1. 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 *link;
} s_ll;



void insert_last(s_ll **, int);
void concat(s_ll **, s_ll **);
void print_list(s_ll *);
s_ll* create_node(int );

int main()
{
  int op = 0, data;
  s_ll *head1 = NULL, *head2 = NULL;

  do{
     printf("1.Insert at list 1\n");
     printf("2.Insert at list 2\n");
     printf("3.Concatenate lists\n");
     printf("4.Print list 1\n");
     printf("5.Print list 2\n");
     printf("6. Exit\n");
     printf("Enter your choice: ");
     scanf(" %d", &op);
```

```
switch(op)
{
   case 1:
   {
       printf("Enter the data to be inserted: ");
       scanf("%d", &data);
       insert_last(&head1, data);
       printf("Inserted at last successfully!!!\n");
   }
   break;
   case 2:
   {
       printf("Enter the data to be inserted: ");
       scanf("%d", &data);
       insert_last(&head2, data);
       printf("Inserted at last successfully!!!\n");
   }
   break;
   case 3:
   {
      concat(&head1, &head2);
      printf("Concatenated the lists successfully!!\n");
   }
   break;
   case 4:
   {
      print_list(head1);
   }
   break;
```

```
            case 5:
            {
                print_list(head2);
            }
            break;
            case 6:
            {
                printf("Exiting!!!\n");
            }
        }


    }while(op != 6);
}

void insert_last(s_ll **head, int data)
{
    s_ll *new = create_node(data);
    s_ll *temp = *head;

    if (*head == NULL) {
        *head = new;
    } else {
        while(temp->link != NULL)
        {
            temp = temp->link;
        }
        temp->link = new;
    }
}
```

```
void print_list(s_ll *head)
{
   if(head == NULL)
   {
      printf("List is empty!!\n");
      return;
   }

   s_ll *temp = head;
   while(temp != NULL)
   {
      printf("%d -> ", temp->data);
      temp = temp->link;
   }
   printf("NULL\n");
}

void concat(s_ll **head1, s_ll **head2)
{
   s_ll *temp = *head1;
   while(temp->link != NULL)
   {
      temp = temp->link;
   }
   temp->link = *head2;
}
s_ll* create_node(int data)
{
   s_ll* new = (s_ll*)malloc(sizeof(s_ll));
   new->data = data;
   new->link = NULL;
```

```
    return new;
}
```

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>


typedef struct AssemblyLine
{
    int lineID;
    char lineName[50];
    int capacity;
    char status[20];
    struct AssemblyLine* next;
} AssemblyLine;


void createLinkedList(AssemblyLine **, int n);
void insertAssemblyLine(AssemblyLine** head, int id, char* name, int capacity, char* status);
void deleteAssemblyLine(AssemblyLine** head, int id);
AssemblyLine* searchAssemblyLine(AssemblyLine* head, int id);
void updateAssemblyLineStatus(AssemblyLine* head, int id, char* status);
void displayAssemblyLines(AssemblyLine* head);

int main()
{
    AssemblyLine* head = NULL;
    int choice, n, id, capacity;
    char name[50], status[20];

    while (1) {
        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);
      createLinkedList(&head, n);
      break;
   case 2:
      printf("Enter details for new assembly line: \n");
      printf("ID: ");
      scanf("%d", &id);
      printf("Name: ");
      scanf(" %[^\n]", name);
      printf("Capacity: ");
      scanf(" %d", &capacity);
      printf("Status: ");
      scanf(" %[^\n]", status);
      insertAssemblyLine(&head, id, name, capacity, status);
      break;
   case 3:
      printf("Enter the ID of the assembly line to delete: ");
```

```
            scanf("%d", &id);
            deleteAssemblyLine(&head, id);
            break;
        case 4:
            printf("Enter the ID of the assembly line to search: ");
            scanf("%d", &id);
            AssemblyLine* result = searchAssemblyLine(head, id);
            if (result) {
                printf("Found: ID = %d, Name = %s, Capacity = %d, Status = %s\n", result->lineID, result->lineName, result->capacity, result->status);
            } else {
                printf("Assembly line not found.\n");
            }
            break;
        case 5:
            printf("Enter the ID and new status: ");
            scanf("%d %s", &id, status);
            updateAssemblyLineStatus(head, id, status);
            break;
        case 6:
            displayAssemblyLines(head);
            break;
        case 7:
            exit(0);
        default:
            printf("Invalid choice!\n");
    }
  }

  return 0;
}
```

```
void createLinkedList(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 line %d:\n ", i + 1);
      printf("ID: ");
      scanf("%d", &newNode->lineID);
      printf("Name: ");
      scanf(" %[^\n]", newNode->lineName);
      printf("Capacity: ");
      scanf(" %d", &newNode->capacity);
      printf("Status: ");
      scanf(" %[^\n]", newNode->status);
      newNode->next = NULL;

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

```
void insertAssemblyLine(AssemblyLine** head, int id, char* name, int capacity, char* status)
{
    AssemblyLine* newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    newNode->lineID = id;
    strcpy(newNode->lineName, name);
    newNode->capacity = capacity;
    strcpy(newNode->status, status);
    newNode->next = NULL;

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

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

    temp->next = newNode;
}


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

    if (*head != NULL && (*head)->lineID == id)
```

```
    {
        *head = (*head)->next;
        free(temp);
        printf("Assembly line deleted successfully.\n");
    }

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

    if (temp == NULL)
    {
        printf("Assembly line not found.\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Assembly line deleted successfully.\n");
    return;
}

AssemblyLine* searchAssemblyLine(AssemblyLine* head, int id)
{
    AssemblyLine* temp = head;
    while (temp != NULL)
    {
        if (temp->lineID == id)
        {
```

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

void updateAssemblyLineStatus(AssemblyLine* head, int id, char* status)
{
    AssemblyLine* temp = searchAssemblyLine(head, id);
    if (temp)
    {
        strcpy(temp->status, status);
        printf("Status updated successfully.\n");
    }
    else
    {
        printf("Assembly line not found.\n");
    }
}

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

    printf("Assembly Lines:\n");
    while (temp != NULL) {
```

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

3.Stack using array

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

#define SUCCESS 0
#define FAILURE -1

typedef struct stack
{
    int capacity;
    int top;
    int *stack;
}Stack_t;

int create_stack(Stack_t *, int );
int Push(Stack_t *, int);
int Pop(Stack_t *);
int Peek(Stack_t *);
void Peep(Stack_t);
int Peekindex(Stack_t stk, int index);

int main()
{
        int choice, element, peek, size, index;
        Stack_t stk;
```

```
printf("Enter the size of the stack: ");
scanf("%d", &size);

create_stack(&stk, size);

        while (1)
        {
            printf("\n1. Push\n2. Pop\n3. Display Stack\n4. Peek(Element at Top)\n5. Peek(Element by index)\n6. Exit\nEnter your choice : ");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1:
                                printf("Enter the element to be pushed in stack : ");
                                scanf("%d", &element);
                                if (Push(&stk, element) == FAILURE)
                                {
                                        printf("INFO : Stack Full\n");
                                }
                                break;
                        case 2:
                                if (Pop(&stk) == FAILURE)
                                {
                                        printf("INFO : Stack is empty\n");
                                }
                                else
                                {
                                   printf("INFO : Pop operation is successfull\n");
                                }
```

```
                        break;
                case 3:
                        Peep(stk);
                        break;
                case 4:
                        if ((peek = Peek(&stk)) == FAILURE)
                        {
                                printf("INFO : Stack is empty\n");
                        }
                        else
                        {
                                printf("INFO : Peek element is %d\n", peek);
                        }
                        break;
              case 5:
                 printf("Enter the index: ");
                 scanf("%d", &index);
                 if(Peekindex(stk, index) != FAILURE)
                 printf("The elemenet at %d is : %d", index,Peekindex(stk, index));
                        break;
              case 6:
                         return SUCCESS;
              default:
                         printf("Invalid Choice.\n");
                        break;
                }
        }
        return 0;
}

int create_stack(Stack_t *stk, int size)
```

```
{
   stk->stack = (int * )malloc(size * sizeof(int));
   stk->top = -1;
   stk->capacity = size;
}

int Push(Stack_t *stk, int element)
{
   if(stk->top == stk->capacity - 1)
   return FAILURE;

   stk->top++;
   stk->stack[stk->top] = element;
}

int Pop(Stack_t *stk)
{
   if(stk->top == -1)
   return FAILURE;

   stk->top--;
}

int Peek(Stack_t *stk)
{
   if(stk->top == -1)
   return FAILURE;

   return stk->stack[stk->top];
}
```

```
int Peekindex(Stack_t stk, int index)
{
   if (stk.top == -1 || index < 0 || index > stk.top)
   {
      printf("Invalid potion!!\n");
      return FAILURE;
   }

   return stk.stack[index];
}



void Peep(Stack_t stk)
{
   if(stk.top == -1)
   {
   printf("Stack is empty!!\n");
   return;
   }
   else
   {
            printf("Top -> ");
            while (stk.top > -1)
            {
                  printf("%d ", stk.stack[(stk.top)--]);
            }
            printf("\n");
   }
}
```