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>
typedef struct Node{
   int Data;
   struct Node *next;
}Node;

Node *head1=NULL;
Node *head2=NULL;
void print_list(Node *);
void create_list(Node **ptrHead,int [],int);
void concatenate_list(Node *list1,Node *list2);

int main(){
   int n;
   int A[]={1,2,3,4};
   int B[]={5,6,7};


   printf("\n\nLIST 1:\n");
   create_list(&head1,A,4);
   print_list(head1);

   printf("\n\nLIST 2:\n");
   create_list(&head2,B,3);
   print_list(head2);
```

```
    printf("\n\nCONCATENATED LIST:\n");
    concatenate_list(head1,head2);
    print_list(head1);



}
void print_list(Node*p){
    while (p!=NULL)
    {
        printf("%d->",p->Data);
        p=p->next;
    }printf("NULL");

}
void create_list(Node **ptrHead,int a[],int n){

    Node *last,*newNode;

    *ptrHead=(Node *)malloc(sizeof(Node));
    (*ptrHead)->Data=a[0];
    (*ptrHead)->next=NULL;
    last=*ptrHead;

    for(int i=1;i<n;i++){
        newNode=(Node *)malloc(sizeof(Node));
        newNode->Data=a[i];
        newNode->next=NULL;
        last->next=newNode;
        last=newNode;

    }
```

```
}

void concatenate_list(Node *list1,Node *list2){
   Node *temp=list1;
   while(temp->next!=NULL){
      temp=temp->next;
   }temp->next=list2;

}
```

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:

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.

Linked List:

The linked list will store a dynamic number of assembly lines, allowing for additions and removals as needed.

Features to Implement

Creation:

Initialize the linked list with a specified number of assembly lines.

Insertion:

Add a new assembly line to the list either at the beginning, end, or at a specific position.

Deletion:

Remove an assembly line from the list by its lineID or position.

Searching:

Search for an assembly line by lineID or lineName and display its details.

Display:

Display all assembly lines in the list along with their details.

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:

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

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>

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;

AssemblyLine *head=NULL;

void print_list(AssemblyLine *);
void create_list(AssemblyLine **ptrHead);
int count_list(AssemblyLine *p);
void delete_list(AssemblyLine *p);
void search_list(AssemblyLine *p);
void update_list(AssemblyLine *p);
```

```
int main(){
   while(1){
      int ch;
      printf("Enter Your choice:\n");
      printf("\n1.Insert New Assembly Line\n2.Delete Assembly Line\n3.Search for Assembly Line\n4.Update Assembly Line Status\n5.Display All Assembly Lines\n6.Exit\n");
      scanf("%d",&ch);

      switch (ch)
      {
      case 1:
      create_list(&head);
      printf("\n");
      break;

      case 2:
      delete_list(head);
      printf("\n");
      break;

      case 3:
      search_list(head);
      printf("\n");
      break;

      case 4:
      update_list(head);
      printf("\n");
      break;
```

```
        case 5:
        print_list(head);
        printf("\n");
        break;

        case 6:
        printf("existing\n");
        exit(0);



        default:
        printf("Invalid option:\n");
           break;
        }
    }

}

void print_list(AssemblyLine *p){
    while (p!=NULL)
    {
        printf("%d\t%s\t%d\t%s\n",p->lineID,p->lineName,p->capacity,p->status);
        p=p->next;
    }printf("NULL");

}

void create_list(AssemblyLine **ptrHead){
    AssemblyLine *newNode=(AssemblyLine*)malloc(sizeof(AssemblyLine));
```

```
    printf("Enter id : ");
    scanf("%d",&newNode->lineID);

    printf("Enter Name : ");
    scanf("%s",newNode->lineName);

    printf("Enter Capacity : ");
    scanf("%d",&newNode->capacity);

    printf("Enter status : ");
    scanf("%s",newNode->status);

     newNode->next=NULL;

    AssemblyLine *ptrTail=*ptrHead;
   if(*ptrHead==NULL){
    *ptrHead=newNode;
   }else{
    while (ptrTail->next!=NULL)
    {
        ptrTail=ptrTail->next;
    }
    ptrTail->next=newNode;


   }


}

int count_list(AssemblyLine *p){
```

```
    int count=0;
    while (p!=0)
    {
      count++;
      p=p->next;
    }
    return count;

}

void delete_list(AssemblyLine *p){
    p=head;

    if (p == NULL) {
        printf("The list is empty. Nothing to delete.\n");
        return;
    }

    int index;
    printf("enter the index to delete:\n");
    scanf("%d",&index);

    AssemblyLine *q=NULL;
    if(index<1 || index>count_list(p)){
        printf("index not found");
        return;

    }
    if(index==1){
        head=head->next;
        free(p);
```

```
        return;
    }else{
        for(int i=0;i<index-1;i++){
            q=p;
            p=p->next;
        }
        q->next=p->next;
        free(p);
    }



}

void search_list(AssemblyLine *p){
    int id,found=0;
    printf("enter id to search");
    scanf("%d",&id);

    if (p == NULL) {
        printf("The list is empty.\n");
        return;
    }

    while(p!=NULL){
        if(id==p->lineID){
            printf("%d\t%s\t%d\t%s\n",p->lineID,p->lineName,p->capacity,p->status);
            found=1;
            return;
        }p=p->next;
    }if(!found){
        printf("not found");
```

```
    }

}

void update_list(AssemblyLine *p){

    int nId,nCapacity;
    char nName[50],nStatus[20];
    int id,found=0;
    printf("enter id to search");
    scanf("%d",&id);

    if (p == NULL) {
        printf("The list is empty.\n");
        return;
    }

    while(p!=NULL){
        if(id==p->lineID){
            printf("ID:");
            scanf("%d",&nId);
            printf("NAME:");
            scanf("%s",nName);
            printf("CAPACITY:");
            scanf("%d",&nCapacity);
            printf("STATUS:");
            scanf("%s",nStatus);

            p->lineID=nId;
            strcpy(p->lineName,nName);
            p->capacity=nCapacity;
```

```
            strcpy(p->status,nStatus);
            return;

        }p=p->next;
    }if(!found){
        printf("not found");
    }



}
```

STACK IMPLIMENTATION USING ARRAY

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

typedef struct Stack{
    int size;
    int top;
    int *S; //pointer pointing to integer array
}Stack;

void CREATE(Stack *);
void PUSH(Stack *,int);
void DISPLAY(Stack *st);
void POP(Stack *);
void PEEK(Stack *st);

int main(){
```

```
    Stack st;
    CREATE(&st);
    PUSH(&st,10);
    PUSH(&st,20);
    PUSH(&st,30);
    PUSH(&st,40);
    DISPLAY(&st);

    // POP(&st);
    // DISPLAY(&st);

    // POP(&st);
    // DISPLAY(&st);

    // POP(&st);
    // DISPLAY(&st);

    // POP(&st);
    // DISPLAY(&st);

    // POP(&st);
    // DISPLAY(&st);

    PEEK(&st);
    PEEK(&st);
    PEEK(&st);

    return 0;

}
```

```
void CREATE(Stack *st){
   printf("Enter the size");
   scanf("%d",&st->size);

   st->top=-1;

   st->S=(int *)malloc(st->size*sizeof(int));


}

void PUSH(Stack *st,int data){
   if(st->top==st->size-1){
      printf("\nOVERFLOW\n");
   }else{
      st->top++;
      st->S[st->top]=data;
   }

}

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

}

void POP(Stack *st){
   int x=0;
```

```
    if(st->top==-1){
        printf("\nSTACK UNDERFLOW\n");
    }else{
        x=st->S[st->top];
        st->top--;

    }
    printf("deleted element = %d\n",x);
}

void PEEK(Stack *st){
    int position;
    printf("\nenter the position\n");
    scanf("%d",&position);
    int x=0;
    if(st->top-position+1 <0){
        printf("position not available");

    }else{
        x=st->top-position+1;
        printf("index value of position %d = %d and value= %d",position,x,st->S[x]);
    }

//     int index,found=0;
//     printf("enter the index to check");
//     scanf("%d",&index);

//     for(int i=st->top;i>=0;i--){
//         if(index==i){
//             printf("\nvalue at index %d = %d\n",index,st->S[i]);
//             found=1;
```

```
//        }
//    }
//    if(!found){
//        printf("index is not present\n");
//    }
// }
```

STACK USING LINKED LIST

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

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

Node *top=NULL;

void DISPLAY();
void PUSH(int data);
void POP();

int main(){
   PUSH(10);
   PUSH(20);
   PUSH(30);
   PUSH(40);
   PUSH(50);

   POP();
```

```
    POP();

    DISPLAY();

}

void DISPLAY(){
    if(top==NULL){
        printf("\nstack is empty\n");
        return;
    }
    Node  *temp=top;
    while(top!=NULL){
        printf("%d\n",temp->data);
        temp=temp->next;
    }

    }

void PUSH(int data){

        Node *newNode=(Node*)malloc(sizeof(Node));
        newNode->data=data;
        newNode->next=top;
        top=newNode;

}

void POP(){
    if(top==NULL){
        printf("STACK UNDERFLOW\n");
```

```
        return;
    }
    Node *temp=top;
    printf("popped element = %d\n",temp->data);
    top=top->next;
    free(temp);
}
```