Matching Engine Design

Features:

1. **MatchingEngine.cpp:**
   Parsing input along with input validation and passing input to the OrderBook.

2. **OrderBook class (OrderBook.hpp/OrderBook.cpp)**
   Member Variables:
   buyBook: Priority queue of OrderUnit pointers ordered in descending order by price.
   sellBook: Priority queue of OrderUnit pointers ordered in ascending order by price.
   buyPriceMap*:* Maps price to OrderUnit pointer present in buyBook.
   sellPriceMap*:* Maps price to OrderUnit pointer present in sellBook.
   idToOrderMap*:* Maps orderid to Order pointer present in SamePriceOrderChain.

   Member Functions:
   outputFillEvent*:*
   Print details of fill events.
   matchOrder*:*
   Check to see if there is a match between SellBook and BuyBook. Worst case time complexity: O(nlog(n)) when the aggressive order creates a TradeEvent on every single possible order in the book, this leads to reorganizing the priority queue(log(n)) n-times, where n is the number of OrderUnits.
   addToBook*:*
   Create new OrderUnit if price level does not exist (Priority Queue Insert : O(log(number of OrderUnits))) or add Order to existing OrderUnit (Doubly Linked List tail append O(1)). buyPriceMap, sellPriceMap and idToOrderMap are updated accordingly.
   deleteOrderUnit:
   If all the Orders of the same price level get cancelled or removed due to a fill event, the OrderUnit is removed from the book (Search the book for the price level O(number of OrderUnits)). buyPriceMap, sellPriceMap and idToOrderMap are updated accordingly.
   CancelOrder:
   Given an orderid, idToOrderMap is used to find the respective order in O(1). This order is then deleted from the doubly linked list in SamePriceOrderChain in O(1). If all the orders in a certain OrderUnit are deleted, then deleteOrderUnit is called.

3. **OrderUnit class (OrderUnit.hpp)**
   Member Variables:
   price: Order price
   side: Order side
   orderChain: SamePriceOrderChain object to store Orders of same price ordered by time of arrival.

   Member Functions:
   getPrice: returns order price
   getSide: returns order side
   insertIntoChain: Calls orderChain function insertIntoChain O(1)
   deleteFromChain: Calls orderChain function deleteFromChain O(1).
   ***deleteChain*:** Calls orderChain function deleteChain O(length of chain).
   getHead: Calls orderChain function getHead O(1).

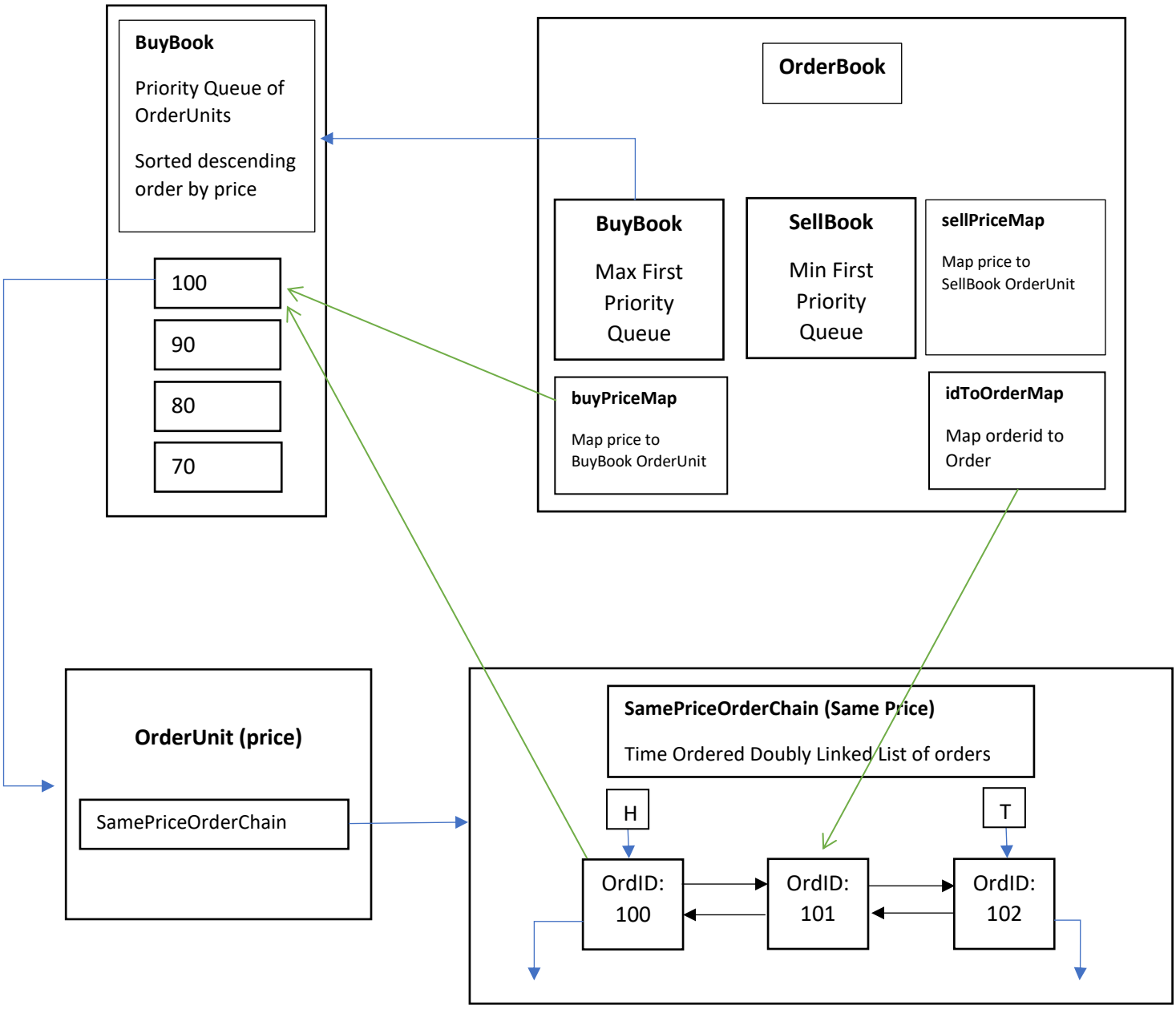
4. **SamePriceOrderChain (SamePriceTimeOrderChain.hpp)**
   Member Variables:
   head: Head of doubly linked list of Orders.
   tail: Tail of doubly linked list of Orders.

   Member Functions:
   getHead: Returns head of doubly linked list.
   insertIntoChain: Insert Order at the tail of SamePriceOrderChain doubly linked list to maintain time order. O(1)
   deleteFromChain: Delete given Order from SamePriceOrderChain doubly linked list. Doubly linked list deletion given pointer is O(1).
   ***deleteChain*:** Delete the entire SamePriceOrderChain, O(length of chain).
   getHead: Returns head of doubly linked list.

5. **Order class (Order.hpp)**
   Member Variables: *msgType, orderid, side, quantity, price*
   Member Functions:
   getNext/getPrev/setNext/setPrev: Getters and setters for next and prev pointers to connect to doubly linked list.
   getQuantity/setQuantity/getPrice/getOrderid: Getter and setters for member variables.
   getOrderUnitPtr/setOrderUnitPtr: Each order has a pointer connecting it back to the OrderUnit to enable O(1) deletion and removal from OrderUnit if the price level is removed.

   Memory allocation has been tested for leaks

Architecture



How to run:

Testcases contains the testcases used to test.

Using makefile

make compile -> compile the program

make test -> run tests

Compiling normally

g++ SamePriceOrderChain.hpp SamePriceOrderChain.cpp OrderUnit.hpp Order.hpp OrderBook.hpp OrderBook.cpp MatchingEngine.cpp -o MatchingEngine

./MatchingEngine