ASSIGNMENT

Linked List

```c
/*
1. Representation of Linked list node in c

struct Node{
    //data fiels
    int a;
    //pointer field (points to the next node)
    struct node *next;
};
2.creating a node for a linked list in c
//*node1 is a user defined datatype
struct node *node1=(struct node *)malloc(sizeof(struct node))  //created a memory for a node
and address is stored in *node1;
3.shortening the node declaration
typedef struct Node{
    //data fiels
    int a;
    //pointer field (points to the next node)
    struct node *next;
}Node;
Node *node1=(Node *)malloc(sizeof(Node));
4.Assigning values to the member elements of the Node
node1->a=10;
node1->next=NULL;
*/
#include<stdio.h>
#include<stdlib.h>
//Define the structure of the node1
typedef struct Node{
    //data fiels
    int data;
    //pointer field (points to the next node)
    struct node *next;
}Node;
int main(){
    //Creating the first Node
    Node *first=(Node *)malloc(sizeof(Node));
    //Assigning the data
    first->data=10;
    //Creating the second Node
    Node *second=(Node *)malloc(sizeof(Node));
    //Assigning the data
    second->data=20;
    //Creating the third Node
    Node *third=(Node *)malloc(sizeof(Node));
    //Assigning the data
    third->data=30;
/*
3 nodes are created first -10,second-20,third-30
*/
//linking of nodes
first->next=second;//this creates link b/w first->second
second->next=third;//second->third
third->next=NULL;//third->null
//first->second->third
//Printing the linked list
/*
1.traverse from first to third
    a)create a temporary pointer of type struct node
        temp  first->second->third
                10      20      30
    b)make the temporary pointer point to the first
        temp ->first->second->third
                10      20      30
    c)move the temppointer from first to third node for printing the entire linked list
        loop
        loop !=NULL
*/
Node *temp;
temp = first;
while(temp!=NULL){
    printf("%d ->",temp->data);
    temp=temp->next;
}
return 0;
}
```

Output:

```
10 ->20 ->30 ->
```

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

typedef struct Node{
    int data;
    struct Node *next;
}Node;
Node* createNode(int data);
int main(){
    Node *first=createNode(10);
    first->next=createNode(20);
    first->next->next=createNode(30);
    first->next->next->next=createNode(40);
    Node *temp;
    temp=first;
    while(temp!=NULL){
        printf("%d->",temp->data);
        temp=temp->next;
    }
    return 0;
}
Node* createNode(int data){
    Node *newNode=(Node *)malloc(sizeof(Node));
    newNode->data=data;
    //Initially assigning the next field of the newly created node to NULL
    newNode->next=NULL;
    return newNode;
}
```

Create a node in a linked list which will have the following details of student

1. Name, roll number, class, section, an array having marks of any three subjects Create a liked for 5 students and print it.

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

typedef struct node
{
    char name[50];
    int roll_no;
    int class;
    char section;
    int marks[3];
    struct node *link;
}student;
int main()
{
    student *head = NULL;
    printf("Enter details of 5 students:\n");
    for(int i=0; i<5; i++)
    {
        student *new = (student *)malloc(sizeof(student));
        printf("\nStudent %d\n", i+1);
        printf("Enter the name of student: ");
        scanf(" %[^\n]", new->name);
        printf("Enter the roll no: ");
        scanf(" %d",&new->roll_no);
        printf("Enter the class and section: ");
        scanf("%d %c", &new->class, &new->section);
        printf("Enter the marks of 3 subjects: \n");
        for(int j=0; j<3; j++)
        {
            printf("Subject %d: ", j+1);
            scanf("%d",&new->marks[j]);
        }
        new->link = NULL;

        if(head == NULL)
        {
            head = new;
        }
        else
        {
            student *temp = head;
            while(temp->link != NULL)
            {
                temp = temp->link;
            }
            temp->link = new;
        }

    }

    //Display details

    printf("\nStudent details\n");
    student *temp = head;
    int i=1;
    while(temp != NULL)
    {
        printf("\nStudent %d\n", i);
        printf("Name: %s\n", temp->name);
        printf("Roll no: %d\n", temp->roll_no);
        printf("Class %d, Sec: %c\n", temp->class, temp->section);
        printf("Marks: %d %d %d\n", temp->marks[0], temp->marks[1], temp->marks[2]);
        i++;
        temp = temp->link;
    }
    return 0;
}
```

Linked List - Insertion of Nodes

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

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

//Function with dual purpose: Creating a new node also adding a new node at the beginning
void InsertFront(Node** ,int );
void InsertMiddle(Node* , int);
//Function with dual purpose: Creating a new node also adding a new node at the end
void InsertEnd(Node**, int);
void printList(Node*);

int main(){
    Node* head = NULL;
    InsertEnd(&head, 6);
    InsertEnd(&head, 1);
    InsertEnd(&head, 5);
    InsertFront(&head, 7 );
    InsertFront(&head, 10 );
    InsertMiddle(head->next,8);
    printList(head);
    return 0;
}

void InsertEnd(Node** ptrHead, int nData){
    //1.Creating a Node
    Node* new_node=(Node *)malloc(sizeof(Node));
    //1.1 Create one more pointer which will point to the last element of the linked list
    Node* ptrTail;
    ptrTail = *ptrHead;
    //2.Enter nData
    new_node->data = nData;
    //3. we have to make the next field as NULL
    new_node->next = NULL;
    //4. If the linked list is empty make ptrHead point to thge new node created
    if(*ptrHead == NULL){
        *ptrHead = new_node;
        return;
    }
    //5. else Traverse till the last node and insert the new node at the end
    while(ptrTail->next != NULL){
        //5.1 MOve the ptrTail pinter till the end
        ptrTail = ptrTail->next;
    }
    ptrTail->next = new_node;
return;
}

void InsertFront(Node** ptrHead,int nData){
     //1. Create a New Node
     Node* new_node = (Node*)malloc(sizeof(Node));
     //2. Assign Data to the new Node
     new_node->data = nData;
     //3. Make the new node point to the first node of the linked list
     new_node->next = (*ptrHead);
     //4. Assign a the address of new Node to ptrHead
    (*ptrHead) = new_node;
}
void InsertMiddle(Node* prev_node,int nData){
    Node* new_node=(Node*)malloc(sizeof(Node));
    new_node->data=nData;
    new_node->next=prev_node->next;
    prev_node->next=new_node;
}
void printList(Node* node){
    while (node != NULL){
        printf("%d ->",node->data);
        node = node->next;
    }
}
```

Output:

```
10 ->7 ->8 ->6 ->1 ->5 ->
```

Problem 1: Reverse a Linked List

Write a C program to reverse a singly linked list. The program should traverse the list, reverse the pointers between the nodes, and display the reversed list.

Requirements:

1. Define a function to reverse the linked list iteratively.
2. Update the head pointer to the new first node.
3. Display the reversed list.

Example Input:

rust

Copy code

Initial list: 10 -> 20 -> 30 -> 40

Example Output:

rust

Copy code

Reversed list: 40 -> 30 -> 20 -> 10

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

```c
#include <stdlib.h>
typedef struct node {
    int data;
    struct node *next;
} Node;
Node* createNode(int data);
void insertEnd(Node **head, int data);
void reverseList(Node **head);
void displayList(Node *head);
int main() {
    Node *head = NULL;
    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    insertEnd(&head, 40);
    printf("Initial list: ");
    displayList(head);
    reverseList(&head);
    printf("Reversed list: ");
    displayList(head);
    return 0;
}
Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(Node **head, int data) {
    Node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void reverseList(Node **head) {
    Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}
void displayList(Node *head) {
    Node *temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" -> ");
        }
        temp = temp->next;
    }
    printf("\n");
}
```

Problem 2: Find the Middle Node

Write a C program to find and display the middle node of a singly linked list. If the list has an even number of nodes, display the first middle node.

Requirements:

1. Use two pointers: one moving one step and the other moving two steps.
2. When the faster pointer reaches the end, the slower pointer will point to the middle node.

Example Input:

rust

Copy code

List: 10 -> 20 -> 30 -> 40 -> 50

Example Output:

scss

Copy code

Middle node: 30

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

typedef struct node {
    int data;
    struct node *next;
} Node;
Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(Node **head, int data);
void findMiddle(Node *head) ;
void displayList(Node *head);
int main() {
    Node *head = NULL;
    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    insertEnd(&head, 40);
    insertEnd(&head, 50);
    printf("List: ");
    displayList(head);
    findMiddle(head);
    return 0;
}
void insertEnd(Node **head, int data) {
    Node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void findMiddle(Node *head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    Node *slow = head;
    Node *fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }
    printf("Middle node: %d\n", slow->data);
}
void displayList(Node *head) {
    Node *temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" -> ");
        }
        temp = temp->next;
    }
    printf("\n");
}
```

Problem 3: Detect and Remove a Cycle in a Linked List

Write a C program to detect if a cycle (loop) exists in a singly linked list and remove it if present. Use Floyd’s Cycle Detection Algorithm (slow and fast pointers) to detect the cycle.

Requirements:

1. Detect the cycle in the list.
2. If a cycle exists, find the starting node of the cycle and break the loop.
3. Display the updated list.

Example Input:

rust

Copy code

List: 10 -> 20 -> 30 -> 40 -> 50 -> (points back to 30)

Example Output:

rust

Copy code

Cycle detected and removed.

Updated list: 10 -> 20 -> 30 -> 40 -> 50

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

typedef struct Node {
    int data;
    struct Node* next;
} Node;
Node* createNode(int data);
void detectAndRemoveCycle(Node* head);
void printList(Node* head);

int main() {

    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);
    head->next->next->next->next = createNode(50);
    head->next->next->next->next->next = head->next->next;

    detectAndRemoveCycle(head);
    printf("Updated list: ");
    printList(head);
    return 0;
}
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to detect and remove the cycle in the linked list
void detectAndRemoveCycle(Node* head) {
    Node *slow = head, *fast = head;
    int cycleDetected = 0;
    // Detect the cycle using Floyd's Cycle Detection Algorithm
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
        // Cycle detected
        if (slow == fast) {
            cycleDetected = 1;
            break;
        }
    }
    if (cycleDetected) {
        printf("Cycle detected.\n");
        // Find the start of the cycle
        slow = head;
        Node* prev = NULL;
        while (slow != fast) {
            prev = fast;
            slow = slow->next;
            fast = fast->next;
        }
        // Remove the cycle
        prev->next = NULL;
        printf("Cycle removed.\n");
    } else {
        printf("No cycle detected.\n");
    }
}
// Function to print the linked list
void printList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```