Even Semester (2022)

BINUS UNIVERSITY
BINUS INTERNATIONAL

Final Project Cover Letter
(Group Work)

Student Information:

	Surname:	Given Name:	Student ID Number:
1.	Silalahi	Joshua Alexander	2502005244
2.	Mazel	Kimberly	2502022250
3.	Clarin	Maria	2501990331

Course Code : COMP6048001 **Course Name :** Data Structures

Class : L2BC

Lecturer : NUNUNG NURUL QOMARIYAH, S.Kom., M.T.I., Ph.D.

Type of Assignments: Term Final Project

Submission Pattern

Due Date : 16 June 2022 **Submission Date :** 15 June 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Joshua Alexander Silalahi

Kimberly Mazel

Maria Clarin

Table of Contents

1. Problem Description

The speed with which restaurants serve their customers is a significant factor. Customer satisfaction and income can both increase from faster service. Customers, for instance, benefit from faster service because their orders arrive sooner. As a result, customers will spend less time waiting for their orders and at the restaurant as a whole. This gives other customers more time and room to eat. To illustrate, a restaurant with fast service will have more customers in a day compared to one with slow service. The in-and-out flow of the first restaurant will be much faster than the second.

Aside from that, restaurants require an organised system in order to ensure a smooth-sailing service in the kitchen environment. Ordering systems have to be well organised with a simple interface that works effectively and efficiently. This is because employees have to be able to operate them quickly and easily while having all the orders arranged and organised, ready to be passed to the kitchen to cook, and later on to proceed with the payment. Simple interfaces also help in making the system more easily understandable.

Overall, an effective ordering system is required to provide high-speed service. The system would require a data structure that allows cashiers to quickly add, remove, and retrieve orders while consuming minimal memory. This project aims to determine which linear data structure is the most efficient for an ordering system.

2. Alternative Data Structures

2.1. Stacks

Stacks are a fundamental data structure that stores items in a linear fashion. It is simple and easy to implement and is efficient in data and function management. The local variables that are stored in a stack when a function is called is automatically destroyed once it is returned. They are not easily corrupted, so it is a secure and reliable way to store data.

Although it appears to be a reasonable option for an ordering system, it has one major problem. The LIFO (Last in, First out) technique is used in stacks, which means that the most recently inserted item is the first to be removed. To illustrate, the top plate in a stack of plates is the last to be set but the first to be used. This is not ideal for an ordering system as the first customer should be the first to get their order and leave. It would not be fair for the last customer to be the first to get their order. Conclusively, **stacks are not a data structure that is fit** for an ordering system, despite its simplicity and ease of implementation.

2.2. Queues

Queues are a type of data structure that works similarly to stacks in that it keeps objects in sequential order. Queues, on the other hand, operate on the FIFO (First in, First out) principle. This means that the first item that is added, will be the first to be removed. Its technique mimics the behaviour of a queue in real life, hence the name.

A queue would be an appropriate data structure for an ordering system because it would replicate the real-life customer queue. The order of the first customer will be the first to be worked on. Workers may keep track of the tasks that need to be

completed in order, and once an order is completed, they can simply remove it from the data structure.

2.3. Linked Lists

Linked lists are a linear data structure that works differently from queues and stacks. Linked lists make use of nodes and pointers, instead of contiguous memory locations. The data is stored in nodes, and each node contains a reference to the next node. The nodes are linked using pointers. Linked lists are particularly useful when a lot of insertions and removals are needed. However, they are more inefficient when searching is involved.

Linked lists are also dynamic, meaning that there is no limit on the number of items that are stored in them. This is appropriate for an ordering system, as there is no way to accurately tell the exact amount of customers they will have in a day. With linked lists, workers will not have to worry about the linked lists reaching their capacity.

2.4. Priority Queues

Priority queues are a form of queues that allow the user to control the order of the items in the queue. As its name suggests, a priority can be assigned to certain items which determine the order in which the items are removed from the queue. Usually, a priority queue is done in ascending or descending order.

In this circumstance, priority queues can be used for time-sensitive orders. For example, orders that take a long time to make can be done first, so that the customer will not have to wait long. However, generally, priority queues are not as preferred compared to regular queues when it comes to an ordering system. This is because restaurants do not require a queue that sorts priorities since most restaurants operate on a ‘first come, first serve’ manner.

	Average Time Complexity			
	Access	Search	Insertion	Deletion
Stack	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)
Linked List	O(n)	O(n)	O(1)	O(1)
Priority Queue	O(1)	O(n)	O(log(n))	O(log(n))

3. Theoretical Analysis

3.1. Queues

How Queues Work in General

Queue is a form of data structure that is somewhat reminiscent of the queues often found in real life. Queues in general concept contains several components such as waiting in a line, waiting process, service, process, as well as the elements itself, which in this project case are customers who are ordering food.

In data structure, Queue is a linear data structure which keeps elements in a sequential order. This means that Queue follows the FIFO (First In First Out) method, and this proves to be perfect for an ordering system as the first customer who orders, would be the first customer to be served or receive their orders. As for Stacks, it is deemed unsuitable for a Restaurant management system such as this one, as Stacks follow the LIFO (Last In First Out Method). If stacks were to be used, then let's say for example there are 10 customers, then the 10th customer would be served first, which is of course unfair to the other customers.

All in all it can be concluded that queues pass the category of a data structure that can be used for an order management app. This is because queues have all the required properties and methods that resemble an ordering line. Queues also prove to be a very efficient data structure as it has a relatively efficient time complexity (shown in the table above).

Queue - Pros and Cons

With any data structure there are of course pros and cons when using it. Therefore, it is very essential to pick the right data structure for your projects. Queues have the ability to manage a large amount of data with ease, and also have an easy insertion and deletion as it follows the First In and First Out rule. Queue is a very quick data transfer method, as it has high speed in data inter-process communication.

As for its drawbacks, queues are not recommended to be used for insertion and deletion of elements from the middle, as it would take a long time to do so which makes it inefficient. Queues also use arrays, and arrays are not that dynamic, so queues have a limited space to work with. This brings on to the next disadvantage, in which the size of a queue needs to be declared first prior to the queue being initialised.

Applications of Queue and Real Life Use Case

Queue has 4 basic operations, the 4 main operations of the queue are enqueue, deque, isFull(), and isEmpty(). The enqueue function, as the name suggests, adds an element to the queue. As for how the FIFO method works, the insertion always happens at the rear or the back of the queue. The enqueue method would be useful to add in customer orders in the program.

The opposite of the enqueue method is the dequeue method. The dequeue method pops out the outer element of the queue. In an order management system, this dequeue method would be used to remove the order at the most front of the queue, which translates to the order being completed or removed. Following the FIFO method, dequeue always happens at the front of the queue.

A queue has a method to check whether it's empty or not using the isEmpty() method. The isEmpty() method traverses through the queue to check whether an element is present in the queue or not, if the elements inside of the array equals to 0, the program would return a message confirming that there are no elements available inside of the queue. Queues are made with an array, and arrays are not dynamic, so the size of the array needs to be set from the beginning.

In order to check whether the array is filled or not, use the isFull() function. The isFull() function would traverse the array to check whether there is some more space to add elements to the array. If the array is already filled to its maximum capacity, then the program would confirm to the user that the array is full and that no other elements can be inserted. The isFull() method is similar to that of a food stock in a restaurant. A restaurant doesn't have an unlimited capacity of food materials, so a maximum limit of production needs to be set each day.

queue.hpp Code

In building the queue.hpp file, it is important to declare variables and make a constructor first. Templates are also essential to create a blueprint for the formula. Templates are used in order for a function or class to work with multiple data types.

```
4   template <class X>
5   class queue
6   {
7       X *arr;         //an array to store queue elements
8       int capacity;   //maximum capacity of the queue (size)
9       int front;      //to track the first element in the queue
10      int rear;       //to track the last element  in the queue
11      int count;      //size count of the queue (to increment)
12
13  public:
14      queue(int size = 10);     //constructor with default size 10
15
16      void remove();            //the functions included in the class queue
17      void add(X x);
18      void peek();
19      int size();
20      bool isEmpty();
21      bool isFull();
22      void display();
23      void checkCap();
24      void formatToAdmin();
25      int totalSize();
26
27  };
28
29  //Constructor
30  template <class X>
31  queue<X>::queue(int size)
32  {
33      arr = new X[size];
34      capacity = size;
35      front = 0;
36      rear = -1;
37      count = 0;
38  }
39
```

Image 1. Attributes, Templates, Methods, And Constructor Declaration.

```
41  //function to add an item to the queue
42  template <class X>
43  void queue<X>::add(X item)
44  {
45      // check for queue overflow
46      if (isFull())
47      {
48          cout << "Order Full" << endl;
49          return;
50      }
51      else
52      {
53          rear = (rear + 1) % capacity;
54          arr[rear] = item;
55          count++;
56      }
57  }
58
59  //function to remove the front element
60  template <class X>
61  void queue<X>::remove()
62  {
63      // check for queue underflow
64      if (isEmpty())
65      {
66          cout << "Currently Empty" << endl;
67          return;
68      }
69      else
70      {
71          front = (front + 1) % capacity;
72          count--;
73      }
74  }
```

Image 2. add() and remove()

The add function is used to add elements into the array. If the array is full then, the program would return a message informing that the queue is full. If the array is not full, the program would add the order into the array, inserting it from the rear of the queue. As for deletion, it is the opposite of a queue. The program would check using the isEmpty() function to determine if there are elements inside of the array. An element could only be removed from the array if the array is not empty. If the array is empty, this operation could not be completed.

3.2. Linked Lists

How Linked Lists Work in General

In general, lists are a popular type of data structures in many programming languages, for example C, C++, Java, Python, etc. Linked list is also a form of data structure that is often implemented in most programs, and is widely used amongst programmers. Linked List implements pointers, which makes it a great tool to learn how pointers work. Through learning how to use linked lists, programmers would be more prepared to handle more complex data structures like graphs and trees.

Linked Lists are a form of data structure, through which objects are set up in a linear order. Linked Lists are determined by using a pointer for each object, this is very different from an array, as an array is determined by using the array indices. In a linked list, each node has the ability to store the data and the address of the next node. A node inside of a linked list consists of a data item and an address of another node, so it is important to always state those variables before proceeding through the linked list.

Linked List itself also has multiple variants such as, singly linked-list, doubly linked list, and circular linked list. For a singly linked list, a set of nodes has two fields, which are data and link. As the name suggests, the data field stores information or data, while the link field is a pointer that points to the next node. As for a doubly linked list, the main difference is that it contains a previous pointer, which means that it has three pointers. As for memories, the singly linked lists takes up less memory as it only has two fields. The singly linked lists only allow traversal to be done through the next node only, so traversal can be made possible in only a direction. As for doubly linked lists, it can traverse two ways, whether it’s to the next node or the previous node. But for this project, we are going to focus solely on the singly linked lists.

Linked Lists - Pros and Cons

Similar to queues, a linked list also has its advantages as well as its drawbacks as no data structures are perfect for everything. Unlike queues, linked lists are more dynamic and also give links to other structures of the program containing subsequent data elements. A linked list is a set of data structures that are not ordered by their placement inside of a system’s memory. Instead, linked lists are included as a part of an info inside of the foundational structure of the system. Therefore, a linked list is a dynamic data structure, this means that it can grow and shrink without any hindrance by allocating and deallocating its memory, through the help of pointers and nodes. So unlike an array, giving an initial size for a linked list is strictly not required. With it being dynamic, a linked list utilises its memories efficiently, and its size can increase

and decrease at run time which results in no memory wasted for unnecessary memory pre-allocation.

That being said, Linked List also has some notable drawbacks. One of the notable disadvantages of linked lists is that it is traversal. Traversal means that an element is not directly accessible by a linked list, while in an array it is much easier to access as it uses index. All that results in linked lists needing more memory space than an array. This is due to pointers requiring more memory to store the address of the next element in the list.

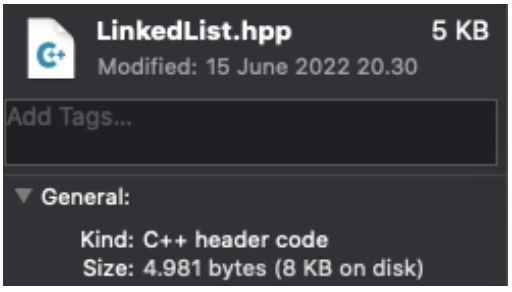
Applications of Linked List and Real Life Use Case

Both Linked Lists and Arrays have similar purposes, as the both of them function to store data. Beside arrays, queues could also be built using the help of Linked List, this makes Linked List a universal data structure which can be combined with other forms of data structures. A linked list would also be a great data structure for programs which need to allocate memory dynamically; this method is where a linked list will prove prominent over other data structures.

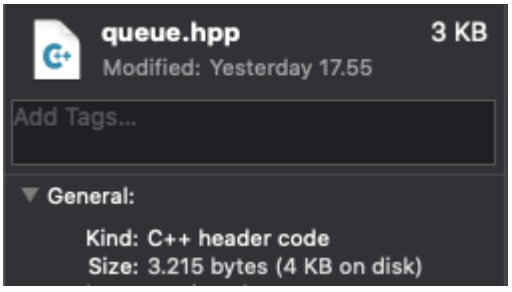
As for the operations, a linked list has 5 main basic operations. The first operation is the traversal method, this method functions to access elements inside of the linked list. The second and third operations are insertion and deletion, similar to queues insertion and deletion are used to add a new element to a linked list and delete an element from the linked list. The fourth main operation would be the Search method. A search method allows the user to find a node inside of a linked list by the help of address and pointers. And the last basic linked list operation is the sort method. With a sort method, all the nodes inside of the linked list would be sorted in an order, depending on how the order is set up. However, a sorting algorithm needs to be applied into the program for it to sort the elements, for example bubble sort, insertion sort, or quicksort to name a few.

3.3. Program Analysis

When containing a small number of orders, the size of the linked list and queue does not differ greatly. However, the size of the linked list's header file is larger than the header file of the queue.



Linked List: 4.981 bytes



Queue: 3.215 bytes

To measure the efficiency between the two, we measured the size of the data structure and the processing time of the program to display the order queue. The following table displays the respective processing time and size for the same orders.

	Queues		Linked List	
No. of orders	Processing Time (seconds)	Size (bytes)	Processing Time (seconds)	Size (bytes)
10	1.08	40	1.12	80
20	1.68	80	1.83	160
40	1.76	160	2.03	320

The processing time is directly proportional to the number of orders. In other words, as the number of orders increases, so does the processing time. The size of linked lists are greater than queues, as linked lists require more memory for the tail and the pointers. The byte size of a node of a linked list is 8, while one element in the queue is 4 bytes.

Queues are easier to implement and have a lower memory usage. However, they are limited by their fixed sizes. On the other hand, linked lists are dynamic and do not have a limit on their size, but are more difficult to implement and take up more memory. Both data structures can be implemented in an ordering system, but the suitability of the data structures depend on the restaurant.

For instance, a small restaurant with limited resources would find it troublesome to implement a linked list. They would need a system with greater memory compared to using a queue. As they are a small restaurant, it is likely that they will have less orders compared to a larger restaurant. A queue would be a sufficient data structure as the limit in size would not be a serious problem for them. Meanwhile, linked lists would be more suitable for a larger restaurant with more orders. As the size of the linked list is dynamic, the larger restaurant would not have to worry about overflowing their ordering system. It is more likely that the bigger restaurant would have sufficient resources to handle the increase of memory usage.

4. Program Manual

When the driver (admindriver.cpp) is run, the user will be shown the program's name "Queue For Eat", followed by a list of actions they can take. These actions include taking an order, completing the top order (removing it), displaying the order queue, clearing the order queue, checking the memory of the data structure and exiting the program. The user can then input a number based on the choice they decide to make.

4.1. Take order

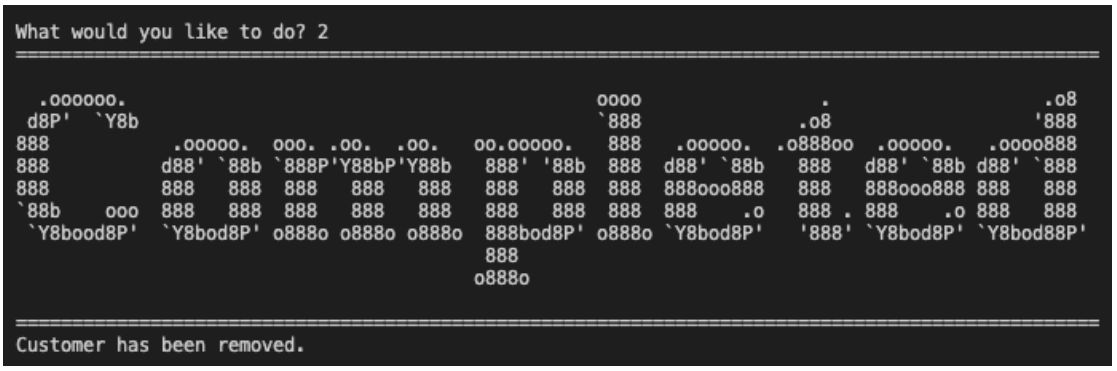
If the user chooses option 1, they will be asked to input a customer name. After inputting a name, the program will display the menu of food that is available. The user can then choose which items they would like to order by entering a number. The program will inform the user of their orders and continue asking for order numbers. Once they are done, the user can input 11 to return to the former option menu.

4.2. Display order queue

If the user would like to view all the orders in the data structure, they can input 3 in the main menu. The program will then display the customer orders, consisting of the order number, the customer name, followed by the orders and total price.

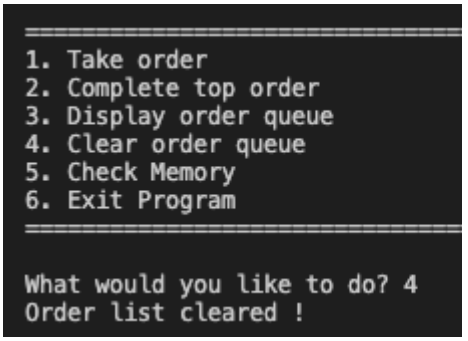
4.3. Complete top order

When the user inputs 2, the program will display the “Completed” view and inform the user that the customer has been removed. Only the first (top) order is removed, the rest will remain.



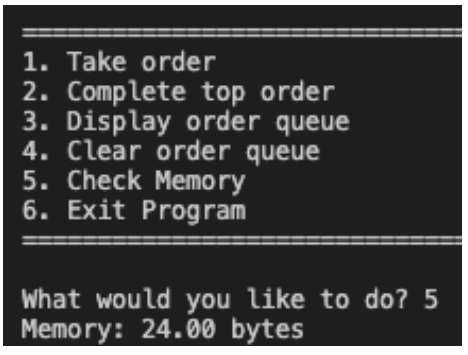
4.4. Clear order queue

If the user inputs 4 in the main menu, the order queue will be cleared. While complete top order only removes the first order, clear order queue removes every order in the data structure. The program will then inform the user that the order list has been cleared.



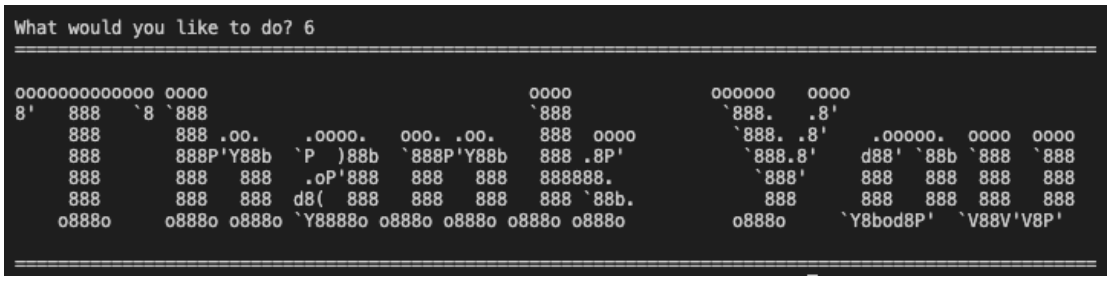
4.5. Check Memory

The 5th option will return the memory used by the data structure, more specifically, the bytes of the customer order queue.



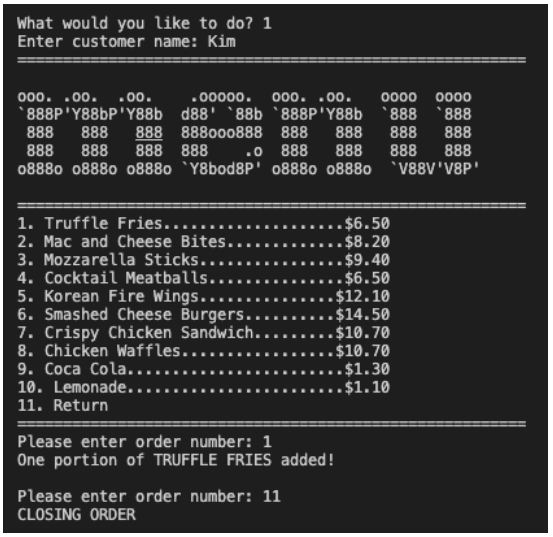
4.6. Exit Program

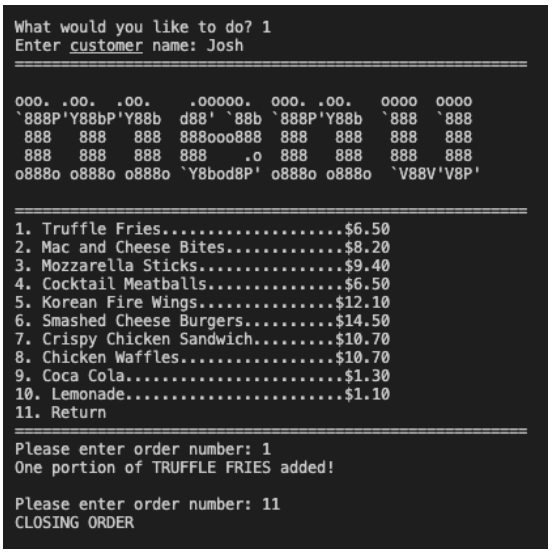
Once the user is finished with the program, they can input 6 in the main menu. The program will then display a “Thank You” message and the program will stop running.



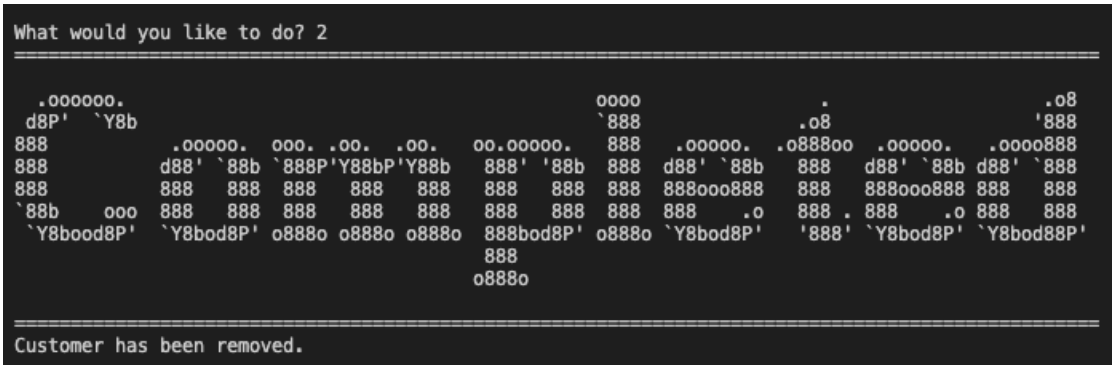
5. Program Evidence

5.1. Taking orders





5.2. Removing top order

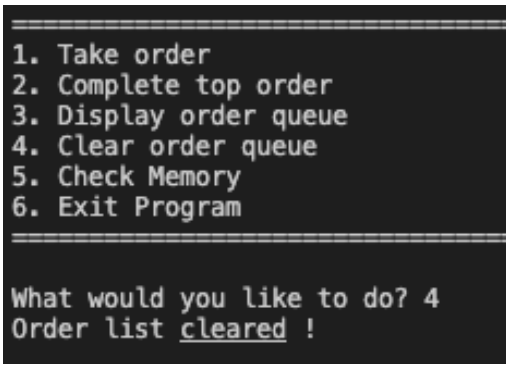


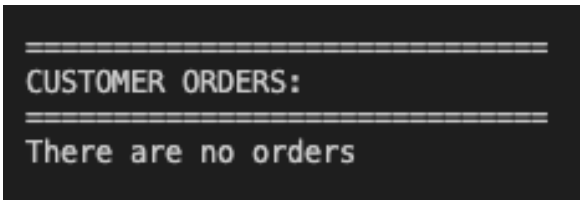
First order for “Kim” was removed. Displaying the queue shows that only the other 2 orders remain. In queues, the order number is not updated, the order number for Josh remains as 2. In linked lists, the order number is updated and changes to 1.

Linked Lists’ display

Queues’ display

5.3. Clear order queue





6. References

GeeksforGeeks. (2022). *Advantages and disadvantages of linked list*. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-linked-list/#:%7E:text=Insertion%20and%20Deletion%20Operations%3A%20Insertion,pointer%20needs%20to%20be%20updated.

GeeksforGeeks. (2022). *Applications, advantages and disadvantages of queue*. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-queue/

GeeksforGeeks. (2022). *Priority queue | set 1 (introduction)*. https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

GeeksforGeeks. (2022). *Applications, advantages and disadvantages of stack*. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/

How to find total memory consumption of c/c++ program. (n.d.). Codeforces. https://codeforces.com/blog/entry/49371

Linked list data structure in c++ with illustration. (2022). Software Testing Help. https://www.softwaretestinghelp.com/linked-list/#:%7E:text=A%20linked%20list%20is%20a%20collection%20of%20nodes%20that%20contain,list%20is%20called%20the%20Head

Standard template library. (n.d.). HackerEarth. https://www.hackerearth.com/practice/notes/standard-template-library/#:%7E:text=Its%20time%20complexity%20is%20O(logN)%20where%20N%20is%20the,element%20in%20the%20priority%20queue