* LAB - 9 :

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("Empty");
        }
        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("List empty\n");
        return head; }
    cur = head->rlink;
    next = cur->rlink;
    head->rlink = next;
    next->llink = head;
    pf("the node deleted is %d", cur->info);
    free node(cur);
    return head;
    }

NODE ddelete_rear(NODE head){
    NODE cur, prev;

    if(head->rlink == head){ pf(empty);
        return head; };

    cur = head->llink;
    prev = cur->llink;
    head->llink = prev;
    prev->rlink = head;
    pf("node deleted is %d, cur->info);
    free node(cur);
    return head;
}
```

```
NODE insert_before (int item, NODE head) {
    NODE temp, cur, prev;

    if (head->rlink == head) {
            pf("Empty");
            return head;
    }

    cur = head->rlink;
    while (cur != head) {

        if (item == cur->info) break;
        cur = cur->rlink;
    }

    if (cur == head) {
            pf("key not found");
            return head }

    prev = cur->llink;
    pf("enter element before %d = ", item);

    temp = getnode();
    sf("%d", &temp->info);
    prev->rlink = temp;
    temp->llink = prev;
    cur->llink = temp;
    temp->rlink = cur;
    return head;
}
```

```
NODE insert_after(int item, NODE head){
    NODE temp, cur, prev;
    if(head->rlink==head){
        printf("empty");
        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;
    }
    prev = cur->rlink;
    printf("enter element after %d =", item);
    temp=getnode();
    scanf("%d",&temp->info);
    cur->rlink = temp;
    temp->llink = cur;
    temp->rlink = prev;
    prev->llink = temp;
    return head;
}
```

```
NODE Search (int item, NODE head) {
    if (head -> rlink == head) {
        pf ("LIST empty");
        return head;
    }
    NODE cur;
    cur = head -> rlink;
    while (cur != head) {
        if (item == cur -> info) break;
        cur = cur -> rlink;
    }
    if (cur == head) {
        pf ("Element not found");
        return head;
    }
    pf ("Search successful, element found");
    return head;
}
```

```
void delete_dup (NODE head) {

    NODE cur, temp, ptr, prev ;
if (head -> rlink == head) {
                pf ("List Empty :");
                return;
    }

    temp = head -> rlink;
    cur = head -> rlink;

while (temp != head) {

    prev = cur;
    cur = temp -> rlink;

    while (cur != head) {

        if (temp -> info == cur -> info) {

            ptr = cur -> rlink ;
            ptr -> llink = cur -> llink;
            ptr = cur -> llink
            ptr -> rlink = cur -> rlink;
            free node (cur);
        }
        cur = cur -> rlink;
    }
    temp = temp -> rlink;
}
    return;
}
```

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

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

```
}
pf("1. insert front 2. insert rear 3. delete front
    4. delete rear 5. display 6. Insert -
    before, 7. insert after. 8. Search
    9. del Duplicate");

sf("%d", &choice);

switch(choice){

case 1: pf("enter item");
        sf("%d", &item);
        last = dinsert_front(item, head);
        break;
case 2: pf("enter");
        sf("%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: pf("enter key");
        sf("%d", &item);
        head = insert_before(item, head);
        break;
case 7: pf("enter key");
        sf("%d", &item);
        head = insert_after(item, head);
        break;
```

```
case 8: pf("Enter the element for search ing");
        sf("%d", &item);
        head = search(item, head);
        break;

case 9: del_dup(head);
        pf("List after deleting duplicate
                elements");
        display(head);
        break;

default: exit(0);
}

}
}
```