Set A

a) Implement a priority queue library (PriorityQ.h) of integers using a static implementation of the queue and implementing the below two operations. Write a driver program that includes queue library and calls different queue operations.
1) Add an element with its priority into the queue.
2) Delete an element from queue according to its priority.

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

#define MAX 5

void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();

int pri_que[MAX];
int front, rear;

void main()
{
   int n, ch;

   printf("\n1 - Insert an element into queue");
   printf("\n2 - Delete an element from queue");
   printf("\n3 - Display queue elements");
   printf("\n4 - Exit");

   create();

   while (1)
   {
      printf("\nEnter your choice : ");
      scanf("%d", &ch);

      switch (ch)
```

```
        {
        case 1:
            printf("\nEnter value to be inserted : ");
            scanf("%d",&n);
            insert_by_priority(n);
            break;
        case 2:
            printf("\nEnter value to delete : ");
            scanf("%d",&n);
            delete_by_priority(n);
            break;
        case 3:
            display_pqueue();
            break;
                case 4:
            exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}

/* Function to create an empty priority queue */
void create()
{
    front = rear = -1;
}

/* Function to insert value into priority queue */
void insert_by_priority(int data)
{
    if (rear >= MAX - 1)
    {
        printf("\nQueue overflow no more elements can be inserted");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
```

```
        pri_que[rear] = data;
        return;
    }
    else
        check(data);
    rear++;
}

/* Function to check priority and place element */
void check(int data)
{
    int i,j;

    for (i = 0; i <= rear; i++)
    {
        if (data >= pri_que[i])
        {
            for (j = rear + 1; j > i; j--)
            {
              pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}

/* Function to delete an element from queue */
void delete_by_priority(int data)
{
    int i;

    if ((front==-1) && (rear==-1))
    {
        printf("\nQueue is empty no elements to delete");
        return;
    }

    for (i = 0; i <= rear; i++)
```

```
    {
        if (data == pri_que[i])
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }

        pri_que[i] = -99;
        rear--;

        if (rear == -1)
            front = -1;
        return;
        }
    }
    printf("\n%d not found in queue to delete", data);
}

/* Function to display queue elements */
void display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
{
        printf("\nQueue is empty");
        return;
    }

    for (; front <= rear; front++)
    {
        printf(" %d ", pri_que[front]);
    }

    front = 0;
}
```

b) A doubly ended queue allows additions and deletions from both the ends that is front and rear. Initially additions from the front will not be possible. To avoid this situation, the array can be treated as if it were circular. Implement a queue library (dstqueue.h) of integers using a static implementation of the circular queue and implementing the nine operations :
1)init(Q), 2) isempty(Q) 3) isFull(Q) 4)getFront(Q), 5)getRear(Q), 6)addFront(Q,x), 7)deleteFront(Q) 8) addRear(Q,x) 9)deleteRear(Q)

```
#include <stdio.h>
#define MAX 10
void addFront(int *, int, int *, int *);
void addRear(int *, int, int *, int *);
int delFront(int *, int *, int *);
int delRear(int *, int *, int *);
void display(int *);
int count(int *);

int main() {
  int arr[MAX];
  int front, rear, i, n;

  front = rear = -1;
  for (i = 0; i < MAX; i++)
    arr[i] = 0;

  addRear(arr, 5, &front, &rear);
  addFront(arr, 12, &front, &rear);
  addRear(arr, 11, &front, &rear);
  addFront(arr, 5, &front, &rear);
  addRear(arr, 6, &front, &rear);
  addFront(arr, 8, &front, &rear);

  printf("\nElements in a deque: ");
  display(arr);

  i = delFront(arr, &front, &rear);
  printf("\nremoved item: %d", i);

  printf("\nElements in a deque after deletion: ");
```

```
  display(arr);

  addRear(arr, 16, &front, &rear);
  addRear(arr, 7, &front, &rear);

  printf("\nElements in a deque after addition: ");
  display(arr);

  i = delRear(arr, &front, &rear);
  printf("\nremoved item: %d", i);

  printf("\nElements in a deque after deletion: ");
  display(arr);

  n = count(arr);
  printf("\nTotal number of elements in deque: %d", n);
}

void addFront(int *arr, int item, int *pfront, int *prear) {
  int i, k, c;

  if (*pfront == 0 && *prear == MAX - 1) {
    printf("\nDeque is full.\n");
    return;
  }

  if (*pfront == -1) {
    *pfront = *prear = 0;
    arr[*pfront] = item;
    return;
  }

  if (*prear != MAX - 1) {
    c = count(arr);
    k = *prear + 1;
    for (i = 1; i <= c; i++) {
      arr[k] = arr[k - 1];
      k--;
    }
    arr[k] = item;
```

```
    *pfront = k;
    (*prear)++;
  } else {
    (*pfront)--;
    arr[*pfront] = item;
  }
}

void addRear(int *arr, int item, int *pfront, int *prear) {
  int i, k;

  if (*pfront == 0 && *prear == MAX - 1) {
    printf("\nDeque is full.\n");
    return;
  }

  if (*pfront == -1) {
    *prear = *pfront = 0;
    arr[*prear] = item;
    return;
  }

  if (*prear == MAX - 1) {
    k = *pfront - 1;
    for (i = *pfront - 1; i < *prear; i++) {
      k = i;
      if (k == MAX - 1)
        arr[k] = 0;
      else
        arr[k] = arr[i + 1];
    }
    (*prear)--;
    (*pfront)--;
  }
  (*prear)++;
  arr[*prear] = item;
}

int delFront(int *arr, int *pfront, int *prear) {
  int item;
```

```
  if (*pfront == -1) {
    printf("\nDeque is empty.\n");
    return 0;
  }

  item = arr[*pfront];
  arr[*pfront] = 0;

  if (*pfront == *prear)
    *pfront = *prear = -1;
  else
    (*pfront)++;

  return item;
}

int delRear(int *arr, int *pfront, int *prear) {
  int item;

  if (*pfront == -1) {
    printf("\nDeque is empty.\n");
    return 0;
  }

  item = arr[*prear];
  arr[*prear] = 0;
  (*prear)--;
  if (*prear == -1)
    *pfront = -1;
  return item;
}

void display(int *arr) {
  int i;

  printf("\n front:  ");
  for (i = 0; i < MAX; i++)
    printf("  %d", arr[i]);
  printf("  :rear");
```

```
}

int count(int *arr) {
  int c = 0, i;

  for (i = 0; i < MAX; i++) {
    if (arr[i] != 0)
      c++;
  }
  return c;
```