PHASE 1 DAY 16

DATA STRUCTURES

LINKED LIST

```
/*


struct Node
{


        //Data Fields

        int a;
        struct Node *next;


};
```

2. Creating a Node for a Linked List in C

```
struct Node *node1 = (struct Node *)malloc(sizeof(struct Node));
```

3. Shortening the Node Declaration

```
typedef struct Node{


        //Data Fields
```

```
        int a;

        //Pointer Field (Points to the next node)
        struct Node *next;
}Node;

Node *node1 = (Node*) malloc(sizeof(Node));
```

4. Assignimg values to the member elements of the Node

```
node1->a = 10;
node1->next = NULL;
/*
1.Representation of linked List Node in c
struct Node{
        //Data Fields
        int a;

        //Pointer Field (Points to the next node)
        struct Node *next;
```

```
};
```

2. Creating a Node for a Linked List in C

```
struct Node *node1 = (struct Node *)malloc(sizeof(struct Node));
```

3. Shortening the Node Declaration

```
typedef struct Node{

        //Data Fields

        int a;

        //Pointer Field (Points to the next node)

        struct Node *next;

}Node;

Node *node1 = (Node*) malloc(sizeof(Node));
```

4. Assignimg 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 Fields

        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;

/*

first          second        third

10             20            30

*/

//Linking of Nodes

first->next = second;   //this create link between first -> second

second->next = third;   // second -> third

third->next = NULL;  //third -> NULL

/*
```

first -> second -> third

10 20 30

*/

// 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 temporatry pointer point to the first

temp -> first -> second -> third

10 20 30

c. Move the temp pointer from first to third node for priting the entire

linked list

loop

loop != NULL

*/

Node *temp;

temp = first;

```
        while(temp != NULL){

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

        temp = temp->next;

        }

        return 0;

}
```

CREATE NODE

1.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 linked list for 5 students and print it.

```
#include <stdio.h>

#include <stdlib.h>


typedef struct Student

{

    char name[50];

    int roll_number;

    int class;

    char section;

    int marks[3];

} Student;


typedef struct Node
```

```
{
    Student student;
    struct Node* next;
} Node;

Node* createNode(Student student)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->student = student;
    newNode->next = NULL;
    return newNode;
}

void add(Node** head, Student student)
{
    Node* newNode = createNode(student);
    if (*head == NULL)
    {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
```

```
        temp->next = newNode;
    }
}

void printList(Node* head)
{
    while (head != NULL)
{
        printf("Name: %s, Roll No: %d, Class: %d, Section: %c, Marks: [%d, %d, %d]\n",
               head->student.name, head->student.roll_number, head->student.class,
               head->student.section, head->student.marks[0],
               head->student.marks[1], head->student.marks[2]);
        head = head->next;
    }
}

int main()
{
    Node* head = NULL;

    for (int i = 0; i < 5; i++)
    {
        Student s;
        printf("Enter details for student %d:\n", i + 1);
        printf("Name: ");
```

```
        scanf(" %[^\n]", s.name);
        printf("Roll Number: ");
        scanf("%d", &s.roll_number);
        printf("Class: ");
        scanf("%d", &s.class);
        printf("Section: ");
        scanf(" %c", &s.section);
        printf("Marks (3 subjects): ");
        scanf("%d %d %d", &s.marks[0], &s.marks[1], &s.marks[2]);
        add(&head, s);
    }

    printf("\nStudent Details:\n");
    printList(head);

    Node* temp;
    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

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

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

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

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

Example Output:

Cycle detected and removed.

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

Answers:

```
#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
//Function with dual purpose: Creating a new node also adding a new node at the end
 void InsertFront(Node**, int);
void InsertMiddle(Node**,int);
void InsertEnd(Node**, int);
void printList(Node*);
void reverseList(Node** head);
void findMiddle(Node* head);
int main()
{
    int choice,n;
    Node* head = NULL;
 while(1)
 {
     printf("Choose operation from Menu:\n1.Insert End\n2.Insert Front\n3.print List\n4.Insert Middle\n5.Reverse List\n6.Find Middle\nEnter your Choice:");
     scanf("%d",&choice);
     switch(choice)
     {
         case 1:
         {
```

```
    printf("Enter value to insert at the end: ");
   scanf("%d", &n);
   InsertEnd(&head,n);
    break;
}
case 2:
{
   printf("Enter value to insert in the front: ");
   scanf("%d", &n);
   InsertFront(&head,n);
    break;
}
case 3:
{
   printList(head);
            break;
}
case 4:
{
   printf("Enter value to insert in the middle: ");
   scanf("%d", &n);
   InsertMiddle(&head,n);
    break;
}
case 5:
```

```
        {
           reverseList(&head);
            break;
        }
        case 6:
        {
             findMiddle(head);
            break;
        }
        default:
        printf("invalid");
    }
 }
    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 printList(Node* node)
{
    if(node==NULL)
    {
        printf("list empty");
    }
    while (node != NULL)
    {
        printf("%d ->",node->data);
        node = node->next;
    }
    printf("\n");
}
void InsertMiddle(Node**ptrHead, int nData)
{
    int data;
    Node* new_node = (Node*)malloc(sizeof(Node));
    Node* temp = *ptrHead;

    printf("Enter the node value after which the new node is to be inserted: ");
    scanf("%d", &data);
    while (temp != NULL && temp->data != data) {
        temp = temp->next;
```

```
    }
    if (temp == NULL)
    {
        printf("Node with value %d not found. No insertion performed.\n", data);
        free(new_node);
        return;
    }
      new_node->data = nData;
    new_node->next = temp->next;
    temp->next = new_node;
}
void reverseList(Node** head)
{
    Node* prev = NULL;
    Node* current = *head;
    Node* next = NULL;

    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
```

```
    *head = prev;
    printf("\nReversed list: ");
    printList(*head);
}

void findMiddle(Node* head)
{
    Node* ptr1 = head;
    Node* ptr2 = head;

    while (ptr2 != NULL && ptr2->next != NULL)
    {
        ptr1 = ptr1->next;
        ptr2 = ptr2->next->next;
    }

    if (ptr1 != NULL)
    {
        printf("\nMiddle node: %d\n", ptr1->data);
    }
}
```