DOUBLY LINKED LISTS.

```
#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)
     {
        pf(" 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 (" 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 (" 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)
         {  pf (" list empty\n");
              return head;
         }
        curr = head -> rlink;
         while (cur! = head)
       {
       if (item == cur -> info)
              break;
          cur = cur -> rlink;
    }
    if.(cur == head)
     {  pf (" key not found \n");
           return head;
     }
```

```
        prev = cur->llink;
pf(" Inter 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-all-key(int item, NODE head).
      { NODE prev, cur, next;
             int count;
      if(head->rlink == head)
           { pf("LE");
                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)
    pf("key not found");
    else
pf(" key found at %d positions & are deleted",
            count);
return head;
}
void search(int item, NODE head)
    { NODE cur;
     if (head -> rlink = =head)
       {
          pf ("list empty");
           return;
       cur = head ->rlink;
       while (cur != head)
        {
    if (item = = cur->info)
           break;
       cur = cur->rlink;
     }
     if (cur = =head)
      { pf (" Search succesfull");
           return;
      }
      pf(" Search unsuccessfull");
  }
void display(NODE head)
    {  NODE temp;
     if (head -> rlink = =head)
        { pf (" dq empty \n");
          return;
     }.
Pf(" Contents of dq");
        temp = head -> rlink;
        while (temp != head)
    {
     Pf ("%d\n", temp -> info);
            temp = temp -> rlink;
    }
   Pf("\n");
}.
```