SENG1120 – Assignment 2 – Report

Chi Tai Nguyen – 3444339

Implementation

My vision for the Simulation is that each component will encapsulate the next's functionality and structure. From a user's perspective, the Simulation class is the medium of interaction with other classes. One Bank Branch may exist at any time and can contain many Tellers. The Tellers, in turn, may contain multiple Customers with the help of a Queue structure. Customers are a distinct type of entities outside the Bank-Teller-Queue system, generated from the top level (Simulation) and processed at the bottom level (from a certain Teller's Queue). In short, the classes' hierarchy increases in the order of: Customer, Queue, Teller, Bank Branch and Simulation.

The core of the program is the method teller::update(), which controls a Teller's and all its pertaining Customers' agendas. teller::update() modifies data at the lowest level (Customer's waiting times), dictates the current instance's operations (decide if a Teller will serve the Customer if the former is free and there is at least one in the Queue) and is used by higher-level classes (Bank Branch performs independent updates for all Tellers). With an efficient and robust teller::update(), the next crucial method is bank_branch::allocate_customers_to_queue(). This method will perform checks with the Bank Branch's Tellers, push new Customers into specific Tellers's Queues, and maintain the flow of the program in conjunction with teller::update().

Testing

My main goal for the program's capabilities is to ensure it runs according to the Assignment's instructions. The program runs flawlessly with appropriate parameters:

- Simulation time must be a positive integer. Some valid cases: 1, 10, 25, 100.
- The number of Tellers must be a positive integer. Some valid cases: 1, 2, 3, 10, 25.
- The average time of arrival for new Customers must be a positive integer. Same idea.

Though I have not defined enough Exceptions for each invalid parameter case, the program does not appear to malfunction or produce unexpected results.

Test ID	**Test Description**	**Expected Result**	**Test Result**
1	./Simulation 0 25 2 3	As shown in Assignment Specifications	Same as Expected Result
2	./Simulation 0 25 -2 3	Exception: Bank Branch needs to have at least 1 Teller	Same as Expected Result
3	./Simulation 0 25 2 -3	Exception: Average arrival time must be positive	Program prints a blank Teller list |-3| + 1 times, no simulation work done

4	./Simulation 0 0 2 3	Exception: Simulation time must be positive	Program prints a blank Teller list, no simulation work done
5	./Simulation 0 100000 10 3	Program produces 100,000 simulation steps	Same as Expected Result, albeit time-consuming
6	./Simulation 0 -25 2 -3	Exception caught	Same as test 4

Reflection

Having worked extensively for this Assignment, I find myself better equipped with the tools and knowledge of C++ programming, OOP-wise. Object Oriented Programming is not a novel concept to me but learning C++ – a mostly 'manual', upgraded version of C (hence inheriting the annoying syntax) – and applying OOP principles and methodology to it was very much enlightening. Assignment 2 was a step up in terms of required technicality and precision compared to the first Assignment, prompting me to look up more documentation and consult experienced peers. Throughout the development of the program, my C++ inventory grew with the use of STL templates and classes, efficient data structures, concise modular design and workarounds with a certain object's immutability. Personally, this Assignment benefited my logical thinking the most.

Some of my suggestions for this Assignment's Specification would be:

- Remove the const keyword from const T queue::front(). Returned objects are immutable and would be easier to work with otherwise, especially when I wanted to directly modify objects in the Queue, instead of making mutable copies and replacing the original items one by one.
- Would be interesting if multiple Bank Branches could exist in the Simulation. In such scenario, Customers would be independently generated for each Bank Branch, and all Bank Branches would operate side-by-side.