HEMANG SINGH

LAB-9

1)

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int info;
    struct node *rlink; *llink;
};
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;
}
NODE dinsert_front(int item, NODE head){
    NODE temp, cur;
    temp = getnode();
    temp->info = item;
    temp->llink = NULL;
    temp->rlink = NULL;
    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;
    temp -> llink = NULL;
    temp -> rlink = NULL;
    cur = head -> link;
    head -> llink = temp;
    temp -> rlink = head;
    cur -> rlink = temp;
    temp -> llink = cur;
    return head;
}
NODE ddelete_front(NODE head){
    NODE cur, next;
    if(head -> rlink == head){
        printf("List is empty\n");
        return head;
    }
```

```
cur = head -> rlink;
next = cur -> rlink;
head -> rlink = next;
next -> llink = head;
printf("Item deleted at the front end
                is : %d ", cur -> info);

free(cur);
return head;

}
NODE ddelet_rear(NODE head) {
    NODE cur, prev;
    if(head -> rlink == head) {
        printf("List is empty \n");
        return head;
    }
    cur = head -> llink;
    prev = cur -> llink;
    prev -> rlink = head;
    head -> llink = prev;
    printf("Item deleted at the rear
                end is %d \n", cur -> info);

    free(cur);
    return head;

}
void ddisplay(NODE head) {
```

```
NODE temp;
if(head -> rlink == head){
    printf("List is empty\n");
}
printf("The contents of the list are :\n");
temp = head -> rlink;
while(temp != head){
    printf("%d\n", temp -> info);
    temp = temp -> rlink;
}
}

void dsearch(int key, NODE head){
NODE cur;
int count;
if(head -> rlink == head){
    printf("List is empty\n");
}
cur = head -> rlink;
count = 1;
while(cur != head && cur -> info != key
{
    cur = cur -> rlink;
    count++;
}
if(cur == head){
```

```
printf ("Search Unsuccessfull \n");
}
else {
    printf("key element found at the
            position %d \n", count);
}
}
NODE dinsert_leftpos(int item, NODE head){
    NODE cur, prev, temp;
    if(head->rlink == head) {
        printf("List is empty \n");
        return head;
    }
    cur = head->rlink;
    while(cur != head) {
        if (cur->info == item) {
            break;
        }
        cur = cur->rlink;
    }
    if(cur == head){
        printf("No Such item found in the
                list \n");
        return head;
    }
    prev = cur->llink;
    temp = getnode();
```

```
temp -> llink = NULL;
temp -> rlink = NULL;
printf("Enter the item to be inserted at the
                    left of the given item\n");

scanf("%d", &temp -> info);
prev -> rlink = temp;
temp -> llink = cur;
cur -> llink = temp;
return head;
}
NODE dlinsert_rightpos(int item, NODE head)
{
NODE temp, cur, next;
if(head -> rlink == head) {
    printf("List is empty\n");
    return head;
}
cur = head -> rlink;
while(cur != head) {
    if(cur -> info == item) {
        break;
    }
    cur = cur -> rlink;
}
if(cur == head) {
    printf("No such item found in
                    the list\n");
```

```
    return head;
}
    next = cur -> rlink;
    temp = getnode();
    temp -> llink = NULL;
    temp -> rlink = NULL;
    printf("Enter the item to be inserted
            inserted at the right of the given
            item: \n");
    scanf("%d", &temp -> info);
    cur -> rlink = temp;
    temp -> llink = cur;
    next -> llink = temp;
    temp -> rlink = next;
    return head;
}
NODE delete_duplicates(int item,
                       NODE head){

    NODE prev, cur, next;
    int count = 0;
    if (head -> rlink == head) {
        printf("List is empty\n");
        return head;
```

```
}
cur = head -> rlink;
while ( cur != head) {
    if ( cur -> info != item) {
        cur = cur -> rlink;

    }
else {
    count ++;
    if ( count == 1) {
        cur = cur -> rlink;
        continue

    }
else {
    prev = cur -> llink;
    next = cur -> rlink;
    prev -> rlink = next;
    next -> llink = prev;
    free( cur);
    cur = next;

    } }
}
if(count == 0) {
    printf(" No such item found in the list\n");
}
else {
```

```
printf(" Removed all the duplicates elements of
        the given item succesfully \n");
}
return head;
}
int main(){
    NODE head;
    int item, choice, key;
    head = getnode();
    head->llink = head;
    head->rlink = head;
    for(;;){
        printf("\n 1: dinsert front \n
            2: dinsert rear \n 3. ddelete
            front \n 4. ddelete_rear \n 5.
            ddisplay \n 6. dsearch \n 7. dinsert
            leftpos\n 8. dinsert rightpos \n
            9. ddelete duplicates \n 10. exit(0)
        scanf("%d", &choice);
        switch(choice){
            case 1: printf(" Enter the item at
                        front end \n");
                scanf("%d", &item);
                head = dinsert_front(item,
                        head);
```

```
        break;
Case 2: printf(" Enter the item at rear end:
                \n");
        scanf(" %d", &item);
        head = dinsert_rear(item, head);
        break;
Case 3: head = ddelete_front(head);
        break;
Case 4: head = ddelete_rear(head);
        break;
Case 5: ddisplay(head);
        break;
Case 6: printf(" Enter the key element
                to be searched :\n");
        scanf("%d", &key);
        dsearch(key, head);
        break;
Case 7: printf(" Enter the key element\n
        scanf("%d", &key);
        head = dinsert_leftpos(key, head)
        break;
```

```
Case 8: printf(" Enter the key element:\n");
        scanf("%d", &key);
        head = direct_rightpos(key, head);
        break;

Case 9: printf(" Enter the key element whose
                duplicates should be
                removed:\n");
        scanf("%d", &key);
        head = ddelete_duplicates(key, head);
        break;

default: exit(0);

    }
}
return 0;
}
```