Data Structure and Algorithm (MCA 271)

Lab Practical –

BY

Himanshu Heda (24225013)

SUBMITTED TO

Prof. Vandna Kansal

SCHOOL OF SCIENCES

2024-2025

Program Description:

Code of the program

Output: - Paste the o/p of the program.

```c
// Implement linked list and it's operations:
// 1. Creation
// 2. Insertion of a node
// 3. Deletion of a node
// 4. Traversal

#include <stdio.h>

#define MAX_NODES 100 // Define the maximum number of nodes

// Define the node structure
struct Node {
    int data;
    int next; // Index of the next node in the array
};

// Define the linked list structure
struct LinkedList {
    struct Node nodes[MAX_NODES];
    int head; // Index of the head node
    int free; // Index of the first free node
};

// Function prototypes
void initList(struct LinkedList* list);
int insert(struct LinkedList* list, int data);
void deleteNode(struct LinkedList* list, int key);
void traverse(struct LinkedList* list);
void display(struct LinkedList* list);

int main() {
    struct LinkedList list;
    initList(&list); // Initialize the linked list
```

```c
int choice, data;

do {
    printf("\nLinked List Operations:\n");
    printf("1. Insert a node\n");
    printf("2. Delete a node\n");
    printf("3. Traverse the list\n");
    printf("4. Display the list\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            if (insert(&list, data)) {
                printf("Node inserted successfully.\n");
            } else {
                printf("Failed to insert node. List may be full.\n");
            }
            break;
        case 2:
            printf("Enter data to delete: ");
            scanf("%d", &data);
            deleteNode(&list, data);
            break;
        case 3:
            printf("Traversing the list:\n");
            traverse(&list);
            break;
        case 4:
            printf("Displaying the list:\n");
            display(&list);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
```

```
        }
    } while (choice != 5);

    return 0;
}

// Function to initialize the linked list
void initList(struct LinkedList* list) {
    list->head = -1; // List is initially empty
    list->free = 0;  // First free node is at index 0

    // Initialize the nodes
    for (int i = 0; i < MAX_NODES - 1; i++) {
        list->nodes[i].next = i + 1; // Link to the next free node
    }
    list->nodes[MAX_NODES - 1].next = -1; // Last node points to -1 (no next
node)
}

// Function to insert a new node at the end of the linked list
int insert(struct LinkedList* list, int data) {
    if (list->free == -1) {
        return 0; // List is full
    }

    int newNodeIndex = list->free; // Get the index of the new node
    list->free = list->nodes[newNodeIndex].next; // Update the free index

    list->nodes[newNodeIndex].data = data; // Set the data

    // If the list is empty, set the head to the new node
    if (list->head == -1) {
        list->head = newNodeIndex;
        list->nodes[newNodeIndex].next = -1; // No next node
    } else {
        // Find the last node
        int lastNodeIndex = list->head;
        while (list->nodes[lastNodeIndex].next != -1) {
            lastNodeIndex = list->nodes[lastNodeIndex].next;
        }
        // Link the new node at the end
```

```
        list->nodes[lastNodeIndex].next = newNodeIndex;
        list->nodes[newNodeIndex].next = -1; // New node is now the last
node
    }

    return 1; // Insertion successful
}

// Function to delete the first occurrence of a node with the given key
void deleteNode(struct LinkedList* list, int key) {
    int currNodeIndex = list->head;
    int prevNodeIndex = -1;

    // Search for the key to be deleted
    while (currNodeIndex != -1 && list->nodes[currNodeIndex].data != key) {
        prevNodeIndex = currNodeIndex;
        currNodeIndex = list->nodes[currNodeIndex].next;
    }

        // If the key was not found
    if (currNodeIndex == -1) {
        printf("Key %d not found in the list.\n", key);
        return;
    }

    // If the key is in the head node
    if (prevNodeIndex == -1) {
        // Change head
        list->head = list->nodes[currNodeIndex].next;
    } else {
        // Unlink the node from the linked list
        list->nodes[prevNodeIndex].next = list->nodes[currNodeIndex].next;
    }

    // Free the node by adding it to the free list
    list->nodes[currNodeIndex].next = list->free; // Link the freed node to
the free list
    list->free = currNodeIndex; // Update the free index

    printf("Node with key %d deleted.\n", key);
}
```

```c
// Function to traverse the linked list and print its elements
void traverse(struct LinkedList* list) {
    if (list->head == -1) {
        printf("The list is empty.\n");
        return;
    }

    int currentIndex = list->head;
    while (currentIndex != -1) {
        printf("%d -> ", list->nodes[currentIndex].data);
        currentIndex = list->nodes[currentIndex].next;
    }
    printf("NULL\n");
}

// Function to display the linked list
void display(struct LinkedList* list) {
    if (list->head == -1) {
        printf("The list is empty.\n");
        return;
    }

    int currentIndex = list->head;
    printf("Linked List: ");
    while (currentIndex != -1) {
        printf("%d ", list->nodes[currentIndex].data);
        currentIndex = list->nodes[currentIndex].next;
    }
    printf("\n");
}
```

OUTPUT : --

```
PS D:\2MCA\DSA> .\linked_list.exe

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert: 1
Node inserted successfully.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert: 2
Node inserted successfully.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert: 3
Node inserted successfully.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
```

```
Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert: 4
Node inserted successfully.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 4
Displaying the list:
Linked List: 1 2 3 4

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 2
Enter data to delete: 2
Node with key 2 deleted.

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
```

```
Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 4
Displaying the list:
Linked List: 1 3 4

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 3
Traversing the list:
1 -> 3 -> 4 -> NULL

Linked List Operations:
1. Insert a node
2. Delete a node
3. Traverse the list
4. Display the list
5. Exit
Enter your choice: 5
Exiting...
PS D:\2MCA\DSA>
```