Solver01.h -implementing Function Pointers

```
#ifndef Solver01_h
#define Solver01_h
double SolveByBisect(double(*Fct)(double x),
double Tgt, double LEnd, double REnd, double Acc)
{
double left=LEnd, right=REnd, mid=(left+right)/2;
double y_left=Fct(left)-Tgt, y_mid=Fct(mid)-Tgt;
while (mid-left>Acc)
{
if ((y_left>0 && y_mid>0)||(y_left<0 && y_mid<0))
{left=mid; y_left=y_mid;}
else right=mid;
mid=(left+right)/2;
y_mid=Fct(mid)-Tgt;
}
return mid;
}
```

Solver01.h (continue)

```
double SolveByNR(double(*Fct)(double x),
double(*DFct)(double x), double Tgt, double Guess, double Acc)
{
double x_prev=Guess;
double x_next=x_prev-(Fct(x_prev)-Tgt)/DFct(x_prev);
while (x_next-x_prev>Acc || x_prev-x_next>Acc)
{
x_prev=x_next;
x_next=x_prev-(Fct(x_prev)-Tgt)/DFct(x_prev);
}
return x_next;
}
#endif
```

Solver02.h –implementing virtual functions

```
#ifndef Solver02_h
#define Solver02_h
class Function
{
public:
virtual double Value(double x)=0;
virtual double Deriv(double x)=0;
};
```

Solver02.h (continue)

```
double SolveByBisect(Function* Fct,
double Tgt, double LEnd, double REnd, double Acc)
{
double left=LEnd, right=REnd, mid=(left+right)/2;
double y_left=Fct->Value(left)-Tgt, y_mid=Fct->Value(mid)-Tgt;
while (mid-left>Acc)
{
if ((y_left>0 && y_mid>0)||(y_left<0 && y_mid<0))
{left=mid; y_left=y_mid;}
else right=mid;
mid=(left+right)/2;
y_mid=Fct->Value(mid)-Tgt;
}
return mid;
```

Solver02.h (continue)

```
double SolveByNR(Function* Fct, double Tgt, double Guess, double
Acc)
{
double x_prev=Guess;
double x_next=x_prev -(Fct->Value(x_prev)-Tgt)/Fct->Deriv(x_prev);
while (x_next-x_prev>Acc || x_prev-x_next>Acc)
{
x_prev=x_next;
x_next=x_prev -(Fct->Value(x_prev)-Tgt)/Fct->Deriv(x_prev);
}
return x_next;
}
#endif
```

Main16.cpp

```
#include "Solver02.h"
#include <iostream>
using namespace std;
class F1: public Function
{ public:
double Value(double x){return x*x-2;}
double Deriv(double x){return 2*x;}
};
class F2: public Function
{ private:
double a;
public:
F2(double a_){a=a_;}
double Value(double x){return x*x-a;}
double Deriv(double x){return 2*x;}
}
int main()
{
F1 MyF1;
F2 MyF2(3.0);
double Acc=0.001, LEnd=0.0, REnd=2.0, Tgt=0.0;
cout << "Root of F1 by bisect: " <<
SolveByBisect(&MyF1,Tgt,LEnd,REnd,Acc) << endl;
cout << "Root of F2 by bisect: " <<
SolveByBisect(&MyF2,Tgt,LEnd,REnd,Acc) << endl;
double Guess=1.0;
cout << "Root of F1 by Newton-Raphson: " <<
SolveByNR(&MyF1,Tgt,Guess,Acc) << endl;
cout << "Root of F2 by Newton-Raphson: " <<
SolveByNR(&MyF2,Tgt,Guess,Acc) << endl;
return 0;
}
```

Solver03.h

```
#ifndef Solver03_h
#define Solver03_h
template<typename Function>
double SolveByBisect(Function* Fct, double Tgt, double LEnd, double REnd, double Acc)
{
double left=LEnd, right=REnd, mid=(left+right)/2;
double y_left=Fct->Value(left)-Tgt, y_mid=Fct->Value(mid)-Tgt;
while (mid-left>Acc)
{
if ((y_left>0 && y_mid>0)||(y_left<0 && y_mid<0))
{left=mid; y_left=y_mid;}
else right=mid;
mid=(left+right)/2;
y_mid=Fct->Value(mid)-Tgt;
}
return mid;
}
template<typename Function>
double SolveByNR(Function* Fct, double Tgt, double Guess, double Acc)
{
double x_prev=Guess;
double x_next=x_prev -(Fct->Value(x_prev)-Tgt)/Fct->Deriv(x_prev);
while (x_next-x_prev>Acc || x_prev-x_next>Acc)
{
x_prev=x_next;
x_next=x_prev -(Fct->Value(x_prev)-Tgt)/Fct->Deriv(x_prev);
}
return x_next;
}
#endif
```

Main17.cpp

```
#include "Solver03.h"
#include <iostream>
using namespace std;
class F1
{
public:
double Value(double x){return x*x-2;}
double Deriv(double x){return 2*x;}
};
class F2
{
private:
double a;
public:
F2(double a_){a=a_;}
double Value(double x){return x*x-a;}
double Deriv(double x){return 2*x;}
};
int main()
{
F1 MyF1;
F2 MyF2(3.0);
double Acc=0.001;
double LEnd=0.0, REnd=2.0;
double Tgt=0.0;
cout << "Root of F1 by bisect: "
<<SolveByBisect<F1>(&MyF1,Tgt,LEnd,REnd,Acc) << endl;
cout << "Root of F2 by bisect: "<<
SolveByBisect<F2>(&MyF2,Tgt,LEnd,REnd,Acc) << endl;
double Guess=1.0;
cout << "Root of F1 by Newton-Raphson: “ <<
SolveByNR<F1>(&MyF1,Tgt,Guess,Acc)
<< endl;
cout << "Root of F2 by Newton-Raphson: “ <<
SolveByNR<F2>(&MyF2,Tgt,Guess,Acc)
<< endl;
return 0;
}
```

EurCall.h

```
#ifndef EurCall_h
#define EurCall_h
class EurCall
{
public:
double T, K;
EurCall(double T_, double K_){T=T_; K=K_;}
double d_plus(double S0, double sigma, double r);
double d_minus(double S0, double sigma, double r);
double PriceByBSFormula(double S0, double sigma, double r);
double VegaByBSFormula(double S0, double sigma, double r);
};
#endif
```

EurCall.cpp

```
#include "EurCall.h"
#include <cmath>
double N(double x)
{
double gamma = 0.2316419;double a1 = 0.319381530;
double a2 =-0.356563782; double a3 = 1.781477937;
double a4 =-1.821255978; double a5 = 1.330274429;
double pi = 4.0*atan(1.0); double k = 1.0/(1.0+gamma*x);
if (x>=0.0)
{
return 1.0-((((a5*k+a4)*k+a3)*k+a2)*k+a1)*k*exp(-
x*x/2.0)/sqrt(2.0*pi);
}
else return 1.0-N(-x);
}
double EurCall::d_plus(double S0, double sigma, double r)
{
return (log(S0/K)+(r+0.5*pow(sigma,2.0))*T)/(sigma*sqrt(T));
}
double EurCall::d_minus(double S0, double sigma, double r)
{
return d_plus(S0,sigma,r)-sigma*sqrt(T);
}
double EurCall::PriceByBSFormula(double S0, double sigma,
double r)
{
return S0*N(d_plus(S0,sigma,r)) -K*exp(-
r*T)*N(d_minus(S0,sigma,r));
}
double EurCall::VegaByBSFormula(double S0, double sigma,
double r)
{
double pi=4.0*atan(1.0);
return S0*exp(-
d_plus(S0,sigma,r)*d_plus(S0,sigma,r)/2)*sqrt(T)/sqrt(2.0*pi);
}
```

Main18.cpp

```
#include "Solver03.h"
#include "EurCall.h"
#include <iostream>
using namespace std;
class Intermediary: public EurCall
{ private:
double S0,r;
public:
Intermediary(double S0_, double r_, double T_, double K_) :
EurCall(T_,K_) {S0=S0_; r=r_;}
double Value(double sigma)
{
return PriceByBSFormula(S0,sigma,r);
}
double Deriv(double sigma)
{
return VegaByBSFormula(S0,sigma,r);
}
};
int main()
{
double S0=100.0;
double r=0.1;
double T=1.0;
double K=100.0;
Intermediary Call(S0,r,T,K);
double Acc=0.001;
double LEnd=0.01, REnd=1.0;
double Tgt=12.56;
cout << "Implied vol by bisect: “ << SolveByBisect<
Intermediary>(&Call,Tgt,LEnd,REnd,Acc) << endl;
double Guess=0.23;
cout << "Implied vol by Newton-Raphson: “ << SolveByNR <
Intermediary>(&Call,Tgt,Guess,Acc) << endl;
return 0;
}
```