1.

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


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




Node *head = NULL;
void display1(Node *);

int NCount(struct Node *);
void create(int A[], int );
int DeleteNode(Node *,int );
int isloop(Node *);

int main() {
    node *t1,*t2;
    int A[] = {20,30,40,60,70};
    int delVar = 0;
    create(A,4);
    //Foring a loop
    t1  =head->next->next;
    t2 = head->next->next->next->next;
    t2->next = t1;

    printf("Original list: ");
    display1(head);
    printf("\n");


   /*printf("Number of nodes (recursion): %d\n", NCount(head));



    delVar = DeleteNode(head,3);

    if (delVar != -1) {
        printf("Deleted node data: %d\n", delVar);
    } else {
        printf("Deletion failed.\n");
    }
```

```
    display1(head);
    printf("\n");
*/
    if (detectLoop(head)) {
        printf("Loop detected in the linked list.\n");
    } else {
        printf("No loop detected in the linked list.\n");
    }

    return 0;
}
//function to display using recursion
void display1(Node *p) {
if (p != NULL) {
printf("%d -> ", p->data);
display1(p->next);
}
}



//function to count using fn

int NCount(struct Node *p){

    int count=0;
    while(p){
        count++;
        p=p->next;
    }
    return count;
}
//create a linked list from an array
void create(int A[], int n) {
    Node *temp, *last;
    head = (Node *)malloc(sizeof(Node)); // Allocate memory for the head node
    head->data = A[0];                   // Assign the first value to the head
    head->next = NULL;                   // Set the next pointer of head to
NULL
    last = head;                         // Last points to the last node
(currently head)

    for (int i = 1; i < n; i++) {
        temp = (Node *)malloc(sizeof(Node)); // Allocate memory for a new node
        temp->data = A[i];                   // Assign data from the array
        temp->next = NULL;                   // Set the next pointer of the new
node to NULL
        last->next = temp;                   // Link the last node to the new
node
```

```
        last = temp;                        // Update last to point to the new
node
    }
}


int DeleteNode(Node *p,int index){
    Node *q = NULL;
    int x = -1,i;
    if(index < 1 || index>NCount(p)){
        return -1;

    }
    if(index == 1){
        x = head->data;
        head= head->next;
        free(p);
        return x;
    }
    else
    {
        p = head;
        for(i=0;i<index-1 && p;i++)
        {
            q = p;
            p=p->next;
        }
            q->next = p->next;
            x = p->data;
            free(p);
            return x;
    }
}
//Detecting a loop in the linked list

int isloop(Node *){
    Node *p = head, *q = head;

    while (p && q && q->next) {
        p = p->next;       // `p` moves one step
        q = q->next->next; // `q` moves two steps

        // If `p` and `q` meet, a loop is detected
        if (p == q) {
            return 1;
        }
    }
    return 0;
}
```

2.

```c
/*
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 create(Node **,int A[], int );
void display1(Node *);
void concateLL(Node *,Node *);

int main(){
    int a1[] = {20,30,40};


    int a2[] = {50,60,70};


    Node *head1 = NULL; //head pointer for Linked List_1;
    Node *head2 = NULL; //head pointer for Linked List_2;


    //Creating Linked List_1 by calling create fn;

    create(&head1,a1,3);


    printf("Linked list_1 : ");
    display1(head1);

    //Creating Linked List_2 by calling create fn;
    create(&head2,a2,3);

    printf("\nLinked list_2 : ");
    display1(head2);
```

```
    //Calling function to Concatinate the two linked list
    concateLL(head1,head2);
    printf("\nLinked list after concatenation : ");
    display1(head1);

    //print the concatenated linked list here using display function
    return 0;

}
//create a linked list from an array
void create(Node **head,int A[], int n) {
    Node *temp, *last;
    *head = (Node *)malloc(sizeof(Node)); // Allocate memory for the head node
of LL
    (*head)->data = A[0];                 // Assign the first value to the
head
    (*head)->next = NULL;                 // Set the next pointer of head to
NULL
    last = (*head);                       // Last points to the last node
(currently head)

    for (int i = 1; i < n; i++) {
        temp = (Node *)malloc(sizeof(Node)); // Allocate memory for a new node
        temp->data = A[i];                // Assign data from the array
        temp->next = NULL;                // Set the next pointer of the new
node to NULL
        last->next = temp;                // Link the last node to the new
node
        last = temp;                      // Update last to point to the new
node
    }
}
void display1(Node *p) {
if (p != NULL) {
printf("%d -> ", p->data);
display1(p->next);
}
}
void concateLL(Node *first,Node *second)
{
    Node *p = first;
    while(p->next!=NULL)
    {
        p=p->next;
    }
    p->next = second;

}
```

3.

```
/*
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:
o   lineID (integer): Unique identifier for the assembly line.
o   lineName (string): Name of the assembly line (e.g., "Chassis Assembly").
o   capacity (integer): Maximum production capacity of the line per shift.
o   status (string): Current status of the line (e.g., "Active", "Under
Maintenance").
o   next (pointer to the next node): Link to the next assembly line in the
list.
2.  Linked List:
o   The linked list will store a dynamic number of assembly lines, allowing
for additions and removals as needed.

Features to Implement
1.  Creation:
o   Initialize the linked list with a specified number of assembly lines.
2.  Insertion:
o   Add a new assembly line to the list either at the beginning, end, or at a
specific position.
3.  Deletion:
o   Remove an assembly line from the list by its lineID or position.
4.  Searching:
o   Search for an assembly line by lineID or lineName and display its details.
5.  Display:
o   Display all assembly lines in the list along with their details.
6.  Update Status:
o   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:
o   Create Linked List of Assembly Lines
o   Insert New Assembly Line
o   Delete Assembly Line
o   Search for Assembly Line
o   Update Assembly Line Status
```

o   Display All Assembly Lines
o   Exit
2. Sample Input/Output:
Input:
o   Number of lines: 3
o   Line 1: ID = 101, Name = "Chassis Assembly", Capacity = 50, Status = "Active".
o   Line 2: ID = 102, Name = "Engine Assembly", Capacity = 40, Status = "Under Maintenance".
Output:
•   Assembly Lines:
o   Line 101: Chassis Assembly, Capacity: 50, Status: Active
o   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;

//Function declaration
void create();
void insertAssemblyLine();
void deleteAssemblyLine();
void searchAssemblyLine();
void updateStatus();
void displayAssemblyLines();

int main()
{
    printf("### Automotive Manufacturing Plant Management System ###\n");
    printf("\n");


    int choice;



    printf("### MENU ###");
    printf("\n1. Create Linked List of Assembly Lines\n2. Insert New Assembly
Line\n3. Delete Assembly Line\n4. Search for Assembly Line\n5. Update Assembly
Line Status\n6. Display All Assembly Lines\n7. Exit");
    printf("\nChoose which operation should be done : ");
    scanf("%d",&choice);
```

```c
    do
    {
        switch (choice)
        {
        case 1:
            create();
            break;
        case 2: insertAssemblyLine(); break;
        case 3: deleteAssemblyLine(); break;
        case 4: searchAssemblyLine(); break;
        case 5: updateStatus(); break;
        case 6: displayAssemblyLines(); break;
        case 7: printf("Exiting the program. Goodbye!\n"); break;

        default:    printf("\nInvalid Choice !");
            break;
        }
    } while (choice != 7);

    return 0;

}
void create(){
    int nALines;
    printf("Number of lines : ");
    scanf("%d",&nALines);
    for (int i = 0; i < nALines; i++) {
        AssemblyLine* newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
        printf("\nEnter details for assembly line %d:\n", i + 1);
        printf("Line ID: ");
        scanf("%d", &newNode->lineID);
        printf("Line Name: ");
        scanf(" %[^\n]", newNode->lineName); // Space before % to consume
leftover newline
        printf("Capacity: ");
        scanf("%d", &newNode->capacity);
        printf("Status: ");
        scanf(" %[^\n]", newNode->status);

        newNode->next = head;
        head = newNode;
    }
}
// Function to insert a new assembly line
void insertAssemblyLine() {
    AssemblyLine* newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    int position, count = 1;
```

```
    printf("\nEnter details for the new assembly line:\n");
    printf("Line ID: ");
    scanf("%d", &newNode->lineID);
    printf("Line Name: ");
    scanf(" %[^\n]", newNode->lineName);
    printf("Capacity: ");
    scanf("%d", &newNode->capacity);
    printf("Status: ");
    scanf(" %[^\n]", newNode->status);
    newNode->next = NULL;

    printf("Enter position to insert (1 for start, 0 for end): ");
    scanf("%d", &position);

    if (position == 1) {
        newNode->next = head;
        head = newNode;
    } else {
        AssemblyLine* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
            count++;
        }
        temp->next = newNode;
    }
}

// Function to delete an assembly line by lineID
void deleteAssemblyLine() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    int id;
    printf("Enter the Line ID to delete: ");
    scanf("%d", &id);

    AssemblyLine *temp = head, *prev = NULL;

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

    if (temp == NULL) {
        printf("Assembly Line with ID %d not found.\n", id);
        return;
```

```
    }

    if (prev == NULL) {
        head = temp->next;
    } else {
        prev->next = temp->next;
    }

    free(temp);
    printf("Assembly Line with ID %d deleted successfully.\n", id);
}

// Function to search for an assembly line
void searchAssemblyLine() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    int id;
    printf("Enter the Line ID to search: ");
    scanf("%d", &id);

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

    printf("Assembly Line with ID %d not found.\n", id);
}

// Function to update the status of an assembly line
void updateStatus() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }

    int id;
    printf("Enter the Line ID to update: ");
```

```
    scanf("%d", &id);

    AssemblyLine* temp = head;
    while (temp != NULL) {
        if (temp->lineID == id) {
            printf("Enter new status: ");
            scanf(" %[^\n]", temp->status);
            printf("Status updated successfully.\n");
            return;
        }
        temp = temp->next;
    }

    printf("Assembly Line with ID %d not found.\n", id);
}

// Function to display all assembly lines
void displayAssemblyLines() {
    if (head == NULL) {
        printf("No assembly lines available.\n");
        return;
    }

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

4.

```
//Creating a stack using ARRAY
#include <stdio.h>
#include <stdlib.h>

//Structure Declaration
struct Stack{
    int size;
    int top;
    int *S;
};


//Function Prototypes :
```

```c
void create(struct Stack *);
void push(struct Stack *,int);
void display(struct Stack *);
int pop(struct Stack *);
int peek(struct Stack *,int);

int main(){
    struct Stack st;
    create(&st);
    int elementPop,peekelmnt,pos;

    //For pushing elements in the stack;
    push(&st,10);
    push(&st,20);
    push(&st,30);
    push(&st,40);
    //For displaying the created stack;
    printf("\nBefore Popping the stack contains these elements : \n");
    display(&st);

    //Popping element from the stack;
    elementPop = pop(&st);

    printf("\nThe Popped element is %d",elementPop);

    //For displaying the created stack;
    printf("\nAfter Popping the stack contain these elements : \n");
    display(&st);

    //For peeking an element getting the index value ..
    printf("\nEnter the index number : ");
    scanf("%d",&pos);
    //Peek operation means : peek(index) returns the value stored in the index
2;
    peekelmnt = peek(&st,pos);

    printf("\nThe picked element is %d from index %d",peekelmnt,pos);


    return 0;
}

//Function for creating the Stack
void create(struct Stack *st){
    printf("\nEnter the size of the array : ");
    scanf("%d",&st->size);

    //As initially the stack is empty initialising top of the stack with -1;
    st->top = -1;
```

```
    //Heap memory is allocated or created;
    st->S = (int*)malloc(st->size*sizeof(int));
}

//Function for pushing element in the stack
void push(struct Stack *st,int x){
    if(st->top==st->size-1)
    {
        printf("\nStack Overflow!!!");
    }else{
        st->top++;
        st->S[st->top]=x;
    }

}

//Function for displaying the stack elements

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("\nStack Underflow or Empty!!");
    }
    else{
        x = st->S[st->top];
        st->top--;
    }
    return x;

}
int peek(struct Stack *st,int pos){
    int x=-1;


    if(st->top-pos+1<0)
    {
        printf("\nInvalid Index!!!");
```

```
    }
    else{
            x = st->S[st->top-pos+1];
            return x;

        }

}
```

5.

```
//Linked List Concept
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct Node {
    int data;
    struct Node *next;
}Node;
Node *head = NULL;
void display1(Node *);
void display2(Node *);
int nCount(Node *);
int rCount(Node *);
int nSum(Node *);
int rSum(Node *);
int nMax(Node *);
int rMax(Node *);
Node* nSearch(Node *, int);
void insert(Node *, int, int);
void create(int *, int);
int DeleteNode(Node *,int );
int isloop(Node *)
int main() {
    Node *t1,*t2;
    int A[] = {10, 20, 30, 40, 50};
    int delVar = 0;
    create(A, 5);
    //Foring a loop
    t1 = head->next->next;
    t2 = head->next->next->next->next;
    t2->next = t1;
    printf("Original list: ");
    display1(head);
    /*
    printf("\n");
    printf("Reversed list: ");
    display2(head);
```

```
    printf("\nNumber of nodes (iteration): %d\n", nCount(head));
    printf("Number of nodes (recursion): %d\n", rCount(head));
    printf("Sum of elements (iteration): %d\n", nSum(head));
    printf("Sum of elements (recursion): %d\n", rSum(head));
    printf("Max of elements (iteration): %d\n", nMax(head));
    printf("Max of elements (recursion): %d\n", rMax(head));
    Node *key = nSearch(head, 20);
    printf("Element found: %d\n", key->data);
    insert(head, 0, 10);
    display1(head);
    printf("\nNumber of nodes (iteration): %d\n", nCount(head));
    insert(head, 4, 50);
    display1(head);
    printf("\nNumber of nodes (recursion): %d\n", rCount(head));
    delVar = DeleteNode(head, 7);
    printf("Node Deleted = %d\n",delVar);
    display1(head);
    delVar = DeleteNode(head, 4);
    printf("Node Deleted = %d\n",delVar);
    display1(head);
    delVar = DeleteNode(head, 4);
    printf("Node Deleted = %d\n",delVar);
    display1(head);
    */
    return 0;
}
//function to display using recursion
void display1(Node *p) {
    if (p != NULL) {
        printf("%d -> ", p->data);
        display1(p->next);
    }
}
//function to display using recursion but in reverse order
void display2(Node *p) {
    if (p != 0) {
        display2(p->next);
        printf("%d <- ", p->data);
    }
}
//function to count the number of nodes in the linked list
int nCount(Node *p) {
    int c = 0;
    while (p) {
        c++;
        p = p->next;
    }
    return c;
```

```
}
//function to count using recursion
int rCount(Node *p) {
    if (p == 0) {
        return 0;
    } else {
        return 1 + rCount(p->next);
    }
}
//function to find sum using iteration
int nSum(Node *p) {
    int sum = 0;
    while (p) {
        sum += p->data;
        p = p->next;
    }
    return sum;
}
//function to find sum using recursion
int rSum(Node *p) {
    int sum = 0;
    if (!p) {
        return 0;
    } else {
        sum += p->data;
        return sum + rSum(p->next);
    }
}
//function to find maximum using iteration
int nMax(Node *p) {
    int max = INT_MIN;
    while(p != NULL) {
        if((p->data) > max) {
            max = p->data;
        }
    p = p->next;
    }
    return max;
}
//function to find max using recursion
int rMax(Node *p) {
    int max = INT_MIN;
    if (p == 0) {
        return INT_MIN;
    }
    else {
        max = rMax(p->next);
        if(max > p->data)
```

```
            return max;
        else
            return p->data;
    }
}
//function to find the element
Node* nSearch(Node *p, int key) {
    while(p != NULL) {
        if(key == p->data)
            return p;
        p = p -> next;
    }
    return NULL;
}
//to insert at a position
void insert(Node *p, int index, int x) {
    Node *t;
    int i;
    if(index < 0 || index > nCount(p)) {
        printf("\nInvalid position!");
    }
    t = (Node*)malloc(sizeof(Node));
    t->data = x;
    if(index == 0) {
        t->next = head;
        head = t;
    } else {
        for(i = 0; i < index-1; i++) {
            p = p->next;
        }
        t->next = p->next;
        p->next = t;
    }
}
//to create a linked list from an array
void create(int A[], int n) {
    Node *p, *last;
    head = (Node *)malloc(sizeof(Node));
    head->data = A[0];
    head->next = NULL;
    last = head;
    for(int i = 1; i < n; i++) {
        p = (Node *)malloc(sizeof(Node));
        p->data = A[i];
        p->next = NULL;
        last->next = p;
        last = p;
    }
```

```
}

int DeleteNode(Node *p,int index){
    Node *q = NULL;
    int x = -1, i;
    if(index < 1 || index > nCount(p)){
        return -1;
    }
    if(index == 1){
       x = head->data;
       head = head->next;
       free(p);
       return x;
    }
    else{
        p = head;
        for(i = 0; i < index - 1 && p;i++){
            q = p;
            p = p->next;
        }
            q->next = p->next;
            x = p->data;
            free(p);
            return x;
    }
}
```