Introduction

For new engineers that we hire, we would like to have an idea about their engineering craftsmanship and technical skills. That is why we ask applicants to fulfil this programming challenge. It also gives you a flavour of the matter that we face in our work as engineers for a trading company. Please, keep in mind that we want to see your skills and that you will be valued accordingly. Furthermore, it is important that you use C++ and Python, as these are our core languages that we use.

Requirements

- A small report in which you describe at least your approach, problem analysis, and solutions.
- Working source code with the solution in C++.
- Modified Python script.
- Unit tests, if used.

Challenge

The challenge is divided over two steps. The first step involves fixing bugs in a Python script that generates random market data.

In the second step of the challenge a C++ program has to be written that takes the random data generated by the Python script and produces its own output file with buy/sell opportunities.

Challenge Step 1

As a part of this challenge a Python script called market_data_generator.py is included. The script's requirements are listed below, but due to a couple of bugs in the script these requirements are not being met.

The goal is to fix the script so that the requirements are met. Note that a lot of the requirements are already fulfilled by the script.

market_data_generator.py requirements:

- The output file contains timestamps and prices using the following format: *HH:MM:SS.mmm,price*
- The price in the output file should be rounded to the tick size (0.05).
- The minimum possible price is equal to the tick size.
- A price entry should only be added to the output file when the price changed. *Note: When the first price is 0.193 and the next is 0.192 it doesn't count as a price change, because when rounding to the tick size the price didn't change (both round to 0.2). So in this case only one entry should be included in the output file.*
- There can't be more than one entry for the same timestamp.
- The timestamps should be in chronological order.

- You may assume that there are no bugs in the random number functions (rand, rand_int and rand_double).
  We have chosen to include these functions because different Python versions generate different random numbers.
- The first 4 lines of the output file should be:
  09:00:00.000,100
  09:00:00.355,100.2
  09:00:01.188,100.1
  09:00:01.258,100.15
- The last 3 lines of the output file should be:
  17:29:58.502,97.45
  17:29:58.587,97.5
  17:29:59.206,97.65
- The file should have 53,749 entries.

Challenge Step 2

The second part of the challenge is to create a C++ tool to generate trading signals based on the movements of the last traded prices as simulated by the python script.

The first step in this process involves reducing the amount of data. This can be done by translating the raw prices into 'bars'. A bar summarizes the movements of a share price during a fixed amount of time. E.g. a 1 minute bar will record the first price observed since the start of the minute (Open), the highest and lowest price observed in this minute (High, Low) and the last price observed in this minute (Close). This means that all the observed trade prices from 09:00 till 17:30 can be summarized in (8.5 x 60) = 510 minute bars.

In the second step of the process, these bars are used to generate signals for a simple trading strategy where we take a trading decision at the end of each minute bar using the closing price of the bar.

The command line tool in C++ should accept as an input a trades data file in the format: hh:mm:ss.ms,trade price

Input Example:
09:00:00.000,100
09:00:00.355,100.2
09:00:01.188,100.1
09:00:01.258,100.15
.....
17:29:58.502,97.45
17:29:58.587,97.5
17:29:59.206,97.65

The tool will generate a signals.csv file with 1 minute bars in the standard format: Open, High, Low, Close, and also calculate a fast simple moving average (of the Close prices of the bars) (https://www.investopedia.com/terms/s/sma.asp) of 21 bars and a slow moving average of 55 bars.

The trading signal that the tool generates is simply based on the moving averages:

- If the fast moving average crosses above the slow moving average (the fast is strictly higher than the slow) we print BUY in the last column.
- If the fast moving average crosses below the slow moving average (the fast is strictly lower than the slow) we print SELL in the last column.

A sample output:
TIME, OPEN, HIGH, LOW, CLOSE, FASTMA, SLOWMA, SIGNAL
09:00,100.00,101.30,100.00,101.15,0.0000,0.0000,
09:01,101.30,102.10,100.75,101.85,0.0000,0.0000,
09:02,102.05,102.90,101.40,101.75,0.0000,0.0000,
....
....

09:55,99.25,99.80,98.50,98.90,101.4476,101.3809,
09:56,99.00,99.35,96.25,96.80,101.3119,101.2891,
09:57,96.75,96.75,95.05,95.85,101.1286,101.1818,SELL
09:58,96.00,96.15,94.25,96.00,100.8595,101.0836,
09:59,96.05,96.30,94.85,95.40,100.5810,100.9509,

....
....
17:29,95.00,97.65,94.45,97.65,96.0571,95.5855,

As you can see in the above example, initially we don't have values for the moving averages because we need at least 21 periods for the fast and 55 for the slow. At the *09:57* bar we get the first trading signal.

Submission

Best of luck! You can submit your report, C++ code and Python script in a zip file to nick@webbtraders.recruitee.com.