```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 *);
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[] = {20, 30, 40, 60, 70};
   create(A, 5);  // Create the linked list
   int delvar = 0;

   // Display the original list before forming a loop
   printf("Original list before loop creation: ");
   display1(head);
   printf("\n");

   // Forming a loop
   t1 = head->next->next;  // 40
   t2 = head->next->next->next->next;  // 70
   t2->next = t1;  // Create the loop

   // Check if the loop exists
   if (isloop(head)) {
      printf("Loop detected in the linked list.\n");
   } else {
      printf("No loop detected in the linked list.\n");
   }
```

```
    // Do not display after loop creation (as it would cause an infinite loop)
    // display1(head);  // This would result in infinite loop!

    // Other operations
    /*
    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, 5);
    printf("Node deleted: %d\n", delvar);
    display1(head);
    */

    return 0;
}

// Function to display the list using iteration
void display1(Node *p) {
    while (p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
}

// Function to display the list using recursion (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 (iteration)
int nCount(Node *p) {
    int c = 0;
    while (p) {
        c++;
        p = p->next;
    }
    return c;
}

// Function to count the number of nodes using recursion
int rCount(Node *p) {
    if (p == 0) {
        return 0;
    } else {
        return 1 + rCount(p->next);
    }
}

// Function to find sum of elements using iteration
int nSum(Node *p) {
    int sum = 0;
    while (p) {
        sum += p->data;
        p = p->next;
    }
    return sum;
}

// Function to find sum of elements using recursion
int rSum(Node *p) {
    if (!p) {
        return 0;
    } else {
        return p->data + rSum(p->next);
    }
}

// Function to find maximum element 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 maximum element using recursion
```

```
int rMax(Node *p) {
    if (p == 0) {
        return INT_MIN;
    } else {
        int max = rMax(p->next);
        return (max > p->data) ? max : p->data;
    }
}

// Function to find an element in the linked list
Node* nSearch(Node *p, int key) {
    while (p != NULL) {
        if (key == p->data)
            return p;
        p = p->next;
    }
    return NULL;
}

// Function to insert a node at a specific index
void insert(Node *p, int index, int x) {
    Node *t;
    int i;

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

    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;
    }
}

// Function to create the 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;
    }
}

// Function to delete a node at a specific index
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 {
        for (i = 0; i < index - 1 && p; i++) {
            q = p;
            p = p->next;
        }
        q->next = p->next;
        x = p->data;
        free(p);
        return x;
    }
}

// Function to detect a loop
int isloop(Node *p) {
    Node *slow, *fast;
    slow = fast = p;

    while (slow && fast && fast->next) {
        slow = slow->next;  // Move slow pointer one step
        fast = fast->next->next;  // Move fast pointer two steps
        if (slow == fast) {  // If they meet, a loop is detected
            return 1;  // Loop detected
        }
    }
    return 0;  // No loop detected
}
```

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>
typedef struct AssemblyLine {
   int lineID;
   char lineName[50];
   int capacity;
   char status[20];
   struct AssemblyLine* next;
}AssemblyLine;
AssemblyLine* create(AssemblyLine*);
AssemblyLine* insert(AssemblyLine*);
AssemblyLine* delete(AssemblyLine*);
AssemblyLine* update(AssemblyLine*);
void search(AssemblyLine*);
void display(AssemblyLine*);
void sdisplay(AssemblyLine*);
int main(){
    int op;
    printf("\nASSEMBLY LINE");
    AssemblyLine* head=NULL;
    while(1){
        printf("\nMenu");
        printf("\n1. Create Linked List of Assembly Lines");
        printf("\n2. Insert New Assembly Line");
        printf("\n3. Delete Assembly Line");
        printf("\n4. Search for Assembly Line");
        printf("\n5. Update Assembly Line Status");
        printf("\n6. Display All Assembly Lines");
        printf("\n7. Exit");
        printf("\nChoose Option::");
        scanf("%d",&op);
        switch(op){
            case 1:
                head=create(head);
                break;
            case 2:
                head=insert(head);
                break;
            case 3:
                head=delete(head);
                break;
            case 4:
                search(head);
                break;
            case 5:
                head=update(head);
                break;
            case 6:
                display(head);
                break;
            case 7:
                printf("\nExiting....");
                free(head);
                return 0;
        }
    }
}
AssemblyLine* create(AssemblyLine* head){
    AssemblyLine *temp,*newnode;
    newnode=(AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("\nEnter LineID::");
    scanf("%d",&newnode->lineID);
    printf("\nEnter Line Name::");
    scanf(" %[^\n]",newnode->lineName);
    printf("\nEnter Line Capacity::");
    scanf("%d",&newnode->capacity);
    printf("\nEnter Status(ACTIVE or UNDERMAINTAINANCE)::");
    scanf(" %[^\n]",newnode->status);
    newnode->next=NULL;
    if(head==NULL){
        head=newnode;
    }
    else{
        temp=head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=newnode;
    }
    return head;
}
AssemblyLine* insert(AssemblyLine* head){
    int pos;
    AssemblyLine *temp=head,*newnode;
    printf("\nEnter position to add new::");
    scanf("%d",&pos);
    for(int i=1;i<pos-1;i++){
        temp=temp->next;
    }
    newnode=(AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("\nEnter LineID::");
    scanf("%d",&newnode->lineID);
    printf("\nEnter Line Name::");
    scanf(" %[^\n]",newnode->lineName);
    printf("\nEnter Line Capacity::");
    scanf("%d",&newnode->capacity);
    printf("\nEnter Status(ACTIVE or UNDERMAINTAINANCE)::");
    scanf(" %[^\n]",newnode->status);
    if(pos==1){
        newnode->next=head;
        head=newnode;
    }
    else{
        newnode->next=temp->next;
        temp->next=newnode;
    }
    return head;

}
AssemblyLine* delete(AssemblyLine *head){
    int pos;
    AssemblyLine *temp=head,*prev;
    printf("\nEnter position of Assembly line to delete::");
    scanf("%d",&pos);
    if(pos==1){
        head=head->next;
    }
    else{
        for(int i=0;i<pos-1;i++){
            prev=temp;
            temp=temp->next;
        }
        prev->next=temp->next;
    }
    free(temp);
    printf("\nAssembly Line Deleted..");
    return head;

}
void search(AssemblyLine *head){
    AssemblyLine *temp=head;
    int id;
    printf("\nEnter ID to search::");
    scanf("%d",&id);
    while(temp!=NULL){
        if(temp->lineID==id){
            sdisplay(temp);
        }
        else{
            printf("\nNOT FOUND");
        }
        temp=temp->next;
    }
}
AssemblyLine* update(AssemblyLine* head){
    int id;
    AssemblyLine *temp=head;
    printf("\nEnter ID of Assembly Line to Update::");
    scanf("%d",&id);
    while(temp!=NULL){
        if(temp->lineID==id){
            printf("\nEnter new Status(ACTIVE or UNDERMAINTAINANCE)::");
            scanf(" %[^\n]",temp->status);
        }
        else{
            printf("\nNOT FOUND");
        }
        temp=temp->next;
    }
    return head;

}
void display(AssemblyLine* head){
    AssemblyLine *temp=head;
    while(temp!=NULL){
        printf("\nLINE %d, NAME::%s, CAPACITY::%d,
STATUS::%s->",temp->lineID,temp->lineName,temp->capacity,temp->status);
        temp=temp->next;
    }
    printf("\nNULL");
}
void sdisplay(AssemblyLine *temp){
    while(temp!=NULL){
        printf("\nLINE %d, NAME::%s, CAPACITY::%d,
STATUS::%s->",temp->lineID,temp->lineName,temp->capacity,temp->status);
        break;
    }
}
```

2)//std::stack<

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

typedef struct{
    int size;
    int top;//index of top element
    int *s;//pointing to integer array

}stack;

void create(stack *);
void push(stack*,int);
void display(stack*);
int pop(stack *);
void peek(stack *);
int main(){

    stack st;

    create(&st);

    push(&st,10);
    push(&st,20);
    push(&st,30);
    push(&st,40);
    display(&st);
    int poped_element=pop(&st);
    printf("Popped element is %d\n",poped_element);
    display(&st);
    peek(&st);
    return 0;
}


void create(stack *st){

  printf("enter size:");
  scanf("%d",&st->size);
  st->top=-1;

  st->s=(int *)malloc(st->size*sizeof(int));

}
```

```c
void push(stack *st,int x){
    if(st->top==st->size-1){
        printf("overflow\n");
    }else{
        st->top++;
        st->s[st->top]=x;
    }

}


void display(stack *st){

    for(int i=st->top;i>=0;i--){
        printf("%d",st->s[i]);
        printf("\n");
    }


}




int pop(stack *st){
    int x=-1;

    if(st->top==-1){
        printf("stack underflow\n");
    }else{
        x=st->s[st->top];
        st->top--;
    }
    return x;
}


void peek(stack *st){
    int index;
    printf("enter the index to peek:");
    scanf("%d",&index);
    if(st->top==-1){
        printf("underflow\n");
    }
    printf("Value at index %d is%d\n",index,st->s[index]);
    printf("top element:%d",st->s[st->top]);

}
```

3)