**ASSIGNMENT**

```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 index);
int isloop(Node*);
int main() {
    Node *t1,*t2;
    int A[] = {10, 20, 30, 40, 50};
    int delVar=0;
    create(A, 5);
    //forming a loop
    t1=head->next->next;
    t2=head->next->next->next->next;
    t2->next=t1;
    int loop=isloop(head);
    if (isloop(head)) {
        printf("Loop is present.\n");
    } else {
        printf("Loop is not present.\n");
    }
    // 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,6);
```

```
    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);
    }
}
```

```c
//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;
            x=p->data;
            free(p);
            return x;

    }

}

int 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 1;
    }
    else
    {
        return 0;
    }
}
```

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 linked list and display.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
// Function declarations
struct Node* create(int A[], int n);
void display(struct Node* head);
void concatenate(struct Node *first, struct Node *second);
int main() {
    // Create two linked lists
    int A[] = {1, 2, 3, 4};
    int B[] = {7, 8, 9};
    struct Node *first = create(A, 4);
    struct Node *second = create(B, 3);
    // Concatenate the lists
    concatenate(first, second);
    // Display the concatenated list
    printf("Concatenated Linked List: ");
    display(first);
    return 0;
}
// Create a linked list from an array
struct Node* create(int A[], int n) {
    struct Node *head, *last, *p;
    head = (struct Node *)malloc(sizeof(struct Node));
    head->data = A[0];
    head->next = NULL;
    last = head;
    for (int i = 1; i < n; i++) {
        p = (struct Node *)malloc(sizeof(struct Node));
        p->data = A[i];
        p->next = NULL;
        last->next = p;
        last = p;
    }
    return head;
}
// Concatenate two linked lists
void concatenate(struct Node *first, struct Node *second) {
    struct Node *p = first;
    while (p->next != NULL) {
        p = p->next;
    }
    p->next = second;
}
// Display the linked list
void display(struct Node *head) {
```

```c
    struct Node *p = head;
    while (p != NULL) {
        printf("%d->", p->data);
        p = p->next;
    }
    printf("NULL\n");
}
```

```c
// 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
```

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

// Structure for a linked list node
typedef struct Node {
    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 Node *next;        // Pointer to the next node
} Assembly_Line;
Assembly_Line* head = NULL;
void create(int n);
void insert();
void delete_line();
void search();
void update_status();
void display();
```

```c
int main() {
    int option;
    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", &option);
        switch (option) {
            case 1:
                printf("Enter the Number of Assembly Lines: ");
                int n;
                scanf("%d", &n);
                create(n);
                break;
            case 2:
                insert();
                break;
            case 3:
                delete_line();
                break;
            case 4:
                search();
                break;
            case 5:
                update_status();
                break;
            case 6:
                display();
                break;
            case 7:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Enter a Valid Option.\n");
        }
    }
}
void create(int n) {
    for (int i = 0; i < n; i++) {
        Assembly_Line* new_node = (Assembly_Line*)malloc(sizeof(Assembly_Line));
        printf("Enter details for line %d:\n", i + 1);
        printf("Line ID: ");
        scanf("%d", &new_node->lineID);
        printf("Line Name: ");
        scanf("%s", new_node->lineName);
        printf("Capacity: ");
        scanf("%d", &new_node->capacity);
        printf("Status: ");
        scanf("%s", new_node->status);
        new_node->next = head;
        head = new_node;
    }
}
void insert() {
    Assembly_Line* new_node = (Assembly_Line*)malloc(sizeof(Assembly_Line));
```

```
    printf("Enter details for the new assembly line:\n");
    printf("Line ID: ");
    scanf("%d", &new_node->lineID);
    printf("Line Name: ");
    scanf("%s", new_node->lineName);
    printf("Capacity: ");
    scanf("%d", &new_node->capacity);
    printf("Status: ");
    scanf("%s", new_node->status);
    new_node->next = head;
    head = new_node;
    printf("New assembly line added successfully.\n");
}
void delete_line() {
    int id;
    printf("Enter the Line ID to delete: ");
    scanf("%d", &id);
    Assembly_Line* temp = head;
    Assembly_Line* 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 deleted successfully.\n");
}
void search() {
    int id;
    printf("Enter the Line ID to search: ");
    scanf("%d", &id);
    Assembly_Line* temp = head;
    while (temp != NULL) {
        if (temp->lineID == id) {
            printf("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);
}
void update_status() {
    int id;
    char new_status[20];
    printf("Enter the Line ID to update status: ");
    scanf("%d", &id);
    Assembly_Line* temp = head;
    while (temp != NULL) {
        if (temp->lineID == id) {
            printf("Enter new status: ");
```

```
            scanf("%s", new_status);
            strcpy(temp->status, new_status);
            printf("Status updated successfully.\n");
            return;
        }
        temp = temp->next;
    }
    printf("Assembly line with ID %d not found.\n", id);
}
void display() {
    Assembly_Line* temp = head;
    if (temp == NULL) {
        printf("No assembly lines found.\n");
        return;
    }
    printf("Assembly Lines:\n");
    while (temp != NULL) {
        printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
               temp->lineID, temp->lineName, temp->capacity, temp->status);
        temp = temp->next;
    }
}
```