-_-
_-

```
//Deleting a node
#include<stdio.h>
#include<stdlib.h>

typedef struct node{
   int data;
   struct node *next;
}NODE;

NODE *head=NULL;// defining globally bcoz any change in the node using fn will no be reflected in the main;

void display(NODE*);
NODE *Lsearch(NODE*,int);
void insert(NODE*,int,int);
void create(int *,int);
int DeleteNode(NODE*,int);

int main(){

   int A[]={10,20,30,40,50};
   create(A,5);
   display(head);
   NODE *srch=Lsearch(head,20);
   printf("\n%d\n",*srch);
   insert(head,3,15);
   display(head);

   int deleteVar=DeleteNode(head,1);
   printf("\nNode deleted is %d\n",deleteVar);
   display(head);

   deleteVar=DeleteNode(head,4);
   printf("\nNode deleted is %d\n",deleteVar);
   display(head);

   return 0;
}
//fn to create an array to linklist
void create(int *arr,int n){
   NODE *t,*last;
   head=(NODE*)malloc(sizeof(NODE));
   head->data=arr[0];
   head->next=NULL;
   last=head;
   for(int i=1;i<n;i++){
      t=(NODE*)malloc(sizeof(NODE));
      t->data=arr[i];
      t->next=NULL;
      last->next=t;
      last=t;

   }
}
```

```
//fn to count number of elemnts in the list
int Ncount(NODE *ptr){
   int count=0;
   while(ptr!=NULL){
      count++;
      ptr=ptr->next;
   }
   return count;
}
//fn to display the linked list
void display(NODE *ptr){
   if(ptr != NULL){
      printf("%d->",ptr->data);
      display(ptr->next);
   }
}

//fn to search for a particular element
NODE *Lsearch(NODE *ptr,int key){
   while(ptr!=NULL){
      if(key == ptr->data){
         return ptr;
      }
      ptr=ptr->next;
   }
   printf("\nnot found\n");
   return NULL;

}

//fn to insert a node
void insert(NODE *ptr,int index,int x){
   NODE *new;
   int i;
   if(index<0||index>Ncount(ptr)){
      printf("invalid position\n");
   }
   new=(NODE*)malloc(sizeof(NODE));
   new->data=x;
   if(index==0){
      new->next=head;
      head=new;
   }
   else{
      for(i=0;i<index-1;i++){
         ptr=ptr->next;
      }
      new->next=ptr->next;
      ptr->next=new;
   }
}

//fn to delete a node
int DeleteNode(NODE*ptr,int index){
   NODE *q=NULL;
   int x=-1;
```

```
    if(index<0||index>Ncount(ptr)){
        return -1;
    }
    if(index==1){
        x=head->data;
        head=head->next;
        free(ptr);
        return x;
    }
    else{
        ptr=head;
        for(int i=0;i<index-1 && ptr;i++){
            q=ptr;
            ptr=ptr->next;
        }
        q->next=ptr->next;
        x=ptr->data;
        free(ptr);
        return x;

    }
}
```

-_-

```
//Checking for a loop
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

typedef struct node{
    int data;
    struct node *next;
}NODE;

NODE *head=NULL;// defining globally bcoz any change in the node using fn will no be reflected in the main;

void display(NODE*);
NODE *Lsearch(NODE*,int);
void insert(NODE*,int,int);
void create(int *,int);
int DeleteNode(NODE*,int);
bool isLoop(NODE*);

int main(){

    int A[]={10,20,30,40,50};
    create(A,5);
    printf("without loop\n");
    display(head);
    printf("\n");
    // creating a loop
    NODE *t1,*t2;
    t1=head->next->next;
    t2=head->next->next->next->next;
    t2->next=t1;
```

```
   bool loop=isLoop(head);
   if(loop==true){
      printf("loop is present\n");
   }
   else{
      printf("loop is not present\n");
   }
   return 0;
}
//fn to create an array to linklist
void create(int *arr,int n){
   NODE *t,*last;
   head=(NODE*)malloc(sizeof(NODE));
   head->data=arr[0];
   head->next=NULL;
   last=head;
   for(int i=1;i<n;i++){
      t=(NODE*)malloc(sizeof(NODE));
      t->data=arr[i];
      t->next=NULL;
      last->next=t;
      last=t;

   }
}
//fn to count number of elemnts in the list
int Ncount(NODE *ptr){
   int count=0;
   while(ptr!=NULL){
      count++;
      ptr=ptr->next;
   }
   return count;
}
//fn to display the linked list
void display(NODE *ptr){
   if(ptr != NULL){
      printf("%d->",ptr->data);
      display(ptr->next);
   }
}

//fn to search for a particular element
NODE *Lsearch(NODE *ptr,int key){
   while(ptr!=NULL){
      if(key == ptr->data){
         return ptr;
      }
      ptr=ptr->next;
   }
   printf("\nnot found\n");
   return NULL;

}

//fn to insert a node
```

```
void insert(NODE *ptr,int index,int x){
   NODE *new;
   int i;
   if(index<0||index>Ncount(ptr)){
      printf("invalid position\n");
   }
   new=(NODE*)malloc(sizeof(NODE));
   new->data=x;
   if(index==0){
      new->next=head;
      head=new;
   }
   else{
      for(i=0;i<index-1;i++){
         ptr=ptr->next;
      }
      new->next=ptr->next;
      ptr->next=new;
   }
}

//fn to delete a node
int DeleteNode(NODE*ptr,int index){
   NODE *q=NULL;
   int x=-1;
   if(index<0||index>Ncount(ptr)){
      return -1;
   }
   if(index==1){
      x=head->data;
      head=head->next;
      free(ptr);
      return x;
   }
   else{
      ptr=head;
      for(int i=0;i<index-1 && ptr;i++){
         q=ptr;
         ptr=ptr->next;
      }
      q->next=ptr->next;
      x=ptr->data;
      free(ptr);
      return x;

   }
}
bool isLoop(NODE *head){
   NODE *p,*q;
   p=q=head;
   do{
      p=p->next;
      q=q->next;
      q=(q!=NULL)?q->next:NULL;

   }while(p&&q&&p!=q);
```

```
    if(p==q){
        return true;
    }
    else{
        return false;
    }
}
```

-_-

```
//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>
#include<stdbool.h>

typedef struct node{
    int data;
    struct node *next;
}NODE;

NODE *head=NULL;// defining globally bcoz any change in the node using fn will no be reflected in the main;
NODE *first=NULL;
NODE *second=NULL;

void display(NODE*);
void create(int *,int);
int Ncount(NODE*);
// void listConcat(NODE*,NODE*);
NODE *listConcat(NODE*,NODE*);

int main(){

    int arrfirst[]={1,2,3,4};
    create(arrfirst,4);
    printf("first list\n");
    display(head);
    printf("\n");
    first=head;
    int arrsecond[]={7,8,9};
    create(arrsecond,3);
    printf("second list\n");
    display(head);
    second=head;
    printf("\n");

    // listConcat(first,second);
    // display(first);
    NODE *concat=listConcat(first,second);
    display(concat);
    printf("\n");

    return 0;
}
//fn to create an array to linklist
void create(int *arr,int n){
    NODE *t,*last;
```

```
    head=(NODE*)malloc(sizeof(NODE));
    head->data=arr[0];
    head->next=NULL;
    last=head;
    for(int i=1;i<n;i++){
        t=(NODE*)malloc(sizeof(NODE));
        t->data=arr[i];
        t->next=NULL;
        last->next=t;
        last=t;

    }
}
//fn to count number of elemnts in the list
int Ncount(NODE *ptr){
    int count=0;
    while(ptr!=NULL){
        count++;
        ptr=ptr->next;
    }
    return count;
}
//fn to display the linked list
void display(NODE *ptr){
    if(ptr != NULL){
        printf("%d->",ptr->data);
        display(ptr->next);
    }
}

// void listConcat(NODE *first,NODE *second){
//     NODE *p;
//     p=first;
//     while(p->next!=NULL){
//         p=p->next;
//     }
//     p->next=second;
//     second=NULL;

// }

NODE *listConcat(NODE *first,NODE *second){
    NODE *p;
    p=first;
    while(p->next!=NULL){
        p=p->next;
    }
    p->next=second;
    second=NULL;
    return first;
}
```

-_-
_-

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
 ********************************************************************************************************
/*
Automotive Manufacturing Plant Management System
*/
#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;

//required
```

```
AssemblyLine *create(int);
void display(AssemblyLine*);
AssemblyLine *Lsearch(AssemblyLine*,int);
void insert(AssemblyLine*,int);
int Delete(AssemblyLine*,int);
void updateStatus(AssemblyLine*);

//add on
int Ncount(AssemblyLine*);

int main(){
  int op;
  int noLine;
  int addindex;
  int delindex;
  int lineID;
  AssemblyLine *srchValue;
  printf("1-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\n");
  while(1){
    printf("enter operation:");
    scanf("%d",&op);
    switch(op){
      case 1:

        printf("enter number of assembly lines to be entered:");
        scanf("%d",&noLine);
        create(noLine);
        break;

      case 2:

        printf("enter the index position for new data:");
        scanf("%d",&addindex);
        printf("insert new assembly line\n");
        insert(head,addindex);
        break;

      case 3:

        printf("enter the node pos of data to be deleted:");
        scanf("%d",&delindex);
        Delete(head,delindex);
        break;

      case 4:
        printf("enter the line id to be searched:");
        scanf("%d",&lineID);
        srchValue=Lsearch(head,lineID);
        if (srchValue != NULL) {
          printf("Line ID: %d\n", srchValue->lineID);
          printf("Line Name: %s\n", srchValue->lineName);
          printf("Capacity: %d\n", srchValue->capacity);
          printf("Status: %s\n", srchValue->status);
        }
        break;
```

```
        case 5:
          updateStatus(head);
          break;
        case 6:
          display(head);
          break;

        case 7:
          printf("exiting system\n");
          free(head);

          break;

        // default:

      }
    }
    return 0;
}

//to create
AssemblyLine *create(int no){
    AssemblyLine *newNode,*temp;
    newNode=temp=NULL;

    for(int i=0;i<no;i++){
      newNode=(AssemblyLine*)malloc(sizeof(AssemblyLine));
      if(newNode==NULL){
        printf("memory allocation failed\n");
        exit(1);
      }
      printf("%d line details\n",i+1);
      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;
      if(head==NULL){
        head=newNode;
      }
      else{
        temp->next=newNode;
      }
      temp=newNode;

    }

    return head;
}

//to count
```

```
int Ncount(AssemblyLine *ptr){
   int count=0;
   while(ptr!=NULL){
      count++;
      ptr=ptr->next;
   }
   return count;
}

void insert(AssemblyLine *ptr,int index){
   AssemblyLine *new;
   int i;
   if(index<0||index>Ncount(ptr)){
      printf("invalid position\n");
      return;
   }
   new=(AssemblyLine*)malloc(sizeof(AssemblyLine));

   printf("line id:");
   scanf("%d",&new->lineID);
   printf("line name:");
   scanf(" %[^\n]",new->lineName);
   printf("capacity:");
   scanf("%d",&new->capacity);
   printf("status:");
   scanf(" %[^\n]",new->status);
   if(index==0){
      new->next=head;
      head=new;
   }
   else{
      for(i=0;i<index-1;i++){
         ptr=ptr->next;
      }
      new->next=ptr->next;
      ptr->next=new;
   }
}

//display
void display(AssemblyLine *ptr){
   if(ptr==NULL){
      printf("no data entered\n");
   }
   while(ptr!=NULL){    //Base condition
      printf("line %d:",ptr->lineID);
      printf("%s,",ptr->lineName);
      printf("capacity:%d,",ptr->capacity);
      printf("Status:%s\n",ptr->status);
      ptr=ptr->next;
   }


}

//delete
```

```
int Delete(AssemblyLine *ptr,int index){
   AssemblyLine *q=NULL,*temp;

   if(index<0||index>Ncount(ptr)){
      return -1;
   }
   if(index==1){
      temp=head;
      head=head->next;
      free(ptr);
      return 0;
   }
   else{
      ptr=head;
      for(int i=0;i<index-1 && ptr;i++){
         q=ptr;
         ptr=ptr->next;
      }
      q->next=ptr->next;
      temp=ptr;
      free(ptr);
      return 0;

   }
}

//search
AssemblyLine *Lsearch(AssemblyLine *ptr,int key){
   while(ptr!=NULL){
      if(key == ptr->lineID){
         printf("search successfull\n");
         return ptr;
      }
      ptr=ptr->next;
   }
   printf("\nnot found\n");
   return NULL;

}

//update status
void updateStatus(AssemblyLine *ptr){
   AssemblyLine *srchValue;
   char status[20];
   int lineID;
   printf("enter the line id to be searched:");
   scanf("%d",&lineID);
   srchValue=Lsearch(head,lineID);
   if(srchValue!=NULL){
      printf("enter the status:");
      scanf(" %[^\n]",status);
      strcpy(srchValue->status,status);
   }


}
```

-_-

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

typedef struct stack{
   int size;
   int top;
   int *S;
}STACK;

void create(STACK *);
void push(STACK*,int);
void display(STACK*);
int pop(STACK*);
int peek(STACK*,int);

int main(){
   STACK st;
   int elementPoped;
   int elementPeeked;
   create(&st);
   push(&st,10);
   push(&st,20);
   push(&st,30);
   push(&st,40);
   push(&st,50);
   push(&st,60);
   printf("before popping\n");
   display(&st);

   // elementPoped=pop(&st);
   // printf("popped element is %d\n",elementPoped);

   // printf("after popping\n");
   // display(&st);7

   elementPeeked=peek(&st,0);
   printf("element peeked is %d\n",elementPeeked);
   elementPeeked=peek(&st,4);
   printf("element peeked is %d\n",elementPeeked);
   elementPeeked=peek(&st,5);
   printf("element peeked is %d\n",elementPeeked);

   return 0;
}

void create(STACK *st){
   printf("enter size:");
   scanf("%d",&st->size);
   st->top=-1;
   st->S=(int *)malloc(st->size*sizeof(int));

}
```

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

}

void display(STACK *st){
   int temp=st->top; //temp is used bcoz since directly passing the memory address the fn changes the index;

   while(temp>=0){
      printf("%d\n",st->S[temp]);
      temp--;
   }
}
//or
/*
void display(STACK *st){
   for(int i=st->top;i>=0;i--){
      printf("%d\n",st->S[i]);
   }
}
*/
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;
}

int peek(STACK *st,int index){
   int data=-1;
   int temp=st->top;
   if(temp<0){
      printf("empty stack\n");
   }
   while(index<=temp){
      if(temp==index){
         data=st->S[temp];
         return data;
      }
      else{
         temp--;
      }
   }
}
```

-_-

```c
//stack_linklist_CREATE_PUSH_POP_PEEK
#include<stdio.h>
#include<stdlib.h>

typedef struct node{
  int data;
  struct node *next;
}STACK;

STACK *top=NULL;

void push(int);
void display(STACK*);
int pop();

int main(){
  int elementPOP;
  push(10);
  push(20);
  push(30);
  push(40);
  printf("before popping\n");
  display(top);
  elementPOP=pop();
  printf("\n");
  printf("element popped is %d\n",elementPOP);
  printf("after popping\n");
  display(top);
  return 0;
}

void push( int data){
  STACK *newNode=(STACK*)malloc(sizeof(STACK));
  if(newNode==NULL){
    printf("stack overflow\n");
  }
  else{
    newNode->data=data;
    newNode->next=top;
    top=newNode;
  }
}

void display(STACK *top){
  while(top!=NULL){
    printf("%d->",top->data);
    top=top->next;
  }

}

int pop(){
  STACK *p;
```

```
    int x=-1;
    if(top==NULL){
        printf("stack is empty");

    }
    else{
        p=top;
        top=top->next;
        x=p->data;
        free(p);
    }
    return x;
}
```

-_-
_-