* 1. Futures Value

#include <iostream>  
#include <math.h>  
using namespace std;  
  
*//1.1 future value  
//) today’s cashflow F0, today’s time t0, future time t1,  
// continuously compounded interest rate r in %*double future\_value(double F0, double t0, double t1, double r)  
{  
 double r\_decimal = 0.01\*r;  
 double F1 = F0\*exp(r\_decimal\*(t1-t0));  
 return F1;  
}  
  
  
int main(){

cout<<future\_value(1000.0,5.0,10.0,20.0);  
  
}

console output:

2718.28

* 1. Discount factor

#include <iostream>  
#include <math.h>  
using namespace std;  
  
*//1.2 discount factor  
//inputs: today’s cashflow F0, future cashflow F1, today’s time t0 , future time t1.  
//output : discount factor df, continuously compounded interest rate r*int df\_and\_r(double F0, double F1, double t0, double t1, double & df, double & r)  
{  
if (t1-t0 == 0.0) {  
 df = 0;  
 r = 0;  
 return -1;  
}  
if ((F0 < 0.0) || (F1 < 0.0)) {  
 df = 0;  
 r = 0;  
 return -2;  
}  
  
*// \*\*\* you have to write the rest \*\*\** df = F0/F1;  
 r = -(log(df)/(t1-t0));  
  
  
return 0;  
}  
  
  
int main(){  
 double F0 = 1000;  
 double F1 = 2718.28;  
 double t0 = 5.0;  
 double t1 = 10.0;  
 double df = 0.0;  
 double r = 0.0;  
  
cout<<"Error code: " <<df\_and\_r(F0,F1,t0,t1,df,r);  
 cout<<"\ndiscount factor: " <<df << "\ninterest rate: "<< r\*100 <<"%" ;  
}

Console output:

Error code: 0  
discount factor: 0.36788  
interest rate: 20%

Using the same result from 1.1 F1 = 2718.28

* 1. is able to return the correct interest input of 20% from 1.1
  2. Bond price from yield

#include <iostream>  
#include <math.h>  
using namespace std;  
  
*//1.3 Bond price from yield  
//yield = 5% -> y = 5*void price\_from\_yield(double F, double c, double y, int n, double & B){  
 double y\_decimal = y \*0.01;  
 double numerator = .5\*c;  
 double denominator = (1+.5\*y\_decimal);  
  
 for(int i=1; i <=n-1; i ++){  
 B+= numerator/pow(denominator,i);  
 }  
 B+= (F+numerator)/pow(denominator,n);  
  
}  
  
int main(){  
 double F = 100;  
 double c = 0;  
 double y = 0;  
 int n = 10;  
 double B = 0.0;  
 price\_from\_yield(F,c,y,n,B);  
 cout<<"B: " << B;  
 cout<<" expected: 100\n";  
  
 F = 100;  
 c = 5;  
 y = 0;  
 B = 0.0;  
 price\_from\_yield(F,c,y,n,B);  
 cout<<"B: " << B;  
 cout<<" expected: "<<F+(n\*c/2);  
  
 F = 100;  
 c = 0;  
 y = 5;  
 B = 0.0;  
 price\_from\_yield(F,c,y,n,B);  
 cout<<"\nB: " << B;  
 cout<<" expected: "<<F/(pow(1+(.5\*y\*0.01),n));  
   
}

Console output:

B: 100 expected: 100  
B: 125 expected: 125  
B: 78.1198 expected: 78.1198

1.4 Yield from bond price

#include <iostream>  
#include <cmath>  
  
using namespace std;  
  
*//1.4 Yield from bond price*static void price\_from\_yield(double F, double c, double y, int n, double & B){  
 double y\_decimal = y \*0.01;  
 double numerator = .5\*c;  
 double denominator = (1+.5\*y\_decimal);  
  
 for(int i=1; i <=n-1; i ++){  
 B+= numerator/pow(denominator,i);  
 }  
 B+= (F+numerator)/pow(denominator,n);  
  
}*//1.3*static int yield\_from\_price(double F, double c, int n, double B\_market,  
 double tol, int max\_iter, double & y){  
 int error\_code = 1;  
 double y\_low = 0.0;  
 double y\_high =100.0;  
  
 double B\_y\_low = 0.0;  
 *//y\_low* price\_from\_yield(F,c,y\_low,n,B\_y\_low);  
 if (abs(B\_y\_low-B\_market) <= tol){  
 y = y\_low;  
 error\_code = 0;  
 return 0;  
 }  
 if (B\_y\_low < B\_market){  
 y=0;  
 error\_code = 1;  
 return 1;  
 }  
  
 *//y\_high* double B\_y\_high = 0.0;  
 price\_from\_yield(F,c,y\_high,n,B\_y\_high);  
 if(abs(B\_y\_high - B\_market) <= tol){  
 y = y\_high;  
 error\_code =0;  
 return 0;  
 }  
 if(B\_y\_high>B\_market){  
 y=0;  
 error\_code=1;  
 return 1;  
 }  
  
 *//bisection loop* for(int i=0; i < max\_iter; ++i){  
 y=(y\_low+y\_high)/2.0;  
 double B = 0.0;  
 price\_from\_yield(F,c,y,n,B); *//12* if (abs(B-B\_market) <= tol){  
 error\_code = 0;  
 return 0;  
 }  
 else if(B > B\_market){  
 y\_low =y;  
 }  
 else{  
 y\_high=y;  
 }  
  
 if(y\_high-y\_low <= tol){  
 error\_code =0;  
 return 0;  
 }  
 }  
  
 *//after loop* if (error\_code ==1){  
 y = 0;  
 return 1;  
 }  
}*// bisection*int main(){  
 double F = 100;  
 double B\_market = 100;  
 double c = 40;  
 double y=0;  
 double tol = 0.001;  
 int n = 10;  
 int max\_iter = 100;  
 int error\_code = yield\_from\_price(F,c,n,B\_market,tol,max\_iter,y);  
 cout<<"error code: " << error\_code <<'\n';  
 cout<<"y: "<< y <<" expected = "<<c<<'\n';  
  
  
 B\_market =90;  
 c = 20;  
 y=0;  
 error\_code = yield\_from\_price(F,c,n,B\_market,tol,max\_iter,y);  
 cout<<"error code: " << error\_code <<'\n';  
 cout<<"y: "<< y <<" expected greater than "<<c<<'\n';  
  
 B\_market =110;  
 c = 20;  
 y=0;  
 error\_code = yield\_from\_price(F,c,n,B\_market,tol,max\_iter,y);  
 cout<<"error code: " << error\_code <<'\n';  
 cout<<"y: "<< y <<" expected less than than "<<c<<'\n';  
  
 B\_market =110;  
 c = 0;  
 y=0;  
 error\_code = yield\_from\_price(F,c,n,B\_market,tol,max\_iter,y);  
 cout<<"error code: " << error\_code <<'\n';  
 cout<<"y: "<< y <<" expected less than than "<<c<<'\n';  
  
}

Console output:

error code: 0  
y: 40.0002 expected = 40  
error code: 0  
y: 23.5039 expected greater than 20  
error code: 0  
y: 16.9548 expected less than than 20  
error code: 1  
y: 0 expected less than than 0