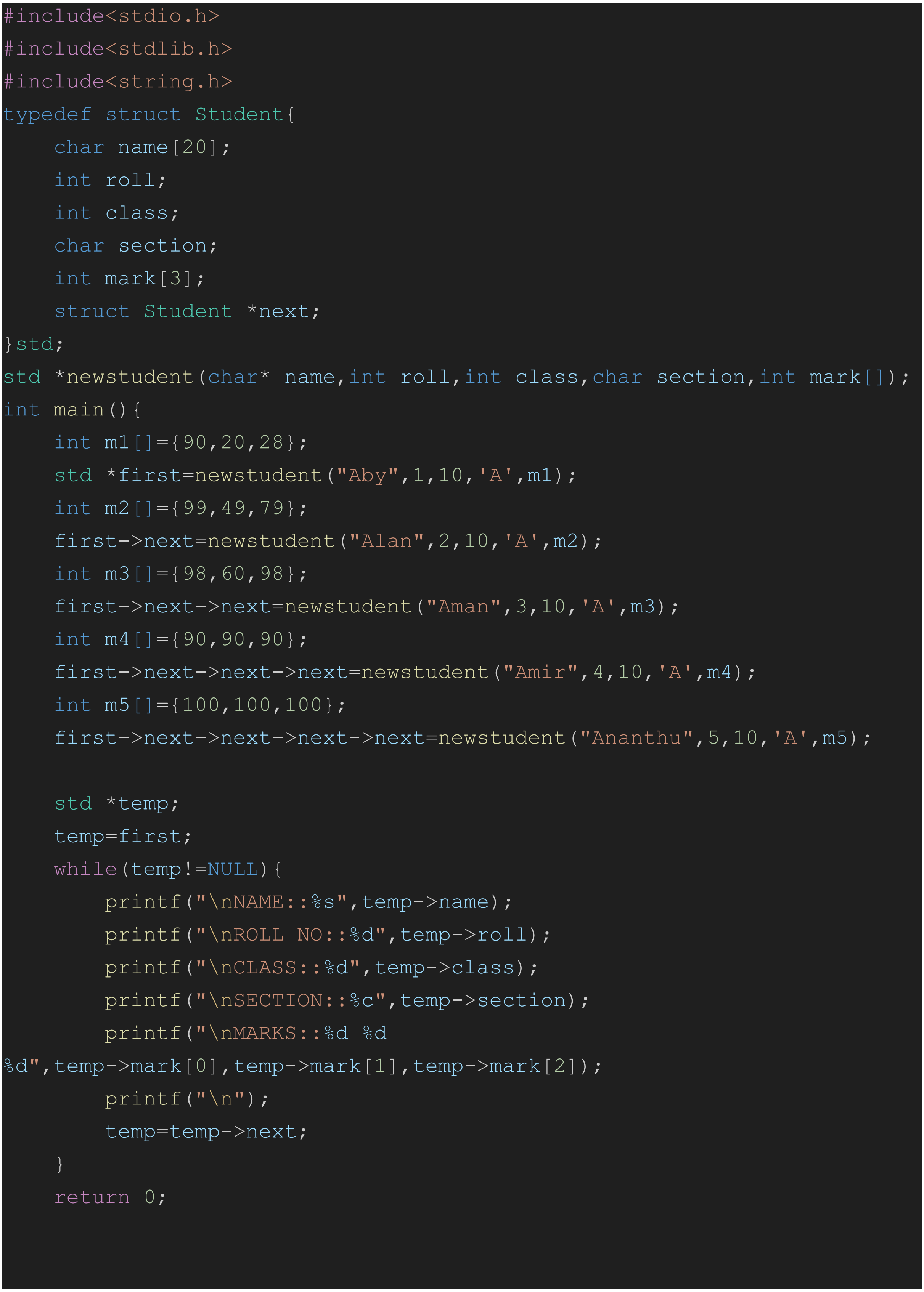
1.create a node in a linked list which will have the following details of student Name, roll number, class, section, an array having marks of any three subjects Create a linked list for 5 students and print it.



```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct Student{
    char name[20];
    int roll;
    int class;
    char section;
    int mark[3];
    struct Student *next;
}std;
std *newstudent(char* name,int roll,int class,char section,int mark[]);
int main(){
    int m1[]={90,20,28};
    std *first=newstudent("Aby",1,10,'A',m1);
    int m2[]={99,49,79};
    first->next=newstudent("Alan",2,10,'A',m2);
    int m3[]={98,60,98};
    first->next->next=newstudent("Aman",3,10,'A',m3);
    int m4[]={90,90,90};
    first->next->next->next=newstudent("Amir",4,10,'A',m4);
    int m5[]={100,100,100};
    first->next->next->next->next=newstudent("Ananthu",5,10,'A',m5);

    std *temp;
    temp=first;
    while(temp!=NULL){
        printf("\nNAME::%s",temp->name);
        printf("\nROLL NO::%d",temp->roll);
        printf("\nCLASS::%d",temp->class);
        printf("\nSECTION::%c",temp->section);
        printf("\nMARKS::%d %d
%d",temp->mark[0],temp->mark[1],temp->mark[2]);
        printf("\n");
        temp=temp->next;
    }
    return 0;
}
std* newstudent(char* name,int roll,int class,char section,int mark[]){
    std *newstd=(std*)malloc(sizeof(std));
    strcpy(newstd->name,name);
    newstd->roll=roll;
    newstd->class=class;
    newstd->section=section;
    for(int i=0;i<3;i++){
        newstd->mark[i]=mark[i];
    }
    newstd->next=NULL;
    return newstd;
}
```

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:

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

Example Output:

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

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:

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

Example Output:

Middle node: 30

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:

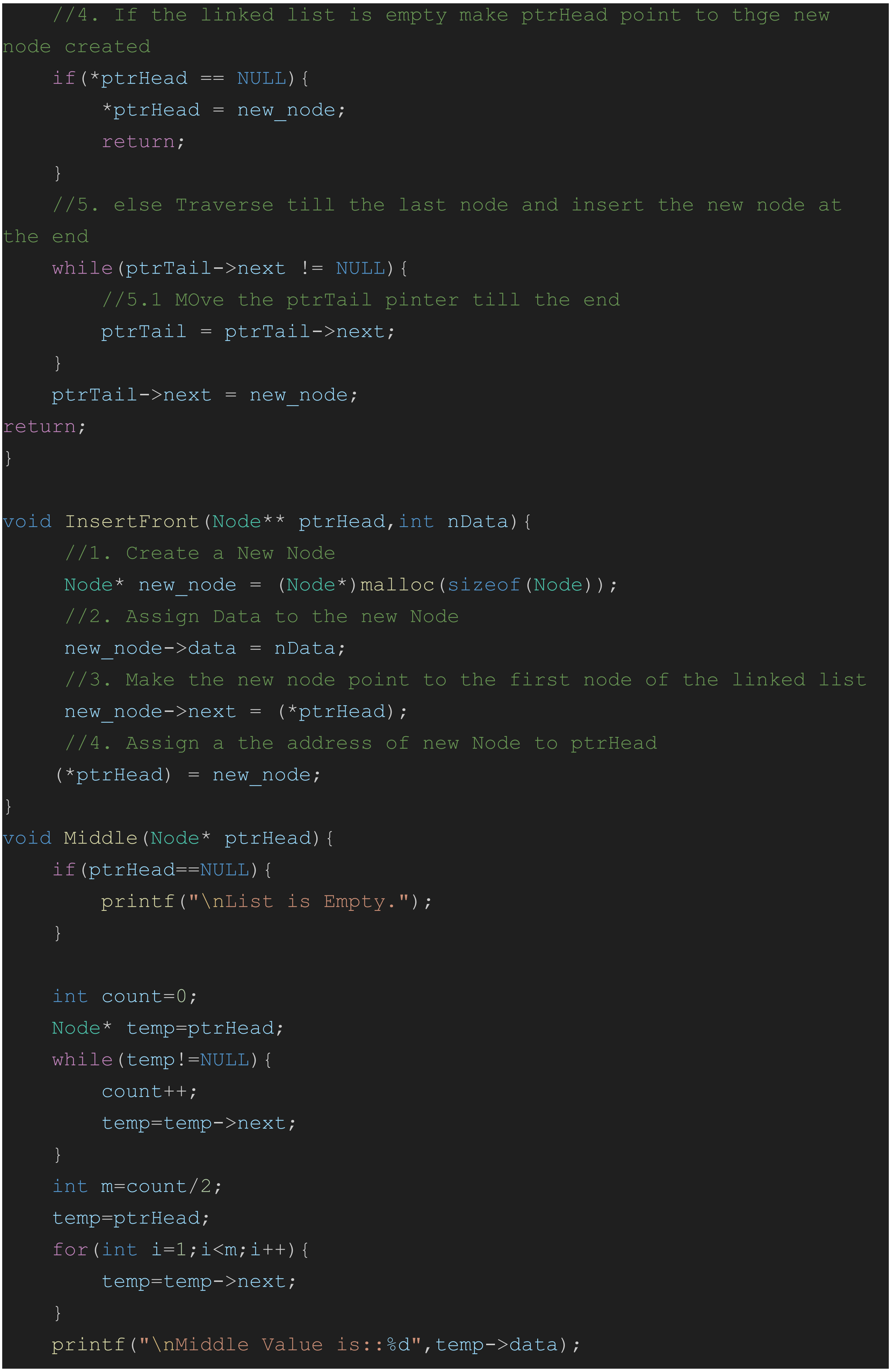
List: 10 -> 20 -> 30 -> 40 -> 50 -> (points back to 30)

Example Output:

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;

//Function with dual purpose: Creating a new node also adding a new
node at the beginning
void InsertFront(Node** ,int );
void Middle(Node*);
//Function with dual purpose: Creating a new node also adding a new
node at the end
void InsertEnd(Node**, int);
void Reverse(Node**);
void printList(Node*);
void RemoveCycle(Node*);

int main(){
    Node* head = NULL;
    InsertEnd(&head, 6);
    InsertEnd(&head, 1);
    InsertEnd(&head, 5);
    InsertFront(&head, 7 );
    InsertFront(&head, 10 );
    printList(head);
    printf("\n");
    printf("\nReversed list::");
    Reverse(&head);
    printList(head);


    Middle(head);
    printf("\n");

    head->next->next->next->next = head->next; // Create a cycle (4 ->
2)

    // Detect cycle
    if (DetectCycle(head)) {
        printf("Cycle detected.\n");
        // Remove cycle
        RemoveCycle(head);
    } else {
        printf("No cycle detected.\n");
    }

    // Print list after removing the cycle
    printf("List after removing cycle: ");
    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 Middle(Node* ptrHead){
    if(ptrHead==NULL){
        printf("\nList is Empty.");
    }

    int count=0;
    Node* temp=ptrHead;
    while(temp!=NULL){
        count++;
        temp=temp->next;
    }
    int m=count/2;
    temp=ptrHead;
    for(int i=1;i<m;i++){
        temp=temp->next;
    }
    printf("\nMiddle Value is::%d",temp->data);
```

```
}
void Reverse(Node** ptrHead) {
    Node* prev = NULL;
    Node* current = *ptrHead;
    Node* next = NULL;

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

    *ptrHead = prev;
}
void RemoveCycle(Node* head) {
    Node* slow = head;
    Node* fast = head;

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

        // Cycle detected
        if (slow == fast) {
            break;
        }
    }

    // If no cycle is detected, return
    if (fast == NULL || fast->next == NULL) {
        return;
    }

    // Find the start of the cycle
    slow = head;
    Node* prev = NULL; // To track the last node in the cycle

    while (slow != fast) {
        prev = fast;   // Track the node before `fast`
        slow = slow->next;
        fast = fast->next;
    }

    // Break the cycle
    prev->next = NULL;
}


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