

## LAB PROGRAM 9

Write a program to implement Doubly Linked List to:

1. Insert a node at the front of the list
2. Insert a node at the end of the list
3. Delete a node from the front of the list
4. Delete a node from the end of the list
5. Insert a node to the left of the key element
6. Insert a node to the right of the key element
7. Delete all the key elements
8. Delete all the duplicate elements
9. Display the list.

```
#include <stdio.h>
#include <process.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 is 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("List is empty. \n");
```

```
return head;
```

```
}
```

```
cur = head -> rlink;
```

```
next = cur -> rlink;
```

```
head -> rlink = next;
```

```
next -> llink = head;
```

```
printf("Node deleted is %d", cur -> info);
```

```
freenode(cur);
```

```
return head;
```

```
}
```

```
NODE ddelete_rear(NODE head)
```

```
{
```

```
NODE cur, prev;
```

```
if (head -> rlink == head)
```

```
{
```

```
printf("List is empty. \n");
```

```
return head;
```

```
}
```

```
cur = head -> llink;
```

```
prev = cur -> llink;
```

```
head -> llink = prev;
```

```
prev -> rlink = head;
```

```
printf("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)
```

```
{
```

```
printf("List is 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("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("List is 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("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;
}
NODE delete_all_key(int item, NODE head)
{
    NODE prev, cur, next;
    int count;
    if (head->rlink == head)
    {
        printf("List is 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;  
        freenode(cur);  
        cur = next;  
    }  
}  
if(count == 0)  
    printf("Key not found.");  
else  
    printf("Key found at %d positions and  
    are deleted. \n", count);  
return head;  
}  
void display(NODE head)  
{  
    NODE temp;  
    if(head->rlink == head)  
    {  
        printf("List is empty. \n");  
        return;  
    }  
    printf("Contents of the list : \n");  
    temp = head->rlink;  
    while(temp != 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 ("\n1: Insert front \n2: Insert rear\n3: Delete front \n4: Delete rear \n5: Insert left position \n6: Insert right position\n7: Delete all key elements \n8: Display\n9: Exit \n");
printf ("Enter the choice : ");
scanf ("%d", &choice);
switch(choice)
{
case 1 : printf ("Enter the item to be inserted at front end : ");
scanf ("%d", &item);
last = dinsert_front (item, head);
break;
case 2 : printf ("Enter the item to be inserted at rear end : ");
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: printf("Enter the key item : ");  
scanf("%d", &item);  
head = insert - leftpos(item, head);  
break;

case 6: printf("Enter the key item : ");  
scanf("%d", &item);  
head = insert - rightpos(item, head);  
break;

case 7: printf("Enter the key item : ");  
scanf("%d", &item);  
head = delete - all - key(item, head);  
break;

case 8: display(head);  
break;

default: exit(0);

}

}

}

## LAB PROGRAM : 10

Write a program to perform addition of long numbers.

```
#include <stdio.h>
#include <process.h>
#include <string.h>
struct NODE
{
    int info;
    struct NODE * link;
};
typedef struct NODE* node;
node getnode()
{
    node x;
    x = (node) malloc(sizeof(struct NODE));
    if(x == NULL)
    {
        printf("Memory is full, \n");
        exit(0);
    }
    return x;
}
node insert_front(node first, int item)
{
    node temp;
    temp = getnode();
    temp->info = item;
    temp->link = first;
    return temp;
}
```

```
node extract (char *s; node head)
```

{

```
int i, n;
```

```
for (i=0; i<strlen(s); i++)
```

{

```
n = s[i] - '0';
```

```
head = insert-front (head, n);
```

}

```
return head;
```

}

```
node add-long (node head1, node head2, node head3)
```

{

```
int temp, sum, carry=0;
```

```
node cur1, cur2;
```

```
cur1 = head1;
```

```
cur2 = head2;
```

```
while (cur1 != NULL && cur2 != NULL)
```

{

```
temp = cur1->info + cur2->info + carry;
```

```
if (temp > 9)
```

{

```
sum = temp % 10;
```

```
carry = temp / 10;
```

}

```
else
```

{

```
sum = temp;
```

```
carry = 0;
```

}

```
head3 = insert-front (head3, sum);
```

```
cur1 = cur1->link;
```

```
cur2 = cur2->link;
```

{

```
while (cur1 != NULL)
{
    temp = cur1 -> info + carry;
    if (temp > 9)
    {
        sum = temp % 10;
        carry = temp / 10;
    }
    else
    {
        sum = temp;
        carry = 0;
    }
    head3 = insert_front(head3, sum);
    cur1 = cur1 -> link;
}

while (cur2 != NULL)
{
    temp = cur2 -> info + carry;
    if (temp > 9)
    {
        sum = temp % 10;
        carry = temp / 10;
    }
    else
    {
        sum = temp;
        carry = 0;
    }
    head3 = insert_front(head3, sum);
    cur2 = cur2 -> link;
}

if (cur1 == NULL && cur2 == NULL)
```

```
{  
    if (carry == 1)  
        head3 = insert-front (head3, carry);  
    }  
    return head3;  
}  
void display(node first)  
{  
    node cur;  
    if (first == NULL)  
    {  
        printf("Empty \n");  
        return;  
    }  
    cur = first;  
    while (cur != NULL)  
    {  
        printf("%d \t", cur->info);  
        cur = cur->link;  
    }  
}  
void main()  
{  
    int ch;  
    node head1 = NULL;  
    node head2 = NULL;  
    node head3 = NULL;  
    char s1[30], s2[30];  
    printf("Enter the first long integer : \n");  
    scanf("%s", s1);  
    head1 = extract (s1, head1);  
    display (head1);  
    printf("Enter the second long integer: \n");
```

```
scanf ("%s", s2);
head2 = extract (s2, head2);
display (head2);
head3 = add_long (head1, head2, head3);
printf ("\n The result of addition is: \n");
display (head3);
```

{}