DAY 16

```
/*
1.representation of linked list node in c

struct node{
   //data fiels
   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 fiels
   int a;

   //pointer field(points to the next node)

   struct node *next;
}Node;

Node *node1=(Node*)malloc(sizeof(Node));

4.Assigning values to the memeber elements of the Node

node1->a=10;
node1->next=null;


*/



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

//define the structure of the node1
```

```
typedef struct node{
    int data;
    struct node *next;

}Node;

int main(){

    //creating the first Node
    Node *first=(Node*)malloc(sizeof(Node));

    //assigning the data

    first->data=10;
    //creating second node
     Node *second=(Node*)malloc(sizeof(Node));

    //assigning the data

    second->data=20;

    //creating third node
     Node *third=(Node*)malloc(sizeof(Node));

    //assigning the data

    third->data=30;

    //Linking the Node

    first->next=second;//this create link between first and second

    second->next=third;//second and third

    third->next=NULL;

    //printing the linked std::list

    /*1.traverse from first to third ,
    a.create a temp pointer of type struct Node

    //10->Null
    Node *first=createNode(10);
    //10->20->NULL
    first->next=createNode(20);

    //10->20->30->NULL
```

```
    first->next->next=createNode(30);
    Node *temp;
    temp=first;
    while(temp!=NULL){
        printf("%d->",temp->data);
        temp=temp->next;
b.Make the temp pointer point to first

    c.move the temp pointer from first to third node for
      printing the linked list

*/
    Node *temp;

    temp=first;

    while(temp!=NULL){
        printf("%d->",temp->data);
        temp=temp->next;
    }




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

//define the structure of the node1

typedef struct node{
    int data;

    struct node *next;

}Node;

Node* createNode(int data);//it returns a pointer to the newly created struct node

int main(){
        }
    return 0;
}

Node* createNode(int data){
    Node *newNode=(Node*)malloc(sizeof(Node));
    newNode->data=data;
     //intially assigning the next field of newly created node to null
    newNode->next=NULL;
```

```
    return newNode;
    }
```

```
    return 0;
}
```

3.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 roll_no;
    int class;
    char section;
    int marks[3];
    struct node *next;
} Node;


Node* createNode(char *name, int roll_no, int class, char section, int marks[]);

int main() {
    int n = 5;  // Number of students

    Node *first = NULL;
    Node *temp = NULL;
```

```
    printf("Enter details for %d students:\n", n);
    for (int i = 0; i < n; i++) {
        char name[50];
        int roll_no, class, marks[3];
        char section;

        printf("\nStudent %d:\n", i + 1);
        printf("Name: ");
        scanf(" %[^\n]s", name);  // Read string with spaces
        printf("Roll Number: ");
        scanf("%d", &roll_no);
        printf("Class: ");
        scanf("%d", &class);
        printf("Section: ");
        scanf(" %c", &section);  // Read single character
        printf("Enter marks for 3 subjects: ");
        for (int j = 0; j < 3; j++) {
            scanf("%d", &marks[j]);
        }

        // Create a new node for each student
        Node* newNode = createNode(name, roll_no, class, section, marks);

        // Link the new node to the list
        if (first == NULL) {
            first = newNode;  // Set first node if the list is empty
        } else {
            temp->next = newNode;  // Link the previous node to the new node
        }
        temp = newNode;  // Move temp to the new node
    }

    // Print the student details in a similar format
    temp = first;
    printf("\nStudent Details:\n");
    while (temp != NULL) {
        printf("Name: %s, Roll Number: %d, Class: %d, Section: %c, Marks: %d, %d, %d\n",
               temp->name, temp->roll_no, temp->class, temp->section, temp->marks[0],
temp->marks[1], temp->marks[2]);
        temp = temp->next;
    }

    return 0;
}

// Function to create a new node
Node* createNode(char *name, int roll_no, int class, char section, int marks[]) {
    // Dynamically allocate memory for a new node
```

```
    Node *newNode = (Node*)malloc(sizeof(Node));

    // Copy student details into the new node
    strcpy(newNode->name, name);
    newNode->roll_no = roll_no;
    newNode->class = class;
    newNode->section = section;
    for (int i = 0; i < 3; i++) {
        newNode->marks[i] = marks[i];
    }
    newNode->next = NULL;  // Set the next pointer to NULL
    return newNode;  // Return the newly created node
}
```

1)INSERTION 3 CASES

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

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

}Node;

void insertFront(Node**,int);
void insertMiddle(Node**,int,int);
//function with dual purpose.creating a new node
void insertEnd(Node**,int);
void printList(Node*);

int main(){

    Node *head=NULL;//list is empty
    insertEnd(&head,6);//double pointer(stores address of a pointer)
    insertEnd(&head,7);
    insertFront(&head,8);
     insertMiddle(&head, 15, 2);
    printList(head);
    return 0;
}
// Function to insert a new node at the end of the linked list
void insertEnd(Node** ptrhead, int nData) {
    // 1. Create a new node
    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }

    // 2. Initialize the new node with data and set next to NULL
    newNode->data = nData;
    newNode->next = NULL;

    // 3. Check if the linked list is empty
    if (*ptrhead == NULL) {
        *ptrhead = newNode;  // Make the new node the head
        return;
    }

    // 4. Traverse the list to find the last node
    Node *ptrTail = *ptrhead;
    while (ptrTail->next != NULL) {
        ptrTail = ptrTail->next;
    }

    // 5. Link the new node at the end
    ptrTail->next = newNode;
}

void insertFront(Node**ptrhead,int nData){
    //1.create a new Node
    Node* newNode=(Node*)malloc(sizeof(Node));
    //2.Assign data to the new Node
    newNode->data=nData;
    //3.make the new node point to the first node of the linked list
    newNode->next=(*ptrhead);
    //4.assign the address of new node to ptrHead
    (*ptrhead)=newNode;

}

void insertMiddle(Node** ptrhead, int nData, int position) {
    // 1. Check if the position is valid
    if (position <= 0) {
        printf("Invalid position.\n");
        return;
    }

    // 2. Create a new node
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
```

```
        return;
    }
    newNode->data = nData;

    // 3. If position is 1, insert at the front
    if (position == 1) {
        newNode->next = *ptrhead;
        *ptrhead = newNode;
        return;
    }

    // 4. Traverse the list to find the previous node
    Node* temp = *ptrhead;
    int currentPosition = 1;
    while (temp != NULL && currentPosition < position - 1) {
        temp = temp->next;
        currentPosition++;
    }

    // 5. If we reached the end or position is out of bounds, print a message
    if (temp == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }

    // 6. Insert the new node after the previous node
    newNode->next = temp->next;
    temp->next = newNode;
}


// Function to print the linked list
void printList(Node* node) {
    if (node == NULL) {
        printf("The list is empty.\n");
        return;
    }

    printf("Linked List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
```

/*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:
Define a function to reverse the linked list iteratively.
Update the head pointer to the new first node.
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 insert(Node**,int);
void printList(Node*);
void reverseList(Node**);
int main(){
    Node* head=NULL;//list is empty
    insert(&head,10);
    insert(&head,20);
    insert(&head,30);
    insert(&head,40);
     insert(&head,50);
    printList(head);
    printf("REversed List\n");
    reverseList(&head);
    printList(head);
    return 0;
}

void insert(Node** ptrHead, int nData) {
    // 1. Allocate memory for the new node
    Node* newNode = (Node*)malloc(sizeof(Node));

    // 2. Assign data to the new node
    newNode->data = nData;
```

```
    newNode->next = NULL;

    // 3. Check if the list is empty
    if (*ptrHead == NULL) {
        *ptrHead = newNode;
        return;
    }

    // 4. Traverse to the last node
    Node* ptrTail = *ptrHead;
    while (ptrTail->next != NULL) {
        ptrTail = ptrTail->next;
    }

    // 5. Link the new node to the last node
    ptrTail->next = newNode;
}


void printList(Node* node){
    if(node==NULL){
        printf("List is empty");
    }
    while(node!=NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

// Function to reverse the linked list iteratively
void reverseList(Node** ptrHead) {
    Node* prev = NULL;
    Node* current = *ptrHead;
    Node* next = NULL;

    // Traverse the list and reverse the links
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    // Update the head to point to the new first node (which is prev)
    *ptrHead = prev;
}
```

/*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:
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:
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 insert(Node**,int data);
void printList(Node*);
void middle(Node*);


int main(){
    Node *head=NULL;

    insert(&head,10);
    insert(&head,20);
    insert(&head,30);
    insert(&head,40);
    insert(&head,50);
    insert(&head,60);
    printList(head);
    printf("\n");
    middle(head);

    return 0;
}

void insert(Node **ptrHead,int nData){
```

```
  Node* newnode=(Node*)malloc(sizeof(Node));
   newnode->data=nData;
   newnode->next=NULL;

   if(*ptrHead==NULL){
      *ptrHead=newnode;
      return;
   }
   Node *ptrTail=*ptrHead;
    while(ptrTail->next!=NULL){
        ptrTail=ptrTail->next;
    }
    ptrTail->next=newnode;

}

 void printList(Node *node){
    if(node==NULL){
        printf("List is empty");
    }
    while(node!=NULL){
        printf("%d->",node->data);
        node=node->next;
    }
    }
 void middle(Node *head1){
    Node *slow=head1;
    Node *fast=head1;

    if(head1==NULL){
        printf("List is empty");
    }
    while(fast!=NULL && fast->next!=NULL){
        slow=slow->next;
        fast=fast->next->next;
    }
    printf("The middle node is: %d\n", slow->data);
}
```

3)/*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:
Detect the cycle in the list.
If a cycle exists, find the starting node of the
cycle and break the loop.

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>

// Define the structure for a node
typedef struct node {
    int data;
    struct node* next;
} Node;

// Function prototypes
void insert(Node** head, int data);
void createCycle(Node* head, int position);
void detectAndRemoveCycle(Node* head);
void printList(Node* head);

int main() {
    Node* head = NULL;

    // Create a linked list
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);
    insert(&head, 50);

    // Create a cycle (50 points back to 30)
    createCycle(head, 3);

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

    // Print the updated list
    printList(head);

    return 0;
}
```

```
// Function to insert a node at the end of the list
void insert(Node** head, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

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

// Function to create a cycle in the list
void createCycle(Node* head, int position) {
    Node* temp = head;
    Node* cycleNode = NULL;
    int count = 1;

    while (temp->next != NULL) {
        if (count == position) {
            cycleNode = temp;
        }
        temp = temp->next;
        count++;
    }
    temp->next = cycleNode;  // Create the cycle
}

// Function to detect and remove a cycle
void detectAndRemoveCycle(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) {
            printf("Cycle detected.\n");
```

```
            // Find the start of the cycle
            slow = head;
            while (slow != fast) {
                slow = slow->next;
                fast = fast->next;
            }

            // Break the cycle
            Node* temp = fast;
            while (temp->next != slow) {
                temp = temp->next;
            }
            temp->next = NULL;  // Remove the loop
            printf("Cycle removed.\n");
            return;
        }
    }

    printf("No cycle detected.\n");
}

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