Assignment-3
CS-200 – Spring 2021
Dr. Shafay Shamail
TA: Muhammad Rohan Hussain (22100063)
Huzaifah Nadeem, Hammad Qayyum, Taha Sardar, Taha Razzaq
Read the handout very carefully.
(Open: Monday, 17 March 2021 – 11:55 AM)
(Deadline: Sunday, 28 March 2021 – 11:55 PM)

Important Guidelines:

1. Submit only one *.cpp file that includes all of the tasks and one main to test each task.
2. Name your code file using the following format: "A3-*yourrollno*.cpp". Do not submit any other file.
3. You are supposed to do this assignment alone. Any kind of collaboration, except for discussing what was taught in the class is strictly prohibited.
4. Your assignment will be checked for plagiarism via MOSS.
5. In case assignment work is found plagiarized you may be awarded a zero. In addition, your case of cheating will be reported to the Disciplinary Committee.
6. Any late assignment or any other file except the above mentioned will not be accepted.
7. Your code needs to **compile and run** as per the requirements specified in the assignment for you to gain marks.

Coding Conventions:

1. Constants are "ALLCAPS" or "ALL_CAPS".
2. Variables are "allsmall" or "all_small".
3. All function names must be "firstWordSmallAllOtherWordsCamelCase".
4. All class names must be "CamelCaseWords".
5. All curly brackets defining a block must be vertically aligned.
6. Inline functions are not allowed. Make sure that you declare all the member functions including constructors, destructor, setters, getters and print functions as out-of-class functions using the scope resolution operator :: .

Assignment 3

McDonald’s Order Queue System

TA: Muhammad Rohan Hussain (22100063)

Total Marks = 100

Introduction

Before reading this document, watch the **tutorial** that has been uploaded (https://www.youtube.com/playlist?list=PLkt3DFCNIchG3aZWT5CbC2SyeXNmjvD3r) that will show a working demo of this system. The tutorial is the primary source. It will be hard to understand this document without watching it. It will give you a good understanding of what you are trying to make. Use the **skeleton code** provided and build on top of it.

Before you start this assignment, please know that if you approach this assignment with the intention of just getting over with it as soon as possible, then it may get frustrating. Please approach it as a fun learning experience, because if you do, it will become easier on you. Either way, you will have to do it. If you do it with the intention to learn, you may get a good grade, a lot of learning, and less stress.

If your concepts of pointers, passing objects to functions, etc. are weak, then you will struggle a lot with this assignment. It’s better to spend some time clearing those concepts, otherwise you will have to spend ten times more time trying to debug problems that arise because of weak concepts.

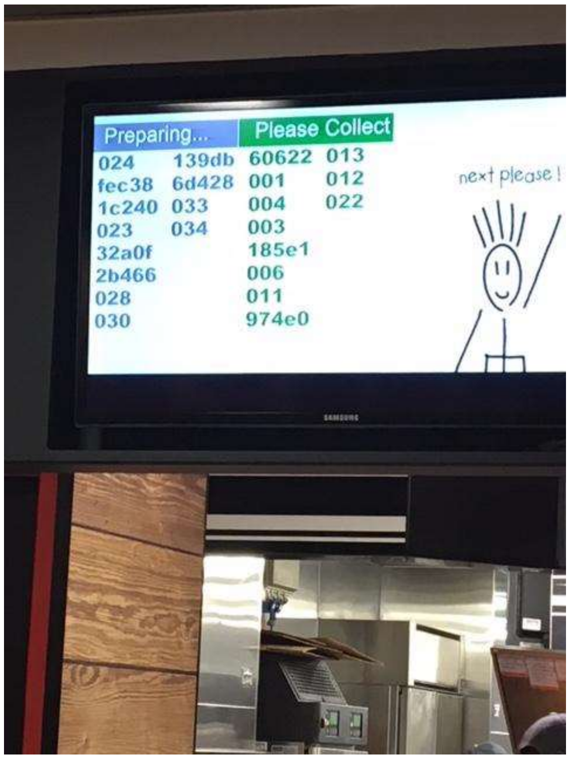
Take it one step at a time. Don’t worry about whether or not you will be able to finish it in time. Just start doing it one small part at a time. You will see that it only *looks* difficult.

And do not leave it till the last day. Start working on it now.

Implementation Steps

Make sure each Step is working perfectly before moving on to the next.

You may have noticed the McDonald's Order Queue System:



Once you place an order at the counter, you are given a receipt that has your bill and **an order number**.

This **order number** immediately appears on the screen in the "Preparing" column, signifying that your order is being prepared.

Once it is prepared, your order number appears in the "Collect" column, asking you to collect it. We will implement this system.

Don't be intimidated. This assignment will guide you through the process as well. Before you read further, think of how you can create a good Object-Oriented Abstraction for this problem (i.e. which Classes should you create).

We need the following functionality:

- Customer can choose Items and place an Order
- The Order is placed, and a receipt is generated which shows the Total Price of all Items and the Order Number.
- The Orders should appear in the Preparing and Collect columns.

Think of what Classes can be made and what will be their attributes and methods. There is more than one way of modeling this problem in Object-Oriented Programming. Only one way is described here. If the abstraction you have come up with is different, but you think it is valid or better, contact TA (Muhammad Rohan Hussain) and if approved by him, you can use your own abstraction. That way, you can even ignore the entire skeleton code and the rest of this document and create this system in your own way. The final result must look like what is shown in the end of this document.

Once individual components and steps are explained, overall working of the system is explained in the end. Create any new methods other than the ones mentioned here in any class you want. You will probably need to do that. You can also add as many arguments (parameters) to methods as you want. You can also choose to not implement a certain method if you feel like it isn't needed.

Step 1 – Class Item [5]

Create a Class called **Item**, which represents an item in McDonald's Menu, for example a Big Mac, or a McFlurry Oreo. An Item has the following attributes and methods:

- name (string)
- price (int)
- A **display()** method that displays the name and price of an item in the console.

The class will also have a parameterized constructor that takes a string and an integer as arguments and sets them as the Object's name and price respectively.

Create any methods and constructors that you need. Don't create methods that aren't needed or used.

Step 2 – Menu Class [5]

Create a class called **Menu**, which represents McDonald's entire Menu. A menu is simply a list of **Items**. In our case, our Menu will be fixed and hard coded in the constructor for simplicity. We will **not** take the Menu as user input. A Menu has the following attribute(s) and methods:

- An array of **Items** (size 10) called **menu_array**
- A default constructor that fills this array with 10 Item Objects. You can hard code this and insert any 10 items you like.

```
    Item menu_array[10] = {Item("Big Mac",500), Item("McFlurry Oreo",150),…};
```

- A display() method that displays the menu. It must display the item number (array index +1), name, and the price of every item in a user friendly manner. You can use the Item.display() method here.

Step 3 – Order Class [30]

We are going to create and manage a dynamic 2D Array in this Class. You may remember that we asked you to do something similar in Lab 8 Task 1. If you solved that perfectly, then you can reuse that code here (with some modifications). If you weren't able to solve Lab 8 Task 1, you can start here afresh.

Create a Class called **Order**. Each object of this class will be a customer's order. Each Order will have the following attributes:

- **order_number (int):** This is the order number that will be displayed on the screen in the columns "Preparing" and "Collect". It must be unique for every order. You can use a static variable for this purpose, or any other implementation.
- **order_items (A 2D Array of integers):** The size of this array will be specified by customer when placing an order. Why is it a 2D Array? Because for each Item in the order, the Array will store three things:
  1) The index of the Item in the Menu Array

2) The quantity of this Item in the Order
3) The bill for this Item (price in integers only, not in floating point numbers)

To help you visualize this 2D Array, look at the following diagram:

0	4	8	9
4	5	2	1
2000	1000	1200	750

The above 2D Array has dimensions 3x4. This means that the customer wants to purchase 4 items in various quantities. If the customer wanted to purchase 6 items, the array dimensions would have been 3x6.

- "0", "4", "8", "9" are the indexes of the items in the **menu_array**.
- "4", "5", "2", "1" are the quantities of each of these Items.
- "2000", "1000", "1200", and "750" are the prices of each of these items x quantities.

For example, let's read the first column of the 2D array. The first value in the column is 0, which is the item's index in **menu_array**. Let's say our **menu_array** has Item("Big Mac",500) at index 0, so the customer wants to order a Big Mac. The rest of the first column of the 2D Array tells us that the customer wants to order 4 Big Macs and the bill for that is 2000. Each Big Mac costs 500, so the total bill of 4 Big Macs will be $500 \times 4 = 2000$.

- **num_items (int):** This stores the number of items in this Order (which is the horizontal size of our 2D array)
- **Total Bill (int):** This is the total price of the order (sum of individual prices of all items). This is simply the sum of the last row of the 2D Array.

[You can have a 4*3 array where columns indicate array index, quantity, and price and rows indicate order of each item.]

The class will have the following methods. You can create more if you want.

- **display():** This method will display all information about this order.
- **~Order():** This destructor will delete the memory allocated for the 2D Array.

Step 4 – Class Linked List and Node [35]

We are going to create a Queue here using a Linked List. Insertions are only allowed at the Tail of the list, and removals are only allowed from the head of the list. TA Rohan's tutorial on Linked Lists may help with this part. If you have implemented a Queue or a Linked List in any previous lab, you can reuse that code here (make sure it is perfectly working before copy pasting it here). If you do that, please mention in the comments which lab you copied that code from. This way we can check to ensure that you didn't copy the code from someone else's lab.

Start by creating a class called **LinkedList** in which you implement a Linked List. Create a class called **Node** as well. Each Node stores the relevant Linked List pointer (next) and an **Order** Object. Thus, each Node in the Linked List is an Order placed by the Customer. This is for our queue of orders. The Node Class must have a destructor that deletes the Order as well.

Make sure the Linked List has a head pointer as well as a tail pointer (the tail pointer always points to the tail of the list). The tail pointer will make **addToTail()** very fast.

The Linked List must have the following methods:

- **addToTail():** This method adds an Order to the tail of the Linked List. It takes the Order as user input by displaying a list of items and each item's number (which is the array index + 1) and asking the user to choose the items. The user first tells us how many items (s)he wants in the order, then (s)he tells us the item numbers (array indices will be calculated by subtracting 1 from item number) of those items and then (s)he tells us the quantities of each of these items. The working of this system is clarified in the tutorial.
- **removeFromHead():** This method removes an Order from the head of the Linked List. Since we add new orders to the Tail of the list, the Order at the head of the list is always the oldest one.
- Friend function **moveOrderToCollectList():** This method will move the head Node of the **preparing_queue** list to the tail of the **collect_queue** list. This means moving the oldest Order from the Preparing list to the Collect list. It will be a friend function of the Linked List class so that it can access the head and tail pointers.
- **Display():** This method will display the Linked List by traversing each Node.
- **~LinkedList():** This destructor traverses the linked list from head to tail and deletes every Node along the way.

These methods can have any arguments you want. You can create any additional methods in any class you want.

Step 5 – Class OrderSystem [25]

Create a class called **OrderSystem** which has the following attributes:

- A Menu object called **menu**.

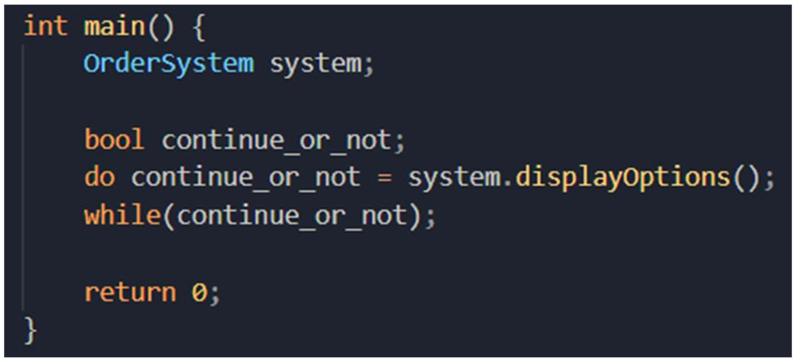
- A Linked List called **preparing_queue:** This Linked List stores all the Orders that are still being prepared.
- A Linked List called **collect_queue:** This Linked List stores all the Orders that are ready to be collected. Once an order is ready, it is moved from the preparing_queue to the collect_queue.

The Class also has these methods:

- **addOrderToQueue:** This method simply creates an Order Object and calls the **preparing_queue** Linked List's **addToTail** method to add this Order to the end of the Linked List. It will also display the **preparing_queue** once the Order has been added (display all details of the order, such as items, order number, total_bill).
- **setOrderToPrepared:** This method will call the **moveOrderToCollectList** function and will move the preparing_queue's head Order to the collect_queue list's tail. Then, it will display the prepared_queue and the collect_queue lists (only display the order number of each order).
- **removeOrderFromQueue:** This method simply removes an Order Object from the head of the **collect_order** Linked List by calling the **removeFromHead** method of the Linked List. It also displays the **collect_queue**.
- **displayQueue:** This method displays both the queues (**preparing_queue, collect_queue)**.
- **displayOptions:** This method displays the console interface to the user and calls the relevant functions according to user input. For example, if the user enters 1, it calls the addOrderToQueue function.

Working

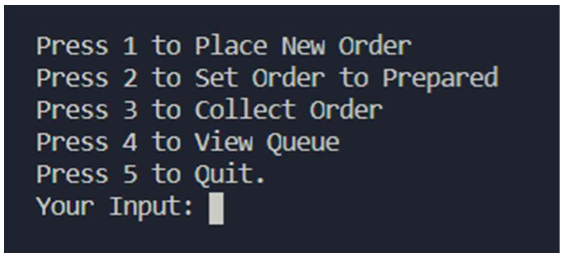
The main function is very simple and has only 5 lines. It creates a **OrderSystem** object called **system**. Then we run an infinite loop which runs as long as the user does not explicitly quit. When the user quits, the **system.displayOptions()** method returns **false**, the while loop condition becomes false, and so the loop exits.



```
int main() {
    OrderSystem system;

    bool continue_or_not;
    do continue_or_not = system.displayOptions();
    while(continue_or_not);

    return 0;
}
```

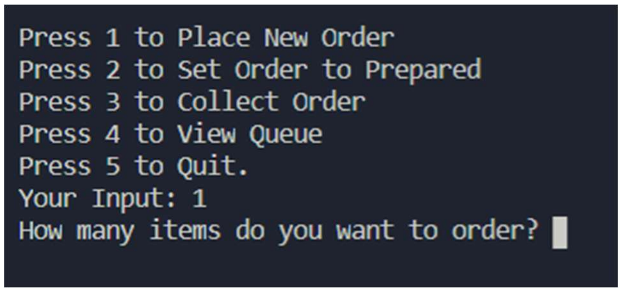
Note: You don't need to do error handling in inputs.

When the program starts, the following prompt menu will appear in the console:



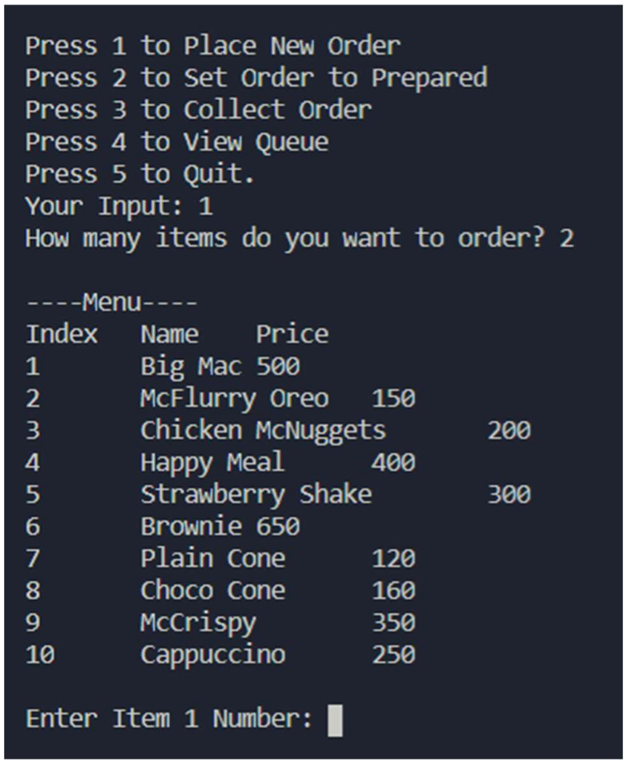
```
Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

If the user inputs 1, the following will appear:



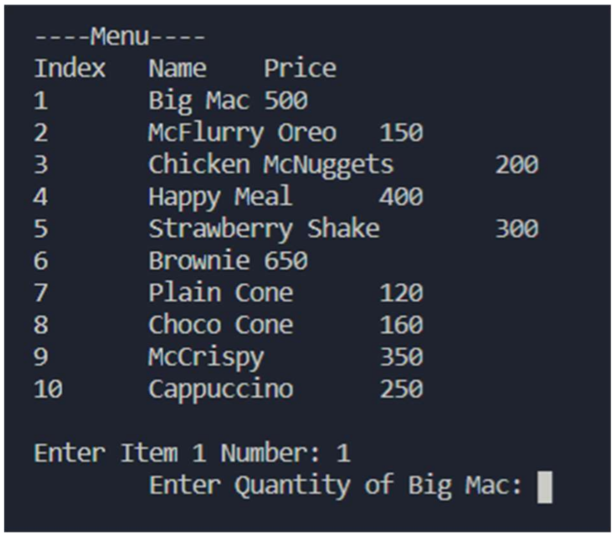
```
Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input: 1
How many items do you want to order?
```

If the user wants to order 2 items, then the following occurs.



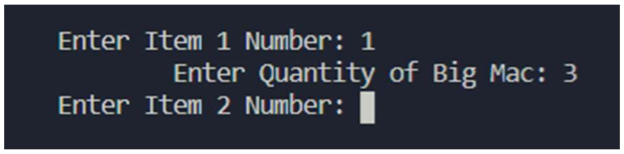
```
Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input: 1
How many items do you want to order? 2

----Menu----
Index   Name    Price
1       Big Mac 500
2       McFlurry Oreo   150
3       Chicken McNuggets       200
4       Happy Meal      400
5       Strawberry Shake        300
6       Brownie 650
7       Plain Cone      120
8       Choco Cone      160
9       McCrispy        350
10      Cappuccino      250

Enter Item 1 Number: █
```

The Menu is displayed, and the user is asked to enter the item number of the first item they want to order. Let’s say the user inputs 1 (Big Mac). Note that in the Menu, Big Mac is at index 0. So you will have to subtract 1 from the user’s input.



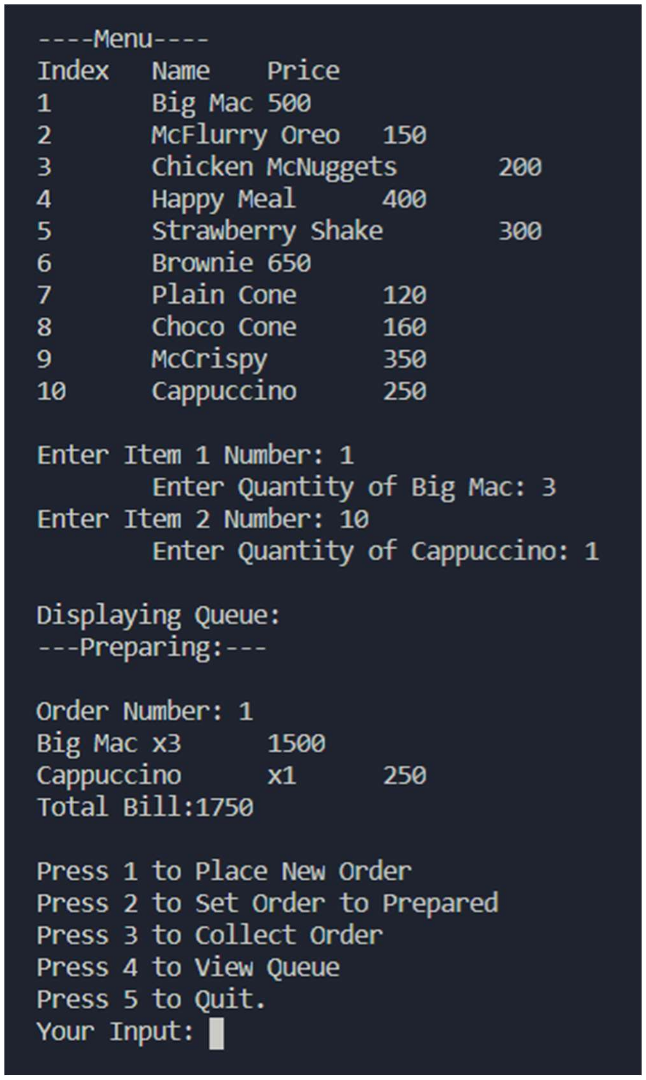
```
----Menu----
Index   Name    Price
1       Big Mac 500
2       McFlurry Oreo   150
3       Chicken McNuggets       200
4       Happy Meal      400
5       Strawberry Shake        300
6       Brownie 650
7       Plain Cone      120
8       Choco Cone      160
9       McCrispy        350
10      Cappuccino      250

Enter Item 1 Number: 1
        Enter Quantity of Big Mac: █
```

The user is then asked the quantity of the item (how many Big Macs). Let’s say the user enters 3.



```
Enter Item 1 Number: 1
        Enter Quantity of Big Mac: 3
Enter Item 2 Number: 
```

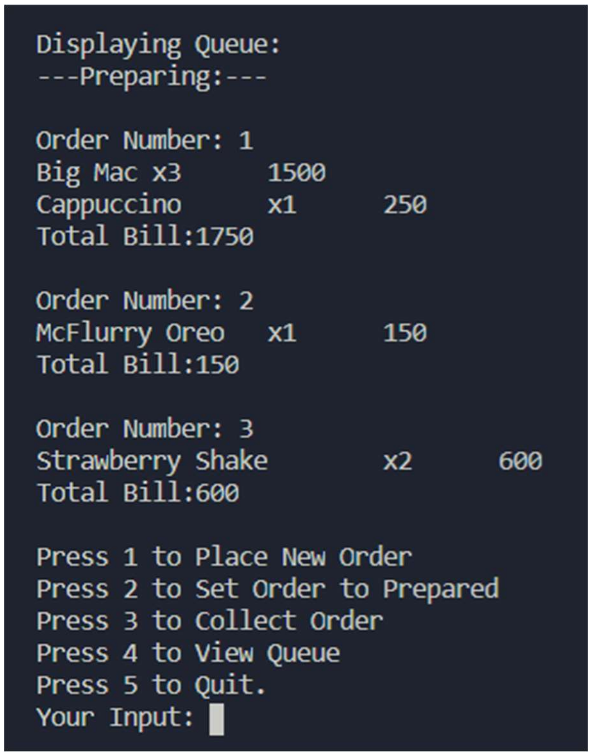
The user then inputs the second item and its quantity. Let’s say the user orders a Cappuccino.



```
----Menu----
Index   Name    Price
1       Big Mac 500
2       McFlurry Oreo   150
3       Chicken McNuggets       200
4       Happy Meal      400
5       Strawberry Shake        300
6       Brownie 650
7       Plain Cone      120
8       Choco Cone      160
9       McCrispy        350
10      Cappuccino      250

Enter Item 1 Number: 1
        Enter Quantity of Big Mac: 3
Enter Item 2 Number: 10
        Enter Quantity of Cappuccino: 1

Displaying Queue:
---Preparing:---

Order Number: 1
Big Mac x3      1500
Cappuccino      x1      250
Total Bill:1750

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input: 
```

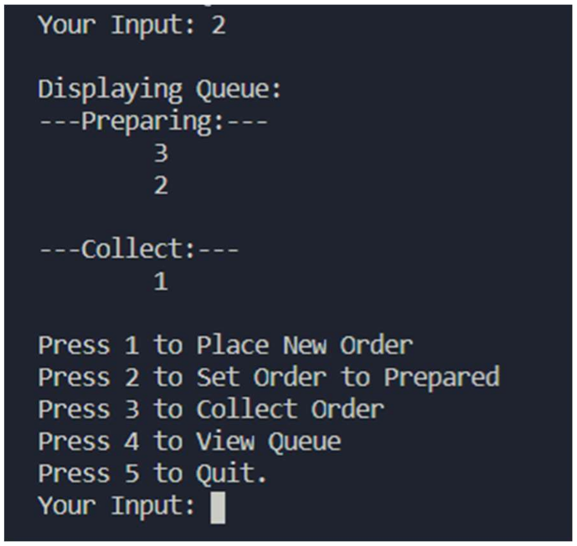
The Preparing Queue is displayed. It has only 1 order in it and its Order Number is 1. The details of the order are visible: 3x Big Macs and 1x Cappuccino. The total amount is 1750. The prompt menu will reappear and ask the user to choose an option.

Let’s say that two more orders are placed. The printed queue will look like this:



```
Displaying Queue:
---Preparing:---

Order Number: 1
Big Mac x3      1500
Cappuccino      x1      250
Total Bill:1750

Order Number: 2
McFlurry Oreo   x1      150
Total Bill:150

Order Number: 3
Strawberry Shake        x2      600
Total Bill:600

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

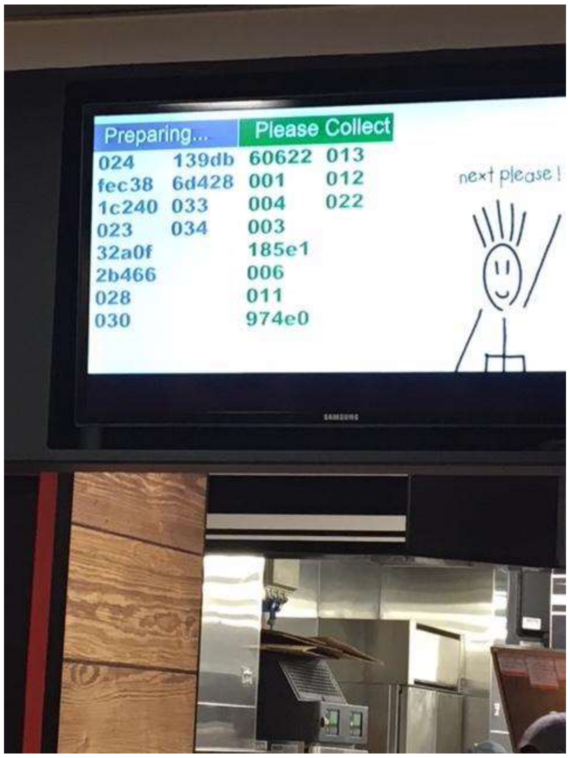
Since we are adding new orders to the Tail of the List, this is why the oldest orders are being printed first because printing is done in the head-to-tail direction.



```
Your Input: 2

Displaying Queue:
---Preparing:---
        3
        2

---Collect:---
        1

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

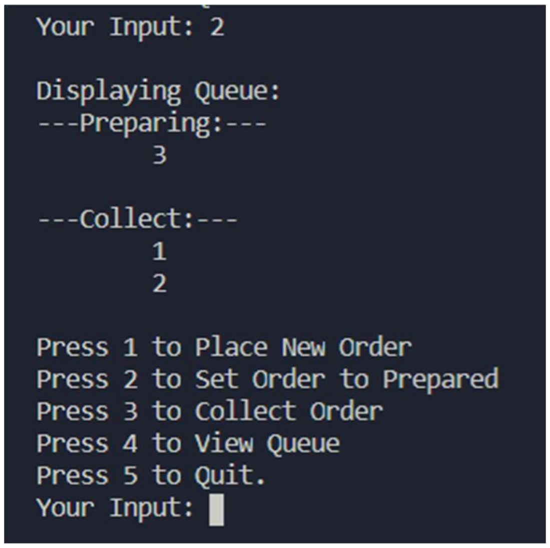
Let’s say that Order Number 1 is now prepared and is ready to be collected. We input 2:

The order numbers of both the **preparing_queue** and the **collect_queue** are displayed. We are not displaying all the details of the order here, which is what happens in McDonald's. Only the order numbers appear.



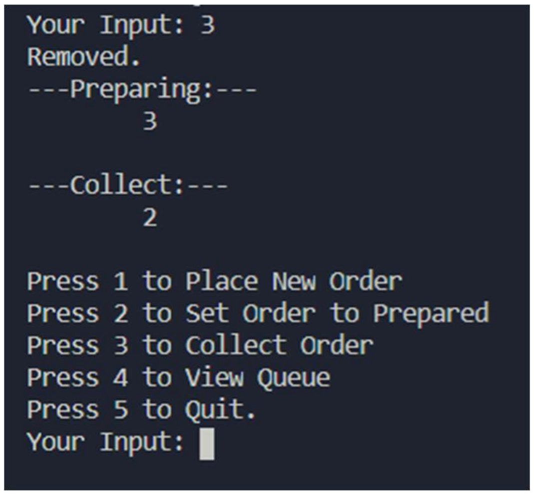
Orders can only be prepared in the order in which they were placed. Order 3 cannot be prepared before Order 2. This is automatically ensured by our Linked List **addToTail()** and **removeFromHead()** methods. Those who order first get served first. *Justice is served (pun intended).*

Let’s say Order Number 2 is also prepared now. We input 2:



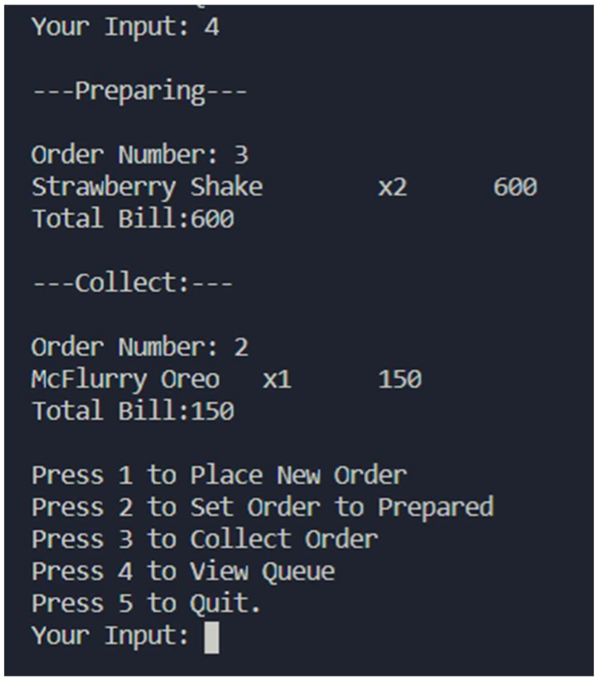
```
Your Input: 2

Displaying Queue:
---Preparing:---
        3

---Collect:---
        1
        2

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

Orders 1 and 2 are in the collect_queue now and can be collected. Let’s say the user now collects the order. It makes sense that since Order 1 was prepared first, it should be collected first. So, simply inputting 3 will serve Order 1.



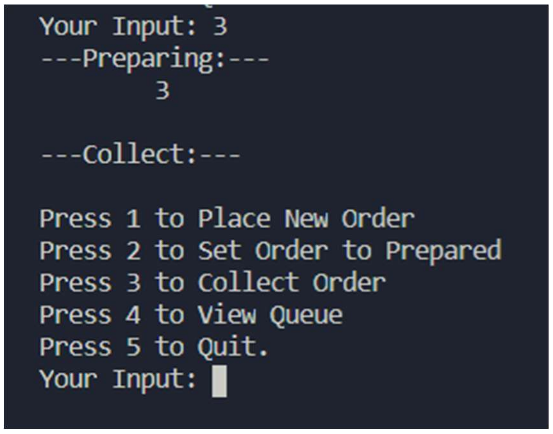
```
Your Input: 3
Removed.
---Preparing:---
        3

---Collect:---
        2

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

You can see that Order 1 was collected and is now not present in any queue. It is officially out of the system. We can view the detailed queues at any time by entering 4.



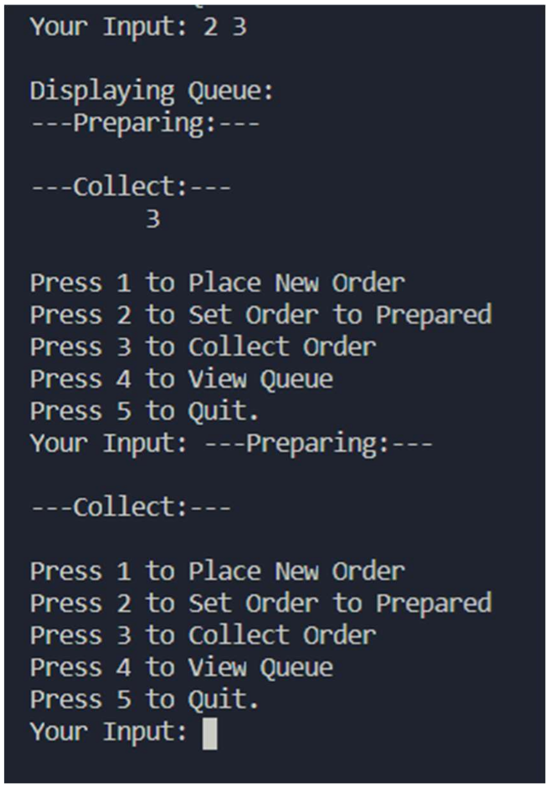
```
Your Input: 4

---Preparing---

Order Number: 3
Strawberry Shake        x2      600
Total Bill:600

---Collect:---

Order Number: 2
McFlurry Oreo   x1      150
Total Bill:150

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

If we Enter 3 again to serve another order, then Order 2 will be collected.



```
Your Input: 3
---Preparing:---
        3

---Collect:---

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input:
```

The **collect_queue** is now empty. Let’s move Order 3 to the **collect_queue** and then collect it. We do this by entering 2 and then 3.



```
Your Input: 2 3

Displaying Queue:
---Preparing:---

---Collect:---
        3

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input: ---Preparing:---

---Collect:---

Press 1 to Place New Order
Press 2 to Set Order to Prepared
Press 3 to Collect Order
Press 4 to View Queue
Press 5 to Quit.
Your Input: 
```

You can see that both queues are empty now. All orders have been served.

Good luck! We hope you learn a lot.