Experiment 5:
Name: Atharv Uday Wadadekar
Roll No: 66
Program:

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

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

Node* createList();
Node* insertAtBeginning(Node* head, int data);
Node* insertAtEnd(Node* head, int data);
Node* insertInMiddle(Node* head, int data, int position);
Node* deleteAtBeginning(Node* head);
Node* deleteAtEnd(Node* head);
Node* deleteFromMiddle(Node* head, int position);
void printList(Node* head);

int main() {
    Node* head = NULL;
    int choice, choice1, choice2, data, position;

    do {
        printf("Menu:\n1.Create List\t2.Insertion\t3.Deletion\t4.Print Linked List\
t5.Exit\nEnter choice:");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                head = createList();
                break;
            case 2:
                do {
                    printf("Insertion Operations:\n1.Insert at beginning\t2.Insert at
End\t3.Insert in Middle\t4.Exit\nEnter choice:");
                    scanf("%d", &choice1);

                    switch (choice1) {
                        case 1:
```

```
                printf("Enter data to insert: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;
            case 2:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position to insert at: ");
                scanf("%d", &position);
                head = insertInMiddle(head, data, position);
                break;
            case 4:
                break;
            default:
                printf("Invalid choice.\n");
        }
      } while (choice1 != 4);
      break;
    case 3:
      do {
        printf("Deletion Operations:\n1.Delete at beginning\t2.Delete at
End\t3.Delete in Middle\t4.Exit\nEnter choice:");
        scanf("%d", &choice2);

        switch (choice2) {
            case 1:
                head = deleteAtBeginning(head);
                break;
            case 2:
                head = deleteAtEnd(head);
                break;
            case 3:
                printf("Enter position to delete from: ");
                scanf("%d", &position);
                head = deleteFromMiddle(head, position);
                break;
            case 4:
```

```
                    break;
                default:
                    printf("Invalid choice.\n");
            }
        } while (choice2 != 4);
        break;
    case 4:
        printList(head);
        break;
    case 5:
        printf("Program exited.\n");
        break;
    default:
        printf("Invalid choice.\n");
  }
} while (choice != 5);

  return 0;
}
Node* createList() {
  Node* head = NULL;
  Node* tail = NULL; // Maintain a tail pointer for efficient insertion at the end
  int n, data;

  printf("Enter the number of nodes: ");
  scanf("%d", &n);

  for (int i = 0; i < n; i++) {
    printf("Enter data for node %d: ", i + 1);
    scanf("%d", &data);

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

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

```
    }

    return head;
}
Node* insertAtBeginning(Node* head, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    return head;
}

Node* insertAtEnd(Node* head, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }

    return head;
}

Node* insertInMiddle(Node* head, int data, int position) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    if (position == 1) {
        newNode->next = head;
        head = newNode;
        return head;
    }

    Node* current = head;
```

```
    Node* prev = NULL;
    int count = 1;

    while (current != NULL && count < position) {
        prev = current;
        current = current->next;
        count++;
    }

    if (count < position) {
        printf("Invalid position for insertion.\n");
    } else {
        newNode->next = current;
        prev->next = newNode;
        printf("Node inserted at position %d.\n", position);
    }

    return head;
}

Node* deleteAtBeginning(Node* head) {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else {
        Node* temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted from the beginning.\n");
    }
    return head;
}

Node* deleteAtEnd(Node* head) {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Node deleted from the end.\n");
    } else {
        Node* current = head;
        while (current->next->next != NULL) {
```

```
            current = current->next;
        }
        free(current->next);
        current->next = NULL;
        printf("Node deleted from the end.\n");
    }

    return head;
}

Node* deleteFromMiddle(Node* head, int position) {
    if (head == NULL) {
        printf("List is empty. Nothing to delete.\n");
    } else if (position == 1) {
        head = deleteAtBeginning(head);
    } else {
        int count = 1;
        Node* current = head;
        Node* prev = NULL;

        while (current != NULL && count < position) {
            prev = current;
            current = current->next;
            count++;
        }

        if (current == NULL) {
            printf("Invalid position to delete.\n");
        } else {
            prev->next = current->next;
            free(current);
            printf("Node deleted from position %d.\n", position);
        }
    }
    return head;
}

void printList(Node* head) {
    printf("Linked List: ");
    Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
```

```
        current = current->next;
    }
    printf("\n");
}
```

Output: