1.

```
/*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

```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;

AssemblyLine* head = NULL;
int N;  // Number of lines

// Function prototypes
void createAssemblyLineList();
void insertAssemblyLine(AssemblyLine* p, int index, int lineID, char* lineName,
int capacity, char* status);
void deleteAssemblyLine(AssemblyLine* p, int lineID);
void searchAssemblyLine(AssemblyLine* p, int lineID);
void updateAssemblyLine(AssemblyLine* p, int lineID, char* status);
void displayAssemblyLine(AssemblyLine* p, int i);

int main() {
    printf("Enter the number of assembly lines: ");
    scanf("%d", &N);
```

```
    int choice;
    int index;
    int lineID;
    char lineName[50];
    int capacity;
    char 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:
                createAssemblyLineList();
                break;
            case 2:
                printf("\nEnter index to insert (0 to %d): ", N);
                scanf("%d", &index);

                if (index < 0 || index > N) {
                    printf("Invalid index! Please enter a value between 0
and %d.\n", N);
                    break;
                }

                printf("\nEnter Line ID: ");
                scanf("%d", &lineID);
                printf("Enter Line Name: ");
                scanf("%s", lineName);
                printf("Enter Capacity: ");
                scanf("%d", &capacity);
                printf("Enter Status: ");
                scanf("%s", status);
```

```
                insertAssemblyLine(head, index, lineID, lineName, capacity,
status);
                break;
            case 3:
                printf("\nEnter Line ID to delete: ");
                scanf("%d", &lineID);
                deleteAssemblyLine(head, lineID);
                break;
            case 4:
                printf("\nEnter Line ID to search: ");
                scanf("%d", &lineID);
                searchAssemblyLine(head, lineID);
                break;
            case 5:
                printf("\nEnter Line ID to update status: ");
                scanf("%d", &lineID);
                printf("Enter New Status: ");
                scanf("%s", status);
                updateAssemblyLine(head, lineID, status);
                break;
            case 6:
                displayAssemblyLine(head, 1);
                break;
            case 7:
                exit(0);
            default:
                printf("\nInvalid choice! Please try again.\n");
        }
    }
}

void createAssemblyLineList() {
    AssemblyLine *last = NULL;

    for (int i = 0; i < N; i++) {
        AssemblyLine *newNode = (AssemblyLine *)malloc(sizeof(AssemblyLine));
        if (!newNode) {
            printf("Memory allocation failed!\n");
            return;
        }

        printf("\nEnter details for Assembly Line %d:\n", i + 1);
        printf("Enter Line ID: ");
```

```
        scanf("%d", &newNode->lineID);
        printf("Enter Line Name: ");
        scanf("%s", newNode->lineName);
        printf("Enter Capacity: ");
        scanf("%d", &newNode->capacity);
        printf("Enter Status: ");
        scanf("%s", newNode->status);
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode; // First node becomes the head
            last = newNode;
        } else {
            last->next = newNode; // Link the new node
            last = newNode;
        }
    }
    printf("\nList of assembly lines created successfully.\n");
}

void insertAssemblyLine(AssemblyLine* p, int index, int lineID, char* lineName,
int capacity, char* status) {
    AssemblyLine *t;
    int i;

    if (index < 0 || index > N) {
        printf("\nInvalid position!\n");
        return;
    }

    t = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    t->lineID = lineID;
    strcpy(t->lineName, lineName);
    t->capacity = capacity;
    strcpy(t->status, status);
    t->next = NULL;

    if (index == 0) { // to insert at the beginning
        t->next = head;
        head = t;
    } else { // to insert at any other position
        for (i = 0; i < index - 1; i++) {
            p = p->next;
```

```
        }
        t->next = p->next;
        p->next = t;
    }

    N++;
    printf("\nNew assembly line inserted successfully.\n");
}

void deleteAssemblyLine(AssemblyLine* p, int lineID) {
    AssemblyLine* q = NULL;

    if (head == NULL) {
        printf("\nThe list is empty!\n");
        return;
    }

    if (lineID == p->lineID) { // to delete the first node
        head = head->next;
        free(p);
        N--;
        printf("\nAssembly Line with ID %d deleted successfully.\n", lineID);
        return;
    } else { // to delete from any other index
        while (p != NULL) {
            if (p->lineID == lineID) {
                q->next = p->next;
                free(p);
                N--;
                printf("\nAssembly Line with ID %d deleted successfully.\n",
lineID);
                return;
            }
            q = p;
            p = p->next;
        }
    }
    printf("\nLine ID %d not found in the list.\n", lineID);
}

void searchAssemblyLine(AssemblyLine* p, int lineID) {
    if (p == NULL) {
        printf("\nThe list is empty.\n");
```

```
        return;
    }

    while (p != NULL) {
        if (lineID == p->lineID) {
            printf("\nAssembly Line Details:\n");
            printf("----------------------------------------\n");
            printf("Line ID      : %d\n", p->lineID);
            printf("Line Name    : %s\n", p->lineName);
            printf("Capacity     : %d\n", p->capacity);
            printf("Status       : %s\n", p->status);
            printf("----------------------------------------\n");
            return;
        }
        p = p->next;
    }
    printf("\nLine ID %d not found in the list.\n", lineID);
}

void updateAssemblyLine(AssemblyLine* p, int lineID, char* status) {
    if (p == NULL) {
        printf("\nThe list is empty.\n");
        return;
    }

    while (p != NULL) {
        if (lineID == p->lineID) {
            strcpy(p->status, status);
            printf("\nAssembly Line status updated successfully.\n");
            return;
        }
        p = p->next;
    }
    printf("\nLine ID %d not found in the list.\n", lineID);
}

void displayAssemblyLine(AssemblyLine* p, int i) {
    if (p != NULL) {
        printf("\nAssembly Line %d Details:\n", i);
        printf("----------------------------------------\n");
        printf("Line ID      : %d\n", p->lineID);
        printf("Line Name    : %s\n", p->lineName);
        printf("Capacity     : %d\n", p->capacity);
```

```
        printf("Status       : %s\n", p->status);
        printf("----------------------------------------\n");

        displayAssemblyLine(p->next, i + 1);
    }
}
```

PS C:\Users\betti\Desktop\Training\Day22> ./task1

Enter the number of assembly lines: 2

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

Enter your choice: 1

Enter details for Assembly Line 1:

Enter Line ID: 101

Enter Line Name: m

Enter Capacity: 5000

Enter Status: active

Enter details for Assembly Line 2:

Enter Line ID: 102

Enter Line Name: s

Enter Capacity: 6000

Enter Status: inactive

List of assembly lines created successfully.

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

Enter your choice: 2

Enter index to insert (0 to 2): 1

Enter Line ID: 103

Enter Line Name: n

Enter Capacity: 7000

Enter Status: active

New assembly line inserted successfully.

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

Enter your choice: 6

Assembly Line 1 Details:

Line ID : 101

Line Name : m

Capacity : 5000

Status : active

Assembly Line 2 Details:

Line ID : 103

Line Name : n

Capacity : 7000

Status : active

Assembly Line 3 Details:

Line ID : 102

Line Name : s

Capacity : 6000

Status : inactive

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

Enter your choice: 3

Enter Line ID to delete: 102

Assembly Line with ID 102 deleted successfully.

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

Enter your choice: 6

Assembly Line 1 Details:

Line ID : 101

Line Name : m

Capacity : 5000

Status : active

Assembly Line 2 Details:

Line ID : 103

Line Name : n

Capacity : 7000

Status : active

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

Enter your choice: 4

Enter Line ID to search: 101

Assembly Line Details:

Line ID : 101

Line Name : m

Capacity : 5000

Status : active

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

Enter your choice: 5

Enter Line ID to update status: 101

Enter New Status: inactive

Assembly Line status updated successfully.

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

Enter your choice: 6

Assembly Line 1 Details:

Line ID : 101

Line Name : m

Capacity : 5000

Status : inactive

Assembly Line 2 Details:

Line ID : 103

Line Name : n

Capacity : 7000

Status : active

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

Enter your choice: 7

2.

```c
//implementation of stack using arrays

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

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 peekIndex(struct stack *, int);
int peekPosition(struct stack , int);
int isEmpty(struct stack *);
```

```c
int isFull(struct stack *);

int main() {
    struct stack st;

    create(&st);

    push(&st, 10);
    push(&st, 20);
    push(&st, 30);
    push(&st, 40);
    push(&st, 50);

    printf("Stack elements are:\n");
    display(&st);

    printf("Popped element: %d\n", pop(&st));
    printf("Stack elements after pop:\n");
    display(&st);

    printf("Result of Peek at 2 (by index): %d\n", peekIndex(&st, 2));
    printf("Result of Peek at 3 (by position): %d\n", peekPosition(st, 3));

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

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

    return 0;
}

void create(struct stack *st) {
    printf("Enter size of stack: ");
```

```
    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\n");
        return;
    }
    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");
    }
}

int pop(struct stack *st) {
    int x = -1;
    if(st->top == -1) {
        printf("Stack Underflow\n");
        return x;
    }
    x = st->S[st->top];
    st->top--;
    return x;
}

int peekIndex(struct stack *st, int index) {
    if(index < 0 || index > st->top) {
        printf("Invalid index\n");
        return -1;
    }
    return st->S[index];
}

int peekPosition(struct stack st, int position) {
    if(st.top - position + 1 < 0 || position > st.top + 1) {
        printf("Invalid index\n");
```

```
        return -1;
    }
    return st.S[st.top - position + 1];
}

int isEmpty(struct stack *st) {
    if(st->top == -1) {
        printf("Stack Underflow\n");
        return 1;
    }
    return 0;
}

int isFull(struct stack *st) {
    if(st->top == st->size - 1) {
        return 1;
    }
    return 0;
}
```

```
PS C:\Users\betti\Desktop\Training\Day22> ./test4
Enter size of stack: 5
Stack elements are:
50
40
30
20
10
Popped element: 50
Stack elements after pop:
40
30
20
10
Result of Peek at 2 (by index): 30
Result of Peek at 3 (by position): 20
Stack is not empty
Stack is not full
```

3.

```c
//implementation of stack using linked list

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

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

Node* top = NULL;
int currentSize = 0; // To track the current size of the stack
int maxSize = 0;     // To store the maximum size of the stack

// Function prototypes
void createStack(int size);
void push(int num);
int pop();
int peek(int index);
int isEmpty();
int isFull();
void display();

int main() {
    int choice;

    while (1) {
        printf("\n1. Create Stack\n");
        printf("2. Push\n");
        printf("3. Pop\n");
        printf("4. Peek\n");
        printf("5. Display\n");
        printf("6. Check if Stack is Empty\n");
        printf("7. Check if Stack is Full\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int size;
```

```
            printf("Enter the maximum size of the stack: ");
            scanf("%d", &size);
            createStack(size);
            break;
        }
        case 2: {
            if (isFull()) {
                printf("Stack is full! Cannot push more elements.\n");
            } else {
                int num;
                printf("Enter the number to push: ");
                scanf("%d", &num);
                push(num);
            }
            break;
        }
        case 3: {
            if (isEmpty()) {
                printf("Stack is empty! Nothing to pop.\n");
            } else {
                int popped = pop();
                printf("Popped element is %d\n", popped);
            }
            break;
        }
        case 4: {
            if (isEmpty()) {
                printf("Stack is empty! Nothing to peek.\n");
            } else {
                int index;
                printf("Enter the index to peek: ");
                scanf("%d", &index);
                int peeked = peek(index);
                if (peeked != -1)
                    printf("Peeked element is %d\n", peeked);
            }
            break;
        }
        case 5:
            display();
            break;
        case 6:
            if (isEmpty()) {
                printf("Stack is empty.\n");
```

```
                } else {
                    printf("Stack is not empty.\n");
                }
                break;
            case 7:
                if (isFull()) {
                    printf("Stack is full.\n");
                } else {
                    printf("Stack is not full.\n");
                }
                break;
            case 8:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
}

// Function to create the stack with a maximum size
void createStack(int size) {
    maxSize = size;
    top = NULL;
    currentSize = 0;
    printf("Stack created with a maximum size of %d\n", maxSize);
}

// Push operation
void push(int num) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed! Stack overflow.\n");
        return;
    }
    newNode->data = num;
    newNode->next = top;
    top = newNode;
    currentSize++;
    printf("Pushed %d onto the stack.\n", num);
}

// Pop operation
```

```c
int pop() {
    if (isEmpty()) {
        printf("Stack underflow!\n");
        return -1;
    }
    int popped = top->data;
    Node* temp = top;
    top = top->next;
    free(temp);
    currentSize--;
    return popped;
}

// Peek operation
int peek(int index) {
    Node* temp = top;
    for (int i = 0; temp != NULL && i < index; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Index out of range!\n");
        return -1;
    }
    return temp->data;
}

// Check if the stack is empty
int isEmpty() {
    return currentSize == 0;
}

// Check if the stack is full
int isFull() {
    return currentSize == maxSize;
}

// Display the stack
void display() {
    if (isEmpty()) {
        printf("Stack is empty!\n");
        return;
    }
    Node* temp = top;
```

```
    printf("Stack elements:\n");
    while (temp != NULL) {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}
```

PS C:\Users\betti\Desktop\Training\Day22> ./task2

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Exit

Enter your choice: 6

Exiting...

PS C:\Users\betti\Desktop\Training\Day22> gcc -o task2 task2.c

PS C:\Users\betti\Desktop\Training\Day22> ./task2

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 1

Enter the maximum size of the stack: 4

Stack created with a maximum size of 4

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 2

Enter the number to push: 20

Pushed 20 onto the stack.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 2

Enter the number to push: 30

Pushed 30 onto the stack.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 2

Enter the number to push: 40

Pushed 40 onto the stack.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 2

Enter the number to push: 50

Pushed 50 onto the stack.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 2

Stack is full! Cannot push more elements.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 10

Invalid choice!

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 3

Popped element is 50

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 4

Enter the index to peek: 2

Peeked element is 20

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 5

Stack elements:

40

30

20

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 4

Enter the index to peek: 0

Peeked element is 40

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 7

Stack is not full.

1. Create Stack
2. Push
3. Pop
4. Peek
5. Display
6. Check if Stack is Empty
7. Check if Stack is Full
8. Exit

Enter your choice: 3

Popped element is 40

1. Create Stack
2. Push
3. Pop
4. Peek
5. Display
6. Check if Stack is Empty
7. Check if Stack is Full
8. Exit

Enter your choice: 3

Popped element is 30

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 3

Popped element is 20

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 6

Stack is empty.

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 5

Stack is empty!

1. Create Stack

2. Push

3. Pop

4. Peek

5. Display

6. Check if Stack is Empty

7. Check if Stack is Full

8. Exit

Enter your choice: 8

Exiting...