LINKED LIST
-linear data structure
-each node has its own data and address of next node
-forms a chain like structure

Format
/*
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;
```

Eg

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
   int a;
   //pointer to next node
   struct Node *next;
}Node;
int main(){
   //creating first node
   Node *first=(Node*)malloc(sizeof(Node));
   //assigning the data
   first->a=10;
   //creating second node
   Node *second=(Node*)malloc(sizeof(Node));
   //assigning the data
   second->a=20;
```

```
  //creating third node
  Node *third=(Node*)malloc(sizeof(Node));
  //assigning the data
  third->a=30;

  /*
    3 nodes are created here
    first          second          third
    10             20              30

  */
  //linking of nodes
  first->next=second;
  second->next=third;
  third->next=NULL;
   /*
    3 nodes are created here
    first    ->     second    ->    third
    10              20              30

  */
  /*traversing from first to third
    a.create a temprory pointer of type struct node
    temp      first    ->     second    ->    third
          10            20            30

    b.make temp pointer point to first
    temp   -> first    ->     second    ->    third
          10            20            30

    c.move the temp pointer from first to third node for printing entire linked list
    loop
    loop !=NULL
  */
  Node *temp;
  temp=first;
  while(temp !=NULL){
    printf("%d->",temp->a);
    temp=temp->next;
  }
  return 0;



}
```
_-
```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
  int a;
  //pointer to next node
  struct Node *next;
}Node;
Node* createNode(int);//pointer function is used since each function call returns the address
int main(){
  //10->null
  Node *first=createNode(10);
```

```c
  //10->20->null
  first->next=createNode(20);
  //10->20->30->null
  first->next->next=createNode(30);

  Node *temp=first;
  while(temp!=NULL){
    printf("%d->",temp->a);
    temp=temp->next;
  }
  return 0;
}
Node *createNode(int a){
  Node *newNode=(Node*)malloc(sizeof(Node));
  newNode->a=a;
  //initially assigning the next field of 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[20];
  int roll;
  int class;
  char section;
  int marks[3];
  struct Node *next;
}NODE;
NODE* create_node();
int main(){
  NODE *first=create_node();
  first->next=create_node();
  first->next->next=create_node();
  // first->next->next->next=create_node();
  // first->next->next->next->next=create_node();
  NODE *temp=first;
  while(temp!=NULL){
    printf("name|\t|roll|\t|class|\t|section|\n");
    printf("%s|\t|%d|\t|%d|\t|%c\n",temp->name,temp->roll,temp->class,temp->section);
    printf("maths marks|\t|physics marks|\t|chem marks|\n");
    printf("%d|\t|%d|\t|%d|\n",temp->marks[0],temp->marks[1],temp->marks[2]);
    printf("------------------------------------------------------\n");
    temp=temp->next;

  }
  return 0;
}
NODE *create_node(){
  NODE *newNode=(NODE*)malloc(sizeof(NODE));
  printf("enter the name of student:");
```

```
    scanf(" %[^\n]",newNode->name);
    printf("enter the roll no of student:");
    scanf("%d",&newNode->roll);
    printf("enter the class of student:");
    scanf("%d",&newNode->class);
    printf("enter the section of student:");
    scanf(" %c",&newNode->section);
    printf("enter marks of 3 subject\n");
    printf("enter mark of maths:");
    scanf("%d",&newNode->marks[0]);
    printf("enter mark of physics:");
    scanf("%d",&newNode->marks[1]);
    printf("enter mark of chem:");
    scanf("%d",&newNode->marks[2]);
    newNode->next=NULL;
    return newNode;
}
```

_-

```
#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, 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 );
    printf("Initial list:\n");
    printList(head);
    printf("\nAfter inserting 15 at position 3:\n");
    InsertMiddle(head, 15, 3); // Insert 15 at position 3
    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 printList(Node* node){
  while (node != NULL){
    printf("%d ->",node->data);
    node = node->next;
  }
}
// Insert a node in the middle at a specified position
void InsertMiddle(Node *head, int nData, int position) {
  Node *new_node = (Node *)malloc(sizeof(Node));
  new_node->data = nData;

  // Handle special case: insert at position 1 (equivalent to InsertFront)
  if (position == 1) {
    new_node->next = head;
    head = new_node;
    return;
  }

  Node *current = head;
  int count = 1;

  // Traverse to the node just before the target position
  while (current != NULL && count < position - 1) {
    current = current->next;
    count++;
  }
  // If the position is invalid (greater than the length of the list), do nothing
  if (current == NULL) {
    printf("Position %d is out of range.\n", position);
    free(new_node);
    return;
  }
  // Insert the new node
  new_node->next = current->next;
  current->next = new_node;
}
```

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>
#include <stdlib.h>

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


void InsertEnd(Node **, int);
void printList(Node *);
void reverseList(Node **);

int main() {
   Node *head = NULL;
   InsertEnd(&head, 10);
   InsertEnd(&head, 20);
   InsertEnd(&head, 30);
   InsertEnd(&head, 40);

   printf("Initial list:\n");
   printList(head);


   reverseList(&head);

   // Display the reversed list
   printf("\nReversed list:\n");
   printList(head);

   return 0;
}


void InsertEnd(Node **ptrHead, int nData) {
   Node *new_node = (Node *)malloc(sizeof(Node));
   new_node->data = nData;
   new_node->next = NULL;
```

```
  if (*ptrHead == NULL) {
    *ptrHead = new_node;
    return;
  }

  Node *ptrTail = *ptrHead;
  while (ptrTail->next != NULL) {
    ptrTail = ptrTail->next;
  }
  ptrTail->next = new_node;
}


void printList(Node *node) {
  while (node != NULL) {
    printf("%d -> ", node->data);
    node = node->next;
  }
  printf("NULL\n");
}


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;
}
```

_-

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;
void InsertEnd(Node **, int);
void printList(Node *);
void findMiddle(Node *);

int main() {
   InsertEnd(&head, 10);
   InsertEnd(&head, 20);
   InsertEnd(&head, 30);
   InsertEnd(&head, 40);
   InsertEnd(&head, 50);
   printf("List:\n");
   printList(head);

   findMiddle(head);

   return 0;
}


void InsertEnd(Node **ptrHead, int nData) {
   Node *new_node = (Node *)malloc(sizeof(Node));
   new_node->data = nData;
   new_node->next = NULL;

   if (*ptrHead == NULL) {
      *ptrHead = new_node;
      return;
   }

   Node *ptrTail = *ptrHead;
   while (ptrTail->next != NULL) {
      ptrTail = ptrTail->next;
   }
   ptrTail->next = new_node;
}


void printList(Node *node) {
   while (node != NULL) {
      printf("%d -> ", node->data);
      node = node->next;
   }
   printf("NULL\n");
}


void findMiddle(Node *head) {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
```

```c
    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);
}
```

-_-

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