Listing 3.1 Options07.h

```
#ifndef Options07_h
#define Options07_h
#include "BinModel02.h"
class EurOption
{
private:
int N;
public:
void SetN(int N_){N=N_;}
virtual double Payoff(double z)=0;
double PriceByCRR(BinModel Model);
};
class AmOption
{
private:
int N;
public:
void SetN(int N_){N=N_;}
virtual double Payoff(double z)=0;
double PriceBySnell(BinModel Model);
};
class Call: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
class Put: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
#endif
```

Listing 3.2 Options07.cpp

```
#include "Options07.h"
#include "BinModel02.h"
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
double EurOption::PriceByCRR(BinModel Model)
{
double q=Model.RiskNeutProb();
```

```
vector<double> Price(N+1);
for (int i=0; i<=N; i++)
{
Price[i]=Payoff(Model.S(N,i));
}
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
Price[i]=(q*Price[i+1]+(1-q)*Price[i])
/(1+Model.GetR());
}
}
return Price[0];
}
double AmOption::PriceBySnell(BinModel Model)
{
double q=Model.RiskNeutProb();
vector<double> Price(N+1);
double ContVal;
for (int i=0; i<=N; i++)
{
Price[i]=Payoff(Model.S(N,i));
}
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
ContVal=(q*Price[i+1]+(1-q)*Price[i])
/(1+Model.GetR());
Price[i]=Payoff(Model.S(n,i));
if (ContVal>Price[i]) Price[i]=ContVal;
}
}
return Price[0];
}
int Call::GetInputData()
{
cout << "Enter call option data:" << endl;
int N;
cout << "Enter steps to expiry N: "; cin >> N;
EurOption::SetN(N); AmOption::SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
return 0;
}
double Call::Payoff(double z)
{
if (z>K) return z-K;
return 0.0;
}
int Put::GetInputData()
{
cout << "Enter put option data:" << endl;
int N;
```

```
cout << "Enter steps to expiry N: "; cin >> N;
EurOption::SetN(N); AmOption::SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
return 0;
}
double Put::Payoff(double z)
{
if (z<K) return K-z;
return 0.0;
}
```

Listing 3.3 Main12.cpp

```
#include "BinModel02.h"
#include "Options07.h"
#include <iostream>
using namespace std;
int main()
{
BinModel Model;
if (Model.GetInputData()==1) return 1;
Call Option1;
Option1.GetInputData();
cout << "European call option price = "
<< Option1.PriceByCRR(Model)
<< endl;
cout << "American call option price = "
<< Option1.PriceBySnell(Model)
<< endl << endl;
Put Option2;
Option2.GetInputData();
cout << "European put option price = "
<< Option2.PriceByCRR(Model)
<< endl;
cout << "American put option price = "
<< Option2.PriceBySnell(Model)
<< endl << endl;
return 0;
}
```

Listing 3.4 Options08.h

```
#ifndef Options08_h
#define Options08_h
#include "BinModel02.h"
class Option
{
private:
int N;
```

```
public:
void SetN(int N_){N=N_;}
int GetN(){return N;}
virtual double Payoff(double z)=0;
};
class EurOption: public virtual Option
{
public:
double PriceByCRR(BinModel Model);
};
class AmOption: public virtual Option
{
public:
double PriceBySnell(BinModel Model);
};
class Call: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
class Put: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
#endif
```

Listing 3.5 Options08.cpp

```
#include "Options08.h"
#include "BinModel02.h"
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
double EurOption::PriceByCRR(BinModel Model)
{
double q=Model.RiskNeutProb();
int N=GetN();
vector<double> Price(N+1);
for (int i=0; i<=N; i++)
{
Price[i]=Payoff(Model.S(N,i));
}
```

```
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
Price[i]=(q*Price[i+1]+(1-q)*Price[i])
/(1+Model.GetR());
}
}
return Price[0];
}
double AmOption::PriceBySnell(BinModel Model)
{
double q=Model.RiskNeutProb();
int N=GetN();
vector<double> Price(N+1);
double ContVal;
for (int i=0; i<=N; i++)
{
Price[i]=Payoff(Model.S(N,i));
}
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
ContVal=(q*Price[i+1]+(1-q)*Price[i])
/(1+Model.GetR());
Price[i]=Payoff(Model.S(n,i));
if (ContVal>Price[i]) Price[i]=ContVal;
}
}
return Price[0];
}
int Call::GetInputData()
{
cout << "Enter call option data:" << endl;
int N;
cout << "Enter steps to expiry N: "; cin >> N;
SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
return 0;
}
double Call::Payoff(double z)
{
if (z>K) return z-K;
return 0.0;
}
int Put::GetInputData()
{
cout << "Enter put option data:" << endl;
int N;
cout << "Enter steps to expiry N: "; cin >> N;
SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
```

```
return 0;
}
double Put::Payoff(double z)
{
if (z<K) return K-z;
return 0.0;
}
```

Listing 3.6 BinLattice01.h

```
#ifndef BinLattice01_h
#define BinLattice01_h
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
class BinLattice
{
private:
int N;
vector< vector<double> > Lattice;
public:
void SetN(int N_)
{
N=N_;
Lattice.resize(N+1);
for(int n=0; n<=N; n++) Lattice[n].resize(n+1);
}
void SetNode(int n, int i, double x)
{Lattice[n][i]=x;}
double GetNode(int n, int i)
{return Lattice[n][i];}
void Display()
{
cout << setiosflags(ios::fixed)
<< setprecision(3);
for(int n=0; n<=N; n++)
{
for(int i=0; i<=n; i++)
cout << setw(7) << GetNode(n,i);
cout << endl;
}
cout << endl;
}
};
#endif
```

Listing 3.7 BinLattice02.h

```
#ifndef BinLattice02_h
```

```
#define BinLattice02_h
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
template<typename Type> class BinLattice
{
private:
int N;
vector< vector<Type> > Lattice;
public:
void SetN(int N_)
{
N=N_;
Lattice.resize(N+1);
for(int n=0; n<=N; n++) Lattice[n].resize(n+1);
}
void SetNode(int n, int i, Type x)
{Lattice[n][i]=x;}
Type GetNode(int n, int i)
{return Lattice[n][i];}
void Display()
{
cout << setiosflags(ios::fixed)
<< setprecision(3);
for(int n=0; n<=N; n++)
{
for(int i=0; i<=n; i++)
cout << setw(7) << GetNode(n,i);
cout << endl;
}
cout << endl;
}
};
#endif
```

Listing 3.8 Options09.h

```
#ifndef Options09_h
#define Options09_h
#include "BinLattice02.h"
#include "BinModel02.h"
class Option
{
private:
int N;
```

```
public:
void SetN(int N_){N=N_;}
int GetN(){return N;}
virtual double Payoff(double z)=0;
};
class EurOption: public virtual Option
{
public:
double PriceByCRR(BinModel Model);
};
class AmOption: public virtual Option
{
public:
double PriceBySnell(BinModel Model,
BinLattice<double>& PriceTree,
BinLattice<bool>& StoppingTree);
};
class Call: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
class Put: public EurOption, public AmOption
{
private:
double K;
public:
void SetK(double K_){K=K_;}
int GetInputData();
double Payoff(double z);
};
#endif
```

Listing 3.9 Options09.cpp

```
#include "Options09.h"
#include "BinModel02.h"
#include "BinLattice02.h"
#include <iostream>
#include <cmath>
using namespace std;
double EurOption::PriceByCRR(BinModel Model)
{
double q=Model.RiskNeutProb();
int N=GetN();
vector<double> Price(N+1);
```

```
for (int i=0; i<=N; i++)
{
Price[i]=Payoff(Model.S(N,i));
}
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
Price[i]=(q*Price[i+1]+(1-q)*Price[i])
/(1+Model.GetR());
}
}
return Price[0];
}
double AmOption::PriceBySnell(BinModel Model,
BinLattice<double>& PriceTree,
BinLattice<bool>& StoppingTree)
{
double q=Model.RiskNeutProb();
int N=GetN();
PriceTree.SetN(N);
StoppingTree.SetN(N);
double ContVal;
for (int i=0; i<=N; i++)
{
PriceTree.SetNode(N,i,Payoff(Model.S(N,i)));
StoppingTree.SetNode(N,i,1);
}
for (int n=N-1; n>=0; n--)
{
for (int i=0; i<=n; i++)
{
ContVal=(q*PriceTree.GetNode(n+1,i+1)
+(1-q)*PriceTree.GetNode(n+1,i))
/(1+Model.GetR());
PriceTree.SetNode(n,i,Payoff(Model.S(n,i)));
StoppingTree.SetNode(n,i,1);
if (ContVal>PriceTree.GetNode(n,i))
{
PriceTree.SetNode(n,i,ContVal);
StoppingTree.SetNode(n,i,0);
}
else if (PriceTree.GetNode(n,i)==0.0)
{
StoppingTree.SetNode(n,i,0);
}
}
}
return PriceTree.GetNode(0,0);
}
int Call::GetInputData()
{
cout << "Enter call option data:" << endl;
int N;
cout << "Enter steps to expiry N: "; cin >> N;
```

```
SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
return 0;
}
double Call::Payoff(double z)
{
if (z>K) return z-K;
return 0.0;
}
int Put::GetInputData()
{
cout << "Enter put option data:" << endl;
int N;
cout << "Enter steps to expiry N: "; cin >> N;
SetN(N);
cout << "Enter strike price K: "; cin >> K;
cout << endl;
return 0;
}
double Put::Payoff(double z)
{
if (z<K) return K-z;
return 0.0;
}
```

Listing 3.10 Main14.cpp

```
#include "BinLattice02.h"
#include "BinModel02.h"
#include "Options09.h"
#include <iostream>
using namespace std;
int main()
{
BinModel Model;
if (Model.GetInputData()==1) return 1;
Put Option;
Option.GetInputData();
BinLattice<double> PriceTree;
BinLattice<bool> StoppingTree;
Option.PriceBySnell(Model,PriceTree,StoppingTree);
cout << "American put prices:" << endl << endl;
PriceTree.Display();
cout << "American put exercise policy:"
<< endl << endl;
StoppingTree.Display();
return 0;
}
```