Ticket Manager Report

The ticket manager program is intended to help employees manage customer waiting lists for tickets to events that are otherwise currently sold out. In my implementation of this I use a Customer class to represent the customer and their details, and a WaitingList class, to act as a queue that holds the customer objects, and any details associated with a specific queue, such as the name of the event that the queue is associated with.

Customer Class

The customer class is fairly standard, with two string variables to store the customer's name and phone number (string was used due to the fact that it proves the most convenient way to take in and store such a large number when it is not intended for arithmetic functions, as is the case with a the phone numbers). The functions are simply the standard "get" and "set" functions, with an additional display function to display the customer's details.

WaitingList Class

The waiting list class implements a circular queue to store the customer objects, with the standard queue functions such as add, remove, and checks to see if the queue is empty or full. In addition, however, the class keeps track of how many people are on the waiting list in a variable that is modified each time customers are added or removed from the queue. The class also has copy constructors defined in order to allow the use of the = operator to reset an old queue to a new queue with the name of the new event, and to adhere to the "Rule of 3" in C++ that if either copy constructor type or a destructor is needed, all three should be defined. The class also has an added function that allows the user to find a customer's position in the queue by going through and comparing names till the customer is found, and either returning that position, or -1 if the end of the queue is reached without finding the customer. There is also a display function that will print out the name of an event/waiting list and the number of people on the list.

main()

The main function will call methods to display a welcome and the events list, and take in the user's choice of list. From there it will enter a menu loop that allows the user to select an option from the waiting list sub menu, or go back to the events list to select another list or exit. When the user selects an option from the waiting list sub menu it will enter a switch statement with 6 cases. Cases 1- 4 are sub-menu options and call one of 4 methods; add a customer to the queue, remove a customer from the queue, check a customer's position in the queue, and reset a waiting list to a new event respectively. The 5^{th} case is when the user has opted to go back to the events list and simply displays the list and gets their new event choice (or choice to exit the program). The 6^{th} is a default case where any failure of input will therefore print an error message and get a new menu choice from the user. The menu will exit the menu loop and finish the program when the user has selected 0 at the waiting list sub menu following by 0 at the events list menu. All user's menu choice inputs are validated before being accepted.

Main, Menu Option Functions

The 4 menu option functions manage the processes of adding/removing customers, checking their position, and resetting a queue by calling the member functions of the WaitingList class, and communicating with the user by displaying feedback messages and calling further helper methods to

get input from the user. There is also a function to insert some data into the waiting lists to save time when trying to test.

Testing

I employed a testing strategy that involved going through and testing the different paths and cases for each main waiting list menu option, with particular focus on boundary cases, as they are the most likely to experience errors and failures of logic (e.g. attempting to work outside the boundaries of an array). I did this because the 4 main option methods call and make use of the other methods, and so testing these methods tests the other methods. For example, I tested that the program correctly adds a customer when the queue is either empty or has only one space left, and that it doesn't add them when it is full, which involves testing the isFull() method in the process since it is called and used in within the addCustomer() method. I tested all other levels in between these examples, however, for practicality, I will only include such border case examples for success and non-success. Such test results are detailed below including their inputs and outputs. All tests yielded the expected result.

main() / Menu Navigation

Input: [Run program]

Result: [Welcome message and list of events with the number of customers on their waiting lists displayed in the format:]

1. Super Junior K.R.Y. - Phonograph Tour
Waiting List: 10 people
2. Harry Potter and the Cursed Child
Waiting List: 0 people
(etc…)

[Asks user to enter number of their selection]

Input: *1*

Result: ---------- *MAIN MENU* ----------
1. Add a customer to the waiting list
2. Remove a customer from the front of the queue
3. Check customer's position in the list
4. Reset and begin a new event waiting list

Please enter the number of your choice or 0 to go back:

Input: *0*

Result: [Displays events list and asks for input again as above]

Input: *0*

Result: [Program ends and exits.]

addCustomer() – Full Queue

Input: [Select the add customer option from the sub menu.]

Result: *Sorry, there are no spaces left on this waiting list. Please try again later.*

addCustomer() – Empty Queue

Input: [Select the add customer option from the sub menu.]

Result: *Please enter the customer's details to add them to the waiting list:*
Surname:

Input: *Morris*

Result: *Phone Number:*

Input: *12345678900*

Result: *Customer successfully added to the back of the queue.*

addCustomer() – 1 Less Than Full Queue

Input: [Select the add customer option from the sub menu.]

Result: *Please enter the customer's details to add them to the waiting list:*
Surname:

Input: *Morris*

Result: *Phone Number:*

Input: *12345678900*

Result: *Customer successfully added to the back of the queue.*
[Investigation shows they were correctly added to the back of the queue]

removeCustomer() – Full Queue

(Customer fields numbered 1 – 10 for ease of understanding which customer/position is removed)

Input: [Select the remove customer option from the sub menu.]

Result: *Customer successfully removed to receive tickets:*
Name: 1
Phone Number: 1
[Customer 1 was the front of the queue so removed the correct customer]

removeCustomer() – 1 in Queue

Input: [Select the remove customer option from the sub menu.]

Result: *Customer successfully removed to receive tickets:*
Name: 10
Phone Number: 10
[Customers 1 – 9 were previously removed, so customer 10 was the only one in the queue and was correctly removed]

removeCustomer() – Empty Queue

Input: [Select the remove customer option from the sub menu.]

Result: *Sorry, it looks like the list was already empty and no one is waiting on tickets.*

checkPosition() – Name 1st in Queue

(Full queue with customer fields all labelled 1 – 10 to indicate customer position in the queue e.g. front customer name and phone number are both "1")

Input: [Select the check customer position option from the sub menu]

Result: *Please enter the customer's surname:*

Input: *1*

Result: *The customer is currently at position 1 in the list.*

checkPosition() – Name Last in Queue

(Full queue with customer fields all labelled 1 – 10 to indicate customer position in the queue e.g. front customer name and phone number are both "1")

Input: [Select the check customer position option from the sub menu]

Result: *Please enter the customer's surname:*

Input: *10*

Result: *The customer is currently at position 10 in the list.*

checkPosition() – Name Never in Queue

(Full queue with customer fields all labelled 1 – 10 to indicate customer position in the queue e.g. front customer name and phone number are both "1")

Input: [Select the check customer position option from the sub menu]

Result: *Please enter the customer's surname:*

Input: *NotInQueue*

Result: *No customer with that name was found on the waiting list. Select 3 from the menu to try again.*

checkPosition() – Name Previously in Queue

(Full queue was labelled 1 – 10 as above, then 1 removed so it contains customers 2- 10, with customer 1 still in the 10th space in the circular array but should not be detected as being in the queue anymore)

Input: [Select the check customer position option from the sub menu]

Result: *Please enter the customer's surname:*

Input: *1*

Result: *No customer with that name was found on the waiting list. Select 3 from the menu to try again.*

resetQueue() – Event Slot Not Empty

Input: [Select reset event option from the sub menu]

Result: *Please enter the name of the new event:*

Input: *Test Event*

Result: *The list was successfully reset and replaced by a new waiting list for the Test event.*

Event: Test Event
[Going back to the events list shows the new list]

resetQueue() – Event Slot Empty

Input: [Select reset event option from the sub menu]

Result: *Please enter the name of the new event:*

Input: *Test Event 2*

Result: *The list was successfully reset and replaced by a new waiting list for the Test Event 2 event.*

Event: Test Event 2
[Going back to the events list shows the new list]