14/12/12 Double linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int info;
 struct node * rlink;
 struct node * llink;
};
type def struct node * NODE;
NODE get node() {
   NODE x;
   x = (NODE) malloc (size of (struct
                        node));
   if (x == NULL)
   {
    printf (" Memory full \n");
    exit(0);
   }
   return x;
  }
  NODE dinsert_rear(int item, NODE
  head){
     NODE temp, cur;
```

```
temp -> info = item;
cur = head -> rlink;
head -> rlink = temp;
temp -> llink = head;
temp -> rlink = cur;
cur -> llink = 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 item deleted is %d\n",
free(cur);
return head;
}
NODE ddelete_rear (NODE head)
```

```
{
NODE cur,prev;
if(head -> rlink == head)
{
printf("dq empty \n")
    retur head;
}
cur = head -> llink;
prev = cur -> llink;
head -> llink = prev;
prev -> rlink = head;
printf("The item deleted is %d \n",
            cur -> info);
   free(cur);
   return head;
}
NODE dlsearch(NODE head, int key,
int x){
   NODE cur, prev, temp;
int f=0, c=1;
   if(head -> rlink == head)
{
printf("list empty \n");
   return head;
}
```

Scanned with CamScanner

```
if (cur -> info == key) {
    f = 1;
    break;
}
cur = cur -> rlink;
    c++;
}
if (f == 1 && z == 0) {
printf ("Search successful, found at
index %d \n", c);
return head;
}
if (f == 1 && z == 1) {
prev = cur -> llink;
printf ("Enter towards left of %d
= ", key);
    temp = getnode ();
    scanf ("%d", &temp -> info);
    prev -> rlink = temp;
    temp -> llink = prev;
    cur -> llink = temp;
    temp -> rlink = cur;
    return head;
    }
    if (f == 1 && z == 2) {
```

```
prev = cur;
cur = cur -> link;
printf("Enter towards right of %d=key);
temp = get node();
scanf("%d", &temp -> info);
prev -> rlink = temp;
temp -> llink = prev;
cur -> llink = temp;
temp -> rlink = temp;
return head;
}
printf("Search unsuccessful \n");
}
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++
prev = cur -> llink;
```

```
next = cur -> rlink;
prev -> rlink = next;
next -> llink = prev;
free(cur);
cur(cur);
cur = next;
}
}
if (count == 0)
printf("Key not found \n");
else
printf("Key found at %d post"
& 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 (temp1 != head)
{
  printf ("%d", 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 ("Enter choice: \n1. Insert front
    \n2. Delete front \n3. Insert rear
    \n4. Delete rear \n5. Simple
    search \n6. Insert left of
    key \n7. Insert right of
key \n8. Delete all occurences
of key \n9. Display \n --
    Any other key to exit --- \n")
```

Scanned with CamScanner

```
switch (choice) {
  case 1: printf("Enter the item at
front end \n");
  scanf("%d", &item);
head = dinsert_front(item, head);
  break;
case 3: printf("enter the item at
  rear end \n");
  scanf("%d", &item);
  head = dinsert_rear(item, head);
  break;
  case 2:
    head = ddelete_front(head);
break;
case 4:
  head = ddelete_rare(head);
break
case 5: printf("Enter key \n");
  scanf("%d", &item);
head = Isearch(head, item 1);
  break;
case 7: printf("Enter key \n");
scanf("%d", &item);
head = Isearch(head, item; 2);
  break.
}
```