LAB 9:-

Program:-

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

struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
        };
typedef struct node *NODE;
NODE getnode()
{
        NODE x;
        x=(NODE)malloc(sizeof(struct node));
        if(x==NULL)
        {
                printf("mem full\n");
                exit(0);
                }
        return x;
        }
void freenode(NODE x)
{
```

```
        free(x);
}
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("dq empty\n");
return head;
}
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("dq 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;
}
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->llink;
  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 delete_specified_value(int item,NODE head)
{
NODE prev,cur,next;
int count;
     if(head->rlink==head)
       {
         printf("List is empty");
         return head;
         }
count=0;
cur=head->rlink;
while(cur!=head)
```

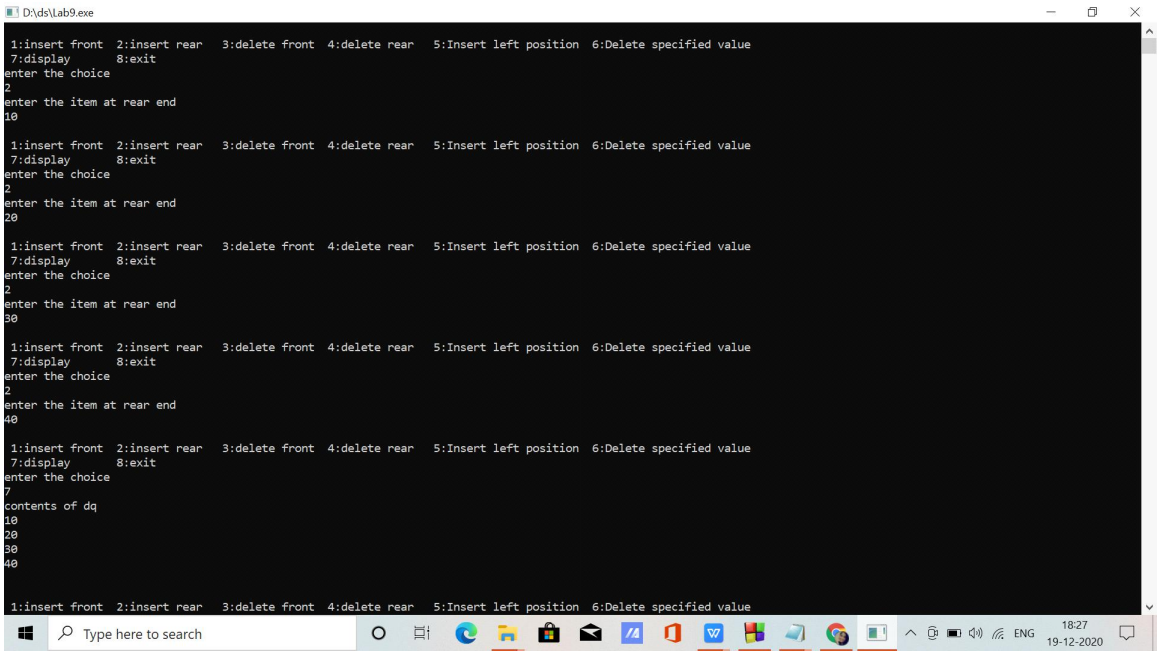
```
{
    if(item!=cur->info)
    cur=cur->rlink;
    else
  {
    count++;
    prev=cur->llink;
    next=cur->rlink;
    prev->rlink=next;
    next->llink=prev;
    freenode(cur);
    cur=next;
  }
}
if(count==0)
    printf("key not found");
    else
  printf("Key found at %d positions and are deleted\n", count);

return head;
}



void display(NODE head)
{
```

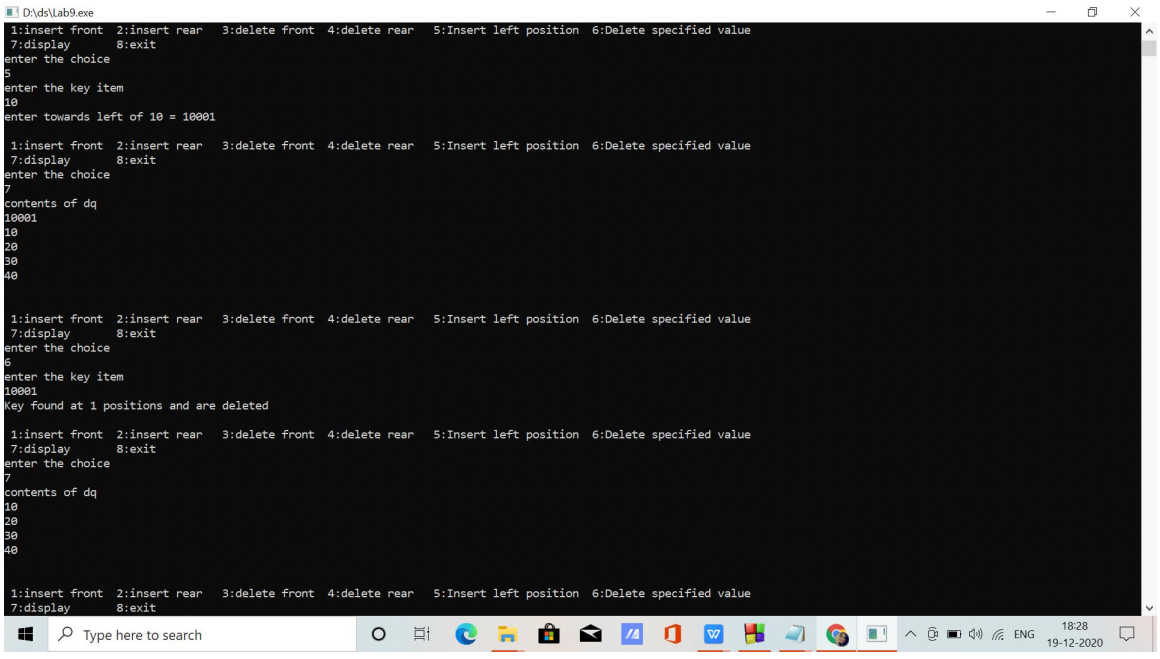
```
NODE temp;
if(head->rlink==head)
{
printf("dq empty\n");
return;
}
printf("contents of dq\n");
temp=head->rlink;
while(temp!=head)
{
printf("%d\n",temp->info);
temp=temp->rlink;
}
printf("\n");
}
void main()
{
NODE head,last;
int item, choice;
head=getnode();
head->rlink=head;
head->llink=head;
for(;;)
{
```

```
printf("\n 1:insert front\t 2:insert rear\t 3:delete front\t 4:delete rear\t 5:Insert left position\t 6:Delete specified value\n 7:display\t 8:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
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;
   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=delete_specified_value(item,head);
```

```
                break;
            case 7: display(head);
                    break;
            default:exit(0);
            }
        }
}
```

Output Screenshot:-





Written Pictures:-

Doubly linked list:-

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *llink;
struct node *rlink;
};
typedef struct node * NODE;
NODE getnode() {
NODE x;
x = (NODE) malloc(sizeof(struct node));
if (x==NULL) {
printf("mem full\n");
exit(0); }
return x; }
void freenode(NODE x)
free(x);
}
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){
pf("dq empty\n");
return head;}
cur = head->rlink;
next = cur->rlink;
head->rlink = next;
next->llink = head;
pf("the node deleted is %d", cur->info);
freenode(cur);
return head;
}
NODE ddelete_rear(NODE head)
{
NODE cur, prev;
if(head->rlink == head)
{
pf("dq empty\n");
return head;
}
```

```
cur = head -> llink;
prev = cur -> llink;
head -> llink = prev;
prev -> rlink = head;
pf ("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;
}
cur = head -> rlink;
while (cur != head)
{
printf ("key not found\n");
return head; }
temp -> llink = prev;
cur -> llink = temp;
temp -> rlink = cur;
return head;
}
NODE delete_specified_value (int item, NODE head)
{
NODE prev, cur, next;
int count;
pf (head -> rlink == head)
{
pf ("\nE
```

```
NODE temp;
if (first == NULL)
printf ("list empty");
for (temp = first; temp != NULL; temp = temp->link)
 {
 printf("%d\n", temp->info);
 }
}
NODE concat (NODE first, NODE second)
 {
 NODE cur;
if (first == NULL)
return second;
if( second == NULL)
 return first;
cur = first;
while ( cur -> link != NULL)
cur = cur->link;
cur -> link = second;
return first;
 }
NODE reverse (NODE first)
 {
NODE cur, temp;
cur = NULL;
while ( first != NULL)
 {
 temp = first;
first = first -> link;
temp-> link = cur;
cur = temp;
} return cur;
```

```
return head;}
while (cur!=head) {
pf (item!=cur->info)
cur = cur -> rlink;
else
{
count++;
prev = cur->llink;
next = cur->rlink;
prev->rlink = next;
next->llink = prev;
freenode(cur);
cur = next;
}}
if (count==0)
pf("key not found");
else
pf("key found @ %d position deleted", cur);
return head;}

void display(NODE head)
NODE temp;
pf(head->rlink==head)
{
pf("dq empty\n");
return;}
pf("contents of dq")
temp = head->rlink;
while(temp!=head)
{
pf("%d",temp->info);
temp = temp->rlink;
}
```

```
void main() {
  NODE head, last;
int item, choice;
 head = getnode();
head → rlink = head;
head → llink = head;
for(;;)
{
pf("1F \t2 IR \t3 DF \t4 DR \t5 insert left
 pos \t6 . delete specified value \t7 : display
\t8 exit \n");
 switch(choice);
 { case 7 : pf(" enter the item @ front end \n");
scanf("%d", &item);
last = dinsert_front(item, head);
 break;
 case 2: printf(" enter the item @ rear
   end \n");
sf("%d", &item);
last = dinsert_rear(item, head);
 break;
case 3: last = ddelete from(head);
   break;
case 4 : east- ddelete rear(head); break;
case 5 : pf(" key \n"); sf("%d", &item);
case 6 : head = delete_specified_value(item,
 break;
case 7 : display (head);
break;
default : exit(0);
}
```