1-linked list operations

```
#include<stdio.h>

#include<stdlib.h>


struct Node {

   int data;

   struct Node *next;

} *head = NULL;


void RDisplay(struct Node*);

void display(struct Node*);

int NCount(struct Node*);

int RCount(struct Node*);

int Nsum(struct Node*);

int RSum(struct Node*);

int MaxElement(struct Node*);

int RMaxElement(struct Node*);

int Lsearch(struct Node*, int);

void insert(struct Node*, int, int);

void create(int A[], int );

int removecycle(struct Node *);

int deletenode(struct Node * ptr, int index);


int main() {

   struct Node *t1, *t2;

   int A[] = {10, 20, 30, 40, 50};

   create(A, 5);
```

```
int temp;


t1=head->next->next;

t2=head->next->next->next->next;

t2->next=t1;


if(removecycle(head))

{

   printf("Cycle detected and removed");

}

else

{

   printf("no cycle");

}

printf("\n");

display(head);

printf("\n");

RDisplay(head);

printf("\n");


int count = RCount(head);

printf("Total number of nodes = %d\n", count);


int sum = Nsum(head);

printf("Total sum = %d\n", sum);


int r_sum = RSum(head);

printf("Recursive sum = %d\n", r_sum);
```

```
int max = MaxElement(head);
printf("Maximum element = %d\n", max);

int r_max = RMaxElement(head);
printf("Recursive maximum element = %d\n", r_max);

temp = Lsearch(head, 30);
if (temp) {
   printf("Element found: %d\n", temp);
}
else {
   printf("Element not found\n");
}

insert(head, 0, 5);
RDisplay(head);
printf("\n");

insert(head, 2, 15);
RDisplay(head);
printf("\n");

int a=deletenode(head,2);
if(a==-1)
{
   printf("no data found");
}
```

```
    else
    {
        printf("element remeoved is %d",a);
    }
    printf("\n");
    RDisplay(head);



    return 0;
}

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

void RDisplay(struct Node *p) {
    if (p != NULL) {
        printf("%d->", p->data);
        RDisplay(p->next);
    }
}

int NCount(struct Node *p) {
    int c = 0;
    while (p!=NULL) {
        c++;
        p = p->next;
    }
    return c;
}

int RCount(struct Node *p) {
    if (p == NULL) {
        return 0;
    }
    else {
        return RCount(p->next) + 1;
    }
}

int Nsum(struct Node *p) {
    int s = 0;
    while (p != NULL) {
        s += p->data;
        p = p->next;
    }
    return s;
}

int RSum(struct Node *p) {
    if (p == NULL) {
        return 0;
    } else {
        return RSum(p->next) + p->data;
    }
}

int MaxElement(struct Node *p) {
    int m = p->data;;
    while (p != NULL) {
        if (p->data > m) {
            m = p->data;
        }
        p = p->next;
    }
    return m;
}

int RMaxElement(struct Node *p) {
    int x = 0;
    if (p == NULL) {
        return 0;
    } else
    {
        x = RMaxElement(p->next);
        return (x > p->data) ? x : p->data;
    }
}

int Lsearch(struct Node *p, int key) {
    while (p != NULL) {
        if (key == p->data) {
            return p->data;
        }
        p = p->next;
    }
    return 0;
}

void insert(struct Node *p, int index, int x) {
    struct Node *t;
    int i;

    if (index < 0 || index > NCount(p)) {
        printf("Invalid position\n");
        return ;
    }

    t = (struct Node*)malloc(sizeof(struct 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;
    }
}

void create(int A[], int n) {
    struct Node *t, *last;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->data = A[0];
    head->next = NULL;
    last = head;

    for (int i = 1; i < n; i++) {
        t = (struct Node*)malloc(sizeof(struct Node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}


int removecycle(struct Node * ptr)
{
    struct Node * p1, *p2;
    p1=ptr;
```

```
  p2=ptr;
  while(ptr!=NULL)
  {
    p1=p1->next;
    p2=p2->next->next;
    if(p1==p2)
    {
      p1=ptr;
      struct Node* prev=NULL;
      while(p1!=p2)
      {
        prev=p2;
        p1=p1->next;
        p2=p2->next;
      }

      prev->next=NULL;
      return 1;
    }
  }
}


int deletenode(struct Node * ptr, int index)
{

  struct Node * q=NULL;
  int x=-1;
  if(index<1 || index> NCount(ptr))
  {
    return -1;
  }
  if (index==1)
  {
    x=head->data;
    head=head->next;
    return x;
  }
  else
  {
    ptr=head;
    for(int i=0;i<index-1;i++)
    {
      q=ptr;
      ptr=ptr->next;
    }
    q->next=ptr->next;
    x=ptr->data;
    return x;


  }
}
```

2- concatenation of linked list

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

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



void display(struct Node*);

struct Node * create(int *, int n);
void concatinate(struct Node * first, struct Node * second);



int main()
{
   int A[] = {10, 20, 30, 40};
   struct Node *first= create(A, 4);

   display(first);
   printf("\n");

   int B[] = {50, 60, 70, 80, 90};
```

```
    struct Node *second= create(B, 5);

    display(second);

    printf("\n");


    concatinate(first,second);

    display(first);


    return 0;

}




void display(struct Node *p) {

    while (p != NULL) {

        printf("%d->", p->data);

        p = p->next;

    }

}




struct Node * create(int C[], int n)

{

    struct Node *t, *last, *head;

    head = (struct Node*)malloc(sizeof(struct Node));

    head->data = C[0];

    head->next = NULL;

    last = head;
    for (int i = 1; i < n; i++)
    {
        t = (struct Node*)malloc(sizeof(struct Node));
        t->data = C[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
    return head;
}

void concatinate(struct Node * first, struct Node * second)
{
    while(first->next!=NULL)
    {
        first=first->next;
    }
    first->next=second;
}
```

3-stack using arrays

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

struct Stack
{
    int size;

    int top;

    int *S;

};


void create(struct Stack *);

void push(struct Stack *,int);

void display(struct Stack *);

int pop(struct Stack*);

int peck(struct Stack* , int);


int main()

{

    struct Stack *st = (struct Stack *)malloc(sizeof(struct Stack));

    create(st);

    push(st,10);

    push(st,20);

    push(st,30);

    push(st,40);

    push(st,50);

    display(st);

    printf("\n");


    int x=pop(st);

    if(x==-1)

    {

        printf("stack is empty");
    }
    else
    {
        printf("Element poped is %d",x);
    }



    printf("\n");
    display(st);
    printf("\n");
    int y=peck(st,1);
    if(y==-1)
    {
        printf("no element");
    }
    else
    {
        printf("Element is %d",y);
    }




    return 0;
}

void create(struct Stack *st)
{
    printf("Enter the size of array: ");
    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");

    }

    else

    {

        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]);


    }
```

```
}



int pop(struct Stack *st)
{
   int x=-1;
   if(st->top==-1)
   {
      return x;
   }
   else
   {
      x=st->S[st->top];
      st->top--;

   }
   return x;
}

int peck(struct Stack*st  , int index)
{

   for(int i=st->top;i>=0;i--)
   {
      if(i==index)
      {
         return st->S[i];
      }
```

```
    }
    return -1;
}
```

4- stack using linked list

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

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

STACK *top = NULL;

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

int main() {
    int elementPOP;

    push(10);
    push(20);
    push(30);
    push(40);
    printf("Before popping:\n");
    display(top);

    elementPOP = pop();
    printf("\nElement 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;
    }
    printf("NULL\n");
}
int pop() {
    STACK *p;
    int x = -1;

    if (top == NULL) {
        printf("Stack is empty\n");
    } else {
        p = top;
        top = top->next;
        x = p->data;
        free(p);
    }

    return x;
}
```

5- /*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

```
#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 *head = NULL;

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

int main() {
   int op;
   int noLine, addindex, delindex, lineID;
   AssemblyLine *srchValue;
```

```
while (1) {
  printf("Menu:\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 operation: ");
  scanf("%d", &op);

  switch (op) {
    case 1:
      head=create();
      break;

    case 2:
      printf("Enter the index position for new data: ");
      scanf("%d", &addindex);
      insert(head, addindex);
      break;

    case 3:
      printf("Enter the node position of data to be deleted: ");
      scanf("%d", &delindex);
      if (Delete(head, delindex) == -1) {
        printf("Invalid position for deletion.\n");
      }
      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");
      return 0;
```

```
            default:
                printf("Invalid operation. Please try again.\n");
        }
    }
    return 0;
}

AssemblyLine *create()
{
    AssemblyLine *newNode= NULL;

    newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("Enter details for 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;
    head = newNode;
    return head;
}

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

void insert(AssemblyLine *ptr, int index) {
    AssemblyLine *new= NULL;
    if (index < 0 || index > Ncount(ptr)) {
        printf("Invalid position\n");
        return;
    }
    new = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("Enter details for the new assembly line:\n");
    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 (int i = 0; i < index - 1; i++) {
            ptr = ptr->next;
        }
        new->next = ptr->next;
        ptr->next = new;
    }
}

void display(AssemblyLine *ptr) {
    if (ptr == NULL) {
        printf("No data entered\n");
        return;
    }
    while (ptr != NULL) {
        printf("Line %d: %s, Capacity: %d, Status: %s\n", ptr->lineID, ptr->lineName, ptr-
>capacity, ptr->status);
        ptr = ptr->next;
    }
}

int Delete(AssemblyLine *ptr, int index)
{
    AssemblyLine *q = NULL, *temp;
    if (index < 1 || index > Ncount(ptr)) {
        return -1;
    }
    if (index == 1)
    {
        temp = head;
        head = head->next;
        free(temp);
        return 0;
    }
    for (int i = 0; i < index - 1 ; i++)
    {
        q = ptr;
        ptr = ptr->next;
    }

    q->next = ptr->next;
    free(ptr);

    return 0;
}

AssemblyLine *Lsearch(AssemblyLine *ptr, int key) {
    while (ptr != NULL) {
        if (key == ptr->lineID) {
            printf("Search successful\n");
            return ptr;
        }
        ptr = ptr->next;
    }
    printf("Not found\n");
    return NULL;
}

void updateStatus(AssemblyLine *ptr) {
    AssemblyLine *srchValue;
    char status[20];
    int lineID;

    printf("Enter the line ID to update: ");
    scanf("%d", &lineID);
    srchValue = Lsearch(ptr, lineID);
    if (srchValue != NULL) {
        printf("Enter the new status: ");
        scanf(" %[^\n]", status);
        strcpy(srchValue->status, status);
    }
}
```