**Data Structures and Algorithms**

**Project Report**

By:

M.Talal Farrukh - 321881

M.Salman Shaheer – 283669

**Introduction and Description:**

This is a Retail Store Simulation. It shows customers entering the shop, choosing items from aisles and then going to cash out. It makes use of different data structures such as linked lists and stacks. It also makes use of algorithms to move customers and staff and to process different decisions made by the customer.

**Objectives:**

The objective is to show the working of the whole shop using data structures and algorithms. There is the issue of storing products/items on a shelf or in the basket. Then there is an issue of determining walking paths for the customer and the staff. And not to mention, the whole shop itself is a linked list of customers. Using data structures and algorithms, these issues are solved. Below, we will give an idea about the working of data structures and algorithms that are used.

*The customer walks to the desired aisle and puts items in their basket.*

Here, we have an algorithm to determine the walking path of the customer according to the randomly generated aisle choice. Next, we have data structures to store the items in the customer’s basket. The shelf where the items are stored are data structures as well (stacks).

*Once done with one aisle, the customer chooses again whether to shop. If yes, then they repeat the above process again.*

Here if the customer chooses to shop again, a new path is selected according to the customer’s randomly generated aisle choice, hence making use of algorithms.

*Once completely done shopping, they go to the counter to cash out. Once done with that as well, they exit the shop.*

So once the customer is done with their shopping, the algorithm for cashing out is executed. This algorithm makes the customer look for suitable cashier queues and walk towards them and then cash out when it is their turn.

*If the amount of item in a shelf falls below a certain amount, a staff appears to refill that aisle.*

An algorithm is made to monitor the quantity of items on the shelves. If the quantity of items fall below a certain number, the algorithm will generate the staff and use it to refill the shelf that requires restocking. Remember, shelves are data structures as well, the items are taken off the shelf when a customer shops from it and items are put back when the staff restocks them.

**Languages, development environment and libraries used:**

The development environment used for this project is **Visual Studio Community 2019**.

The programming language used is **C++**.

Many libraries were used. For graphics, **SFML** libraries were used. This required linking of those libraries in order to be able to use them.

SFML libraries:

1. Graphics.hpp
2. System.hpp
3. Window.hpp
4. Audio.hpp

Other libraries used:

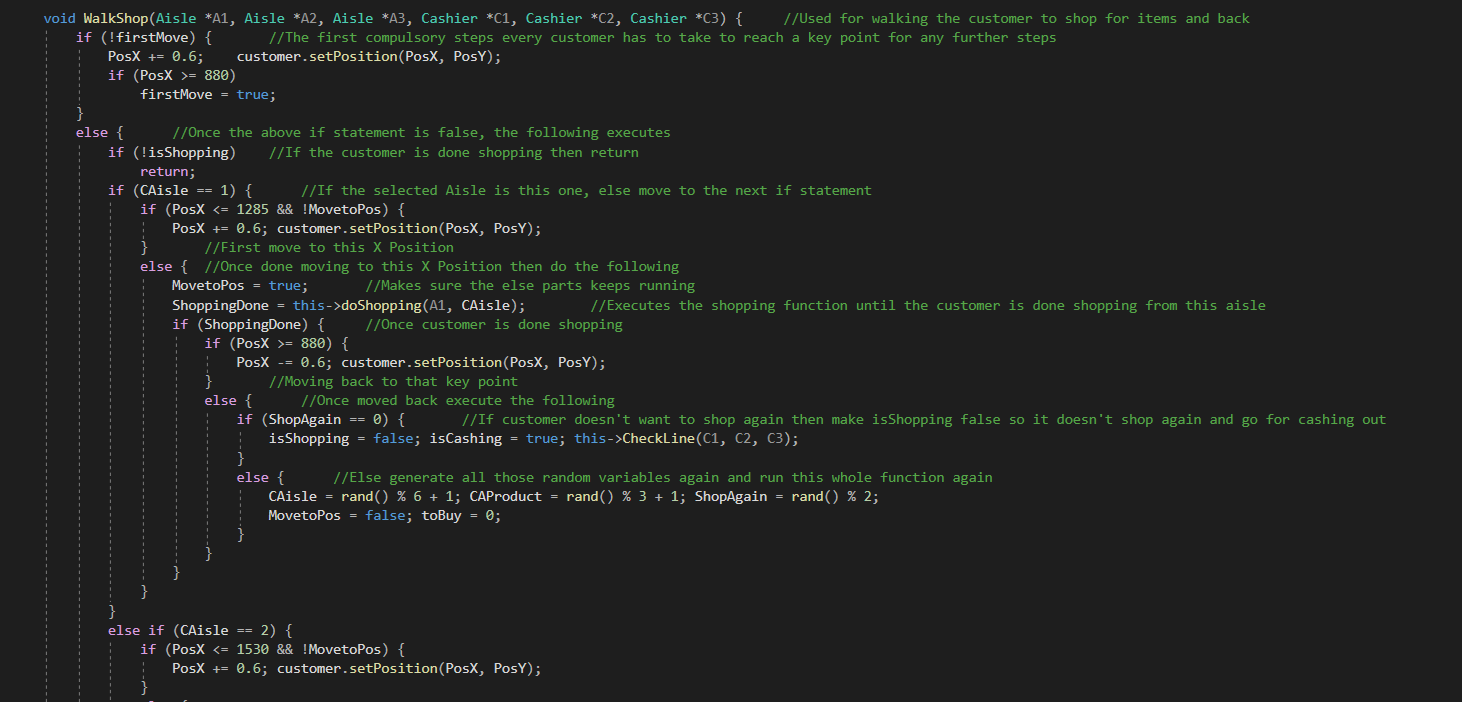
1. string.h
2. stdlib.h
3. time.h

**Code and Algorithm:**

The whole program adds up to 996 lines. So it is not possible to add all the code. However, we will explain and analyze the important algorithms used.

Before we analyze the algorithms, we need to keep in mind that all the algorithms were tailored according to the while loop which keeps running in main. This while loop keeps the graphics window open until the user closes it. Since all functions need to be run in this while loop, the functions were made according to it.

**Shopping Algorithm:**



This algorithm is longer than in the picture. But this part should be enough to explain the whole algorithm.

The key variables here are CAisle, CAProduct and ShopAgain. We can say these variables are the customer’s decisions.

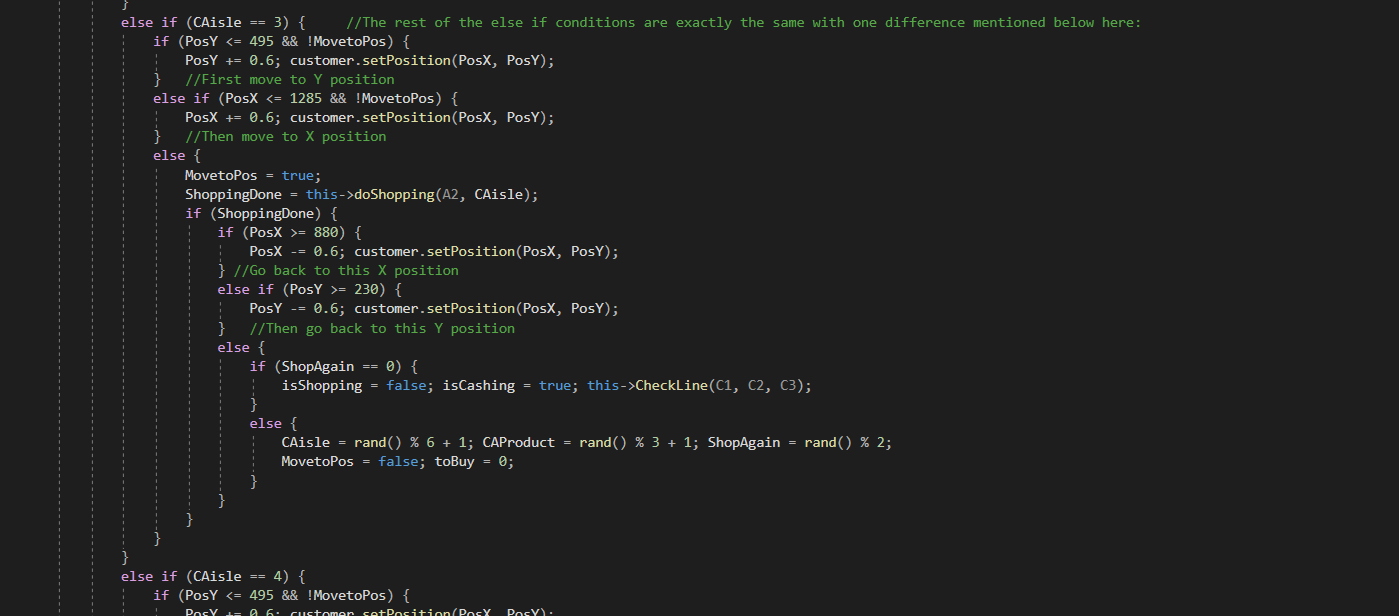
At the start, the customer needs to reach a **key** point. So the customer walks 0.6 steps/frame in the X direction. The Big-O notation here will be O(KeyPointX-CurrentPositionX).

For this algorithm to run any further, the isShopping variable needs to be true. If not, the function returns at the point and doesn’t execute any further. We could say that the Big-O notation here will be O(n).

For Scenario 2 where isShopping is true, the algorithm jumps to the if statement for the selected Aisle. We will assume Aisle 1 is selected (CAisle = 1). Once this aisle is selected, the customer will walk 0.6 steps/frame towards the X direction since this path comes directly comes in its line of sight. Then it needs to put items in their basket. The maximum amounts of items that a customer can pick from the shelf at a time is 3. So the customer picks between 1-3 items. Each item is picked after 3 seconds. Once the customer is done shopping, the customer moves to the **key** point by moving 0.6 steps/frame backwards in the X direction. Here, the customer generates randomly whether they want to shop again or no. If yes, then the whole shopping process is repeated. The Big-O notation in this scenario would be something like this: ShopAgain(O(DestinationX-CurrentPosX)\*2 + O(AmountofProducts\*3)).

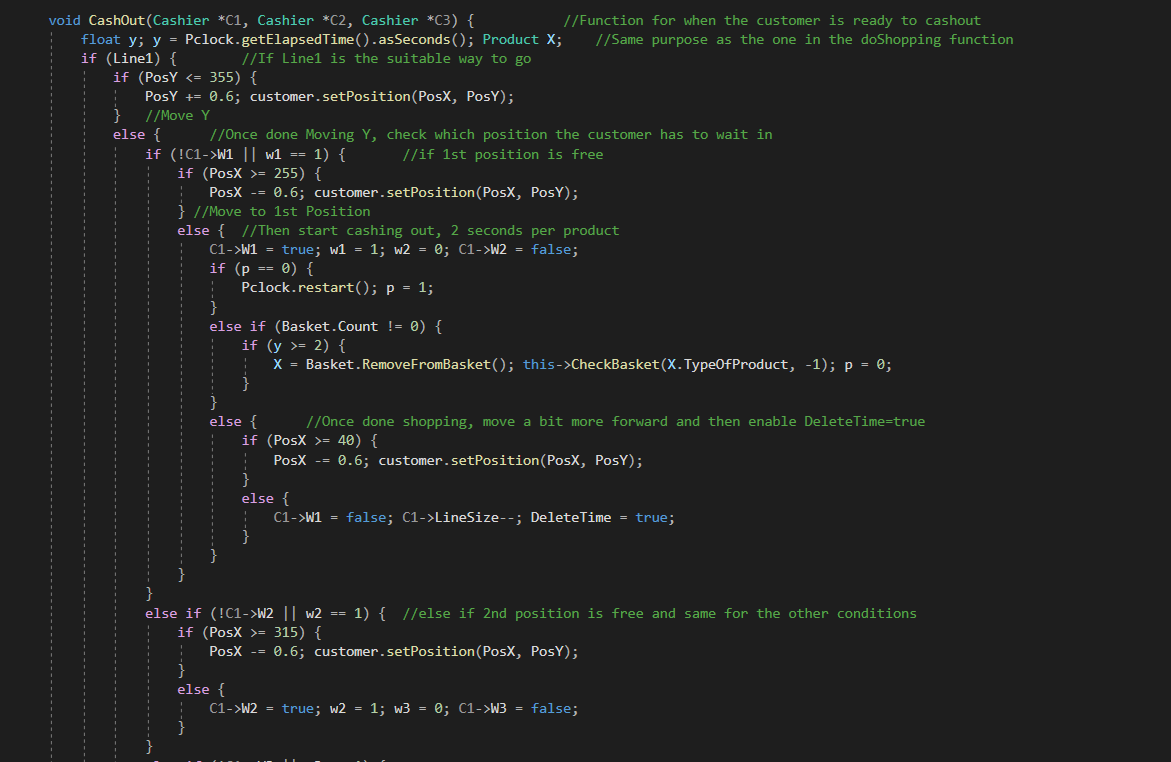
The process will be same for CAisle = 2, the customer needs to only walk a few steps further in the X direction.

Let us come to Scenario 3 now.



This is the scenario where CAisle is a number between 3 and 6. The whole process is the same except one part, the customer has to move in the Y direction back and forth as well. In Scenario 2, the customer walks 0.6X steps first before shopping. In this case, the customer will first walk 0.6 steps/frame in the Y direction and once it reaches the desired point, it’ll walk 0.6 steps/frame in the X direction. The 2nd change will be when the customer is done shopping. After walking back 0.6X steps back, the customer will move 0.6 steps/frame back up in the Y direction to reach the **key** point. This will change the Big-O notation slightly. It will be something like this: ShopAgain(O(DestinationY-CurrentPosY)\*2 + O(DestinationX-CurrentPosX)\*2 + O(AmountofProducts\*3)).

**Cash out algorithm:**



The Cash out algorithm depends on the Shopping Algorithm’s Scenario 2 and 3. These scenarios have a function where the customer checks which of these 3 cashier queues are empty or have less size and then the queue and its path is determined and carried to this algorithm.

In this scenario we will assume the customer has determined queue 1 (Line1). The customer will first walk 0.6 steps/frame down in the Y direction and the 0.6 steps/frame in the X direction. Once in position, the customer will start cashing out. It takes 2 seconds to process 1 product.

In another scenario, the line is already occupied and the customer has to wait in either position 2, 3 or 4.

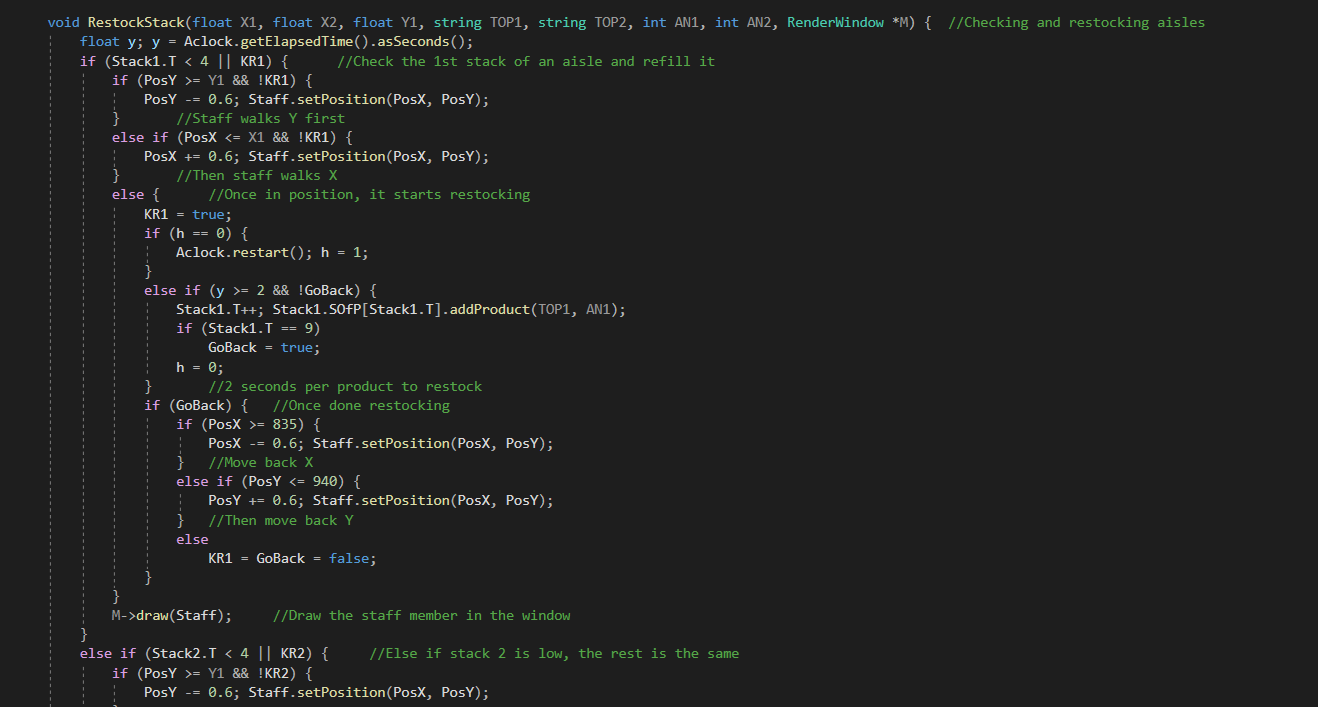
So for the first scenario where the customer immediately goes to cash out, the Big-O notation would be something like this: O(DestinationY-CurrentPosY) + O(DestinationX-CurrentPosX) + O(ProductsinBasket)\*2

In case of waiting, we will represent the above equation with Z, where Z is the Big-O notations for the customers ahead of the current customer.

So: O(Z)\*1,2,3 + O(DestinationY-CurrentPosY) + O(DestinationX-CurrentPosX) + O(ProductsinBasket)\*2

The rest of the lines have the exact same process.

**Restocking Algorithm:**



There are 3 aisles where each aisle is divided into 2 shelves, or stacks as we call it in this program. This algorithm is for one aisle.

The algorithm monitors both the stacks, Stack1 and Stack2. The trigger amount for restocking is 4, if the quantity of items falls below this, the algorithm will execute. Else it will just keep monitoring. The Big-O notation in this case would be O(n).

If the quantity of a stack falls to 4, the staff walks 0.6 steps/frame in the Y direction till the desired Y destination is reached. Then it walks 0.6 steps/frame in the X direction to reach the X destination. Once in position, the staff starts restocking the shelf until it is full. The maximum amount is 10 per shelf. It takes 2 seconds to put 1 item in the shelf. Once done restocking, the staff walks back 0.6X and 0.6Y steps from where it came from. The Big-O notation for this algorithm would be: O(DestinationY-CurrentPosY)\*2 + O(DestinationX-CurrentPosX)\*2 + O(12).

*Since there are 6 products to restock and it takes 2 seconds/product to restock, 6\*2 = 12.*

These 3 algorithms were the most important part of the program. The whole program revolves around these. The rest are normal functions that do not consume much time and resources, they are executed very quickly. Here are some of the honorable mentions of some of those functions:

1. void CreateCustomer() from Shop.h – this is used to add customer to the linked list of the shop.
2. void DeleteCustomer() from Shop.h – this is used to remove customer from the linked list of the shop.
3. void DetectCustomer() and void DetectAisle() from Shop.h – these two functions are used to display the quantity of products when the user hovers over the shelves or a customer.
4. bool doShopping() from Customer.h – this function is used to transfer items from the shelf to the basket.
5. void CheckLine() from Customer.h – this function is used to determine which queue to go to when going for cashing out.
6. Product RemoveFromStack() from Aisle.h – this function removes the items from the shelf when the customer shops.
7. void PutInBasket() from BasketOfCustomer.h – this function puts the items in the customer’s shopping basket.
8. Product RemoveFromBasket() from BasketOfCustomer.h – this function removes products from a customer’s basket at cash out.

**Output:**

Since the simulation is graphical, there is not much to show. Below are the screenshots of some parts of the simulation.









Some points to note:

1. The yellow circle represent the customers and the green represent the staff
2. The blue boxes in the pictures appear when the user hovers over the customer or the brown shelves on the right. The pointer isn’t visible but the screenshots were taken when the pointer was hovered over them.

**Conclusion:**

The program makes use of data structures and algorithms efficiently. The data structures are used to store customers in the shop in the form of linked lists, the basket of a customer is a linked list, the item shelves are stacks and the cashier counters are queues. The algorithms ensure the smooth flow of the simulation without consuming much time and resources. All the objectives have been fulfilled.