7. WAP to implement doubly

a) Create doubly linked list

2) Insert a new node to the left of the node

c) delete a node based on specified value

d) display the contents of list

pseudocode

```
struct node{
    struct node *prev,*next;
    int data
};
insertatfront
void insertfront(int data){
    struct node *newnode;
    newnode->data=data;
    newnode->prev=null
    newnode->next=head
    if(head==null) head=tail=newnode
    else
        head->prev=newnode
        head=newNode
Void insertatend(int data){
    struct node *newnode;
    newnode->data=data
    newnode->next=null
    newnode->prev=tail
    if(tail=null) head=tail=newNode
    else tail->next=newnode, tail=newNode
void insertatpos(int data, pos)
    int i
    struct node *newnode
case 1: if pos=1
        insert at front at(data)
case 2: for(i=1; i<pos-1 && temp!=null; i++) temp=temp->next
        if(temp==null || temp->next=null)
            insert at end(data)
```

```
newnode->next = temp->next
newnode->prev = temp
temp->next->prev = newnode

void deleteatfront()
    struct node *temp
    if (head == null) printf("empty")
    temp = head
    head = head->next
    if (head != null) head->prev = null
    else tail = null
    free(temp)

void deleteatend()
    struct node *temp
    if (tail == null) printf("empty")
    temp = tail;
    tail = tail->prev
    if (tail != null) tail->next = null
    else head = null;
    free(temp)

void deletebyvalue(int value)
    struct node *temp = head
    if (head == null) printf("empty")
    while (temp != null && temp->data != value)
        temp = temp->next
    if (temp == null) printf("element not found")
    if (temp == head) deleteatfront()
    if (temp == tail) deleteatend
    else temp->prev->next = temp->next
        temp->next->prev = temp->prev
        free(temp)

void display()
    struct node *temp = head
    while (temp != tail)
        printf("%d", temp->data);
        temp = temp->next;
    return;
```

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *prev, *next; };
struct Node *head = NULL, *tail = NULL;
void createList (int n){
    int data;
    struct Node *newNode;
    for (i=1; i<=n; i++)
    { printf("Enter your 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 insertAtFront (int data)
{ struct Node *newNode = (struct Node*) malloc (sizeof (struct Node));
  newNode->data = data;
  newNode->prev = NULL;
  newNode->next = head;
  if (head == NULL)
      head = tail = newNode;
  else
      { head->prev = newNode;
        head = newNode;
  } }
void insertAtEnd (int data){
```

```
struct Node * newNode = (struct
newNode -> data = data;
newNode -> prev = tail;
newNode -> next = NULL;
if (tail == NULL)
    head = tail = newNode;
else {
    tail -> next = newNode;
    tail = newNode;
}
}
void insertatposition (int data, int pos){
    int i;
    struct Node * newNode, *temp = head;
    if (pos == 1){
        insertatfront(data);
        return;
    }
    newNode = (struct node *) malloc (sizeof (struct node));
    newNode -> data = data;
    for (i=1; i < pos-1 && temp != NULL; i++)
        temp = temp -> next;
    if (temp == NULL || temp -> next == NULL){
        insertatEnd(data);
        return;
    }
    newNode -> next = temp -> next;
    newNode -> prev = temp;
    temp -> next -> prev = newNode;
    temp -> next = newNode;
}
void deleteatfront(){
    struct Node *temp;
    if (head == NULL)
        printf("List is empty\n");
    return;
}
```

```
temp = head;
head = head->next;
if (head != NULL)
    head->prev = NULL
else head = NULL
free(temp);
}
void deleteatend()
{ struct node *temp;
  if (head == NULL) {
    printf("List is empty\n");
    return;
  }
  temp = tail;
  tail = tail->prev;
  if (tail == NULL)
      tail->next = NULL;
  else head = NULL;
  free(temp);
}
void deletebyvalue(int value) {
struct node *temp = head;
if (head == NULL)
  { printf("List is empty\n");
    return;
  }
while (temp != NULL && temp->data != value)
    temp = temp->next;
if (temp == head)
    deleteatfront();
else if (temp == tail)
    deleteatend();
else temp->prev->next = temp->next
temp->next->prev = temp->prev
```

```
void display(){
    struct node * temp = head;
    printf("List:");
    while(temp != NULL){
        printf("%d->", temp->data);
        temp = temp->data;
    }
    printf("NULL\n");
}

int main(){
    int choice, n, data, pos, value;
    printf("Enter number of nodes to create:");
    scanf("%d",&n);
    createList(n)
    while(1){
        printf("\n1.Display\n");
        printf("2.Insert at front\n");
        printf("3.Insert at end\n");
        printf("4.Insert at position\n");
        printf("5.Delete at front\n");
        printf("6.Delete at end\n");
        printf("7.Delete by value\n");
        printf("8.Exit\n");
        printf("Enter choice:");
        scanf("%d",&choice);
    switch(choice){
        case 1:
            display();
            break;
        case 2:
            printf("Enter data:");
            scanf("%d",&data);
            insert at front(data);
            break;
```

```
case 3:
    printf("Enter data");
    scanf("%d", &data);
    insertatend(data);
    break;
case 4: printf("Enter data and position:");
    scanf("%d%d", &data, &pos);
    insertatposition(data, pos);
    break;
case 5: deleteatfront()
    break;
case 6: deleteatend()
    break;
case 7: printf("Enter value to delete:");
    scanf("%d", &value);
    deletebyvalue(value);
    break;
case 8: exit(0)
default: printf("Invalid choice\n");
}
return 0;
}
```

o/p

Enter number of nodes to create: 3

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 30

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at end
6. Delete by value

8 Exit
Enter choice : 2
Enter data : 5

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 3
Enter data : 40

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 4
Enter choice : 5

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 6

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 7
Enter value to delete : 30

1. Display
2. Insert at front
3. Insert at end
4. Insert at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 1

List : 10 ⇄ 5 ⇄ 2 ⇄ NULL

1. Display
2. Insert at front
3. Insert at position
4. Delete at position
5. Delete at front
6. Delete at end
7. Delete by value
8. Exit

Enter choice : 8