```
// WRITTEN BY VILJAMAS SIMSONAS
// Prints candlestick data
void MerkelMain::printCandleData()
{
    // Enables virtual terminal processing if program runs on
Windows
    #ifdef _WIN32

        SetConsoleOutputCP(CP_UTF8);
        setvbuf(stdout, nullptr, _IOFBF, 1000);
        DWORD consoleMode;
        GetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
&consoleMode);
        consoleMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
        SetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
consoleMode);

    #endif

    // Get user input
    std::string input = "";

    std::cout << "ENTER << PRODUCT,ASK|BID >> TO PRINT CANDLE DATA"
<< std::endl << std::endl;

    std::getline (std::cin, input);

    // Creates vector of candlestick data
    std::vector<Candlestick> candlesticks =
Candlestick::calculateCandlesticks(orderBook, input);

    std::cout << std::endl;

    // Prints data of each candle in the vector
    for (size_t i = 0; i < 5; ++i) {
        std::cout << "CandleStick " << i + 1 << "\n";
        std::cout << "High  → " << candlesticks[i].high  << "\n";

        std::cout << "Open  → " << candlesticks[i].open  << "\n";
        std::cout << "Close → " << candlesticks[i].close << "\n";
        std::cout << "Low   → " << candlesticks[i].low   << "\n";
        std::cout << "Time  → " << candlesticks[i].date  << "\n";
```

```
        std::cout << "————————————————————————\n";
    }
}

// WRITTEN BY VILJAMAS SIMSONAS
// Draws candlesticks
void MerkelMain::drawCandlesticks()
{
    // Get user input
    std::string input = "";

    std::cout << "ENTER << PRODUCT,ASK|BID >> TO DRAW CANDLESTICKS"
<< std::endl << std::endl;

    std::getline (std::cin, input);

    // Creates vector of all orderBook data based on pair/askOrBid
input
    std::vector<Candlestick> candlestickVector =
Candlestick::calculateCandlesticks(orderBook, input);

    // Creates 10 candlestick object subvector
    std::vector<Candlestick>
candleSubvector(candlestickVector.begin() + 1,
candlestickVector.begin() + 11);

    // Draws candlesticks
    candleDataGraphing candlestickGraphObject;

    candlestickGraphObject.drawCandlesticks(candleSubvector,
input);//build a new graph for this group
}

// WRITTEN BY VILJAMAS SIMSONAS
// Draws volume graph
void MerkelMain::drawVolumeGraph()
{
    // Get user input
    std::string input = "";

    std::cout << "ENTER << PRODUCT,ASK|BID >> TO DRAW VOLUME GRAPH"
<< std::endl << std::endl;
```

```
    std::getline (std::cin, input);

    // Creates vector of all orderBook data based on pair/askOrBid
input
    std::vector<Candlestick> candlestickVector =
Candlestick::calculateCandlesticks(orderBook, input);

    // Creates 10 candlestick object subvector
    std::vector<Candlestick>
candleSubvector(candlestickVector.begin() + 1,
candlestickVector.begin() + 11);

    // Draws volume graph
    candleDataGraphing volumeGraphObject;

    volumeGraphObject.drawVolumeGraph(candleSubvector, input);
}
```

```
// WRITTEN BY VILJAMAS SIMSONAS
// Same as getHighPrice() but no vector input
double OrderBook::getHighestPrice()
{
    double max = orders[0].price;
    for (OrderBookEntry& e : orders)
    {
        if (e.price > max)max = e.price;
    }
    return max;
}

// WRITTEN BY VILJAMAS SIMSONAS
// Same as getLowPrice() but no vector input
double OrderBook::getLowestPrice()
{
    double min = orders[0].price;
    for (OrderBookEntry& e : orders)
    {
        if (e.price < min)min = e.price;
    }
    return min;
```

```
}

// WRITTEN BY VILJAMAS SIMSONAS
// Delete all order book items
void OrderBook::deleteAll()
{
    orders.clear();
}

// WRITTEN BY VILJAMAS SIMSONAS
// Returns order book entry based on index
OrderBookEntry OrderBook::getOrderEntry(int index)
{
    if (index < 0 || index >= orders.size())
    {
        throw std::out_of_range("Bad index. Index out of bounds!");
    }
    return orders[index];
}

// WRITTEN BY VILJAMAS SIMSONAS
// Returns total size of orders
int OrderBook::getOrdersVectorSize()
{
    return orders.size();
}
```

```
// WRITTEN ALL BY VILJAMAS SIMSONAS
#pragma once

#include <string>
#include "OrderBook.h"

// Candlestick class implementation
class Candlestick
{
    public:

        Candlestick(double high, double open,   double close,
                    double low,  double volume, std::string date)
```

```
            : high(high),
              open(open),
              close(close),
              low(low),
              volume(volume),
              date(date)

            {}

        double high;
        double open;
        double close;
        double low;
        double volume;
        std::string date;

        // Creates vector of candle data objects
        static std::vector <Candlestick>
calculateCandlesticks(OrderBook & orderBook,
                                                    const
std::string & inputString);
};

OrderBookType transform(const std::string& str);
```

```
// WRITTEN ALL BY VILJAMAS SIMSONAS
#include <iostream>
#include <string>
#include <vector>
#include "Candlestick.h"
#include "CSVReader.h"
#include "OrderBook.h"
#include "OrderBookEntry.h"

// String to order type transform
OrderBookType transform(const std::string & string)
{
         if (string == "ask")     return OrderBookType::ask;
    else if (string == "bid")     return OrderBookType::bid;
    else if (string == "asksale") return OrderBookType::asksale;
```

```
    else if (string == "bidsale") return OrderBookType::bidsale;
    else
        return OrderBookType::unknown;
}

// Creates vector of candle data objects using orderBook data
std::vector <Candlestick>
Candlestick::calculateCandlesticks(OrderBook & orderBook, const
std::string & inputString)
{
    std::vector <Candlestick> candlestickVector;

    // Tokenise request into product pair and ask or bid request
type
    std::vector <std::string> input =
CSVReader::tokenise(inputString, ',');

    // Sets currency pair/askOrBid based on tokenised input
    std::string pair;
    OrderBookType askOrBid;

    if (input.size() != 2)
    {
        std::cout << "BAD INPUT!" << inputString << std::endl;
    }
    else
    {
        // Product pair
        pair = input[0];

        // Ask or bid request
        askOrBid = transform(input[1]);
    }

    // Builds data of the candle data
    // object and gets pushed into the vector
    OrderBook currentPeriodOrders;

    double firstCandle = true;
    double highPrice   = 0;
    double openPrice   = 0;
    double closePrice  = 0;
```

```
    double lowPrice    = 0;
    double volume      = 0;
    double runTotal    = 0;
    double count       = 0;

    OrderBookEntry compOrder = orderBook.getOrderEntry(0);

    for (int i = 0; i < orderBook.getOrdersVectorSize(); i++)
    {
        OrderBookEntry order = orderBook.getOrderEntry(i);

        if (OrderBookEntry::compareByTimestamp(compOrder, order))
        {
            // Checks if the candle created is the first one
            // to start correct open Price close Price chain
            if (firstCandle)
            {
                closePrice = runTotal / count;
                openPrice  = closePrice;

                // Sets conditional to false
                firstCandle = false;
            }
            else
            {
                closePrice = runTotal / count;
                lowPrice   = currentPeriodOrders.getLowestPrice();
                highPrice  = currentPeriodOrders.getHighestPrice();

                // Appends new candle data object to the returned
vector

candlestickVector.emplace_back(Candlestick(highPrice,

 openPrice,

 closePrice,
                                                        lowPrice,
                                                        volume,

 order.timestamp));
```

```
                // Updates the compared order to the current one
                compOrder = order;

                // Resets all counters
                count     = 0;
                volume    = 0;
                runTotal  = 0;
                openPrice = closePrice;

                // Deletes current period orders used for candle
data object
                currentPeriodOrders.deleteAll();
            }
        }
        // Still same time period
        else
        {
            // Ensures correct currency pair and askOrBid is taken
            if (order.orderType == askOrBid && order.product ==
pair)
            {
                // Increase order price total amount
                runTotal += (order.price * order.amount);

                // Increase total volume
                volume += order.price;

                // Increase total order amount
                count  += order.amount;

                // Add this order to group of orders used for the
candle data object
                currentPeriodOrders.insertOrder(order);
            }
        }
    }
    return candlestickVector;
}
```

```
// WRITTEN ALL BY VILJAMAS SIMSONAS
#pragma once
```

```
#include <vector>
#include "CandleStick.h"

// Candle data graphing class implementation
class candleDataGraphing
{
    public:

        int  candleDataGraphing::getScale(std::string product);

        // Candlestick  drawing section
        void candleDataGraphing::buildCandlestickStrings(double
step, double low, Candlestick& candle, std::array<std::string, 24>&
strings);

        void
candleDataGraphing::drawCandlesticks(std::vector<Candlestick>
candlesticks, std::string productType);

        // Volume graph drawing section
        void candleDataGraphing::buildVolumeGraphStrings(double
step, Candlestick& candle, std::array<std::string, 8>& strings);

        void
candleDataGraphing::drawVolumeGraph(std::vector<Candlestick>
candlesticks, std::string productType);

};
```

```
// WRITTEN ALL BY VILJAMAS SIMSONAS
#ifdef _WIN32
    #include <windows.h>
    #ifdef min
        #undef min
        #undef max
    #endif
#endif
#include <algorithm>
#include <array>
#include <iomanip>
```

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <stdio.h>
#include <vector>
#include "Candlestick.h"
#include "CandleDataGraphing.h"

// Changes scale of graph based on currency pair
int candleDataGraphing::getScale(std::string pair)
{
    int scaleSize = 2;

    if     (pair == "BTC/USDT")
    {
        scaleSize = 2;
    }
    else if(pair == "ETH/USDT")
    {
        scaleSize = 4;
    }
    else if(pair == "DOGE/USDT")
    {
        scaleSize = 6;
    }
    else if(pair == "ETH/BTC")
    {
        scaleSize = 7;
    }
    else if(pair == "DOGE/BTC")
    {
        scaleSize = 8;
    }
    return scaleSize;
}

// Creates array of strings used to draw the candlesticks
void candleDataGraphing::buildCandlestickStrings(double sizeOfStep,
                                                 double low,
                                                 Candlestick&
candlestick,
```

```
 std::array<std::string, 24>& stringArray)
{
    //Enables virtual terminal processing if program runs on Windows
    #ifdef _WIN32

        SetConsoleOutputCP(CP_UTF8);
        setvbuf(stdout, nullptr, _IOFBF, 1000);
        DWORD consoleMode;
        GetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
&consoleMode);
        consoleMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
        SetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
consoleMode);

    #endif

    //Sets colours
    const std::string Red   = "\033[31m";
    const std::string Green = "\033[32m";
    const std::string White = "\033[0m" ;

    // Sets each candles' part positions dependant
    // on the candles low end value and the stepping
    // values of sizeOfStep (y axis scaling values)
    int highPosition  = int((candlestick.high  - low) / sizeOfStep);
    int openPosition  = int((candlestick.open  - low) / sizeOfStep);
    int closePosition = int((candlestick.close - low) / sizeOfStep);
    int lowPosition   = int((candlestick.low   - low) / sizeOfStep);

    // Sets the candle to red or green based on open and close
values
    std::string colour = (candlestick.close < candlestick.open) ?
Red : Green;

    // Adds empty bottom string sections below candle
    for (int i = 0; i < lowPosition; i++)
    {
        stringArray[i].append(colour + "              " + White);
    }

    // Adds bottom wick strings
```

```
    for (int i = lowPosition; i < std::min(openPosition,
closePosition); ++i)
    {
        stringArray[i]     += colour + "       |        " + White;
    }

    // Adds close value body string
    stringArray[std::min(openPosition, closePosition)]     += colour
+ "      ███       " + White;

    // Adds extra body strings if candle's body is bigger than  2
 body  ( > ███ x 2 ) strings in y scale terms
    for (int i = std::min(openPosition, closePosition) + 1; i <
std::max(openPosition, closePosition); i++)
    {
        stringArray[i]     += colour + "      ███       " + White;
    }

    // Adds open value body string if body is longer than 1 height
in y scale terms
    if(openPosition != closePosition)
    {
        stringArray[std::max(openPosition, closePosition)] += colour
+ "      ███       " + White;
    }

    // Adds top wick strings
    for (int i = std::max(openPosition, closePosition) + 1; i <
highPosition + 1; i++)
    {
        stringArray[i]     += colour + "       |        " + White;
    }

    // Adds empty top string sections above candle
    for (int i = highPosition + 1; i < 24; i++)
    {
        stringArray[i]     += colour + "                " + White;
    }
}

// Draws the candlesticks based on the stringArray made on
buildCandlestickStrings()
```

```
void candleDataGraphing::drawCandlesticks(std::vector<Candlestick>
candles,
                                          std::string inputString)
{
    // Sets highest/lowest candle from the candle data vector
    Candlestick topCandle    = *std::max_element(candles.begin(),
candles.end(), [](Candlestick A, Candlestick B){return A.high <
B.high;});
    Candlestick bottomCandle = *std::min_element(candles.begin(),
candles.end(), [](Candlestick A, Candlestick B){return A.low  <
B.low ;});

    // Tokenise console pair/askOrBid input
    std::vector<std::string> input =
CSVReader::tokenise(inputString, ',');

    // Sets currency pair based on tokenised input
    std::string pair;

    if (input.size() != 2)
    {
        std::cout << "BAD INPUT!" << inputString << std::endl;
    }
    else
    {
        pair = input[0];
    }

    // Sets graph scaling based on currency pair
    int scaleSize = getScale(pair);

    // Sets value scaling for candle data based on highest/lowest
candle
    double sizeOfStep = (topCandle.high - bottomCandle.low) / 23;


    // Initialises stringArray where candle strings are stored for
drawing
    std::array<std::string, 24> stringArray;

    // Start of string stream creation, aka drawing from left to
right line by line
```

```
    for(int i = 0; i < stringArray.size(); i++)
    {
        std::stringstream stream;

        stream << std::fixed << std::setprecision(scaleSize) <<
std::setw(5) << bottomCandle.low + i * sizeOfStep;

        stringArray[i] = stream.str() + " ┤";
    }

    std::cout << "" << std::endl;

    // Build strings based on candle data and stepping sized used
for drawing the candlesticks
    for(int i = 0; i < candles.size(); i++)
    {
        buildCandlestickStrings(sizeOfStep, bottomCandle.low,
candles[i], stringArray);
    }

    // Print PRICE string
    std::cout << "" << std::endl << "  PRICE" << std::endl <<
std::endl;

    // Print all the stream strings created with
buildCandlestickStrings()
    for(int i = stringArray.size() - 1; i >= 0; i--)
    {
        std::cout << stringArray[i] << std::endl;
    }

    // Bottom graph line print
    std::cout<<"           └──";

    for(int i = 0; i < candles.size() - 1; i++)
    {
        std::cout << "────────────────";
    }

    // Print currency pair of graph
    std::cout << std::endl << " " << pair;
```

```
        // Print timestamps of each candle
    for(int i = 0; i < candles.size(); i++)
    {
        std::string candleDate = candles[i].date;

        std::vector<std::string> tokenisedDates =
CSVReader::tokenise(candleDate       , ' ');
        std::vector<std::string> candleTime     =
CSVReader::tokenise(tokenisedDates[1], '.');

        std::cout << "      " << candleTime[0];
    }
    std::cout << std::endl << std::endl;
}

// Creates array of strings used to draw the volume graph
void candleDataGraphing::buildVolumeGraphStrings(double sizeOfStep,
Candlestick& candlestick, std::array<std::string, 8>& stringArray) {

    //Enables virtual terminal processing if program runs on Windows
    #ifdef _WIN32

        SetConsoleOutputCP(CP_UTF8);
        setvbuf(stdout, nullptr, _IOFBF, 1000);
        DWORD consoleMode;
        GetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
&consoleMode);
        consoleMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
        SetConsoleMode(GetStdHandle(STD_OUTPUT_HANDLE),
consoleMode);

    #endif

    //Sets colours
    const std::string Red   = "\033[31m";
    const std::string Green = "\033[32m";
    const std::string White = "\033[0m" ;

    // Sets the candle to red or green based on open and close
values
    std::string colour = (candlestick.close < candlestick.open) ?
Red : Green;
```

```
    // Sets top position of volume bar
    int highestPosition = int(candlestick.volume / sizeOfStep);

    // Adds bottom bar part string
    stringArray[0]     += colour + "      ▄▄▄▄      " + White;

    // Adds rest of bar body strings above
    for(int i = 1; i < highestPosition; i++)
    {
        stringArray[i] += colour + "      ▄▄▄▄      " + White;
    }

    // Adds empty space strings above the bar
    for(int i = highestPosition; i < 8; i++)
    {
        stringArray[i] += colour + "                " + White;;
    }
}

// Draws the volume bars based on the stringArray made on
buildVolumeGraphStrings()
void candleDataGraphing::drawVolumeGraph(std::vector<Candlestick>
candles, std::string inputString){

    // Initialises stringArray where volume bar strings are stored
for drawing
    std::array<std::string, 8> stringArray;

    // Sets biggest volume bar from the candle data vector
    Candlestick biggestBar = *std::max_element(candles.begin(),
candles.end(), [](Candlestick A, Candlestick B){return A.volume <
B.volume;});//find candlestick with highest volume to set the scale

    // Sets biggest volume bar value
    double highestVolume = biggestBar.volume;

    // Sets value scaling for candle data based on highest volume
bar
    double sizeOfStep = highestVolume / 7;

    // Adjusts correctly margin in the graphical representation
```

```
    highestVolume += sizeOfStep;

    // Sets currency pair based on tokenised input
    std::string pair;

    std::vector<std::string> input =
CSVReader::tokenise(inputString, ',');
    if (input.size() != 2)
    {
        std::cout << "BAD INPUT!" << inputString << std::endl;
    }
    else
    {
        pair = input[0];
    }

    // Sets graph scaling based on currency pair
    int scaleSize = getScale(pair);

    // Start of string stream creation, aka drawing from left to
right line by line
    for(int i = 0; i < stringArray.size(); i++)
    {
        std::stringstream stream;

        stream << std::fixed << std::setprecision(scaleSize) <<
std::setw(10) << 0 + i * sizeOfStep;

        stringArray[i] = stream.str() + " ┤";
    }

    // Build strings based on candle data and stepping sized used
for drawing the volume graph
    for(int i = 0; i < candles.size(); i++)
    {
        buildVolumeGraphStrings(sizeOfStep, candles[i],
stringArray);
    }

    // Print PRICE string
    std::cout << std::endl << "  VOLUME" << std::endl << std::endl;
```

```
    // Print all the stream strings created with
buildVolumeGraphStrings()
    for(int i = stringArray.size() - 1; i >= 0; i--)
    {
        std::cout << stringArray[i] << std::endl;
    }

    // Bottom graph line print
    std::cout << "            └──";

    for(int i = 0; i < candles.size()-1; i++)
    {
        std::cout << "────────────────";
    }

    // Print currency pair of graph
    std::cout <<std::endl << "  " << pair;

    // Print timestamps of each volume bar
    for(int i = 0; i < candles.size(); i++)
    {
        std::string candleDate = candles[i].date;

        std::vector<std::string> tokenisedDates =
CSVReader::tokenise(candleDate       , ' ');
        std::vector<std::string> candleTime     =
CSVReader::tokenise(tokenisedDates[1], '.');

        std::cout << "      " << candleTime[0];
    }
    std::cout << std::endl << std::endl;
}
```