/*
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){
      if(temp->next != NULL){
      printf("%d -> ",temp->data);
```

```
        }
        else
        {
            printf("%d ",temp->data);
        }
        temp = temp->next;
    }
    return 0;
}
```

1)

```
#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;
}


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

typedef struct node{
   int data;
   struct node *link;

}Node;

void InsertLast(Node **head,int data);
void insertLast(Node **head,int data);
void insertMiddle(Node **head,int data,int val_before,int val_after);
void printlist(Node *);
int main(){
   Node *head = NULL;
   while(1){


   int op;
   printf("Enter operation (1 for InsertLast, 3 for PrintList, 0 to Exit): ");
   scanf("%d",&op);


   if(op == 1)
   {
   int data;
```

```
    printf("Enter data to insert: ");
    scanf("%d",&data);
    InsertLast(&head,data);
    }
    else if(op == 2){
        int data;
        printf("Enter data to insert: ");
        scanf("%d",&data);
        insertLast(&head,data);
    }
    else if(op == 3){
        int data,val_before,val_after;
        printf("Enter data to insert: ");
        scanf("%d",&data);
        printf("Enter the values before the data and after the data: ");
        scanf("%d %d",&val_before,&val_after);
        insertMiddle(&head,data,val_before,val_after);
    }

    else if(op == 4){
        printlist(head);
    }
    else if(op ==0)
    {
        printf("Exiting\n");
        break;
    }
    }
}

void InsertLast(Node **head,int data){
    Node *new = (Node *)malloc(sizeof(Node));
    if(new == NULL){
        return;

    }
    else{
        new->data = data;
        new->link = NULL;

    }

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

void insertLast(Node **head,int data)
{
    Node *new = (Node *)malloc(sizeof(Node));
```

```
   if(new == NULL){
      return;

   }
   else{
      new->data = data;
      new->link = NULL;

   }

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

void printlist(Node *head){

   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }

   while(head != NULL){
      printf("%d->",head->data);
      head = head->link;
   }
   printf("NULL\n");
}

void insertMiddle(Node **head,int data,int val_before,int val_after)
{
   Node *new = (Node *)malloc(sizeof(Node));
   if(new == NULL){
      return;

   }
   else{
      new->data = data;
      new->link = NULL;

   }

   if(*head == NULL)
   {
      printf("List is empty!\n");
   }


      Node *prev = *head;
      while(prev != NULL && prev->link != NULL){
         if(prev->data == val_before  && prev->link->data == val_after){
```

```
            new->link = prev->link;
            prev->link = new;
            printf("Node inserted successfully.\n");
            return;

          }

      prev = prev->link;
      }
}
```

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 *link;

}Node;

void InsertLast(Node **head,int data);
void insertFirst(Node **head,int data);
void insertMiddle(Node **head,int data,int val_before,int val_after);
void reverse(Node **head);
void printlist(Node *);
int main(){
    Node *head = NULL;
    while(1){


    int op;
    printf("Enter operation (\n1 for InsertLast\n 2for Insert first\n3 for Insert Middle
\n4 for PrintList\n5 for Reverse Linked lis\n 0 to Exit\n ");
    scanf("%d",&op);


    if(op == 1)
    {
    int data;
    printf("Enter data to insert: ");
    scanf("%d",&data);
```

```
InsertLast(&head,data);
}
else if(op == 2){
    int data;
    printf("Enter data to insert: ");
    scanf("%d",&data);
    insertFirst(&head,data);
}
else if(op == 3){
    int data,val_before,val_after;
    printf("Enter data to insert: ");
    scanf("%d",&data);
    printf("Enter the values before the data and after the data: ");
    scanf("%d %d",&val_before,&val_after);
    insertMiddle(&head,data,val_before,val_after);
}

else if(op == 4){
    printlist(head);
}

else if(op == 5){
    reverse(&head);
}
else if(op ==0)
```

```
    {
        printf("Exiting\n");
        break;
    }
    }
}

void InsertLast(Node **head,int data){
    Node *new = (Node *)malloc(sizeof(Node));
    if(new == NULL){
        return;

    }
    else{
        new->data = data;
        new->link = NULL;

    }

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

```
        temp = temp->link;
    }
    temp->link = new;
    }
}

void insertFirst(Node **head,int data)
{
    Node *new = (Node *)malloc(sizeof(Node));
    if(new == NULL){
        return;

    }
    else{
        new->data = data;
        new->link = NULL;

    }

    if(*head == NULL){
        *head = new;
    }
    else{
        Node *temp = *head;
        *head = new;
```

```
        new->link = temp;
    }
}

void printlist(Node *head){

    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    while(head != NULL){
        printf("%d->",head->data);
        head = head->link;
    }
    printf("NULL\n");
}

void insertMiddle(Node **head,int data,int val_before,int val_after)
{
    Node *new = (Node *)malloc(sizeof(Node));
    if(new == NULL){
        return;

    }
```

```
else{
    new->data = data;
    new->link = NULL;

}

if(*head == NULL)
{
    printf("List is empty!\n");
}


    Node *prev = *head;
    while(prev != NULL && prev->link != NULL){
        if(prev->data == val_before  && prev->link->data == val_after){

            new->link = prev->link;
            prev->link = new;
            printf("Node inserted successfully.\n");
            return;

        }

    prev = prev->link;
    }
```

```
}


void reverse(Node **head){
   Node *prev = NULL;
   Node *cur = *head;
   Node *next = NULL;
   while(cur != NULL){
      next = cur->link;
      cur->link = prev;
      prev = cur;
      cur = next;
   }
   *head = prev;
   printf("Reversed successfully!\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

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

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

void InsertLast(Node **head, int data);
void printlist(Node *head);
void findMiddle(Node *head);

int main() {
    Node *head = NULL;

    // Insert data into the linked list
    InsertLast(&head, 10);
```

```
    InsertLast(&head, 20);
    InsertLast(&head, 30);
    InsertLast(&head, 40);
    InsertLast(&head, 50);

    printf("List: ");
    printlist(head);

    // Find and display the middle node
    findMiddle(head);

    return 0;
}

void InsertLast(Node **head, int data) {
    Node *new = (Node *)malloc(sizeof(Node));
    if (new == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    new->data = data;
    new->link = NULL;

    if (*head == NULL) {
        *head = new;
```

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

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

    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->link;
    }
    printf("NULL\n");
}

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

```
        return;
    }

    Node *slow = head;
    Node *fast = head;

    while (fast != NULL && fast->link != NULL) {
        slow = slow->link;   // Move slow one step
        fast = fast->link->link; // Move fast two steps
    }

    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

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

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

void insertLast(Node **head,int data);
void printlist(Node *head);
void createCycle(Node **head,int cycle_index);
void find_cyclic(Node **head);
void remove_cycle(Node **head);

int main(){
   Node *head = NULL;
   int op;
   do{
      printf("\nSingle Linked List operations\n");
      printf("1. Insert at last\n");
      printf("2. Print list\n");
      printf("3. Create cycle (for testing)\n");
      printf("4.Find cyclic linked list\n");
      printf("5. Remove cycle\n");
      printf("6. Exit\n");
      printf("Choose an option: ");
      scanf("%d", &op);

       switch(op)
      {
         case 1:
         {
            // Insert last
            int data;
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insertLast(&head, data);
            printf("Inserted at last successfully!!!\n");

         }
         break;
         case 2:
         {
             // Print list
```

```
            printlist(head);
        }
        break;
        case 3:
        {
            // Create a cycle (for testing)
            int cycle_index;
            printf("Enter the index where you want to create a cycle (1-based): ");
            scanf("%d", &cycle_index);
            createCycle(&head, cycle_index);

        }
        break;
        case 4:
        {
            // Detect cycle
            find_cyclic(&head);

        }
        break;
        case 5:
        {
            // Remove cycle
            remove_cycle(&head);
        }
        break;
        case 6:
        {
            printf("Exiting!!!\n");
        }
        break;
        default:
            printf("Invalid option !! Please try again\n");
      }
   }while(op != 6);
}

void insertLast(Node **head,int data){
   Node *new = (Node *)malloc(sizeof(Node));
   if(new == NULL){
      return;

   }
   else{
      new->data = data;
      new->link = NULL;

   }

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

```
    temp->link = new;
    }
}

void printlist(Node *head){

    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    while(head != NULL){
        printf("%d->",head->data);
        head = head->link;
    }
    printf("NULL\n");
}

void createCycle(Node **head,int cycle_index){
    if (*head == NULL){
        printf("List is empty, cannot create a cycle.\n");
        return;
    }

    Node *temp = *head;
    Node *cycle_start_node = NULL;
    int index = 1;

    while (temp->link != NULL)
    {
        if (index == cycle_index)
        {
            cycle_start_node = temp;
        }
        temp = temp->link;
        index++;
    }

    if (cycle_start_node != NULL)
    {
        temp->link = cycle_start_node;
        printf("Cycle created at index %d\n", cycle_index);
    }
    else
    {
        printf("Invalid index. No cycle created.\n");
    }
}

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

```
    while(fast != NULL && fast->link != NULL){
        fast = fast->link->link;
        slow = slow->link;

        if(fast == slow){
            printf("Cycle detected in the list.\n");
            return;
        }
    }
    printf("No cycle found in the list.\n");
}


void remove_cycle(Node **head)
{
    if (*head == NULL)
    {
        printf("Error: List is empty!\n");
        return;
    }

    Node *fast = *head;
    Node *slow = *head;

    while(fast != NULL && fast->link != NULL)
    {
        fast = fast->link->link;
        slow = slow->link;

        if(slow == fast)
        {

            slow = *head;
            while(slow != fast)
            {
                slow = slow->link;
                fast = fast->link;
            }


            Node *temp = fast;
            while(temp->link != fast)
            {
                temp = temp->link;
            }
            temp->link = NULL;
            printf("Cycle removed successfully.\n");
            return;
        }
    }
    printf("No cycle to remove.\n");
}
```