Date _____ Page _____

Day - 147

Dequeue

⇉ As we know, in stack, we push & pop from the end of the stack.

⇉ And in queue, we do push in the end & pop from start.

⇉ Now, Dequeue comes into the picture.

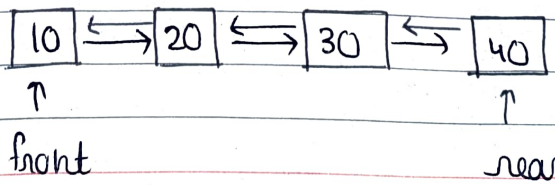
⇉ In dequeue, we can do both push & pop from front & back.

⇉ So, operations of Dequeue:

1. Push_front.
2. Push_back.
3. Pop_front.
4. Pop_back.
5. start.
6. End.

⇉ It can be implemented by using array & Linked List.

⇉ Implementation by using LL



Date _____ Page _____

⇉ We will use Doubly LL.

⇉ When only one element left i.e front == rear then —
for pop_front, we have to make rear NULL & for pop_back, we have to make front NULL.

Code

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

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

class Dequeue{
    Node *front, *rear;
    public:
    Dequeue(){
        front = rear = NULL;
    }
```

Date _____ Page _____

```
push_front(int x){
    if(front == NULL){
        front = rear = new Node(x);
        return;
    }
    else{
        Node *temp = new Node(x);
        temp->next = front;
        front->prev = temp;
        front = temp;
    }
}

push_back(int x){
    if(front == NULL){
        front = rear = new Node(x);
        return;
    }
    else{
        Node *temp = new Node(x);
        rear->next = temp;
        temp->prev = rear;
        rear = temp;
    }
}
```

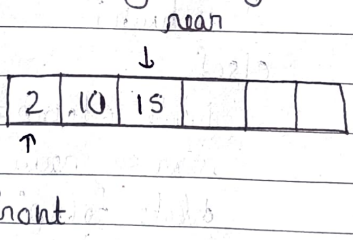
Date _____ Page _____

```
void pop_front(){
    if (front == NULL)
        return;
    else{
        Node *temp = front;
        front = front->next;
        delete temp;
        if(front)
            front->prev = NULL;
        else
            rear = NULL;
    }
}

void pop_back(){
    if(front == NULL)
        return;
    else{
        Node *temp = rear;
        rear = rear->prev;
        delete temp;
        if(rear)
            rear->next = NULL;
        else
            front = NULL;
    }
}
```

Date _____ Page _____

```
int start(){
    if(front == NULL){
        return -1;
    else{
        return front->data;
    }

int end(){
    if(front == NULL)
        return -1;
    else
        return rear->data;
    }
}
```

⇒ Implementation by using Array:



⇒ We will use circular array.

For full

(rear + 1) % size == front;

For front

(front - 1 + size) % size;

Date _____ Page _____

for rear

(rear + 1) % size;

Code

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

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

    void push_front(int x){
        if(isEmpty()){
            front = rear = 0;
            arr[0] = x;
            return;
        }
```

Date _____ Page _____

```
    else if( isFull() ){
        return;
    }
    else{
        front = (front - 1 + size) % size;
        arr[front] = x;
        return;
    }
}

void push_back(int x){
    if( isEmpty()){
        front = rear = 0;
        arr[0] = x;
    }
    else if( isFull())
        return;
    else{
        rear = (rear + 1) % size;
        arr[rear] = x;
        return;
    }
}

void pop_front(){
    if (isEmpty())
        return;
```

Date _____ Page _____

```
        else{
            if( front == rear )
                front = rear = -1;
            else
                front = (front+1) %size;
        }
    }

    void pop_back (){
        if( isEmpty() )
            return;
        else{
            if( front == rear )
                front = rear = -1;
            else
                rear = (rear -1 +size) %size;
        }
    }

    int start (){
        if( isEmpty() )
            return -1;
        else{
            return arr[front];
    }
```

Date _____ Page _____

* Deque in STL:

⇒ deque <int> d;
(Initialization)

Operations:

⇒ Push_front.
⇒ Push_back.
⇒ Pop_front.
⇒ Pop_back.
⇒ front.
⇒ back.

⇒ We can implement stack & queue by using Deque.

⇒ Deque have —