L3 Homework assignment

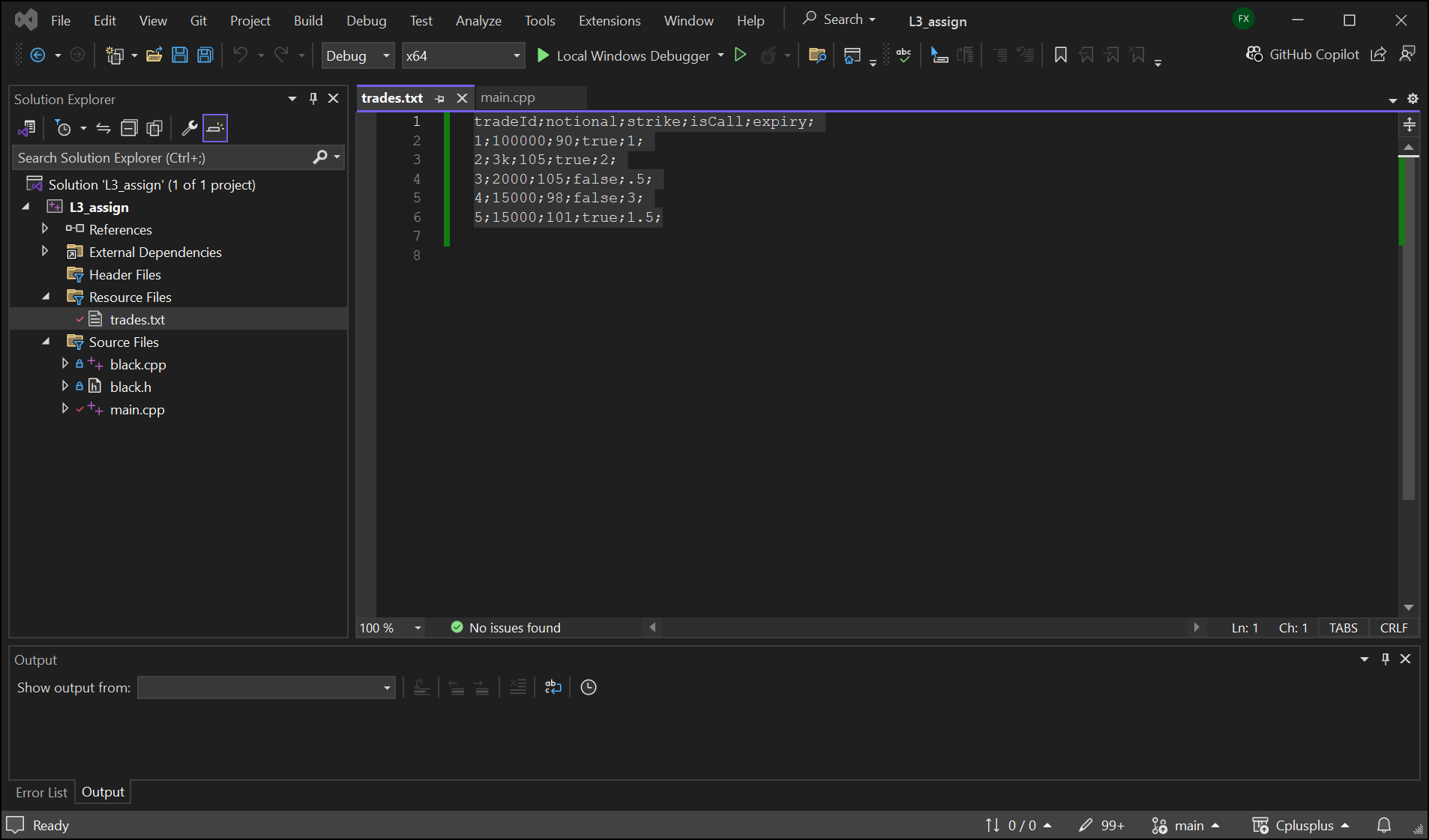
Create a project of console application, compute the vanilla option present value using Black Sholes model.

Requirements:

* Contains multiple files in this project. Create a header file of black.h for declaration of function, and an implementation file Black.cpp for pricing function
* Create a main cpp for main program and include black.h in this file
* Create a function of read line from trade.txt and a function of save result into txt file in main.cpp
* Create a split function to split ling in string into vector of <string>
* In main program:
  + Load the set of pre-defined trades in a txt file and insert into vector of trade struct
  + Create a loop and compute the pv of each trade the vector;
  + Store all trade Pv in an vector, and then output the Pv into a result.txt file
  + Implement the error handling for the computation to check any possible error

Note:

For visual studio user, can create an empty project of windows console application. And then add attached black.h, black.cpp and main.cpp into this project by adding existing item.



For vs code user (mac), just need to update task.json for below setting by amending task.json arg:[] from **${file} into ${workspaceFolder}/\*.cpp**

|  |  |
| --- | --- |
| **${file}** | Currently opened file |
| **${workspaceFolder}** | Path of the folder opened in VS Code |

And then put all cpp in your vs code folder.

Example main.cpp for reference.

#include <iostream>

#include <fstream>

#include <vector>

#include <stdexcept>

#include <string>

#include "black.h"

using namespace std;

struct OptionTrade

{

double notional;

double strike;

double expiry;

bool isCall;

};

void splitString(vector<string>&output, const string& inputLine, const char separator)

{

output.clear();

size\_t start = 0;

size\_t end = inputLine.find(separator);

while (end != std::string::npos) {

output.push\_back(inputLine.substr(start, end - start)); // Extract substring

start = end + 1; // Move past the separator

end = inputLine.find(separator, start); // Find next separator

}

output.push\_back(inputLine.substr(start)); // Add last part of the string

}

void loadTradeFromFile(vector<OptionTrade>& tradesSet, const string& filePath)

{

/\*

load trade data from file

insert into vector

\*/

std::ifstream inputFile(filePath); // Open the file

try {

if (inputFile) {

string line;

if (getline(inputFile, line)) {

cout << "Skipped first row: " << line << endl; // Optional: Print skipped line

}

while (getline(inputFile, line)) { // Read each line

cout << line << endl; // Print the line (or process it)

vector<string> lineOfTrade;

splitString(lineOfTrade, line, ';');

int tradeId = stoi(lineOfTrade[0]);

if (tradeId < 6) {

OptionTrade ot;

ot.expiry = stod(lineOfTrade[4]);

ot.isCall = lineOfTrade[3] == "true"? true: false;

ot.notional = stod(lineOfTrade[1]);

ot.strike = stod(lineOfTrade[2]);

tradesSet.push\_back(ot);

}

}

inputFile.close(); // Close the file

}

else

cout << "file does not exsits" << endl;

} catch (const exception& e) {

cerr << e.what() << endl;

}

}

void writeResultToFile(const vector<double>& result, const string& fileName)

{

ofstream outputFile(fileName); // Create or open the file for writing

if (!outputFile) {

cerr << "Error: Could not create or open file!" << endl;

}

else {

for (auto& re : result) {

outputFile << "trade pv: " << re <<endl;

}

}

outputFile.close();

cout << "Data written to file successfully!" << std::endl;

}

int main () {

cout << "compute option pv task is started." << endl;

vector<OptionTrade> tradesSet;

string file = "../trades.txt";

loadTradeFromFile(tradesSet, file);

vector<double> pvResult;

double spot = 100;

double vol = .2;

double rate = 0.045;

for (auto& trade : tradesSet)

{

/\*

call black sholes model here

\*

\*/

double n = trade.notional;

double pv = BlackScholes(trade.notional, trade.strike , trade.expiry, spot, vol, rate, trade.isCall);

pvResult.push\_back(pv);

}

/\*

save result back into a file

\*/

writeResultToFile(pvResult, "../result.txt");

cout << "compute option pv task is completed." << endl;

return 0;

}