CA2 Report: Vaccine Ordering

The overall design of my system includes classes in separate files to represent single entities of customers, invoices, and orders. There is also a file named ‘fileIO’ which validates and reads the input file. Lastly, my main function can be found in the file named ‘ordering’ which also contains other functions such as adding new customers and orders.

I first started by creating the fileIO to be able to process the input file. The first function named ‘validateArguments’ simply checks that the number of arguments to the program is valid, which is two arguments including the program name. Next, I made a function named ‘isValidLine’ which takes in a line of the input file in the form of a string and checks to see if it is valid, returning a boolean value. I have assumed that a line in the input file cannot be longer than 45 characters as that is the longest possible line according to the coursework specification, this being the new customer line. Therefore, the function first checks to see if the line is greater than 45 characters. Then, it checks that the first character on the line is either a ‘C’, ‘S’ or, an ‘E’. If these characters are not found and the line is not empty (text files can often have an empty line at the bottom of the file), the function returns false. Lastly, the function checks that for each type of line, the parameters provided in the rest of the line are valid.

The last function is the file is named ‘readInput’ and runs once the input file has been validated. With the original argc and argv input variables, it reads the input file and pushes each line to a vector of strings which is then returned.

I then created the customer class, a new customer object is created for each new customer line in the input text file. Each customer has four attributes. A customer number and name provided from the input line stored as an integer and string respectively. An integer containing the total order quantity for the current day. Lastly, a vector of invoices was added later to store each invoice object generated at the end of the day or when an express order is created by a customer. These attributes are private with setter methods for the customer number and the total order quantity and a setter method for the total order quantity as well. There are also two private methods, the first adds an invoice object to the invoice vector and the second prints the details of the last vector in the invoice array. I have made the assumption that the creation of customers should only be stored during one running of the program. Rather than be able to create customers in one input file and then add orders to that customer in a separate running of the file with a different input file. There is also one constructor defined for this class with the customer number and name as parameters.

The next class I created was a simple order class representing a single standard or express order in the system. Each order has four attributes all of which are read from the new order input file line. Three integers are representing the date of the order, the customer number of the order, and the quantity of the order. The type of the order is also stored as a char of either ‘S’ or ‘X’ depending on the order type. There is also a constructor defined which requires all four of these attributes. With just getter methods for these attributes, I have assumed that a customer or a system administrator cannot edit an order once it has been placed. A similar class was also created to handle the creation of invoices that are stored in the customer object. The quantity attribute in an invoice represents the total quantity of vaccines ordered by the customer since the last end-of-day or express order. So when an invoice is printed, the customer's total order quantity is displayed rather than the quantity of their last order.

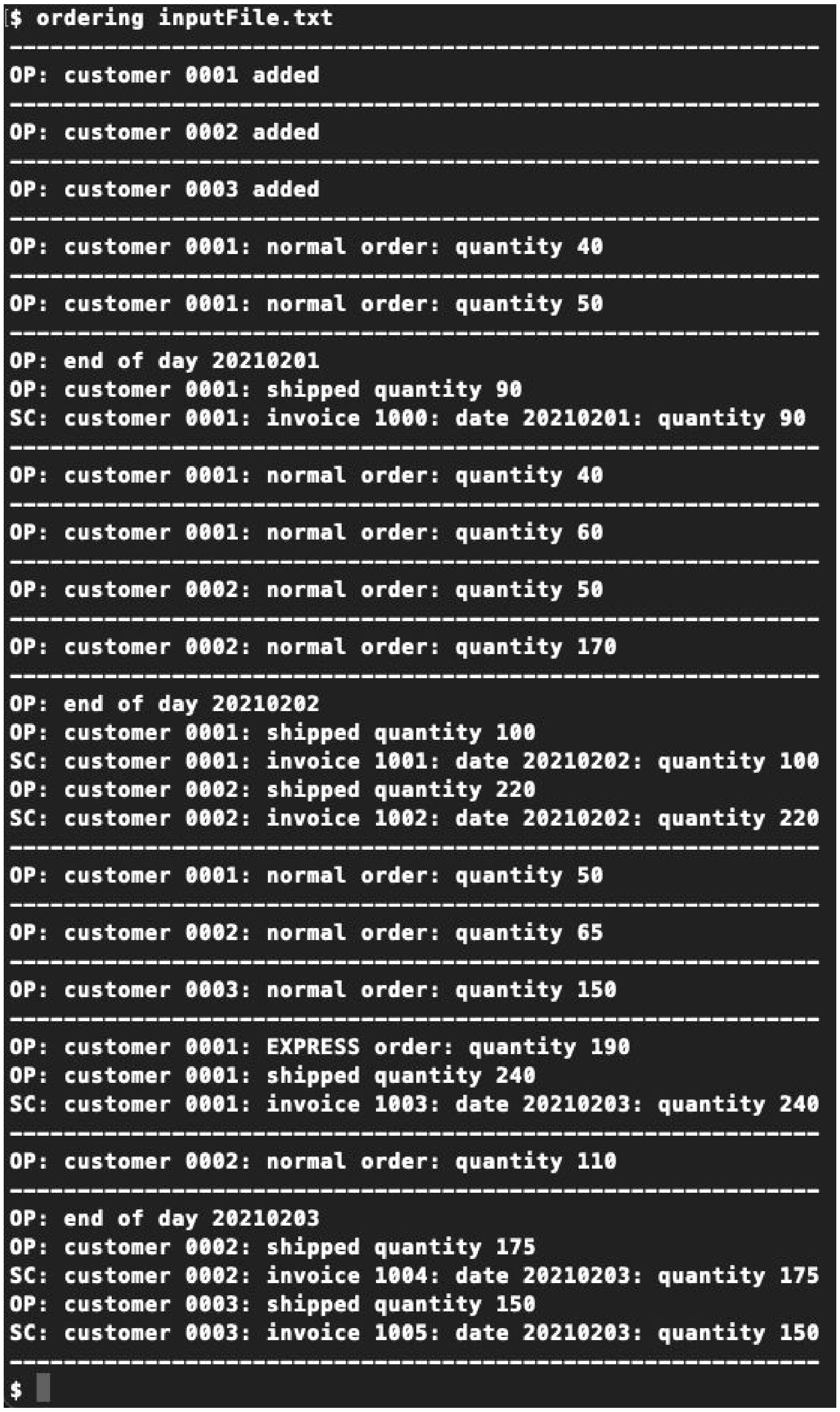
The main function is located in the file named 'ordering'. It first validates the program arguments and validates and reads the input text file. The text file lines, customer objects, and order objects are all stored as vectors. Whilst vectors are relatively expensive to use, there is no mention in the coursework specification that the program needs to run especially quickly which makes sense considering the coursework context. The invoice number is then defined as 1000, each time an invoice is created this is incremented by one. Similar to a previous assumption, this assumes that the invoice number always starts from 1000 for each running of the program. The main function then iterates through each line of the vector 'lines' using a switch statement to determine the type of line ('C', 'S' or 'E'). If the line is adding a new customer, a function named 'addCustomer' is run. This function takes in the line as a string as well as the vector of customers. The customer's number and name are read from the line and a new customer object is created and pushed to the vector 'customers'. The function then prints that a new customer has been added in the specified format.

If the line is adding a new order for a customer, the function 'addOrder' runs which similarly reads the required parameters from the line and pushes an order object to the vector 'orders'. It also increments the customer's total order quantity by iterating through the vector 'customers', finding the customer with the matching number from the new order and using the customer class quantity setter to increment the total order quantity. The new order line is then printed in the required format.

Once the order is added, the order object is checked to see if it is an express order. If it is, the 'sendOrder' function runs. This function iterates through all orders in the vector 'orders' to both delete the orders that are about to be sent and get the date of the orders to be used for the invoice. Due to the fact that all orders are sent at the end of the day, all of a customer's orders in this vector will have the same date. The customer object that made the orders to be sent is then found and a new invoice object is created and added to the customer's vector of invoices. Next, the 'OP' shipping line is printed and the customer's total order quantity is set to 0. Lastly, the invoice number is incremented and the 'SC' invoice is printed using the method 'printInvoice' in the customer class.

If the input text file line starts with an 'E', the 'endDay' function runs. This function first reads the date from the line and prints the end-of-day line. It then iterates through all customers in the vector 'customers' and runs the 'sendOrder' function for each to send off any remaining orders to customers at the end of each day. The function 'sendOrder' checks the total order quantity of the customer is greater than 0 to make sure invoices are not generated for customers if they have not made any orders on that particular day.

Program running using example input file



```
[$ ordering inputFile.txt
------------------------------------------------------------
OP: customer 0001 added
------------------------------------------------------------
OP: customer 0002 added
------------------------------------------------------------
OP: customer 0003 added
------------------------------------------------------------
OP: customer 0001: normal order: quantity 40
------------------------------------------------------------
OP: customer 0001: normal order: quantity 50
------------------------------------------------------------
OP: end of day 20210201
OP: customer 0001: shipped quantity 90
SC: customer 0001: invoice 1000: date 20210201: quantity 90
------------------------------------------------------------
OP: customer 0001: normal order: quantity 40
------------------------------------------------------------
OP: customer 0001: normal order: quantity 60
------------------------------------------------------------
OP: customer 0002: normal order: quantity 50
------------------------------------------------------------
OP: customer 0002: normal order: quantity 170
------------------------------------------------------------
OP: end of day 20210202
OP: customer 0001: shipped quantity 100
SC: customer 0001: invoice 1001: date 20210202: quantity 100
OP: customer 0002: shipped quantity 220
SC: customer 0002: invoice 1002: date 20210202: quantity 220
------------------------------------------------------------
OP: customer 0001: normal order: quantity 50
------------------------------------------------------------
OP: customer 0002: normal order: quantity 65
------------------------------------------------------------
OP: customer 0003: normal order: quantity 150
------------------------------------------------------------
OP: customer 0001: EXPRESS order: quantity 190
OP: customer 0001: shipped quantity 240
SC: customer 0001: invoice 1003: date 20210203: quantity 240
------------------------------------------------------------
OP: customer 0002: normal order: quantity 110
------------------------------------------------------------
OP: end of day 20210203
OP: customer 0002: shipped quantity 175
SC: customer 0002: invoice 1004: date 20210203: quantity 175
OP: customer 0003: shipped quantity 150
SC: customer 0003: invoice 1005: date 20210203: quantity 150
------------------------------------------------------------
$
```