a. Implement the following operations over Double Ended Queue (DEQUE).

i. Insert elements from left.
ii. Insert elements from right.
iii. Delete elements from left.
iv. Delete elements from right.

Program –

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30

typedef struct dequeue
{
     int data[MAX];
     int rear, front;
} dequeue;

void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void enqueueR(dequeue *p, int x);
void enqueueF(dequeue *p, int x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);

void main()
{
     int i, x, op, n;
     dequeue q;

     initialize(&q);

     do
     {

     printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\
n4.Delete(rear)\n5.Delete(front)");
          printf("\n6.Print\n7.Exit\n\nEnter your
choice:");
          scanf("%d", &op);

          switch (op)
          {
          case 1:
               printf("\nEnter number of elements:");
```

```
                    scanf("%d", &n);
                    initialize(&q);
                    printf("\nEnter the data:");

                    for (i = 0; i < n; i++)
                    {
                         scanf("%d", &x);
                         if (full(&q))
                         {
                              printf("\nQueue is full!!");
                              exit(0);
                         }
                         enqueueR(&q, x);
                    }
                    break;

               case 2:
                    printf("\nEnter element to be inserted:");
                    scanf("%d", &x);

                    if (full(&q))
                    {
                         printf("\nQueue is full!!");
                         exit(0);
                    }

                    enqueueR(&q, x);
                    break;

               case 3:
                    printf("\nEnter the element to be
     inserted:");
                    scanf("%d", &x);

                    if (full(&q))
                    {
                         printf("\nQueue is full!!");
                         exit(0);
                    }

                    enqueueF(&q, x);
                    break;

               case 4:
                    if (empty(&q))
                    {
```

```
                    printf("\nQueue is empty!!");
                    exit(0);
               }

               x = dequeueR(&q);
               printf("\nElement deleted is %d\n", x);
               break;

          case 5:
               if (empty(&q))
               {
                    printf("\nQueue is empty!!");
                    exit(0);
               }

               x = dequeueF(&q);
               printf("\nElement deleted is %d\n", x);
               break;

          case 6:
               print(&q);
               break;

          default:
               break;
          }
     } while (op != 7);
     return 0;
}

void initialize(dequeue *P)
{
     P->rear = -1;
     P->front = -1;
}

int empty(dequeue *P)
{
     if (P->rear == -1)
          return (1);

     return (0);
}

int full(dequeue *P)
{
     if ((P->rear + 1) % MAX == P->front)
          return (1);

     return (0);
}

void enqueueR(dequeue *P, int x)
{
     if (empty(P))
     {
          P->rear = 0;
          P->front = 0;
          P->data[0] = x;
     }
     else
     {
          P->rear = (P->rear + 1) % MAX;
          P->data[P->rear] = x;
     }
}

void enqueueF(dequeue *P, int x)
{
     if (empty(P))
     {
          P->rear = 0;
          P->front = 0;
          P->data[0] = x;
     }
     else
     {
          P->front = (P->front - 1 + MAX) % MAX;
          P->data[P->front] = x;
     }
}

int dequeueF(dequeue *P)
{
     int x;

     x = P->data[P->front];

     if (P->rear == P->front) //delete the last element
          initialize(P);
     else
          P->front = (P->front + 1) % MAX;
```

```
        return (x);
}

int dequeueR(dequeue *P)
{
        int x;

        x = P->data[P->rear];

        if (P->rear == P->front)
                initialize(P);
        else
                P->rear = (P->rear - 1 + MAX) % MAX;

        return (x);
}

void print(dequeue *P)
{
        if (empty(P))
        {
                printf("\nQueue is empty!!");
                exit(0);
        }

        int i;
        i = P->front;

        while (i != P->rear)
        {
                printf("\n%d", P->data[i]);
                i = (i + 1) % MAX;
        }

        printf("\n%d\n", P->data[P->rear]);
}
```

b. Implement the following operations over Priority Queue.

i. Insert elements

ii. Delete elements

Program –

```
#include<stdio.h>
#include<stdlib.h>

typedef struct pq
{
     int d, p;
     struct pq *next;
}prio;

void insert(prio **head, int item, int n)
{
     prio *loc, *locp, *newnode;
     loc = *head;
     while(loc != NULL && loc->p>=n)
     {
          locp = loc;
          loc = loc->next;
     }
     newnode = (prio*)malloc(sizeof(prio));
     newnode->d=item;
     newnode->p=n;
     if(*head == loc)
     {
          newnode->next=*head;
          *head=newnode;
     }
     else
     {
          newnode->next=locp->next;
          locp->next = newnode;
     }
     return;
}

void delete(prio **head)
{
     int item;
     prio *temp;
     if(*head==NULL)
     {
          printf("Underflow\n");
          return -1;
     }
     item = (*head)->d;
     temp = *head;
     *head = temp->next;
     temp->next=NULL;
     free(temp);
     return item;
}

int main()
{
     prio *head = NULL;
    int element, n, num;

    do
    {
        printf("\nOperations to be performed:\n");
        printf("1. Insertion\n");
        printf("2. Deletion\n");
        printf("3. Exit.\n");

        printf("\nEnter the operation (1-3): ");
        scanf("%d", &num);

        if(num == 1)
        {
            printf("\nEnter the element to insert: ");
            scanf("%d", &element);
            printf("Enter the priority: ");
            scanf("%d", &n);
            insert(&head,item,n);
        }
        else if(num == 2)
        {
            n = delete(&head);
            if(n!=-1)
               printf("deleted: %d\n", n);
        }
    }while(num != 3);

    return 0;
}
```