C++ programming test

We need to develop a backtesting system to evaluate the PnL of some mid-frequency strategies.

Strategies trade multiple times during the day by requesting target positions to trade into – target positions are not necessarily reached and depends on the underlying volume of the asset over the execution interval.

We have 2 csv files:

1. Target positions data file (target_positions.csv) containing the positions the strategy would like to hold at different times

security_id	unique security identifier
target_position_arrival_time	time at which the target position is generated and made available for processing in nanosecs since Unix epoch
target_position_end_time	time by which the execution into the target position ends in nanosecs since Unix epoch
target_position	target position to trade into

   Note, the execution interval is the time interval between target_position_arrival_time (rounded up to the next minute) and target_position_end_time (rounded up to the next minute)

2. Minute bar data file (bar_data.csv) containing the price and volume information over 1 minute intervals

security_id	unique security identifier
interval_start_time	start time of the interval in nanosecs since Unix epoch
interval_end_time	end time of the interval in nanosecs since Unix epoch
last_trade_price	last trade price at the interval_end_time
interval_average_spread	average bid-ask spread in the interval
interval_traded_volume	total volume traded in the interval

Simulation assumptions:

1. We have one very simple execution logic but we expect to develop more execution logics later on for the trader to choose from:
   a. Trade-price execution logic: trade at the average trade_price in the execution interval + **X** percent of the average bid-ask spread in the execution interval.
2. We cannot trade more than **Y** percent of the traded volume in any interval – This means that we won't necessarily reach every target position we request (Note, in the case when Y is infinite, we would always reach all target positions)

3. To simplify the framework, we assume that the target position will not change before a prior order expires. In other words, for the same security and two different target positions (1) and (2) it is always true that if target_position_arrival_time (1) < target_position_arrival_time (2) then target_position_end_time (1) < target_position_arrival_time (2).
4. Trading is done with no market impact.
5. All simulations start with no initial positions.

As the backtesting system developer, you don't know the strategies. Your system should allow a strategy to post target positions (in the target_positions.csv file) and simulate the execution using the assumptions above. The system should replicate closely the behavior in production where target positions are published during the day by an independent process that triggers a call to the execution module. The execution module derives the orders, works the orders using any of the available execution algorithms we have, get back the fills and update the positions.

From the trader's point of view, the use case for this system is simple: the trader develops a strategy and then runs it in the backtesting environment. He provides the input (target_positions.csv) and gets back:

1. the fills (security_id, fill_time, fill_quantity, fill_price) – you have to decide what rule to use to timestamp fills and derive the fill_time.
2. the PnL time series

Note that the history we backtest on is huge (let's say it covers years of history and thousands of assets): the trader wants his simulation to run as fast as possible.

There are just two parameters in the backtesting system the trader can change: **X** and **Y** described earlier.

Your job is to design and implement this backtesting system:

- Design: define the exact perimeter of what your system will do, **clarify things if necessary** (there are some little details in the previous text which are voluntarily not perfectly defined), design the architecture, define how the strategies should interact with the system.
- Implement: implement what you have designed in C++, a sample target_positions.csv file (with a limited number of assets) and a bar_data.csv file have been provided to you to test on.

It's up to you to decide what is important or not, you can decide to focus on one thing and ignore another, as long as you describe why. Please make sure you comment your code and describe the architecture decisions you make (for example, what classes and data structures you use, multithreading etc.).

The project should use C++ 11 / 14 / 17. If possible, try to minimize the use of libraries: try to use the standard library only.

You can use gcc and then provide either a Makefile or a CMakeFile, please make sure that you provide a well packaged solution so that we do not waste time trying to open it, compile it, etc.