/*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;


void insertLast(Node **head,int data);
Node *createNode(Node *newNode,int data);
void concatenate(Node **head1,Node *head2);
void display(Node *head);

int main() {
   Node *head1 = NULL;
   insertLast(&head1, 1);
   insertLast(&head1, 2);
   insertLast(&head1, 3);
   insertLast(&head1, 4);
   printf("First linked list is:\n");
   display(head1);

   Node *head2 = NULL;
   insertLast(&head2, 7);
   insertLast(&head2, 8);
   insertLast(&head2, 9);
   printf("Second linked list is:\n");
   display(head2);

   concatenate(&head1, head2);
   printf("The concatenated linked list is:\n");
   display(head1);

   return 0;
}
void insertLast(Node **head,int data){

   Node *newNode = createNode(newNode,data);
   if(newNode == NULL){
      printf("Memory allocation failed!\n");
      return;
   }

   if(*head == NULL){
      *head = newNode;
      //printf("Successfully inserted at last\n");
      return;
   }
   Node *temp = *head;
   while(temp->next != NULL){
      temp = temp->next;
   }
   temp->next = newNode;
   //printf("Successfully inserted at last\n");
```

```
   return;


}

Node *createNode(Node *newNode,int data)
{
   newNode = (Node *)malloc(sizeof(Node));
   if(newNode == NULL){
      return NULL;
   }
   newNode->data = data;
   newNode->next = NULL;

   return newNode;
}

void concatenate(Node **head1,Node *head2){
   Node *temp1 = *head1;

   while(temp1->next != NULL && head2 != NULL){
      temp1 = temp1->next;
   }
   temp1->next = head2;
   return;
}

void display(Node *head){
   Node *temp = head;
   while(temp){
      printf("%d->",temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
```

Q:Linked list operations

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

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


void insertLast(Node **head,int data);
Node *createNode(Node *newNode,int data);
void display(Node *head);
void insertFirst(Node **head,int data);
void insertPos(Node **head,int data,int pos);
void deleteLast(Node **head);
void deleteFirst(Node **head);
void deletePos(Node **head,int pos);
int nodeCount(Node *p);
void createLoop(Node **p,int pos);
```

```
void findLoop(Node **head);

int main(){
   int op;
   Node *head = NULL;
   do{
      printf("\nSelect the option:\n1.Insert last\n2.Insert first\n3.Insert at a position\n4.Delete first
node\n5.Delete last node\n6.Delete node at a position\n7.Display\n8.To create a loop at a
position\n9.To find the loop present or not\n0.Exit\n");
      scanf("%d",&op);

      switch(op){
         case 1:
            {
               int data;
               printf("Enter the data to be entered:");
               scanf("%d",&data);
               insertLast(&head,data);
               printf("Succesfully inserted at last\n");
               break;
            }
         case 2:
            {
               int data;
               printf("Enter the data to be entered:");
               scanf("%d",&data);
               insertFirst(&head,data);
               printf("Succesfully inserted at first\n");
               break;
            }
         case 3:
            {
               int data;
               printf("Enter the data to be entered:");
               scanf("%d",&data);
               int pos;
               printf("\nEnter the position where the data to be added:");
               scanf("%d",&pos);
               insertPos(&head,data,pos);
               break;
            }

         case 4:
            {
               deleteFirst(&head);
               break;
            }

         case 5:
            {
               deleteLast(&head);
               break;
            }
         case 6:
            {
               int pos;
               printf("Enter the position at which the node have to be deleted:");
```

```
            scanf("%d",&pos);
            if(pos <= 0 || pos > nodeCount(head)){

               printf("Position is unavailable\n");
               return 0;
            }

            deletePos(&head,pos);
            break;
         }

      case 7:
         {
            display(head);
            break;
         }

      case 8:
         {
            int pos;
            printf("Enter position where loop have to be created:");
            scanf("%d",&pos);
            createLoop(&head,pos);
            break;
         }
      case 9:
         {
            findLoop(&head);
            break;
         }
      case 0:
         {
            printf("Exiting...\n");
            break;
         }
    }
  }while(op != 0);
}

void insertLast(Node **head,int data){

  Node *newNode = createNode(newNode,data);
  if(newNode == NULL){
    printf("Memory allocation failed!\n");
    return;
  }

  if(*head == NULL){
    *head = newNode;
    //printf("Successfully inserted at last\n");
    return;
  }
  Node *temp = *head;
  while(temp->next != NULL){
    temp = temp->next;
  }
  temp->next = newNode;
```

```
    //printf("Successfully inserted at last\n");
    return;


}

void insertFirst(Node **head,int data){

    Node *newNode = createNode(newNode,data);
    if(newNode == NULL)
    {
        printf("Memory allocation Failed\n");
        return;
    }
    if(*head == NULL){
        *head = newNode;
        //printf("Succesfully inserted at first\n");
        return;
    }
    Node *temp = *head;
    *head = newNode;
    newNode->next = temp;
    //printf("Succesfully inserted at first\n");
    return;
}

void insertPos(Node **head,int data,int pos){
    if(pos == 1){
        insertFirst(head,data);
        printf("Inserted at position %d\n",pos);
        return;
    }

    Node *newNode = createNode(newNode,data);
    if(newNode == NULL){
        printf("Memory allocation failed\n");
        return;
    }
    Node *temp = *head;
    Node *prev = NULL;
    int count=0;
    while(temp){
        count++;
        if(count == pos){
            newNode->next = temp;
            prev->next = newNode;
            printf("Successfully inserted at position %d",pos);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    if(pos == count+1){
        insertLast(head,data);
        printf("Succesfully inserted at position %d\n",pos);
        return;
    }
```

```
    else{
        printf("Invalid position! Try again\n");
    }
}

void deleteLast(Node **head){
    if(*head == NULL)
    {
        printf("List is empty\n");

    }
    Node *temp = *head;
    Node *prev = NULL;
    while(temp->next != NULL)
    {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
    printf("Succesfully deleted last node\n");
    return;
}

void deleteFirst(Node **head){
    if(*head == NULL){
        printf("List is empty\n");
        return;
    }
    Node *temp = *head;
    *head = temp->next;
    free(temp);
    printf("Succesfully deleted first node\n");
    return;
}

void deletePos(Node **head,int pos){
    if(*head == NULL){
        printf("List is empty\n");
        return;
    }
    int count=0;
    Node *temp = *head;
    Node *prev = NULL;
    while(temp){
        count++;
        if(pos == 1){
            *head = temp->next;
            free(temp);
            printf("Deleted node at %d position\n",pos);
            return;
        }
        else if(count == pos){
            prev->next = temp->next;
            free(temp);
            printf("Deleted the node at %d position\n",pos);
            return;
```

```
        }
        prev = temp;
        temp = temp->next;
    }
}
Node *createNode(Node *newNode,int data)
{
    newNode = (Node *)malloc(sizeof(Node));
    if(newNode == NULL){
        return NULL;
    }
    newNode->data = data;
    newNode->next = NULL;

    return newNode;
}


void display(Node *head){
    Node *temp = head;
    while(temp){
        printf("%d->",temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


//To calculate total number of nodes
int nodeCount(Node *p){
    int count =0;
    while(p != NULL){
        count++;
        p=p->next;
    }
    return count;

}

void createLoop(Node **head,int pos){
    Node *temp = *head;
    Node *startNode = NULL;
    int count = 1;
    while(temp->next != NULL){
        if(count == pos){
            startNode = temp;
        }
        temp = temp->next;
        count++;
    }
    if(startNode != NULL){
        temp->next = startNode;
        printf("Successfully created loop at %d position\n",pos);
        return;
    }
    else{
        printf("Cannot find position %d\n",pos);
```

```
      return;

   }
}

void findLoop(Node **head){
   Node *slow = *head;
   Node *fast = *head;
   int pos=0;
   while(fast != NULL && fast->next != NULL){

      fast = fast->next->next;
      slow = slow->next;
      if(slow == fast){
         slow = *head;
         pos = 1;

         while(slow != fast){
            slow = slow->next;
            fast = fast->next;
            pos++;
         }
         printf("Cycle starts at position %d\n",pos);
         return;
      }

   }
   printf("No loop found.\n");
   return;

}
```

/

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>

// 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 createLink(AssemblyLine **head,int n);
AssemblyLine *createNode(AssemblyLine *newNode);
void display(AssemblyLine *head);
void insertLast(AssemblyLine **head);
void deletePos(AssemblyLine **head,int );
AssemblyLine *searchLine(AssemblyLine *head,int ID);
void update(AssemblyLine **head,int ID);

int main(){
   int op;
   AssemblyLine *head = NULL;
   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");
      scanf("%d",&op);

      switch(op){
         case 1:
            {
               int n;
               printf("Enter number os assembly lines to be added:");
               scanf("%d",&n);
               createLink(&head,n);
```

```
      printf("Successfully created assembly line\n");
      break;
   }
case 2:
   {
      insertLast(&head);
      printf("Inserted succesfully\n");
      break;
   }
case 3:
   {
      int ID;
      printf("Enter the ID of assembly line to be deleted:");
      scanf("%d",&ID);
      deletePos(&head,ID);
      break;
   }

case 4:
   {
      int ID;
      printf("Enter the ID of assembly line to be searched:");
      scanf("%d",&ID);
      AssemblyLine *Line = searchLine(head,ID);
      if(Line == NULL){
         printf("Assembly line od ID %d cannot found\n",ID);

      }else{
         printf("Line %d: ",Line->lineID);
         printf("%s, ",Line->lineName);
         printf("Capacity: %d,",Line->capacity);
         printf(" Status: %s",Line->status);
      }
      break;
   }

case 5:
   {
      int ID;
      printf("Enter the ID of assembly line to be deleted:");
      scanf("%d",&ID);

      update(&head,ID);
      display(head);
      break;
   }

case 6:
   {
      display(head);
      break;
   }



case 7:
   {
```

```
                printf("Exiting...\n");
                break;
            }
        }
    }while(op != 7);
}

void createLink(AssemblyLine **head,int n){
    for(int i=1;i<=n;i++){
        AssemblyLine *newNode = createNode(newNode);
        if(newNode == NULL){
            printf("Memory allocation failed\n");
            return;
        }

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

AssemblyLine *createNode(AssemblyLine *newNode){
    newNode = (AssemblyLine *)malloc(sizeof(AssemblyLine));
    if(newNode == NULL){
        return NULL;
    }
    printf("Enter the line ID: ");
    scanf(" %d",&newNode->lineID);

    printf("Enter Name of the assembly line:");
    scanf("  %[^\n]",newNode->lineName);

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

    printf("enter Current status of the line:");
    scanf(" %[^\n]",newNode->status);

    newNode->next = NULL;
    return newNode;


}

void insertLast(AssemblyLine **head){

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

```
    if(*head == NULL){
        *head = newNode;
        //printf("Successfully inserted at last\n");
        return;
    }
    AssemblyLine *temp = *head;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = newNode;
    //printf("Successfully inserted at last\n");
    return;


}

void deletePos(AssemblyLine **head,int ID){
    if(*head == NULL){
        printf("List is empty\n");
        return;
    }

    AssemblyLine *temp = *head;
    AssemblyLine *prev = NULL;
    while(temp){
        if(temp->lineID == ID){
            if(temp == *head){
                *head = temp->next;
                free(temp);
            }else{
                prev->next = temp->next;
                 free(temp);
            }
            printf("Deleted assembly line of ID %d\n",ID);
            return;
        }

        prev = temp;
        temp = temp->next;
    }
}

AssemblyLine *searchLine(AssemblyLine *head,int ID){
     if(head == NULL){
        printf("List is empty\n");
        return NULL;
    }
    AssemblyLine *temp = head;
    while(temp){
        if(temp->lineID == ID){
            printf("Assembly line found\n");
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
```

```
}


void update(AssemblyLine **head,int ID){
   if(head == NULL){
      printf("List is empty\n");
      return;
   }
   AssemblyLine *temp = *head;
   while(temp){
      if(temp->lineID == ID){
         char stat[20];
         printf("Enter the status to be updated:");
         scanf("  %[^\n]",stat);
         strcpy(temp->status,stat);
         return;
      }
      temp = temp->next;
   }
   printf("Assembly line of ID %d cannot found\n",ID);

}
void display(AssemblyLine *head){
   AssemblyLine *temp = head;
   while(temp){

      printf("Line %d: ",temp->lineID);
      printf("%s, ",temp->lineName);
      printf("Capacity: %d,",temp->capacity);
      printf(" Status: %s",temp->status);
      printf("\n");
      temp = temp->next;
   }

}
```