Program 10: Doubly Linked List

(a) Insertion at Begining
(b) Insertion at End
(c) Insertion at Position
(d) Deletion at Begining
(e) Deletion at End
(f) Deletion at specific value

Pseudo code:

```
void insertAtfront(int data){
    head -> prev = newNode;
    newNode -> next = head;
    head = newNode; }

Void insertAtend (int data){
while(temp -> next != NULL){ temp = temp -> next; }
temp -> next = newNode;
newNode -> prev = temp;

void insertAtPos(int data, int pos){
for(i; i<pos && temp!=NULL; i++){temp = temp ->next}
temp -> next -> prev = newNode;
newNode -> next = temp -> next;
temp -> next = NewNode;
newNode = prev = temp;

void delete specific (int value)
if (temp->next->!= NULL)
    temp -> next -> prev = temp -> prev;
temp -> prev -> next = temp -> next;
free(temp);
```

Date
Page

code:

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

struct Node *head = NULL;

void createList(int n) {
    if (n <= 0) {
        printf("node count must be greater than 0\n");
        return;
    }
    struct Node *temp = NULL;

    for(int i = 1; i <= n; i++) {
        int data;
        printf(" Enter data for node %d: ", i);
        scanf("%d", &data);

        struct Node *newNode = (struct Node *) malloc
                                (sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;
        newNode->prev = NULL;

        if (head == NULL) {
    temp = head = newNode;
        }else {
            temp->next = newNode;
            newNode->prev = temp;
            temp = newNode;
    }
    }
}
```

```
void insertAtLeft(int data, int pos) {
    struct Node *newNode = (struct Node *)malloc(
                           sizeof(struct Node));

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

    if (head == NULL || pos <= 1) {
        newNode -> next = head;
        if (head != NULL)
            head -> prev = newNode;
        head = newNode;
        return;
    }

    struct Node *temp = head;
    for (int i=1; i<pos-1 && temp -> next != NULL; i++)
        temp = temp -> next;

    newNode -> next = temp -> next;
    newNode -> prev = temp;

    if (temp -> next != NULL)
        temp -> next -> prev = newNode;

    temp -> next = newNode;
}

void deleteSpecific(int value) {
    if (head == NULL) {
        printf("List is empty"); return; }
```

```
struct Node *temp = head;

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

if (temp == NULL) {
    printf("Value not found");
    return;
}

if(temp == head){
    deleteAt begining();}
else if (temp->next == NULL)
    deleteAt End();
else{
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    free(temp);
}
}

int main() {
    int choice, n, pos;
    printf("Create list");
    printf("insert at left");
    printf("delete specific");
    printf("display");
    printf("exit");
    while (1) {
        printf("enter your choice:");
        scanf("%d", &choice);
        switch (choice) {
```

```
case 1:
    printf("enter no of nodes");
    scanf("%d", &n);
    createList(n);
    break;

case 2: printf("enter data & pos");
    scanf("%d %d", &n, &pos");
    insertAtLeft(n, pos);
    breaks;

case 3: delete specific();
        break

case 4: displayList();
        break;

case 5: exit(1);
        break;
}
```

Output:

```
Enter choice: 1
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
Enter choice 4
Enter data and position: 5: 2
Enter choice 9
List: 5 <-> 1 <-> 2 <-> 3 <-> 4 <-> NULL
Ente choice 7
```

Date
Page

Enter value to delete : 5
Enter choice : 9
List : 1 ⇔ 2 ⇔ 3 ⇔ 9 ⇔ NULL
Enter choice : 10
Exiting....

O/p seen