**Practical No. 4**

Implement a single linked list and perform an operation like Insertion,Deletion and Transversal.

1. Program for Insertion

#include <stdio.h>

#include <stdlib.h>

// Definition of a Node

typedef struct Node {

    int value;

    struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int value) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->value = value;

    newNode->next = NULL;

    return newNode;

}

// Function to append a node to the end of the list

void append(Node\*\* head, int value) {

    Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

        return;

    }

    Node\* last = \*head;

    while (last->next != NULL) {

        last = last->next;

    }

    last->next = newNode;

}

// Function to insert a node at the beginning of the list

void insertAtBeginning(Node\*\* head, int value) {

    Node\* newNode = createNode(value);

    newNode->next = \*head;

    \*head = newNode;

}

// Function to insert a node after a specific node

void insertAfter(Node\* prevNode, int value) {

    if (prevNode == NULL) {

        printf("The given previous node cannot be NULL\n");

        return;

    }

    Node\* newNode = createNode(value);

    newNode->next = prevNode->next;

    prevNode->next = newNode;

}

// Function to insert a node at a specific position

void insertAtPosition(Node\*\* head, int position, int value) {

    if (position < 0) {

        printf("Position cannot be negative\n");

        return;

    }

    if (position == 0) {

        insertAtBeginning(head, value);

        return;

    }

    Node\* newNode = createNode(value);

    Node\* current = \*head;

    for (int i = 0; current != NULL && i < position - 1; i++) {

        current = current->next;

    }

    if (current == NULL) {

        printf("The previous node is NULL or position is out of range\n");

        free(newNode);

        return;

    }

    newNode->next = current->next;

    current->next = newNode;

}

// Function to traverse and print the linked list

void traverse(Node\* head) {

    Node\* current = head;

    while (current != NULL) {

        printf("%d -> ", current->value);

        current = current->next;

    }

    printf("NULL\n");

}

// Main function to demonstrate the linked list operations

int main() {

    Node\* head = NULL;

    // Append elements to the linked list

    append(&head, 1);

    append(&head, 3);

    append(&head, 4);

    printf("Linked list after appending 1, 3, 4:\n");

    traverse(head);  // Output: 1 -> 3 -> 4 -> NULL

    // Insert at the beginning

    insertAtBeginning(&head, 0);

    printf("Linked list after inserting 0 at the beginning:\n");

    traverse(head);  // Output: 0 -> 1 -> 3 -> 4 -> NULL

    // Insert after a specific node (node with value 1)

    Node\* temp = head;

    while (temp != NULL && temp->value != 1) {

        temp = temp->next;

    }

    if (temp != NULL) {

        insertAfter(temp, 2);

    }

    printf("Linked list after inserting 2 after 1:\n");

    traverse(head);  // Output: 0 -> 1 -> 2 -> 3 -> 4 -> NULL

    // Insert at a specific position (position 3)

    insertAtPosition(&head, 3, 5);

    printf("Linked list after inserting 5 at position 3:\n");

    traverse(head);  // Output: 0 -> 1 -> 2 -> 5 -> 3 -> 4 -> NULL

    // Free allocated memory

    Node\* current = head;

    Node\* next;

    while (current != NULL) {

        next = current->next;

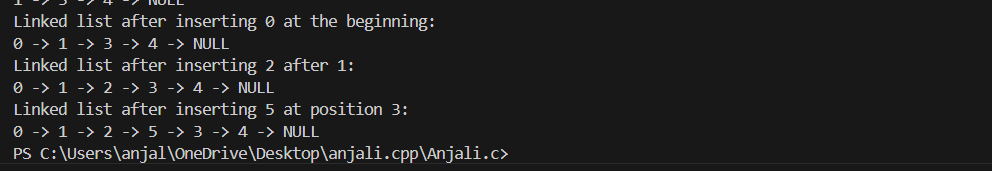
        free(current);

        current = next;

    }

    return 0;

}



1. Program for Deletion

#include <stdio.h>

#include <stdlib.h>

// Definition of a Node

typedef struct Node {

    int value;

    struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int value) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->value = value;

    newNode->next = NULL;

    return newNode;

}

// Function to append a node to the end of the list

void append(Node\*\* head, int value) {

    Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

        return;

    }

    Node\* last = \*head;

    while (last->next != NULL) {

        last = last->next;

    }

    last->next = newNode;

}

// Function to traverse and print the linked list

void traverse(Node\* head) {

    Node\* current = head;

    while (current != NULL) {

        printf("%d -> ", current->value);

        current = current->next;

    }

    printf("NULL\n");

}

// Function to delete a node with a specific value

void deleteNode(Node\*\* head, int value) {

    Node\* temp = \*head;

    Node\* prev = NULL;

    // If the head node itself holds the value to be deleted

    if (temp != NULL && temp->value == value) {

        \*head = temp->next; // Change head

        free(temp); // Free old head

        return;

    }

    // Search for the node to be deleted

    while (temp != NULL && temp->value != value) {

        prev = temp;

        temp = temp->next;

    }

    // If the value was not found

    if (temp == NULL) {

        printf("Value %d not found in the list.\n", value);

        return;

    }

    // Unlink the node from the linked list

    prev->next = temp->next;

    free(temp); // Free the node

}

// Main function to demonstrate the linked list operations

int main() {

    Node\* head = NULL;

    // Append elements to the linked list

    append(&head, 1);

    append(&head, 2);

    append(&head, 3);

    // Traverse the linked list

    printf("Linked list before deletion:\n");

    traverse(head);  // Output: 1 -> 2 -> 3 -> NULL

    // Delete a node with value 2

    deleteNode(&head, 2);

    // Traverse the linked list after deletion

    printf("Linked list after deleting 2:\n");

    traverse(head);  // Output: 1 -> 3 -> NULL

    // Free allocated memory

    Node\* current = head;

    Node\* next;

    while (current != NULL) {

        next = current->next;

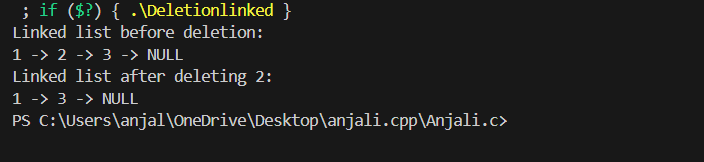
        free(current);

        current = next;

    }

    return 0;

}



1. Program for Traversal

#include <stdio.h>

#include <stdlib.h>

// Definition of a Node

typedef struct Node {

    int value;

    struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int value) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    if (!newNode) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->value = value;

    newNode->next = NULL;

    return newNode;

}

// Function to append a node to the end of the list

void append(Node\*\* head, int value) {

    Node\* newNode = createNode(value);

    if (\*head == NULL) {

        \*head = newNode;

        return;

    }

    Node\* last = \*head;

    while (last->next != NULL) {

        last = last->next;

    }

    last->next = newNode;

}

// Function to traverse and print the linked list

void traverse(Node\* head) {

    Node\* current = head;

    while (current != NULL) {

        printf("%d -> ", current->value);

        current = current->next;

    }

    printf("NULL\n");

}

// Main function to demonstrate the linked list operations

int main() {

    Node\* head = NULL;

    // Append elements to the linked list

    append(&head, 1);

    append(&head, 2);

    append(&head, 3);

    // Traverse the linked list

    traverse(head);  // Output: 1 -> 2 -> 3 -> NULL

    // Free allocated memory

    Node\* current = head;

    Node\* next;

    while (current != NULL) {

        next = current->next;

        free(current);

        current = next;

    }

    return 0;

}

