Date __/__/__

Q Doubly linked list with primitive operations

a) Create a doubly linked list
b) Insert a new node to the left of the node
c) Delete the node based on a specific value
d) Delete the content of the list
e) Delete the Duplicates

```
# include <stdio.h>
# include <stdlib.h>
struct node
 {
      int info;
      struct node * rlink;
      struct node * llink;
 };
typedef struct node * NODE;
NODE getnode ()
 {
    NODE x;
   x = (NODE) malloc (size of (struct node));
   if (x == NULL)
   {
    printf ("mem full\n");
    exit (0);
    }
 return x;
}
void freenode (NODE x)
 {
   free (x);
 }
```

Page No.

```
NODE dinsert-front(int item , NODE head)
{
 NODE temp, cur;
 temp = getnode();
 temp-> info = item;
  cur = head ->rlink;
   head -> rlink = temp;
   temp -> llink = head;
   temp -> rlink = cur;
   cur -> llink = temp;
  return head;
}
NODE dinsert_rear(int item, NODE head)
 {
  NODE temp , cur;
  temp = getnode();
   temp -> info = item;
   cur = head ->llink;
  head -> llink = temp;
  temp -> rlink = head;
  temp -> llink = cur;
  cur -> rlink = temp;
   return head;
 }
NODE ddelete-front(NODE head)
{
   NODE cur, next;
   if (head-> rlink == head)
   {
        printf(" list empty \n");
        return head;
   }
```

Date __ / __ / ____

```
cur = head ->rlink;
next = cur ->rlink;
head ->rlink =next
next ->llink = head;
printf ("the node deleted is %d", cur ->info);
freenode (cur);
return head;
}
NODE ddelete_rear(NODE head)
{
    NODE cur, prev;
     if(head ->rlink ==head)
      {
        printf ("list empty \n");
         return head;
      }
     cur = head->llink;
     prev = cur -> llink;
      head ->llink = prev;
       prev ->rlink = head;
     printf ("the node deleted is %d", cur->info);
     freenode (cur);
     return head;
   }
NODE insert_leftpos (int item, NODE head)
{
     NODE temp , cur, prev;
     if (head ->rlink ==head)
       {
         printf("list empty \n");
       return head;
       }
```

Date ___/___/___

```
cur = head -> rlink;
while (cur != head)
{
  if (item == cur -> info) break;
   cur = cur -> rlink;
  }
if (cur == head)
 {
  printf ("Key not found \n");
   return head;
 }
prev = cur -> link;
printf ("enter towards left of %d = ", item);
temp = getnode ();
scanf ("%d", & temp -> info);
prev -> rlink = temp;
temp -> llink = prev;
 cur -> llink = temp;
 temp -> rlink = cur;
  return head;
}
NODE insert_rightpos (int item, NODE head)
{
    NODE temp, cur, prev;
    if (head -> rlink == head)
      {
         printf ("list empty \n");
         return head;
      }
    cur = head -> rlink;
```

Date __/__/____

```
while (cur != head)
{
  if (item == cur->info) break;
   cur = cur->rlink;
}
if (cur == head)
{
  printf ("Key not found \n");
  return head;
 }
prev = cur-> link;
printf ("enter towards right of %d = ", item);
 temp = getnode ();
 scanf ("%d", & temp -> info);
 prev -> llink = temp;
 temp -> llink = cur;
 cur -> rlink = temp;
 temp -> rlink = prev;
 return head;
}
NODE  delete_all_key( int item, NODE head)
{
   NODE prev, cur, next;
   int count;
   if (head -> rlink == head)
   {
      printf ("List Empty \n");
      return head;
   }
    count = 0;
    cur = head -> rlink;
```

Page No.

Date ___/___/___

```
while (cur != head)
{
  if (item != cur -> info)
   cur = cur -> rlink;
  else
  {
     count ++;
     if (count ==1)
      {
         cur = cur -> rlink;
         continue;
      }
      prev = cur -> llink;
      next = cur -> rlink;
      prev -> rlink = next;
      next -> llink = prev;
        freenode (cur);
       cur = next;
    }
  }
   if (count ==0)
   printf ("Key not found");
   else
    printf ("Key found at %d position \n",
                                       count);
   return head;
 }
void search-info ( int item , NODE head)
{
  NODE cur;
  if (head -> rlink == head)
```

Date __/__/____

```
 {
   printf("list empty\n");
  }
cur = head->rlink;
while (cur != head)
 {
   if (item == cur->info)
    {
       printf("Search succesfull\n");
       break;
    }
   cur = cur->rlink;
 }
 if (cur == head)
  {
    printf("Element not found\n");
  }
}
void display(NODE head)
 {
   NODE temp;
   if (head->rlink == head)
    {
       printf(" list empty\n");
       return;
    }
   for (temp = head->rlink; temp != head;
                          temp = temp->rlink)
 printf("%d\n", temp->info);
}
```

Page No.

Scanned with CamScanner

Date ___/___/___

```
void main()
{
  int item, choice, key;
  NODE head, last;
  head = getnode();
  head->rlink = head;
  head -> llink = head;
  for(;;)
 {
   printf("1: Insert Front \n 2: Insert Rear\n
          3: Delete Front \n 4: Delete Rear \n 5.
  Insert-left of node");
 printf("\n Insert right of node \n 7. Delete Duplicate
  \n 8. Search Item \n 9. Display \n 10. exit \n");
 printf(" enter the choice : \n");
 scanf("%d", &choice);
 printf("----\n");
 switch(choice)
 {
   case 1: printf("enter the item at front end \n")
           scanf("%d", &item);
           last = dinsert_front(item, head);
           break;
 Case 2 : printf("enter the item at rear end\n")
          scanf("%d", &item);
          last = dinsert-rear(item, head);
          break;
 Case 3 : last = ddelete-front(head);
          break;
 Case 4 : last = ddelete_rear(head);
          break;
```

}

Page No.

Date ___/___/___

```
case 5: printf ("enter the key item\n");
        scanf ("%d", & item);
        head = insert_leftpos( item ,head);
        break;
case 6: printf ("enter the key item \n");
        scanf ("%d", item)
        head = insert_rightpos (item ,head);
        break;
case 7: printf ("enter the key item \n");
        scanf ("%d", & item);
        head = delete_all_key( item, head);
        break;
case 8 : printf ("enter the key item \n");
         scanf ("%d" , & item );
         search_info (item, head);
         break;
case 9 : display (head);
         break;
default : exit (0);
}
}
}
```