This test consists of 1 problem.

## Problem 1.

In C++, implement a quoting algorithm for a single instrument for a simple market that only supports order insertion and deletion. You are welcome to use any STL or Boost functionality, but please do not use other third-party libraries. Your code should implement the `InstrumentQuoter` class.

Requirements:

- Orders must not be closer to the theoretical price than the offset at the time they're sent (e.g., `theoreticalPrice - orderPrice >= quoteOffset` for buy orders).
- Orders must be sent with price aligned to the exchange tick (e.g., for `tickWidth = 0.5`, valid order prices would be `..., 9.5, 10.0, 10.5, ...`).
- Orders must not cross the exchange best bid/offer at the time they're sent (e.g., for `tickWidth = 0.5`, `theoreticalPrice = 10.0`, `quoteOffset = 0.5`, and `offerPrice = 9.0`, the buy order should have order price 8.5 rather than 9.5).
- Previous orders on a side must be confirmed removed before a new add request is sent on the same side.

Assumptions:

- You can assume all code wiring is done for you (i.e., you do not need to worry about how the `InstrumentQuoter::On*` methods are called).
- You can assume that all `InstrumentQuoter` methods are called from the same thread (i.e., you do not need to worry about concurrency and synchronisation).
- You can assume order requests will eventually succeed (i.e., you do not need to handle exchange rejections).
- You can ignore the effect of trades (i.e., you do not need to "refill" quotes).
- You can ignore exchange volume rules (i.e., the quote volume provided to you accounts for any exchange volume rules).
- If you prefer, you may — but are not required to — assume that negative or zero order prices are valid for this instrument.

```
// Represents the quoting algorithm for a single instrument.
// THIS IS THE CLASS YOU NEED TO IMPLEMENT.
class InstrumentQuoter
{
public:
  InstrumentQuoter(               // Instantiates the quoter for a single instrument
    std::string const &feedcode, // The exchange identifier of the instrument
    double quoteOffset,          // The minimum desired distance between our theoretical price and each order we send
    uint32_t quoteVolume,        // The desired volume for each order we send
    double tickWidth,            // The distance between valid exchange price levels
    Execution &execution);       // An execution service to add and remove orders (interface described below)

  void OnTheoreticalPrice(       // Notifies the quoter of a new theoretical price for the instrument
    double theoreticalPrice);    // The new theoretical price

  void OnBestBidOffer(           // Notifies the quoter of a new best bid/offer from the exchange
    double bidPrice,             // The best price bid on the exchange
    double offerPrice);          // The best price offer on the exchange

  void OnOrderAddConfirm(        // Notifies the quoter that an "order add" request has been confirmed
    uint32_t id);                // The request identifier of the order

  void OnOrderRemoveConfirm(     // Notifies the quoter that an "order remove" request has been confirmed
    uint32_t id);                // The request identifier of the order
};
```

```
// Provides methods to send order requests to the exchange.
// ASSUME THIS CLASS IS ALREADY DEFINED. DO NOT IMPLEMENT.
class Execution
{
public:
  void requestOrderAdd(           // Requests an order to be added at the exchange
    uint32_t id,                  // A caller-selected request identifier for this order
    std::string const &feedcode,  // The exchange identifier of the instrument
    char orderSide,               // The side of the order ('B' for buy, 'S' for sell)
    double orderPrice,            // The price of the order
    uint32_t orderVolume);        // The volume of the order

  void requestOrderRemove(        // Requests an order to be removed at the exchange
    uint32_t id);                 // The request identifier used in 'requestOrderAdd'
};
```