1BM19ET004.
Akash Shrivastava

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("Memory full"),
       exit(0);
}

return x;
}

void freenode(NODE x)
    {
       free(x);
    }
NODE dinsert_front(int item, NODE head)
{
NODE head, cur,
temp = gethode();
```

```
temp -> info = item;
cur = head -> rlink
head -> rlink = temp;
head -> llink = head
temp -> rlink = head;
temp -> llink = cur;
cur -> rlink = temp;
return head;
}

NODE delete_front(NODE head)
{
NODE cur, next;
if (head -> rlink == head)
{
   printf("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");
}
cur = head->llink;
prev = cur->llink;
head->llink = prev;
prev->rlink = head;
printf("Node deleted is %d", cur->info);
free node(cur);
return head;
}

void display(NODE head)
{
NODE temp;
if(head->rlink == head)
{
  printf("Empty \n");
return;
}
printf("contents of list: \n");
temp = head->rlink;
while(temp != head)
{
 printf("%d", temp->info);
 temp = temp->rlink;
}
 printf("\n");
}
```

```
void Search (NODE head)
{
 bool flag = false;
NODE x = head -> ulink -> llink;
NODE temp = head -> ulink;
int item, count = 1;
printf("Enter value:");
scanf("%d", &item);
while (temp != x)
 {
    if (temp -> info == item)
      {
        flag = true;
       break;
      }
   temp = temp -> ulink;
   count++;
}
if (flag)
printf("Element found at position: %d", count);
 else if (temp == x)
 printf("Element not found \n");
}

NODE insert_leftpos (NODE Read)
{
NODE temp, cur, prev;
int item;
printf("enter the item");
scanf("%d", &item);
if (head -> ulink == Read)
```

```
printf("List empty");
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 -> rlink;
printf("Enter the item 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(NODE head)
{
NODE temp, cur, next;
int item;
printf("Enter item");
```

```
scanf("%d",&item);
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");
return head;}
next=cur->rlink;
printf("enter item towards right of %d",item);
temp=getnode();
scanf("%d",&temp->info);
next->llink=temp;
temp->rlink=next;
cur->rlink=temp;
temp->llink=cur;
return head;
}

int main()
{
NODE head,last,y;
int item,choice;
head=getnode();
head->rlink=head;
head->llink=head;
```

```
for(;;)
{
printf("\n1. Insert front\n2. Insert rear\n3. delete front\n4. delete
rear\n5. Display\n6. Search\n7. Insert before key node\n
8. Insert after key node\n9. Delete Occurances\n");
printf("enter the choice");
scanf("%d", &choice);
Switch(choice)
{
case 1: print("Enter item:");
        scanf("%d", &item);
        last = dinsert_front(item,head);
        break;
case 2: printf("Enter item:");
        scanf("%d", &item);
        last=dinsert_rear(item,head);
        break;
case 3: last=delete_front(head);
        break.
case 4: last=delete_rear(head);
        break;
case 5: display(head);
        break;
case 6: search(head)
        break.
case 7: y=insert_leftpos(head);
        break;
case 8: y=insert_rightpos(head);
        break;
Case 9: delete_all_key(head);
        break;
}
```