Program

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

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

struct node* insertAtBeginning(struct node* head, int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    if (head == NULL) {
        newNode->prev = NULL;
        newNode->next = NULL;
        return newNode;
    }
    newNode->next = head;
    head->prev = newNode;
    newNode->prev = NULL;
    return newNode;
}

struct node* insertAtEnd(struct node* head, int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    if (head == NULL) {
        newNode->prev = NULL;
        newNode->next = NULL;
        return newNode;
    }
    struct node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
    newNode->next = NULL;
    return head;
}

struct node* insertAtPosition(struct node* head, int data, int position) {
    if (position == 0) {
        return insertAtBeginning(head, data);
    }
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    struct node* temp = head;
    for (int i = 0; i < position - 1; i++) {
        if (temp == NULL) {
            free(newNode);
            return head;
        }
        temp = temp->next;
    }
    if (temp == NULL) {
        free(newNode);
        return head;
    }
    newNode->next = temp->next;
```

Experiment No: 13
Date:

DOUBLY LINKEDLIST

Aim:

To implement doubly linked list

Algorithm:

1. Start
2. Node structure

```
Struct Node {
    int data
    struct Node* Prev
    struct Node* Next
}
```

3. Function insertAtBeginning(head, data):

```
    newNode = Allocate Memory for Node
    newNode->data = data
    If head is NULL{
        newNode->prev = NULL
        newNode->next = NULL
        Return newNode
    }
    else:
        newNode->next = head
        head->prev = newNode
        newNode->prev = NULL
        Return newNode
```

4. Function insertAtEnd(head, data):

```
    newNode = Allocate Memory for Node
    newNode->data = data
    If head is NULL:
        newNode->prev = NULL
        newNode->next = NULL
        Return newNode
    Else:
        temp = head
        While temp->next is not NULL:
            temp = temp->next
        temp->next = newNode
        newNode->prev = temp
        newNode->next = NULL
        Return head
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    return head;
}

struct node* removeFromFront(struct node* head) {
    if (head == NULL) {
        return NULL;
    }
    struct node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
    return head;
}

struct node* removeFromRear(struct node* head) {
    if (head == NULL) {
        return NULL;
    }
    struct node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    if (temp->prev != NULL) {
        temp->prev->next = NULL;
    } else {
        head = NULL;
    }
    free(temp);
    return head;
}

struct node* removeFromPosition(struct node* head, int position) {
    if (head == NULL) {
        return NULL;
    }
    if (position == 0) {
        return removeFromFront(head);
    }
    struct node* temp = head;
    for (int i = 0; i < position; i++) {
        if (temp == NULL) {
            return head;
        }
        temp = temp->next;
    }
    if (temp == NULL) {
        return head;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    free(temp);
    return head;
```

5. Function insertAtPosition(head, data, position):

```
If position is 0:
    Return insertAtBeginning(head, data)
newNode = Allocate Memory for Node
newNode->data = data
temp = head
For i from 0 to position - 1:
    If temp is NULL
        Free Memory for newNode
        Return head
    temp = temp->next
If temp is NULL:
    Free Memory for newNode
    Return head
newNode->next = temp.next
newNode->prev = temp
If temp->next is not NULL
    temp->next->prev = newNode
temp->next = newNode
Return head
```

6. Function removeFromFront(head):

```
If head is NULL:
    Return NULL
Else
    temp = head
    head = head->next
    If head is not NULL
        head->prev = NULL
    Free Memory for temp
    Return head
```

7. Function removeFromRear(head)

```
If head is NULL
    Return NULL
Else
    temp = head
    While temp->next is not NULL:
        temp = temp->next
    If temp->prev is not NULL:
        temp->prev->next = NULL
    Else
        head = NULL
    Free Memory for temp
    Return head
```

8. Function removeFromPosition(head, position):

```
If head is NULL:
    Return NULL
If position is 0:
    Return removeFromFront(head)
temp = head
For i from 0 to position
    If temp is NULL
        Return head
    temp = temp->next
If temp is NULL
    Return head
If temp->prev is not NULL
    temp->prev->next = temp->next
```

```
}

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

int main() {
    struct node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Remove from Front\n");
        printf("5. Remove from Rear\n");
        printf("6. Remove from Position\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at beginning: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 2:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position: ");
                scanf("%d", &position);
                head = insertAtPosition(head, data, position);
                break;
            case 4:
                head = removeFromFront(head);
                break;
            case 5:
                head = removeFromRear(head);
                break;
            case 6:
                printf("Enter position to remove from: ");
                scanf("%d", &position);
                head = removeFromPosition(head, position);
                break;
            case 7:
                displayList(head);
                break;
            case 8:
                exit(0);
```

```
If temp->next is not NULL
    temp->next->prev = temp->prev
Free Memory for temp
Return head
```

9. Function displayList(head):

```
temp = head
Print "Doubly Linked List: "
While temp is not NULL:
    Print temp->data
    temp = temp->next
```

10. In main function display a menu and respond to the user's input.

11. Stop

```c
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Output

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Remove from Front
5. Remove from Rear
6. Remove from Position
7. Display List
8. Exit
Enter your choice: 1
Enter data to insert at beginning: 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Remove from Front
5. Remove from Rear
6. Remove from Position
7. Display List
8. Exit
Enter your choice: 1
Enter data to insert at beginning: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Remove from Front
5. Remove from Rear
6. Remove from Position
7. Display List
8. Exit
Enter your choice: 7
Doubly Linked List: 20 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Remove from Front
5. Remove from Rear
6. Remove from Position
7. Display List
8. Exit
Enter your choice: 8
```

Result:

Program has been executed successfully and obtained the output.