LAB PROGRAM - 7

Q. WAP to implement Doubly linked list with primitive operations.

a) Create a doubly linked list
b) Insert a new node to the left of the node.
c) Delete the node based on specific value.
d) Display the contents to the list.

Pseudocode:

```
Struct Node
 Int data
 Struct Node *prev *next

function createlist
 int i, data
 Struct Node * newnode
for (i=1 ; i<=n ; i++)
    printf ("Enter data)

newnode = (struct Node* ) malloc
          (sizeof (struct Node));

 newnode -> data = data
 newnode -> prev = newnode -> next
                 = NULL;

if (head == NULL
    head = tail = newNode
```

```
else
tail->next = newNode
newNode->prev = tail
tail = newNode

function insert at end
struct node *newNode = (struct node*)
malloc (sizeof (struct node));

~~newNode~~
NewNode -> data = data;
newNode -> prev = NULL
newNode -> prev = tail

if (tail == NULL)
    head = tail = newnode
    ~~tail = newNode~~
else
    tail -> next = newNode
    tail = newNode

function delete by value

struct node *temp = head;
if (head == NULL)
    print (list empty)
    return

while temp != NULL && temp -> data != value
    temp = temp -> next

if (temp == NULL)
    printf ("value not found")
    return.
```

```
if (temp == head)
  delete at front();
else if (temp = tail)
    delete at end()

else
    temp -> prev -> next = temp -> next;
    temp -> prev -> prev = temp -> prev
     free (temp)

Function DISPLAY()
{
  struct Node * temp = head;
  printf ("LIST (forward):").
  while (temp != NULL).
    printf ("%d <-> ", temp -> data)
     temp = (temp -> next);
  print ("NULL");
```

code

```
# include <stdio.h>
# include <stdlib.h>

struct node {
        int data;
      struct node *next, * prev;
   };

Struct Node *head = NULL;
Struct Node *tail = NULL;
```

```
void creatlist (int n)
{
  int i, data;
  struct node *newnode;
for (i=1 ; i<=n ; i++)
{
  printf ("Enter data for node %d: ", i);
  scanf ("%d", &data);

newnode = (struct node *) malloc (sizeof (
                  (struct node));
newnode => data = data;
newnode -> prev = newnode -> next = NULL;
if (head == NULL)
  {
   head = tail = newnode;
  }
  else {
       tail -> next = newnode;
       newnode -> prev = tail;
        tail = newnode;
  }
}
}

void insertleft (int key, int data)
{
  struct node * temp = head;
  struct node * newnode;

while (temp != NULL && temp -> data != value)

  {
    temp = temp -> next; }
```

```
if (temp == NULL)
{
  printf ("value %d not found.\n",
                value);
   return;
}
newnode = (struct node *) malloc
          (sizeof (struct node));
newnode -> data = data;
if (temp == head)
  {
    newnode -> prev = NULL;
    newnode -> next = head;
     head -> prev = newnode;
      head = newnode;
      return;
  }.
  newnode -> next = temp;
  newnode -> prev = temp -> prev;
   temp -> prev -> next = newnode;
   temp -> prev = newnode;
}.

void deletevalue (int value)
{
 struct node *temp = head;

while (temp != NULL && temp -> data !=
          value)
 {
    temp = temp -> next;
 }
  if (temp == head)
  {
```

```
head = head->next;
if (head != NULL)
    head->prev = NULL;
    free (temp);
     return;
}
if (temp == tail)
{
    tail = tail->prev;
    tail->next = NULL;
     free (temp);
      return;
}
temp->prev->next = temp->next;
temp->next->prev = temp->prev;
 free (temp);
}
void display()
{
 struct node *temp = head;
   printf("\n Doubly linked list:");

   while (temp != NULL)
   {
     printf ("%d", temp->data);
      temp = temp->next;
   }
      printf ("\n");
}

int main()
{
  int n, choice, val, data;
  printf ("enter number of nodes:");
```

```
scanf("%d", &n);
createLPSt(n);
display();

while(1)
{
printf("\n1. Insert left \n");
printf("2. Delete \n");
printf("3. Display \n");
printf("4. Exit \n");
scanf("%d", &ch);

switch(ch)
{
case1:
    printf("Entere existing value:");
    scanf("%d", &key);
    printf("Entere data:");
    scanf("%d", &data);
    InsertLeft(key, data);
    break;

case2:
    printf("Entere value to delete:");
    scanf("%d", &key);
    deletevalue(key);
    break;

case3:
    display();
    break;
case4:
    exit(0);
}
```

```
}
  return 0;
}
```

output :

```
enter number of nodes : 4
enter data for node 1 : 1
enter data for node 2 : 2
enter data for node 3 : 3
enter data for node 4 : 4

LIST : 1 2 3 4
1. Insert
2. delete
3. Display
4. EXIT
Enter choice : 1
enter expsing value : 1
Enter data : 9

1. Insert left
2. Delete
3. Display
4. EXIT
Enter choice : 2
Enter value to delete 3
1. Insert left
2. Delete
3. Display
4. EXIT

Enter choice : 3
LIST : 9 1 2 4
```