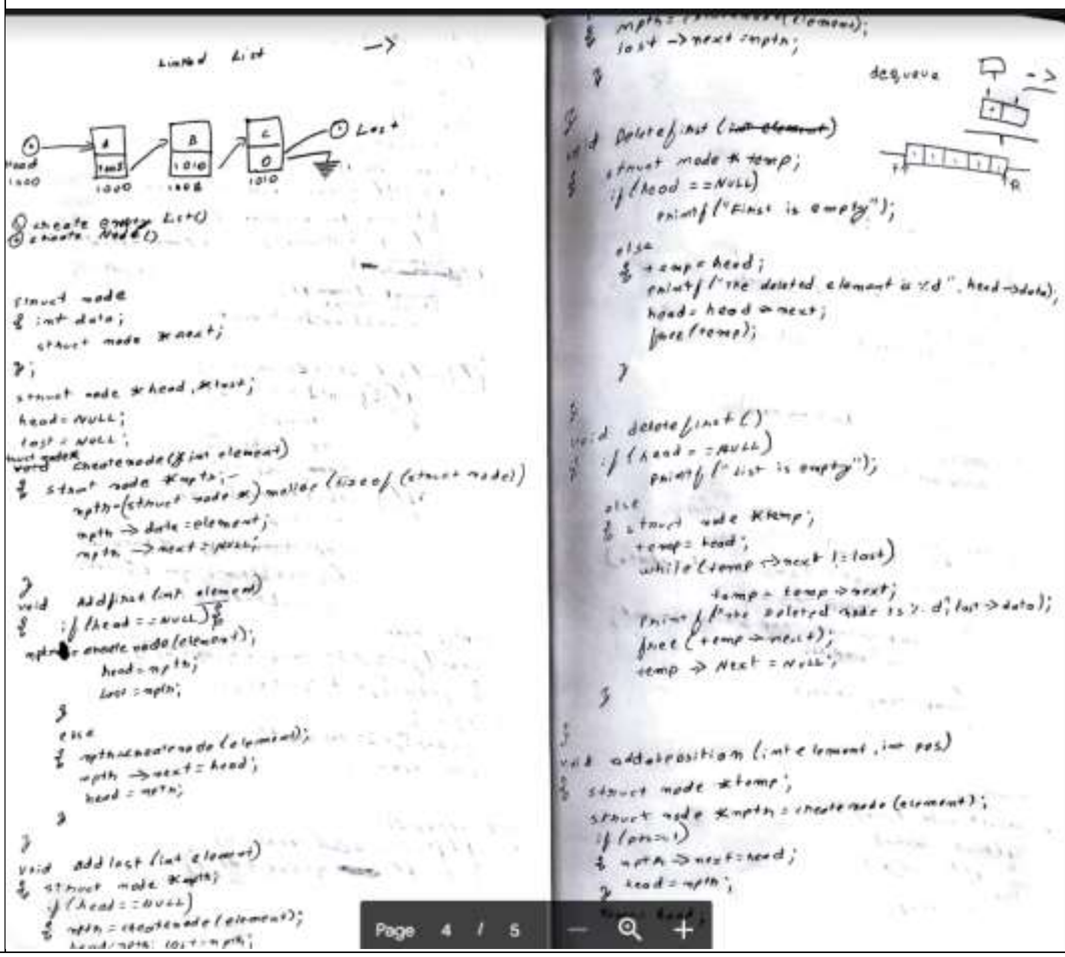
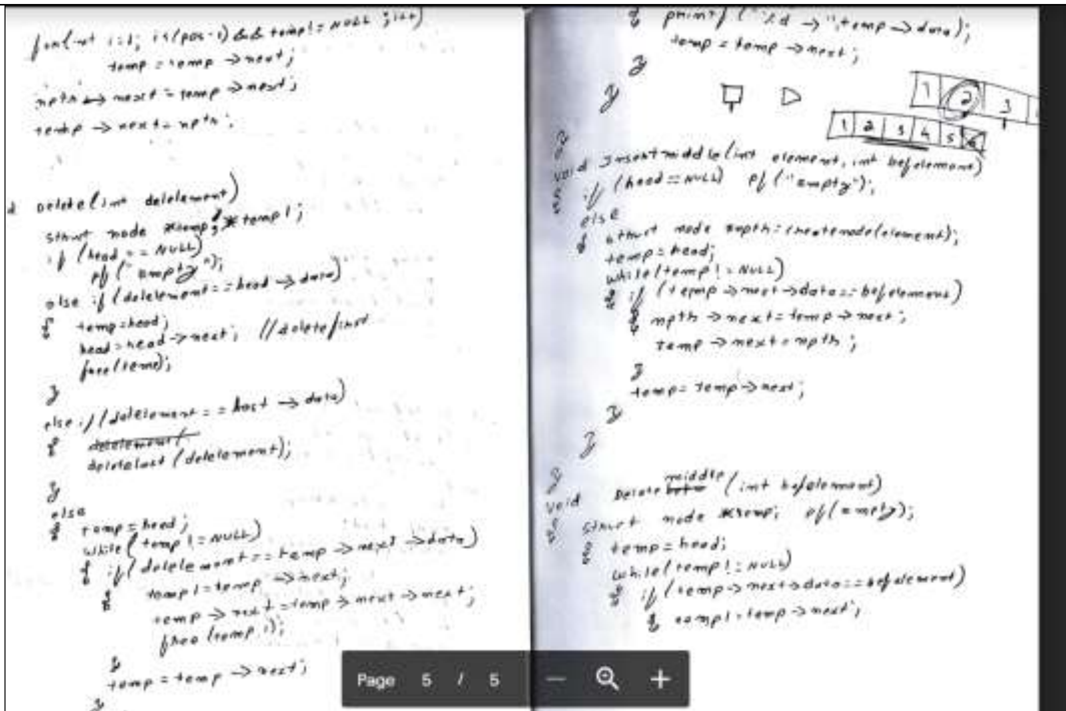
Internal Assessment Test II–Dec 2022-Solution

Sub:	Data Structures and Applications					SubCode:	21CS32	Branch:	ISE
Date:	26.12.2022	Duration:	90min's	MaxMarks:	50	Sem/Sec:	III A, B &C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Give the node structure of a single linked list of integers and write the C function to perform the following operation, a. create list b. Insert node at the end c. Delete the first node d. Display all the nodes	10	CO3	L2

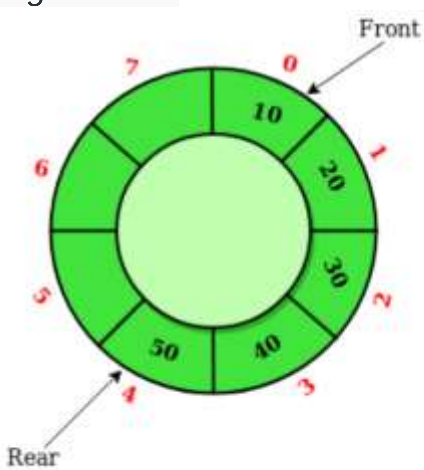
Answer:

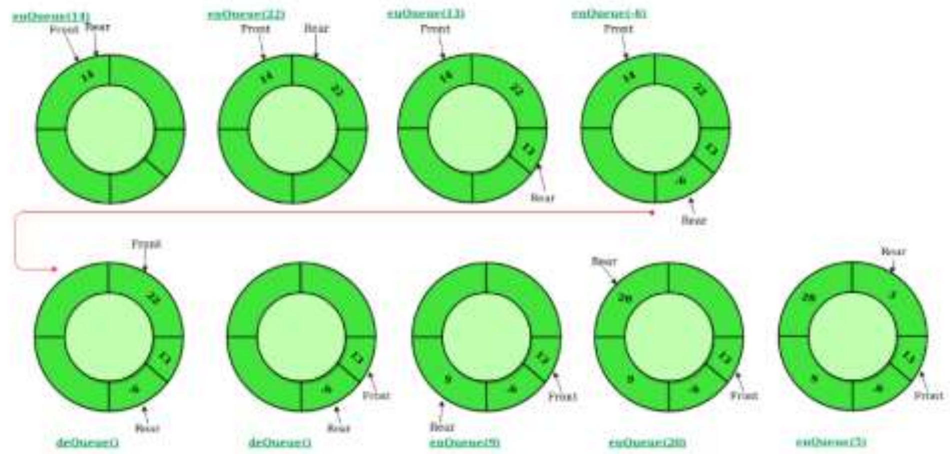




```
for(int i=1; i<(pos-1) && temp!=NULL; i++)
    temp = temp->next;
nptr->next = temp->next;
temp->next = nptr;

void delete(int delelement)
  struct node *temp, *temp1;
  if(head == NULL)
     pf("empty");
  else if(delelement == head->data)
  { temp=head;
    head=head->next;   //delete first
    free(temp);
  }
  else if(delelement == last->data)
  { deletelast(delelement);
  }
  else
  { temp=head;
    while(temp!=NULL)
    { if(delelement == temp->next->data)
      { temp1=temp->next;
        temp->next=temp->next->next;
        free(temp1);
      }
      temp=temp->next;
    }

{ printf("%d ->", temp->data);
  temp = temp->next;
}
}

void insertmiddle(int element, int befelement)
{ if(head==NULL)  pf("empty");
  else
  { struct node *nptr=createnode(element);
    temp=head;
    while(temp!=NULL)
    { if(temp->next->data==befelement)
      { nptr->next=temp->next;
        temp->next=nptr;
      }
      temp=temp->next;
    }
  }
}

void Deletemiddle(int befelement)
{ struct node *temp1;   pf(empty);
  temp=head;
  while(temp!=NULL)
  { if(temp->next->data==befelement)
    { temp1=temp->next;
```

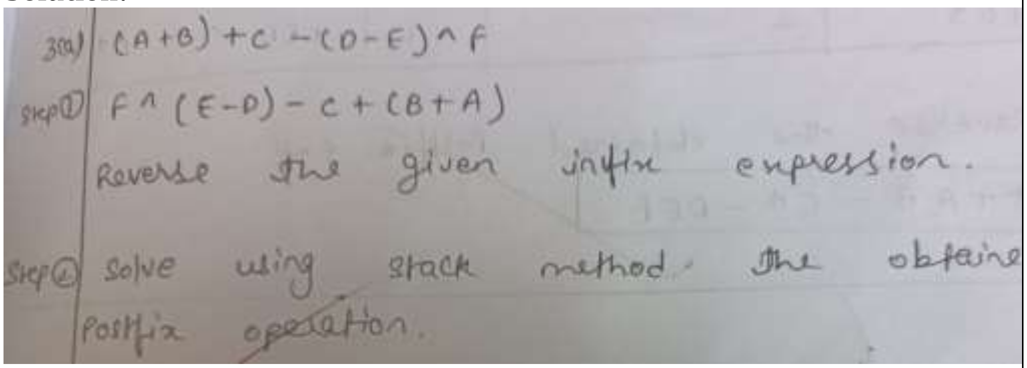
Page 5 / 5

2	Explain in detail the various operations of a Circular queue with proper examples and the C functions. **Solution:** *A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.*  n a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.	10	CO2	L1



```
#include<stdio.h>
#define MAX 5
int count=0,front=0,rear=-1;
int Queue[MAX];
void Append(int item)
{
if(count==MAX)
{
Printf("QUEUE is full:");
}
else
{
rear=(rear+1)%MAX;
queue[rear]=item
}
}
Void serve()
{
if(count==0)
printf("Queue is empty");
else
{
int item=queue[front];
front=(front+1)%MAX;
return item;
}
}
Void Display()
{
if(count==0)
printf("queue is emty");
else
{
 int j=front;
 for(int i=0;i<count;i++)
{
printf("%d\t", queue[j]);
j=(j+1)%MAX;
```

	} } }			
3a)	Convert the following Infix expression to prefix expression using stack method along with clear steps. Expression: (A+B)+C-(D-E)^F **Solution:**	5	CO2	L3

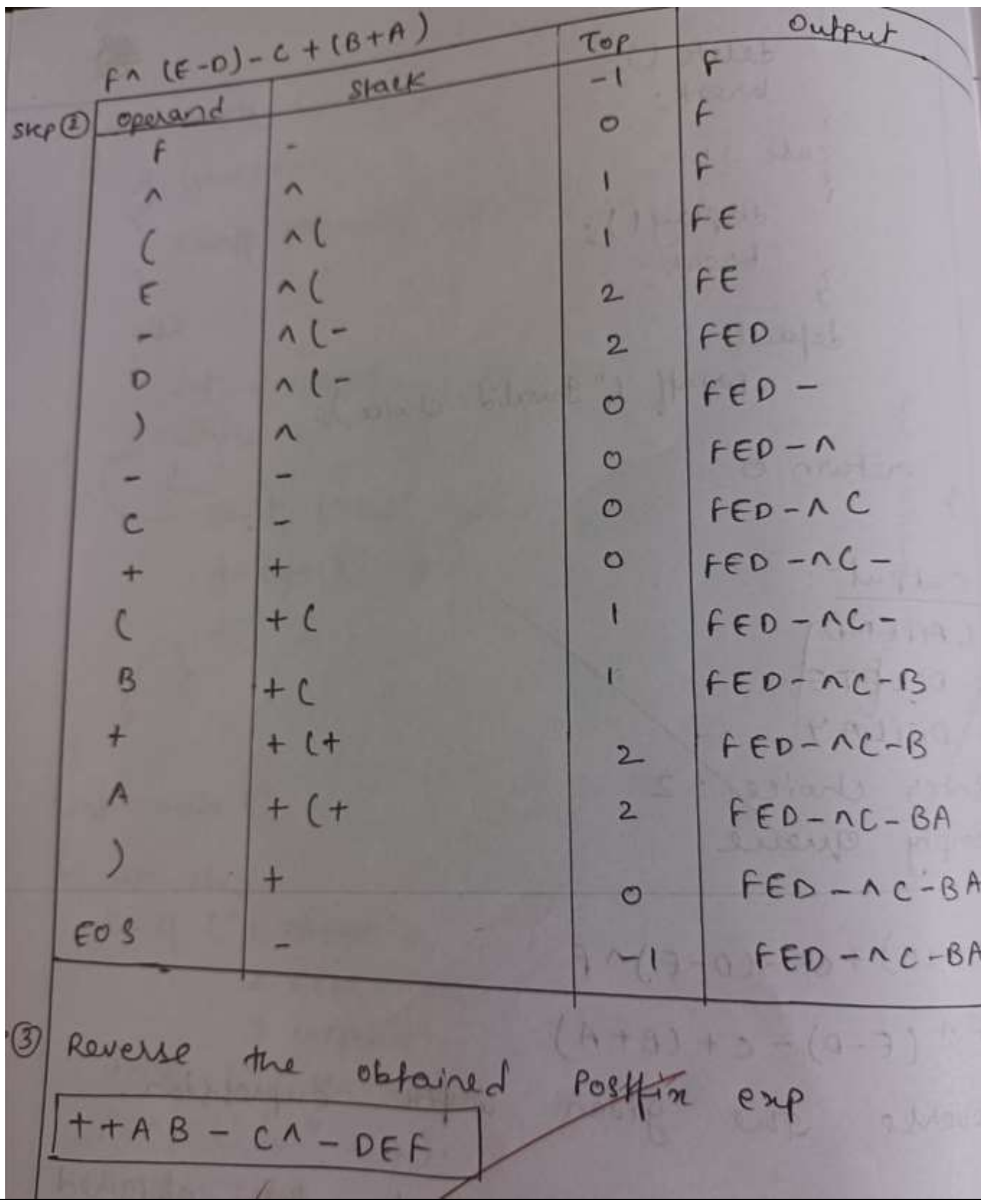


3(a) (A+B)+C-(D-E)^F

Step ① F^(E-D)-C+(B+A)

Reverse the given infix expression.

Step ② Solve using stack method the obtained postfix operation.



F^(E-D)-C+(B+A)

Step ②

Operand	Stack	Top	Output
F	-	-1	F
^	^	0	F
(	^(	1	F
E	^(	1	FE
-	^(-	2	FE
D	^(-	2	FED
)	^	0	FED -
-	-	0	FED -^
C	-	0	FED -^ C
+	+	0	FED -^C -
(	+(	1	FED -^C -
B	+(	1	FED -^C -B
+	+(+	2	FED -^C -B
A	+(+	2	FED -^C - BA
)	+	0	FED -^C -BA
EOS	-	-1	FED -^C -BA

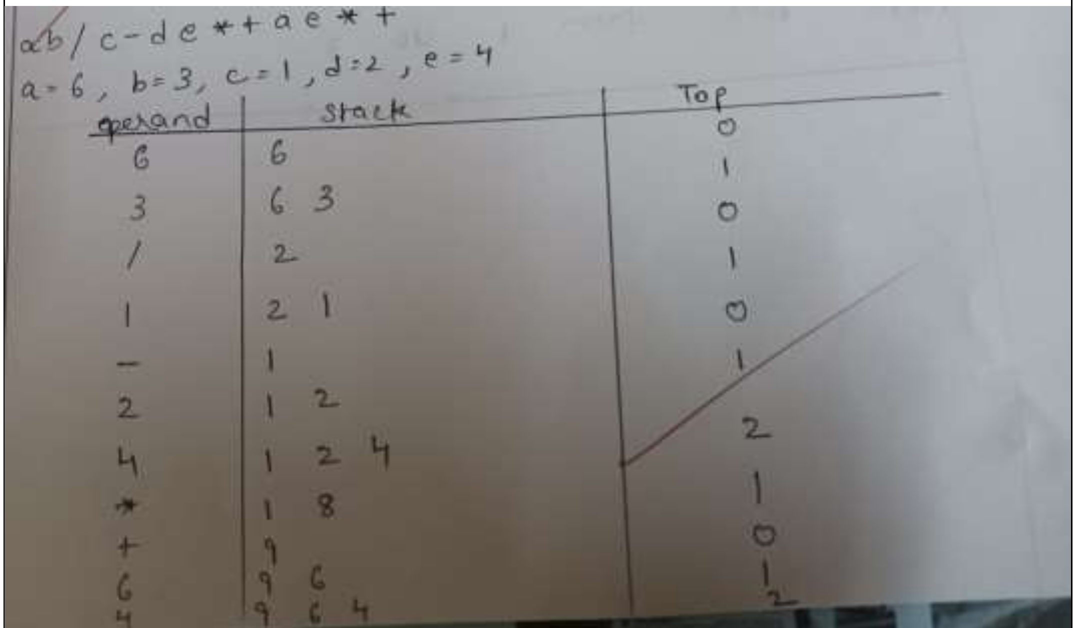
③ Reverse the obtained Postfix exp

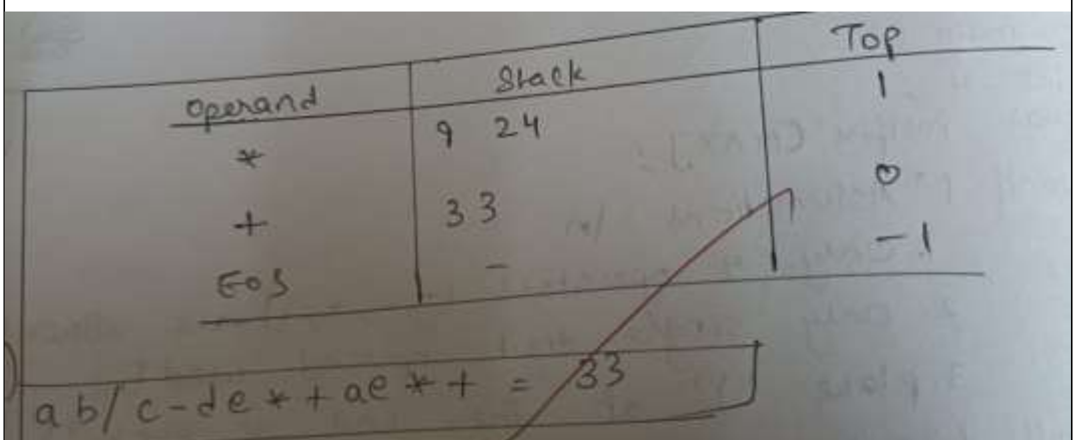
++AB-C^-DEF

3b)	Write a function to evaluate the postfix expression and trace the same for the expression using stack.	5	CO2	L3

Expression: ab/c-de*+ae*+, where a=6,b=3,c=1,d=2,e=4

Solution:





4	What is the drawback of linear queue how it overcome in circular queue? Explain with neat diagram along with C functions.	10	CO2	L2

Solution:

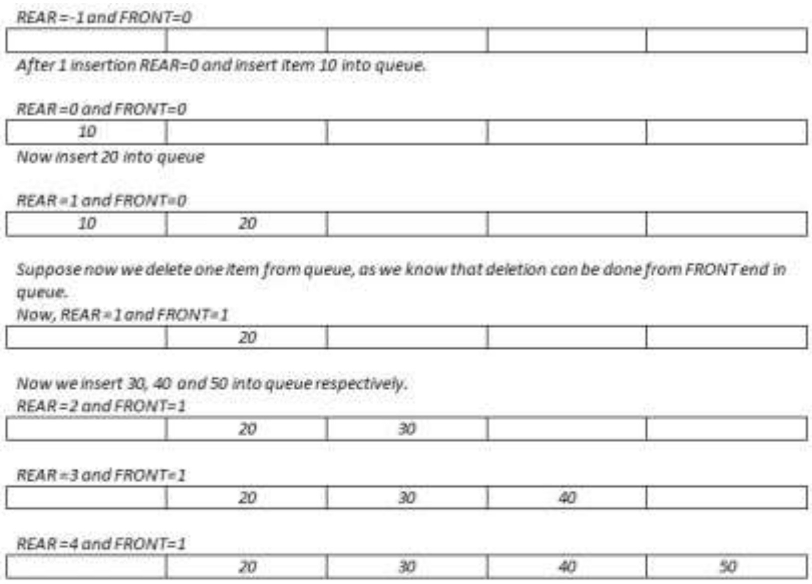
Drawback of Linear Queue

The linear queue suffers from serious drawback that performing some operations, we can not insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we can not insert items into queue.

Operations of Linear Queue:

In Array implementation **FRONT** pointer initialized with **0** and **REAR** initialized with **-1**.

Consider the implementation :- If there is 5 items in a Queue




```
#include <stdio.h>

#define MAX 5

//Declaration of Queue
typedef struct queue
{
    int front   ;
    int rear    ;
    int ele[MAX]    ;
}Queue;

//Intialze Queue
void init(Queue *q)
{
    q->rear  = -1;
    q->front =  0;
}

//To check Queue is full or not
int isFull(Queue *q)
{
    int full=0;

    if( q->rear == MAX -1)
        full = 1;

    return full;
}

//To check Queue is empty or not
int isEmpty(Queue *q)
{
    int empty=0;

    if( q->front == q->rear+1 )
        empty = 1;
```

```
    return empty;
}

//Insert item into queue
void insertQueue(Queue *q,int item)
{
    if( isFull(q) )
    {
        printf("\nQueue Overflow");
        return;
    }

    q->ele[++(q->rear)] = item;
    printf("\nInserted item : %d",item);
}

//Delete item from queue
int deleteQueue(Queue *q, int * item)
{
    if( isEmpty(q) )
    {
        printf("\nQueue Underflow");
        return -1;
    }
    *item = q->ele[(q->front)++];
    return 0;
}

int main()
{
    int item = 0;
    Queue q;

    init(&q);

    insertQueue(&q,10);
    insertQueue(&q,20);
    insertQueue(&q,30);
    insertQueue(&q,40);
    insertQueue(&q,50);
    insertQueue(&q,60);

    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);
    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);
    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);
    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);
    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);
    if( deleteQueue( &q, &item ) == 0 )
        printf("\nDeleted item : %d",item);

    printf("\n");
    return 0;
}
```

How to overcome Linear Queue Drawvacks

•We can solve this problem by joining the front and rear end of a queue to make the queue as a circular queue .

•Circular queue is a linear data structure. It follows FIFO principle.

	•In circular queue the last node is connected back to the first node to make a circle			
5	With diagrammatic representation explain the various operations of Dequeue along with the proper C functions.	10	CO2	L2

Solution:

Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends.

Operations on Deque:

Mainly the following four basic operations are performed on queue:

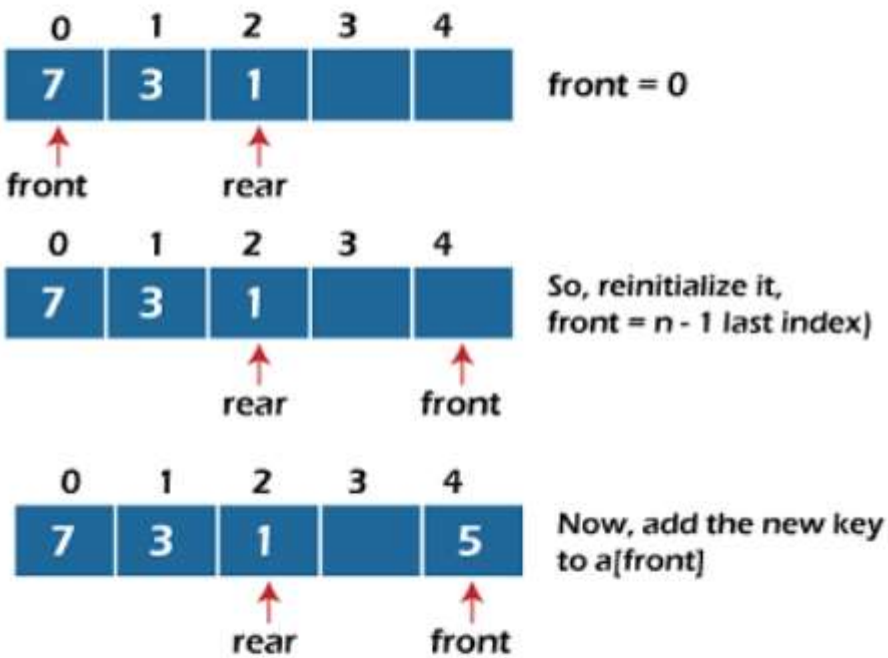
insertFront(): Adds an item at the front of Deque.

insertLast(): Adds an item at the rear of Deque.

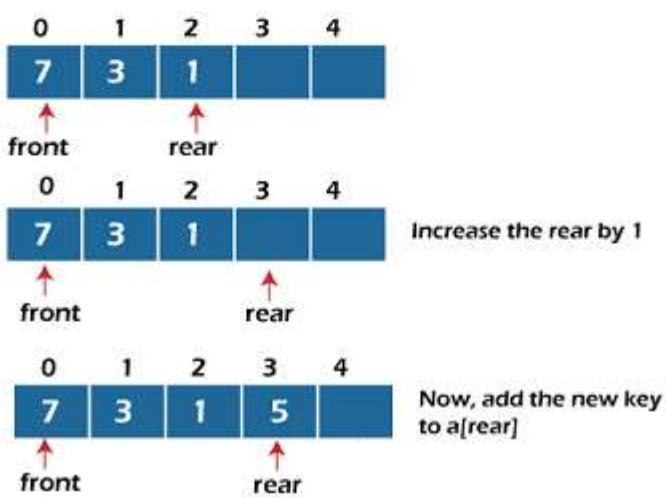
deleteFront(): Deletes an item from the front of Deque.

deleteLast(): Deletes an item from the rear of Deque.

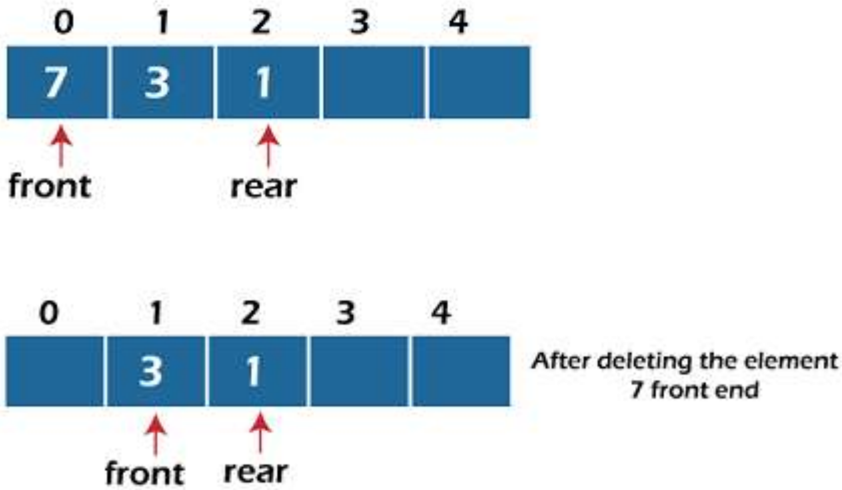
Insertion at the front end:



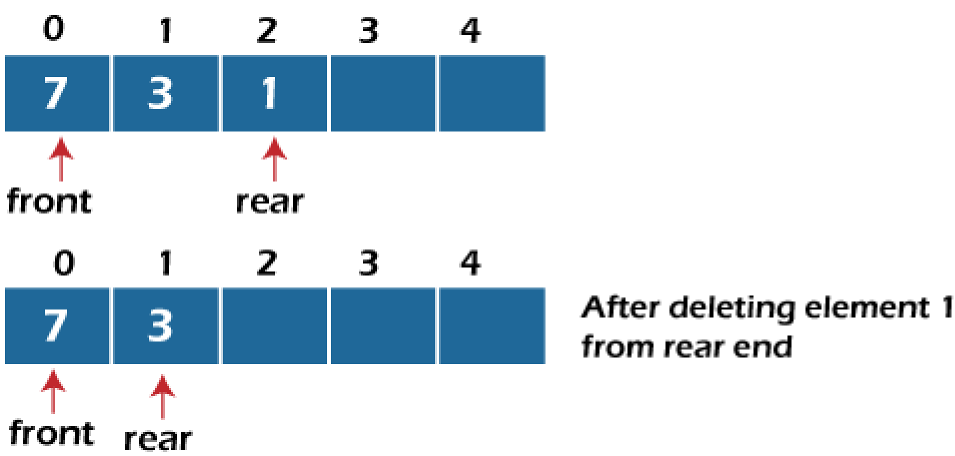
Insertion at the rear end



Deletion at the front end



Deletion at the rear end



Dequeue C fuction:

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int deque[MAX];

int front=-1,rear=-1;

void display();

void insertfront();

void insertrear();

void deletefront();

void deleterear();

int choice,item;
```

```
int main()
{
   while(1)
   {
     printf("\n Menu");
     printf("\n 1.Insert from Front:");
     printf("\n 2.Insert from Rear:");
     printf("\n 3.Delete from Front:");
     printf("\n 4.Delete from Rear:");
     printf("\n 5.Display");
     printf("\n 6.Exit \n");
     printf("Enter the choice \n");
     scanf("%d",&choice);
   switch(choice)
   {
   case 1:
   insertfront();
   break;

   case 2:
   insertrear();
   break;

   case 3:
   deletefront();
   break;
```

```
    case 4:
    deleterear();
    break;

    case 5:
    display();
    break;

    case 6:
    exit(0);

    default:
    printf("\n Invalid Choice");
    getch();
    break;
    }
    }
    return 0;
}

void insertfront()
{
    if(front==0)
    {
     printf("\n Queue is FULL");
    }
    else
```

```
    {
     front=front-1;
     printf("\n Enter a no:");
     scanf("%d",&item);
     deque[front]=item;
}
}
void insertrear()
{
    if(rear==MAX-1)
    {
    printf("\n Queue is FULL");
    }
    else
    {
    rear=rear+1;
    printf("\n Enter a no.:");
    scanf("%d",&item);
    deque[rear]=item;
}
}
void deletefront()
{
    if(front==MAX)
    {
      printf("\n Queue is EMPTY");
    }
```

```
   else
   {
   item=deque[front];
   front=front+1;
   printf("\n No. deleted is %d",item);
}
}
void deleterear()
{
   if(rear==-1)
   {
     printf("\n Queue is EMPTY");
   }
   else
   {
   item=deque[rear];
   rear=rear-1;
   printf("\n No. deleted is %d",item);
   }
}
void display()
{
int i;
printf("\n The Queue is::");
for(i=front;i<=rear;i++)
{
  printf("%d \t ", deque[i]);
```

	} }			
6	Elaborate on solving Towers of Honai problem using recursion along with examples, C functions and a recursion tree.	10	CO2	L2

Solution:

Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**. The objective of the puzzle is to move the entire stack to another rod (here considered **C**), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

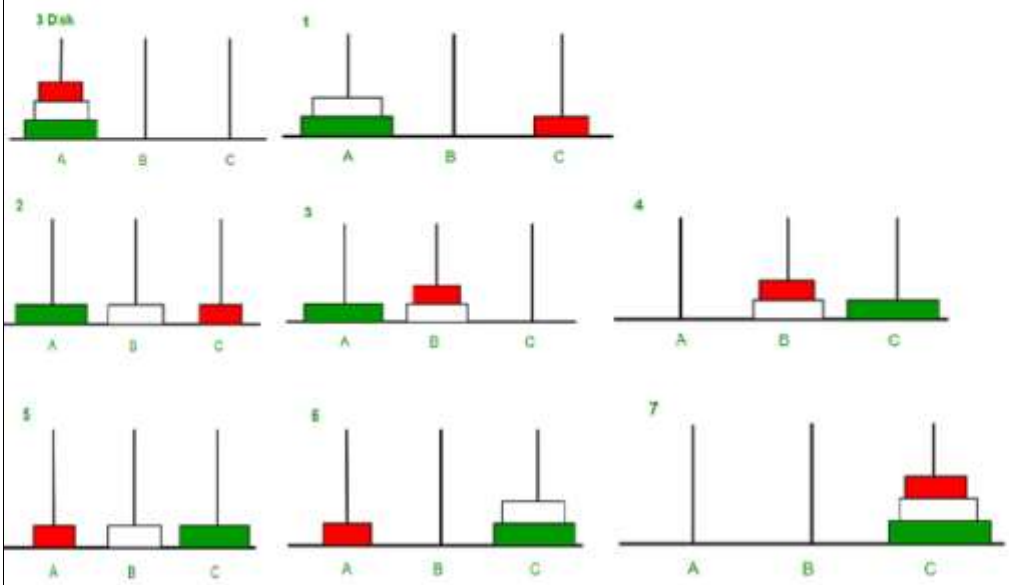
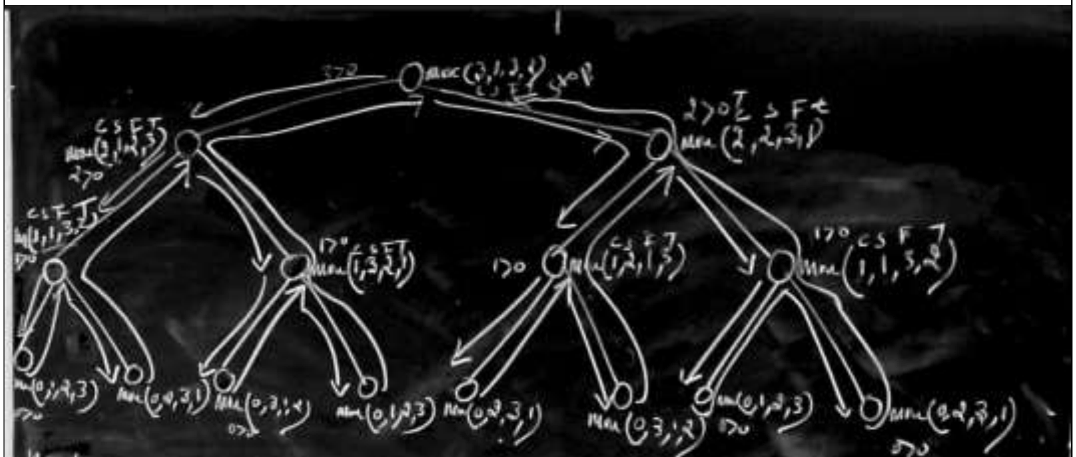


Image illustration for 3 disks

```
void towerOfHanoi(int n, char from_rod, char to_rod,
                  char aux_rod)
{
    if (n == 0) {
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod
         << " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

// Driver code
int main()
{
    int N = 3;

    // A, B and C are names of rods
    towerOfHanoi(N, 'A', 'C', 'B');
    return 0;
```

	} **Recursion Tree:** 			

Faculty Signature CCI Signature HOD Signature