.56 Press Enter
High call option return: 10.032
Low call option return: 1.56
Sum: 28.98
Average: 5.796
Standard deviation: 1.67898

Press any key to continue …
```

Calculation of the geometric average based on percentages of stock prices.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{

/* The stock prices express percentages and we are trying to calculate the geometric
mean or average. We insert the percentage values of each stock in an array format.*/

double stockPrice[6];
stockPrice[1] = 10;
stockPrice[2] = 12;
stockPrice[3] = 13;
stockPrice[4] = 14;
stockPrice[5] = 15;
stockPrice[6] = 16;

// We define the variable that we want to calculate.

double Geometricaverage;

// We write the mathematical equation of the geometric mean.

Geometricaverage =
pow(stockPrice[1]*stockPrice[2]*stockPrice[3]*stockPrice[4]*stockPrice[5]*stockPrice[6],0.166666);

// The output function.
cout<<"The geometric average:"<<Geometricaverage<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
The geometric average: 13.18 ( to 2.d.p.).

Press any key to continue …
```

Calculation of the geometric average based on percentages of call option price returns.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{

/* The call option price returns are expressed in percentages and we are trying to
calculate the geometric mean or average. We insert the percentage values of each call
option in an array format.*/

double CallOptionPriceReturns[6];
CallOptionPriceReturns[1] = 5;
CallOptionPriceReturns[2] = 3;
CallOptionPriceReturns[3] = 2;
CallOptionPriceReturns[4] = 4;
CallOptionPriceReturns[5] = 7;
CallOptionPriceReturns[6] = 8;

// We define the variable that we want to calculate.

double Geometricaverage;

// We write the mathematical equation of the geometric mean.

Geometricaverage =
pow(CallOptionPriceReturns[1]*CallOptionPriceReturns[2]*CallOptionPriceReturns[
3]*CallOptionPriceReturns[4]*CallOptionPriceReturns[5]*CallOptionPriceReturns[6
],0.166666);

// The output.
cout<<"The geometric average:"<<Geometricaverage<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
The geometric average: 4.34 ( to 2.d.p.).

Press any key to continue …
```

Calculation of the geometric average based on percentages of put option price returns.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{

/* The put option price returns are expressed in percentages and we are trying to
calculate the geometric mean or average.*/

// We insert the percentage values of each put option in an array format.

double PutOptionPriceReturns[6];
PutOptionPriceReturns[1] = 2;
PutOptionPriceReturns[2] = 3.3;
PutOptionPriceReturns[3] = 2.45;
PutOptionPriceReturns[4] = 4.21;
PutOptionPriceReturns[5] = 7.4;
PutOptionPriceReturns[6] = 4.34;

// We define the variable that we want to calculate.

double Geometricaverage;

// We write the mathematical equation of the geometric mean.

Geometricaverage =
pow(PutOptionPriceReturns[1]*PutOptionPriceReturns[2]*PutOptionPriceReturns[3]
*PutOptionPriceReturns[4]*PutOptionPriceReturns[5]*PutOptionPriceReturns[6],0.1
66666);

// The output.

cout<<"The geometric average:"<<Geometricaverage<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
The geometric average: 3.60 ( to 2.d.p.).

Press any key to continue …
```

Calculation of the geometric average based on percentages of index option price returns.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{

/* The Index option price returns are expressed in percentages and we are trying to
calculate the geometric mean or average. We insert the percentage values of each
index option in an array format.*/

double IndexOptionPriceReturns[6];
IndexOptionPriceReturns[1] = 2;
IndexOptionPriceReturns[2] = 2.31;
IndexOptionPriceReturns[3] = 2.35;
IndexOptionPriceReturns[4] = 4.51;
IndexOptionPriceReturns[5] = 7.56;
IndexOptionPriceReturns[6] = 4.39;

// We define the variable that we want to calculate.

double Geometricaverage;

// We write the mathematical equation of the geometric mean.

Geometricaverage =
pow(IndexOptionPriceReturns[1]*IndexOptionPriceReturns[2]*IndexOptionPriceReturns[3]*IndexOptionPriceReturns[4]*IndexOptionPriceReturns[5]*IndexOptionPriceReturns[6],0.166666);

// The output.
cout<<"The geometric average:"<<Geometricaverage<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
The geometric average: 3.43 ( to 2.d.p.).

Press any key to continue …
```

Calculate the portfolio return, the variance and the standard deviation in C++ for options with different returns and standard deviations.

```
// Calculate the portfolio return, the variance and the standard deviation or risk of the
// portfolio in C++ for options with different returns and standard deviations.
// Call option A expected return                              0.6
// Call option A standard deviation                           0.2
// Call option A weight                                       0.35
// Correlation coefficient between A and B                    0.5
// Call option B expected return                              0.1
// Call option B standard deviation                           0.7
// Call option B weight                                       0.45
//Correlation coefficient between B and C                    -0.1
// Call option C expected return                              0.3
// Call option C standard deviation                           0.4
// Call option C weight                                       0.2
//Correlation coefficient between A and C                    -0.4

#include <iostream>
#include<cmath>
using namespace std;

int main()
{
 //Insert the given information by using the double function as we have fractions.

double CalloptionAexpectedreturn = 0.6;
double CalloptionAstandarddeviation = 0.2;
double CalloptionAweight = 0.35;
double CorrelationcoefficientbetweenAandB = 0.5;
double CalloptionBexpectedreturn = 0.1;
double CalloptionBstandarddeviation = 0.7;
double CalloptionBweight = 0.45;
double CorrelationcoefficientbetweenBandC = -0.1;
double CalloptionCexpectedreturn = 0.3;
double CalloptionCstandarddeviation = 0.4;
double CalloptionCweight = 0.2;
double CorrelationcoefficientbetweenAandC = -0.4;

/* Define the variables that we want to calculate. In other words, the portfolio return,
variance and standard deviation.*/

double Portfolioreturn;
double Portfoliovariance;
double PortfoliostdDev;

// Write the formula of portfolio return.
Portfolioreturn =
(CalloptionAexpectedreturn*CalloptionAweight)+(CalloptionBweight*CalloptionBexpectedreturn)+(CalloptionCexpectedreturn*CalloptionCweight);

// Write the formula for portfolio variance.

Portfoliovariance = pow(CalloptionAweight*CalloptionAstandarddeviation,2)+
pow(CalloptionBweight*CalloptionBstandarddeviation,2)+
pow(CalloptionCweight*CalloptionCstandarddeviation,2)
+(2*CalloptionAweight*CalloptionBweight*CalloptionAstandarddeviation*CalloptionBstandarddeviation*CorrelationcoefficientbetweenAandB)
+(2*CalloptionAweight*CalloptionCweight*CorrelationcoefficientbetweenAandC*CalloptionAstandarddeviation*CalloptionCstandarddeviation)
+(2*CalloptionBweight*CalloptionCweight*CorrelationcoefficientbetweenBandC*CalloptionBstandarddeviation*CalloptionCstandarddeviation);

// Write the formula for portfolio standard deviation.

PortfoliostdDev = sqrt(Portfoliovariance);

// Output functions.

cout<< "Portfolio return:"<<Portfolioreturn<<endl;
cout<< "Portfolio variance:"<<Portfoliovariance<<endl;
cout<< "Portfolio standard deviation:"<<PortfoliostdDev<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Portfolio return: 0.315
Portfolio variance:0.12 to (2.d.p.).
Portfolio standard deviation: 0.35 to (2.d.p.).


Press any key to continue …
```

Calculate the portfolio return, the variance and the standard deviation or risk of the portfolio.

```
/* Calculate the portfolio return, the variance and the standard deviation or risk of the
portfolio.*/

// Share A expected return                          0.11
// Share A standard deviation                       0.15
// Share A amount                                  50000
// Correlation coefficient between A and B    0.30
// Share B expected return                          0.25
// Share standard deviation                         0.20
// Share B amount                                   50000
// The total amount of the portfolio is             100000

#include <iostream>
#include<cmath>
using namespace std;

int main()
{

double shareAexpRet = 0.25;
double shareAstddev = 0.10;
double shareAamount = 50000;
double correlationAB = 0.40;
double shareBexpRet = 0.35;
double shareBstddev = 0.18;
double shareBamount = 50000;
double totalportfoliovalue = 100000;

/* Define the variables that we want to calculate.In other words, the portfolio return,
variance and the standard deviation.*/

double weightA;
double weightB;
double Portfolioreturn;
double Portfoliovariance;
double PortfoliostdDev;

// Calculate the weights for each share.

weightA = shareAamount / totalportfoliovalue;
weightB = shareBamount / totalportfoliovalue;

// Write the formula of portfolio return.

Portfolioreturn = (shareAexpRet*weightA)+(shareBexpRet*weightB);
// Write the formula for portfolio variance.

Portfoliovariance = pow(weightA*shareAstddev,2)+ pow(weightB*shareBstddev,2)
+(2*weightA*weightB*shareAstddev*shareBstddev*correlationAB);

// Write the formula for portfolio standard deviation.

PortfoliostdDev = sqrt(Portfoliovariance);

// Output functions.

cout<< "Portfolio return:"<<Portfolioreturn<<endl;
cout<< "Portfolio variance:"<<Portfoliovariance<<endl;
cout<< "Portfolio standard deviation:"<<PortfoliostdDev<<endl;
//Wait for the user to read the output on the console
system ("PAUSE");
return 0;
}
```

<u>Output</u>

After compiling and debugging , the DOS window or console will open and display the following results:

```
Portfolio return: 0.3
Portfolio variance:0.0142
Portfolio standard deviation: 0.119164


Press any key to continue …
```

Exercise and solution.

```
/*It is required to calculate the average return and the expected risk expressed as
standard deviation. The following numbers represent percentage returns for each class
of call, put and index options.*/


//Data provided

//Call option returns of Danske bank = 3.5, 4.7, 7.9, -4.2, 5.4, 10.3.
//Put option returns of Danske bank =  2.7, 4.9, 5.7, 8.5, 9.3, 11.5.
/*Index option returns of OMX Copenhagen stockmarket = 3.7, 7.9, 8.9, 10.2,-6.8,
5.9.*/

#include <iostream>
#include<math.h>
using namespace std;

int main()
{

//Identify the variables.

double sum1;
double sum2;
double sum3;
double average1;
double average2;
double average3;
double sumDiffSqr1;
double sumDiffSqr2;
double sumDiffSqr3;
double stdDev1;
double stdDev2;
double stdDev3;

// We insert the given numbers expressed in percentages in array format.

double arrayCalloptionreturns[6] = {3.5,4.7,7.9,-4.2,5.4,10.3};
double arrayPutoptionreturns[6] = {2.7, 4.9, 5.7, 8.5,9.3,11.5};
double arrayIndexoptionreturns[6] ={3.7, 7.9, 8.9, 10.2, -6.8, 5.9};

/* The following expression inserts a variable x that tells the computer
that the first observation 0 is for example 3.5 and x is less than 6 to include all
observations. We write the summation equation.*/

for (int x = 0; x < 6; x++)
sum1 += arrayCalloptionreturns[x];
for (int x = 0; x < 6; x++)
sum2 += arrayPutoptionreturns[x];
```

```
for (int x = 0; x < 6; x++)
sum3 += arrayIndexoptionreturns[x];

// We insert the formula for the average of different classes.
average1 = sum1/6;
average2 = sum2/6;
average3 = sum3/6;

/* We calculate the sumDiffSqr. We calculate the difference between each number
and the average. Then, we square it and we sum it.*/

sumDiffSqr1 = 0;
for (int x = 0; x < 6; x++)
{
        sumDiffSqr1 = sumDiffSqr1 + pow((arrayCalloptionreturns[x]-average1),2);
        }

sumDiffSqr2 = 0;
for (int x = 0; x < 6; x++)
{
        sumDiffSqr2 = sumDiffSqr2 + pow((arrayPutoptionreturns[x]-average2),2);
        }
sumDiffSqr3 = 0;
for (int x = 0; x < 6; x++)
{
        sumDiffSqr3 = sumDiffSqr3 + pow((arrayIndexoptionreturns[x]-average3),2);
        }

/* Insert the formula of the sample standard deviation by taking into account the
formual n-1. We are taking the square root of the sumDiffSqr and divide by the total
number of observations less 1.*/

stdDev1 = sqrt(sumDiffSqr1/5);
stdDev2 = sqrt(sumDiffSqr2/5);
stdDev3 = sqrt(sumDiffSqr3/5);

/* By the function cin we tell the program to allow us to insert the numerical values of
each class of call, put and index options in the console by pressing Enter. After
entering the numbers the program will calculate automatically the sum, the average
and the standard deviation.*/

// Output functions.
cout<< "Enter arrayCalloptionretunrs" <<endl;
cin>> arrayCalloptionreturns[0];
cin>> arrayCalloptionreturns[1];
cin>> arrayCalloptionreturns[2];
cin>> arrayCalloptionreturns[3];
cin>> arrayCalloptionreturns[4];
cin>> arrayCalloptionreturns[5];
cout<< "Sum:"<<sum1<<endl;
```

```
cout<< "Average:"<<average1<<endl;
cout<<" Sample standard deviation:"<<stdDev1<<endl;

cout<< "Enter arrayPutoptionretunrs" <<endl;
cin>> arrayPutoptionreturns[0];
cin>> arrayPutoptionreturns[1];
cin>> arrayPutoptionreturns[2];
cin>> arrayPutoptionreturns[3];
cin>> arrayPutoptionreturns[4];
cin>> arrayPutoptionreturns[5];
cout<< "Sum:"<<sum2<<endl;
cout<< "Average:"<<average2<<endl;
cout<<"Sample standard deviation:"<<stdDev2<<endl;

cout<< "Enter arrayIndexoptionretunrs" <<endl;
cin>> arrayIndexoptionreturns[0];
cin>> arrayIndexoptionreturns[1];
cin>> arrayIndexoptionreturns[2];
cin>> arrayIndexoptionreturns[3];
cin>> arrayIndexoptionreturns[4];
cin>> arrayIndexoptionreturns[5];
cout<< "Sum:"<<sum3<<endl;
cout<< "Average:"<<average3<<endl;
cout<<"Sample standard deviation:"<<stdDev3<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter arrayCalloptionreturns
3.5  Press Enter
4.7  Press Enter
7.9  Press Enter
-4.2 Press Enter
5.4  Press Enter
10.3 Press Enter

Sum: 27.6
Average: 4.6
Sample standard deviation: 4.95 to (2.d.p.).
```

```
Enter arrayPutoptionreturns
2.7   Press Enter
4.9   Press Enter
5.7   Press Enter
8.5   Press Enter
9.3   Press Enter
11.5 Press Enter

Sum: 42.6
Average: 7.1
Sample standard deviation: 3.23 to (2.d.p.).

Enter arrayIndexoptionreturns
3.7 Press Enter
7.9 Press Enter
8.9 Press Enter
10.2 Press Enter
-6.8  Press Enter
5.9 Press Enter

Sum: 29.8
Average: 4.97 to (2.d.p.).
Sample standard deviation: 6.20



Press any key to continue …
```

Example of pricing a forward contract and solution.

A forward contract of 6 month has a market price of 52 Pounds when the spot price of the underlying commodity is 49 Pounds. There are no costs of carry and the discount rate is 7 percent. Calculate the value of the forward contract.

Solution

The current price of the forward contract (F_0) should be equal to the value of the underlying commodity, (S_0) at the discount rate r. Thus, the equation is as follows:

$$F_0 = S_0(1+r)$$

$$F_0 = S_0(1+r) = 49\left(1+\frac{0.07}{2}\right) = 50.715 \text{ pounds}$$

Application of pricing a forward contract in C++

```
//Pricing a forward contract.

#include <iostream>
using namespace std;

int main()
{
   double valueforwardcontract;
   double spotprice = 49;
   double discountrate = 0.07;

// Insert the mathematical formula.

 valueforwardcontract = spotprice*(1+discountrate/2);

// Output functions.
cout<< "Enter spotprice" <<endl;
cin>> spotprice;
cout<< "Enter discountrate" <<endl;
cin>> discountrate;
cout<< "Value forward contract:"<<valueforwardcontract<<endl;
system ("PAUSE");
return 0;
}
```

Output

The numerical value is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter spot price
49
Enter discount rate
0.07
Value forward contract: 50.715
Press any key to continue …
```

Example of interest rate payment and solution.

A trader wants to calculate the interest amount that he / she will receive in three months from a forward contract of a Euribor deposit paying a Euro deposit rate of 3.55%. The principal amount is 300,000

The mathematical formula is as follows:

$$\text{Interest payment} = \text{principal x [interest rate x} (t_{\text{days}} / 360)]$$

Interest payment = 300,000 x [0.0355 x (90/360)]
Interest payment = 2662.5 Euro.

First example: application of interest payment in C++

```
//Interest payment.

#include <iostream>
using namespace std;

int main()
{
   double interestpayment;
   double principal = 300000;
   double interestrate = 0.0355;
   double days = 90;

 // Insert the mathematical formula.

 interestpayment = principal*(interestrate *days/360);

// Output functions.

cout<< "Enter principal" <<endl;
cin>> principal;
cout<< "Enter interestrate" <<endl;
cin>> interestrate;
cout<< "Interest payment:"<<interestpayment<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

The numerical value is expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal
300000
Enter interest rate
0.0355
Interest payment: 2662.5

Press any key to continue …
```

<u>Second example: application of interest payment in C++ by changing the layout of the cout and cin function.</u>

```
//Interest payment.

#include <iostream>
using namespace std;

int main()
{
   double interestpayment;
   double principal = 300000;
   double interestrate = 0.0355;
   int days = 90;

// Insert the mathematical formula.

interestpayment = principal*(interestrate *days/360);

// Output functions.

cout<< "Enter principal, interest rate" <<endl;
cin>> principal >> interestrate;
cout<< "Interest payment:"<<interestpayment<<endl;
system ("PAUSE");
return 0;
}
```

Output

The numerical value is expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, interest rate
300000 0.0355
Interest payment: 2662.5

Press any key to continue …
```

Third example: application of interest payment in C++ by adding formatting output values in the output functions.

```
//Interest payment.

#include <iostream>
using namespace std;

int main()
{
   double interestpayment;
   double principal = 300000;
   double interestrate = 0.0355;
   double days = 90;
   interestpayment = principal*(interestrate *days/360);

//Formating output values and output functions.

cout.setf(ios::fixed);             // First statement.
cout.setf(ios::showpoint);       // Second statement.
cout.precision(3);                 // Third statement. To show the final numerical
                                     // value rounded to three decimal places.

cout<< "Enter principal, interest rate" <<endl;
cin>> principal >> interestrate;
cout<< "Interest payment:"<<interestpayment<<endl;
system ("PAUSE");
return 0;
}
```

Output

The numerical value is expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, interest rate
300000 0.0355
Interest payment: 2662.500

Press any key to continue …
```

Fourth example: application of interest payment in C++ by adding the Euro currency in the output functions.

```
//Interest payment.

#include <iostream>
using namespace std;

int main()
{

//Identify the variables.

   double interestpayment;
   double principal = 300000;
   double interestrate = 0.0355;
   double days = 90;

Insert the mathematical formula.

interestpayment = principal*(interestrate *days/360);

//Formating output values and output functions.

cout.setf(ios::fixed);               // First statement.
cout.setf(ios::showpoint);          // Second statement.
cout.precision(3);                  // Third statement. To show the final numerical
                                     // value rounded to three decimal places.

cout<< "Enter principal, interest rate" <<endl;
cin>> principal >> interestrate;
cout<< "Interest payment in Euro:"<<interestpayment<<endl;
system ("PAUSE");
return 0;
}
```

Output

The numerical value is expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, interest rate
300000 0.0355
Interest payment in Euro: 2662.500

Press any key to continue …
```

Example and solution of calculating profits and losses on futures contracts.

An investor buys 8 S&P 500 futures contracts at $2189. He has closed the futures contract position with a price of $2236. The multiplier for S&P 500 futures contracts is 250 dollars. Did he record a profit or a loss?

Solution

The mathematical formula is as follows:

$$\Pr ofit / Loss = number \text{ of contracts x contract size x}\left(\mathrm{f}_{\mathrm{T}} - f_0\right)$$

Where: f_T is the final contract price.
f_0 is the initial contract price.

$$Gain = 8 \text{ x } 250 \text{ x}\left(2236 - 2189\right) = 94000 \text{ USD}$$

Application of calculating profits and losses on futures contracts in C++

```
// calculating profits and losses on futures contracts.

#include <iostream>
using namespace std;

int main()
{
   double profitorloss ;
   int numberofcontracts =8;
   double buyingprice = 2189;
   double sellingprice = 2236;
   int multiplier = 250;

// Insert the mathematical formula.

profitorloss = numberofcontracts*multiplier*(sellingprice - buyingprice);

// Output functions.

cout<< "Enter number of contracts" <<endl;
cin>> numberofcontracts;
cout<< "Enter buying price" <<endl;
cin>> buyingprice;
cout<< "Enter selling price" <<endl;
cin>> sellingprice;
cout<< "Enter multiplier" <<endl;
cin>> multiplier;
cout<< "Profit or loss:"<<profitorloss<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

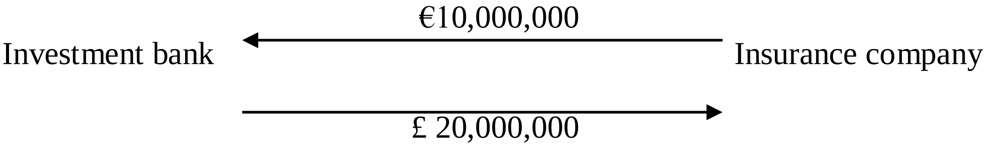
The numerical value is expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter number of contracts
8
Enter buying price
2189
Enter selling price
2236
Enter multiplier
250
Profit or loss: 94000   ( In this case, we have profit or gain).

Press any key to continue …
```

Example and solution of semiannually currency swap between an investment bank and an insurance company.

The investment bank borrows 20,000,000 Pounds from the insurance company at a fixed rate of 5% for 1 year. The insurance company borrows from the investment bank 10,000,000 Euros at a fixed rate of 5% for 1 year. Calculate the interest payments for the three years, if we assume semiannual payments?

At the beginning of the contract



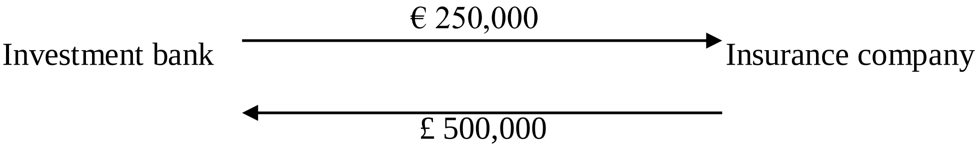
The insurance company pays the investment bank the following interest payment.

£20,000,000 x 0.05/2 = 500,000 Pounds.

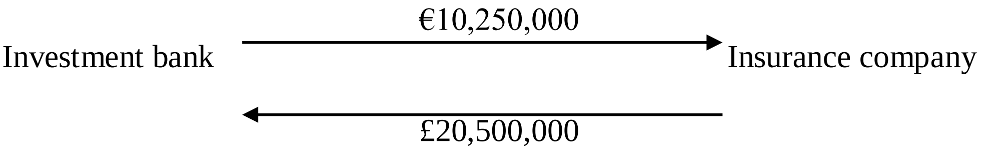
The investment bank pays the insurance the following interest payment.

€10,000,000 x 0.05/2 =250,000 Euros.

The following arrows illustrate the interest payments.



At the end of the third year, the two parties exchange the principal amounts in addition to the final interest payments.



Application of semiannually currency swap between an investment bank and an insurance company in C++

```
//Semiannual swaps interest payments between an investment bank and an insurance
//company.

#include <iostream>
using namespace std;

int main()
{
   double bankpaysinsurance ;
   double insurancepaysbank ;
   double bankborrowsfrominsurance =20000000;
   double insuranceborrowsfrombank = 10000000;
   double interestrate = 0.05;
   int semiannualpayment = 2;

//Insert the mathematical formulas.

bankpaysinsurance = insuranceborrowsfrombank *(interestrate/semiannualpayment);
insurancepaysbank = bankborrowsfrominsurance *(interestrate/semiannualpayment);


// Output functions.

cout<< "Enter bank borrows from insurance" <<endl;
cin>> bankborrowsfrominsurance ;
cout<< "Enter insurance borrows from bank " <<endl;
cin>> insuranceborrowsfrombank;
cout<< "Enter interest rate" <<endl;
cin>> interestrate;
cout<< "Enter semi annual payment" <<endl;
cin>> semiannualpayment;
cout<< " Bank pays insurance:"<< bankpaysinsurance <<endl;
cout<< " Insurance pays  bank :"<< insurancepaysbank <<endl;


system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter bank borrows from insurance
20000000
Enter insurance borrows from bank
10000000
Enter interest rate
0.05
Enter semi annual payment
2
Bank pays insurance: 250000
Insurance pays bank: 500000

Press any key to continue …
```

Example and solution of interest rate swap.

Interest rate swaps are very popular agreements between two parties in the debt or fixed-income department of the investment banks. The fixed income department of Bank A pays a fixed rate of 5.5% upon the principal of 50,000,000 Pounds. In contrast, Bank B pays a floating or reference rate of LIBOR accounted to 7.2%. The payment frequency is every 6 months for 2 years.

$$\begin{tikzcd}[column sep=huge] \text{Bank A} \arrow[r, shift left, "\text{Fixed payment : £50,000,000 x 0.055/2 = 1,375,000}"] & \text{Bank B} \arrow[l, shift left, "\text{Floating payment: £50,000,000 x 0.072/2 = 1,800,000}"] \end{tikzcd}$$

Suppose in the second year that the LIBOR has increased by 12 basis points or 0.12%. Then, the floating payment will change and the calculation will be as follows:

Second year floating payment for Bank B = 50,000,000 x 0.0732 / 2 = 1,830,000 Pounds.

Bank A will continue to pay the same fixed amount, namely, £50,000,000 x 0.055/2 = £1,375,000.

Application of interest rate swap in C++

```
//Semiannual swaps interest payments between two parties.

#include <iostream>
using namespace std;

int main()
{
   double fixedpayment ;
   double floatingpayment1 ;
   double floatingpayment2 ;
   double principal = 50000000;
   double interestrate1 = 0.055;
   double interestrate2 = 0.072;
   double interestrate3 = 0.0732;
   int semiannualpayment = 2;
// Insert the mathematical formulas.

fixedpayment = principal*(interestrate1/semiannualpayment);
floatingpayment1 = principal*(interestrate2/semiannualpayment);
floatingpayment2 = principal*(interestrate3/semiannualpayment);

// Output functions.

cout<< " Fixed payment:"<< fixedpayment <<endl;
cout<< " Floating payment 1 :"<< floatingpayment1 <<endl;
cout<< " Floating payment 2 :"<< floatingpayment2 <<endl;


system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter bank borrows from insurance
Fixed payment: 1.375 e+006

Floating payment 1: 1.8 e+006

Floating payment 2:  1.83 e+006

Press any key to continue …
```

Delta – neutral hedge.

Delta – neutral hedge is common used in risk management to keep the value of the portfolio neutral due to changes in the share price. It is achieved from a long position in a share and a short position in a call option. The mathematical formula to determine the number of options is as follows:

$$Delta\text{ - neutral hedge} = \frac{\text{Number of shares hedged}}{\text{delta of call option}}$$

Thus, if the investment bank has bought 30,000 shares of Vodafone and the delta of the call option of the same company is 0.50, then the numbers of call options that are needed to purchase to form a delta-neutral hedge are as follows:

Delta hedge = 30,000 / 0.50 = 60,000 options or 600 option contracts.

Application of delta- neutral hedge in C++

```
// Delta neutral hedge.

#include <iostream>
using namespace std;

int main()
{
   double deltaneutralhedge ;
   double numberofshares = 30000 ;
   double deltaofcalloption = 0.50 ;

// Insert the mathematical equation.

deltaneutralhedge = numberofshares/deltacalloption;


// Output function.

cout<< " Delta neutral hedge:"<< deltaneutralhedge <<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Delta neutral hedge: 60000

Press any key to continue …
```

Weighted mean price of a portfolio.

Let’s assume that we have a portfolio of four options with their market prices and the number of shares bought. It is required to calculate the weighted mean price of the portfolio.

Options	Price expressed in $	Number of shares	Weight	Weight x Price
A	14.00	300	0.4	5.6
B	12.00	200	0.3	3.6
C	8.00	100	0.1	0.8
D	5.00	150	0.2	1
Total		750	1	

Source: author’s calculation

The mathematical formula is as followed:

$$\overline{X}_w = \sum w_i P_i$$

Where : $\overline{X}_w$ is the weighted mean price of the portfolio.

w_i are the weights.

P_i are the prices.

By substituting the numbers from the Table into the equation we have the following:

$$\overline{X}_w = 5.6 + 3.6 + 0.8 + 1 = 11.$$

Application of weighted mean price of a portfolio in C++

```
// Weighted mean price of a portfolio.

#include <iostream>
using namespace std;

int main()
{
double portfoliomean ;

double price[4];
price[1] = 14;
price[2] = 12;
price[3] = 8;
price[4] = 5;

double weight[4];
weight[1] = 0.4;
weight[2] = 0.3;
weight[3] = 0.1;
weight[4] = 0.2;

// Insert the mathematical equation.

portfoliomean =
(price[1]*weight[1]+price[2]*weight[2]+price[3]*weight[3]+price[4]*weight[4]);


// Output function.
cout<<" Weighted mean of the portfolio:"<<portfoliomean<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Weighted mean of the portfolio: 11

Press any key to continue …
```

Example of Eurodollar futures.

An investor wants to calculate the futures price of a 1-month Eurodollar time deposits based on a LIBOR rate of 3.40%. The initial principal is 1 million Pounds.

Solution

The price of the Eurodollar futures contract will be as follows:

$$\textit{Futures}\text{ price of Eurodollar time deposit } = \text{Principal x}\left[1-\text{LIBOR rate x}\left(\frac{\text{days}}{360}\right)\right]$$

$$\textit{Futures}\text{ price of Eurodollar time deposit } = 1{,}000{,}000\text{ x}\left[1-0.034\text{ x}\left(\frac{30}{360}\right)\right] = \text{....... Pounds}$$

$$\text{Futures price of Eurodollar time deposit } = 1{,}000{,}000\text{ x } 0.997166667 = 997166.67\,\textit{pounds}$$

Application of Eurodollar futures in C++

```
//The price of Eurodollar future.

#include <iostream>
using namespace std;

int main()
{
   double Eurodollarfuture;
   double principal = 1000000;
   double interestrate = 0.034;
   int days = 30;

// Insert the mathematical formula.

Eurodollarfuture = principal*(1- interestrate *days/360);

// Output function.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<< "Enter principal, interest rate" <<endl;
cin>> principal >> interestrate;
cout<< "Price of Eurodollar future:"<<Eurodollarfuture<<endl;
system ("PAUSE");
return 0;
}
```

Output

The price of the Eurodollar future is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, interest rate
1000000   0.034
Price of Eurodollar future: 997166.67 pounds

Press any key to continue …
```

Example of forward rate agreement.

An investment bank buys a 4 x 6 FRA from a building society in the UK for 4.0% by paying 2 million Pounds. 80 days latter, LIBOR is 5%. Who will receive the FRA payment and for how much?

Solution.

The mathematical formula for Forward Rate Agreement, (FRA) is as follows:

$$FRA \text{ Payment } = \text{Notional amount x} \left(\frac{(\text{variable rate - fixed rate})(t_{days}/360)}{1 + \text{var}\,iable \text{ rate}(\text{t}_{\text{days}}/360)} \right)$$

The timeline schedule is as follows:

t = 0 4months 6 months

$$FRA \text{ Payment } = 2{,}000{,}000 \text{ x} \left(\frac{(0.05 - 0.04)(120/360)}{1 + 0.05(80/360)} \right)$$

= ………. Please complete the calculation.

Helpful hint: If the FRA payment is positive, then, the building society is the payer and the investment bank receives the FRA payment.

If the FRA payment is negative, then the investment bank is the payer and the building society receives the FRA payment.

Application of forward rate agreement in C++

```
//The price of forward rate agreement.

#include <iostream>
using namespace std;

int main()
{
   double FRApayment;
   double principal = 2000000;
   double variablerate = 0.05;
   double fixedrate = 0.04;

// Insert the mathematical formula.
FRApayment= principal*(variablerate - fixedrate)*0.333333333/1.011111111;
// Output function.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(1);
cout<< "Forward rate payment:"<<FRApayment<<endl;
system ("PAUSE");
return 0;
}
```

Output

The payment is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following result:

```
Forward rate payment:  6593.4 pounds

Press any key to continue …
```

Example of futures on Treasury bills.

An investor wants to calculate the futures price of a 3-month Treasury bills. Treasury bills are short-term notes of limited period of 1- month, and 3 - month respectively. The initial principal is 5 million Pounds and the discount rate is 3.50%.

Solution.

The price of the Treasury bill futures will be as follows:

$$\textit{Futures}\text{ price Treasury Bill} = \text{Principal x}\left[1 - \text{discount rate x}\left(\frac{t_{\text{days}}}{360}\right)\right]$$

Please complete the calculation …….

Application of futures on Treasury bills in C++

```
//The price of  futures on Treasury bills.

#include <iostream>
using namespace std;

int main()
{
    double futurestreasurybills;
    double principal = 5000000;
    double discountrate = 0.035;
    int days = 90;

// Insert the mathematical formula.

futurestreasurybills = principal*(1- discountrate *days/360);

// Output function.

cout<< "Enter principal, discount rate" <<endl;
cin>> principal >> discountrate;
cout<< "Price of futures on Treasury bills:"<<futurestreasurybills<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, interest rate
5000000   0.035
Price of  futures on Treasury bills: 4956250 or 4.95625 e+006

Press any key to continue …
```

Example of stock index futures.

For example, if you bought 10 contracts of the Dow Jones Industrial index at 10,000 and you expected an aggressive bull market that reaches the value of 16,000, then, the 6000 points increase are multiplied by the standardized value of 250. If the initial principal of investment is $100,000, the mathematical formula for the gains will be as follows:

$10 \text{ x } 100{,}000 \text{ x } 6000 \text{ x } 250 = 1.5 \text{ x } 10^{12}$ Dollars

Application of stock index futures in C++

```
//The gains from stock index futures

#include <iostream>
using namespace std;

int main()
{
   double gains;
   double principal = 100000;
   double pointsincrease = 6000;
   int number_of_contracts = 10;
   int multiplier = 250;

// Insert the mathematical formula.

gains = principal*pointsincrease*multiplier*number_of_contracts;

// Output functions.

cout<< "Enter principal, pointsincrease, multiplier, number_of_contracts" <<endl;
cin>> principal >> pointsincrease>>multiplier>>number_of_contracts;
cout<< "Gains from stock index futures:"<<gains<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, points increase, multiplier,number_of_contracts
100000
6000
250
10
```

```
Gains from stock index futures: 1.5 e+012

Press any key to continue …
```

Example of currency futures.

A manufacturer wants to calculate the currency futures price of EURO/USD traded in Frankfurt derivative market. For example, in August the contract is quoted as 1.35 EURO /USD and the contract size is 20,000. The principal is 300,000 Euro.

Solution.

The contract price of 1 contract is as follows:

$$\begin{aligned}&\textit{Futures}\text{ price of currency}\\&\text{future EURO/USD} = \text{principal } * \text{contract size x August quoted price} = \quad \ldots..\ \text{Euro}\end{aligned}$$

Please complete the calculatio n....

Application of currency futures in C++

```
//The futures price of currency.

#include <iostream>
using namespace std;

int main()
{
   double futurespriceofcurrency;
   double principal = 300000;
   double quote = 1.35;
   int contractsize = 20000;

// Insert the mathematical formula.
futurespriceofcurrency = principal*quote*contractsize ;

// Output functions.

cout<< "Enter principal, quote, contract size" <<endl;
cin>> principal >> quote>> contractsize;
cout<< "Futures price of currency:"<<futurespriceofcurrency<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

The result is in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Enter principal, quote, contract size
300000
1.35
20000

Futures price of currency: 8.1 e+009

Press any key to continue …
```

Example of calculating implied volatility and the new theoretical call option price due to volatility.

```
# include<cmath>
#include <iostream>
using namespace std;

double Call(double S, double K, double r, double q, double T,
  double sig);


int main()
{
  double S =50.0, K=50.0 , r=0.04, q = 0.01, T = 0.50;

  double sig, optPrice, TheoreticalPrice, tolerance ;
  int i;

  optPrice = 8.34;  // Option price.
  sig = 0;
  tolerance = 0.1;

  /* Specification for the sigma or volatility with tolerance level
  in relation with the theoretical and market option price.*/
  for(i = 1; i < 50 ; i++)
  {
    sig += tolerance;

    TheoreticalPrice = Call(S, K, r, q, T, sig);

    if((TheoreticalPrice) > (optPrice))
    {
      tolerance = -(tolerance * 0.1);
      tolerance = -tolerance;

    }
  }

 // Output functions.

  cout << "Implied volatility: " << sig << endl;
  cout << "Option price: " << optPrice << endl;
  cout << "Theoretical Price due to volatility:"<< TheoreticalPrice << endl;

  system("PAUSE");
  return 0;
}

// Calculation of the cumulative normal distribution.
```

```
double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}
// Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S*exp(-q*T)*norm_cdf(d1) - K*exp(-r*T)*norm_cdf(d2);
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Implied volatility: 0.611  or 61.1%
Option price: 8.34
Theoretical price due to volatility: 8.82

Press any key to continue …
```

<u>Example of calculating implied volatility and the new theoretical put option price due to volatility.</u>

```
# include<cmath>
#include <iostream>
using namespace std;

double Put(double S, double K, double r, double q, double T,
  double sig);


int main()
{
  double S =70.0, K=70.0 , r=0.04, q = 0.01, T = 0.50;

  double sig, optPrice, TheoreticalPrice, tolerance ;
  int i;

  optPrice = 5.25;  //  Option price.
  sig = 0;
  tolerance = 0.1;

  /* Specification for the sigma or volatility with tolerance level
  in relation with the theoretical and market option price.*/
  for(i = 1; i < 70 ; i++)
  {
    sig += tolerance;

    TheoreticalPrice = Put(S, K, r, q, T, sig);

    if((TheoreticalPrice) > (optPrice))
    {
      tolerance = -(tolerance * 0.1);
      tolerance = -tolerance;

    }
  }

 // Output functions.

  cout << "Implied volatility: " << sig << endl;
  cout << "Option price: " << optPrice << endl;
  cout << "Theoretical price due to volatility: "<<TheoreticalPrice<<endl;

  system("PAUSE");
  return 0;
}
```

```
// Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}
// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*norm_cdf(-d2) - S*exp(-q*T)*norm_cdf(-d1) ;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Implied volatility: 0.311  or 31.1%
Option price: 5.25
Theoretical price due to volatility: 5.55

Press any key to continue …
```

Please try based on the previous examples to illustrate the following example in C++. If you have difficulties e-mail me.

Active return, active risk and information ratio.

Active return is the difference in returns between a portfolio and the index or benchmark that is measured.

Active return = r_p - r_b

Where: r_p is the portfolio return.
r_b is the benchmark or index return.

If the portfolio return is 0.80 and the benchmark return of the index is 0.70 then, the active return is………

Please complete the calculation.

Active risk or tracking error is the standard deviation of the difference of returns between a portfolio and the benchmark or index.

$$Active\,\text{risk} = \sqrt{\frac{\sum(\mathrm{r_p} - r_b)^2}{n-1}}$$

$Where:$ $\mathrm{r_p}$ is portfolio return.
$\mathrm{r_b}$ is portfolio benchmark return. It could be for example an index.
n is the number of assets.

If the portfolio return is 0.80, the number of assets is 10 and the benchmark return of the index is 0.40 then, the active risk is………

Please complete the calculation.

Information ratio shows the consistency of the fund manager towards the active return. The mathematical formula is as follows:

$$IR = \frac{\bar{r}_p - \bar{r}_b}{\sqrt{\frac{\sum (\mathrm{r_p} - r_b)^2}{n-1}}}$$

$Where : \bar{r}_p$ is the average of portfolio return.

$\bar{\mathrm{r}}_\mathrm{b}$ is the average benchmark or index return.

$\sqrt{\frac{\sum (r_p - r_b)^2}{n-1}}$ is the active risk

It could be calculated very easily in Excel software. I will illustrate a simple table with the relevant calculations.

Day	r_p	r_b	r_p - r_b
1	0.03	0.02	0.01
2	0.02	0.014	0.006
3	-0.04	0.034	-0.074
4	0.05	0.067	-0.017
5	0.08	0.012	0.068
6	-0.01	-0.056	0.046
7	0.07	0.031	0.039
8	0.034	0.023	0.011
9	-0.021	0.015	-0.036
10	0.045	0.001	0.044
Average	**0.0258**	**0.0161**	
Standard deviation			**0.043**

Source: author's calculation

By substituting the values that we have found in terms of r_p =0.0258, r_b = 0.0161 and standard deviation = 0.043 in the following equation we have:

$$R = \frac{\bar{r}_p - \bar{r}_b}{\sqrt{\frac{\sum (\mathrm{r_p} - r_b)^2}{n-1}}}$$

$$R = \frac{0.0258 - 0.0161}{0.043} = \frac{0.0097}{0.043} = 0.23\,(\text{to 2d.p.})$$

But what is the interpretation of the information ratio of 0.23. It means that the fund manager gained around 23 basis points of active return per unit of active risk. The higher is this number, the better the manager is performing in relation to active risk.

Please convert the following exercise in C++ language programming.

Calculate the balance at the end of the 5th day from changes in the futures prices from 90 to 89, 91, 95, 97, 99. The trader has bought 50 futures contract for settlement in May. The initial margin is $8.

Day	Beginning balance	Futures price	Gain or loss	Ending balance
0	400	90	0	400
1	400	89	(50)	
2	350	91	100	
3	450	95	200	
4	650	97	100	
5	750	99	100	

Source: author's illustration.

Beginning balance = initial margin x number of contracts.
Beginning balance = 8 x 50 = $400

Please complete the values of the column ending balance.

Application of the srand() rand() functions. The srand() function is related to the system time. The computer generates every time different random numbers. The function rand() is related to random numbers and is converting the numbers to the value between 0 and 199. It is very useful method that is used in simulations such as Monte Carlo simulation to price derivatives products.

```
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
  int k;
  int arraySize ;


  cout << "Enter array size: "<<endl;
  cin >> arraySize;

  int dataarray[arraySize];

  // Generate the array with random numbers between 0 and 199
  srand(time(0));
  for(k = 0; k < arraySize; k++)
  {
    dataarray[k] = rand()%200;
    cout << dataarray[k] << endl;
  }

 system("PAUSE");
  return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results. **Every time you will get different numbers by using the functions srand() and rand().**

```
Enter array size:
8
134
161
83
129
57
24
20
58
Press any key to continue …
```

Example of profit or loss of a call option portfolio.

```
// Profit or loss of a call option portfolio.

#include <iostream>
using namespace std;

int main()
{

  double portfoliovalue1, portfoliovalue2, portfoliovalue3,
  portfoliovalue4, portfoliovalue5,portfoliovalue6,portfoliovalue7,
  portfoliovalue8,portfoliovalue9,portfoliovalue10 ;    // portfolio value.

 double shareprice[10] ;              // number of positions of share prices.
  double strikeprice[10];             // number of positions of strike prices.
  double optionpremium [10];     // Call option premiums.
  double contractsize = 100;        // contract size.

/* Identifying the number of positions in relation to the share prices, strike prices and
option premium.*/

  shareprice[1] = 10.25;
  shareprice[2] = 20.13;
  shareprice[3] = 30.22;
  shareprice[4] = 40.56;
  shareprice[5] = 50.89;
  shareprice[6] = 51.23;
  shareprice[7] = 53.33;
  shareprice[8] = 54.63;
  shareprice[9] = 66.23;
  shareprice[10] = 68.23;

  strikeprice[1] = 3.34;
  strikeprice[2] = 8.34;
  strikeprice[3] = 18.78;
  strikeprice[4] = 28.34;
  strikeprice[5] = 31.12;
  strikeprice[6] = 22.02;
  strikeprice[7] = 34.23;
  strikeprice[8] = 47.12;
  strikeprice[9] = 48.45;
  strikeprice[10] = 50.12;

  optionpremium[1] = 1.50;
  optionpremium[2] = 2.30;
  optionpremium[3] = 2.56;
  optionpremium[4] = 3.10;
  optionpremium[5] = 3.25;
  optionpremium[6] = 3.45;
  optionpremium[7] = 3.56;
  optionpremium[8] = 4.32;
  optionpremium[9] = 5.12;
  optionpremium[10] = 5.78;

// Calculate the call option portfolio values.

    portfoliovalue1 = (shareprice[1]-(strikeprice[1]+ optionpremium[1]))* contractsize;
    portfoliovalue2 = (shareprice[2]-(strikeprice[2]+ optionpremium[2]))* contractsize;
    portfoliovalue3 = (shareprice[3]-(strikeprice[3]+ optionpremium[3]))* contractsize;
    portfoliovalue4 = (shareprice[4]-(strikeprice[4]+ optionpremium[4]))* contractsize;
    portfoliovalue5 = (shareprice[5]-(strikeprice[5]+ optionpremium[5]))* contractsize;
    portfoliovalue6 = (shareprice[6]-(strikeprice[6]+ optionpremium[6]))* contractsize;
    portfoliovalue7 = (shareprice[7]-(strikeprice[7]+ optionpremium[7]))* contractsize;
    portfoliovalue8 = (shareprice[8]-(strikeprice[8]+ optionpremium[8]))* contractsize;
    portfoliovalue9 = (shareprice[9]-(strikeprice[9]+ optionpremium[9]))* contractsize;
    portfoliovalue10 = (shareprice[10]-(strikeprice[10]+ optionpremium[10]))*
    contractsize;

   //Output functions.

   cout <<"Portfolio value 1:"<< portfoliovalue1<<endl;
   cout <<"Portfolio value 2:"<< portfoliovalue2<<endl;
   cout <<"Portfolio value 3:"<< portfoliovalue3<<endl;
   cout <<"Portfolio value 4:"<< portfoliovalue4<<endl;
   cout <<"Portfolio value 5:"<< portfoliovalue5<<endl;
   cout <<"Portfolio value 6:"<< portfoliovalue6<<endl;
   cout <<"Portfolio value 7:"<< portfoliovalue7<<endl;
   cout <<"Portfolio value 8:"<< portfoliovalue8<<endl;
   cout <<"Portfolio value 9:"<< portfoliovalue9<<endl;
   cout <<"Portfolio value 10:"<< portfoliovalue10<<endl;

  system("PAUSE");
  return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Portfolio value 1: 541
Portfolio value 2: 949
Portfolio value 3: 888
Portfolio value 4: 912
Portfolio value 5: 1652
Portfolio value 6: 2576
Portfolio value 7: 1554
Portfolio value 8: 319
Portfolio value 9: 1266
Portfolio value 10: 1233

Press any key to continue …
```

Example of profit or loss of a put option portfolio.

```
// Profit or loss of a put option portfolio.

#include <iostream>
using namespace std;

int main()
{

  double portfoliovalue1, portfoliovalue2, portfoliovalue3,
  portfoliovalue4, portfoliovalue5,portfoliovalue6,portfoliovalue7,
  portfoliovalue8,portfoliovalue9,portfoliovalue10 ;                    // portfolio value.

  double shareprice[10] ;            // number of positions of share prices.
  double strikeprice[10];             // number of positions of strike prices.
  double optionpremium [10];    // Put option premiums.
  double contractsize = 100;       // contract size.

/* Identifying the number of positions in relation to the share prices, strike prices and
option premium.*/


  shareprice[1] = 10.25;
  shareprice[2] = 20.13;
  shareprice[3] = 30.22;
  shareprice[4] = 40.56;
  shareprice[5] = 50.89;
  shareprice[6] = 51.23;
  shareprice[7] = 53.33;
  shareprice[8] = 54.63;
  shareprice[9] = 66.23;
  shareprice[10] = 68.23;

  strikeprice[1] = 3.34;
  strikeprice[2] = 8.34;
  strikeprice[3] = 18.78;
  strikeprice[4] = 28.34;
  strikeprice[5] = 31.12;
  strikeprice[6] = 22.02;
  strikeprice[7] = 34.23;
  strikeprice[8] = 47.12;
  strikeprice[9] = 48.45;
  strikeprice[10] = 50.12;

  optionpremium[1] = 1.50;
  optionpremium[2] = 2.30;
  optionpremium[3] = 2.56;
  optionpremium[4] = 3.10;
  optionpremium[5] = 3.25;
  optionpremium[6] = 3.45;
  optionpremium[7] = 3.56;
  optionpremium[8] = 4.32;
  optionpremium[9] = 5.12;
  optionpremium[10] = 5.78;

// Calculate the call option portfolio values.


    portfoliovalue1 = (strikeprice[1]- optionpremium[1]-shareprice[1])* contractsize;
    portfoliovalue2 = (strikeprice[2]- optionpremium[2]- shareprice[2])* contractsize;
    portfoliovalue3 = (strikeprice[3]- optionpremium[3]- shareprice[3])* contractsize;
    portfoliovalue4 = (strikeprice[4]- optionpremium[4]- shareprice[4])* contractsize;
    portfoliovalue5 = (strikeprice[5]- optionpremium[5]- shareprice[5])* contractsize;
    portfoliovalue6 = (strikeprice[6]- optionpremium[6]- shareprice[6])* contractsize;
    portfoliovalue7 = (strikeprice[7]- optionpremium[7]- shareprice[7])* contractsize;
    portfoliovalue8 = (strikeprice[8]- optionpremium[8]- shareprice[8])* contractsize;
    portfoliovalue9 = (strikeprice[9]- optionpremium[9]- shareprice[9])* contractsize;
    portfoliovalue10 = (strikeprice[10]- optionpremium[10]- shareprice[10])*
contractsize;

   cout <<"Portfolio value 1:"<< portfoliovalue1<<endl;
   cout <<"Portfolio value 2:"<< portfoliovalue2<<endl;
   cout <<"Portfolio value 3:"<< portfoliovalue3<<endl;
   cout <<"Portfolio value 4:"<< portfoliovalue4<<endl;
   cout <<"Portfolio value 5:"<< portfoliovalue5<<endl;
   cout <<"Portfolio value 6:"<< portfoliovalue6<<endl;
   cout <<"Portfolio value 7:"<< portfoliovalue7<<endl;
   cout <<"Portfolio value 8:"<< portfoliovalue8<<endl;
   cout <<"Portfolio value 9:"<< portfoliovalue9<<endl;
   cout <<"Portfolio value 10:"<< portfoliovalue10<<endl;

  system("PAUSE");
  return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Portfolio value 1: -841
Portfolio value 2: -1409
Portfolio value 3: -1400
Portfolio value 4: -1532
Portfolio value 5: -2302
Portfolio value 6: -3266
Portfolio value 7: -2266
Portfolio value 8: -1183
Portfolio value 9: -2290
Portfolio value 10: -2389

Press any key to continue …
```

One – period binomial method.

A share is traded in the Danish stock exchange at 50 DKK. First of all, we would like to calculate the possible price changes based on probabilities. The risk - free rate is 5% and the value of the call option has an exercise price of 50 DKK. It is required to calculate the expected value of the option at t =1. In addition, it is required to calculate the value of the option today discounted at the given risk-free rate.

We assume two scenarios. The first one is that the share will increase by 20% or by a factor of 1 + 0.20 = 1.20. The second one is that the share will decrease by 0.833. The number 0.833 is calculated as 1 / 1.20 = 0.833.

The mathematical formulas are as follows:

Down move = D = 1 / up move = 1 / 1.20 = 0.833

$$\pi_U = \text{risk - neutral probabilit y of an up movement } = \frac{1 + \text{rf - D}}{\text{U - D}}$$

Where : rf is the risk - free rate.

D is the down move.

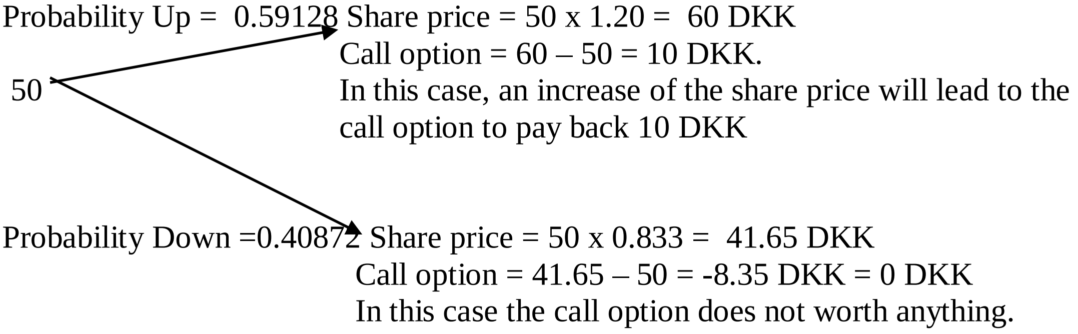
U is the up move.

$$\pi_{\text{U}} = \frac{1 + 0.05 - 0.833}{1.20 - 0.833} = \frac{0.217}{0.367} = 0.59128$$

$$\pi_D = risk \text{ - neutral probabilit y of a down movement } = 1 - 0.59128 = 0.40872$$

Thus, the payoff or the one-period binomial tree for share and option prices for the two possible scenarios will be as follows:

t = 0 t = 1



Probability Up = 0.59128 Share price = 50 x 1.20 = 60 DKK

Call option = 60 – 50 = 10 DKK.

50

In this case, an increase of the share price will lead to the call option to pay back 10 DKK

Probability Down =0.40872 Share price = 50 x 0.833 = 41.65 DKK

Call option = 41.65 – 50 = -8.35 DKK = 0 DKK

In this case the call option does not worth anything.

The expected value of the call option at t = 1 is as follows:

Expected call option = (10 x 0.59128) + (0 x 0.40872) = 5.9128 + 0 = 5.9128 DKK.

The value of the call option discounted at 5% risk – free rate is as follows:

C_{today} = 5.9128 / 1.05 = 5.63 DKK (to 2 d.p.).

Application of one – period binomial method in C++

```
//One - period binomial method.

#include <iostream>
using namespace std;

int main()
{
   double shareprice = 50;
   double exerciseprice = 50;
   double upmove = 1.20;
   double downmove = 0.833;
   double riskfreerate = 0.05;
   double riskprobofupmove;
   double riskprobofdownmove;
   double shareprice1;
   double shareprice2;
   double calloptionprice1;
   double calloptionprice2;
   double expectedcalloption;
   double valueofcalldiscounted;


// Insert the mathematical formulas of risk probability of up and down move.

riskprobofupmove = (1+riskfreerate-downmove) / (upmove - downmove);
riskprobofdownmove = 1 - riskprobofupmove;

// The payoff or one period binomial tree for share and option prices is as follows:

shareprice1 = shareprice * upmove;
calloptionprice1 = shareprice1 - exerciseprice;

shareprice2 = shareprice *downmove;
calloptionprice2 = shareprice2 - exerciseprice;
calloptionprice2 =0;

// The expected value of the call option at t=1 is as follows:

expectedcalloption = (calloptionprice1* riskprobofupmove) + (calloptionprice2 *
riskprobofdownmove);

// The value of the call option discounted at 5% risk - free rate is as follows:

valueofcalldiscounted = expectedcalloption / 1.05;
// Output functions.

cout<< "Risk probability of up move:" <<riskprobofupmove<<endl;
cout<< "Risk probability of down move:"<<riskprobofdownmove <<endl;
cout<< "Share price 1:"<<shareprice1 <<endl;
cout<< "Call option price 1:"<<calloptionprice1 <<endl;
cout<< "Share price 2:"<<shareprice2 <<endl;
cout<< "Call option price 2:"<< calloptionprice2<<endl;
cout<< "Expected call option:"<<expectedcalloption <<endl;
cout<< "Value of call option discounted:"<<valueofcalldiscounted <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Risk probability of up move: 0.59128
Risk probability of down move: 0.40872
Share price 1: 60
Call option price 1: 10
Share price 2:  41.65
Call option price 2: 0
Expected call option: 5.9128
Value of call option discounted: 5.63

Press any key to continue …
```

Two – period binomial method.

This method is based on the one – period binomial method, but it is extended to include a second period, t = 2. Thus, we have three periods. t = 0, t =1 and t =2. We use the same example of the previous section but extended to an additional period.

A share is traded in the Danish stock exchange at 50 DKK. First of all, we would like to calculate the possible price changes based on probabilities. The risk - free rate is 5% and the value of the call option has an exercise price of 50 DKK. It is required to calculate the expected value of the option. In addition, it is required to calculate the value of the option today discounted at the given risk-free rate.

We assume two scenarios. The first one is that the share will increase by 20% or by a factor of 1 + 0.20 = 1.20. The second one is that the share will decrease by 0.833. The number 0.833 is calculated as 1 / 1.20 = 0.833.

The mathematical formulas are as follows:

Down move = D = 1 / up move.

$$\pi_U = \text{risk - neutral probabilit y of an up movement } = \frac{1 + \text{rf - D}}{\text{U - D}}$$

Where : rf is the risk - free rate.

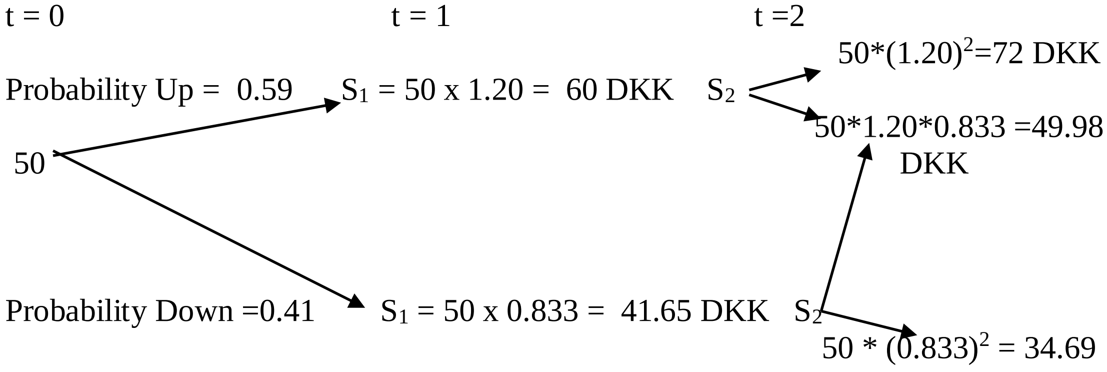
D is the down move.

U is the up move.

$$\pi_{\text{U}} = \frac{1 + 0.05 - 0.833}{1.20 - 0.833} = \frac{0.217}{0.367} = 0.59128$$

$$\pi_D = \textit{risk} \text{ - neutral probabilit y of a down movement } = 1 - 0.59128 = 0.40872$$

Thus, the payoff or the two-period binomial tree for share prices for the two possible scenarios will be as follows:



The value of the call options for the period t =2 for both ups and downs are as follows:

Upside movement

C_1 = 72 – 50 = 22

C_2 = 49.98 – 50 = -0.02 =0 Negative values mean that the option does not worth.

Downside movement

C_1 = 49.98 -50 = -0.02 =0

C_2 = 34.69 -50 = -15.31 =0

The expected value of the call options for both movements for period t =1 are calculated based on the following equations:

Upside movement

$$Call_{up} = \frac{\left(\text{Probabilit y up x C}_1\right) + \left(\text{Pr}\,\textit{obability}\text{ down xC}_2\right)}{1 + \text{rf}} = \frac{\left(0.59128x22\right) + (0.40872x0)}{1.05} = \frac{13.008}{1.05}$$

$$Call_{up} = 12.3887 \text{ DKK}$$

Downside movement

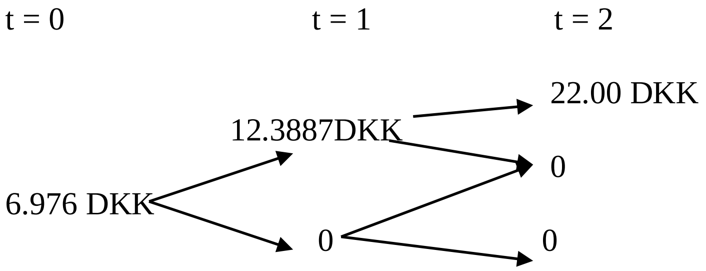
$$Call_{down} = \frac{\left(\text{Pr}\,\textit{obability}\text{ up x C}_1\right) + \left(\text{Pr}\,\textit{obability}\text{ down x C}_2\right)}{1 + \text{rf}} = \frac{\left(0.59128x\,0\right) + (0.40872x\,0)}{1.05} = 0$$

The value of the call option today discounted is calculated based on the following formula:

$$Call_{today} = \frac{\left(\text{Pr}\,\textit{obability}\text{ up x Call}_{\text{up}}\right) + \left(\text{Pr}\,\textit{obability}\text{ down x Call}_{\text{down}}\right)}{1 + rf} = \frac{0.59128 \text{ x } 12.3887 + 0}{1.05}$$

$$Call_{today} = \frac{7.32519}{1.05} = 6.976 \text{ (to 3.d.p.)}$$

The final format of the two-period binomial tree for call options is as follows:



Application of two – period binomial method in C++

```
//Two - period binomial method.

#include <iostream>
#include<cmath>
using namespace std;

int main()
{
   double shareprice = 50;
   double exerciseprice = 50;
   double upmove = 1.20;
   double downmove = 0.833;
   double riskfreerate = 0.05;
   double riskprobofupmove;
   double riskprobofdownmove;
   double shareprice1;
   double shareprice2;
   double shareprice3;
   double shareprice4;
   double shareprice5;
   double calloptionprice1;
   double calloptionprice2;
   double calloptionprice3;
   double calloptionprice4;
   double calloptionprice5;
   double expectedcalloptionupside;
   double expectedcalloptiondownside;
   double valueofcalldiscounted;


// Insert the mathematical formulas of risk probability of up and down move.

riskprobofupmove = (1+riskfreerate-downmove) / (upmove - downmove);
riskprobofdownmove = 1 - riskprobofupmove;

// The payoff or two period binomial tree for share and option prices is as follows:

shareprice1 = shareprice * upmove;

shareprice2 = shareprice *downmove;

shareprice3 = shareprice * pow(upmove,2);
calloptionprice3 = shareprice3 - exerciseprice;

shareprice4 = shareprice * pow(downmove,2);
calloptionprice4 = shareprice4 - exerciseprice;
calloptionprice4 =0;
shareprice5 = shareprice * upmove * downmove;
calloptionprice5 = shareprice5 - exerciseprice;
calloptionprice5 =0;

// The expected value of the call option at t=2 is as follows:
expectedcalloptionupside = (riskprobofupmove * calloptionprice3
+ riskprobofdownmove* calloptionprice5) / 1.05;

expectedcalloptiondownside = (riskprobofupmove *calloptionprice4
+ riskprobofdownmove * calloptionprice5) / 1.05;

// The value of the call option discounted at 5% risk - free rate is as follows:

valueofcalldiscounted = (riskprobofupmove * expectedcalloptionupside
+riskprobofdownmove* expectedcalloptiondownside)/ 1.05;

// Output functions.

cout<< "Risk probability of up move:" <<riskprobofupmove<<endl;
cout<< "Risk probability of down move:"<<riskprobofdownmove <<endl;
cout<< "Share price 1:"<<shareprice1 <<endl;
cout<< "Share price 2:"<<shareprice2 <<endl;
cout<<" Share price 3:"<<shareprice3<<endl;
cout<<" Share price 4:"<<shareprice4<<endl;
cout<<" Share price 5:"<<shareprice5<<endl;
cout<< "Call option price 3:"<< calloptionprice3<<endl;
cout<< "Call option price 4:"<< calloptionprice4<<endl;
cout<< "Call option price 5:"<< calloptionprice5<<endl;
cout<< "Expected call option upside:"<<expectedcalloptionupside <<endl;
cout<< "Expected call option downside:"<<expectedcalloptiondownside <<endl;
cout<< "Value of call option discounted:"<<valueofcalldiscounted <<endl;

system ("PAUSE");
return 0;
}
```

<u>Output</u>

After compiling and debugging , the DOS window or console will open and display the following results:

```
Risk probability of up move: 0.59128
Risk probability of down move: 0.40872
Share price 1: 60
Share price 2: 41.65
Share price 3: 72
Share price 4: 34.69
Share price 5: 49.98
Call option price 3: 22
Call option price 4: 0
Call option price 5: 0
Expected call option upside: 12.3887
Expected call option downside: 0
Value of call option discounted: 6.976

Press any key to continue …
```

Stock index option.

Let’s assume that a practitioner wants to buy a stock index call option that is related to the general movement of the S&P 500 index. The dollar multiplier for S&P 500 option contract is 250 dollars. The strike price of the index is 1240. The August premium is 20 index points.

Question.

Calculate the cost of the August call and the net profit involved if the S&P 500 index reaches the price of 1540.

Solution.

Cost of the August call = premium expressed in index points x dollar multiplier
Cost of the August call = 20 x 250 = 5000 USD

The net profit incurred is calculated as follows:

(Ending index value – beginning index value x dollar multiplier) – cost of the August call.

(1540 – 1240) x 250 – 5000 = 70,000 USD.

Application of stock index option in C++

```
//Cost of call and net profit of stock index option.

#include <iostream>
using namespace std;

int main()
{
    double costofcall;
    double netprofit ;
    int premium = 20;
    int multiplier  = 250;
   double endingindexvalue = 1540;
   double beginningindexvalue = 1240;


// Insert the mathematical formula.

costofcall = premium * multiplier ;
netprofit = (endingindexvalue - beginningindexvalue) * multiplier - costofcall;
// Output functions.

cout<< "Cost of call:"<<costofcall<<endl;
cout<< "Net profit:"<<netprofit<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Cost of call: 5000
Net profit:  70000

Press any key to continue …
```

Exercise of an European interest rate option.

Calculate the profit or loss of the interest rate difference between the actual and the strike rate of a call option. Let's assume that an interest call option on a 6 –month Eurodollar has a strike rate of 4%. At expiration, the 6-month Eurodollar rate is 5%. The invested principal is 50,000,000 Euros. The duration of the contract is expressed in days divided by 360 days. Thus, 6 –months equal 180 days.

The mathematical formula for profit or loss is as follows:

$$\text{Pr}\mathit{ofit}\text{ or loss of call interest rate option } = \text{ Principal *(actual rate - strike rate)*}(\frac{\text{days}}{360})$$

$$\text{Pr}\mathit{ofit}\text{ or loss of call interest rate option } = \text{ 50,000,000*(0.05 -0.04) *}({}^{180}\!/_{360})$$

$$\text{Pr}\mathit{ofit}\text{ or loss of call interest rate option } =$$

Application of an European interest rate option in C++

```
//Profit or loss of call interest rate option.

#include <iostream>
using namespace std;

int main()
{
    double profitorlosscallinterestrateoption;   // call interest rate option.
    double principal = 50000000;
    double actualrate  = 0.05;
    double strikerate = 0.04;
   double duration = 180;
   double days = 360;

// Insert the mathematical formula.

profitorlosscallinterestrateoption = principal *(actualrate – strikerate)*duration/days;

// Output function.

cout<< "Profit or loss of call interest rate option:"<<profitorlosscallinterestrateoption
<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figure is expressed in Euros. After compiling and debugging , the DOS window or console will open and display the following result:

```
Profit or loss of call interest rate option: 250000

Press any key to continue …
```

Currency option.

Let’s assume that a wealthy investor buys a call currency option because it expects a rise in the exchange rate parity of the EURO against the USD, EURO/USD. The spot exchange rate is 1/1.3568. The strike price is 1.3568. The premium expressed as cents per Euro is 1.56. The initial principal is 100,000 USD.

Question.

Calculate the net profit involved if the strike price increases to 1.3987.

Solution.

The wealthy investor to record a profit, he or she should add to the strike price 1.3568 the premium expressed as cents per Euro. In our case, the breakeven point is 1.3568 + 0.0156 = 1.3724. Above the price of 1.3724, he or she starts to record a profit. If the strike price reaches 1.3987, then the profit will be as follows:

Initial principal x (Ending strike price – beginning strike price – premium paid) =
100,000 x (1.3987 – 1.3568 – 0.0156) = 2630 USD net profit

Application of currency option in C++

```
//Profit or loss of currency option.

#include <iostream>
using namespace std;

int main()
{
   double netprofit;
   double principal = 100000;
   double endingstrikeprice  = 1.3987;
   double beginningstrikeprice = 1.3568;
   double premiumpaid = 0.0156;

// Insert the mathematical formula.

netprofit = principal *(endingstrikeprice – beginningstrikeprice - premiumpaid);

// Output function.

cout<< "Net profit:"<<netprofit <<endl;

system ("PAUSE");
return 0;
}
```

Output

The figure is expressed in USD. After compiling and debugging , the DOS window or console will open and display the following result:

```
Net profit: 2630

Press any key to continue …
```

Example of calculating the payments of an interest rate cap based on different LIBOR rates.

Interest rate cap is an agreement between two parties where one party pay the other at a specified period of time in which the interest rate or London interbank offered rate ,LIBOR, exceeds the strike price. It is used this contract to hedge against interest rate fluctuations. Let’s assume that an interest cap has a value of 5.5% and the LIBOR prices for the next three years are 7.5%, 8.3% and 4.65%. The principal is $50,000,000 and the payments take place semiannually.

The mathematical formula that is used to calculate the payments of each year is as follows:

Interest rate payment =[principal x (LIBOR rate – cap rate)/0.5]

Therefore, the first year interest rate payment is as follows:

Year 1

Interest rate payment = [50,000,000 x (0.075 – 0.055)/2] = $ ………

Application of payments of an interest rate cap based on different LIBOR rates in C++

```
// Payments of an interest rate cap based on different LIBOR rates.

#include <iostream>
using namespace std;

int main()
{
   double interestpay1;
   double interestpay2;
   double interestpay3;
   double principal = 50000000;
   double liborrate1  = 0.075;
   double liborrate2  = 0.083;
   double caprate = 0.055;
   double duration = 2;

// Insert the mathematical formula.

interestpay1 = principal *(liborrate1 - caprate) /duration ;
interestpay2 = principal *(liborrate2 - caprate) / duration ;
// Output functions.

cout<< "Interest rate payment 1:"<<interestpay1 <<endl;
cout<< "Interest rate payment 2:"<<interestpay2 <<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Interest rate payment 1: 500000
Interest rate payment 2: 700000

Press any key to continue …
```

A six month interest rate cap has a rate of 0.08 and the principal is 20,000,000 USD. The settlements is done quarterly. The first quarter the 3 –month libor rate is 0.096 and the second quarter the rate is 0.087. Calculate the payoff for the cap the first and the second quarter?

Solution.

Payoff the first quarter = 20,000,000 * $(0.096 - 0.08)/4$ = 80000 USD

Payoff the second quarter = 20,000,000 * $(0.087 - 0.08)/4 =$=35000 USD

Application of payoff of interest rate cap in C++

```
//Payoff of interest rate cap.

#include <iostream>
using namespace std;

int main()
{
   double payoff1;
   double payoff2;
   double principal = 20000000;
   double caprate = 0.08;
   double liborrate1 = 0.096;
   double liborrate2 = 0.087;
   double period = 4;

// Insert the mathematical formulas.

payoff1 = (principal*(liborrate1-caprate)/period);
payoff2 = (principal*(liborrate2-caprate)/period);

// Output functions.
cout<< "Payoff1:" <<payoff1<<endl;
cout<< "Payoff2:" <<payoff2<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Payoff1: 80000
Payoff2: 35000
```

Press any key to continue …

Swaption.

An investor purchased a 1 year European swaption with exercise price 7.50%. The principal is 30,000,000USD. The floating rate payments are based on LIBOR. The 90, 180, 270, and 360 day annualized LIBOR rates and present value factors are as follows:

LIBOR	Rate	Present value factors
90 day LIBOR	4%	0.990099
180 day LIBOR	5.5%	0.973236
270 day LIBOR	6%	0.956938
360 day LIBOR	7%	0.934579

Calculate the semi-annual and annualized swap rate?
Calculate the net cash flow at each payment and the value of swaption at maturity.

Solution

The present value factors are calculated as follows:

$$PV_{90\text{ days}} = \frac{1}{1+\left(0.04*\frac{90}{360}\right)} = 0.990099$$

$$PV_{180\text{ days}} = \frac{1}{1+\left(0.055*\frac{180}{360}\right)} = 0.973236$$

$$PV_{270\text{ days}} = \frac{1}{1+\left(0.06*\frac{270}{360}\right)} = 0.956938$$

$$PV_{360\text{ days}} = \frac{1}{1+\left(0.07*\frac{360}{360}\right)} = 0.934579$$

Semi - annual swap rate =

$$\frac{1-0.934579}{0.990099+0.973236+0.956938+0.934579} = \frac{0.065421}{3.854852} = 0.01697$$

Annulaized swap rate = $0.01697*360/90 = 0.06788$ or 6.788%

The swaption is in the money because the exercise price 7.50% is greater than the market annualized swap rate 6.788%.

The net cash flow is as follows:

netCashFlow = (0.075 – 0.06788)*90/360 * 30000000 = 53400 USD

The value of swaption at maturity is as follows:

valueSwaption = 53400 *$(0.990099 + 0.973236 + 0.956938 + 0.934579)$ = 205,849.1 USD

Application of swaption in C++

```
//Swaption.

#include <iostream>
using namespace std;

int main()
{
   double semiAnnualRate;
   double annualizedSwapRate;
   double netCashFlow;
   double valueSwaption;
   double exerciseRate = 0.075;
   double principal = 30000000;
   double PV90;
   double PV180;
   double PV270;
   double PV360;
   double rate90 = 0.04;
   double rate180 = 0.055;
   double rate270 = 0.06 ;
   double rate360 = 0.07;

// Insert the mathematical formulas.

PV90 = 1 /(1+(rate90 * 90/360));
PV180 = 1/(1+(rate180 * 180/360));
PV270 = 1/(1+(rate270 * 270/360));
PV360 = 1/(1+(rate360* 360/360));

semiAnnualRate = (1-PV360)/(PV90 +PV180 +PV270 +PV360);
annualizedSwapRate = semiAnnualRate * (360/90);
// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(5);
cout<< "Semi - annual rate:" <<semiAnnualRate<<endl;
cout<< "Annualized swap rate:" <<annualizedSwapRate<<endl;

system ("PAUSE");
return 0;
}
```

<u>Output</u>

The figures are expressed in % and USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Semi – annual rate: 0.01697
Annualized swap rate: 0.06788   or 6.788%

Press any key to continue …
```

Please repeat the above swaption problem and calculate the net cash flow and the value of swaption at maturity expressed in USD.

Put –call parity.

The put – call parity shows the relationship between a call and a put option with the same expiration, strike and share prices. The mathematical formula is as follows:

$$C + \frac{X}{(1+rf)^T} = P + S$$

$Where:$ C is the price of the call.

P is the price of the put.

X is the strike price of both the put and the call.

r_f is the rsik - free rate.

T is the time to maturity or expiration .

S is the share price for both the call and the put option.

Example

Calculate the price of a call if the price of a put is 5.34 Pounds, the share price is 94, the strike price is 97, the risk-free rate is 4% and the time to maturity, T=0.5. By rearranging formula (20) and solving for the price of a call, we have the following results:

$$C = P + S - \frac{X}{(1+rf)^T} = 5.34 + 94 - \frac{97}{(1.04)^{0.5}}$$

$C = 4.22$ pounds

Application of put - call parity in C++

```
// Put - call parity. Calculating the price of a call.

#include <iostream>
# include <cmath>
using namespace std;

int main()
{
  double call;
  double put = 5.34;
  double shareprice = 94;
  double strikeprice = 97;
  double  disriskfreerate = 1.04;
  double maturity = 0.5;
// Insert the mathematical formula.

call  =  put + shareprice - strikeprice / pow(disriskfreerate,maturity);


// Output function.

cout<< "Call :"<<call <<endl;
system ("PAUSE");
return 0;
}
```

Output

The figure is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following result:

```
Call: 4.22

Press any key to continue …
```

Example of margin payments in the clearinghouse.

An investor open a margin account with a minimum initial margin of 5,200 dollars per contract. The maintenance margin is 1,500 dollars per contract. He/she buys 7 July wheat futures contracts with a standard contract size of 200 bushels priced at 230 dollars per bushel. The July contract size in the next 2 days recorded the prices of 220 and 210 dollars per bushel. Show the cash flows, gains and losses for the buyer and the seller?

Solution.

Initial margin = 7 x 5,200 = 36,400 dollars

Maintenance margin = 7 x 1,500 = 10,500 dollars

The cash flows, gains and losses for the next two days will be as follows:

Day 1

Cash flows for the buyer's: 36,400 + 14,000 = 50,400 dollars. The investor's recorded a gain.

Cash flows for the seller's: 36,400 – 14,000 = 22,400 dollars. The investor's recorded a loss.

Helpful hint: the amount of 14,000 is calculated from the difference of the bushel prices, (f_0-f) x number of contracts x standard contract size. Thus, we have: (230-220) x 7 x 200 = 14,000 dollars. The same principle applies for the second day but with different bushel price.

Day 2

Please do the calculations for the second day.

This is why I am stressing the importance that the investors should buy or sell **ONLY** when the market is aggressively increasing or falling. If he/she gets trapped in a market that is not strong bull or bear, then, he/she will have to compensate regularly for the losses or he/she will experience a very small profit. If the amounts as the days passes is below the maintenance margin, then, the investor's will receive a margin call and he/she should add the capital required to proceed with the contract. If the investor's is short of money, then, the position is closed with the incurred losses.

Application of margin payments in the clearinghouse in C++

```
// Margin payments.

#include <iostream>
using namespace std;

int main()
{
   double cashflowbuyer ;
   double cashflowseller ;
   double initialmargin = 5200;
   double maintenancemargin = 1500;
   int contractnumber = 7;
   int bushelcontractsize = 200;
   double bushelprice1 = 230;
   double bushelprice2 = 220;
   double initialmarginwithcontract ;
   double maintenancemarginwithcontract;
   double bushelcontract ;

// Insert the mathematical formulas.

initialmarginwithcontract = initialmargin * contractnumber;
maintenancemarginwithcontract = maintenancemargin * contractnumber;

bushelcontract = (bushelprice1 – bushelprice2) * bushelcontractsize *
contractnumber;

cashflowbuyer = initialmarginwithcontract + bushelcontract;
cashflowseller = initialmarginwithcontract – bushelcontract;

// Output functions.

cout<< "Initial margin with contract :"<<initialmarginwithcontract <<endl;
cout<< "Maintenance margin with contract :"<<maintenancemarginwithcontract
<<endl;
cout<< "Bushel contract :"<<bushelcontract <<endl;
cout<< "Cash flow buyer :"<<cashflowbuyer <<endl;
cout<< "Cash flow seller :"<<cashflowseller <<endl;


system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in dollars. After compiling and debugging , the DOS window or console will open and display the following results:

```
Initial margin with contract: 36400
Maintenance margin with contract: 10500
Bushel contract:  14000
Cash flow buyer: 50400
Cash flow seller:  22400

Press any key to continue …
```

Example of calculating a margin call due to changes of price futures.

An investor has opened a short position of 20 Soybean futures contract. The initial margin was 500 Pounds and the maintenance margin was 430 Pounds per contract. The price change that will create a margin call is as follows:

Solution.

$$\text{Margin call created at a price} = \frac{\text{initial margin - maintenanc e margin}}{\text{contract size}} = \frac{500 - 430}{20} = 3.5$$

Application of margin call in C++

```
// Margin call.

#include <iostream>
using namespace std;

int main()
{
   double initialmargin = 500;
   double maintenancemargin = 430;
   int futurescontract = 20;
   double margincall;

// Insert the mathematical formula.

 margincall =( initialmargin – maintenancemargin) / futurescontract;

// Output function.

cout<< "Margin call created at a price :"<<margincall<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figure is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following result:

```
Margin call created at a price: 3.5

Press any key to continue …
```

Example of simulating share prices by showing different price scenarios of the highest and the lowest value.

```
/* Simulated share prices by showing different price scenarios
of the highest and the lowest value.*/

#include <iostream>
#include <ctime>
#include<cmath>

using namespace std;

int main()
{

int i ;
int numberOfSimulation = 6;
const int maxSize = 8;
double normSamps[maxSize];
double sharePrice[maxSize];
double randomNumber[maxSize];


double S0 = 60;     // initial share price.
double r = 0.03;    //  risk – free interest rate.
double q = 0.02;    // dividend yield.
double sig = 0.20;  // volatility
double dt = 1/250;  // maturity expressed by 250 trading days.

srand (time(0));
for (i=0; i< maxSize; i++)
{
     randomNumber[i] =((rand()%30000)+ 3000)/300;
}

    for(i=0; i <maxSize; i++)
    { normSamps[i]=0;
      sharePrice[i] =0;
    }

sharePrice[0]= S0;
int numberofsteps = 2;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Initial share price:"<<S0<<endl;


 for (int i =0; i<1; i++)
```

```
    {

     sharePrice[i] = sharePrice[i-1] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
1]*randomNumber[i]*sig *sqrt(dt));   // We simulate the share price by using random
number.

    sharePrice[i-1] = sharePrice[i-2] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
2]*randomNumber[i-2]*sig *sqrt(dt));   // We simulate the share price by using
random number.

      //Output functions.

        cout<<"Share price[i]:"<<sharePrice[i]<<endl;
         cout<<"Share price[i-1]:"<<sharePrice[i-1]<<endl;

      }


    system ("PAUSE");
    return 0;
    }
```

<u>Output</u>

The figures are expressed in Euro. **<u>Everytime, you will get different figures as we have used random numbers without defining a Box-Muller algorithm</u>**. In my case, after compiling and debugging , the DOS window or console will open and display the following results:

```
Initial share price in Euro: 60
Share price[i] in Euro: 42.000
Share price[i-1] in Euro: 82.000
Press any key to continue …
```

Example of simulating share prices by showing different price scenarios.

```
// Simulated share prices by showing different price scenarios.

#include <iostream>
#include <ctime>
#include<cmath>

using namespace std;

int main()
{

int i ;
int numberOfSimulation = 6;
const int maxSize = 8;
double normSamps[maxSize];
double sharePrice[maxSize];
double randomNumber[maxSize];


double S0 = 60;     // initial share price.
double r = 0.03;    //  risk – free interest rate.
double q = 0.02;    // dividend yield.
double sig = 0.20;  // volatility
double dt = 1/250;  // maturity expressed by 250 trading days.

srand (time(0));
for (i=0; i< maxSize; i++)
{
        randomNumber[i] =((rand()%30000)+ 3000)/300;
}

       for(i=0; i <maxSize; i++)
       { normSamps[i]=0;
         sharePrice[i] =0;
       }

sharePrice[0]= S0;
int numberofsteps = 2;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Initial share price in Euro:"<<S0<<endl;

 for (int i =0; i<1; i++)

     {
```

```
     sharePrice[i] = sharePrice[i-1] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
1]*randomNumber[i-1]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

    sharePrice[i-1] = sharePrice[i-2] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
2]*randomNumber[i-2]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

      sharePrice[i-2] = sharePrice[i-3] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
3]*randomNumber[i-3]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

        sharePrice[i-3] = sharePrice[i-4] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
4]*randomNumber[i-4]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

         sharePrice[i-4] = sharePrice[i-5] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
5]*randomNumber[i-5]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

       sharePrice[i-5] = sharePrice[i-6] *exp((r-q-0.5*sig*sig)*dt + normSamps[i-
6]*randomNumber[i-6]*sig
       *sqrt(dt));   // We simulate the share price by using random number.

      //Output functions.

          cout<<"Share price[i]in Euro:"<<sharePrice[i]<<endl;
          cout<<"Share price[i-1] in Euro:"<<sharePrice[i-1]<<endl;
          cout<<"Share price[i-2] in Euro:"<<sharePrice[i-2]<<endl;
          cout<<"Share price[i-3] in Euro:"<<sharePrice[i-3]<<endl;
          cout<<"Share price[i-4] in Euro:"<<sharePrice[i-4]<<endl;
          cout<<"Share price[i-5] in Euro:"<<sharePrice[i-5]<<endl;


       }


     system ("PAUSE");
     return 0;
     }
```

Output

The figures are expressed in Euro. **Everytime, you will get different figures as we have used random numbers without defining a Box-Muller algorithm**. In my case, after compiling and debugging , the DOS window or console will open and display the following results:

```
Initial share price in Euro: 60
Share price[i] in Euro: 100.000
Share price[i-1] in Euro: 85.000
Share price[i-2] in Euro: 38.000
Share price[i-3] in Euro: 49.000
Share price[i-4] in Euro: 32.000
Share price[i-5] in Euro: 17.000
Press any key to continue …
```

Example of calculating a call option using Monte Carlo simulation. Part of the formulas were taken from the book Financial Modeling using C++, p458.

```
// Calculation of a call option using Monte Carlo simulation.

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double termval( double K, double S0);

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double sharePrice;
double optPrice;
double payoff = 0 ;
double randomNumber;


double S0 = 60;     // initial share price
double K = 50;      // strike price
double r = 0.05;    // risk - free interest rate
double q = 0.02;    // dividend yield
double sig = 0.45;  // volatility
double T = 0.5;     // maturity

srand (time(0));

       randomNumber = ((rand()%10000)+1000)/100;

      for(i=0; i <maxSize; i++)
      { normSamps[i]=0;
      }

 for (int i = 0; i <= numberOfSimulation; i ++)
 {
    sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
      *sqrt(T));            // We simulate the share price by using random numbers.

    payoff += termval(K, sharePrice);       /* We use the for loop to calculate the sum
of payoffs for the given simulation number.*/
    }
    optPrice = (payoff * exp(-r*T))/ numberOfSimulation;  /*  Calculate the price of
the call option based on the payoff.*/
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Call option price:"<<optPrice<<endl;

    system ("PAUSE");
    return 0;
    }
double termval( double K, double S0)
{
      double val = 0;
      {
            if(S0>K) val = S0 - K;
            }
            return val;
            }
```

Output

The figure is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following result:

```
Call option price: 7.705

Press any key to continue …
```

Example of calculating a put option using Monte Carlo simulation. Part of the formulas were taken from the book Financial Modeling using C++, p458.

```
// Calculation of a put option using Monte Carlo simulation.

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double termval( double K, double S0);

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double sharePrice;
double optPrice;
double payoff = 0 ;
double randomNumber;


double S0 = 50;     // initial share price
double K = 58;      // strike price
double r = 0.05;    //  risk – free interest rate
double q = 0.02;    // dividend yield
double sig = 0.45;  // volatility
double T = 0.5;     // maturity

srand (time(0));

      randomNumber = ((rand()%10000)+1000)/100;

     for(i=0; i <maxSize; i++)
     { normSamps[i]=0;
     }

 for (int i = 0; i <= numberOfSimulation; i ++)
 {
    sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
      *sqrt(T));            // We simulate the share price by using random numbers.

    payoff += termval(K, sharePrice);        /*We use the for loop to calculate the sum
of payoffs for the given simulation number.*/
    }
```

```
   optPrice = (payoff * exp(-r*T))/ numberOfSimulation;  /*  Calculate the price of
the put option based on the payoff.*/

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Put option price:"<<optPrice<<endl;

   system ("PAUSE");
   return 0;
   }
double termval( double K, double S0)
{
    double val = 0;
    {
        if(S0<K) val = K - S0;
        }
        return val;
        }
```

Output

The figure is expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following result:

```
Put option price: 9.509

Press any key to continue …
```

Example of calculating a call option using Monte Carlo simulation and comparing the numerical value with the Black and Scholes model.
Part of the formulas were taken from the book Financial Modeling using C++, p458.

```
/* Calculation of a call option using Monte Carlo simulation and comparing it
 with the Balck and Scholes model.*/

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double Call(double S0, double K, double r, double q, double T,
double sig);

double termval( double K, double S0);

 // Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
    return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
      + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}
```

```
   // Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S0, double K, double r, double q, double T,
double sig){

  double d1, d2;

  d1 = (log(S0/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S0*exp(-q*T)*N(d1) - K*exp(-r*T)*N(d2) ;
}

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double sharePrice;
double optPrice;
double payoff = 0 ;
double randomNumber;


double S0 = 60;     // initial share price
double K = 50;      // strike price
double r = 0.05;    //  risk –free interest rate
double q = 0.02;    // dividend yield
double sig = 0.45;  // volatility
double T = 0.5;     // maturity

srand (time(0));

        randomNumber = ((rand()%10000)+1000)/100;

       for(i=0; i <maxSize; i++)
       { normSamps[i]=0;
       }

 for (int i = 0; i <= numberOfSimulation; i ++)
 {
    sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
      *sqrt(T));             // We simulate the share price by using random numbers.

    payoff += termval(K, sharePrice);        /* We use the for loop to calculate the sum
 of payoffs for the given simulation number.*/
    }
    optPrice = (payoff * exp(-r*T))/ numberOfSimulation;  /* Calculate the price of
the call option based on the payoff.*/

  //Calculation of the call price based on the Black and Scholes model.

double callPrice;

callPrice = Call (S0,K,r,q,T,sig);

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Monte Carlo call option price:"<<optPrice<<endl;
cout<<"Black and Scholes call price: " <<callPrice << endl;
    system ("PAUSE");
    return 0;
    }

double termval( double K, double S0)
{
      double val = 0;
      {
            if(S0>K) val = S0 - K;
            }
            return val;
            }
```

<u>Output</u>

The figures are expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Monte Carlo call option price: 7.705
Black and Scholes call price: 13.414

Press any key to continue …
```

<u>Question</u>

Could you please explain the deviation between the simulated price and the Black and Scholes price?

[Hint: It could be due to the random number definition or lack of algorithm. The standard error is quite high.]

Calculation of a Monte Carlo call option using antithetic variables and comparing it with the Balck and Scholes model. The purpose of using antithetic variables is to improve the estimation results by reducing the standard error of the simulation results. Part of the formulas were taken from the book Financial Modeling using C++, p461.

```
/* Calculation of a Monte Carlo call option using antithetic variables and comparing it
 with the Balck and Scholes model. The purpose of using antithetic variables
 is to improve the estimation results by reducing the standard error
 of simulation results.*/

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double Call(double S0, double K, double r, double q, double T,
double sig);

double termval( double K, double S0);

 // Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
   return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
     + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}

   // Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S0, double K, double r, double q, double T,
double sig){

  double d1, d2;

  d1 = (log(S0/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return S0*exp(-q*T)*N(d1) - K*exp(-r*T)*N(d2) ;
}

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double optPrice;
double payoff = 0 ;
double randomNumber;
double sharePrice;

double S0 = 100;     // initial share price
double K = 60;      // strike price
double r = 0.07;    // risk-free interest rate
double q = 0.02;    // dividend yield
double sig = 0.10;  // volatility
double T = 0.5;     // maturity

srand (time(0));

        randomNumber = ((rand()%30000)+3000)/300;

       for(i=0; i <maxSize; i++)
       { normSamps[i]=0;

       }
```

```
 /* We calculate twice the stock price and the option payoffs to generate a better
estimate. The first equation calculate the share price with a positive normal deviate
We want that the simulated call option price is close to the Black and Scholes
price. In the second equation of the share price estimation, we deduct the normal
deviates normSamps. We are creating perfectly negatively correlated simulation
results to eliminate the standard error. The antithetic variables reduce the standard
error.   */

for (int i = 0; i <= numberOfSimulation; i ++)
 {
   sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
    *sqrt(T));          // We simulate the share price by using random numbers.

   payoff += termval(K, sharePrice);        // We use the for loop to calculate the sum
// of payoffs for the given simulation number.

   sharePrice = S0 *exp((r-q-0.5*sig*sig)*T - normSamps[i-1]*randomNumber*sig
    *sqrt(T));

   payoff += termval(K, sharePrice);

   }
   optPrice = (payoff * exp(-r*T))/ numberOfSimulation/2;  //  Calculate the price of
//the call option based on the payoff.


 //Calculation of the call price based on Black and Scholes model.

double callPrice;

callPrice = Call (S0,K,r,q,T,sig);

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Call option price:"<<optPrice<<endl;
cout<<"Black and Scholes call price: " <<callPrice << endl;
   system ("PAUSE");
   return 0;
   }

double termval( double K, double S0)
{
    double val = 0;
    {
        if(S0>K) val = S0 - K;
        }
        return val;
        }
```

Output

The figures are expressed in Euro. After compiling and debugging , the DOS window or console will open and display the following results:

```
Call option price: 40.822
Black and Scholes call price: 41.069

Press any key to continue …
```

Example of calculating a put option using Monte Carlo simulation and comparing the numerical value with the Black and Scholes model. Part of the formulas were taken from the book Financial Modeling using C++, p458.

```
/* Calculation of a put option using Monte Carlo simulation and comparing it
 with the Balck and Scholes model.*/

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double Put(double S0, double K, double r, double q, double T,
  double sig);

double termval( double K, double S0);


// Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
    return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
      + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}
   // Mathematical formulas to calculate d1, d2 and the European put price.
```

```
double Put(double S0, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S0/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*N(-d2) – S0*exp(-q*T)*N(-d1) ;
}

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double sharePrice;
double optPrice;
double payoff = 0 ;
double randomNumber;


double S0 = 50;     // initial share price
double K = 58;      // strike price
double r = 0.05;    // risk-free interest rate
double q = 0.02;    // dividend yield
double sig = 0.45;  // volatility
double T = 0.5;     // maturity

srand (time(0));

         randomNumber = ((rand()%10000)+1000)/100;

        for(i=0; i <maxSize; i++)
        { normSamps[i]=0;
        }

 for (int i = 0; i <= numberOfSimulation; i ++)
 {
     sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
        *sqrt(T));             // We simulate the share price by using random numbers.

     payoff += termval(K, sharePrice);        /* We use the for loop to calculate the sum
of payoffs for the given simulation number.*/
     }
    optPrice = (payoff * exp(-r*T))/ numberOfSimulation;  /* Calculate the price of
the put option based on the payoff.*/

    //Calculation of the put price based on the Black and Scholes model.

double putPrice;

putPrice = Put (S0,K,r,q,T,sig);

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Monte Carlo put option price:"<<optPrice<<endl;
cout<<"Black and Sholes put price: " <<putPrice << endl;
    system ("PAUSE");
    return 0;
    }
double termval( double K, double S0)
{
      double val = 0;
      {
            if(S0<K) val = K - S0;
            }
            return val;
            }
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Monte Carlo put option price: 9.509
Black and Scholes put price: 10.812

Press any key to continue …
```

Question.

Could you please explain the deviation between the simulated price and the Black and Scholes price?

[Hint: It could be due to the random number definition or lack of algorithm.]

Calculation of a Monte Carlo put option using antithetic variables and comparing it with the Balck and Scholes model. The purpose of using antithetic variables is to improve the estimation results by reducing the standard error of the simulation results. Part of the formulas were taken from the book Financial Modeling using C++, p461.

```
/* Calculation of  a Monte Carlo put option using antithetic variables and comparing it
 with the Balck and Scholes model. The purpose of using antithetic variables
 is to improve the estimation results by reducing the standard error
 of simulation results.*/

#include <iostream>
#include <ctime>
#include<cmath>
using namespace std;

double Put(double S0, double K, double r, double q, double T,
double sig);

double termval( double K, double S0);

 // Calculation of the cumulative normal distribution.

double NP(double x);
double N(double x);

double NP(double x)
{
  return (1.0/sqrt(2.0 * 3.1415)* exp(-x*x*0.5));
}

double N(double x)
{
  double b1 = 0.319381530;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double k;

  k = 1/(1+0.2316419*x);

  if (x >= 0.0)
  {
    return (1 - NP(x)*((b1*k) + (b2* k*k) + (b3*k*k*k)
      + (b4*k*k*k*k) + (b5*k*k*k*k*k)));
  }
  else
  {
    return (1-N(-x));
  }
}

     // Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S0, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S0/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

  return K*exp(-r*T)*N(-d2) - S0*exp(-q*T)*N(-d1) ;
}

int main()
{

int i ;
int numberOfSimulation = 199999;
const int maxSize = 200000;
double normSamps[maxSize];
double optPrice;
double payoff = 0 ;
double randomNumber;
double sharePrice;

double S0 = 70;     // initial share price
double K = 100;      // strike price
double r = 0.07;    //  risk –free interest rate
double q = 0.02;    // dividend yield
double sig = 0.10;  // volatility
double T = 0.5;     // maturity

srand (time(0));

        randomNumber = ((rand()%30000)+3000)/300;

       for(i=0; i <maxSize; i++)
       { normSamps[i]=0;

       }
  /* We calculate twice the stock price and the option payoffs to generate a better
estimate. The first equation calculate the share price with a positive normal deviate
   We want that the simulated call option price is close to the Black and Scholes
   price. In the second equation of the share price estimation, we deduct the normal
   deviates normSamps. We are creating perfectly negatively correlated simulation
   results to eliminate the standard error. The antithetic variables reduce the standard
   error. */

  for (int i = 0; i <= numberOfSimulation; i ++)
  {
     sharePrice = S0 *exp((r-q-0.5*sig*sig)*T + normSamps[i-1]*randomNumber*sig
       *sqrt(T));          // We simulate the share price by using random numbers.

     payoff += termval(K, sharePrice);        // We use the for loop to calculate the sum
// of payoffs for the given simulation number.

     sharePrice = S0 *exp((r-q-0.5*sig*sig)*T - normSamps[i-1]*randomNumber*sig
       *sqrt(T));

     payoff += termval(K, sharePrice);

     }
     optPrice = (payoff * exp(-r*T))/ numberOfSimulation/2;  //  Calculate the price of
//the call option based on the payoff.


  //Calculation of call price based on Black and Scholes model.

double putPrice;

putPrice = Put (S0,K,r,q,T,sig);

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(3);
cout<<"Put option price:"<<optPrice<<endl;
cout<<"Black and Scholes put price: " <<putPrice << endl;
     system ("PAUSE");
     return 0;
     }

double termval( double K, double S0)
{
       double val = 0;
       {
             if(S0<K) val = K-S0;
             }
             return val;
             }
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Put option price: 27.430
Black and Scholes put price: 27.257

Press any key to continue …
```

Calculate the share prices of a call option using a Cox, Ross, Rubinstein,(CRR) binomial 3 steps tree using a vector. The multipliers of up and down move depend on volatility and length of steps.

```
/* Calculate the share prices of a call option
 using a Cox, Ross, Rubinstein,(CRR) binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>
 #include<vector>
using namespace std;


 int main()
 {
     double S = 60;              //share price
     double K = 50;              // strike price
     double r = 0.05;            // risk –free interest rate
     double q = 0.02;            //  dividend yield
     double T = 0.5;             // life to maturity
     double sig = 0.3;           // volatility

     // Identify the variables. u = multiplier of upmove.
     //d = multiplier of downmove. dt = steps expressed in years.

     double u;
     double d;
     double dt;

   int i ,j;
  const int n = 3; // Identify tree steps binomial tree.

     // Insert the mathematical formulas.

     dt = T/n;
     u = exp(sig*sqrt(dt));
     d = 1/u;

double sharetree[i][j];
int arraysize[10];
vector<double> dataarray;
for (i=0; i<=j; i++)
{
       for (j=1; j<=n; j++)
   {
  if (i==0)  sharetree[i][j] = S * u;
   else  sharetree[i][j] = S* d;

   if (i==0) sharetree[i][j] = S * pow(u,2);
```

```
    else sharetree[i][j] = S* pow(d,2);

  if (i==0) sharetree[i][j] = S*u*d;
  if (i==0) sharetree[i][j] = S*pow(u,2)*d ;
  if (i==0) sharetree[i][j] = S* pow(d,2)*u;

   if (i==0) sharetree[i][j] = S * pow(u,3);
    else sharetree[i][j] = S* pow(d,3);

    }
}

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(5);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;

cout<<"Enter array size:"<<endl;
cin>> arraysize[10];

sharetree[i][j] = S ;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price [0]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S * u;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price upmove [1]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S* d;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove[1]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S * pow(u,2);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price upmove [2]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S*u*d;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price up and downmove [2]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow(d,2);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove [2]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow (u,3);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price upmove [3]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S*pow(u,2)*d ;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price *upmove* downmove*upmove [3]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow(d,2)*u;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price *upmove * downmove*downmove [3]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S* pow(d,3);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove [3]:"<<sharetree[i][j]<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.16667
Multiplier of upmove: 1.13029
Multiplier of downmove: 0.88473
Enter array size:
10
Share price[0]: 60.00
Share price upmove [1]: 67.82
Share price downmove [1]: 53.08
Share price upmove [2]: 76.65
Share price up and downmove[2]:60.00
Share price downmove [2]: 46.96
Share price upmove [3]: 86.64
Share price upmove*downmove* upmove [3]: 67.82
Share price upmove*downmove*downmove[3]: 53.08
Share price downmove [3]: 41.55

Press any key to continue …
```

Calculate the call option prices using a Cox, Ross, Rubinstein,(CRR) binomial 3 steps tree. Please e-mail me to send you the binomial tree in Excel if this is complicated. I will also convert this example in VBA. Please check the VBA section in the document Financial Derivatives.

```
/* Calculate the call option prices using a Cox, Ross, Rubinstein,(CRR)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;


 int main()
 {
    double S = 60;                  //share price
    double K = 50;                  // strike price
    double r = 0.05;                // risk-free interest rate
    double q = 0.02;                // dividend yield. If the dividend is zero, then, it is
                                    // not included in the equation of risk neutral
                                    // probability of upmove.
    double T = 0.5;                 // life to maturity
    double sig = 0.3;               // volatility

    // Identify the variables.
    // u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

  const int n = 3; // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double callpayoffupmovestep3;    // (Shareprice * upmove^3)-K
double callpayoff3i;             // (Shareprice * upmove^2 * downmove)-K
double callpayoff3ii;               // (Shareprice * downmove^2*upmove)-K
double callpayoffdownmovestep3;  // (Shareprice * downmove^3) -K

// The option prices are calculated backwards.

callpayoffupmovestep3 = S*pow(u,3)-K;
callpayoff3i = S * pow(u,2) * d -K;
callpayoff3ii =S * pow(d,2)*u-K;
callpayoffdownmovestep3 = S*pow(d,3)-K;
callpayoffdownmovestep3 =0;

double callpriceupmovestep2;
double callpriceupdownmovestep2;
double callpricedownmovestep2;

callpriceupmovestep2 = (disfact*(riskprobofupmove*callpayoffupmovestep3)
+(1-riskprobofupmove)*callpayoff3i);

callpriceupdownmovestep2 =(disfact*(riskprobofupmove*callpayoff3i)
+(1-riskprobofupmove)*callpayoff3ii);

callpricedownmovestep2 =(disfact*(riskprobofupmove*callpayoff3ii)
+(1-riskprobofupmove)*callpayoffdownmovestep3);

double callpriceupmovestep1;
double callpricedownmovestep1;

callpriceupmovestep1 = (disfact*(riskprobofupmove*callpriceupmovestep2)
+(1-riskprobofupmove)*callpriceupdownmovestep2);

callpricedownmovestep1=(disfact*(riskprobofupmove*callpriceupdownmovestep2)
+(1-riskprobofupmove)*callpricedownmovestep2);


double callpricestep0;

callpricestep0= (disfact*(riskprobofupmove*callpriceupmovestep1)
+(1-riskprobofupmove)*callpricedownmovestep1);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;

cout<<"Call payoff upmove step [3]:"<<callpayoffupmovestep3<<endl;
cout<<"Call payoff [3i]:"<<callpayoff3i<<endl;
```

```
cout<<"Call payoff [3ii]:"<<callpayoff3ii<<endl;
cout<<"Call payoff downmove step [3]:"<<callpayoffdownmovestep3<<endl;
cout<<"Call price upmove step[2]:"<<callpriceupmovestep2<<endl;
cout<<"Call price updownmove step [2]:"<<callpriceupdownmovestep2<<endl;
cout<<"Call price downmove step [2]:"<<callpricedownmovestep2<<endl;
cout<<"Call price upmove step [1]:"<<callpriceupmovestep1<<endl;
cout<<"Call price downmove step [1]:"<<callpricedownmovestep1<<endl;
cout<<"Call price step[0]:"<<callpricestep0<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.1667
Multiplier of upmove: 1.1303
Multiplier of downmove: 0.8847
Disount factor: 0.9917
Risk neutral probability of upmove: 0.4898
Call payoff upmove step [3]: 36.6406
Call payoff [3i]: 17.8174
Call payoff [3ii]: 3.0837
Call payoff downmove step [3]: 0.0000
Call price upmove step [2]: 26.8887
Call price updownmove step [2]: 10.2283
Call price downmove step [2]: 1.4980
Call price upmove step [1]: 18.2798
Call price downmove step [1]: 5.7328
Call price step [0]: 11.8044
Press any key to continue …
```

Compare the call option price using a Cox,Ross, Rubinstein, (CRR), binomial 3 steps tree with the Black and Scholes model.

```
/*Compare the call option price using a Cox,Ross, Rubinstein, (CRR), binomial 3
steps tree with the Black and Scholes model.*/

/* Calculate the call option prices using a Cox, Ross, Rubinstein,(CRR)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;

 // Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}

// Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

    return S*exp(-q*T)*norm_cdf(d1) - K*exp(-r*T)*norm_cdf(d2) ;
}
```

```
 int main()
 {
    double S = 60;                        //share price
    double K = 50;                        // strike price
    double r = 0.05;                      // risk-free interest rate
    double q = 0.02;                      // dividend yield. If the dividend is zero, then, it is
                                          // not included in the equation of risk neutral
                                          // probability of upmove.
    double T = 0.5;                       // life to maturity
    double sig = 0.3;                     // volatility

    // Identify the variables.
    // u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

 const int n = 3; // Identify tree steps binomial tree.

 // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double callpayoffupmovestep3;    // (Shareprice * upmove^3)-K
double callpayoff3i;                   // (Shareprice * upmove^2 * downmove)-K
double callpayoff3ii;                      // (Shareprice * downmove^2*upmove)-K
double callpayoffdownmovestep3;  // (Shareprice * downmove^3) -K

// The option prices are calculated backwards.

callpayoffupmovestep3 = S*pow(u,3)-K;
callpayoff3i = S * pow(u,2) * d -K;
callpayoff3ii =S * pow(d,2)*u-K;
callpayoffdownmovestep3 = S*pow(d,3)-K;
callpayoffdownmovestep3 =0;

double callpriceupmovestep2;
double callpriceupdownmovestep2;
double callpricedownmovestep2;
```

```
callpriceupmovestep2 = (disfact*(riskprobofupmove*callpayoffupmovestep3)
+(1-riskprobofupmove)*callpayoff3i);

callpriceupdownmovestep2 =(disfact*(riskprobofupmove*callpayoff3i)
+(1-riskprobofupmove)*callpayoff3ii);

callpricedownmovestep2 =(disfact*(riskprobofupmove*callpayoff3ii)
+(1-riskprobofupmove)*callpayoffdownmovestep3);

double callpriceupmovestep1;
double callpricedownmovestep1;

callpriceupmovestep1 = (disfact*(riskprobofupmove*callpriceupmovestep2)
+(1-riskprobofupmove)*callpriceupdownmovestep2);

callpricedownmovestep1=(disfact*(riskprobofupmove*callpriceupdownmovestep2)
+(1-riskprobofupmove)*callpricedownmovestep2);


double callpricestep0;

callpricestep0= (disfact*(riskprobofupmove*callpriceupmovestep1)
+(1-riskprobofupmove)*callpricedownmovestep1);

//Calculation of call price based on the Black and Scholes model.

double callPrice;

callPrice = Call (S,K,r,q,T,sig);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;
cout<<"Call payoff upmove step [3]:"<<callpayoffupmovestep3<<endl;
cout<<"Call payoff [3i]:"<<callpayoff3i<<endl;
cout<<"Call payoff [3ii]:"<<callpayoff3ii<<endl;
cout<<"Call payoff downmove step [3]:"<<callpayoffdownmovestep3<<endl;
cout<<"Call price upmove step[2]:"<<callpriceupmovestep2<<endl;
cout<<"Call price updownmove step [2]:"<<callpriceupdownmovestep2<<endl;
cout<<"Call price downmove step [2]:"<<callpricedownmovestep2<<endl;
cout<<"Call price upmove step [1]:"<<callpriceupmovestep1<<endl;
cout<<"Call price downmove step [1]:"<<callpricedownmovestep1<<endl;
cout<<"Call price step[0]:"<<callpricestep0<<endl;
```

```
cout<<" Black and Scholes call price:"<<callPrice<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.1667
Multiplier of upmove: 1.1303
Multiplier of downmove: 0.8847
Disount factor: 0.9917
Risk neutral probability of upmove: 0.4898
Call payoff upmove step [3]: 36.6406
Call payoff [3i]: 17.8174
Call payoff [3ii]: 3.0837
Call payoff downmove step [3]: 0.0000
Call price upmove step [2]: 26.8887
Call price updownmove step [2]: 10.2283
Call price downmove step [2]: 1.4980
Call price upmove step [1]: 18.2798
Call price downmove step [1]: 5.7328
Call price step [0]: 11.8044
Black and Scholes call price: 11.7183
Press any key to continue …
```

Calculate the put option prices using a Cox, Ross, Rubinstein,(CRR) binomial 3 steps tree. The multipliers of up and down move depend on volatility and the length of steps.

```
/* Calculate the put option prices using a Cox, Ross, Rubinstein,(CRR)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;

 int main()
 {
     double S = 60;                  // share price
     double K = 70;                  // strike price
     double r = 0.02;               // risk-free interest rate
     double q = 0.03;                // dividend yield
     double T = 0.5;                 // life to maturity
     double sig = 0.2;              // volatility

     // Identify the variables. u = multiplier of upmove.
     //d = multiplier of downmove. dt = steps expressed in years.
     //riskprobofupmove = risk neutral probability of upmove.
     //disfact = discount factor.

     double u;
     double d;
     double dt;
     double riskprobofupmove;
     double disfact;

const int n = 3; // Identify tree steps binomial tree.

     // Insert the mathematical formulas.

     dt = T/n;
     u = exp(sig*sqrt(dt));
     d = 1/u;
     disfact = exp(-r*dt);
     riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double putpayoffupmovestep3;    // K - (Shareprice * upmove^3)
double putpayoff3i;             // K- (Shareprice * upmove^2 * downmove)
double putpayoff3ii;            // K-(Shareprice * downmove^2*upmove)
double putpayoffdownmovestep3;  // K-(Shareprice * downmove^3)
```

```
// The option prices are calculated backwards.

putpayoffupmovestep3 = K-S*pow(u,3);
putpayoffupmovestep3=0;
putpayoff3i = K- S * pow(u,2) * d;
putpayoff3ii =K- S * pow(d,2)*u;
putpayoffdownmovestep3 = K - S*pow(d,3);


double putpriceupmovestep2;
double putpriceupdownmovestep2;
double putpricedownmovestep2;

putpriceupmovestep2 = (disfact*(riskprobofupmove*putpayoffupmovestep3)
+(1-riskprobofupmove)*putpayoff3i);

putpriceupdownmovestep2 =(disfact*(riskprobofupmove*putpayoff3i)
+(1-riskprobofupmove)*putpayoff3ii);

putpricedownmovestep2 =(disfact*(riskprobofupmove*putpayoff3ii)
+(1-riskprobofupmove)*putpayoffdownmovestep3);

double putpriceupmovestep1;
double putpricedownmovestep1;

putpriceupmovestep1 = (disfact*(riskprobofupmove*putpriceupmovestep2)
+(1-riskprobofupmove)*putpriceupdownmovestep2);

putpricedownmovestep1=(disfact*(riskprobofupmove*putpriceupdownmovestep2)
+(1-riskprobofupmove)*putpricedownmovestep2);


double putpricestep0;

putpricestep0= (disfact*(riskprobofupmove*putpriceupmovestep1)
+(1-riskprobofupmove)*putpricedownmovestep1);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;

cout<<"Put payoff upmove step[3]:"<<putpayoffupmovestep3<<endl;
cout<<"Put payoff [3i]:"<<putpayoff3i<<endl;
cout<<"Put payoff [3ii]:"<<putpayoff3ii<<endl;
```

```
cout<<"Put payoff downmove step[3]:"<<putpayoffdownmovestep3<<endl;
cout<<"Put price upmove step[2]:"<<putpriceupmovestep2<<endl;
cout<<"Put price updownmove step[2]:"<<putpriceupdownmovestep2<<endl;
cout<<"Put price downmove step[2]:"<<putpricedownmovestep2<<endl;
cout<<"Put price upmove step[1]:"<<putpriceupmovestep1<<endl;
cout<<"Put price downmove step[1]:"<<putpricedownmovestep1<<endl;
cout<<"Put price step[0]:"<<putpricestep0<<endl;

system ("PAUSE");
return 0;
}
```

<u>Output</u>

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.1667
Multiplier of upmove: 1.0851
Multiplier of downmove: 0.9216
Disount factor: 0.9967
Risk neutral probability of upmove: 0.4694
Put payoff upmove step [3]: 0.0000
Put payoff [3i]: 4.8955
Put payoff [3ii]: 14.7043
Put payoff downmove step [3]: 23.0353
Put price upmove step [2]: 2.5975
Put price updownmove step [2]: 10.0923
Put price downmove step [2]: 19.1017
Put price upmove step [1]: 6.5701
Put price downmove step [1]: 14.8568
Put price step [0]: 10.9566
Press any key to continue …
```

Compare the put option price using a Cox,Ross, Rubinstein, (CRR), binomial 3 steps tree with the Black and Scholes model.

```
/*Compare the put option price using a Cox,Ross, Rubinstein, (CRR), binomial 3
steps tree with the Black and Scholes model.*/

/* Calculate the put option prices using a Cox, Ross, Rubinstein,(CRR)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;

  // Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}
// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

    return K*exp(-r*T)*norm_cdf(-d2) - S*exp(-q*T)*norm_cdf(-d1) ;
}








 int main()
 {
    double S = 60;                  // share price
    double K = 70;                  // strike price
    double r = 0.02;                // risk-free interest rate
    double q = 0.03;                // dividend yield
    double T = 0.5;                 // life to maturity
    double sig = 0.2;               // volatility

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

  const int n = 3;      // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double putpayoffupmovestep3;         // K - (Shareprice * upmove^3)
double putpayoff3i;                  // K- (Shareprice * upmove^2 * downmove)
double putpayoff3ii;                 // K-(Shareprice * downmove^2*upmove)
double putpayoffdownmovestep3;   // K-(Shareprice * downmove^3)

// The option prices are calculated backwards.

putpayoffupmovestep3 = K-S*pow(u,3);
putpayoffupmovestep3=0;
putpayoff3i = K- S * pow(u,2) * d;
putpayoff3ii =K- S * pow(d,2)*u;
putpayoffdownmovestep3 = K - S*pow(d,3);


double putpriceupmovestep2;
double putpriceupdownmovestep2;
double putpricedownmovestep2;

putpriceupmovestep2 = (disfact*(riskprobofupmove*putpayoffupmovestep3)
+(1-riskprobofupmove)*putpayoff3i);
```

```
putpriceupdownmovestep2 =(disfact*(riskprobofupmove*putpayoff3i)
+(1-riskprobofupmove)*putpayoff3ii);

putpricedownmovestep2 =(disfact*(riskprobofupmove*putpayoff3ii)
+(1-riskprobofupmove)*putpayoffdownmovestep3);

double putpriceupmovestep1;
double putpricedownmovestep1;

putpriceupmovestep1 = (disfact*(riskprobofupmove*putpriceupmovestep2)
+(1-riskprobofupmove)*putpriceupdownmovestep2);

putpricedownmovestep1=(disfact*(riskprobofupmove*putpriceupdownmovestep2)
+(1-riskprobofupmove)*putpricedownmovestep2);


double putpricestep0;

putpricestep0= (disfact*(riskprobofupmove*putpriceupmovestep1)
+(1-riskprobofupmove)*putpricedownmovestep1);

//Calculation of put price based on the Black and Scholes model.

double putPrice;

putPrice = Put (S,K,r,q,T,sig);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;
cout<<"Put payoff upmove step[3]:"<<putpayoffupmovestep3<<endl;
cout<<"Put payoff [3i]:"<<putpayoff3i<<endl;
cout<<"Put payoff [3ii]:"<<putpayoff3ii<<endl;
cout<<"Put payoff downmove step[3]:"<<putpayoffdownmovestep3<<endl;
cout<<"Put price upmove step[2]:"<<putpriceupmovestep2<<endl;
cout<<"Put price updownmove step[2]:"<<putpriceupdownmovestep2<<endl;
cout<<"Put price downmove step[2]:"<<putpricedownmovestep2<<endl;
cout<<"Put price upmove step[1]:"<<putpriceupmovestep1<<endl;
cout<<"Put price downmove step[1]:"<<putpricedownmovestep1<<endl;
cout<<"Put price step[0]:"<<putpricestep0<<endl;
cout<<" Black and Scholes put price:"<<putPrice<<endl;
system ("PAUSE");
return 0;
```

```
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.1667
Multiplier of upmove: 1.0851
Multiplier of downmove: 0.9216
Disount factor: 0.9967
Risk neutral probability of upmove: 0.4694
Put payoff upmove step [3]: 0.0000
Put payoff [3i]: 4.8955
Put payoff [3ii]: 14.7043
Put payoff downmove step [3]: 23.0353
Put price upmove step [2]: 2.5975
Put price updownmove step [2]: 10.0923
Put price downmove step [2]: 19.1017
Put price upmove step [1]: 6.5701
Put price downmove step [1]: 14.8568
Put price step [0]: 10.9566
Black and Scholes put price: 10.7864
Press any key to continue …
```

Calculate the share prices of a call option using a Jarrow – Rudd, (JR), binomial 3 steps tree using a vector. The key difference between the Cox, Ross, Rubinstein,(CRR) binomial tree and the Jarrow – Rudd, (JR), binomila tree is in defining the parmeters of the multiplier of up and down move. The multipliers depend on a drift in addition to volatility and length of steps.

```
/* Calculate the share prices of a call option using Jarrow - Rudd, (JR), binomial 3
steps tree. */

 #include <iostream>
 #include<cmath>
 #include<vector>
using namespace std;


 int main()
 {
    double S = 90;              //share price
    double K = 85;              // strike price
    double r = 0.04;            // risk -free interest rate
    double q = 0.03;            //  dividend yield. If the dividend is zero, then, it is
                                // not included in the equation of the multiplier of upmove
                                // and downmove.

    double T = 0.5;             // life to maturity
    double sig = 0.2;           // volatility

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.

    double u;
    double d;
    double dt;

  int i,j ;
  const int n = 3; // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = (exp((r-q -0.5*sig*sig)*dt + sig*sqrt(dt)));
    d = (exp((r-q -0.5*sig*sig)*dt - sig*sqrt(dt)));

double sharetree[i][j];
int arraysize[10];
vector<double> dataarray;
for (i=0; i<=j; i++)
{
      for (j=1; j<=n; j++)
```

```
   {
  if (i==0)  sharetree[i][j] = S * u;
   else  sharetree[i][j] = S* d;

   if (i==0) sharetree[i][j] = S * pow(u,2);
    else sharetree[i][j] = S* pow(d,2);

  if (i==0) sharetree[i][j] = S*u*d;
  if (i==0) sharetree[i][j] = S*pow(u,2)*d ;
  if (i==0) sharetree[i][j] = S* pow(d,2)*u;

   if (i==0) sharetree[i][j] = S * pow(u,3);
    else sharetree[i][j] = S* pow(d,3);

    }
}

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(4);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;

cout<<"Enter array size:"<<endl;
cin>> arraysize[10];

sharetree[i][j] = S ;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price [0]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S * u;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price upmove [1]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S* d;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove[1]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S * pow(u,2);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

```
cout<<" Share price upmove [2]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S*u*d;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price up and downmove [2]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow(d,2);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove [2]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow (u,3);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price upmove [3]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S*pow(u,2)*d ;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price *upmove* downmove*upmove [3]:"<<sharetree[i][j]<<endl;


sharetree[i][j] = S* pow(d,2)*u;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price *upmove * downmove*downmove [3]:"<<sharetree[i][j]<<endl;

sharetree[i][j] = S* pow(d,3);
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<" Share price downmove [3]:"<<sharetree[i][j]<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.1667
Multiplier of upmove: 1.0833
Multiplier of downmove: 0.9201
Enter array size:
10
Share price[0]: 90
Share price upmove [1]:97.49
Share price downmove [1]: 82.81
Share price upmove [2]: 105.61
Share price up and downmove[2]:89.70
Share price downmove [2]: 76.19
Share price upmove [3]: 114.41
Share price upmove*downmove* upmove [3]:97.17
Share price upmove*downmove*downmove[3]: 82.53
Share price downmove [3]: 70.10
Press any key to continue …
```

Calculate the call option prices using a Jarrow – Rudd, (JR), binomial 3 steps tree. The key difference between the Cox, Ross, Rubinstein,(CRR) binomial tree and the Jarrow – Rudd, (JR), binomila tree is in defining the parmeters of the multiplier of up and down move.

```
/* Calculate the call option prices using Jarrrow and Rudd(J,R)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>

using namespace std;


 int main()
 {
    double S = 90;                  // share price
    double K = 85;                  // strike price
    double r = 0.04;                // risk – free interest rate
    double q = 0.03;                // dividend yield. If the dividend is zero, then, it is
                                    // not included in the equation of the multiplier of
                                     // upmove and downmove.
    double T = 0.5;                 // life to maturity
    double sig = 0.2;               // volatility

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

  const int n = 3; // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = (exp((r-q -0.5*sig*sig)*dt + sig*sqrt(dt)));
    d = (exp((r-q -0.5*sig*sig)*dt - sig*sqrt(dt)));
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double callpayoffupmovestep3;        // (Shareprice * upmove^3)-K
double callpayoff3i;                 // (Shareprice * upmove^2 * downmove)-K
double callpayoff3ii;                // (Shareprice * downmove^2*upmove)-K
double callpayoffdownmovestep3;  // (Shareprice * downmove^3) -K


callpayoffupmovestep3 = S*pow(u,3)-K;
callpayoff3i = S * pow(u,2) * d -K;
callpayoff3ii =S * pow(d,2)*u-K;
callpayoff3ii =0;
callpayoffdownmovestep3 = S*pow(d,3)-K;
callpayoffdownmovestep3 =0;

double callpriceupmovestep2;
double callpriceupdownmovestep2;
double callpricedownmovestep2;

callpriceupmovestep2 = (disfact*(riskprobofupmove*callpayoffupmovestep3)
+(1-riskprobofupmove)*callpayoff3i);

callpriceupdownmovestep2 =(disfact*(riskprobofupmove*callpayoff3i)
+(1-riskprobofupmove)*callpayoff3ii);

callpricedownmovestep2 =(disfact*(riskprobofupmove*callpayoff3ii)
+(1-riskprobofupmove)*callpayoffdownmovestep3);

double callpriceupmovestep1;
double callpricedownmovestep1;

callpriceupmovestep1 = (disfact*(riskprobofupmove*callpriceupmovestep2)
+(1-riskprobofupmove)*callpriceupdownmovestep2);

callpricedownmovestep1=(disfact*(riskprobofupmove*callpriceupdownmovestep2)
+(1-riskprobofupmove)*callpricedownmovestep2);


double callpricestep0;

callpricestep0= (disfact*(riskprobofupmove*callpriceupmovestep1)
+(1-riskprobofupmove)*callpricedownmovestep1);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;
cout<<"Call payoff upmove step3:"<<callpayoffupmovestep3<<endl;
cout<<"Call payoff 3i:"<<callpayoff3i<<endl;
cout<<"Call payoff 3ii:"<<callpayoff3ii<<endl;
```

```
cout<<"Call payoff downmove step3:"<<callpayoffdownmovestep3<<endl;
cout<<"Call price upmove step2:"<<callpriceupmovestep2<<endl;
cout<<"Call price up down move step2:"<<callpriceupdownmovestep2<<endl;
cout<<"Call price down move step2:"<<callpricedownmovestep2<<endl;
cout<<"Call price upmove step1:"<<callpriceupmovestep1<<endl;
cout<<"Call price down movestep1:"<<callpricedownmovestep1<<endl;
cout<<"Call price step0:"<<callpricestep0<<endl;

system ("PAUSE");
return 0;
}
```

<u>Output</u>

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.17
Multiplier of upmove: 1.08
Multiplier of downmove: 0.92
Disount factor: 0.99
Risk neutral probability of upmove: 0.50
Call payoff upmove step [3]: 29.41
Call payoff [3i]: 12.17
Call payoff [3ii]: 0
Call payoff downmove step [3]: 0
Call price upmove step [2]: 20.69
Call price updownmove step [2]: 6.04
Call price downmove step [2]: 0
Call price upmove step [1]: 13.30
Call price downmove step [1]: 3.00
Call price step [0]: 8.11
Press any key to continue …
```

Compare the call option price using a Jarrow - Rudd, (JR), binomial 3 steps tree with the Black and Scholes model.

```
/*Compare the call option price using a Jarrow - Rudd, (JR), binomial 3 steps tree
with the Black and Scholes  model.*/

/* Calculate the call option prices using Jarrrow and Rudd(J,R)
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>

using namespace std;

 // Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}

// Mathematical formulas to calculate d1, d2 and the European call price.

double Call(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

    return S*exp(-q*T)*norm_cdf(d1) - K*exp(-r*T)*norm_cdf(d2) ;
}
```

```
 int main()
 {
    double S = 90;                    // share price
    double K = 85;                    // strike price
    double r = 0.04;                 // risk - free interest rate
    double q = 0.03;               // dividend yield. If the dividend is zero, then, it is
                                   // not included in the equation of the multiplier of
                                   // upmove and downmove.
    double T = 0.5;                // life to maturity
    double sig = 0.2;              // volatility

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

 const int n = 3; // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = (exp((r-q -0.5*sig*sig)*dt + sig*sqrt(dt)));
    d = (exp((r-q -0.5*sig*sig)*dt - sig*sqrt(dt)));
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double callpayoffupmovestep3;    // (Shareprice * upmove^3)-K
double callpayoff3i;                    // (Shareprice * upmove^2 * downmove)-K
double callpayoff3ii;                        // (Shareprice * downmove^2*upmove)-K
double callpayoffdownmovestep3;  // (Shareprice * downmove^3) -K


callpayoffupmovestep3 = S*pow(u,3)-K;
callpayoff3i = S * pow(u,2) * d -K;
callpayoff3ii =S * pow(d,2)*u-K;
callpayoff3ii =0;
callpayoffdownmovestep3 = S*pow(d,3)-K;
callpayoffdownmovestep3 =0;

double callpriceupmovestep2;
double callpriceupdownmovestep2;
double callpricedownmovestep2;
```

```
callpriceupmovestep2 = (disfact*(riskprobofupmove*callpayoffupmovestep3)
+(1-riskprobofupmove)*callpayoff3i);

callpriceupdownmovestep2 =(disfact*(riskprobofupmove*callpayoff3i)
+(1-riskprobofupmove)*callpayoff3ii);

callpricedownmovestep2 =(disfact*(riskprobofupmove*callpayoff3ii)
+(1-riskprobofupmove)*callpayoffdownmovestep3);

double callpriceupmovestep1;
double callpricedownmovestep1;

callpriceupmovestep1 = (disfact*(riskprobofupmove*callpriceupmovestep2)
+(1-riskprobofupmove)*callpriceupdownmovestep2);

callpricedownmovestep1=(disfact*(riskprobofupmove*callpriceupdownmovestep2)
+(1-riskprobofupmove)*callpricedownmovestep2);

double callpricestep0;

callpricestep0= (disfact*(riskprobofupmove*callpriceupmovestep1)
+(1-riskprobofupmove)*callpricedownmovestep1);

//Calculation of call price based on the Black and Scholes model.

double callPrice;

callPrice = Call (S,K,r,q,T,sig);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;

cout<<"Call payoff upmove step3:"<<callpayoffupmovestep3<<endl;
cout<<"Call payoff 3i:"<<callpayoff3i<<endl;
cout<<"Call payoff 3ii:"<<callpayoff3ii<<endl;
cout<<"Call payoff downmove step3:"<<callpayoffdownmovestep3<<endl;
cout<<"Call price upmove step2:"<<callpriceupmovestep2<<endl;
cout<<"Call price up down move step2:"<<callpriceupdownmovestep2<<endl;
cout<<"Call price down move step2:"<<callpricedownmovestep2<<endl;
cout<<"Call price upmove step1:"<<callpriceupmovestep1<<endl;
cout<<"Call price down movestep1:"<<callpricedownmovestep1<<endl;
cout<<"Call price step0:"<<callpricestep0<<endl;
cout<<" Black and Scholes call price:"<<callPrice<<endl;
```

```
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.17
Multiplier of upmove: 1.08
Multiplier of downmove: 0.92
Disount factor: 0.99
Risk neutral probability of upmove: 0.50
Call payoff upmove step [3]: 29.41
Call payoff [3i]: 12.17
Call payoff [3ii]: 0
Call payoff downmove step [3]: 0
Call price upmove step [2]: 20.69
Call price updownmove step [2]: 6.04
Call price downmove step [2]: 0
Call price upmove step [1]: 13.30
Call price downmove step [1]: 3.00
Call price step [0]: 8.11
Black and Scholes call price: 7.98
Press any key to continue …
```

Calculate the put option prices using a Jarrow – Rudd, (JR), binomial 3 steps tree.

```
/* Calculate the put option prices using Jarrow and Rudd, (J,R),
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>

using namespace std;


 int main()
 {
    double S = 60;
    double K = 70;
    double r = 0.02;
    double q = 0.03;
    double T = 0.5;
    double sig = 0.2;

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

  const int n = 3; // Identify tree steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = (exp((r-q -0.5*sig*sig)*dt + sig*sqrt(dt)));
    d = (exp((r-q -0.5*sig*sig)*dt - sig*sqrt(dt)));
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double putpayoffupmovestep3;        // K - (Shareprice * upmove^3)
double putpayoff3i;                 // K- (Shareprice * upmove^2 * downmove)
double putpayoff3ii;               // K-(Shareprice * downmove^2*upmove)
double putpayoffdownmovestep3;  // K-(Shareprice * downmove^3)
```

```
// The option prices are calculated backwards.

putpayoffupmovestep3 = K-S*pow(u,3);
putpayoffupmovestep3=0;
putpayoff3i = K- S * pow(u,2) * d;
putpayoff3ii =K- S * pow(d,2)*u;
putpayoffdownmovestep3 = K - S*pow(d,3);


double putpriceupmovestep2;
double putpriceupdownmovestep2;
double putpricedownmovestep2;

putpriceupmovestep2 = (disfact*(riskprobofupmove*putpayoffupmovestep3)
+(1-riskprobofupmove)*putpayoff3i);

putpriceupdownmovestep2 =(disfact*(riskprobofupmove*putpayoff3i)
+(1-riskprobofupmove)*putpayoff3ii);

putpricedownmovestep2 =(disfact*(riskprobofupmove*putpayoff3ii)
+(1-riskprobofupmove)*putpayoffdownmovestep3);

double putpriceupmovestep1;
double putpricedownmovestep1;

putpriceupmovestep1 = (disfact*(riskprobofupmove*putpriceupmovestep2)
+(1-riskprobofupmove)*putpriceupdownmovestep2);

putpricedownmovestep1=(disfact*(riskprobofupmove*putpriceupdownmovestep2)
+(1-riskprobofupmove)*putpricedownmovestep2);


double putpricestep0;

putpricestep0= (disfact*(riskprobofupmove*putpriceupmovestep1)
+(1-riskprobofupmove)*putpricedownmovestep1);

//Output functions.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;
cout<<"Put payoff upmove step[3]:"<<putpayoffupmovestep3<<endl;
cout<<"Put payoff [3i]:"<<putpayoff3i<<endl;
cout<<"Put payoff [3ii]:"<<putpayoff3ii<<endl;
cout<<"Put payoff downmove step[3]:"<<putpayoffdownmovestep3<<endl;
```

```
cout<<"Put price upmove step[2]:"<<putpriceupmovestep2<<endl;
cout<<"Put price updownmove step[2]:"<<putpriceupdownmovestep2<<endl;
cout<<"Put price downmove step[2]:"<<putpricedownmovestep2<<endl;
cout<<"Put price upmove step[1]:"<<putpriceupmovestep1<<endl;
cout<<"Put price downmove step[1]:"<<putpricedownmovestep1<<endl;
cout<<"Put price step[0]:"<<putpricestep0<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.17
Multiplier of upmove: 1.08
Multiplier of downmove: 0.92
Disount factor: 1.00
Risk neutral probability of upmove: 0.50
Put payoff upmove step [3]: 0.00
Put payoff [3i]: 5.86
Put payoff [3ii]: 15.53
Put payoff downmove step [3]: 23.73
Put price upmove step [2]: 2.93
Put price updownmove step [2]: 10.69
Put price downmove step [2]: 19.61
Put price upmove step [1]: 6.80
Put price downmove step [1]: 15.13
Put price step [0]: 10.95
Press any key to continue …
```

Compare the put option price using a Jarrow - Rudd, (JR), binomial 3 steps tree with the Black and Scholes model.

```
/*Compare the put option price using a Jarrow - Rudd, (JR), binomial 3 steps tree
with //the Black and Scholes model.*/

/* Calculate the put option prices using Jarrow and Rudd, (J,R),
binomial 3 steps tree. */

 #include <iostream>
 #include<cmath>

using namespace std;

// Calculation of the cumulative normal distribution.

double norm_cdf (const double& x)
{

double k = 1/(1+0.2316419*x);
double k_sum = k*(0.319381530 + k*(-0.356563782 + k*(1.781477937 +
 k*(-1.821255978 + k*(1.330274429)))));
  if (x >= 0.0)
  {
return (1.0 - (1.0/(pow(2*M_PI, 0.5))) * exp(-0.5*x*x)* k_sum);
} else {
return 1.0 - norm_cdf(-x);
}
}

// Mathematical formulas to calculate d1, d2 and the European put price.

double Put(double S, double K, double r, double q, double T,
  double sig){

  double d1, d2;

  d1 = (log(S/K) + (r-q +(sig*sig)*0.5 ) * T )
    / (sig * sqrt(T));
  d2 = d1 - sig*sqrt(T);

    return K*exp(-r*T)*norm_cdf(-d2) - S*exp(-q*T)*norm_cdf(-d1) ;
}


 int main()
 {
    double S = 60;
    double K = 70;
    double r = 0.02;
    double q = 0.03;
    double T = 0.5;
    double sig = 0.2;

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.
    //riskprobofupmove = risk neutral probability of upmove.
    //disfact = discount factor.

    double u;
    double d;
    double dt;
    double riskprobofupmove;
    double disfact;

  const int n = 3; // Identify tree steps binomial tree.

 // Insert the mathematical formulas.

    dt = T/n;
    u = (exp((r-q -0.5*sig*sig)*dt + sig*sqrt(dt)));
    d = (exp((r-q -0.5*sig*sig)*dt - sig*sqrt(dt)));
    disfact = exp(-r*dt);
    riskprobofupmove = (exp((r-q)*dt)-d)/(u-d);


double putpayoffupmovestep3;        // K - (Shareprice * upmove^3)
double putpayoff3i;                 // K- (Shareprice * upmove^2 * downmove)
double putpayoff3ii;               // K-(Shareprice * downmove^2*upmove)
double putpayoffdownmovestep3;  // K-(Shareprice * downmove^3)

// The option prices are calculated backwards.

putpayoffupmovestep3 = K-S*pow(u,3);
putpayoffupmovestep3=0;
putpayoff3i = K- S * pow(u,2) * d;
putpayoff3ii =K- S * pow(d,2)*u;
putpayoffdownmovestep3 = K - S*pow(d,3);


double putpriceupmovestep2;
double putpriceupdownmovestep2;
double putpricedownmovestep2;

putpriceupmovestep2 = (disfact*(riskprobofupmove*putpayoffupmovestep3)
+(1-riskprobofupmove)*putpayoff3i);

putpriceupdownmovestep2 =(disfact*(riskprobofupmove*putpayoff3i)
+(1-riskprobofupmove)*putpayoff3ii);
```

```
putpricedownmovestep2 =(disfact*(riskprobofupmove*putpayoff3ii)
+(1-riskprobofupmove)*putpayoffdownmovestep3);

double putpriceupmovestep1;
double putpricedownmovestep1;

putpriceupmovestep1 = (disfact*(riskprobofupmove*putpriceupmovestep2)
+(1-riskprobofupmove)*putpriceupdownmovestep2);

putpricedownmovestep1=(disfact*(riskprobofupmove*putpriceupdownmovestep2)
+(1-riskprobofupmove)*putpricedownmovestep2);


double putpricestep0;

putpricestep0= (disfact*(riskprobofupmove*putpriceupmovestep1)
+(1-riskprobofupmove)*putpricedownmovestep1);

//Calculation of put price based on the Black and Scholes model.

double putPrice;

putPrice = Put (S,K,r,q,T,sig);

//Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Discount factor:"<<disfact<<endl;
cout<< "Risk neutral probability of upmove:"<<riskprobofupmove<<endl;
cout<<"Put payoff upmove step[3]:"<<putpayoffupmovestep3<<endl;
cout<<"Put payoff [3i]:"<<putpayoff3i<<endl;
cout<<"Put payoff [3ii]:"<<putpayoff3ii<<endl;
cout<<"Put payoff downmove step[3]:"<<putpayoffdownmovestep3<<endl;
cout<<"Put price upmove step[2]:"<<putpriceupmovestep2<<endl;
cout<<"Put price updownmove step[2]:"<<putpriceupdownmovestep2<<endl;
cout<<"Put price downmove step[2]:"<<putpricedownmovestep2<<endl;
cout<<"Put price upmove step[1]:"<<putpriceupmovestep1<<endl;
cout<<"Put price downmove step[1]:"<<putpricedownmovestep1<<endl;
cout<<"Put price step[0]:"<<putpricestep0<<endl;
cout<<" Black and Scholes put price:"<<putPrice<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in USD. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.17
Multiplier of upmove: 1.08
Multiplier of downmove: 0.92
Disount factor: 1.00
Risk neutral probability of upmove: 0.50
Put payoff upmove step [3]: 0.00
Put payoff [3i]: 5.86
Put payoff [3ii]: 15.53
Put payoff downmove step [3]: 23.73
Put price upmove step [2]: 2.93
Put price updownmove step [2]: 10.69
Put price downmove step [2]: 19.61
Put price upmove step [1]: 6.80
Put price downmove step [1]: 15.13
Put price step [0]: 10.95
Black and Scholes put price: 10.79
Press any key to continue …
```

Calculate the interest rates of an option bond based on a two – period binomial tree. The risk – neutral probability of an up and down move in the interest rate tree is always 50% or 0.5.

```
/* Calculate the interest rates of an option bond using binomial 2 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;


 int main()
 {
    double BP = 100;                // bond price
    double strike = 100;            // strike price
    double AC = 7;                 // annual coupon
    double T = 0.5;                // maturity
    double sig = 0.3;              // volatility
    double r = 4;                  // interest rate expressed in percentage.

     // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.

    double u;
    double d;
    double dt;
    double p = 0.5;          // risk -neutral probability of upmove.

  const int n = 2; // Identify two steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;

// Identify the interest rates variables.

double interestrate0;
double interestrate1i;
double interestrate1ii;
double interestrate2i;
double interestrate2ii;
double interestrate2iii;

// Insert the mathematical formulas.

interestrate0 = r ;
interestrate1i = r * u;
interestrate1ii = r * d;
interestrate2i = r * pow(u,2);
interestrate2ii= r*u*d;
interestrate2iii = r* pow(d,2);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Interest rate [0]:"<<interestrate0<<endl;
cout<<" Interest rate upmove [1i]:"<<interestrate1i<<endl;
cout<<" Interest rate downmove[1ii]:"<<interestrate1ii<<endl;
cout<<" Interest rate upmove [2i]:"<<interestrate2i<<endl;
cout<<" Interest rate up and downmove [2ii]:"<<interestrate2ii<<endl;
cout<<" Interest rate downmove [2iii]:"<<interestrate2iii<<endl;

system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in percentages. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Interest rate [0]: 4.00
Interest rate upmove [1i]: 4.65
Interest rate downmove [1ii]: 3.44
Interest rate upmove [2i]: 5.40
Interest rate up and downmove[2ii]: 4.00
Interest rate downmove [2iii]: 2.96
Press any key to continue …
```

Calculate the bond prices of an option based on different interest rates using a binomial 2 steps tree.

```
/* Calculate the bond prices of an option
based on different interest rates using a binomial 2 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;


 int main()
 {
    double BP = 100;              // bond price
    double strike =100;           // strike price
    double AC = 7;               // annual coupon
    double T = 0.5;              // maturity
    double sig = 0.3;            // volatility
    double r = 4;                // interest rate

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.

    double u;
    double d;
    double dt;
    double p = 0.5;          // risk -neutral probability of up and down move.

   const int n = 2; // Identify two steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;


// Identify the bond price variables.

double bondPrice0;
double bondPrice1i;
double bondPrice1ii;
double bondPrice2i;
double bondPrice2ii;
double bondPrice2iii;

// Identify the discounted interest rates.
double disinterest0 = 1.04;
double disinterest1i = 1.0465;
double disinterest1ii = 1.0344;
double disinterest2i = 1.054;
double disinterest2ii = 1.04;
double disinterest2iii = 1.0296;

// Insert the mathematical formulas. It is a backward induction methodology.

bondPrice2i = (BP+AC)*(1/disinterest2i) ;
bondPrice2ii=(BP+AC)*(1/disinterest2ii) ;
bondPrice2iii = (BP+AC)*(1/disinterest2iii);
bondPrice1i = ((((bondPrice2i+AC)*p)+(bondPrice2ii+AC)*p)/disinterest1i);
bondPrice1ii = ((((bondPrice2ii+AC)*p)+(bondPrice2iii+AC)*p)/disinterest1ii);
bondPrice0 = ((((bondPrice1i+ AC)*p)+(bondPrice1ii+ AC)*p)/disinterest0);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;

cout<<" Bond price upmove [2i]:"<<bondPrice2i<<endl;
cout<<" Bond price up and downmove [2ii]:"<<bondPrice2ii<<endl;
cout<<" Bond price downmove [2iii]:"<<bondPrice2iii<<endl;
cout<<" Bond price upmove [1i]:"<<bondPrice1i<<endl;
cout<<" Bond price downmove[1ii]:"<<bondPrice1ii<<endl;
cout<<" Bond price [0]:"<<bondPrice0<<endl;
system ("PAUSE");
return 0;
}
```

<u>Output</u>

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Bond price upmove [2i]: 101.52
Bond price up and downmove[2ii]: 102.88
Bond price downmove [2iii]: 103.92
Bond price upmove [1i]: 104.35
Bond price downmove [1ii]: 106.73
Bond price [0]: 108.21
Press any key to continue …
```

Calculate the bond call option prices based on different interest rates using a binomial 2 steps tree.

```
/* Calculate the bond call option prices based on different interest rates using a
binomial 2 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;

 int main()
 {
    double BP = 100;          // bond price
    double strike = 100;      // strike price
    double AC = 7;            // annual coupon expressed in %
    double T = 0.5;           // maturity
    double sig = 0.3;         // volatility
    double r = 4;             // interest rate

    // Identify the variables. u = multiplier of upmove.
    //d = multiplier of downmove. dt = steps expressed in years.

    double u;
    double d;
    double dt;
    double p = 0.5;          // risk -neutral probability of up and down move.

  const int n = 2; // Identify two steps binomial tree.

    // Insert the mathematical formulas.

    dt = T/n;
    u = exp(sig*sqrt(dt));
    d = 1/u;


// Identify the bond price variables.

double bondPrice0;
double bondPrice1i;
double bondPrice1ii;
double bondPrice2i;
double bondPrice2ii;
double bondPrice2iii;

// Identify the discounted interest rates.
double disinterest0 = 1.04;
double disinterest1i = 1.0465;
double disinterest1ii = 1.0344;
double disinterest2i = 1.054;
double disinterest2ii = 1.04;
double disinterest2iii = 1.0296;

// Insert the mathematical formulas. It is a backward induction methodology.

bondPrice2i = (BP+AC)*(1/disinterest2i) ;
bondPrice2ii=(BP+AC)*(1/disinterest2ii) ;
bondPrice2iii = (BP+AC)*(1/disinterest2iii);
bondPrice1i = ((((bondPrice2i+AC)*p)+(bondPrice2ii+AC)*p)/disinterest1i);
bondPrice1ii = ((((bondPrice2ii+AC)*p)+(bondPrice2iii+AC)*p)/disinterest1ii);
bondPrice0 = ((((bondPrice1i+ AC)*p)+(bondPrice1ii+ AC)*p)/disinterest0);

double callPriceUpmovestep2i;
double callPriceUpdownmovestep2ii;
double callPriceDownmovestep2iii;
double callPriceUpmovestep1i;
double callPriceDownmovestep1ii;
double callPricestep0;

callPriceUpmovestep2i = bondPrice2i - strike;
callPriceUpdownmovestep2ii = bondPrice2ii - strike;
callPriceDownmovestep2iii = bondPrice2iii - strike;
callPriceUpmovestep1i=
(((callPriceUpmovestep2i*p)+(callPriceUpdownmovestep2ii*p))/disinterest1i);

callPriceDownmovestep1ii=
(((callPriceUpdownmovestep2ii*p)+(callPriceDownmovestep2iii*p))/disinterest1ii);

callPricestep0=
(((callPriceUpmovestep1i*p)+(callPriceDownmovestep1ii*p))/disinterest0);

//Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;

cout<<" Bond price upmove [2i]:"<<bondPrice2i<<endl;
cout<<" Bond price up and downmove [2ii]:"<<bondPrice2ii<<endl;
cout<<" Bond price downmove [2iii]:"<<bondPrice2iii<<endl;
cout<<" Bond price upmove [1i]:"<<bondPrice1i<<endl;
cout<<" Bond price downmove[1ii]:"<<bondPrice1ii<<endl;
cout<<" Bond price [0]:"<<bondPrice0<<endl;

cout<<"Call price upmove step [2i]:"<<callPriceUpmovestep2i<<endl;
cout<<"Call price updownmove step [2ii]:"<<callPriceUpdownmovestep2ii<<endl;
cout<<"Call price downmove step [2iii]:"<<callPriceDownmovestep2iii<<endl;
```

```
cout<<"Call price upmove step [1i]:"<<callPriceUpmovestep1i<<endl;
cout<<"Call price downmove step [1ii]:"<<callPriceDownmovestep1ii<<endl;
cout<<"Call price step [0]:"<<callPricestep0<<endl;


system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Bond price upmove [2i]: 101.52
Bond price up and downmove[2ii]: 102.88
Bond price downmove [2iii]: 103.92
Bond price upmove [1i]: 104.35
Bond price downmove [1ii]: 106.73
Bond price [0]: 108.21
Call price upmove step [2i]: 1.52
Call price updownmove step [2ii]: 2.88
Call price downmove step [2iii]: 3.92
Call price upmove step [1i]: 2.10
Call price downmove step [1ii]: 3.29
Call price step [0]: 2.59
Press any key to continue …
```

Calculate the bond put option prices based on different interest rates using a binomial 2 steps tree.

```
/* Calculate the bond put option prices based on different interest rates using a
binomial 2 steps tree. */

 #include <iostream>
 #include<cmath>
 using namespace std;

 int main()
 {
     double BP = 100;           // bond price
     double strike = 102;       // strike price
     double AC = 7;             // annual coupon expressed in %
     double T = 0.5;            // maturity
     double sig = 0.3;          // volatility
     double r = 4;              // interest rate

     // Identify the variables. u = multiplier of upmove.
     //d = multiplier of downmove. dt = steps expressed in years.

     double u;
     double d;
     double dt;
     double p = 0.5;           // risk -neutral probability of up and down move.

   const int n = 2; // Identify two steps binomial tree.

     // Insert the mathematical formulas.

     dt = T/n;
     u = exp(sig*sqrt(dt));
     d = 1/u;


// Identify the bond price variables.

double bondPrice0;
double bondPrice1i;
double bondPrice1ii;
double bondPrice2i;
double bondPrice2ii;
double bondPrice2iii;

// Identify the discounted interest rates.
double disinterest0 = 1.04;
double disinterest1i = 1.0465;
double disinterest1ii = 1.0344;
double disinterest2i = 1.054;
double disinterest2ii = 1.04;
double disinterest2iii = 1.0296;

// Insert the mathematical formulas. It is a backward induction methodology.

bondPrice2i = (BP+AC)*(1/disinterest2i) ;
bondPrice2ii=(BP+AC)*(1/disinterest2ii) ;
bondPrice2iii = (BP+AC)*(1/disinterest2iii);
bondPrice1i = ((((bondPrice2i+AC)*p)+(bondPrice2ii+AC)*p)/disinterest1i);
bondPrice1ii = ((((bondPrice2ii+AC)*p)+(bondPrice2iii+AC)*p)/disinterest1ii);
bondPrice0 = ((((bondPrice1i+ AC)*p)+(bondPrice1ii+ AC)*p)/disinterest0);

double putPriceUpmovestep2i;
double putPriceUpdownmovestep2ii;
double putPriceDownmovestep2iii;
double putPriceUpmovestep1i;
double putPriceDownmovestep1ii;
double putPricestep0;

putPriceUpmovestep2i = strike - bondPrice2i ;
putPriceUpdownmovestep2ii = strike - bondPrice2ii;
putPriceUpdownmovestep2ii = 0;   // the put option has no intrinsic value. It is

//negative so, it has zero value.

putPriceDownmovestep2iii = strike - bondPrice2iii;
putPriceDownmovestep2iii =0;       // the put option has no intrinsic value. It is

//negative so, it has zero value.

putPriceUpmovestep1i=
(((putPriceUpmovestep2i*p)+(putPriceUpdownmovestep2ii*p))/disinterest1i);

putPriceDownmovestep1ii=
(((putPriceUpdownmovestep2ii*p)+(putPriceDownmovestep2iii*p))/disinterest1ii);

putPricestep0=
(((putPriceUpmovestep1i*p)+(putPriceDownmovestep1ii*p))/disinterest0);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<"Put price upmove step [2i]:"<<putPriceUpmovestep2i<<endl;
cout<<"Put price updownmove step [2ii]:"<<putPriceUpdownmovestep2ii<<endl;
cout<<"Put price downmove step [2iii]:"<<putPriceDownmovestep2iii<<endl;
```

```
cout<<"Put price upmove step [1i]:"<<putPriceUpmovestep1i<<endl;
cout<<"Put price downmove step [1ii]:"<<putPriceDownmovestep1ii<<endl;
cout<<"Put price step [0]:"<<putPricestep0<<endl;


system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Put price upmove step [2i]: 0.48
Put price updownmove step [2ii]: 0.00
Put price downmove step [2iii]: 0.00
Put price upmove step [1i]: 0.23
Put price downmove step [1ii]: 0.00
Put price step [0]: 0.11
Press any key to continue …
```

Calculate the caplet option payments based on different interest rates using a binomial 2 steps tree.

/* Calculate the caplet values based on an interest rate cap using a binomial 2 steps tree. It is a similar example of a call option on interest rates. The payment is based on the difference between the current interest rate and the cap rate multiplied by the principal and divided by the discounted current interest rate. The buyer receives a payment when the current interest rate exceeds the cap strike price. */

```
 #include <iostream>
 #include<cmath>
 using namespace std;


 int main()
 {

     double strike = 0.02;                  //  cap strike price
     double principal = 30000000;    //  principal
     double T = 0.5;                        // maturity
     double sig = 0.3;                      // volatility
     double r = 0.04;                       // interest rate

     // Identify the variables. u = multiplier of upmove.
     //d = multiplier of downmove. dt = steps expressed in years.

     double u;
     double d;
     double dt;
     double p = 0.5;           // risk -neutral probability of up and down move.

   const int n = 2; // Identify two steps binomial tree.

     // Insert the mathematical formulas.

     dt = T/n;
     u = exp(sig*sqrt(dt));
     d = 1/u;


// Identify the caplet price variables.

double capletPrice0;
double capletPrice1i;
double capletPrice1ii;
double capletPrice2i;
double capletPrice2ii;
double capletPrice2iii;
// Identify the interest rates in each node.
double interest0 = 0.04;
double interest1i = 0.0465;
double interest1ii = 0.0344;
double interest2i = 0.054;
double interest2ii = 0.04;
double interest2iii = 0.0296;

// Identify the discounted interest rates.
double disinterest0 = 1.04;
double disinterest1i = 1.0465;
double disinterest1ii = 1.0344;
double disinterest2i = 1.054;
double disinterest2ii = 1.04;
double disinterest2iii = 1.0296;

// Insert the mathematical formulas. It is a backward induction methodology.

capletPrice2i = (principal*(interest2i-strike)/disinterest2i) ;
capletPrice2ii = (principal*(interest2ii-strike)/disinterest2ii) ;
capletPrice2iii = (principal*(interest2iii-strike)/disinterest2iii) ;
capletPrice1i = (((capletPrice2i*p)+(capletPrice2ii*p))/disinterest1i);
capletPrice1ii = (((capletPrice2ii*p)+(capletPrice2iii*p))/disinterest1ii);
capletPrice0 = (((capletPrice1i*p)+(capletPrice1ii*p))/disinterest0);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Caplet price upmove [2i]:"<<capletPrice2i<<endl;
cout<<" Caplet price up and downmove [2ii]:"<<capletPrice2ii<<endl;
cout<<" Caplet price downmove [2iii]:"<<capletPrice2iii<<endl;
cout<<" Caplet price upmove [1i]:"<<capletPrice1i<<endl;
cout<<" Caplet price downmove[1ii]:"<<capletPrice1ii<<endl;
cout<<" Caplet price [0]:"<<capletPrice0<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Caplet price upmove [2i]: 967741.94
Caplet price up and downmove[2ii]: 576923.08
Caplet price downmove [2iii]: 279720.28
Caplet price upmove [1i]: 738014.82
Caplet price downmove [1ii]: 414077.42
Caplet price [0]: 553890.50
Press any key to continue …
```

Calculate the floorlet option payments based on different interest rates using a binomial 2 steps tree.

/* Calculate the floorlet values based on an interest rate floor using a binomial 2 steps tree. It is a similar example of a put option on interest rates. It protects the holder from declining interest rates. The payment is based on the difference between the floor rate and the current interest rate multiplied by the principal and divided by the discounted current interest rate. */

```
 #include <iostream>
 #include<cmath>
 using namespace std;


 int main()
 {

     double strike = 0.056;                  // floor strike price
     double principal = 30000000;    //  principal
     double T = 0.5;                         // maturity
     double sig = 0.3;                       // volatility
     double r = 0.04;                         // interest rate

     // Identify the variables. u = multiplier of upmove.
     //d = multiplier of downmove. dt = steps expressed in years.

     double u;
     double d;
     double dt;
     double p = 0.5;           // risk -neutral probability of up and down move.

   const int n = 2; // Identify two steps binomial tree.

     // Insert the mathematical formulas.

     dt = T/n;
     u = exp(sig*sqrt(dt));
     d = 1/u;


// Identify the floorlet price variables.

double floorletPrice0;
double floorletPrice1i;
double floorletPrice1ii;
double floorletPrice2i;
double floorletPrice2ii;
double floorletPrice2iii;
// Identify the interest rates in each node.
double interest0 = 0.04;
double interest1i = 0.0465;
double interest1ii = 0.0344;
double interest2i = 0.054;
double interest2ii = 0.04;
double interest2iii = 0.0296;

// Identify the discounted interest rates.
double disinterest0 = 1.04;
double disinterest1i = 1.0465;
double disinterest1ii = 1.0344;
double disinterest2i = 1.054;
double disinterest2ii = 1.04;
double disinterest2iii = 1.0296;

// Insert the mathematical formulas. It is a backward induction methodology.

floorletPrice2i = (principal*(strike - interest2i)/disinterest2i) ;
floorletPrice2ii = (principal*(strike - interest2ii)/disinterest2ii) ;
floorletPrice2iii = (principal*(strike - interest2iii)/disinterest2iii) ;
floorletPrice1i = (((floorletPrice2i*p)+(floorletPrice2ii*p))/disinterest1i);
floorletPrice1ii = (((floorletPrice2ii*p)+(floorletPrice2iii*p))/disinterest1ii);
floorletPrice0 = (((floorletPrice1i*p)+(floorletPrice1ii*p))/disinterest0);

// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Steps expressed in years:"<< dt<<endl;
cout<<"Multiplier of upmove:"<< u<<endl;
cout<<"Multiplier of downmove:"<< d<<endl;
cout<<" Floorlet price upmove [2i]:"<<floorletPrice2i<<endl;
cout<<" Floorlet price up and downmove [2ii]:"<<floorletPrice2ii<<endl;
cout<<" Floorlet price downmove [2iii]:"<<floorletPrice2iii<<endl;
cout<<" Floorlet price upmove [1i]:"<<floorletPrice1i<<endl;
cout<<" Floorlet price downmove[1ii]:"<<floorletPrice1ii<<endl;
cout<<" Floorlet price [0]:"<<floorletPrice0<<endl;
system ("PAUSE");
return 0;
}
```

Output

The figures are expressed in pounds. After compiling and debugging , the DOS window or console will open and display the following results:

```
Steps expressed in years: 0.25
Multiplier of upmove: 1.16
Multiplier of downmove: 0.86
Floorlet price upmove [2i]: 56926.00
Floorlet price up and downmove[2ii]: 461538.46
Floorlet price downmove [2iii]: 769230.77
Floorlet price upmove [1i]: 247713.55
Floorlet price downmove [1ii]: 594919.39
Floorlet price [0]:  405111.99
Press any key to continue …
```

Solution of the above examples.

Active return, active risk and information ratio.

Active return is the difference in returns between a portfolio and the index or benchmark that is measured.

Active return = $r_p - r_b$

Where: r_p is the portfolio return.
r_b is the benchmark or index return.

If the portfolio return is 0.80 and the benchmark return of the index is 0.70 then, the active return is………

Please complete the calculation.

Application of active return in C++

```
// Active return.

#include <iostream>
using namespace std;

int main()
{
   double portfolioReturn = 0.8;
   double benchmarkReturn = 0.70;
   double activeReturn;


// Insert the mathematical formula.

 activeReturn = portfolioReturn - benchmarkReturn;

// Output function.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<< "Active return :"<<activeReturn<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Active return : 0.10
Press any key to continue …
```

Active risk or tracking error is the standard deviation of the difference of returns between a portfolio and the benchmark or index.

$$Active\,\text{risk} = \sqrt{\frac{\sum(\text{r}_\text{p} - r_b)^2}{n-1}}$$

$Where:$ r_p is portfolio return.

r_b is portfolio benchmark return. It could be for example an index.

n is the number of assets or observatio ns.

If the portfolio return is 0.80, the number of observations is 10 and the benchmark return of the index is 0.40 then, the active risk is………

Please complete the calculation.

Application of active risk in C++

```
// Active risk.

#include <iostream>
#include <cmath>
using namespace std;


int main()
{
  double portfolioReturn = 0.80;
  double benchmarkReturn = 0.40;
  int numberOfObservations = 10;   // we use the formula n-1 = 10-1 =9.
  double activeRisk;


// Insert the mathematical formula.

 activeRisk = sqrt(pow(portfolioReturn - benchmarkReturn,2)/9);

// Output function.
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<< "Active risk :"<<activeRisk<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Active risk : 0.13
Press any key to continue …
```

Information ratio shows the consistency of the fund manager towards the active return. The mathematical formula is as follows:

$$IR = \frac{\bar{r}_p - \bar{r}_b}{\sqrt{\frac{\sum(\mathrm{r_p} - r_b)^2}{n-1}}}$$

$Where : \bar{r}_p$ is the average of portfolio return.

$\bar{\mathrm{r}}_\mathrm{b}$ is the average benchmark or index return.

$\sqrt{\frac{\sum\left(r_p - r_b\right)^2}{n-1}}$ is the active risk

It could be calculated very easily in Excel software. I will illustrate a simple table with the relevant calculations.

Day	r_p	r_b	r_p - r_b
1	0.03	0.02	0.01
2	0.02	0.014	0.006
3	-0.04	0.034	-0.074
4	0.05	0.067	-0.017
5	0.08	0.012	0.068
6	-0.01	-0.056	0.046
7	0.07	0.031	0.039
8	0.034	0.023	0.011
9	-0.021	0.015	-0.036
10	0.045	0.001	0.044
Average	**0.0258**	**0.0161**	
Standard deviation			**0.043**

Source: author's calculation

By substituting the values that we have found in terms of r_p =0.0258, r_b = 0.0161 and standard deviation = 0.043 in the following equation we have:

$$R = \frac{\bar{r}_p - \bar{r}_b}{\sqrt{\frac{\sum(\mathrm{r_p} - r_b)^2}{n-1}}}$$

$$R = \frac{0.0258 - 0.0161}{0.043} = \frac{0.0097}{0.043} = 0.23\,(\text{to 2d.p.})$$

But what is the interpretation of the information ratio of 0.23. It means that the fund manager gained around 23 basis points of active return per unit of active risk. The higher is this number, the better the manager is performing in relation to active risk.

Application of information ratio in C++

```
// Calculation of the information ratio.

#include <iostream>
#include<cmath>
using namespace std;
int main()
{

double informationRatio;

double portfolioReturn[10];
portfolioReturn[1] = 0.03;
portfolioReturn[2] = 0.02;
portfolioReturn[3] = -0.04;
portfolioReturn[4] = 0.05;
portfolioReturn[5] = 0.08;
portfolioReturn[6] = -0.01;
portfolioReturn[7] = 0.07;
portfolioReturn[8] = 0.034;
portfolioReturn[9] = -0.021;
portfolioReturn[10]= 0.045;

double average1;


// Insert the mathematical formula.

average1 = (portfolioReturn[1] + portfolioReturn[2] + portfolioReturn[3] +
portfolioReturn[4] + portfolioReturn[5] + portfolioReturn[6]
+ portfolioReturn[7] + portfolioReturn[8] + portfolioReturn[9]+
portfolioReturn[10])/10;


double benchmarkReturn[10];
benchmarkReturn[1] = 0.02;
benchmarkReturn[2] = 0.014;
benchmarkReturn[3] = 0.034;
benchmarkReturn[4] = 0.067;
benchmarkReturn[5] = 0.012;
benchmarkReturn[6] = -0.056;
benchmarkReturn[7] = 0.031;
benchmarkReturn[8] = 0.023;
benchmarkReturn[9] = 0.015;
benchmarkReturn[10]= 0.001;

double average2;

// Insert the mathematical formula.
```

```
average2 = (benchmarkReturn[1] + benchmarkReturn[2] + benchmarkReturn[3] +
benchmarkReturn[4] + benchmarkReturn[5] + benchmarkReturn[6]
+ benchmarkReturn[7] + benchmarkReturn[8] + benchmarkReturn[9]+
benchmarkReturn[10])/10;

/* Insert the mathematical formulas to calculate the differences between the portfolio
and benchmark return and then square it.*/

double diff1, diff2, diff3, diff4, diff5, diff6, diff7, diff8,
diff9,diff10;
double sumDiff;
double average3;

diff1 = portfolioReturn[1] - benchmarkReturn[1];
diff2 = portfolioReturn[2] - benchmarkReturn[2];
diff3 = portfolioReturn[3] - benchmarkReturn[3];
diff4 = portfolioReturn[4] - benchmarkReturn[4];
diff5 = portfolioReturn[5] - benchmarkReturn[5];
diff6 = portfolioReturn[6] - benchmarkReturn[6];
diff7 = portfolioReturn[7] - benchmarkReturn[7];
diff8 = portfolioReturn[8] - benchmarkReturn[8];
diff9 = portfolioReturn[9] - benchmarkReturn[9];
diff10 = portfolioReturn[10] - benchmarkReturn[10];

// Find the sum from the differences.


sumDiff = diff1 + diff2 + diff3 + diff4 + diff5 + diff6 + diff7 + diff8 +diff9 +diff10;

// Find the average from the differences.

average3 = sumDiff /10;

// The square functions.

double diff1Sq, diff2Sq, diff3Sq, diff4Sq, diff5Sq, diff6Sq, diff7Sq, diff8Sq,
diff9Sq,diff10Sq;

diff1Sq = pow(portfolioReturn[1] - benchmarkReturn[1]-average3,2);
diff2Sq = pow(portfolioReturn[2] - benchmarkReturn[2]-average3,2);
diff3Sq = pow(portfolioReturn[3] - benchmarkReturn[3]-average3,2);
diff4Sq = pow(portfolioReturn[4] - benchmarkReturn[4]-average3,2);
diff5Sq = pow(portfolioReturn[5] - benchmarkReturn[5]-average3,2);
diff6Sq = pow(portfolioReturn[6] - benchmarkReturn[6]-average3,2);
diff7Sq = pow(portfolioReturn[7] - benchmarkReturn[7]-average3,2);
diff8Sq = pow(portfolioReturn[8] - benchmarkReturn[8]-average3,2);
diff9Sq = pow(portfolioReturn[9] - benchmarkReturn[9]-average3,2);
diff10Sq = pow(portfolioReturn[10] - benchmarkReturn[10]-average3,2);
```

```
// The sum function of the differences squared.

double sumDiffSquare;

sumDiffSquare = diff1Sq + diff2Sq + diff3Sq + diff4Sq + diff5Sq + diff6Sq + diff7Sq
+ diff8Sq +diff9Sq +diff10Sq;

// Insert the formula for the sample standard deviation.

double Stdev;

Stdev = sqrt(sumDiffSquare/9);

//Insert the formula for the information ratio.

informationRatio = (average1 -average2)/Stdev;

// Output function.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<"Information ratio:"<<informationRatio<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Information ratio : 0.23
Press any key to continue …
```

Please convert the following exercise in C++ language programming.

Calculate the balance at the end of the 5th day from changes in the futures prices from 90 to 89, 91, 95, 97, 99. The trader has bought 50 futures contract for settlement in May. The initial margin is $8.

Day	Beginning balance	Futures price	Gain or loss	Ending balance
0	400	90	0	400
1	400	89	(50)	
2	350	91	100	
3	450	95	200	
4	650	97	100	
5	750	99	100	

Source: author's illustration.

Beginning balance = initial margin x number of contracts.
Beginning balance = 8 x 50 = $400

Please complete the values of the column ending balance.

Solution.

Day	Beginning balance	Futures price	Gain or loss	Ending balance
0	400	90	0	400
1	400	89	(50)	350
2	350	91	100	450
3	450	95	200	650
4	650	97	100	750
5	750	99	100	850

The ending balance at the 5th day in the margin account will be 850 USD.

Illustration of the volatility smile.

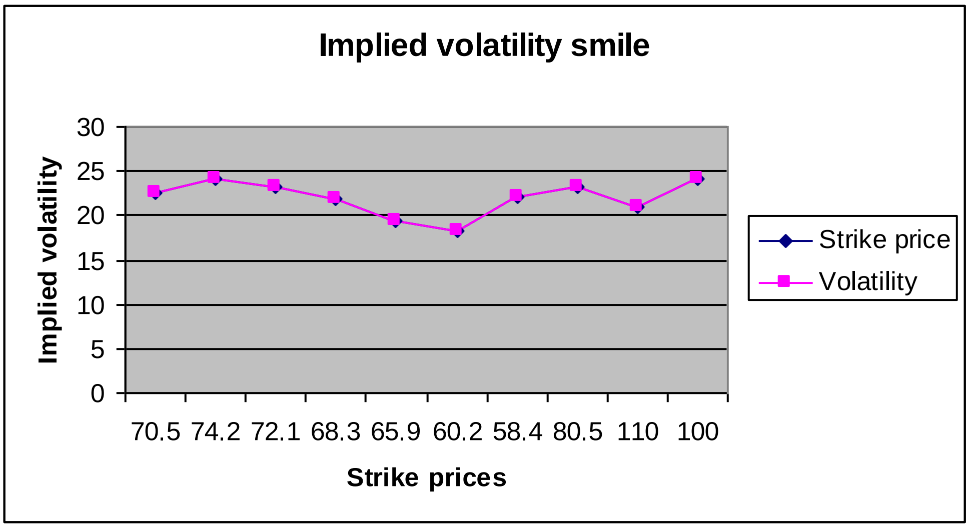
```
// Illustration of the volatility smile.

#include <iostream>
using namespace std;
int main()
{

// The numerical values are expressed in USD.

double strikePrice[10];
strikePrice[1] = 70.54;
strikePrice[2] = 74.21;
strikePrice[3] = 72.12;
strikePrice[4] = 68.34;
strikePrice[5] = 65.89;
strikePrice[6] = 60.23;
strikePrice[7] = 58.36;
strikePrice[8] = 80.45;
strikePrice[9] = 110.12;
strikePrice[10]= 100.34;

// The numerical values are expressed in percentages.

double impliedVolatility[10];
impliedVolatility[1] = 22.56;
impliedVolatility[2] = 24.16;
impliedVolatility[3] = 23.22;
impliedVolatility[4] = 21.86;
impliedVolatility[5] = 19.36;
impliedVolatility[6] = 18.16;
impliedVolatility[7] = 22.11;
impliedVolatility[8] = 23.22;
impliedVolatility[9] = 21.02;
impliedVolatility[10] = 24.21;


// Output functions.

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
std::cout<<"Strike price 1: " <<strikePrice[1]<<std::endl;
std::cout<<"Strike price 2: " <<strikePrice[2]<<std::endl;
std::cout<<"Strike price 3: " <<strikePrice[3]<<std::endl;
std::cout<<"Strike price 4: " <<strikePrice[4]<<std::endl;
std::cout<<"Strike price 5: " <<strikePrice[5]<<std::endl;
std::cout<<"Strike price 6: " <<strikePrice[6]<<std::endl;
std::cout<<"Strike price 7: " <<strikePrice[7]<<std::endl;
std::cout<<"Strike price 8: " <<strikePrice[8]<<std::endl;
std::cout<<"Strike price 9: " <<strikePrice[9]<<std::endl;
std::cout<<"Strike price 10: " <<strikePrice[10]<<std::endl;
std::cout<<"Implied volatility 1: " <<impliedVolatility[1]<<std::endl;
std::cout<<"Implied volatility 2: " <<impliedVolatility[2]<<std::endl;
std::cout<<"Implied volatility 3: " <<impliedVolatility[3]<<std::endl;
std::cout<<"Implied volatility 4: " <<impliedVolatility[4]<<std::endl;
std::cout<<"Implied volatility 5: " <<impliedVolatility[5]<<std::endl;
std::cout<<"Implied volatility 6: " <<impliedVolatility[6]<<std::endl;
std::cout<<"Implied volatility 7: " <<impliedVolatility[7]<<std::endl;
std::cout<<"Implied volatility 8: " <<impliedVolatility[8]<<std::endl;
std::cout<<"Implied volatility 9: " <<impliedVolatility[9]<<std::endl;
std::cout<<"Implied volatility 10: " <<impliedVolatility[10]<<std::endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
StrikePrice 1 = 70.54;
StrikePrice 2 = 74.21;
StrikePrice 3 = 72.12;
StrikePrice 4= 68.34;
StrikePrice 5 = 65.89;
StrikePrice 6 = 60.23;
StrikePrice 7 = 58.36;
StrikePrice 8 = 80.45;
StrikePrice 9 = 110.12;
StrikePrice 10= 100.34;
ImpliedVolatility 1 = 22.56;
ImpliedVolatility 2 = 24.16;
ImpliedVolatility 3 = 23.22;
ImpliedVolatility 4 = 21.86;
ImpliedVolatility 5 = 19.36;
ImpliedVolatility 6 = 18.16;
ImpliedVolatility 7 = 22.11;
ImpliedVolatility 8 = 23.22;
ImpliedVolatility 9 = 21.02;
ImpliedVolatility 10 = 24.21;
```

The chart in Excel will be as follows:


Implied volatility smile
Implied volatility
30
25
20
15
10
5
0
70.5
74.2
72.1
68.3
65.9
60.2
58.4
80.5
110
100
Strike prices
Strike price
Volatility

Calculation of percentage holding period return of a bond,(HPR).

A fixed income portfolio trader purchases a bond at 92.40. After 6 months the trader sells the bond at par or 100. Before selling the bond he receies a coupon of 3.75. Calculate the percentage hoding period return of the bond.

HPR = $\dfrac{P_t + C}{P_{t-1}} - 1 = \dfrac{100 + 3.75}{92.40} - 1 = 0.122835$

Percentage HPR = 0.122835*100 = 12.2835%

Application of Percentage holding period return of a bond in C++

```
// Percentage Holding period return of a bond.

#include <iostream>
using namespace std;

int main()
{
  double ParValue = 100;
  double Coupon = 3.75;
  double PurchasingPrice = 92.40;
  double PercentageHoldingPeriodReturn;

// Insert the mathematical formula.

 PercentageHoldingPeriodReturn = ((ParValue+Coupon)/PurchasingPrice-1)*100;

// Output function.

cout<< "Percentage Holding Period return
:"<<PercentageHoldingPeriodReturn<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Percentage Holding Period Return: 12.2835
Press any key to continue …
```

Calculation of correlation between Nordea bank and Danske bank.

The covariance between Nordea and Danske bank is 0.048. The standard deviation of Nordea bank is 0.20 and the standard deviation of Danske bank is 0.32. What is the correlation between these two banks?

$\rho_{1,2} = \frac{Cov_{1,2}}{\sigma_1 * \sigma_2} = \frac{0.048}{0.20 * 0.32} = 0.75$ There is a positive linear correlation between the two banks.

Application of calculation of correlation between Nordea bank and Danske bank in C++

```
// Calculation of correlation between Nordea and Danske bank.

#include <iostream>
using namespace std;

int main()
{

  //Identify the variables.

  double Covariance = 0.048;
  double StandardDeviationNordea = 0.20;
  double StandardDeviationDanske = 0.32;
  double Correlation;


// Insert the mathematical formula.

 Correlation = Covariance/(StandardDeviationNordea*StandardDeviationDanske);


// Output function.

cout<< "Correlation :"<<Correlation<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Correlation: 0.75
Press any key to continue …
```

Calculation of covariance between Piraeus and Alpha bank.

The correlation between Piraeus and Alpha bank is 0.15. The standard deviation of Piraeus bank is 0.17 and the standard deviation of Alpha bank is 0.12. What is the covariance between these two banks?

$$Cov_{1,2} = \rho_{1,2} * \sigma_1 * \sigma_2 = 0.15 * 0.17 * 0.12 = 0.00306$$

Application of calculation of covariance between Piraeus and Alpha bank in C++

```
// Calculation of covariance between Piraeus and Alpha bank.
#include <iostream>
using namespace std;

int main()
{

   //Identify the variables.

   double Correlation = 0.15;
   double StandardDeviationPiraeus = 0.17;
   double StandardDeviationAlpha = 0.12;
   double Covariance;


// Insert the mathematical formula.

  Covariance = Correlation*StandardDeviationPiraeus*StandardDeviationAlpha;


// Output function.

cout<< "Covariance :"<<Covariance<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Covariance: 0.00306
Press any key to continue …
```

Calculation of risk premium on corporate bond.

A trader estimates that corporate bond have a historical return of 0.043. If treasury bills returns was 0.022. Calculate the risk premium on coporate bond?

$$Risk\ \text{premium},\ \text{RP} = \frac{1+\text{r}_{\text{cb}}}{1+\text{r}_{\text{tb}}} - 1 = \frac{1+0.043}{1+0.022} - 1 = 0.0205$$

$$RP = 2.05\%$$

Application of calculation of risk premium on corporate bond in C++

```
// Calculation of  risk premium on corporate bond.
#include <iostream>
using namespace std;

int main()
{

    //Identify the variables.

   double ReturnCorporateBond = 0.043;
   double ReturnTreasuryBill = 0.022;
   double RiskPremiumPercentage;

// Insert the mathematical formula.

  RiskPremiumPercentage = ((1+ ReturnCorporateBond)/(1+ ReturnTreasuryBill)-
1)*100 ;

// Output function.

cout<< " Risk Premium Percentage :"<< RiskPremiumPercentage <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Risk Premium Percentage: 2.05%

Press any key to continue …
```

Calculation of portfolio variance and standard deviation.

An investment manager collected the following information about expected return, standard deviation, and correlation.

Asset	Expected return, (ER).	Standard deviation	Correlation
Shares	0.12	0.20	0.0
Silver	0.08	0.12	

If the portfolio weights consist of 0.80shares and 0.20 silver, then, calculate the variance and the standard deviation.

The variance and standard deviation mathematical formulas are as follows:

Variance: $s^2 = w_1^2 * \sigma_1^2 + w_2^2 * \sigma_2^2 + 2w_1 w_2 \rho_{1,2} \sigma_1 \sigma_2$

Standard deviation $\sigma_p = \sqrt{w_1^2 * \sigma_1^2 + w_2^2 * \sigma_2^2 + 2w_1 w_2 \rho_{1,2} \sigma_1 \sigma_2}$

As the $\rho_{1,2}$ or the correlation is zero, we do not include the third term in our equation.

Application of calculation of portfolio variance and standard deviation in C++

```
// Calculation of portfolio variance and standard deviation.
#include <iostream>
#include<cmath>
using namespace std;

int main()
{

   //Identify the variables.

   double WeightShare = 0.80;
   double StandardDeviationShare = 0.20;
   double WeightSilver= 0.20;
   double StandardDeviationSilver = 0.12;
   double Variance;
   double StandardDeviationPercentage;

/* Insert the mathematical formulas.As correlation is zero, we do not include the last
part of the equation.*/

  Variance =pow(WeightShare*StandardDeviationShare,2)+
   pow(WeightSilver*StandardDeviationSilver,2) ;
  StandardDeviationPercentage = sqrt(Variance)*100;

// Output functions.
cout<< " Variance :"<< Variance <<endl;
cout<< " Standard deviation Percentage :"<< StandardDeviationPercentage <<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Variance: 0.026176
Standard deviation Percentage: 16.179%

Press any key to continue …
```

Calculation of beta portfolio.

An investment trader has opened 1 short position on gold and 2 long positions on silver and FTSE100 index.The short position requires a negative sign. Calculate the beta portfolio. The beta portfolio is as follows:

$$\beta_p = \frac{\text{Amount}}{Market\,\text{index}} * beta$$

The following data are given:

Asset	Amount in Pounds	Beta
Gold	-200,000 Short position	- 0.5
Silver	270,000	0.5
FTSE100 index	400,000	1.0

Solution

$$\beta_p = \frac{\text{- 200,000}}{400{,}000} * (-0.5) + \frac{270{,}000}{400{,}000} * (0.5) + \frac{400{,}000}{400{,}000} * (1.0) = 1.5875$$

Application of beta portfolio in C++

```
// Calculation of beta portfolio.
#include <iostream>
using namespace std;

int main()
{
  double GoldAmount = -200.000;
  double SilverAmount = 270.000;
  double FTSE100Amount = 400.000;
  double GoldBeta = -0.5;
  double SilverBeta= 0.5;
  double FTSE100beta = 1.0;
  double BetaPortfolio;

// Insert the mathematical formula.

BetaPortfolio=GoldAmount/FTSE100Amount*-0.5+SilverAmount
/FTSE100Amount*0.5+FTSE100Amount/FTSE100Amount*1.0;

// Output function.

cout<< " Beta Portfolio :"<< BetaPortfolio <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Beta portfolio: 1.5875

Press any key to continue …
```

Calculation of Jensen’s alpha.

A fund manager has collected the following data:

Fund return: 0.086
Risk-free rate: 0.031
Fund beta: 1.5
FTSE 100 market return: 0.051

Calculate Jensen’s alpha.

Solution

The mathematical formula is as follows:

$$\alpha = r_i - [r_f + \beta_i (r_m - r_f)] = 0.086 - [0.031 + 1.5 * (0.051 - 0.031)] = 0.025$$

Application of Jensen’s alpha in C++

```
// Calculation of Jensen's alpha.
#include <iostream>
using namespace std;

int main()
{
   double FundReturn  = 0.086;
   double Riskfreerate = 0.031;
   double Beta = 1.5;
   double FTSE100MarketReturn = 0.051;
   double JensenAlpha;

// Insert the mathematical formula.

JensenAlpha=FundReturn-(Riskfreerate+Beta*(FTSE100MarketReturn-
Riskfreerate));

// Output function.
cout<< " Jensen's alpha :"<< JensenAlpha <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Jensen’s alpha: 0.025

Press any key to continue …
```

Calculation of beta portfolio using weights.

The beta of share A is 0.70 and the beta of share B is 0.30. The weight of share A is 0.50 and the weight of share B is 0.50. What is the beta portfolio?

The mathematical formula of beta portfolio is as follows:

$$\beta_p = w_A \beta_A + w_B \beta_B$$

Solution.

$$\beta_p = 0.50 * 0.70 + 0.50 * 0.30 = 0.5$$

Application of beta portfolio in C++

```
// Calculation of beta portfolio.
#include <iostream>
using namespace std;

int main()
{
   double WeightShareA = 0.50;
   double WeightShareB = 0.50;
   double BetaShareA =  0.70;
   double BetaShareB = 0.30;
   double BetaPortfolio;

// Insert the mathematical formula.

  BetaPortfolio = WeightShareA*BetaShareA+WeightShareB*BetaShareB;

// Output function.

cout<< " Beta Portfolio :"<< BetaPortfolio <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Beta portfolio: 0.5

Press any key to continue …
```

Calculation of the expected security return percentage.

An investment trader estimated the beta of a security as 1.5,the risk-free rate as 0.05 and the market retun as 0.15. Calculate the expected return of the security.

The mathematial formula of the CAPM is as follows:

$ER_i = r_f + \beta_i (r_m - r_f) = 0.05 + 1.5 * (0.15 - 0.05) = 0.2$

ER_i percentage = 20%

Application of the expected security return percentage in C++

```
// Calculation of the expected security return.
#include <iostream>
using namespace std;

int main()
{
   double Riskfreerate = 0.05 ;
   double Beta = 1.5 ;
   double MarketReturn = 0.15 ;
   double ExpectedReturnPercentage;

// Insert the mathematical formula.

ExpectedReturnPercentage = (Riskfreerate+Beta*(MarketReturn-Riskfreerate))*100;

// Output function.

cout<< " Expected Return Percentage  :"<< ExpectedReturnPercentage <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Expected return Percentage: 20%

Press any key to continue …
```

Calculation of the maintenance margin.

An investor purchased 50 Pounds shares and borrow 25 Pounds. The margin call is 30.13 pounds. Calculate the maintenance margin. The initial equity is 50 – 25 = 25 Pounds.

$$\mathit{Ma}\,\mathrm{int}\;\mathit{enance}\;\mathrm{margin} = \frac{\mathrm{Initial\ \ equity\ + margin\ \ call\ -\ initial\ \ price}}{\mathit{m}\arg \mathit{in}\ \mathrm{call}}$$

$$\mathit{Ma}\,\mathrm{int}\;\mathit{enance}\;\mathrm{margin} = \frac{25 + 30.13 - 50}{30.13} = 0.170262\ \mathrm{or}\ 17.0262\%$$

Application of the maintenance margin in C++

```
/ Calculation of the maintenance margin.
#include <iostream>
using namespace std;

int main()
{
  //Identify the variables.

  double InitialEquity  = 25 ;
  double MarginCall  = 30.13 ;
  double InitialPrice = 50 ;
  double MaintenanceMarginPercentage;

// Insert the mathematical formula.

MaintenanceMarginPercentage = (InitialEquity+MarginCall-
InitialPrice)/MarginCall*100 ;

// Output function.

cout<< " Maintenance Margin Percentage  :"<< MaintenanceMarginPercentage
<<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Maintenance margin percentage :  17.0262%
Press any key to continue …
```

<u>Calculation of the market price of a Treasury bond.</u>

A trader has estimated the spot rates of a 4 year Treasury bond.

Year	Spot rate or r
1	5.25%
2	5.50%
3	5.77%
4	6.10%

What is the market price of a Treasury bond with 8% annual coupon and a face value of 100 pounds?

The mathematical formula is as follows:

$$P = \frac{Coupon}{(1+r)} + \frac{Coupon}{(1+r)^2} + ... + \frac{Coupon + 100}{(1+r)^n}$$

<u>Solution.</u>

$$P = \frac{8}{1.0525} + \frac{8}{1.0550^2} + \frac{8}{1.0577^3} + \frac{108}{1.0610^4} = 106.77$$

Application of the market price of a Treasury bond in C++

```
// Calculation of the market price of a Treasury bond.
#include <iostream>
#include <cmath>
using namespace std;

int main()
{

   //Identify the variables.

   double CouponYear1 = 8 ;
   double CouponYear2 = 8;
   double CouponYear3 = 8;
   double CouponYear4 = 108 ;
   double SpotRateYear1 = 1.0525 ;
   double SpotRateYear2 = 1.0550;
   double SpotRateYear3 = 1.0577;
   double SpotRateYear4 = 1.0610;
   double MarketPrice;

// Insert the mathematical formula.

MarketPrice =
CouponYear1/SpotRateYear1+CouponYear2/pow(SpotRateYear2,2)+CouponYear3/
pow(SpotRateYear3,3)+CouponYear4/pow(SpotRateYear4,4);

// Output function.

cout<< " Market Price  :"<< MarketPrice <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Market Price : 106.77

Press any key to continue …
```

Gain or loss of stock index futures contracts.

An experienced investment trader bought 30 stock indes futures contracts at a price of 1455.34 pounds. He sold them at a price of 1577.22. If the pounds multiplier is 250, then, estimate the gain or loss of his trade.

The mathematical formula is as follows:

Gain or loss = Number of contracts * multiplier * $(f_t - f_0)$

Gain or loss = 30 *250 * (1577.22-1455.34) = 914100 gain.

Application of gain or loss of stock index futures contracts in C++

```
// Calculation of gain or loss of stock index futures contracts.
#include <iostream>
using namespace std;

int main()
{

   //Identify the variables.

   double NumberofContracts = 30 ;
   double Multiplier = 250;
   double BuyingPrice = 1455.34;
   double SellingPrice = 1577.22 ;
   double GainorLoss;

// Insert the mathematical formula.

GainorLoss = NumberofContracts*Multiplier*(SellingPrice-BuyingPrice);

// Output function.

cout<< " Gain or loss  :"<< GainorLoss <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Gain or loss : 914100

Press any key to continue …
```

The payoff of an interest rate call option.

An interest rate call option on 6 month LIBOR has a strike rate at 3%. The amount of the contract is 100000000 Pounds. At expiration the 6 month or 180 day LIBOR rate is 4.75%. Calculate the payoff at expiration.

The mathematical formula is as follows:

$$\mathit{Option}\ \text{payoff} = \text{Amount of the contract} * [(\text{LIBOR - strike}) * (\text{tdays}/360)]$$

$$\text{Option payoff} = 100000000 * [(0.0475 - 0.03) * (180/360)] = 875000 \text{ Pounds}$$

Application of payoff of an interest rate call option in C++

```
// Calculation of the payoff of an interest rate call option.
#include <iostream>
using namespace std;

int main()
{

   //Identify the variables.

  double AmountofContract = 100000000 ;
  double LIBOR = 0.0475;
  double Strike = 0.03;
  double tdays = 180 ;
  double Payoff;

// Insert the mathematical formula.

Payoff = AmountofContract*(LIBOR-Strike)*tdays/360;

// Output function.

cout<< " Payoff  :"<< Payoff <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Payoff : 875000

Press any key to continue …
```

Calculation of semiannual payment of Deutsche bank and JPMorgan currency swap.

Deutsche bank enters ino a three- year currency swap with JPMorgan. Under the swap terms, Deutsche bank will pay a fixed rate of 3% on 20000000 pounds ad JPMorgan will pay a fixed rate of 2.5% on 8000000 USD every 6 months. Calculate the first semiannual payment.

The mathematical formula is as follows:

Semi annual payment of Deutsche bank = principal * fixed rate * tdays/ 360 = 20000000*0.03*180/360 = 300000 Pounds.

Semi annual payment of JPMorgan = principal * fixed rate * tdays/ 360 = 8000000*0.025*180/360 = 100000 USD

Application of semiannual payment of Deutsche bank and JPMorgan currency swap in C++

```
// Calculation of semiannual payment of Deutsche bank and JPMorgan currency swap.
#include <iostream>
using namespace std;

int main()
{
   double PrincipalDeutscheBank = 20000000 ;
   double PrincipalJPMorganBank = 8000000;
   double FixedRateDeutscheBank = 0.03;
   double FixedRateJPMorgan = 0.025;
   double tdays = 180 ;
   double SemiannualPaymentDeutscheBank;
   double SemiannualPaymentJPMorgan;

// Insert the mathematical formulas.

SemiannualPaymentDeutscheBank =PrincipalDeutscheBank
*FixedRateDeutscheBank*tdays/360;
SemiannualPaymentJPMorgan =
PrincipalJPMorganBank*FixedRateJPMorgan*tdays/360;
// Output function.

cout<< " Semi annual payment Deutsche bank   :"<<
SemiannualPaymentDeutscheBank <<endl;
cout<< " Semi annual payment JPMorgan   :"<< SemiannualPaymentJPMorgan
<<endl;
system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following results:

```
Semiannual payment Deutsche bank : 300000
Semannual payment JPMorgan: 100000

Press any key to continue …
```

Calculation of the Sharpe ratio.

An investment manager has achieved a 15% annual return with variance 0.30. The risk-free rate of return is 8%. Calculate the Sharpe ratio.

The mathematical formula is as follows:

$$\textit{Sharpe}\ \text{ratio} = \frac{\mathrm{r_p} - r_f}{\sigma_p} = \frac{0.15 - 0.08}{\sqrt{0.30}} = 0.127802 = 0.13$$

$$\begin{aligned} \textit{Where}: & \mathrm{r_p}\ \text{is the portfolio return.} \\ & \mathrm{r_f}\ \text{is the risk - free rate.} \\ & \sigma_\mathrm{p}\ \textit{is the}\ \text{portfolio standard deviation.} \end{aligned}$$

Application of the Sharpe ratio in C++

```
// Calculation of the Sharpe ratio.
#include <iostream>
# include <math.h>
using namespace std;

int main()
{
  //Identify the variables.

  double PotfolioReturn = 0.15 ;
  double RiskfreeRate = 0.08;
  double Variance = 0.30;
  double SharpeRatio;

// Insert the mathematical formula.

SharpeRatio =(PotfolioReturn-RiskfreeRate)/sqrt(Variance) ;

// Output function.

cout<< "Sharpe ratio:"<< SharpeRatio <<endl;

system ("PAUSE");
return 0;
}
```

Output

After compiling and debugging , the DOS window or console will open and display the following result:

```
Sharpe ratio: 0.127802
```

```
Press any key to continue …
```

Reference and essential reading

Chandan Sengupta (2007), Financial Modeling using C++. Wiley Finance.