ABSTRACT

Queue in data structures is a linear collection of different data types which follow a specific order while performing various operations. It can only be modified by the addition of data entities at one end or the removal of data entities at another. By convention, the end where insertion is performed is called Rear, and the end at which deletion takes place is known as the Front.

These constraints of queue make it a First-In-First-Out (FIFO) data structure, i.e., the data element inserted first will be accessed first, and the data element inserted last will be accessed last. This is equivalent to the requirement that once an additional data element is added, all previously added elements must be removed before the new element can be removed. That's why more abstractly, a queue in a data structure is considered being a sequential collection.

This is the brief information about the queue data structure.

TABLE OF CONTENTS

IMPLEMENTATION OF QUEUES IN PETROL BUNK

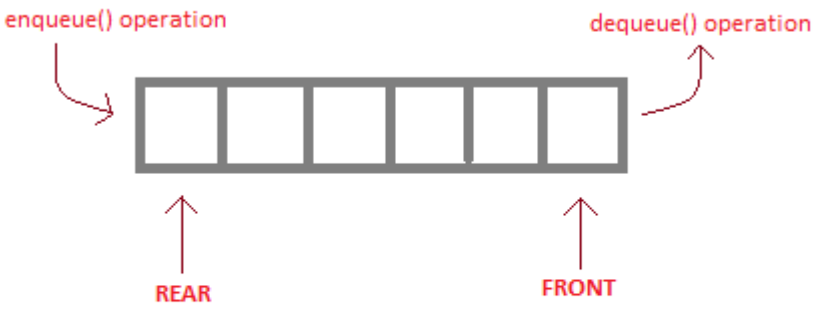
Page No.

CHAPTER 1

INTRODUCTION

1.1 Introduction:

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.



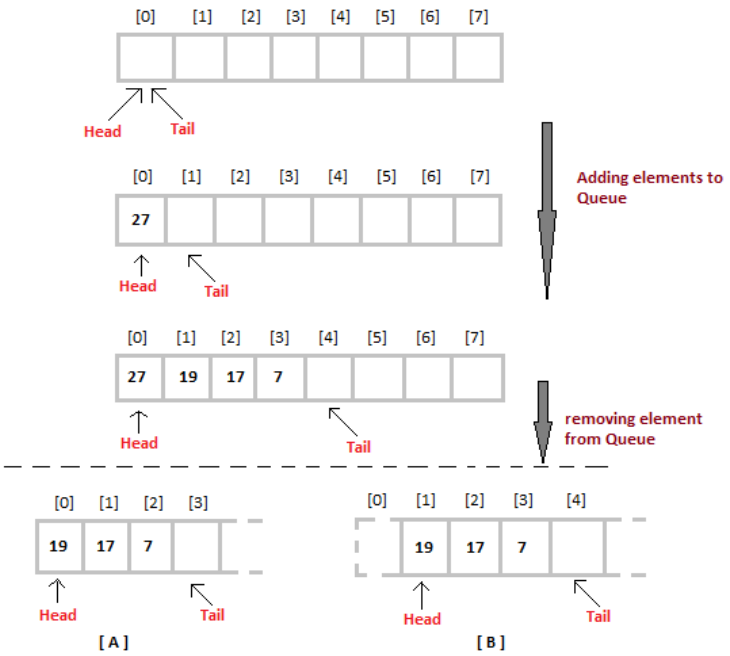
enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

1.2 QUEUE REPRESENTATION:

Queue can be implemented using an Array, Stack or Linked List. The easiest way of implementing a queue is by using an Array.

Initially the head(FRONT) and the tail(REAR) of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the tail keeps on moving ahead, always pointing to the position where the next element will be inserted, while the head remains at the first index.



When we remove an element from Queue, we can follow two possible approaches (mentioned [A] and [B] in above diagram). In [A] approach, we remove the element at head position, and then one by one shift all the other elements in forward position.

In approach [B] we remove the element from head position and then move head to the next position.

In approach [A] there is an overhead of shifting the elements one position forward every time we remove the first element.

In approach [B] there is no such overhead, but whenever we move head one position ahead, after removal of first element, the size on Queue is reduced by one space each.

Basic features of a queue:

1. **Like stack, queue is also an ordered list of elements of similar data types.**
2. **Queue is a FIFO(First in First Out) structure.**
3. **Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.**
4. **peek( ) function is oftenly used to return the value of first element without dequeuing it.**

CHAPTER-2

OPERATIONS OF QUEUES

2.1 Operations:

Unlike arrays and linked lists, elements in the queue cannot be operated from their respective locations. They can only be operated at two data pointers, front and rear. Also, these operations involve standard procedures like initializing or defining data structure, utilizing it, and then wholly erasing it from memory. Here, you must try to comprehend the operations associated with queues:

- **Enqueue() - Insertion of elements to the queue.**
- **Dequeue() - Removal of elements from the queue.**
- **Peek() - Acquires the data element available at the front node of the queue without deleting it.**
- **isFull() - Validates if the queue is full.**
- **isNull() - Checks if the queue is empty.**

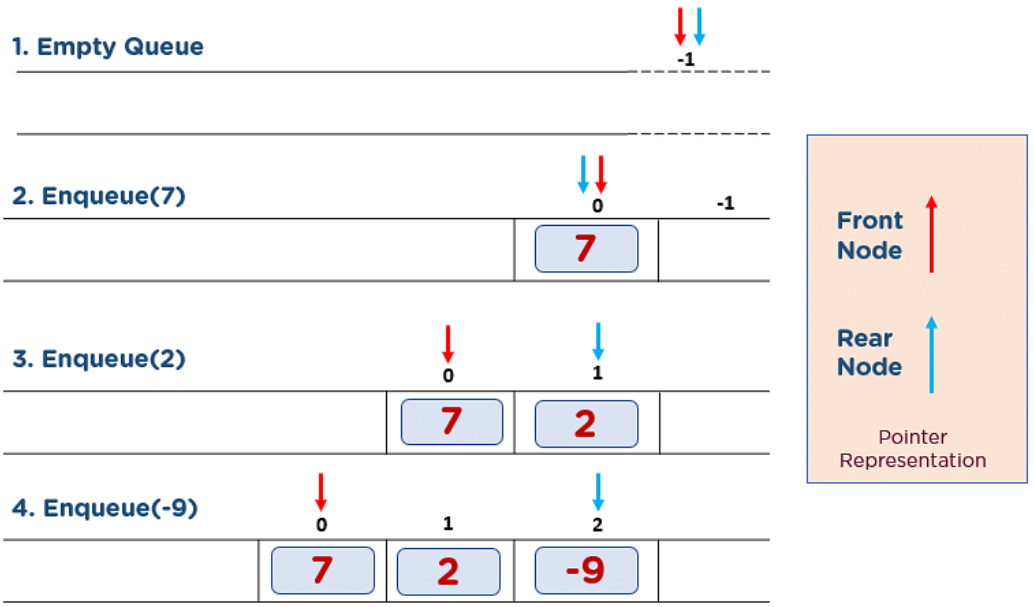
When you define the queue data structure, it remains empty as no element is inserted into it. So, both the front and rear pointer should be set to -1 (Null memory space). This phase is known as data structure declaration in the context of programming.

First, understand the operations that allow the queue to manipulate data elements in a hierarchy.

Enqueue() Operation

The following steps should be followed to insert (enqueue) data element into a queue -

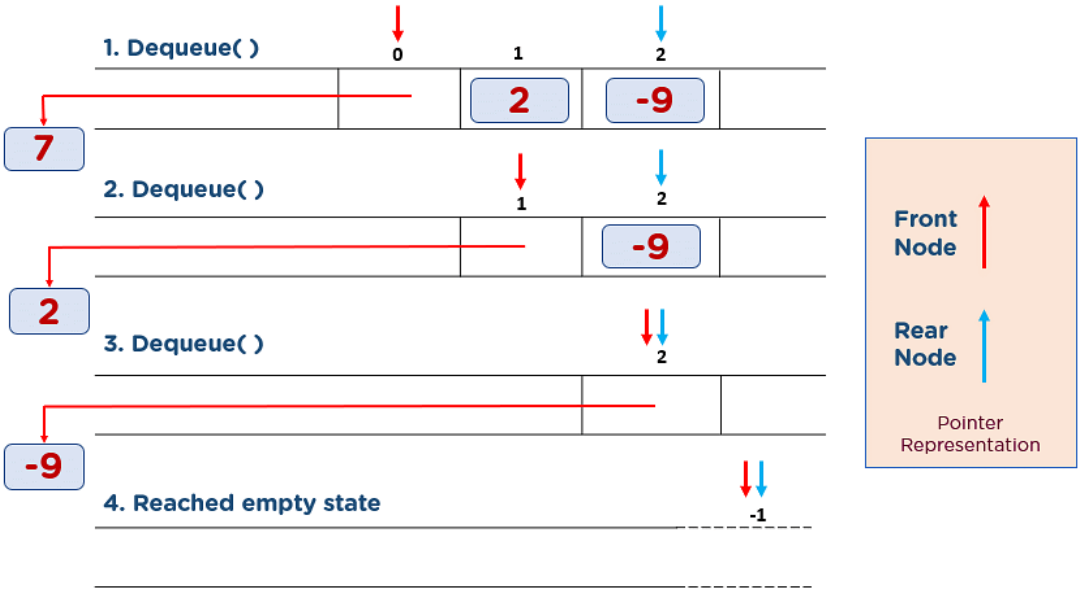
- **Step 1: Check if the queue is full.**
- **Step 2: If the queue is full, Overflow error.**
- **Step 3: If the queue is not full, increment the rear pointer to point to the next available empty space.**
- **Step 4: Add the data element to the queue location where the rear is pointing.**
- **Step 5: Here, you have successfully added 7, 2, and -9.**



Dequeue() Operation

Obtaining data from the queue comprises two subtasks: access the data where the front is pointing and remove the data after access. You should take the following steps to remove data from the queue -

- **Step 1: Check if the queue is empty.**
- **Step 2: If the queue is empty, Underflow error.**
- **Step 3: If the queue is not empty, access the data where the front pointer is pointing.**
- **Step 4: Increment front pointer to point to the next available data element.**
- **Step 5: Here, you have removed 7, 2, and -9 from the queue data structure.**



Now that you have dealt with the operations that allow manipulation of data entities, you will encounter supportive functions of the queues -

Peek() Operation

This function helps in extracting the data element where the front is pointing without removing it from the queue. The algorithm of Peek() function is as follows-

- **Step 1: Check if the queue is empty.**
- **Step 2: If the queue is empty, return "Queue is Empty."**
- **Step 3: If the queue is not empty, access the data where the front pointer is pointing.**
- **Step 4: Return data.**

isFull() Operation

This function checks if the rear pointer is reached at MAXSIZE to determine that the queue is full. The following steps are performed in the isFull() operation -

- **Step 1: Check if rear == MAXSIZE - 1.**
- **Step 2: If they are equal, return "Queue is Full."**
- **Step 3: If they are not equal, return "Queue is not Full."**

isNull() Operation

The algorithm of the isNull() operation is as follows -

- **Step 1: Check if the rear and front are pointing to null memory space, i.e., -1.**
- **Step 2: If they are pointing to -1, return "Queue is empty."**
- **Step 3: If they are not equal, return "Queue is not empty."**

Now that you have covered all the queue operations.

2.2 PROBLEM STATEMENT:

Implementing queues at petrol bunk where the 1st vehicle is to be filled with petrol/diesel and other vehicles has to wait.

CHAPTER-3

CODING AND RESULTS

3.1 Code:

```
        #include <stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
   printf("\n QUEUES OF VEHICLES IN PETROL BUNK \n\n");
      printf("1.Insert vehicle number to queue \n");
      printf("2.Delete vehicle from queue \n");
      printf("3.Display all vehicles of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
```

```
            display();
            break;
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue is full \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Insert the vehicle number in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue is empty\n");
        return ;
    }
    else
    {
        printf("vehicle deleted from queue is : %d\n", queue_array[front]);
```

```
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("The first vehicle is filled first then other has to wait to fill!!.. \n ");
        printf("the vehicle queue at petrol bunk  is : ");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /* End of display() */
```

OUTPUT:

main.c

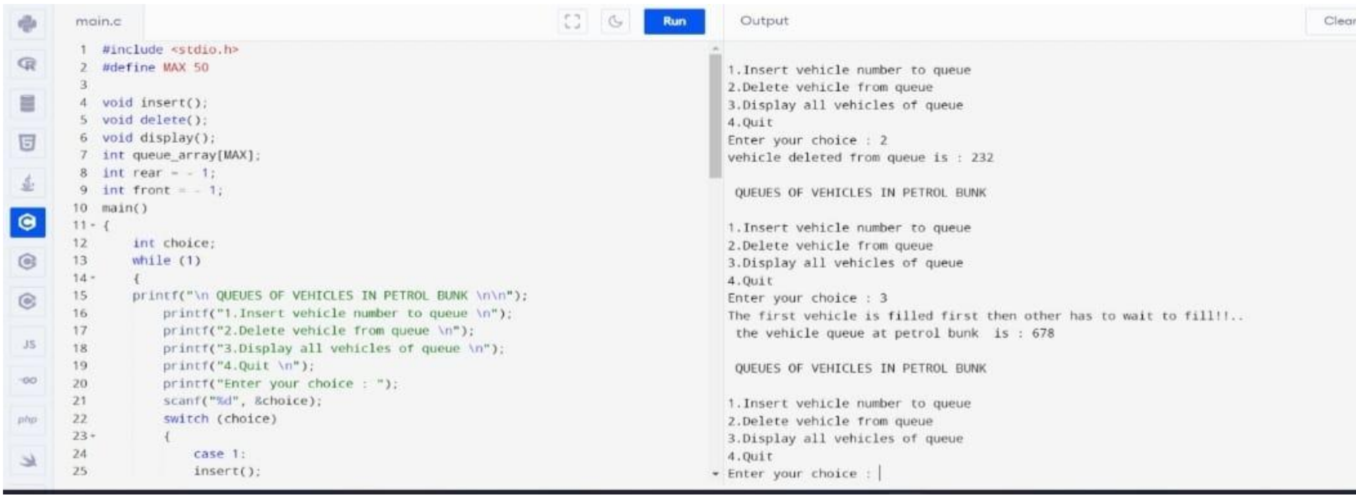
Run

```c
#include <stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
    printf("\n QUEUES OF VEHICLES IN PETROL BUNK \n\n");
        printf("1.Insert vehicle number to queue \n");
        printf("2.Delete vehicle from queue \n");
        printf("3.Display all vehicles of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
```

Output

Clear

```
/tmp/gqOKSyM5bR.o
QUEUES OF VEHICLES IN PETROL BUNK

1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice : 1
Insert the vehicle number in queue : 232
QUEUES OF VEHICLES IN PETROL BUNK

1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice : 1
Insert the vehicle number in queue : 678
QUEUES OF VEHICLES IN PETROL BUNK

1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice : 2
vehicle deleted from queue is : 232
```



main.c

Run

```c
#include <stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
    printf("\n QUEUES OF VEHICLES IN PETROL BUNK \n\n");
        printf("1.Insert vehicle number to queue \n");
        printf("2.Delete vehicle from queue \n");
        printf("3.Display all vehicles of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
```

Output

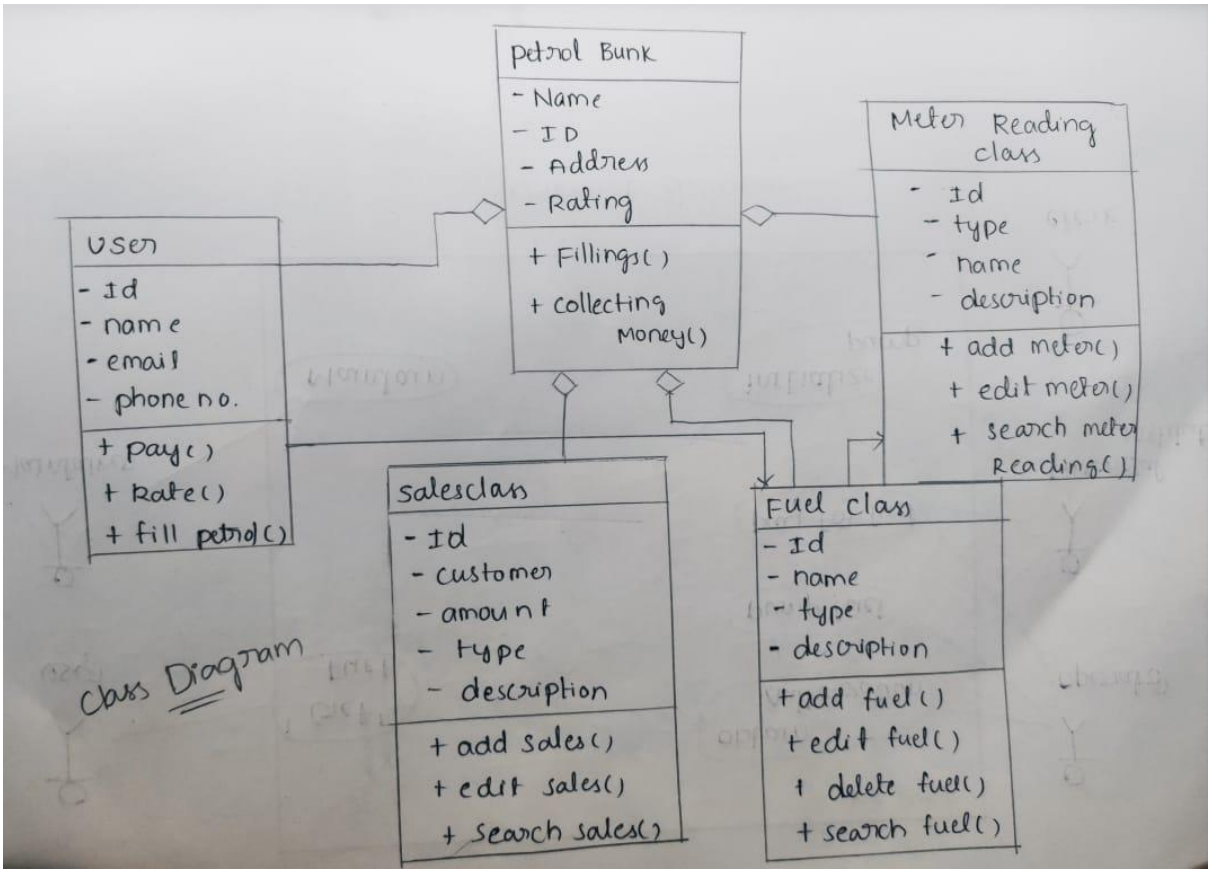
Clear

```
1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice : 2
vehicle deleted from queue is : 232

 QUEUES OF VEHICLES IN PETROL BUNK

1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice : 3
The first vehicle is filled first then other has to wait to fill!!..
 the vehicle queue at petrol bunk  is : 678

 QUEUES OF VEHICLES IN PETROL BUNK

1.Insert vehicle number to queue
2.Delete vehicle from queue
3.Display all vehicles of queue
4.Quit
Enter your choice :
```

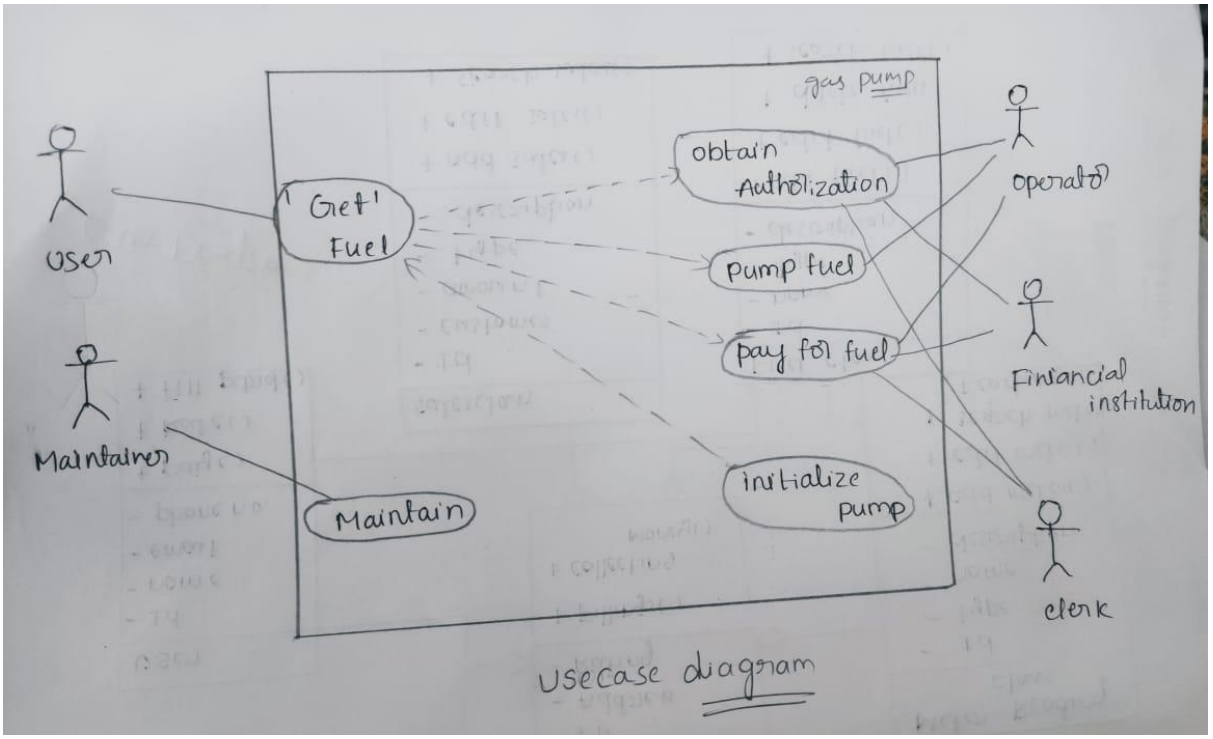
CHAPTER-4

4.U.M.L Diagrams

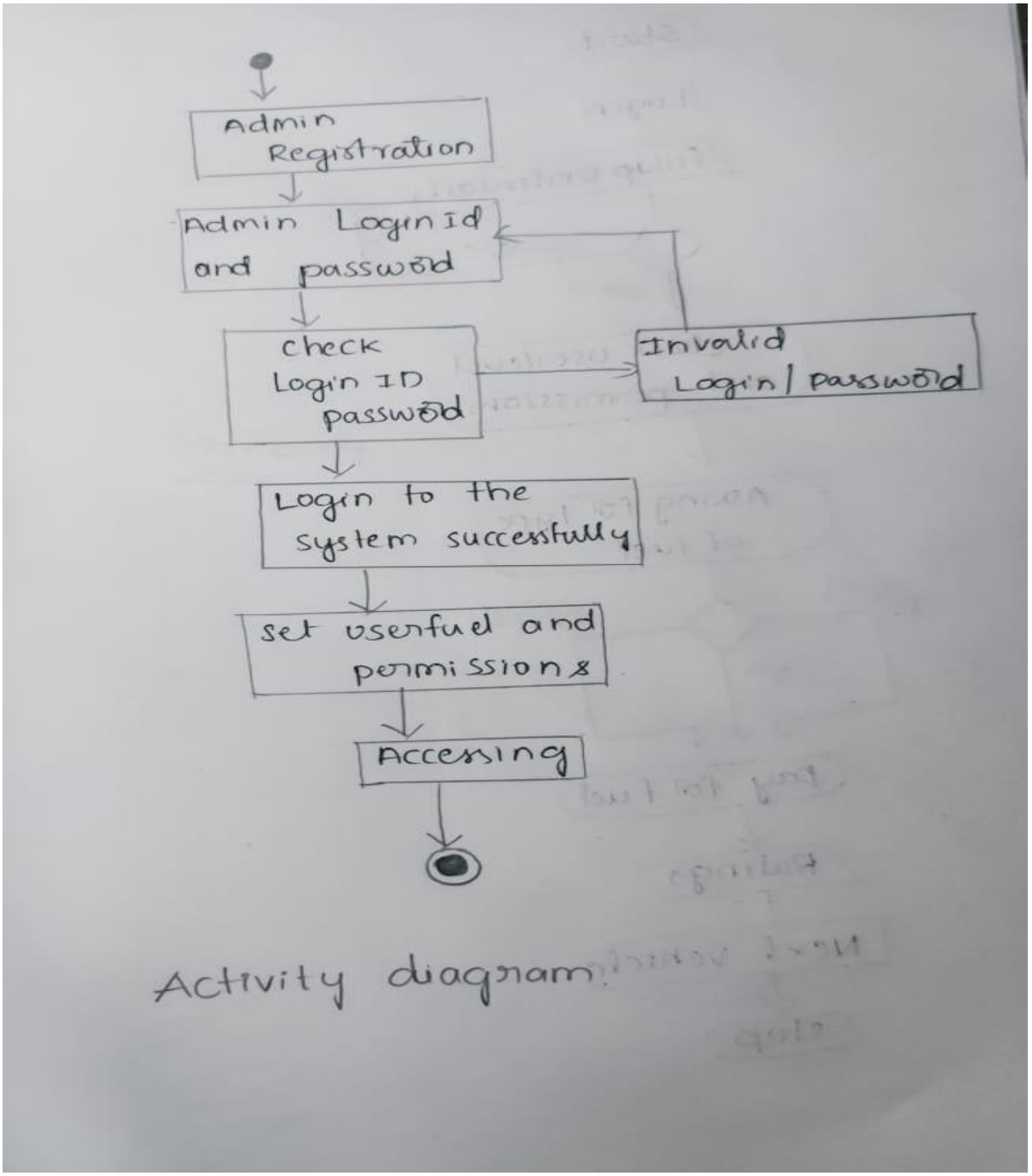
4.1Class diagram:



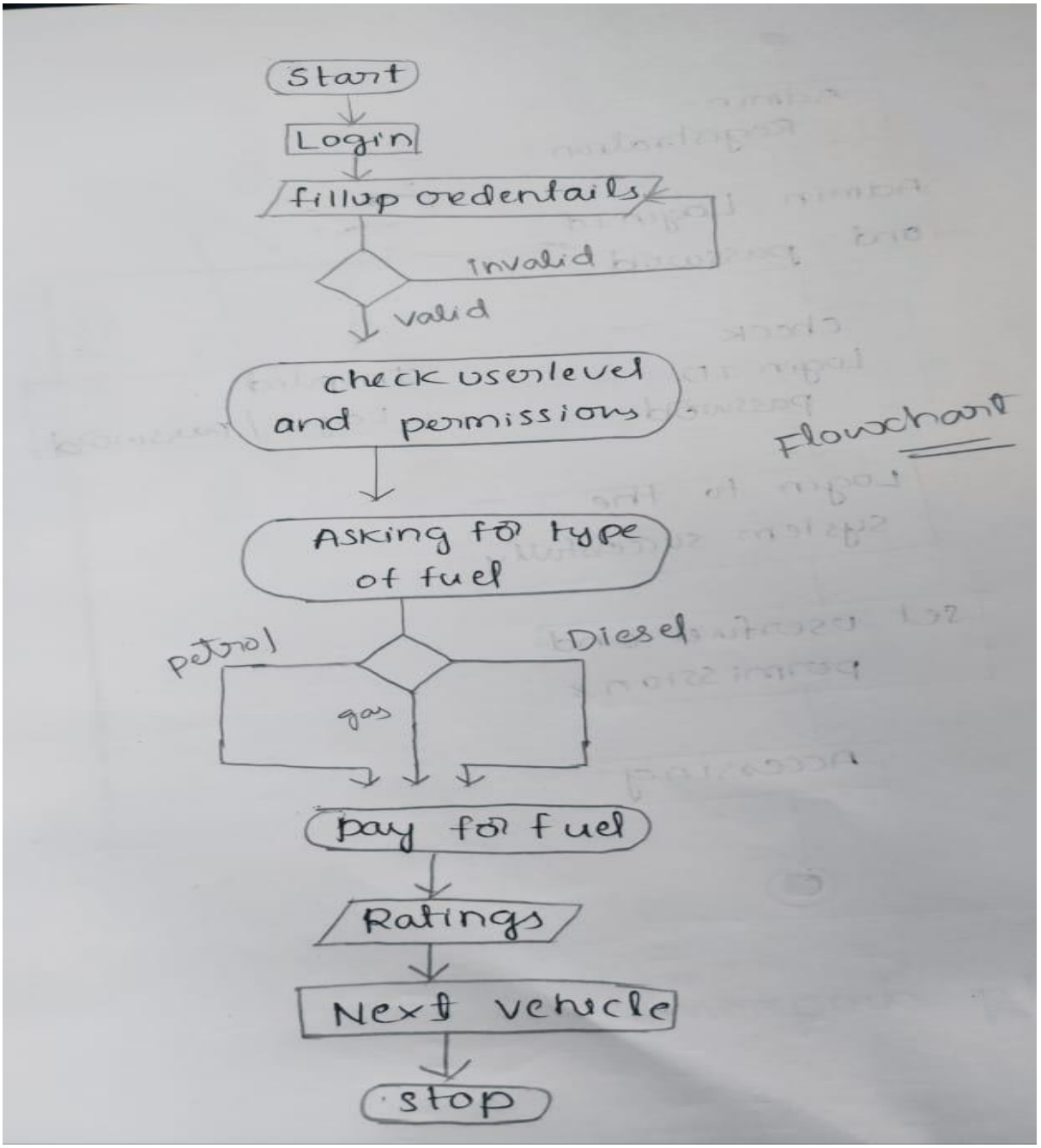
4.2 Usecase diagram:



4.3: Activity diagram:



4.4 Flowchart:



CHAPTER-5

CONCLUSION AND FUTURE WORK

5.1 Conclusion:

In computer science, a queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence. By convention, the end of the sequence at which elements are added is called the back, tail, or rear of the queue, and the end at which elements are removed is called the head or front of the queue, analogously to the words used when people line up to wait for goods or services.

The operation of adding an element to the rear of the queue is known as *enqueue*, and the operation of removing an element from the front is known as *dequeue*. Other operations may also be allowed, often including a *peek* or *front* operation that returns the value of the next element to be dequeued without dequeuing it.

The operations of a queue make it a first-in-first-out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. A queue is an example of a linear data structure, or more abstractly a sequential collection. Queues are common in computer programs, where they are implemented as data structures coupled with access routines, as an abstract data structure or in object-oriented languages as classes. Common implementations are circular buffers and linked lists.

Queues provide services in computer science, transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer. Another usage of queues is in the implementation of breadth-first search.

5.2 Future Work:

Are long lines avoidable? Some businesses would answer no, even though there are proven strategies for minimizing customer wait times.

That leads to another question: How much effort should businesses put into trying to shorten long lines? Sure, customers might feel less impatient, but is that reason enough to address the problem?

A **VIRTUAL QUEUE MANAGEMENT SYSTEM** provides relief and a better overall experience for customers. However, that's just part of what can be accomplished; there are a variety of benefits to be reaped from the best queuing solutions. Here are nine ways a **VIRTUAL QUEUE** makes life better for organizations and their customers.

1. **IMPROVED EMPLOYMENT EFFICIENCY:**

A long line may require employees to actually manage it—from trying to figure out who is next to calming down frustrated customers to interrupting their work to tell someone, "I'm helping this customer now; the line starts back there." Moreover, customers agitated by a long wait may be less focused and more short-tempered when their turn finally arrives.

A **queue systeme eliminates the time wastage distractions that addup over the course of the day managing customers,** ultimately allowing more customers to be seen and freeing up resources for other tasks. Happier customers also generally make for less frazzled employees, who then stay more focused on efficiency.

2.improved service quality:

Employees who can focus on one customer at a time and not worry about the throng of **people visibly waiting in the line** are better able to serve that one customer. They don't feel the need to rush the interaction and are more inclined to go the extra mile because there is less pressure to get to the next customer.

A queue management system can also collect information about customers and **their needs while they are waiting**. That intel becomes instantly available to the staff, who can tailor service or designate a specialist for each customer who reaches the front of the virtual queue.

3.better storewide sales:

In retail, a traditional, stand-in-line queue forces people to spend time waiting and disrupts the customer journey. The sight of a long line can drive customers to grab their one item, get in line, endure the wait, and then leave—if they stay at all. Although queues can be designed to encourage impulse purchases, the customer is still being denied access to the rest of the store.

Virtual queues upend this reality by giving customers access to browse the entire store while they wait their turn to be served, improving customer flow. Because people aren't confined to a certain space, the potential for **additional sales decreses**. Impulse purchases are still in play, but with a queue management system, those purchases aren't limited to a few nearby items.

4.REDUCED WASTE TIMES:

A virtual queue's combination of improved employee efficiency and shorter, more productive interactions decreases how much time customers spend waiting for their turn. Moreover, because they aren't tethered to a physical line and can move freely around (or

outside) the store, their **PERCEPTION OF HOW LONG THEY ARE WAITING ALSO DECREASES**. In other words, what was once a 20-minute wait may take only 15 minutes and can feel like 10.

5, HIGHER STAFF SATISFACTION:

Most employees prefer to interact with happier, less anxious customers, which a queue management solution can help with. Virtual queues also make employees more informed and more helpful when assisting their customers.

Both of these outcomes can lead to employees feeling more satisfied and connected to their jobs. This matters; **RESEARCH HAS SHOWN** that engaged employees are more productive, more highly appreciated by customers, and less likely to quit—and that businesses with engaged employees are 21 percent more profitable.

6.INCREASED CUSTOMER ROYALTY:

Customers appreciate not only great service but also little things that make their shopping experience and their lives easier. A queue management system may not reduce wait times significantly for every customer, but even just giving people a few minutes back or not making them stand in a line or sit in a crowded waiting area—particularly during a pandemic—shows that you care and improves the overall customer experience.

Loyal customers are repeat customers, and repeat customers are 90 percent more likely to make additional purchases, **ACCORDING TO RESEARCH FORM HUBSPOT** Virtual queues prioritize customers, who, in turn, may decide to prioritize you right back.

7. STREAM LINED COMMUNICATION:

We've all been somewhere—a meat counter, a post office, a DMV—where we had to take a number and wait for it to be called. Besides being highly impersonal, this setup dictates that the first communication between customer and staff involves shouting.

Despite the feeling of relief when your number is called, this inefficiency does nothing for the relationship being established between consumer and provider.

A digital queue management system establishes communication early and streamlines it during and even beyond the transaction. Via their smartphones, customers can be alerted as to their place in line and can provide information that will help staff serve them better.

8. BETTER CUSTOMER DATA:

The digital backbone of a virtual queue management system offers something traditional queues struggle with: pure data. Advanced reporting capabilities of the best solutions can measure wait times, peak and slow periods, customer satisfaction, and more. This easily gathered and retrievable data can help inform strategy as well as improve efficiency and, subsequently, the customer experience.

9. REDUCED OPERATIONAL CODES:

A queue management system benefits organizations by improving efficiency, eliminating the logistical requirements (e.g., barrier expenditures, floor space, customer traffic flow) of a physical queue, and generating valuable insight from user data. All these advantages reduce operational costs and, thus, boost the bottom line. This might be the most powerful argument for adopting a virtual queue.

A pattern emerges with all these benefits: They tend to build upon each other. For example, happier customers make happier employees, who become more productive, which reduces costs—and the data can back up the results. Qtrac can help you realize these benefits and more with our industry-leading queue management solutions.