Experiment 8

CODE:

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   newNode->prev = NULL;
   return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      newNode->next = *head;
      newNode->prev = *head;
   } else {
      struct Node* tail = (*head)->prev;
      newNode->next = *head;
      newNode->prev = tail;
      tail->next = newNode;
```

```
        (*head)->prev = newNode;
        *head = newNode;
    }
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head;
        newNode->prev = *head;
    } else {
        struct Node* tail = (*head)->prev;
        newNode->next = *head;
        newNode->prev = tail;
        tail->next = newNode;
        (*head)->prev = newNode;
    }
}

void insertAfterNode(struct Node* head, int key, int data) {
    struct Node* temp = head;
    do {
        if (temp->data == key) {
            struct Node* newNode = createNode(data);
            newNode->next = temp->next;
            newNode->prev = temp;
            temp->next->prev = newNode;
            temp->next = newNode;
            return;
        }
        temp = temp->next;
    } while (temp != head);
    printf("Node with data %d not found\n", key);
}
```

```
void insertBeforeNode(struct Node** head, int key, int data) {
    struct Node* temp = *head;
    do {
        if (temp->data == key) {
            if (temp == *head) {
                insertAtBeginning(head, data);
            } else {
                struct Node* newNode = createNode(data);
                newNode->next = temp;
                newNode->prev = temp->prev;
                temp->prev->next = newNode;
                temp->prev = newNode;
            }
            return;
        }
        temp = temp->next;
    } while (temp != *head);
    printf("Node with data %d not found\n", key);
}

void deleteNode(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node *curr = *head, *prev = NULL;

    if (curr->data == key && curr->next == *head) {
        free(curr);
        *head = NULL;
        return;
    }

    if (curr->data == key) {
        prev = curr->prev;
```

```
        prev->next = curr->next;
        curr->next->prev = prev;
        *head = curr->next;
        free(curr);
        return;
    }

    while (curr->next != *head && curr->data != key) {
        curr = curr->next;
    }

    if (curr->data == key) {
        prev = curr->prev;
        prev->next = curr->next;
        curr->next->prev = prev;
        free(curr);
    } else {
        printf("Node with data %d not found\n", key);
    }
}

void traverseList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    int count = 0;
    struct Node* temp = head;
    do {
        count++;
        temp = temp->next;
    } while (temp != head);

    printf("Number of nodes in the list: %d\n", count);
}
```

```
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = head;
    printf("Linked list elements: ");
    do {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(head)\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, key;

    printf("Enter initial data for the list (enter -1 to stop):\n");
    while (1) {
        printf("Enter data: ");
        scanf("%d", &data);
        if (data == -1)
            break;
        if (head == NULL) {
            head = createNode(data);
            head->next = head;
            head->prev = head;
        } else {
            insertAtEnd(&head, data);
        }
    }
```

```
do {
        printf("\nMenu:\n");
        printf("1. Display List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Insert at End\n");
        printf("4. Insert After a Node\n");
        printf("5. Insert Before a Node\n");
        printf("6. Delete a Node\n");
        printf("7. Traverse List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                displayList(head);
                break;

            case 2:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;

            case 3:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 4:
                printf("Enter key to insert after: ");
                scanf("%d", &key);
                printf("Enter data: ");
                scanf("%d", &data);
                insertAfterNode(head, key, data);
```

```
            break;

        case 5:
            printf("Enter key to insert before: ");
            scanf("%d", &key);
            printf("Enter data: ");
            scanf("%d", &data);
            insertBeforeNode(&head, key, data);
            break;

        case 6:
            printf("Enter key to delete: ");
            scanf("%d", &key);
            deleteNode(&head, key);
            break;

        case 7:
            traverseList(head);
            break;

        case 8:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 8);

   return 0;
}
```

Output:

```
Enter initial data for the list (enter -1 to stop):
Enter data: 10
Enter data: 20
Enter data: 30
Enter data: -1

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 1
Linked list elements: 10 <-> 20 <-> 30 <-> (head)

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 2
Enter data: 2

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
```

5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 3
Enter data: 3

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 4
Enter key to insert after: 20
Enter data: 25

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 5
Enter key to insert before: 10
Enter data: 5

Menu:

1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 6
Enter key to delete: 2

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 7
Number of nodes in the list: 6

Menu:
1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 1
Linked list elements: 5 <-> 10 <-> 20 <-> 25 <-> 30 <-> 3 <-> (head)

Menu:

1. Display List
2. Insert at Beginning
3. Insert at End
4. Insert After a Node
5. Insert Before a Node
6. Delete a Node
7. Traverse List
8. Exit
Enter your choice: 8
Exiting...

=== Code Execution Successful ===