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 liked for 5 students and print it.

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

typedef struct Node
{
   char name[50];
   int rollno;
   int class;
   char section;
   int marks[3];
   struct Node *next;
} Node;

Node* createnode()
{
   Node* new_node = (Node *)malloc(sizeof(Node));


   printf("Name: ");
   scanf("%s", new_node->name);
   printf("Roll Number: ");
   scanf("%d", &new_node->rollno);
   printf("Class: ");
   scanf("%d", &new_node->class);
```

```
    printf("Section: ");
    scanf(" %c", &new_node->section);
    printf("Enter marks for 3 subjects: ");
    for (int j = 0; j < 3; j++)
    {
        scanf("%d", &new_node->marks[j]);
    }

    new_node->next = NULL;
    return new_node;
}

int main()
{
    Node *head = NULL, *temp = NULL;


    for (int i = 1; i <= 5; i++)
    {
        Node* new_node = createnode();

        if (head == NULL)
        {
            head = new_node;
        }
        else
        {
            temp = head;
```

```
        while (temp->next != NULL)
        {
           temp = temp->next;
        }
        temp->next = new_node;
      }
   }



   printf("\nStudent Details:\n");
   temp = head;
   while (temp != NULL)
   {
      printf("Name: %s\n", temp->name);
      printf("Roll Number: %d\n", temp->rollno);
      printf("Class: %d\n", temp->class);
      printf("Section: %c\n", temp->section);
      printf("Marks: %d, %d, %d\n", temp->marks[0], temp->marks[1], temp->marks[2]);
      printf("\n");
      temp = temp->next;
   }


   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

```
#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("the orginal linked list is\n");
    printList(head);

    reverseList(&head);

    printf("\n");
    printf("the reversed linked list is\n");
    printList(head);
    return 0;
}

void InsertEnd(Node** ptrHead, int nData) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    Node* ptrTail = *ptrHead;

    new_node->data = nData;
    new_node->next = NULL;

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

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

    ptrTail->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;
}



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 printList(Node*);
void findMiddle(Node*);

int main() {
   Node* head = NULL;

   InsertEnd(&head, 10);
   InsertEnd(&head, 20);
   InsertEnd(&head, 30);
   InsertEnd(&head, 40);
   InsertEnd(&head, 50);

   printf("the orginal linked list is\n");
   printList(head);


   printf("\n");

   findMiddle(head);
```

```
return 0;
}

void InsertEnd(Node** ptrHead, int nData) {
   Node* new_node = (Node*)malloc(sizeof(Node));
   Node* ptrTail = *ptrHead;

   new_node->data = nData;
   new_node->next = NULL;

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

   while (ptrTail->next != NULL) {
      ptrTail = ptrTail->next;
   }

   ptrTail->next = new_node;
}




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

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

    printf("The middle element is %d\n", ptr2->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** head, int data);
void printList(Node* head);
int detectAndRemoveCycle(Node* head);
```

```
int main() {
    Node* head = NULL;

    InsertEnd(&head, 10);
    InsertEnd(&head, 20);
    InsertEnd(&head, 30);
    InsertEnd(&head, 40);
    InsertEnd(&head, 50);

    head->next->next->next->next->next = head->next->next;

    if (detectAndRemoveCycle(head)) {
        printf("Cycle detected and removed.\n");
        printf("Updated list: ");
        printList(head);
    } else {
        printf("No cycle detected.\n");
    }

    return 0;
}

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

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

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

int detectAndRemoveCycle(Node* head) {
    Node *slow = head, *fast = head;

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

        if (slow == fast) {
            slow = head;
            Node* prev = NULL;
            while (slow != fast) {
                prev = fast;
                slow = slow->next;
                fast = fast->next;
            }
```

```
            prev->next = NULL;
            return 1;
        }
    }

    return 0;
}

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