HomeWork2

Q.2.2) Modify the pay-off class so that it can handle double-digital options.

Solution : **Step 1**.) Overloaded the constructor of Payoff Class for Double Digital , added one more variable in ENUM, two more private variable for upper and lower bound strike price and one switch case for double digital.

Code Snippet:

Payoff1.h

```
enum  OptionType {call, put, double_digital};
PayOff(double Strike_, OptionType TheOptionsType_);
PayOff(double Strike_Lower,double Strike_Upper, OptionType TheOptionsType_); // CONSTRUCTOR OVERLOADED FOR
DOUBLE DIGITALS
double operator()(double Spot) const;
private:
double Strike;
double Strike_Low; // ADDED LOWER BOUND
double Strike_Up; // ADDED UPPER BOUND
```

Payoff1.cpp

```
PayOff::PayOff(double Strike_, OptionType TheOptionsType_)
:
   Strike(Strike_),Strike_Low(0),Strike_Up(0), TheOptionsType(TheOptionsType_)
{
}

PayOff::PayOff(double Strike_Lower,double Strike_Upper, OptionType TheOptionsType_)  // Overloading the Constructor
adding 2 arguments for Lower and Upper Strike Price
:
Strike(0),Strike_Low(Strike_Lower),Strike_Up(Strike_Upper), TheOptionsType(TheOptionsType_)
{
}

double PayOff::operator ()(double spot) const
{
   switch (TheOptionsType)
   {
   case call :
      return max(spot-Strike,0.0);
```

```
    case put:
        return max(Strike-spot,0.0);

    case double_digital:   // added double digit Payoff case
            if (spot <= Strike_Low)
                return 0;
            if (spot >= Strike_Up)
                return 0;

            return 1;


    default:
        throw("unknown option type found.");

    }
}
```

Step 2.) in the main.cpp call the constructor of Payoff class which takes two Strike prices-(Lower and upper bound).

Code Snippet:

main.cpp

```
PayOff doubleDigitalPayoff(Strike_Low, Strike_Up, PayOff::double_digital); // Payoff Object for Double Digital

....
double resultDouble = SimpleMonteCarlo2(doubleDigitalPayoff,
                                        Expiry,
                                        Spot,
                                        Vol,
                                        r,
                                        NumberOfPath);
```

Q.2.3) Test whether on your compiler using const speeds code up.

Solution : Used <time.h> file and used its clock() method to get the time at the start and end of the code, Code with **const** took less time to execute than the code without **const** in it.

Code Snippet:

Main.cpp

```
#include <time.h>

clock_t start1,end1;
start1=clock();// getting time at the start of the code

double Expiry;
double Strike;
double Spot;

....

end1 = clock(); // end of the code

cout<<"Elapsed time-clock :"<< (double)(end1-start1)/ CLOCKS_PER_SEC<<"\n";
```

Attached Results Screenshots in next page:

Result Without Const: 0.305844 seconds

```
Enter expiry:
1

Enter Strike:
100

Enter spot:
150

Enter vol:
0.08

Enter r:
0.05

Enter low barrier
100

Enter up barrier
200

Number of paths:
1000000

the price are 54.8877 for the call and
0 for the put

 the price for double digital with low barrier
= 100 and up barrier = 200 is 0.949979
Elapsed time-clock :0.305844
Program ended with exit code: 0
```

Result With Const: 0.288407 seconds

```
Enter expiry:
1

Enter Strike:
100

Enter spot:
150

Enter vol:
0.08

Enter r:
0.05

Enter low barrier
100

Enter up barrier
200

Number of paths:
1000000

the price are 54.8877 for the call and
0 for the put

 the price for double digital with low barri
= 100 and up barrier = 200 is 0.949979
Elapsed time-clock :0.288407
Program ended with exit code: 0
```