Final Report

**Exercise 8.6**

Since an American knock-in option is a path dependent option, we need to consider the situation that whether or not the option has knocked-in. The basic idea is still using the binomial tree, while we should modify our tree class so that it will fit our specific demands.

In this new tree, we need to store four variables at each nodes. The first one is the price of the stock at that time, the second one is the price of the option if it has been activated before this time, the third one is the price of the option if it hasn’t been activated before this time, and the fourth one is a variable which indicate whether the option is activated at this very time point.

|  |
| --- |
| Stock price |
| Option price(activated) |
| Option price(not-activated) |
| 1/0(activate/not activate now) |

The reason why we need two kinds of option prices is we don’t know whether it has been activated or not. So when we discounting along the three’s end to the root, once we detect that this option is activated we need to use the activated option price, on the other side, if we detect that this option is not activated (the stock price is still under the barrier in terms of the up and in option), we need to use the not-activated option price.

What’s more, note that the option price at each node is calculated from the next step by discounting the prices, however for this path dependent option, we can’t simply discounted from the next step’s prices. We should have a rule for discounting in order to get the option price at this node:

1. Option price(activated) is discounted from the next step’s option prices(activated) and compared it to the value of option if it is executed right now.
2. Option price(not-activated) is discounted by using the activated price of the next steps node if the option is activated at the next node or the not-activated price if the option is not activated at the next node.

Note that once the option is activated at this node, it will only be discounted by using its own activated option price. These rules will guarantee we get the right price considering whether it will be activated or not.

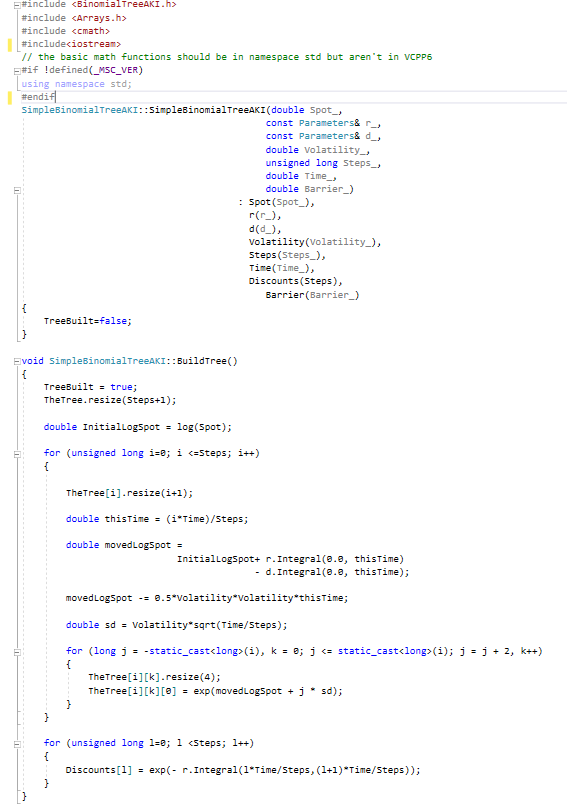
The getting price algorithm is like this:

1. Produce the four variables at each node of the last step, where the activated option price is the payoff of the option and the not-activated option price is all zero.
2. Get these four variables at each node of the second last step, the option price is discounted following the rules above.
3. Repeat the second step of this algorithm until we get the four variables of the option at time 0 and we take the option price(not-activated) as the final result.

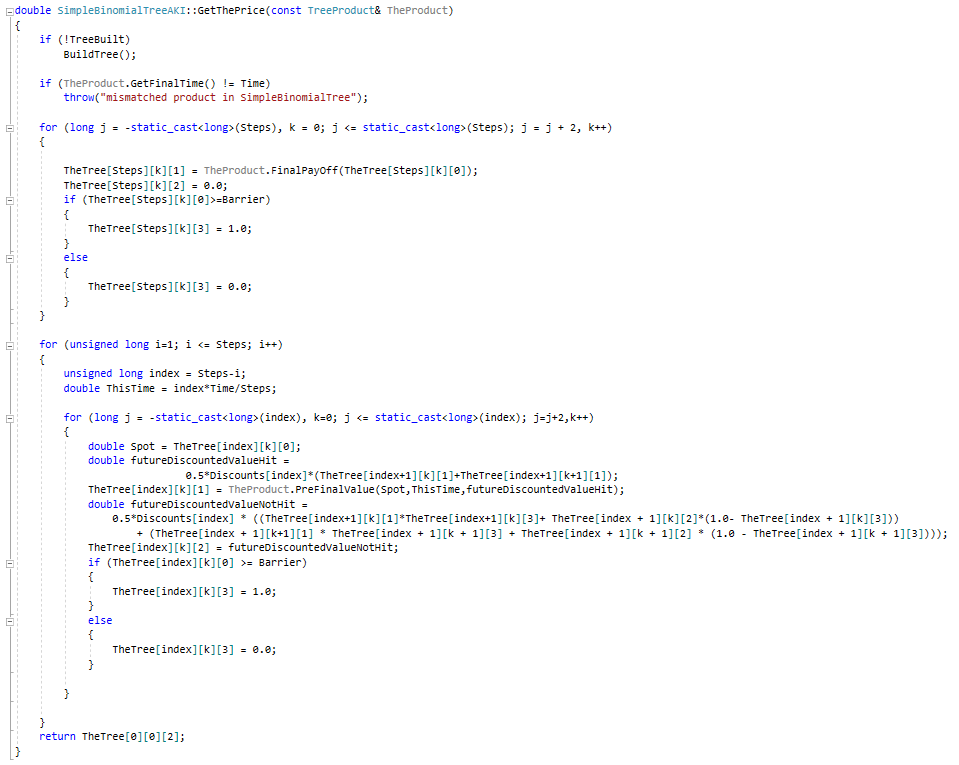
As for the code, we need to rewrite the “BinomialTree” class to let it compatible for the American knock-in option. So we construct a “BinomialTreeAKI” for doing this job. And we take the American up and in option for an example.

In the header file, we need to add another parameter which is the barrier and we need to modify the tree structure, we need to get the a vector of a vector of a vector which refers to the steps, nodes and those four variables.

For the source file, we should modify the existing algorithm:

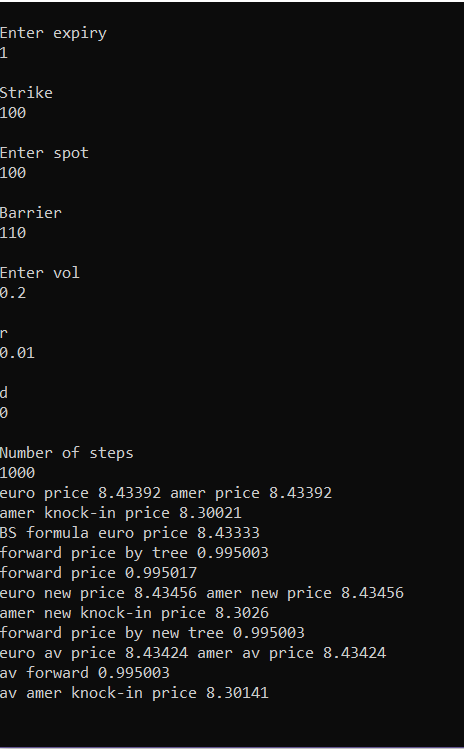


In the “BiuldTree()” method, all we have done is change the thing of pair to vector.



In the “GetThePrice()” method, The most important modification here is the calculation of the third variable which is the option price(not-activated), here we use the 1/0 variable as an indicator function and get the price like this:

After a few changes on the main function, we get the result like this:



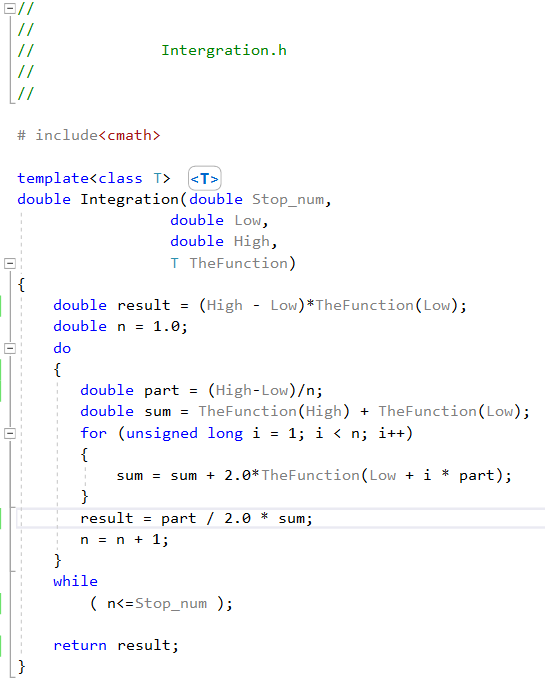
As we can see in the result, the American knock-in option price is less than the American vanilla option, which is caused by the knock-in rules and barrier itself.

**Exercise 9.2**

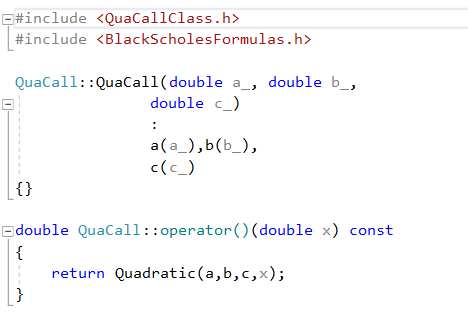
For a numerical integration routine, I use the trapezium rule and the rule is:

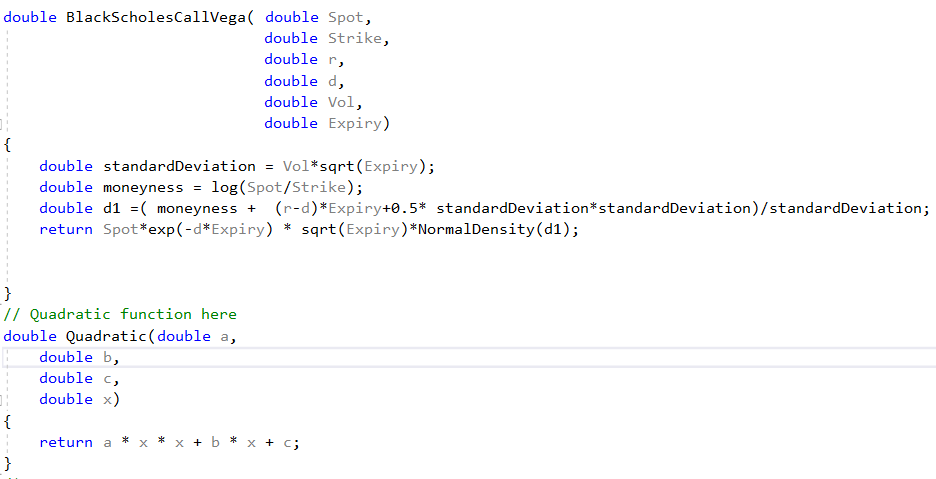
Where and .

So we write a integral file that has almost same structure of the bisection function, while the specific work they do is not same. According to this trapezium rule, the integration function is like this:

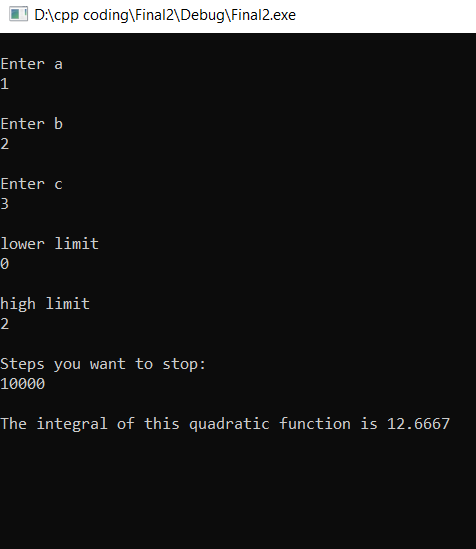


We set a stop number to indicate the largest amount of partition the program will take to calculate this integration. In order to test the code we need some specific function. Here we use the quadratic function which has 3 parameters and we write this function class similar to the B-S formula.





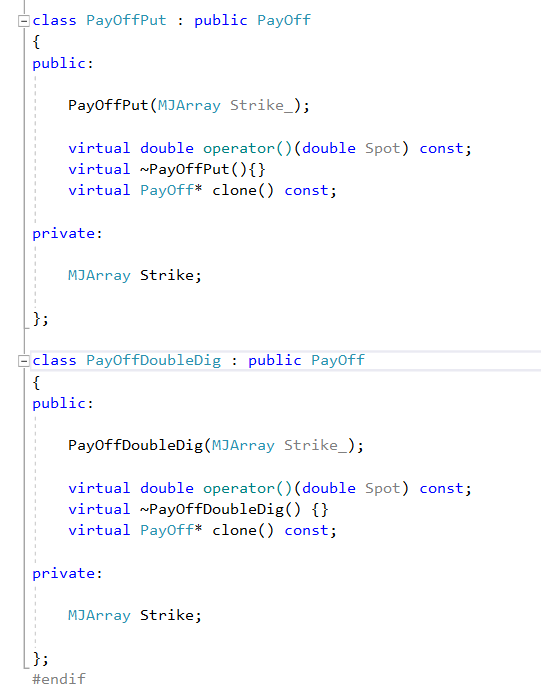
For a quadratic function like: f(x) = a\*x^2+b\*x+c, we give the following result:



**Exercise 10.2**

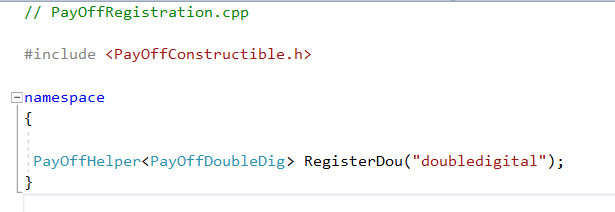
In order to handle the double digital option, or more generally, handle options with multiple parameters, we need to modify the input type of all the class. This time we don’t use the double type, we use an MJArrays type to deal with multiple parameters.

First of all, we need to change all the payoff class to let them take an array as a input:

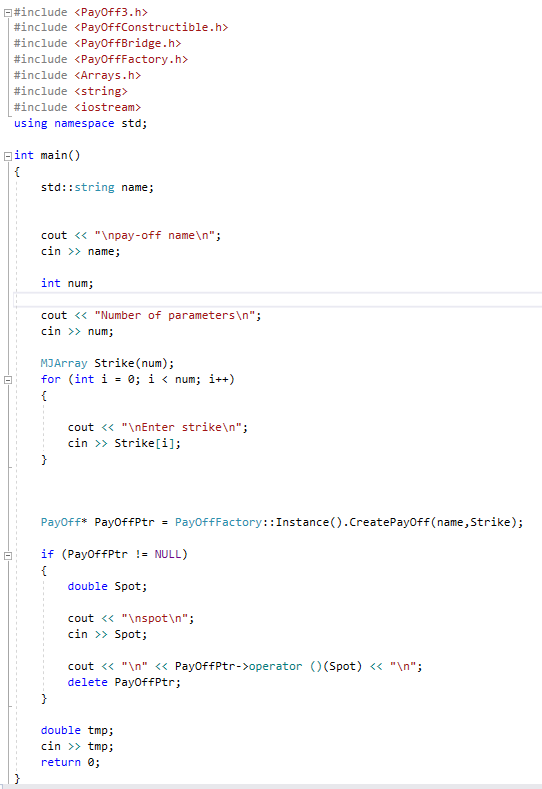


While in the class of the factory, we need also do the same thing. The factory is in fact doing the link job, which means we just need change the type of variable.

What’s more, we need register the double digital option before we start the program:



And the main function is like:



As we run this code, it will ask how many parameters we need and ask the specific amount of each of them. Then according to the name of the payoff, it will give the right answer:

