Mini Project Documentation

SECJ2013 DATA STRUCTURE AND ALGORITHM

SEMESTER I, SESSION 2020/2021

Topic: Food Ordering System

Lecturer: Dr. Johanna binti Ahmad

Name	Matric No.
CHONG KAI JIE	A19EC0036
DESMOND CHIENG CHEE HONG	A19EC0040
YONG JING XUAN	A19EC0177
CHEONG CHIEN LI	A19EC0186
SEE WEN XIANG	A19EC0206

Section: 06

Programme: Bachelor of Computer Science (Software Engineering)

1.0   Problem Analysis

In this project, we will be developing a simple food ordering system. Before we start to identify the problems that we should solve in order for this project to be successful, we should first understand what a typical food ordering system should do, and how it functions.

In a typical food ordering system, there should be two main user groups, the first group being the customers and the next group being the staff members, who will be preparing the foods ordered by the customers. There are a few basic functionalities that the customers should be able to perform when using the food ordering system.

First and foremost, the customers should be able to see what are the foods offered by the restaurant, from their menu. This will allow users to have a list of available options to pick from, when placing their orders. This brings us to the next basic functionality that a typical food ordering system should have. Customers should be able to place orders on their desired food via the system. This is because after looking through the menu, the customer's next action should naturally be to place an order. Then, during the ordering process, the customers should be able to see the total price of their orders, so that they know how much they should be paying for the list of foods that they have ordered.

On the other hand, the staff members, or specifically the kitchen crew members of the restaurant, would have several basic functionalities that any typical food ordering system should provide. Firstly, the orders placed by the customers should be able to be sent to the kitchen crew members, so that they can keep track of all the on-going orders. The main interface for the kitchen crew users should be to see a list of all uncompleted orders, so that they could work on completing them in sequence. Next, the system should allow the kitchen crew members to view every order in greater details, showing them all the food that should be prepared within a specific order. This means that kitchen crew members should be able to see what they should prepare for each order in order to complete them. Lastly, a kitchen crew member should be able to complete an order whenever they are done with preparing and delivering the foods ordered within that order. Completing an order would mean that it would be removed from their list of on-going orders, so that they would not mistakenly try to complete an already completed order.

Now that we know the basic functionalities that a typical food ordering system should have, we can start to look into the problems that we will have to solve while working on this project. There are a few problems that we will have to solve when developing this project.

Firstly, we should be able to solve the issue of displaying the menu of the restaurant within the system, so that the customers would be able to easily view it. This process can be solved by reading the information of the menu from an external file, and then displaying them within the system in an organised manner, by applying linked-list to a queue data structure.

Next, the customers should have the ability to search through the menu according to food names or their prices. Hence, one of the problems that we will have to solve when developing this project, is to implement searching algorithms that allow the customers to search the menu based on given searching keys. This can be solved by applying searching methods to search through the nodes within the linked-lists to find the ones that matches with the user's searches.

In addition, in order to allow users to place orders on specific foods from the menu, another problem that we will have to solve in this project is to search through the nodes containing the information of the foods, according to a specific identifier for the food. This will allow users to order food by simply entering the unique identifier detail about the food that they wish to order.

Lastly, when the customers wish to complete their orders, we will have to solve the problem of inserting a node of the order to the linked-list of orders that are kept track by the kitchen crew. This means that we have to efficiently insert nodes into a queue of orders so that they can be monitored accurately.

As for the kitchen crew users, we will have to solve problems such as displaying the information of all nodes within the linked-list of orders placed by the customers. Then, we will have to solve the problem of searching through the list of orders in order to identify a specific node to be displayed for the kitchen crew to view. Lastly, we will have to solve the problem of efficiently deleting nodes for the queue of orders whenever the kitchen crew completes an order.

These are the basic problems that we will have to face while developing this project. All of these problems should be solved effectively when the final executable application file is produced, so that we can test its ability to perform its expected functionalities. The success of this project will be defined by the ability of our final application to perform all of the tasks as stated above, by implementing the queue data structure using linked-list concepts.
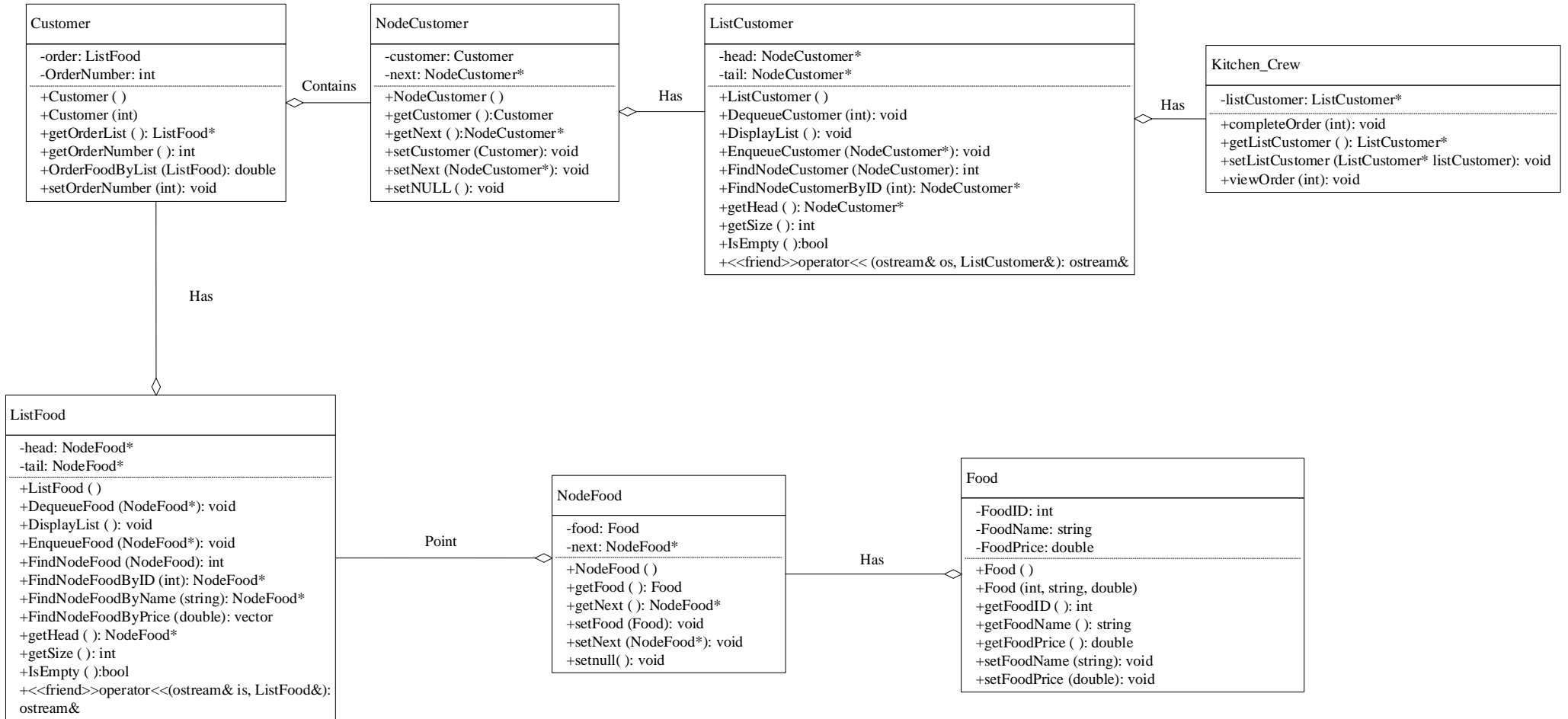
2.0 Class Design

Customer

-order: ListFood
-OrderNumber: int

+Customer ( )
+Customer (int)
+getOrderList ( ): ListFood*
+getOrderNumber ( ): int
+OrderFoodByList (ListFood): double
+setOrderNumber (int): void

NodeCustomer

-customer: Customer
-next: NodeCustomer*

+NodeCustomer ( )
+getCustomer ( ):Customer
+getNext ( ):NodeCustomer*
+setCustomer (Customer): void
+setNext (NodeCustomer*): void
+setNULL ( ): void

ListCustomer

-head: NodeCustomer*
-tail: NodeCustomer*

+ListCustomer ( )
+DequeueCustomer (int): void
+DisplayList ( ): void
+EnqueueCustomer (NodeCustomer*): void
+FindNodeCustomer (NodeCustomer): int
+FindNodeCustomerByID (int): NodeCustomer*
+getHead ( ): NodeCustomer*
+getSize ( ): int
+IsEmpty ( ):bool
+<<friend>>operator<< (ostream& os, ListCustomer&): ostream&

Kitchen_Crew

-listCustomer: ListCustomer*

+completeOrder (int): void
+getListCustomer ( ): ListCustomer*
+setListCustomer (ListCustomer* listCustomer): void
+viewOrder (int): void

Contains

Has

Has

Has

Has

Point

ListFood

-head: NodeFood*
-tail: NodeFood*

+ListFood ( )
+DequeueFood (NodeFood*): void
+DisplayList ( ): void
+EnqueueFood (NodeFood*): void
+FindNodeFood (NodeFood): int
+FindNodeFoodByID (int): NodeFood*
+FindNodeFoodByName (string): NodeFood*
+FindNodeFoodByPrice (double): vector
+getHead ( ): NodeFood*
+getSize ( ): int
+IsEmpty ( ):bool
+<<friend>>operator<<(ostream& is, ListFood&): ostream&

NodeFood

-food: Food
-next: NodeFood*

+NodeFood ( )
+getFood ( ): Food
+getNext ( ): NodeFood*
+setFood (Food): void
+setNext (NodeFood*): void
+setnull( ): void

Food

-FoodID: int
-FoodName: string
-FoodPrice: double

+Food ( )
+Food (int, string, double)
+getFoodID ( ): int
+getFoodName ( ): string
+getFoodPrice ( ): double
+setFoodName (string): void
+setFoodPrice (double): void

Figure 1: UML Class Diagram of Food Ordering System
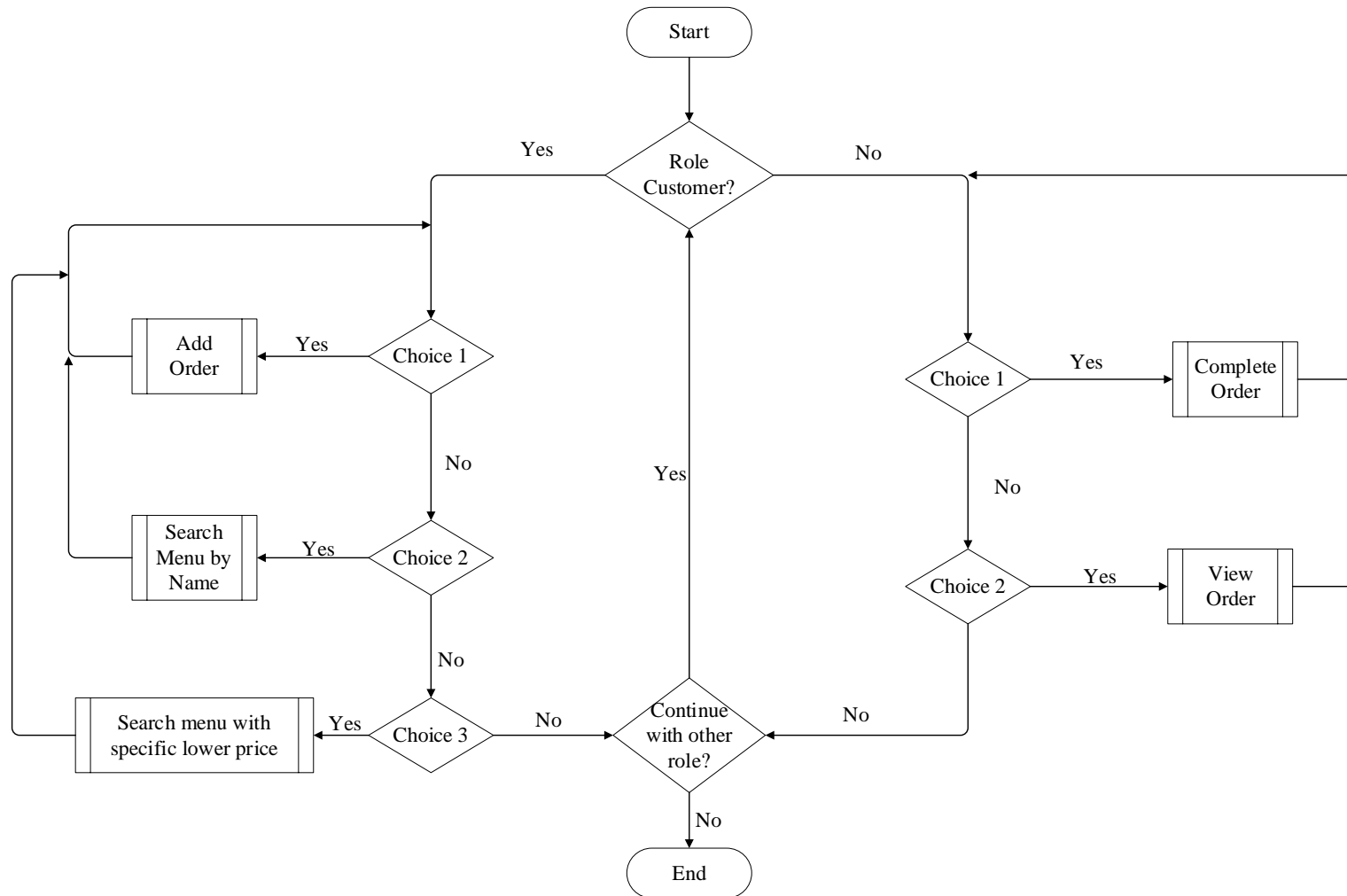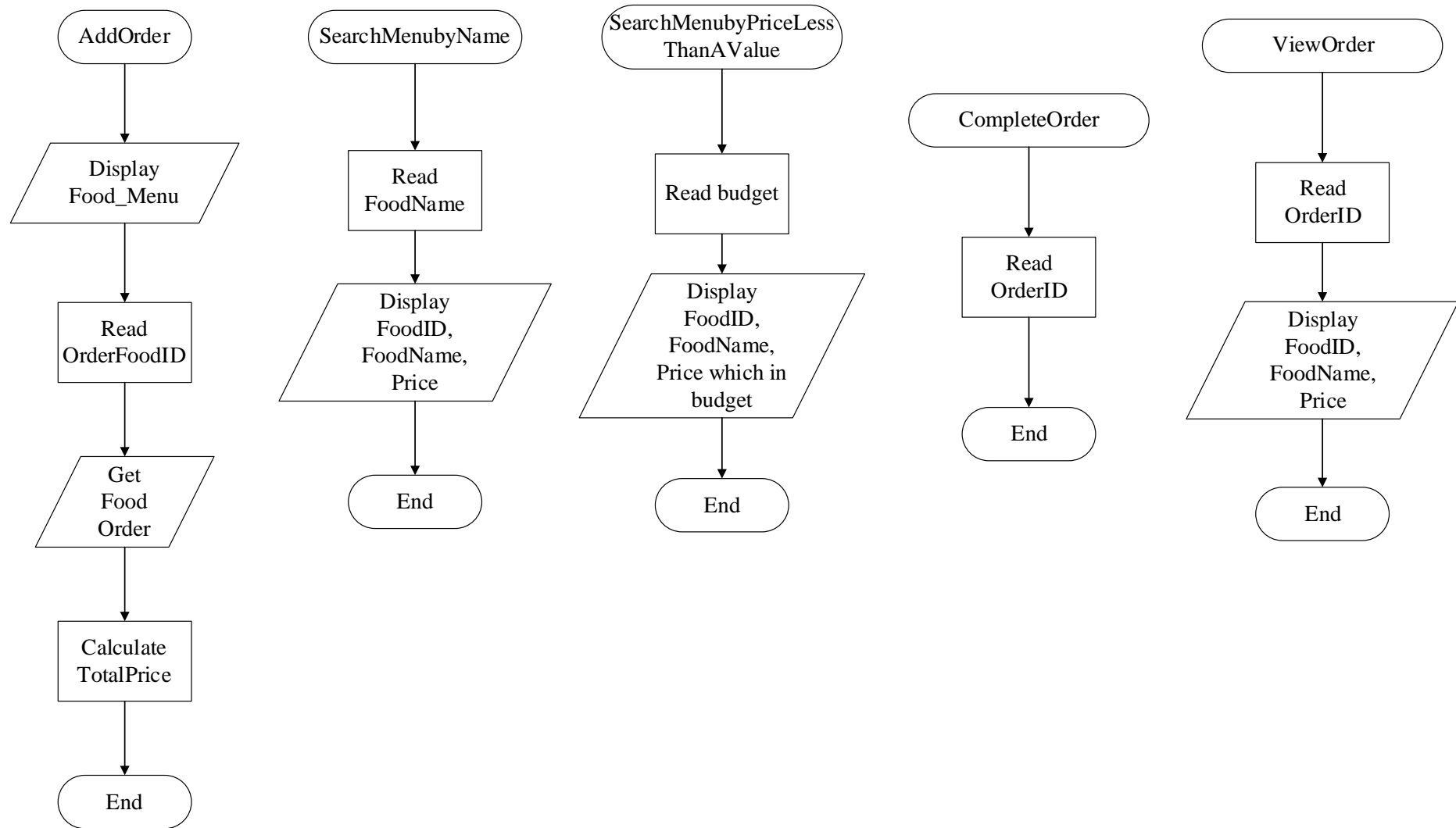
3.0   Flowchart

Figure 2: Flowchart of Entire Program

Figure 2.4 Flowchart of subroutines of the program