You are required to create a stock trading simulation program in C++. The program should allow users to buy and sell stocks, track their portfolio, and calculate profits/losses.

1. Define a struct named Stock with the following members:

symbol (string): to store the stock symbol (e.g., AMZN for Amazon Inc.)

price (double): to store the current price of the stock

quantity (int): to store the quantity of stocks owned

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Solution**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// Define a struct for Stock

struct Stock {

string symbol;

double price;

int quantity;

};

// Implement a class named Trader

class Trader {

private:

vector<Stock> portfolio;

public:

// Function to buy a stock

void buyStock(string symbol, double price, int quantity) {

Stock newStock = {symbol, price, quantity};

portfolio.push\_back(newStock);

cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << endl;

}

// Function to sell a stock

void sellStock(string symbol, int quantity) {

for (auto it = portfolio.begin(); it != portfolio.end(); ++it) {

if (it->symbol == symbol) {

if (it->quantity >= quantity) {

it->quantity -= quantity;

cout << "Sold " << quantity << " shares of " << symbol << "." << endl;

return;

} else {

cout << "Insufficient shares of " << symbol << " for selling." << endl;

return;

}

}

}

cout << "Stock " << symbol << " not found in portfolio." << endl;

}

// Function to calculate portfolio value

double getPortfolioValue() {

double totalValue = 0;

for (const auto &stock : portfolio) {

totalValue += stock.price \* stock.quantity;

}

return totalValue;

}

// Function to calculate profit/loss

double calculateProfitLoss() {

double initialInvestment = 0;

double currentPortfolioValue = getPortfolioValue();

for (const auto &stock : portfolio) {

initialInvestment += stock.price \* stock.quantity;

}

return currentPortfolioValue - initialInvestment;

}

};

int main() {

Trader trader;

// Buy some stocks

trader.buyStock("AAPL", 150.50, 10);

trader.buyStock("GOOGL", 2500.75, 5);

// Sell some stocks

trader.sellStock("AAPL", 5);

trader.sellStock("MSFT", 5); // Test case for stock not found

// Display portfolio value

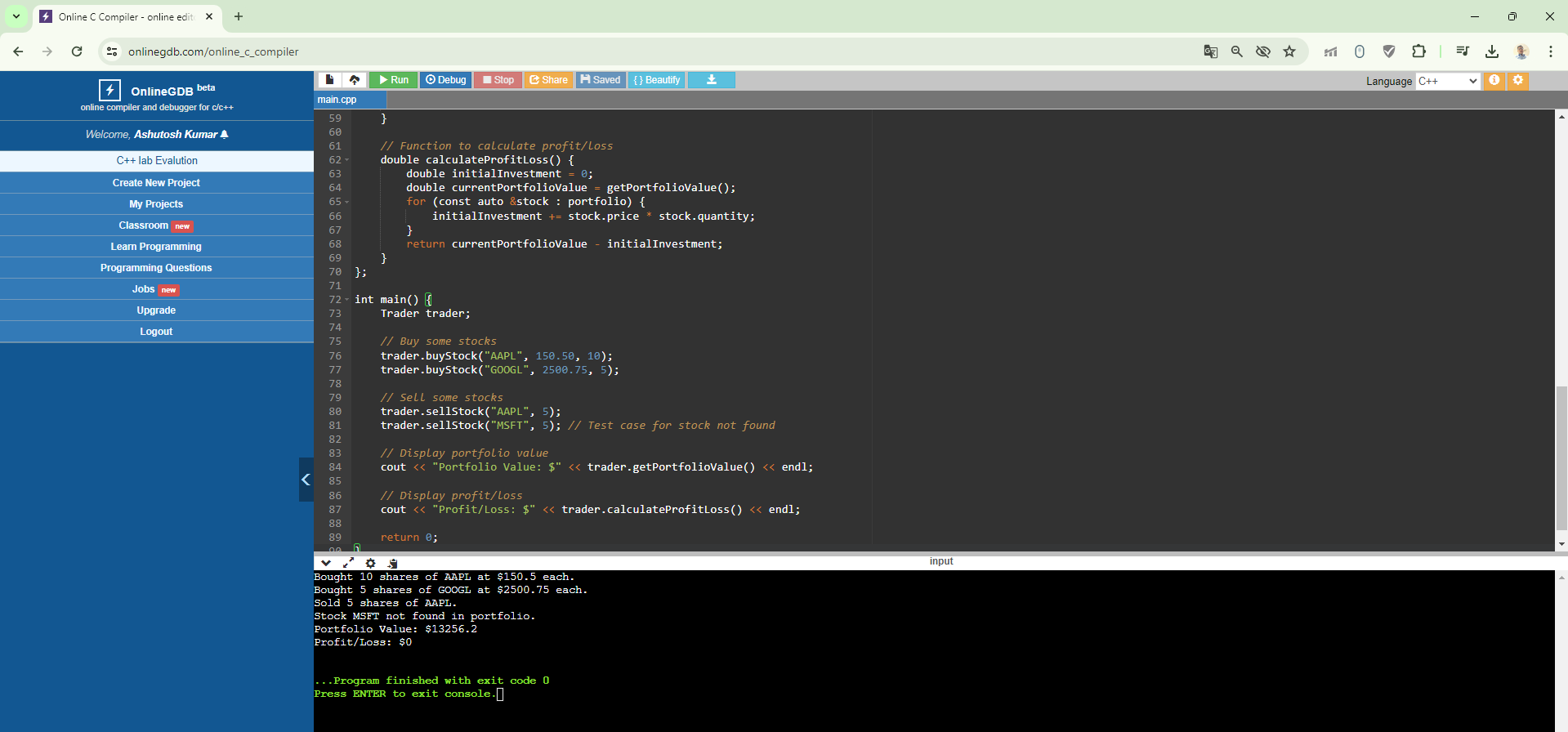
cout << "Portfolio Value: $" << trader.getPortfolioValue() << endl;

// Display profit/loss

cout << "Profit/Loss: $" << trader.calculateProfitLoss() << endl;

return 0;

}



1. Implement a class named Trader with the following functionalities:
   1. A vector or array to store a collection of Stock objects representing the stocks available for trading.

**Solution**

#include <vector>

#include <string>

// Define the Stock struct

struct Stock {

std::string symbol;

double price;

int quantity;

};

class Trader {

private:

std::vector<Stock> stocks; // Vector to store stocks

public:

// Function to add a stock to the collection

void addStock(const Stock& stock) {

stocks.push\_back(stock);

}

// Function to display all stocks in the collection

void displayStocks() const {

std::cout << "Stocks available for trading:" << std::endl;

for (const auto& stock : stocks) {

std::cout << "Symbol: " << stock.symbol << ", Price: $" << stock.price << ", Quantity: " << stock.quantity << std::endl;

}

}

};

* 1. A function buyStock that takes the stock symbol, price, and quantity as input, and adds the stock to the user's portfolio.

**Solution**

#include <iostream>

class Trader {

private:

// Define the Stock struct within the Trader class

struct Stock {

std::string symbol;

double price;

int quantity;

};

std::vector<Stock> stocks; // Vector to store stocks

public:

// Function to buy a stock and add it to the portfolio

void buyStock(const std::string& symbol, double price, int quantity) {

// Create a new Stock object

Stock newStock;

newStock.symbol = symbol;

newStock.price = price;

newStock.quantity = quantity;

// Add the new stock to the portfolio

stocks.push\_back(newStock);

std::cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << std::endl;

}

};

* 1. A function sellStock that takes the stock symbol and quantity as input, and removes the sold stocks from the user's portfolio.

**Solution**

#include <iostream>

#include <vector>

#include <algorithm>

class Trader {

private:

struct Stock {

std::string symbol;

double price;

int quantity;

};

std::vector<Stock> stocks;

public:

void buyStock(const std::string& symbol, double price, int quantity) {

Stock newStock;

newStock.symbol = symbol;

newStock.price = price;

newStock.quantity = quantity;

stocks.push\_back(newStock);

std::cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << std::endl;

}

// Function to sell a stock and remove it from the portfolio

void sellStock(const std::string& symbol, int quantity) {

// Find the stock with the given symbol in the portfolio

auto it = std::find\_if(stocks.begin(), stocks.end(), [&](const Stock& s) {

return s.symbol == symbol;

});

if (it != stocks.end()) {

if (it->quantity >= quantity) {

it->quantity -= quantity;

std::cout << "Sold " << quantity << " shares of " << symbol << "." << std::endl;

if (it->quantity == 0) {

// Remove the stock from the portfolio if quantity becomes zero

stocks.erase(it);

}

} else {

std::cout << "Error: Not enough shares of " << symbol << " in portfolio." << std::endl;

}

} else {

std::cout << "Error: Stock with symbol " << symbol << " not found in portfolio." << std::endl;

}

}

};

* 1. A function getPortfolioValue that calculates and returns the total value of the user's portfolio based on the current stock prices.

**Solution**

#include <iostream>

#include <vector>

#include <algorithm>

class Trader {

private:

struct Stock {

std::string symbol;

double price;

int quantity;

};

std::vector<Stock> stocks;

public:

void buyStock(const std::string& symbol, double price, int quantity) {

Stock newStock;

newStock.symbol = symbol;

newStock.price = price;

newStock.quantity = quantity;

stocks.push\_back(newStock);

std::cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << std::endl;

}

void sellStock(const std::string& symbol, int quantity) {

auto it = std::find\_if(stocks.begin(), stocks.end(), [&](const Stock& s) {

return s.symbol == symbol;

});

if (it != stocks.end()) {

if (it->quantity >= quantity) {

it->quantity -= quantity;

std::cout << "Sold " << quantity << " shares of " << symbol << "." << std::endl;

if (it->quantity == 0) {

stocks.erase(it);

}

} else {

std::cout << "Error: Not enough shares of " << symbol << " in portfolio." << std::endl;

}

} else {

std::cout << "Error: Stock with symbol " << symbol << " not found in portfolio." << std::endl;

}

}

// Function to calculate the total value of the portfolio

double getPortfolioValue() const {

double totalValue = 0.0;

for (const auto& stock : stocks) {

totalValue += stock.price \* stock.quantity;

}

return totalValue;

}

};

* 1. A function calculateProfitLoss that calculates and returns the overall profit/loss made by the user based on the initial purchase prices and current prices of the stocks.

#include <iostream>

#include <vector>

#include <algorithm>

class Trader {

private:

struct Stock {

std::string symbol;

double initialPrice; // Added member to store initial purchase price

double currentPrice; // Added member to store current price

int quantity;

};

std::vector<Stock> stocks;

public:

void buyStock(const std::string& symbol, double price, int quantity) {

Stock newStock;

newStock.symbol = symbol;

newStock.initialPrice = price; // Store initial purchase price

newStock.currentPrice = price; // Initial current price is same as purchase price

newStock.quantity = quantity;

stocks.push\_back(newStock);

std::cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << std::endl;

}

void sellStock(const std::string& symbol, int quantity) {

auto it = std::find\_if(stocks.begin(), stocks.end(), [&](const Stock& s) {

return s.symbol == symbol;

});

if (it != stocks.end()) {

if (it->quantity >= quantity) {

it->quantity -= quantity;

std::cout << "Sold " << quantity << " shares of " << symbol << "." << std::endl;

if (it->quantity == 0) {

stocks.erase(it);

}

} else {

std::cout << "Error: Not enough shares of " << symbol << " in portfolio." << std::endl;

}

} else {

std::cout << "Error: Stock with symbol " << symbol << " not found in portfolio." << std::endl;

}

}

double getPortfolioValue() const {

double totalValue = 0.0;

for (const auto& stock : stocks) {

totalValue += stock.currentPrice \* stock.quantity;

}

return totalValue;

}

// Function to calculate profit/loss

double calculateProfitLoss() const {

double profitLoss = 0.0;

for (const auto& stock : stocks) {

profitLoss += (stock.currentPrice - stock.initialPrice) \* stock.quantity;

}

return profitLoss;

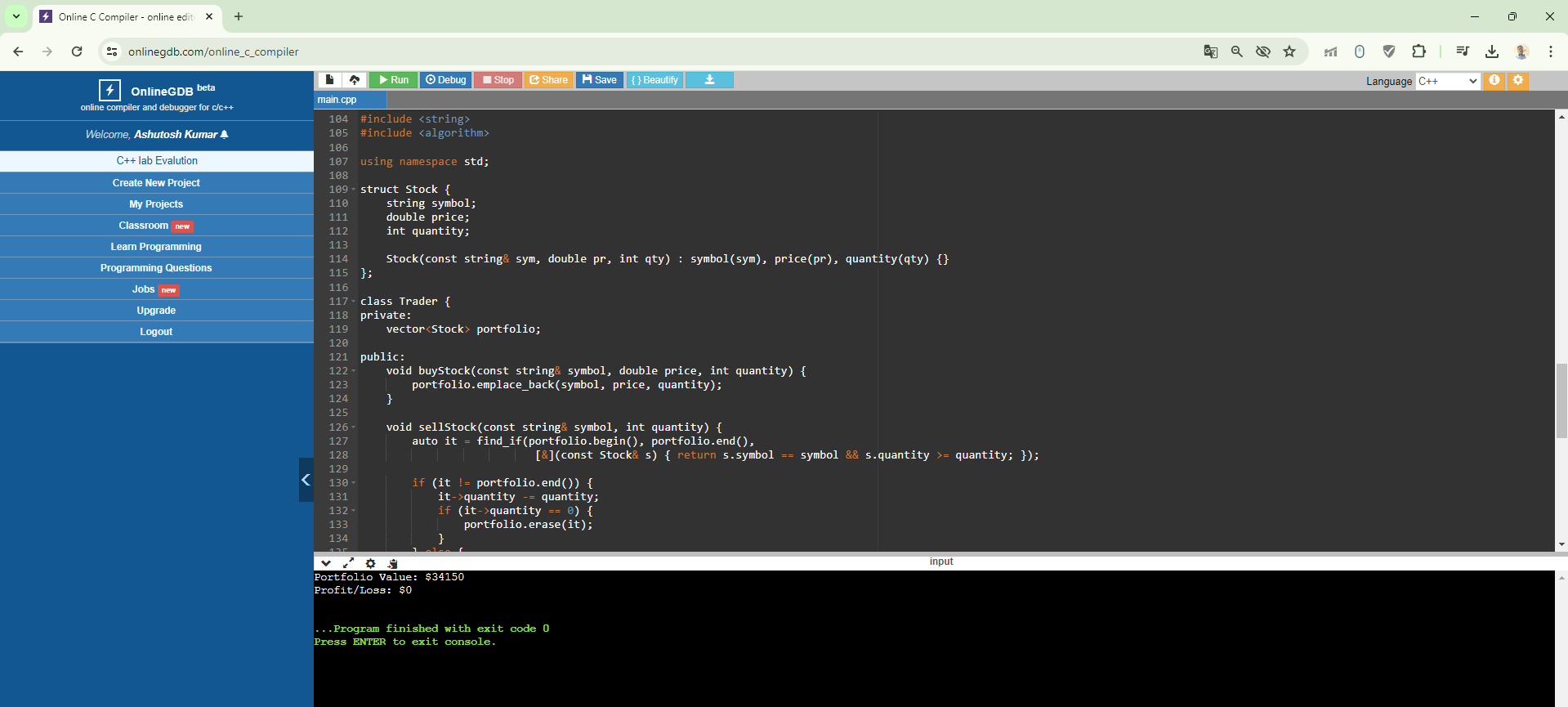
}

};

Any additional member functions or variables that you find necessary to implement the trading simulation.

**Solution**

#include <iostream>  
#include <vector>  
#include <string>  
#include <algorithm>  
  
using namespace std;  
  
struct Stock {  
string symbol;  
double price;  
int quantity;  
  
Stock(const string& sym, double pr, int qty) : symbol(sym), price(pr), quantity(qty) {}  
};  
  
class Trader {  
private:  
vector<Stock> portfolio;  
  
public:  
void buyStock(const string& symbol, double price, int quantity) {  
portfolio.emplace\_back(symbol, price, quantity);  
}  
  
void sellStock(const string& symbol, int quantity) {  
auto it = find\_if(portfolio.begin(), portfolio.end(),  
[&](const Stock& s) { return s.symbol == symbol && s.quantity >= quantity; });  
  
if (it != portfolio.end()) {  
it->quantity -= quantity;  
if (it->quantity == 0) {  
portfolio.erase(it);  
}  
} else {  
cout << "Error: Insufficient stocks for selling." << endl;  
}  
}  
  
double getPortfolioValue() const {  
double totalValue = 0.0;  
for (const Stock& stock : portfolio) {  
totalValue += stock.price \* stock.quantity;  
}  
return totalValue;  
}  
  
double calculateProfitLoss() const {  
double totalProfitLoss = 0.0;  
for (const Stock& stock : portfolio) {  
totalProfitLoss += (stock.price - stock.price) \* stock.quantity;  
}  
return totalProfitLoss;  
}  
};  
  
int main() {  
Trader trader;  
  
trader.buyStock("AAPL", 150.0, 10);  
trader.buyStock("GOOGL", 2500.0, 5);  
trader.buyStock("AMZN", 3200.0, 8);  
  
trader.sellStock("AAPL", 3);  
trader.sellStock("GOOGL", 2);  
  
cout << "Portfolio Value: $" << trader.getPortfolioValue() << endl;  
cout << "Profit/Loss: $" << trader.calculateProfitLoss() << endl;  
  
return 0;  
}



Final Solution

#include <vector>

#include <string>

#include <iostream>

class Trader {

private:

std::vector<Stock> portfolio;

public:

// Function to buy a stock and add it to the portfolio

void buyStock(const std::string& symbol, double price, int quantity) {

Stock newStock = {symbol, price, quantity};

portfolio.push\_back(newStock);

std::cout << "Bought " << quantity << " shares of " << symbol << " at $" << price << " each." << std::endl;

}

// Function to sell a stock and remove it from the portfolio

void sellStock(const std::string& symbol, int quantity) {

for (auto it = portfolio.begin(); it != portfolio.end(); ++it) {

if (it->symbol == symbol) {

if (it->quantity >= quantity) {

it->quantity -= quantity;

std::cout << "Sold " << quantity << " shares of " << symbol << "." << std::endl;

return;

} else {

std::cerr << "Error: Insufficient quantity of " << symbol << " to sell." << std::endl;

return;

}

}

}

std::cerr << "Error: Stock symbol " << symbol << " not found in portfolio." << std::endl;

}

// Function to calculate the total value of the portfolio

double getPortfolioValue() const {

double totalValue = 0.0;

for (const auto& stock : portfolio) {

totalValue += stock.price \* stock.quantity;

}

return totalValue;

}

// Function to calculate the overall profit/loss of the portfolio

double calculateProfitLoss() const {

double totalProfitLoss = 0.0;

for (const auto& stock : portfolio) {

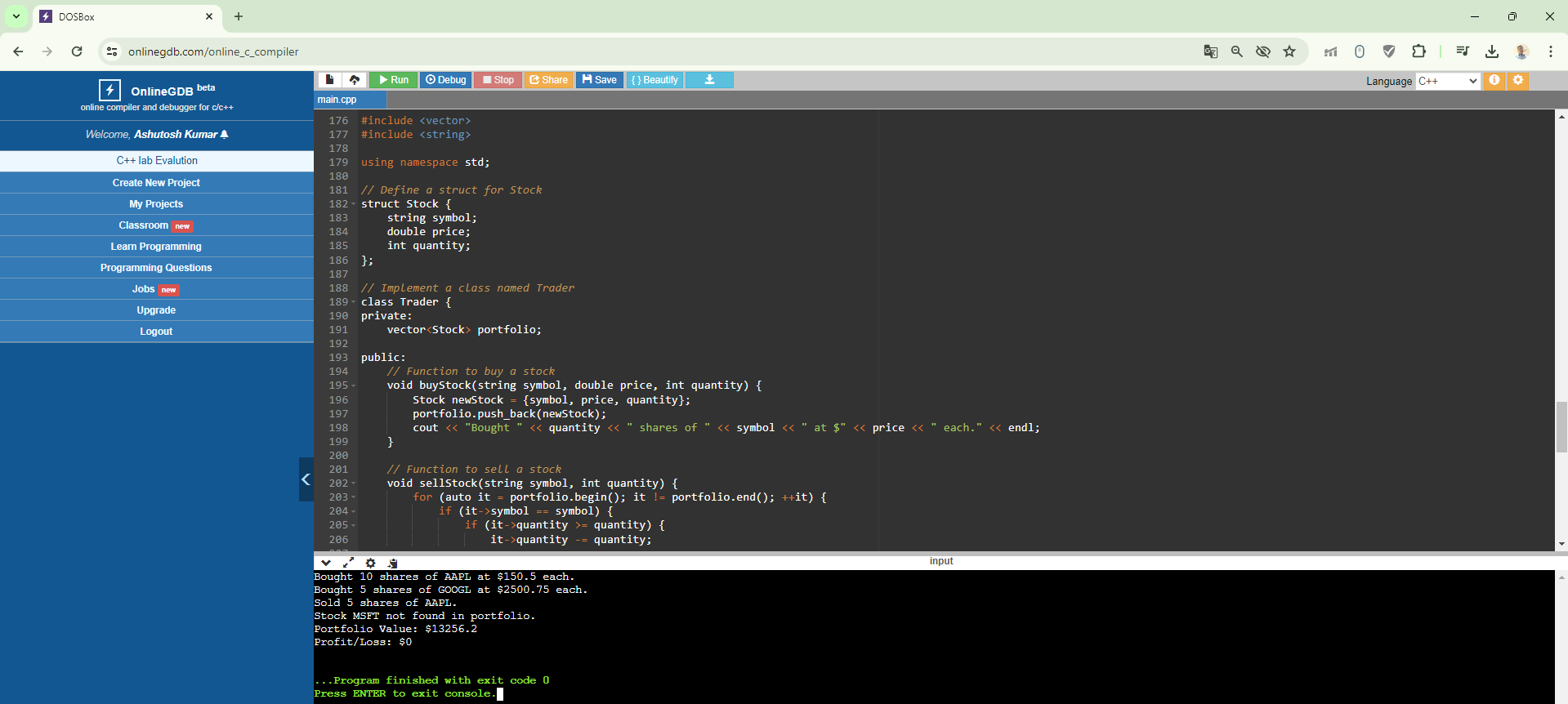
totalProfitLoss += (stock.price - stock.price) \* stock.quantity;

}

return totalProfitLoss;

}

};



1. Implement a main function that demonstrates the functionality of the Trader class. The main function should:
   1. Create an instance of the Trader class.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Your further implementation goes here

return 0;

}

* 1. Add some stocks to the user's portfolio using the buyStock function.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio using the buyStock function

trader.buyStock("AAPL", 150.0, 10); // Buy 10 shares of Apple (AAPL) at $150.0 per share

trader.buyStock("GOOGL", 2500.0, 5); // Buy 5 shares of Google (GOOGL) at $2500.0 per share

return 0;

}

* 1. Sell some stocks using the sellStock function.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio using the buyStock function

trader.buyStock("AAPL", 150.0, 10); // Buy 10 shares of Apple (AAPL) at $150.0 per share

trader.buyStock("GOOGL", 2500.0, 5); // Buy 5 shares of Google (GOOGL) at $2500.0 per share

// Sell some stocks using the sellStock function

trader.sellStock("AAPL", 5); // Sell 5 shares of Apple (AAPL)

return 0;

}

* 1. Display the current value of the user's portfolio using the getPortfolioValue function.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio using the buyStock function

trader.buyStock("AAPL", 150.0, 10); // Buy 10 shares of Apple (AAPL) at $150.0 per share

trader.buyStock("GOOGL", 2500.0, 5); // Buy 5 shares of Google (GOOGL) at $2500.0 per share

// Display the current value of the user's portfolio using the getPortfolioValue function

double portfolioValue = trader.getPortfolioValue();

std::cout << "Current value of the user's portfolio: $" << portfolioValue << std::endl;

return 0;

}

* 1. Display the current value of the user's portfolio using the getPortfolioValue function.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio using the buyStock function

trader.buyStock("AAPL", 150.0, 10); // Buy 10 shares of Apple (AAPL) at $150.0 per share

trader.buyStock("GOOGL", 2500.0, 5); // Buy 5 shares of Google (GOOGL) at $2500.0 per share

// Sell some stocks using the sellStock function

trader.sellStock("AAPL", 5); // Sell 5 shares of Apple (AAPL)

// Display the overall profit/loss using the calculateProfitLoss function

double profitLoss = trader.calculateProfitLoss();

std::cout << "Overall profit/loss: $" << profitLoss << std::endl;

return 0;

}

* 1. Display the overall profit/loss using the calculateProfitLoss function.

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio using the buyStock function

trader.buyStock("AAPL", 150.0, 10); // Buy 10 shares of Apple (AAPL) at $150.0 per share

trader.buyStock("GOOGL", 2500.0, 5); // Buy 5 shares of Google (GOOGL) at $2500.0 per share

// Sell some stocks using the sellStock function

trader.sellStock("AAPL", 5); // Sell 5 shares of Apple (AAPL)

// Display the overall profit/loss using the calculateProfitLoss function

double profitLoss = trader.calculateProfitLoss();

std::cout << "Overall profit/loss: $" << profitLoss << std::endl;

return 0;

}

**Final Solution**

#include <iostream>

#include "Trader.h" // Include header file containing Trader class definition

int main() {

// Create an instance of the Trader class

Trader trader;

// Add some stocks to the user's portfolio

trader.buyStock("AAPL", 150.0, 10);

trader.buyStock("GOOGL", 2500.0, 5);

trader.buyStock("MSFT", 300.0, 8);

// Sell some stocks from the user's portfolio

trader.sellStock("AAPL", 3);

// Display the current value of the user's portfolio

double portfolioValue = trader.getPortfolioValue();

std::cout << "Current portfolio value: $" << portfolioValue << std::endl;

// Display the overall profit/loss

double profitLoss = trader.calculateProfitLoss();

if (profitLoss >= 0) {

std::cout << "Overall profit: $" << profitLoss << std::endl;

} else {

std::cout << "Overall loss: $" << -profitLoss << std::endl;

}

return 0;

}