**NAME** :- Manish Shashikant Jadhav

**UID** :-

**BRANCH** :- Comps -B. **BRANCH:** B.

**EXPERIMENT 3: Implement of given problem statement using Linked List.**

**SUBJECT** :- DS (DATA STRUCTURES)

**TOPIC** **1** :- Implement a LinkedList ADT.

**CODE** :-

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

// create a LinkedList node with 'data'

struct Node {

    int data;

    struct Node \*next;

};

// create a LinkedList node with 'data'

struct Node\* create\_node(int data) {

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

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        exit(1);

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

// pos=-1 indicates insert at end

// pos=0 indicates add at beginning

// This creates a new LinkedList node and inserts it at position 'pos' in the current linked list originating from head

void insert\_at\_pos(struct Node\*\* head, int data, int pos) {

    if (pos < 0) {

        printf("Invalid position for insertion.\n");

        return;

    }

    struct Node\* newNode = create\_node(data);

    if (pos == 0) {

        newNode->next = \*head;

        \*head = newNode;

    } else {

        struct Node\* current = \*head;

        int i;

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

            current = current->next;

        }

        if (current == NULL) {

            printf("Invalid position for insertion.\n");

            return;

        }

        newNode->next = current->next;

        current->next = newNode;

    }

}

// pos=-1 indicates delete last node

// pos=0 indicates delete first node

// This deletes the LinkedList node at position 'pos' in the current linked list originating from head

void delete\_at\_pos(struct Node\*\* head, int pos) {

    if (\*head == NULL) {

        printf("List is empty, cannot delete.\n");

        return;

    }

    if (pos < 0) {

        printf("Invalid position for deletion.\n");

        return;

    }

    if (pos == 0) {

        struct Node\* temp = \*head;

        \*head = (\*head)->next;

        free(temp);

    } else {

        struct Node\* current = \*head;

        int i;

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

            current = current->next;

        }

        if (current == NULL || current->next == NULL) {

            printf("Invalid position for deletion.\n");

            return;

        }

        struct Node\* temp = current->next;

        current->next = temp->next;

        free(temp);

    }

}

// delete linkedlist node with first occurrence of given value from linked list originating from 'head'

void delete\_by\_value(struct Node\*\* head, int value) {

    struct Node\* current = \*head;

    struct Node\* prev = NULL;

    while (current != NULL) {

        if (current->data == value) {

            if (prev == NULL) {

                \*head = current->next;

            } else {

                prev->next = current->next;

            }

            free(current);

            return;

        }

        prev = current;

        current = current->next;

    }

    printf("Value not found in the list.\n");

}

// gets the node at position 'pos' in linkedlist originating from 'head'

// return 'null' if no node found along with informative message

struct Node\* get\_node\_at\_pos(struct Node\* head, int pos) {

    if (pos < 0) {

        printf("Invalid position.\n");

        return NULL;

    }

    struct Node\* current = head;

    int i;

    for (i = 0; i < pos && current != NULL; i++) {

        current = current->next;

    }

    if (current == NULL) {

        printf("Invalid position.\n");

        return NULL;

    }

    return current;

}

// Return the node with the first occurrence of value

// return 'null' if no node found along with informative message

int find\_first(struct Node\* head, int value) {

    int pos = 0;

    struct Node\* current = head;

    while (current != NULL) {

        if (current->data == value) {

            return pos;

        }

        current = current->next;

        pos++;

    }

    return -1; // Value not found

}

// display entire linked list, starting from head, in a well-formatted way

void display(struct Node\* head) {

    struct Node\* current = head;

    while (current != NULL) {

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

        current = current->next;

    }

    printf("NULL\n");

}

// deallocate the memory being used by the entire linkedlist, starting from head

void free\_linkedlist(struct Node\*\* head) {

    struct Node\* current = \*head;

    struct Node\* next;

    while (current != NULL) {

        next = current->next;

        free(current);

        current = next;

    }

    \*head = NULL;

}

// reverse a linkedlist - reverse a given linked list

void reverse(struct Node\* head) {

    struct Node\* prev = NULL;

    struct Node\* current = \*head;

    struct Node\* next;

    while (current != NULL) {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    \*head = prev;

}

int main() {

    struct Node\* head = NULL;

    insert\_at\_pos(&head, 28, 0);

    insert\_at\_pos(&head, 13, 1);

    insert\_at\_pos(&head, 9, 1);

    insert\_at\_pos(&head, 2, 1);

    insert\_at\_pos(&head, 15, 1);

    insert\_at\_pos(&head, 7, 1);

    printf("Initial linked list: ");

    display(head);

    delete\_at\_pos(&head, 1);

    printf("Linked list after deleting node at position 1: ");

    display(head);

    delete\_by\_value(&head, 20);

    printf("Linked list after deleting node with value 20: ");

    display(head);

    struct Node\* node\_at\_pos = get\_node\_at\_pos(head, 3);

    if (get\_node\_at\_pos != NULL) {

        printf("Node at position 3: %d\n", node\_at\_pos->data);

    }

    int pos = find\_first(head, 9);

    if (pos != -1) {

        printf("First occurrence of 9 is at position: %d\n", pos);

    } else {

        printf("Value 9 not found in the list.\n");

    }

    reverse(&head);

    printf("Reversed linked list: ");

    display(head);

    struct Node\* node\_at\_posi = get\_node\_at\_pos(head, 3);

    if (get\_node\_at\_pos != NULL) {

        printf("Node at position 3: %d\n", node\_at\_posi->data);

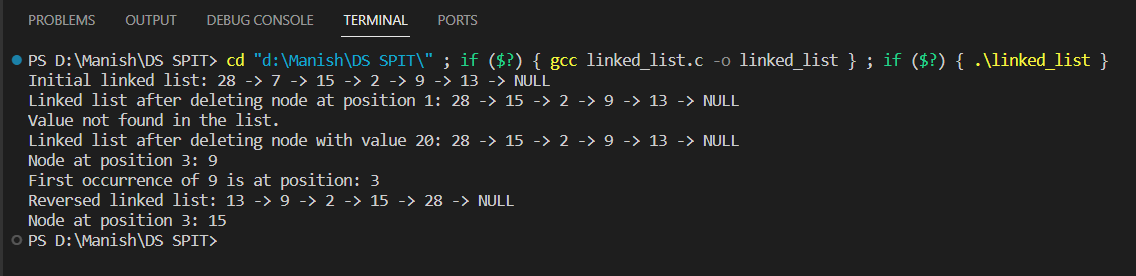
    }

    free\_linkedlist(&head);

    return 0;

}

**OUTPUT** :-



**Algorithm:**

**1. Structure Definition:**

- Define a structure `struct node` to represent a linked list node.

- Define a global variable `struct node head` to point to the head of the linked list.

- Define global variables `struct node temp` and `struct node prev` to assist in various operations.

**2. Creating a Node:**

- Define a function `struct node create\_node(int data)` to create a new node.

- Allocate memory for the new node using `malloc`.

- Initialize the `data` and `next` fields of the node.

- Return a pointer to the newly created node.

**3. Insertion at a Position:**

- Define a function `void insert\_at\_pos(int pos, int data)` to insert a node at a specified position.

- Check if `pos` is less than 0 to insert at the end, or if it's 0 to insert at the beginning.

- Traverse the list to find the node before the specified position.

- Adjust pointers to insert the new node in the correct position.

**4. Deletion at a Position:**

- Define a function `void delete\_at\_pos(struct node head, int pos)` to delete a node at a specified position.

- Handle cases for deleting the first node, last node, and a node in between.

- Free the memory allocated for the deleted node.

**5. Deletion by Value:**

- Define a function `void delete\_by\_value(int value)` to delete the first occurrence of a node with a given

value.

- Traverse the list to find the node with the specified value.

- Adjust pointers to remove the node.

**6. Getting a Node at a Position:**

- Define a function `struct node get\_node\_at\_pos(struct node head, int pos)` to get a node at a specified

position.

- Traverse the list to find the node at the specified position and return a pointer to it.

**7. Finding the First Node with a Value:**

- Define a function `struct node find\_first(struct node head, int value)` to find the first node with a given

value.

- Traverse the list and return a pointer to the first node with the specified value.

**8. Reversing the Linked List:**

- Define a function `struct node reverse(struct node head)` to reverse the linked list.

- Use three pointers (temp, prev, and nextnode) to reverse the pointers of each node.

- Update the head to point to the new first node (previously the last node).

**9. Displaying the Linked List:**

- Define a function `void display()` to display the linked list.

- Traverse the list and print the data of each node.

**10. Main Function:**

- In the `main` function:

- Perform a series of insertions, deletions, and other operations on the linked list.

- Display the linked list at various stages to demonstrate the changes.

- Reverse the linked list and display it.

- Free the memory used by the linked list to prevent memory leaks.

**TOPIC** **2** :- Add two positive numbers using linked lists.

**CODE** :-

#include <stdio.h>

#include <stdlib.h>

/\* Linked list node \*/

typedef struct Node {

    int data;

    struct Node\* next;

}Node;

/\* Function to create a

new node with given data \*/

Node\* newNode(int data)

{

    Node\* new\_node = (Node \*)malloc(sizeof(Node));

    new\_node->data = data;

    new\_node->next = NULL;

    return new\_node;

}

/\* Function to insert a node at the

beginning of the Singly Linked List \*/

void push(Node\*\* head\_ref, int new\_data)

{

    /\* allocate node \*/

    Node\* new\_node = newNode(new\_data);

    /\* link the old list of the new node \*/

    new\_node->next = (\*head\_ref);

    /\* move the head to point to the new node \*/

    (\*head\_ref) = new\_node;

}

/\* Adds contents of two linked lists and

return the head node of resultant list \*/

Node\* addTwoLists(Node\* first, Node\* second)

{

    // res is head node of the resultant list

    Node\* res = NULL;

    Node \*temp, \*prev = NULL;

    int carry = 0, sum;

    // while both lists exist

    while (first != NULL || second != NULL) {

        // Calculate value of next digit in resultant list.

        // The next digit is sum of following things

        // (i) Carry

        // (ii) Next digit of first list (if there is a next digit)

        // (ii) Next digit of second list (if there is a next digit)

        sum = carry + (first ? first->data : 0) + (second ? second->data : 0);

        // update carry for next calculation

        carry = (sum >= 10) ? 1 : 0;

        // update sum if it is greater than 10

        sum = sum % 10;

        // Create a new node with sum as data

        temp = newNode(sum);

        // if this is the first node then set it as head of the resultant list

        if (res == NULL)

            res = temp;

        // If this is not the first node then connect it to the rest.

        else

            prev->next = temp;

        // Set prev for next insertion

        prev = temp;

        // Move first and second pointers to next nodes

        if (first)

            first = first->next;

        if (second)

            second = second->next;

    }

    if (carry > 0)

        temp->next = newNode(carry);

    // return head of the resultant list

    return res;

}

Node\* reverse(Node\* head)

{

    if (head == NULL || head->next == NULL)

        return head;

    // reverse the rest list and put the first element at the end

    Node\* rest = reverse(head->next);

    head->next->next = head;

    head->next = NULL;

    // fix the head pointer

    return rest;

}

// A utility function to print a linked list

void printList(Node\* node)

{

    while (node != NULL) {

        printf("%d ",node->data);

        node = node->next;

    }

    printf("\n");

}

/\* Driver code \*/

int main(void)

{

    Node\* res = NULL;

    Node\* first = NULL;

    Node\* second = NULL;

    // create first list

    push(&first, 9);

    push(&first, 2);

    push(&first, 9);

    push(&first, 6);

    push(&first, 7);

    printf("First list is ");

    printList(first);

    // create second list

    push(&second, 7);

    push(&second, 2);

    push(&second, 1);

    printf("Second list is ");

    printList(second);

    // reverse both the lists

    first = reverse(first);

    second = reverse(second);

    // Add the two lists

    res = addTwoLists(first, second);

    // reverse the res to get the sum

    res = reverse(res);

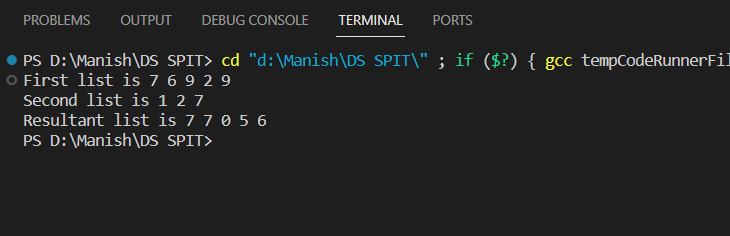
    printf("Resultant list is ");

    printList(res);

    return 0;

}

**OUTPUT** :-



**Algorithm:**

**1. Define a Node structure:**

- Create a structure named `Node` with two fields: `data` (integer data) and `next` (pointer to the next Node).

**2. Create a function to create a new Node:**

- Create a function named `newNode` that takes an integer `data` as an argument.

- Inside the function:

- Allocate memory for a new Node using `malloc`.

- Initialize the `data` field with the given data.

- Set the `next` field to `NULL`.

- Return the newly created Node.

**3. Create a function to push a Node to the beginning of a linked list:**

- Create a function named `push` that takes a double pointer to a Node (`Node\*\* head\_ref`) and an integer `new\_data`.

- Inside the function:

- Create a new Node using the `newNode` function with `new\_data`.

- Link the new Node to the existing list by setting its `next` field to the current head of the list.

- Update the head of the list to point to the new Node.

**4. Create a function to add two linked lists:**

- Create a function named `addTwoLists` that takes two pointers to Nodes (`first` and `second`) representing the linked lists to be added.

- Initialize a pointer `res` to `NULL` to store the result.

- Initialize two temporary pointers `temp` and `prev` to `NULL`.

- Initialize an integer `carry` to `0`.

- Use a loop to traverse both linked lists until both of them reach the end:

- Calculate the sum of the current nodes' data along with the carry.

- Update the carry if the sum is greater than or equal to 10.

- Create a new Node with the sum (modulo 10) as its data.

- If `res` is `NULL`, set `res` to the new Node; otherwise, link the `prev` Node to the new Node.

- Update `prev` to the new Node.

- Move to the next nodes in both `first` and `second` lists.

- After the loop, if there is a carry left, create a new Node with the carry and attach it to the end of the result list.

- Return `res` as the head of the resultant list.

**5. Create a function to reverse a linked list:**

- Create a function named `reverse` that takes a pointer to a Node (`head`) as input.

- Check if the linked list is empty or contains only one element. If so, return `head` as it is.

- Recursively reverse the rest of the list and put the first element at the end.

- Update the `next` pointers to reverse the list.

- Return the new head of the reversed list.

**6. Create a function to print a linked list:**

- Create a function named `printList` that takes a pointer to a Node (`node`) as input.

- Use a loop to traverse the linked list and print each node's data.

- Print a newline character to separate the lists.

**7. Main function:**

- Inside the `main` function:

- Create three pointers to Nodes: `res`, `first`, and `second`.

- Create the first linked list (`first`) by pushing elements onto it using the `push` function.

- Create the second linked list (`second`) in a similar manner.

- Reverse both linked lists using the `reverse` function.

- Add the two linked lists together using the `addTwoLists` function and store the result in `res`.

- Reverse the result again to get the final sum.

- Print the original lists and the resultant list.

**8. Return 0 to indicate successful execution.**

