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>

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

// Function to create a node
Node* Create_Node() {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    printf("Enter the name: ");
    scanf(" %[^\n]", newNode->name);
    printf("Enter Roll Number: ");
    scanf("%d", &newNode->roll_no);
    printf("Enter class: ");
    scanf("%d", &newNode->class);
    getchar();
    printf("Enter section: ");
```

```
    scanf("%c", &newNode->section);
    printf("Enter marks of 3 subjects: ");
    for (int i = 0; i < 3; i++) {
        scanf("%d, ", &newNode->marks[i]);
    }
    newNode->next = NULL;
    return newNode;
}

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

    // Create and link 5 nodes
    for (int i = 0; i < 5; i++) {
        Node *newNode = Create_Node();
        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = temp->next;
        }
    }

    // Print the student details
    printf("\nSTUDENT DETAILS\n");
    printf("NAME\tROLLNO\tCLASS\tSECTION\tMARKS\n");
    temp = head;
    while (temp != NULL) {
        printf("%s\t%d\t%d\t%c\t", temp->name, temp->roll_no, temp->class, temp->section);
        for (int i = 0; i < 3; i++) {
```

```
            printf("%d", temp->marks[i]);


        }
        printf("\n");
        temp = temp->next;
    }
  return 0;
}
```

Problem : 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:

Define a function to reverse the linked list iteratively.

Update the head pointer to the new first node.

Display the reversed list.

Example Input:

Initial list: 10 -> 20 -> 30 -> 40

Example Output:

Reversed list: 40 -> 30 -> 20 -> 10

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

typedef struct Node {
    int data;
    struct Node* next;
}Node;
Node* createNode(int data);
```

```
Node* reverseLinkedList( Node* head);
void displayList( Node* head);

int main() {

    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);

    printf("Initial list: ");
    displayList(head);



    head = reverseLinkedList(head);

    printf("Reversed list: ");
    displayList(head);

    return 0;
}

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

struct Node* reverseLinkedList(struct 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;
    return head;
}

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

Problem : 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:

Use two pointers: one moving one step and the other moving two steps.

When the faster pointer reaches the end, the slower pointer will point to the middle node.

Example Input:

List: 10 -> 20 -> 30 -> 40 -> 50

Example Output:

Middle node: 30

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

// Define the structure of a node
typedef struct Node {
    int data;
    struct Node *next;
} Node;
Node* Create_Node(int data);
void Find_Middle(Node *head);

int main() {

    Node *head = Create_Node(10);
    head->next = Create_Node(20);
    head->next->next = Create_Node(30);
    head->next->next->next = Create_Node(40);
    head->next->next->next->next = Create_Node(50);

    Find_Middle(head);



    return 0;
}

Node* Create_Node(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
```

```
    if (!newNode) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void Find_Middle(Node *head) {
    Node *slow = head;
    Node *fast = head;


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


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