Date _____ Page _____

Day - 142

Queue

⇒ Queue is like line of people.

⇒ Queue is used in our messaging world i.e. in whatsapp, etc.

⇒ Queue is also used in our OS.

Operations of Queue

1. Push: Insert the element. It always done in the end.
2. Pop: Delete the element. It always done from start.
3. Front: Return the top element.
4. IsFull: Return 1 if queue is full otherwise 0.
5. IsEmpty: Return 1 if queue is empty otherwise 0.

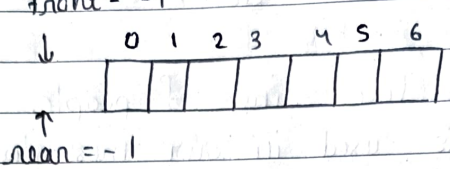
⇒ Queue is also called FIFO (First In First Out)

⇒ Implementation

Queue can be implemented by using array & linked list.

Date _____ Page _____

By using array



⇉ We will require two variable front & rear.

⇉ Front is used for pop & Rear is used for push.

⇉ When front & rear both are -1 then we will increase both by 1.

⇉ When front becomes greater than rear, update front & rear both to -1.

⇉ Because now our queue is empty.

Code

```
class Queue{
    int *arr;
    int front, rear;
    int size;
    public:
    Queue( int n){
        arr = new int[n];
        front = -1, rear = -1;
        size = n;
    }
```

Date

Page

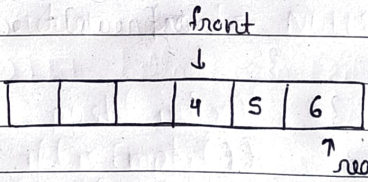
~~void push~~

```
bool isEmpty(){
  return front == -1;
}
bool isFull(){
    return rear == size - 1;
}

void push(int x){
    if( isEmpty()){
        front = rear = 0;
        arr[0] = x;
    }
    else if( isFull()){
        cout<< "Queue Overflow";
        return;
    }
    else{
        rear = rear+1;
        arr[rear] = x;
    }
};

void pop(){
    if( isEmpty()){
        cout<< "Queue Underflow";
        return;
    }
```

Date _____ Page _____

```
        else{
            if(front == rear){
                front = rear = -1;
            }
            else
                front = front+1;
            }
        }
    }
    int start(){
        if( isEmpty()){
            cout<<" Queue is Empty";
            return -1;
        else
            return arr[front];
        }
}
```

⇉ Now, suppose, our queue is like —



⇉ If we will try to insert an element into this queue by using above implementation then it will give queue overflow but we can see that three spaces are available in our queue.

Date _____ Page _____

⇉ So, to solve this problem, we will use circular queue.

⇉ So, in this, we will push element in the start too.

⇉ So, in circular queue, condition for full becomes →

(rear+1) % size == front

⇉ For push

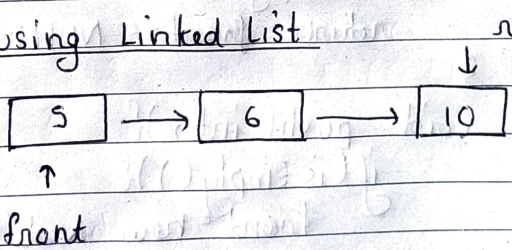
rear = (rear+1) % size;

⇉ For pop

front = (front+1) % size;

⇉ We used circular queue to proper utilize our queue.

By using Linked List



Code

Date Page

```
class Node{
    public:
    int data;
    Node * next;

    Node (int value){
        data = value;
        next = NULL;
    }
};

class Queue{
    Node * front;
    Node * rear;
    public:
    Queue(){
        front = rear = NULL;
    }
    bool isEmpty(){
        return front == NULL;
    }
    void push(int x){
        if(isEmpty()){
            front = new Node(x);
            rear = front;
            return;
        }
```

Date _____ Page _____

```
        else{
            rear -> next = new Node(x);
            rear = rear -> next;
        }
    }
    void pop(){
        if( isEmpty()){
            cout << " Queue Underflow ";
            return;
        }
        else{
            Node *temp = front;
            front = front -> next;
            delete temp;
        }
    }

    int start(){
        if( isEmpty()){
            cout << " Queue is Empty";
            return -1;
        }
        else{
            return front -> data;
        }
    }
}
```

Date Page

⇒ So, we can use queue without implementing by using standard template library.

Operations

1. Push.
2. Pop.
3. Size.
4. Front.
5. Empty.