PHASE 1 DAY 22

1.Delete node added

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

typedef struct Node {
    int data;
    struct Node *next;
}Node;

Node *head = NULL;

void display1(Node *);
void display2(Node *);
int nCount(Node *);
int rCount(Node *);
int nSum(Node *);
int rSum(Node *);
int nMax(Node *);
int rMax(Node *);
Node* nSearch(Node *, int);
void insert(Node *, int, int);
void create(int *, int);
int delete(Node*,int);

int main()
{
    int A[] = {20, 30, 40, 60, 70};
    create(A, 5);

    printf("Original list: ");
    display1(head);
    printf("\n");
    printf("Reversed list: ");
    display2(head);
    printf("\nNumber of nodes (iteration): %d\n", nCount(head));
    printf("Number of nodes (recursion): %d\n", rCount(head));
    printf("Sum of elements (iteration): %d\n", nSum(head));
    printf("Sum of elements (recursion): %d\n", rSum(head));
    printf("Max of elements (iteration): %d\n", nMax(head));
    printf("Max of elements (recursion): %d\n", rMax(head));
    Node *key = nSearch(head, 20);
```

```
    printf("Element found: %d\n", key->data);
    insert(head, 0, 10);
    display1(head);
    printf("\nNumber of nodes (iteration): %d\n", nCount(head));
    insert(head, 4, 50);
    display1(head);
    printf("\nNumber of nodes (recursion): %d\n", rCount(head));
    int delvar=delete(head,5);
    printf("node deleted=%d\n",delvar);
    display1(head);
    return 0;
}

//function to display using recursion
void display1(Node *p) {
    if (p != NULL) {
        printf("%d -> ", p->data);
        display1(p->next);
    }
}

//function to display using recursion but in reverse order
void display2(Node *p) {
    if (p != 0) {
        display2(p->next);
        printf("%d <- ", p->data);
    }
}

//function to count the number of nodes in the linked list
int nCount(Node *p) {
    int c = 0;
    while (p) {
        c++;
        p = p->next;
    }
    return c;
}

//function to count using recursion
int rCount(Node *p) {
    if (p == 0) {
        return 0;
    } else {
```

```
        return 1 + rCount(p->next);
    }
}

//function to find sum using iteration
int nSum(Node *p) {
    int sum = 0;
    while (p) {
        sum += p->data;
        p = p->next;
    }
    return sum;
}

//function to find sum using recursion
int rSum(Node *p) {
    int sum = 0;
    if (!p) {
        return 0;
    } else {
        sum += p->data;
        return sum + rSum(p->next);
    }
}

//function to find maximum using iteration
int nMax(Node *p) {
    int max = INT_MIN;
    while(p != NULL) {
        if((p->data) > max) {
            max = p->data;
        }
    p = p->next;
    }
    return max;
}

//function to find max using recursion
int rMax(Node *p) {
    int max = INT_MIN;
    if (p == 0) {
        return INT_MIN;
    }
    else {
```

```
        max = rMax(p->next);
        if(max > p->data)
            return max;
        else
            return p->data;
    }
}

//function to find the element
Node* nSearch(Node *p, int key) {
    while(p != NULL) {
        if(key == p->data)
            return p;
        p = p -> next;
    }
    return NULL;
}

//to insert at a position
void insert(Node *p, int index, int x) {
    Node *t;
    int i;

    if(index < 0 || index > nCount(p)) {
        printf("\nInvalid position!");
    }

    t = (Node*)malloc(sizeof(Node));
    t->data = x;

    if(index == 0) {
        t->next = head;
        head = t;
    } else {
        for(i = 0; i < index-1; i++) {
            p = p->next;
        }
        t->next = p->next;
        p->next = t;
    }
}

//to create a linked list from an array
void create(int A[], int n) {
```

```
    Node *p, *last;
    head = (Node *)malloc(sizeof(Node));

    head->data = A[0];
    head->next = NULL;
    last = head;

    for(int i = 1; i < n; i++) {
        p = (Node *)malloc(sizeof(Node));
        p->data = A[i];
        p->next = NULL;
        last->next = p;
        last = p;
    }
}

int delete(Node *p,int index)
{
    Node* q=NULL;
    int x=-1,i;
    if(index<1 || index>nCount(p))
    {
        return -1;
    }
     if(index==1)
     {
         x=head->data;
         head=head->next;
         free(p);
         return x;
     }
    else{
        p=head;
        for(i=0;i<index-1&&p;i++)
        {
            q=p;
            p=p->next;
        }
        q->next=p->next;
        x=p->data;
        free(p);
        return x;
    }}
```

Checking for loops

```
int isloop(Node* head)
{
   Node *p, *q;
   p = q = head;

   while (q != NULL && q->next != NULL)
   {
      p = p->next;
      q = q->next->next;

      if (p == q)
      {
         return 1;
      }
   }

   return 0;
}
```

create two linked list in one linked {1,2,3,4}and in the 2nd linked list will have value{7,8,9}.COncatenate both the linked list and display the concatenated linked list.

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

typedef struct Node
{
   int data;
   struct Node *next;
}Node;

Node *head = NULL;

void display1(Node *);

 Node* concatenate(Node*,Node*);
 Node* create(int A[], int n) ;


int main()
{
```

```
    int A[] = {1,2,3,4};
    int B[] ={7,8,9};
    Node * head1=create(A,4);
    Node *head2= create(B,3);
    printf("List 1:");
    display1(head1);
    printf("List 2:");
    display1(head2);

    concatenate(head1,head2);
    return 0;

}

void display1(Node *p)
{
    if (p != NULL)
    {
        printf("%d -> ", p->data);
        display1(p->next);
    }
}

Node* create(int A[], int n)
{
    Node *p, *last;
    head = (Node *)malloc(sizeof(Node));

    head->data = A[0];
    head->next = NULL;
    last = head;

    for(int i = 1; i < n; i++) {
        p = (Node *)malloc(sizeof(Node));
        p->data = A[i];
        p->next = NULL;
        last->next = p;
        last = p;
    }
    return head;
}
Node* concatenate(Node*p,Node*q)
{
 Node *t1=p;

    while(p->next!=NULL)
```

```
    {
        p=p->next;

    }
    p->next=q;
    printf("\nAfter merging:");
    display1(t1);
}
```

Problem Statement: Automotive Manufacturing Plant Management System

Objective:

Develop a program to manage an automotive manufacturing plant's operations using a linked list in C programming. The system will allow creation, insertion, deletion, and searching operations for managing assembly lines and their details.

Requirements

Data Representation

Node Structure:

Each node in the linked list represents an assembly line.

Fields:

1.
    - **lineID (integer): Unique identifier for the assembly line.**
    - **lineName (string): Name of the assembly line (e.g., "Chassis Assembly").**
    - **capacity (integer): Maximum production capacity of the line per shift.**
    - **status (string): Current status of the line (e.g., "Active", "Under Maintenance").**
    - **next (pointer to the next node): Link to the next assembly line in the list.**
2. **Linked List:**
    - **The linked list will store a dynamic number of assembly lines, allowing for additions and removals as needed.**

Features to Implement

1. **Creation:**
    - **Initialize the linked list with a specified number of assembly lines.**
2. **Insertion:**

    - **Add a new assembly line to the list either at the beginning, end, or at a specific position.**
3. **Deletion:**
    - **Remove an assembly line from the list by its lineID or position.**
4. **Searching:**
    - **Search for an assembly line by lineID or lineName and display its details.**
5. **Display:**
    - **Display all assembly lines in the list along with their details.**
6. **Update Status:**
    - **Update the status of an assembly line (e.g., from "Active" to "Under Maintenance").**

Example Program Flow

Menu Options:

Provide a menu-driven interface with the following operations:

1.
    - **Create Linked List of Assembly Lines**
    - **Insert New Assembly Line**
    - **Delete Assembly Line**
    - **Search for Assembly Line**
    - **Update Assembly Line Status**
    - **Display All Assembly Lines**
    - **Exit**
2. **Sample Input/Output:**
   Input:
    - **Number of lines: 3**
    - **Line 1: ID = 101, Name = "Chassis Assembly", Capacity = 50, Status = "Active".**
    - **Line 2: ID = 102, Name = "Engine Assembly", Capacity = 40, Status = "Under Maintenance".**

Output:

Assembly Lines:

-
    - **Line 101: Chassis Assembly, Capacity: 50, Status: Active**
    - **Line 102: Engine Assembly, Capacity: 40, Status: Under Maintenance**

Linked List Node Structure in C

#include <stdio.h>

```
#include <stdlib.h>

#include <string.h>


// Structure for a linked list node

typedef struct AssemblyLine {

  int lineID;                  // Unique line ID

  char lineName[50];             // Name of the assembly line

  int capacity;                // Production capacity per shift

  char status[20];               // Current status of the line

  struct AssemblyLine* next;       // Pointer to the next node

} AssemblyLine;
```

Operations Implementation

1. Create Linked List

- **Allocate memory dynamically for AssemblyLine nodes.**
- **Initialize each node with details such as lineID, lineName, capacity, and status.**

2. Insert New Assembly Line

- **Dynamically allocate a new node and insert it at the desired position in the list.**

3. Delete Assembly Line

- **Locate the node to delete by lineID or position and adjust the next pointers of adjacent nodes.**

4. Search for Assembly Line

- **Traverse the list to find a node by its lineID or lineName and display its details.**

5. Update Assembly Line Status

- **Locate the node by lineID and update its status field.**

6. Display All Assembly Lines

- **Traverse the list and print the details of each node.**

Sample Menu

Menu:

1. Create Linked List of Assembly Lines

2. Insert New Assembly Line

3. Delete Assembly Line

4. Search for Assembly Line

5. Update Assembly Line Status

6. Display All Assembly Lines

7. Exit

ANSWER:

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

typedef struct AssemblyLine {
    int lineID;                    // Unique line ID
    char lineName[50];             // Name of the assembly line
    int capacity;                  // Production capacity per shift
    char status[20];               // Current status of the line
    struct AssemblyLine* next;     // Pointer to the next node
} AssemblyLine;

void createList(AssemblyLine** head, int n);
void insertAssemblyLine(AssemblyLine** head, int position);
void deleteAssemblyLine(AssemblyLine** head, int lineID);
void searchAssemblyLine(AssemblyLine* head, int lineID);
void updateStatus(AssemblyLine* head, int lineID, const char* newStatus);
void displayAll(AssemblyLine* head);

int main() {
    AssemblyLine* head = NULL;
    int choice, n, lineID, position;
    char newStatus[20];

    do {
        printf("\nMenu:\n");
        printf("1. Create Linked List of Assembly Lines\n");
```

```
printf("2. Insert New Assembly Line\n");
printf("3. Delete Assembly Line\n");
printf("4. Search for Assembly Line\n");
printf("5. Update Assembly Line Status\n");
printf("6. Display All Assembly Lines\n");
printf("7. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the number of assembly lines: ");
        scanf("%d", &n);
        createList(&head, n);
        break;
    case 2:
        printf("Enter the position to insert the assembly line (1 for start): ");
        scanf("%d", &position);
        insertAssemblyLine(&head, position);
        break;
    case 3:
        printf("Enter the line ID to delete: ");
        scanf("%d", &lineID);
        deleteAssemblyLine(&head, lineID);
        break;
    case 4:
        printf("Enter the line ID to search: ");
        scanf("%d", &lineID);
        searchAssemblyLine(head, lineID);
        break;
    case 5:
        printf("Enter the line ID to update status: ");
        scanf("%d", &lineID);
        printf("Enter the new status: ");
        scanf("%s", newStatus);
        updateStatus(head, lineID, newStatus);
        break;
    case 6:
        displayAll(head);
        break;
    case 7:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Try again.\n");
}
```

```
    } while (choice != 7);

    return 0;
}

void createList(AssemblyLine** head, int n)
{
    AssemblyLine* temp = NULL;
    AssemblyLine* newNode = NULL;
    for (int i = 0; i < n; i++) {
        newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
        printf("Enter details for assembly line %d:\n", i + 1);
        printf("Line ID: ");
        scanf("%d", &newNode->lineID);
        printf("Line Name: ");
        scanf(" %[^\n]", newNode->lineName);
        printf("Capacity: ");
        scanf("%d", &newNode->capacity);
        printf("Status: ");
        scanf("%s", newNode->status);
        newNode->next = NULL;

        if (*head == NULL)
        {
            *head = newNode;
        }
        else
        {
            temp->next = newNode;
        }
        temp = newNode;
    }
}

void insertAssemblyLine(AssemblyLine** head, int position)
{
    AssemblyLine* newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("Enter details for the new assembly line:\n");
    printf("Line ID: ");
    scanf("%d", &newNode->lineID);
    printf("Line Name: ");
    scanf(" %[^\n]", newNode->lineName);
    printf("Capacity: ");
    scanf("%d", &newNode->capacity);
    printf("Status: ");
    scanf("%s", newNode->status);
```

```
    newNode->next = NULL;

    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
    } else {
        AssemblyLine* temp = *head;
        for (int i = 1; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
        if (temp != NULL) {
            newNode->next = temp->next;
            temp->next = newNode;
        } else {
            printf("Invalid position!\n");
            free(newNode);
        }
    }
}

void deleteAssemblyLine(AssemblyLine** head, int lineID)
{
    AssemblyLine *temp = *head, *prev = NULL;

    while (temp != NULL && temp->lineID != lineID)
    {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL)
    {
        printf("Assembly line with ID %d not found.\n", lineID);
        return;
    }

    if (prev == NULL)
    {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
    printf("Assembly line with ID %d deleted successfully.\n", lineID);
}
```

```
void searchAssemblyLine(AssemblyLine* head, int lineID)
{
    AssemblyLine* temp = head;

    while (temp != NULL) {
        if (temp->lineID == lineID) {
            printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
                   temp->lineID, temp->lineName, temp->capacity, temp->status);
            return;
        }
        temp = temp->next;
    }
    printf("Assembly line with ID %d not found.\n", lineID);
}

void updateStatus(AssemblyLine* head, int lineID, const char* newStatus)
{
    AssemblyLine* temp = head;

    while (temp != NULL) {
        if (temp->lineID == lineID)
        {
            strcpy(temp->status, newStatus);
            printf("Status updated successfully.\n");
            return;
        }
        temp = temp->next;
    }
    printf("Assembly line with ID %d not found.\n", lineID);
}

void displayAll(AssemblyLine* head)
{
    AssemblyLine* temp = head;

    if (temp == NULL)
    {
        printf("No assembly lines to display.\n");
        return;
    }

    while (temp != NULL)
    {
        printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
               temp->lineID, temp->lineName, temp->capacity, temp->status);
```

```
        temp = temp->next;
    }
}
```

Stack

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

struct Stack{
    int size;
    int top;
    int *S;
};
//Function Prototypes
void create(struct Stack *);
void push(struct Stack *,int );
void display(struct Stack *);
int pop(struct Stack *);
int peek(struct Stack*,int);

int main()
{
    struct Stack st;
    int elementPopped;
    create(&st);
    push(&st,60);
    push(&st,49);
    push(&st,10);
    push(&st,20);
    push(&st,30);
    push(&st,40);
    display(&st);
    elementPopped = pop(&st);
    printf("The Popped element is : %d \n",elementPopped);
    display(&st);
    int peekedelement=peek(&st,1);
    printf("Peeked element=%d\n",peekedelement);
    return 0;
}

void create(struct Stack *st)
```

```
{
    printf("Enter Size: ");
    scanf("%d",&st->size);
    st->top = -1;
    st->S=(int *)malloc(st->size*sizeof(int));
}

void push(struct Stack *st,int x){
    if(st->top == st->size-1){
        printf("Stack Overflow");
    }else
    {
        st->top++;
        st->S[st->top] = x;
    }
  //  printf("top=%d\n",st->top);
}

void display(struct Stack *st){
    for(int i = st->top;i >= 0;i--){
        printf("%d",st->S[i]);
        printf("\n");
    }
   printf("\n");
}

int pop(struct Stack *st)
{
    int x = -1;
    if(st->top == -1){
        printf("Stack Underflow \n");
    }
    else{
        x = st->S[st->top];
        st->top--;
    }
    return x;
}

int peek(struct Stack *st, int pos)
{
    if (pos <= 0 || pos > st->top + 1)
    {
        printf("Invalid position\n");
        exit(0);
    }
```

```
    return st->S[st->top - pos + 1];
}
```

IMPLEMENT STACK USING LINKED LIST

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

typedef struct StackNode {
    int data;
    struct StackNode *next;
} StackNode;

void push(StackNode **top, int data);
int pop(StackNode **top);
void display(StackNode *top);

int main() {
    StackNode *top = NULL;
    int elementPopped, peekedElement;

    push(&top, 60);
    push(&top, 49);
    push(&top, 10);
    push(&top, 20);
    push(&top, 30);
    push(&top, 40);

    display(top);

    elementPopped = pop(&top);
    printf("The Popped element is: %d\n", elementPopped);

    display(top);

    peekedElement = peek(top, 1);
    printf("Peeked element = %d\n", peekedElement);

    return 0;
}

void push(StackNode **top, int data) {
    StackNode *newNode = (StackNode *)malloc(sizeof(StackNode));
    if (newNode == NULL) {
        printf("Stack Overflow\n");
        return;
```

```
    }
    newNode->data = data;
    newNode->next = *top;
    *top = newNode;
}

int pop(StackNode **top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
    int poppedData = (*top)->data;
    StackNode *temp = *top;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}


void display(StackNode *top) {
    StackNode *current = top;
    while (current != NULL) {
        printf("%d\n", current->data);
        current = current->next;
    }
    printf("\n");
}
```