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.

ANSWER:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct Node {
   int data;
   struct Node *next;
}Node;
Node *head1 = NULL, *head2=NULL;
void create(Node**,int [],int );
void display(Node *);
void concatenate(Node *, Node *);
int main()
{
   int list1[]={1,2,3,4};
   int list2[]={7,8,9};
   create(&head1,list1,4);
   create(&head2,list2,3);
   concatenate(head1,head2);
   printf("Concatenated list=");
   display(head1);return 0;
}
void create(Node ** head,int arr[],int n)
{
   Node *p,*last;
   *head=(Node*)malloc(sizeof(Node));
   (*head)->data=arr[0];
   (*head)->next=NULL;
   last=*head;
   for(int i=1;i<n;i++){
      p=(Node*)malloc(sizeof(Node));
      p->data=arr[i];p->next=NULL;
      last->next=p;last=p;

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

```
    }
    printf("NULL\n");
}
void concatenate(Node *head1, Node *head2)
{
    Node *q=head1;
    while(q->next!=NULL){
        q=q->next;
    }
    q->next=head2;
    head2=NULL;
}
```

2)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

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;

AssemblyLine* head = NULL;

// Function prototypes
void createLinkedList(AssemblyLine**, int);
```

```
void insertAssemblyLine(AssemblyLine**, int);
void deleteAssemblyLine(AssemblyLine**, int);
void searchAssemblyLine(AssemblyLine*, int);
void updateAssemblyLine(AssemblyLine*, int, char*);
void displayAllAssemblyLines(AssemblyLine*);

int main() {
    int choice, index, lineID;
    char newStatus[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);
        getchar(); // Consume newline character

        switch (choice) {
        case 1:
            printf("Enter number of assembly lines: ");
            int n;
            scanf("%d", &n);
            createLinkedList(&head, n);
            break;
        case 2:
            printf("Enter the index to insert the new assembly line: ");
            scanf("%d", &index);
            insertAssemblyLine(&head, index);
            break;
        case 3:
            printf("Enter line ID to delete: ");
            scanf("%d", &lineID);
            deleteAssemblyLine(&head, lineID);
            break;
        case 4:
            printf("Enter line ID to search: ");
            scanf("%d", &lineID);
            searchAssemblyLine(head, lineID);
```

```
            break;
        case 5:
            printf("Enter line ID to update status: ");
            scanf("%d", &lineID);
            getchar();
            printf("Enter new status (e.g., Active, Inactive): ");
            scanf("%[^\n]", newStatus);
            updateAssemblyLine(head, lineID, newStatus);
            break;
        case 6:
            displayAllAssemblyLines(head);
            break;
        case 7:
            printf("Exiting program...\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

void createLinkedList(AssemblyLine** head, int n) {
    AssemblyLine *p, *last = NULL;
    for (int i = 0; i < n; i++) {
        p = (AssemblyLine*)malloc(sizeof(AssemblyLine));
        printf("Enter details for Line %d:\n", i + 1);
        printf("Line ID: ");
        scanf("%d", &p->lineID);
        getchar();
        printf("Line name: ");
        scanf("%[^\n]", p->lineName);
        getchar();
        printf("Capacity: ");
        scanf("%d", &p->capacity);
        getchar();
        printf("Status: ");
        scanf("%[^\n]", p->status);
        p->next = NULL;

        if (*head == NULL) {
            *head = p;
        } else {
            last->next = p;
```

```
        }
        last = p;
    }
}

void insertAssemblyLine(AssemblyLine** head, int index) {
    if (index <= 0) {
        printf("Invalid index. Please provide a positive index.\n");
        return;
    }

    AssemblyLine* p = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("Enter details for the new assembly line:\n");
    printf("Line ID: ");
    scanf("%d", &p->lineID);
    getchar();
    printf("Line name: ");
    scanf("%[^\n]", p->lineName);
    getchar();
    printf("Capacity: ");
    scanf("%d", &p->capacity);
    getchar();
    printf("Status: ");
    scanf("%s", p->status);
    p->next = NULL;

    if (index == 1) {
        p->next = *head;
        *head = p;
        return;
    }

    AssemblyLine* current = *head;
    for (int i = 1; current != NULL && i < index - 1; i++) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Index is greater than the length of the list.\n");
        free(p);
        return;
    }

    p->next = current->next;
```

```
    current->next = p;
}

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

    AssemblyLine *current = *head, *previous = NULL;

    if (current != NULL && current->lineID == lineID) {
        *head = current->next;
        free(current);
        printf("Assembly line with ID %d deleted.\n", lineID);
        return;
    }

    while (current != NULL && current->lineID != lineID) {
        previous = current;
        current = current->next;
    }

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

    previous->next = current->next;
    free(current);
    printf("Assembly line with ID %d deleted.\n", lineID);
}

void searchAssemblyLine(AssemblyLine* head, int lineID) {
    AssemblyLine* current = head;
    while (current != NULL) {
        if (current->lineID == lineID) {
            printf("Line ID: %d\n", current->lineID);
            printf("Line Name: %s\n", current->lineName);
            printf("Capacity: %d\n", current->capacity);
            printf("Status: %s\n", current->status);
            return;
        }
        current = current->next;
```

```
    }
    printf("Assembly line with ID %d not found.\n", lineID);
}

void updateAssemblyLine(AssemblyLine* head, int lineID, char* newStatus) {
    AssemblyLine* current = head;
    while (current != NULL) {
        if (current->lineID == lineID) {
            strcpy(current->status, newStatus);
            printf("Status updated for line ID %d.\n", lineID);
            return;
        }
        current = current->next;
    }
    printf("Assembly line with ID %d not found.\n", lineID);
}

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

    AssemblyLine* current = head;
    printf("\nAssembly Lines:\n");
    while (current != NULL) {
        printf("Line ID: %d\n", current->lineID);
        printf("Line Name: %s\n", current->lineName);
        printf("Capacity: %d\n", current->capacity);
        printf("Status: %s\n", current->status);
        current = current->next;
    }
}
```

3)implementation of stack using array

ANSWER:

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

// Define Stack structure
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 isFull(struct Stack*);
int isEmpty(struct Stack*);

// Main function
int main() {
    struct Stack st;
    int elementPeeked, index;
    int elementPopped;

    create(&st);

    push(&st, 10);
    push(&st, 20);
    push(&st, 30);
    push(&st, 40);

    display(&st);

    elementPopped = pop(&st);
    printf("Popped element is %d \n", elementPopped);
    display(&st);

    printf("Enter the position to peek: ");
    scanf("%d", &index);

    elementPeeked = peek(&st, index);
    if (elementPeeked != -1) {
        printf("Peeked element is %d \n", elementPeeked);
    }

    if (isFull(&st)) {
        printf("Stack is full\n");
    } else {
        printf("Stack is not full\n");
    }
```

```
    if (isEmpty(&st)) {
        printf("Stack is empty\n");
    } else {
        printf("Stack is not empty\n");
    }

    // Free allocated memory
    free(st.S);

    return 0;
}

// Function to create the stack
void create(struct Stack *st) {
    printf("Enter size of array: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->S = (int*)malloc(st->size * sizeof(int));
}

// Function to push an element onto the stack
void push(struct Stack *st, int x) {
    if (st->top == st->size - 1) {
        printf("Stack overflow\n");
    } else {
        st->top++;
        st->S[st->top] = x;
    }
}

// Function to display elements of the stack
void display(struct Stack *st) {
    if (st->top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d\n", st->S[i]);
    }
    printf("\n");
}
```

```
// Function to pop an element from the stack
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;
}

// Function to peek an element at a specific position
int peek(struct Stack *st, int pos) {
    int x = -1;
    if (pos <= 0 || st->top - pos + 1 < 0) {
        printf("Invalid position\n");
    } else {
        x = st->S[st->top - pos + 1];
    }
    return x;
}

// Function to check if the stack is full
int isFull(struct Stack *st) {
    return st->top == st->size - 1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack *st) {
    return st->top == -1;
}
```

4)implementation of stack using LL

ANSWER:

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

// Define the Node structure
struct Node {
    int data;
    struct Node* link;
};
```

```
// Global pointer to the top of the stack
struct Node *top = 0;

// Function prototypes
void push(int);
void display();
int pop();
void peek();
int isFull();
int isEmpty();

// Main function
int main() {
    push(10);
    push(20);
    push(30);
    push(40);
    display();
    peek();
    pop();
    display();

    if (isFull()) {
        printf("Stack is full\n");
    } else {
        printf("Stack is not full\n");
    }

    if (isEmpty()) {
        printf("Stack is empty\n");
    } else {
        printf("Stack is not empty\n");
    }

    return 0;
}

// Function to push an element onto the stack
void push(int x) {
    struct Node *newnode;
    newnode = (struct Node*)malloc(sizeof(struct Node));

    if (newnode == NULL) { // Check for stack overflow
```

```
        printf("Stack overflow \n");
        return;
    }

    newnode->data = x;
    newnode->link = top;
    top = newnode;
}

// Function to display the stack elements
void display() {
    struct Node *temp;
    temp = top;

    if (top == NULL) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements:\n");
        while (temp != NULL) {
            printf("%d\n", temp->data);
            temp = temp->link;
        }
    }
}

// Function to pop an element from the stack
int pop() {
    struct Node *temp;
    int poppedData;

    if (top == NULL) { // Check for stack underflow
        printf("Stack underflow\n");
        return -1;
    }

    temp = top;
    poppedData = top->data;
    printf("Popped element is %d\n", top->data);
    top = top->link;
    free(temp);

    return poppedData;
}
```

```
// Function to get the top element of the stack
void peek() {
    if (top == NULL) {
        printf("Stack is empty\n");
    } else {
        printf("Top element is %d\n", top->data);
    }
}

// Function to check if the stack is full
int isFull() {
    struct Node *temp = (struct Node*)malloc(sizeof(struct Node));

    if (temp == NULL) {
        return 1; // Stack is full
    }

    free(temp);
    return 0; // Stack is not full
}

// Function to check if the stack is empty
int isEmpty() {
    return top == NULL;
}
```