EXP 6: Implementation of Singly Linked

Aim

To write and execute a C program that implements a **Singly Linked List** and performs the following operations using a menu-driven approach:

- Insert a node at the beginning
- Insert a node at the end
- Insert a node after a given node
- Delete a node from the beginning
- Delete a node from the end
- Delete a specified node by value
- Search an element in the linked list
- Display all elements of the linked list

Theory

A **Linked List** is a linear data structure used to store data in a sequence.It is made up of a group of **nodes**.

Each node has **two parts**:

1. **Data** – stores the value
2. **Link** – stores the address of the next node

The first node is called the **Head**.The last node contains **NULL** in its link part.

Linked list elements are **not stored in continuous memory locations**.Memory is allocated dynamically at runtime using pointers.

The main advantage of a linked list over an array is that **insertion and deletion are easy**. There is no need to shift elements or reallocate memory.Nodes can be inserted or deleted at any position in the list if the previous node is known.This makes linked lists more flexible than arrays.

However, linked lists have some disadvantages.They **do not support random access**.
To access an element, the list must be traversed from the beginning.Operations like searching, finding the last node, or inserting at a position require **sequential traversal**.

Thus, linked lists are efficient for dynamic data storage but slower for accessing elements compared to arrays.

Linked lists are used to implement **stacks, queues, trees, and graphs**.

Program

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

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

struct node *head = NULL;

/* Function declarations */
void insert_begin();
void insert_last();
void insert_after();
void delete_begin();
void delete_last();
void delete_specified();
void search();
void display();

int main() {
    int choice;

    do {
        printf("\n\n--- LINKED LIST MENU ---");
        printf("\n1. Insert at beginning");
        printf("\n2. Insert at last");
        printf("\n3. Insert after a given node");
        printf("\n4. Delete from beginning");
        printf("\n5. Delete from last");
        printf("\n6. Delete a specified node");
        printf("\n7. Search an element");
        printf("\n8. Display");
        printf("\n9. Exit");

        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1: insert_begin(); break;
            case 2: insert_last(); break;
            case 3: insert_after(); break;
            case 4: delete_begin(); break;
            case 5: delete_last(); break;
            case 6: delete_specified(); break;
            case 7: search(); break;
            case 8: display(); break;
            case 9: printf("Exiting program"); break;
            default: printf("Invalid choice");
        }
    } while(choice != 9);

    return 0;
}

/* Insert at beginning */
void insert_begin() {
    struct node *newnode;
    int value;

    newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter value: ");
    scanf("%d", &value);

    newnode->data = value;
    newnode->next = head;
    head = newnode;

    printf("Node inserted at beginning");
}

/* Insert at last */
void insert_last() {
    struct node *newnode, *temp;
    int value;

    newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter value: ");
    scanf("%d", &value);

    newnode->data = value;
    newnode->next = NULL;

    if (head == NULL) {
        head = newnode;
    } else {
        temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newnode;
    }
    printf("Node inserted at last");
}

/* Insert after a given node */
void insert_after() {
    struct node *temp, *newnode;
    int key, value;

    if (head == NULL) {
        printf("List is empty");
        return;
    }

    printf("Enter value after which to insert: ");
    scanf("%d", &key);

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

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

    newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter new value: ");
    scanf("%d", &value);

    newnode->data = value;
    newnode->next = temp->next;
    temp->next = newnode;

    printf("Node inserted after %d", key);
}

/* Delete from beginning */
void delete_begin() {
    struct node *temp;

    if (head == NULL) {
        printf("List is empty");
        return;
    }

    temp = head;
    head = head->next;
    free(temp);

    printf("Node deleted from beginning");
}

/* Delete from last */
void delete_last() {
    struct node *temp, *prev;

    if (head == NULL) {
        printf("List is empty");
        return;
    }

    if (head->next == NULL) {
        free(head);
        head = NULL;
    } else {
        temp = head;
        while (temp->next != NULL) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = NULL;
        free(temp);
    }
    printf("Node deleted from last");
}

/* Delete a specified node */
void delete_specified() {
    struct node *temp, *prev;
    int key;
    if (head == NULL) {
        printf("List is empty");
        return;
    }

    printf("Enter value to delete: ");
    scanf("%d", &key);

    if (head->data == key) {
        temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted");
        return;
    }

    temp = head;
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

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

    prev->next = temp->next;
    free(temp);
    printf("Node deleted");
}

/* Search an element */
void search() {
    struct node *temp;
    int key, pos = 1;

    if (head == NULL) {
        printf("List is empty");
        return;
    }

    printf("Enter element to search: ");
    scanf("%d", &key);

    temp = head;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Element found at position %d", pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf("Element not found");
}

/* Display */
void display() {
    struct node *temp;

    if (head == NULL) {
        printf("List is empty");
        return;
    }

    temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}
```