Q1 .

To Solve this problem , I created two subclasses with payoff as my base class. Below is my PowerOption.h and PowerOption.cpp files where I have declared and defined my PowerCallOption and PowerPutOption classes and Main class where I'm instantiating the objects and calling these different Power Call and Put options.

```
//

#ifndef PowerOption_h
#define PowerOption_h



#include "PayOff2.h"

class PowerCallOption : public PayOff
{
public:
    PowerCallOption(double Strike_, unsigned long Power_);
    virtual double operator()(double Spot) const;
    virtual ~PowerCallOption(){}

private:
    double Strike;
    unsigned long Power;


};

class PowerPutOption : public PayOff
{
public:
    PowerPutOption(double Strike_, unsigned long Power_);
    virtual double operator()(double Spot) const;
    virtual ~PowerPutOption(){}

private:
    double Strike;
    unsigned long Power;
```

Figure 1PowerOption.h

```cpp
//
//  PowerOption.cpp
//  Homework_3
//
//  Created by HIMANSHU KUMAR on 21/10/18.
//  Copyright © 2018 HIMANSHU KUMAR. All rights reserved.
//

#include <stdio.h>
#include"PowerOption.h"
#include<math.h>

PowerCallOption::PowerCallOption(double Strike_, unsigned long Power_): Strike(Strike_),Power(Power_){

}

double PowerCallOption::operator()(double Spot) const
{

    return std::max(pow(Spot,Power) - Strike,0.0);
}


PowerPutOption::PowerPutOption(double Strike_, unsigned long Power_): Strike(Strike_),Power(Power_){

}

double PowerPutOption::operator()(double Spot) const
{

    return std::max(Strike - pow(Spot,Power),0.0);
}
```

Figure 2PowerOption.cpp

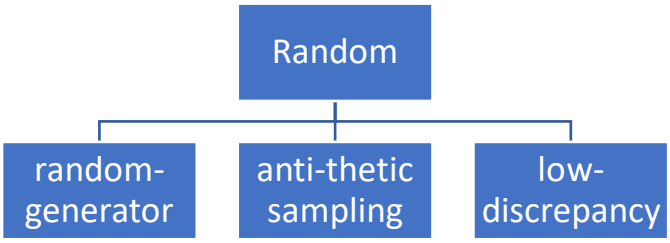
```
cout << "\nNumber of paths: \n";
cin >> NumberOfPath;


PayOffDoubleDigital thePayOff(Low,Up); // prepare payoff object
PowerCallOption theCallPayOff(Strike,Power);// prepare Call Power option object
PowerPutOption thePutPayOff(Strike,Power); // prepare Call Power option object

double CallPower = SimpleMonteCarlo2(theCallPayOff,
                                     Expiry,
                                     Spot,
                                     Vol,
                                     r,
                                     NumberOfPath);

double PutPower = SimpleMonteCarlo2(thePutPayOff,
                                    Expiry,
                                    Spot,
                                    Vol,
                                    r,
                                    NumberOfPath);


double result = SimpleMonteCarlo2(thePayOff,
                                    Expiry,
                                    Spot,
                                    Vol,
                                    r,
                                    NumberOfPath);


cout << "\n the price for double digital with low barrier = "<<Low << " and up barrier = "<<Up<<" is "<<
cout << "\n the price for Power Call with Strike = "<<Strike << " and Power = "<< Power<<" is "<< CallPo
```

Figure 3Main.cpp

Q2.

1.) For evil boss's demands for Std error and convergence table. We can create a base statistic class and instead of calculating the mean values in the simple-main file, we will gather the data from simple-main.cpp and pass on the data to these different base classes inherited from statistics class. The variables return can be individual data elements, like dumping only one value or vectors having all the data from each run.

2.) For taking care of Anti-thetic sampling and low -discrepancy simulations. We can create a base random generator class and use it for implementing Anti-thetic , low discrepancy and other random -generator methods. For eg the variable and methods will be methods : GetUniforms, Skip, SetSeed, resetDimensionality and members will be OddEven , NextVariates.



3.) For taking of geometric and arithmetic average Asian Calls and puts. We need to create a base class with data members such as Delivery Time, Payoff, Number of times and Member Functions as MaxnumberOfCashFlows, PossibleCashFlows etc.