Delete Operation

```
int DeleteNode(struct Node*p,int index){
   struct Node* q=NULL;
   int x=-1,i;
   if(index<1){
      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;
   }
}
```

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>

struct Node {
   int data;
```

```
    struct Node *next;
};

void display(struct Node* p);
void create(struct Node** p, int* A, int n);
void concatenate(struct Node* first, struct Node* second);

int main() {
    struct Node *first = NULL, *second = NULL;
    int A[] = {1, 2, 3, 4};
    int B[] = {7, 8, 9};

    // Create first linked list
    create(&first, A, 4);
    printf("Linked list 1 = ");
    display(first);

    // Create second linked list
    create(&second, B, 3);
    printf("\nLinked list 2 = ");
    display(second);

    // Concatenate second list to first
    concatenate(first, second);
    printf("\nConcatenated linked list = ");
    display(first);

    return 0;
}

void display(struct Node* p) {
```

```
    if (p == NULL) {
        printf("Empty list\n");
        return;
    }
    while (p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
}

void create(struct Node** p, int* A, int n) {
    struct Node *t, *last;
    *p = (struct Node*)malloc(sizeof(struct Node));
    (*p)->data = A[0];
    (*p)->next = NULL;
    last = *p;

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

void concatenate(struct Node* first, struct Node* second) {
    if (first == NULL) {
        first = second;
        return;
```

```
    }
    while (first->next != NULL) {
        first = first->next;
    }
    first->next = second;
}
```

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

1. **Node Structure:**
   Each node in the linked list represents an assembly line.
   Fields:
   - 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

1. **Menu Options:**
   Provide a menu-driven interface with the following operations:
    - 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

```
#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;
void create(AssemblyLine**,int);
AssemblyLine *createNode(int,char *,int,char *);
void insert(AssemblyLine *,int,char *,int,char *,int);
void delete(AssemblyLine *,int);
void search(AssemblyLine *,int);
void update(AssemblyLine *,int,char *);
void display(AssemblyLine *);
int main(){
   AssemblyLine *head=NULL;
   int lineID,capacity,choice,numLines;
   char lineName[50],status[20];
   while(1){
      printf("Menu: \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",&numLines);
      create(&head,numLines);
      break;
   case 2:
      printf("Enter line ID: ");
      scanf("%d",&lineID);
      printf("Enter line name: ");
      scanf("%s",&lineName);
      printf("Enter capacity: ");
      scanf("%d",&capacity);
      printf("Enter status: ");
      scanf("%s",&status);
      printf("Enter the index: ");
      int index;
      scanf("%d",&index);
      insert(head,lineID,lineName,capacity,status,index);
      break;
   case 3:
      printf("Enter the line ID to delete: ");
```

```
            scanf("%d",&lineID);
            delete(head,lineID);
            break;
        case 4:
            printf("Enter the line ID to search: ");
            scanf("%d",&lineID);
            search(head,lineID);
            break;
        case 5:
            printf("Enter the line ID to update: ");
            scanf("%d",&lineID);
            printf("Enter new status: ");
            scanf("%s",&status);
            update(head,lineID,status);
            break;
        case 6:
            display(head);
            break;
        case 7:
            printf("Exiting...");
            exit(0);
            break;
        default:
            printf("Invalid option");
            break;
      }

   }
}
void create(AssemblyLine** head,int numLines){
   for(int i=0;i<numLines;i++){
```

```
        int lineID, capacity;
        char lineName[50], status[20];
        printf("Enter the line ID: ");
        scanf("%d",&lineID);
        printf("Enter line name: ");
        scanf("%s",&lineName);
        printf("Enter capacity: ");
        scanf("%d",&capacity);
        printf("Enter status: ");
        scanf("%s",&status);
        AssemblyLine *newnode = createNode(lineID, lineName,capacity,status);
        newnode->next = *head;
        *head = newnode;
    }
}
AssemblyLine *createNode(int lineID,char lineName[],int capacity,char status[]){
    AssemblyLine *newnode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    newnode->lineID = lineID;
    strcpy(newnode->lineName,lineName);
    newnode->capacity=capacity;
    strcpy(newnode->status,status);
    newnode->next = NULL;
    return newnode;
}
void insert(AssemblyLine *head,int lineID,char lineName[],int capacity,char status[],int index){
    AssemblyLine *newnode = createNode(lineID, lineName,capacity,status);
    if(index==1){
        newnode->next = head;
        head = newnode;
    }else{
        AssemblyLine *p=head;
```

```
        for(int i=0;i<index-1;i++){
            p=p->next;
        }
        newnode->next = p->next;
        p->next = newnode;
    }
}
void delete(AssemblyLine *head,int lineID){
    AssemblyLine *p=head;
    AssemblyLine *q=NULL;
    if(p!=NULL && p->lineID==lineID){
        head = p->next;
        free(p);
        return;
    }
    while(p!=NULL && p->lineID!=lineID){
        q = p;
        p = p->next;
    }
    if(p==NULL){
        printf("Assembly line with ID %d not found",lineID);
    }
    q->next = p->next;
    free(p);
}
void search(AssemblyLine *head, int lineID) {
    AssemblyLine *p = head;
    while (p != NULL) {
        if (p->lineID == lineID) {
            printf("Assembly line found: ID: %d, Name: %s, Capacity: %d, Status: %s\n",
                p->lineID, p->lineName, p->capacity, p->status);
```

```
            return;
        }
        p = p->next;
    }
    printf("Assembly line with ID %d not found\n", lineID);
}
void update(AssemblyLine *head,int lineID,char newstatus){
    AssemblyLine * p=head;
    while(p!=NULL){
        if(p->lineID==lineID){
            strcpy(p->status,newstatus);
            printf("Status updated successfully");
        }
        p=p->next;
    }
    printf("Assembly line with ID %d not found",lineID);
}
void display(AssemblyLine* head) {
    if (head == NULL) {
        printf("No assembly lines to display.\n");
        return;
    }
    AssemblyLine* p = head;
    while (p != NULL) {
        printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",p->lineID, p->lineName, p->capacity,
p->status);
        p = p->next;
    }
}
```

STACK

```
#include<stdio.h>
#include<stdlib.h>
struct Stack{
   int size;
   int top;
   int *S;
};
void create(struct Stack *);
void push(struct Stack *,int);
void display(struct Stack *);
int pop(struct Stack *);
int peek(struct Stack,int);
int stacktop(struct Stack);
void isEmpty(struct Stack);
void isFull(struct Stack);
int main(){
   struct Stack st;
   create(&st);
   push(&st,10);
   push(&st,20);
   push(&st,30);
   push(&st,40);
   display(&st);
   int elementPopped = pop(&st);
   printf("The popped element is %d\n",elementPopped);
   display(&st);
   printf("Enter the position you want to peek: ");
   int pos;
   scanf("%d",&pos);
   int peekElement = peek(st,pos);
```

```
    printf("The element at the position %d is %d",pos,peekElement);
    int topElement = stacktop(st);
    printf("\nStack top element is %d",topElement);
    isEmpty(st);
    isFull(st);
}
void create(struct Stack *st){
    printf("Enter the 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;
    }
}
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");
```

```
    }else{
        x = st->S[st->top];
        st->top--;
    }
    return x;
}

int peek(struct Stack st,int pos){
    int x=0;
    if(st.top-pos+1 < 0){
        printf("Invalid position");
    }else{
       x = st.S[st.top-pos+1];
    }
    return x;
}
int stacktop(struct Stack st){
    return st.S[st.top];

}
void isEmpty(struct Stack st){
    if(st.top==-1){
        printf("\nStack is empty");
    }else{
        printf("\nStack is not empty");
    }
}
void isFull(struct Stack st){
    if(st.top==st.size-1){
        printf("\nStack is full");
    }else{
```

```
        printf("\nStack is not full");
    }
}
```

Stack using Linked list

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

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

struct Stack {
    struct Node* top;
    int size;
};
void initStack(struct Stack* stack) {
    stack->top = NULL;
    stack->size = 0;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
```

```
    }

    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
    stack->size++;
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow! Cannot pop from an empty stack.\n");
        return -1;
    }

    struct Node* temp = stack->top;
    int poppedValue = temp->data;
    stack->top = stack->top->next;
    free(temp);
    stack->size--;
    return poppedValue;
}


int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Cannot peek.\n");
        return -1;
    }
    return stack->top->data;
}
```

```
int size(struct Stack* stack) {
    return stack->size;
}



int main() {
    struct Stack stack;
    initStack(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("Top of stack: %d\n", peek(&stack));

    printf("Pop: %d\n", pop(&stack));
    printf("Pop: %d\n", pop(&stack));

    printf("Size of stack: %d\n", size(&stack));

    printf("Is stack empty? %s\n", isEmpty(&stack) ? "Yes" : "No");

    return 0;
}
```