LINKED LIST

/*
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 node
```

```
typedef struct Node{
    int data;
    struct Node *next;
}Node;
int main(){
    //Creating first node
    Node *first = (Node*)malloc(sizeof(Node));
    //Assigning the data
    first->data = 10;
    //Creating second node
    Node *second = (Node*)malloc(sizeof(Node));
    //Assigning the data
    second->data = 20;
    //Creating third node
    Node *third = (Node*)malloc(sizeof(Node));
    //Assigning the data
    third->data = 30;

    //Linking of nodes
    first->next = second; //create link b/w first & second
    second->next = third; //link b/w second & third
    third->next = NULL; // third pointing to NULL

    //Printing the linked list
    /*
    1.traverse from first to third
        a.create a temporary pointer of type struct Node
        b.Make the temporary pointer point to the first
        c.Move the temp pointer from first to third node printing the linked list
            use loop until loop!=NULL
    */
    Node *temp;
    temp = first;
    while(temp!=NULL){
        printf("%d ->",temp->data);
        temp = temp->next;
    }
    return 0;
}

#include<stdio.h>
#include<stdlib.h>
//Define the structure of node
typedef struct Node{
    int data;
    struct Node *next;
}Node;
Node* createNode(int data);
int main(){
    //10->NULL
    Node *first = createNode(10);
    //10->20->NULL
    first->next = createNode(20);
    //10->20->-30->NULL
    first->next->next = createNode(30);

    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;
}
```

<u>Insertion</u>

```
#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 );
    InsertMiddle(&head, 6, 4);
    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 **ptrHead,int target,int nData){
   Node *temp = *ptrHead;
   while(temp!=NULL && temp->data!=target){
      temp=temp->next;
   }
   if(temp!=NULL){
      Node *new_node = (Node *)malloc(sizeof(Node));
      new_node->data = nData;
      new_node->next = temp->next;
      temp->next = new_node;
   }
   else{
      printf("Target not found\n");
   }
}
void printList(Node* node){
   while (node != NULL){
      printf("%d ->",node->data);
      node = node->next;
   }
}
```

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 reverseList(Node**);
void printList(Node*);

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

```
    InsertEnd(&head,40);
    printf("Initial list: ");
    printList(head);
    reverseList(&head);
    printf("Reversed list: ");
    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 reverseList(Node ** ptrHead){
   Node *last = NULL;
   Node *current = *ptrHead;
   Node *next = NULL;

   while(current!=NULL){
      next = current->next;
      current->next = last;
      last = current;
      current = next;
   }
   *ptrHead = last;
}
void printList(Node* node){
   while (node != NULL){
      printf("%d ->",node->data);
      node = node->next;
   }
}
```

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
```

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct node{
    int data;
    struct node *next;
}Node;
void InsertEnd(Node**, int);
void findmiddle(Node*);
void printList(Node*);

int main(){
    Node* head = NULL;
    InsertEnd(&head,10);
    InsertEnd(&head,20);
    InsertEnd(&head,30);
    InsertEnd(&head,40);
    InsertEnd(&head,50);
    printf("List: ");
    printList(head);
    findmiddle(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 findmiddle(Node *head){

    if(head==NULL){

        printf("List is empty");

    }

    Node* slow = head;

    Node* fast = head;

    while(fast!=NULL && fast->next!=NULL){

        slow = slow->next;

        fast = fast->next->next;

    }

    printf("\nMiddle node = %d",slow->data);

}
```

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

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
```

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

typedef struct node{
   int data;
   struct node *next;
}Node;
void InsertEnd(Node**, int);
void printList(Node*);
int removecycle(Node*);
void createcycle(Node*);

int main(){
   Node* head = NULL;
   InsertEnd(&head,10);
   InsertEnd(&head,20);
   InsertEnd(&head,30);
   InsertEnd(&head,40);
   InsertEnd(&head,50);
   createcycle(head);
   removecycle(head);
   printf("Updated list: ");
   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 printList(Node* node){
    while (node != NULL){
        printf("%d ->",node->data);
        node = node->next;
    }
}
int removecycle(Node *head){
    Node* slow = head;
    Node* fast = head;
    while(fast!=NULL && fast->next!=NULL){
        slow = slow->next;
        fast = fast->next->next;

        if(slow==fast){
        break;
    }
    }

    slow = head;
    while(slow!=fast){
        slow = slow->next;
        fast = fast->next;
    }
    Node *cyclestart = slow;
    Node *temp = cyclestart;
    while(temp->next!=cyclestart){
        temp = temp->next;
    }
    temp->next = NULL;
}
void createcycle(Node* head) {
    Node* temp = head;
    while (temp != NULL && temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head->next->next;  // 30 -> points back to 30
}
```