FE545 Homework Assignment #1

Due Date: Feb. 13th (Tuesday).

European options define the timeframe when holders of an options contract may exercise their contract rights. The rights for the option holder include buying the underlying asset or selling the underlying asset at the specified contract price - the strike price. With European options, the holder may only exercise their rights on the day of expiration.

An European call option gives the owner the right to acquire the underlying security at expiry. For an investor to profit from a call option, the stock's price, at expiry, has to be trading high enough above the strike price to cover the cost of the option premium. An European put option allows the holder to sell the underlying security at expiry. For an investor to profit from a put option, the stock's price, at expiry, has to be trading far enough below the strike price to cover the cost of the option premium.

This makes European options less complicated than other types of options. Options sellers don't have to worry about the option getting exercised early. Similarly, the options buyers don?t have to spend time trying to determine the optimal time to exercise the contract. They simply wait until the expiration date to decide.

Suppose the underlying asset price S_t at time t, and strike price K and expiration T. And then the call and put payoffs (at expiration time $t = T$) will be:

$$C_T(S_T, K) = max(S_T - K, 0), \text{ and } P_T(S_T, K) = max(K - S_T, 0)$$

Implement PayOff classes in $PayOff.h$ according to the following template:

```
#ifndef __PAY_OFF__
#define __PAY_OFF__

#include <algorithm> // This is needed for the std::max
// comparison function, used in the pay-off calculations

class PayOff {
 public:
  PayOff(); // Default (no parameter) constructor
  virtual ~PayOff() {}; // Virtual destructor

  // Overloaded () operator, turns the PayOff into an abstract
  // function object
  virtual double operator(const double& S) const = 0;
};
```

```
class BasePayOffParameters {

  public:
  PayOffParameters(); // Default (no parameter) constructor
  virtual ~PayOffParameters() {} = 0; // Virtual destructor
}

class PayOffParameters : BasePayOffParameters {

  public:
  PayOffParameters(const double& K);
  virtual ~PayOffParameters() {}; // Virtual destructor

  double GetStrke();
}

class PayOffCall : public PayOff {
 private:
  double K; // Strike price

 public:
  PayOffCall(const PayOffParameters& Param_);
  virtual ~PayOffCall() {};

  // Virtual function is now over-ridden
```

```
  // (not pure-virtual anymore)
  virtual double operator(const double& S) const;
};

class PayOffPut : public PayOff {
 private:
  double K; // Strike

 public:
  PayOffPut(const PayOffParameters& Param_);
  virtual ~PayOffPut() {};
  virtual double operator(const double& S) const;
};

#endif
```

A double digital option is a particular variety of option (a financial derivative). At maturity, the payoff is 1 if the spot price of the underlying asset is between two numbers, the lower and upper strikes of the option; otherwise, it is 0.

A double digital option is similar to the exotic option with a few exceptions. for instance a double digital option has two strike prices that is the expected price during the trade season. The option has two types of strikes namely the lower and the upper strikes.

A double digital with lower strike K_1 and upper strike K_2 can be replicated by going long a digital option with strike K_1 and short another digital option with strike K_2.

For this assignment, you need to define a derived class called $PayOffDoubleDigital$ from the base class called $PayOff$ which implements the PayOff operator according to formula (1).

$$D_T = \begin{cases} 0 & \text{for} \quad S_T \geq K_2 \\ 0 & \text{for} \quad S_T \leq K_1 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

The class $DoubleDigital.h$ should follow the following prototype:

```
#ifndef __PayOffDoubleDigital__
#define __PayOffDoubleDigital__



#include <vector>
#include "PayOff.h"

class DoubleDigitalPayOffParameters : BasePayOffParameters{

  public:
  DoubleDigitalPayOffParameters(const double& K_1, const double& K_2
  virtual ~PayOffParameters() {}; // Virtual destructor
```

```
  double GetLowerStrke();
  double GetUpperStrike();
};

class PayOffDoubleDigital : public PayOff {
 private:
  double K1; // Strike lower level
  double K2; // Strike upper level

 public:
  PayOffDoubleDigital(PayOffParameter Param_);
  virtual ~PayOffDoubleDigital() {};

  // Pay-off operator (this will determine arithmetic or geometric)
  virtual double operator(const double& S) const;
};

#endif
```

You will also need to implement a *SimpleMonteCarlo* class with a function to implement the simulation procedure:

```
#ifndef __Option_Class__SimpleMonteCarlo__
#define __Option_Class__SimpleMonteCarlo__
```

```
#include <iostream>
#include "DoubleDigital.h"

double SimpleMonteCarlo(const PayOff& ThePayOff_,
                        double Spot,
                        double Vol,
                        double r,
                        unsigned long NumberOfPaths);


#endif /* defined(__Option_Class__SimpleMonteCarlo__) */
```

Overall, your new option pricer should include the following files:

- Random.h
- Random.cpp
- PayOff.h
- PayOff.cpp
- DoubleDigital.h
- DoubleDigital.cpp
- SimpleMC.h
- SimpleMC.cpp

- DDMain.cpp

Please use the following parameter and generate Double Digital option price:

$$\begin{aligned} T &= 1 \\ S(0) &= 50 \\ K_1 &= 43 \\ K_2 &= 57 \\ \sigma &= 0.30 \\ r &= 0.05 \end{aligned}$$

Please run the simulation for 200 times to calculate expected double digital option price, and compare it with the European put and call options with strike price $K = 50$ (note all other parameters are the same as the Double Digital option).

Homework Honor Policy: You are allowed to discuss the problems between yourselves, but once you begin writing up your solution, you must do so independently, and cannot show one another any parts of your written solutions.