1.

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



//Node Declaration
typedef struct Node{
    //Data Fields
    int a;
    //Pointer Field (Points to the next node)
    struct Node *next;
}Node;

/*//Representation of linked List Node
struct Node{
    //Data Fields
    int a;
    //Pointer Field (Points to the next node)
    struct Node *next;
};*/


int main(){

    //Creating the first node
    Node *first = (Node*)malloc(sizeof(Node));
    //Assigning the first node
    first->a = 10;
    first->next = NULL;

    //Creating the second  node
    Node *second = (Node*)malloc(sizeof(Node));
    //Assigning the first node
    second->a = 20;


    //Creating the third node
    Node *third = (Node*)malloc(sizeof(Node));
    //Assigning the first node
    third->a = 30;

    //Linking NODE :
    first->next=second;//This creates a link btw the first and the second node
    second->next=third;//Link btw
    third->next=NULL;

  /*
```

```
        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)
    {
        printf("%d -> ",temp->a);
        temp = temp->next;
    }

    printf("NULL\n");

    return 0;


}
```

2.

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

3.

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

// Defining structure of the student node
typedef struct Student {
    char name[50];
    int rollNumber;
    char class[10];
    char section;
    int marks[3];
    struct Student* next;
} Student;

// Function to create a new student node
Student* createNode(char name[], int rollNumber, char class[], char section,
int marks[]) {
    Student* newNode = (Student*)malloc(sizeof(Student));
    strcpy(newNode->name, name);
    newNode->rollNumber = rollNumber;
    strcpy(newNode->class, class);
    newNode->section = section;
    for (int i = 0; i < 3; i++) {
        newNode->marks[i] = marks[i];
    }
    newNode->next = NULL;
    return newNode;
}

// Function to print the linked list
void printList(Student* head) {
    Student* temp = head;
    while (temp != NULL) {
        printf("Name: %s\n", temp->name);
        printf("Roll Number: %d\n", temp->rollNumber);
        printf("Class: %s\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;
    }
}

int main() {
    // Creating nodes
    int marks1[] = {85, 90, 78};
    int marks2[] = {88, 76, 92};
```

```
    int marks3[] = {80, 85, 88};
    int marks4[] = {90, 91, 89};
    int marks5[] = {95, 96, 97};

    Student* head = createNode("Alice", 1, "10", 'A', marks1);
    head->next = createNode("Bob", 2, "10", 'A', marks2);
    head->next->next = createNode("Charlie", 3, "10", 'A', marks3);
    head->next->next->next = createNode("David", 4, "10", 'A', marks4);
    head->next->next->next->next = createNode("Eve", 5, "10", 'A', marks5);

    // Printing the linked list
    printList(head);

    // Freeing memory
    Student* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

4.

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

```
}
void InsertMiddlle(Node* ptrTail, int nData){
     //1. Create a New Node
     Node* new_node = (Node*)malloc(sizeof(Node));


}
```

5.

```
/*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>

// Node structure for linked list
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to reverse the linked list
Node* reverseList(Node* head) {
```

```
    Node* prev = NULL;
    Node* current = head;
    Node* next = NULL;

    while (current != NULL) {
        next = current->next; // Save the next node
        current->next = prev; // Reverse the pointer
        prev = current;       // Move prev one step forward
        current = next;       // Move current one step forward
    }

    return prev; // New head of the reversed list
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" -> ");
        }
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    // Create initial linked list: 10 -> 20 -> 30 -> 40
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);

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

    // Reverse the linked list
    head = reverseList(head);

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

    // Free memory
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
```

```
        free(temp);
    }

    return 0;
}
```

6.

```
/*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>

// Node structure for the linked list
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to find the middle node of the linked list
void findMiddleNode(Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
```

```c
    Node* slow = head;
    Node* fast = head;

    // Move 'fast' two steps and 'slow' one step until 'fast' reaches the end
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // 'slow' now points to the middle node
    printf("Middle node: %d\n", slow->data);
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" -> ");
        }
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    // Create linked list: 10 -> 20 -> 30 -> 40 -> 50
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);
    head->next->next->next->next = createNode(50);

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

    // Find and print the middle node
    findMiddleNode(head);

    // Free memory
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
```

```
    return 0;
}
```

7.

```
/*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>

// Node structure for the linked list
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to detect and remove a cycle in the linked list
void detectAndRemoveCycle(Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
```

```c
    Node* slow = head;
    Node* fast = head;

    // Step 1: Detect if a cycle exists
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) { // Cycle detected
            printf("Cycle detected.\n");
            break;
        }
    }

    // If no cycle is detected
    if (fast == NULL || fast->next == NULL) {
        printf("No cycle detected.\n");
        return;
    }

    // Step 2: Find the start of the cycle
    slow = head;
    Node* prev = NULL; // To keep track of the node before the meeting point
    while (slow != fast) {
        prev = fast;    // Update the previous node
        slow = slow->next;
        fast = fast->next;
    }

    // 'slow' and 'fast' meet at the starting node of the cycle
    printf("Cycle starts at node with data: %d\n", slow->data);

    // Step 3: Break the cycle
    prev->next = NULL;
    printf("Cycle removed.\n");
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" -> ");
        }
        temp = temp->next;
    }
    printf("\n");
```

```
}

int main() {
    // Create linked list: 10 -> 20 -> 30 -> 40 -> 50
    Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    head->next->next->next = createNode(40);
    head->next->next->next->next = createNode(50);

    // Introduce a cycle: Point 50 -> 30
    head->next->next->next->next->next = head->next->next;

    // Detect and remove the cycle
    detectAndRemoveCycle(head);

    // Print the updated list
    printf("Updated list: ");
    printList(head);

    // Free memory
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```