1BM19CS158

```
#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\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 ("Doubly linked list empty\n");
  return head;
}
cur = head->rlink;
next = cur->rlink;
head->rlink = next;
next->llink = head;
printf ("In The node deleted is %d", cur->info);
freenode (cur);
```

```
return head;
}
NODE dldelete_rear(NODE head)
{
  NODE cur, prev;
  if (head->rlink == head)
  {
    printf("Doubly linked list empty\n");
    return head;
  }
  cur=head->llink;
  prev = cur->llink;
  head->llink = prev;
  prev->rlink = head;
  printf("\n The node deleted is %d", cur->info);
  freenode(cur);
  return head;
}

void display(NODE head)
{
  NODE temp;
  if (head->rlink == head)
  {
    printf("Doubly linked list empty\n");
    return;
  }
  printf("Contents of the doubly linked list\n");
  temp= head->rlink;
  while (temp!=head)
  {
    printf("%d\t", temp->info);
```

```
    temp = temp->rlink;
}
printf("\n");

NODE delete_all_key(int item, NODE head)
{
    NODE prev, cur, next;
    int count;
    if (head->rlink == head)
    {
        printf("Doubly linked list empty\n");
        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;
            free node(cur);
            cur = next;
        }
    }
    if (count == 0)
        printf("Key not found\n");
```

```
else
printf ("Key found at %d positions and are deleted \n", count);
return head;
}

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

```
void search (NODE head)
{
  struct node *ptr;
  int item, i=0, flag;
  ptr = head;
  if (ptr == NULL)
  {
    printf ("Doubly linked list empty \n");
  }
  else
  {
  printf ("\nEnter item which you want to search ?\n");
  scanf ("%d", &item);
  while (ptr != NULL)
  {
    if (ptr->info == item)
    {
      printf ("\nitem found at location %d", i+1);
      flag = 0;
      break;
    }
    else
    {
      flag = 1;
      printf ("\n Item not found \n");
      break;
    }
  }
 }
}
 int main ()
{
```

```
NODE head, last;
int item, choice, choice1;
head = getnode();
head->rlink = head;
head->llink = head;

for(;;)
{
printf("\n1: insert front\n2: insert rear\n3: delete front\n
4: delete rear\n5: display\n6: delete all occurences\n
7: simple search\n8: insert node before & after the
key node\n9: 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: display(head);
        break;
case 6: printf("enter element to be deleted\n");
        scanf("%d", &item);
```

```
delete_all_key (item, head);
break;
case 7: search (head);
        break;
case 8:
printf ("\n1. insert towards left\n 2: insert towards right \n");
printf ("enter the choice\n");
scanf ("%d", &choice1);
if (choice1 ==1)
{
printf ("enter the key element \n");
scanf ("%d", &item);
last = insert_leftpos (item, head);
 break;
}
else
{
printf ("enter the key element\n");
scanf ("%d", &item);
last = insert_rightpos (item, head);
 break;
}
    default : exit (0);
    }
}
return 0;
}
```