DAY_22_ASSIGNMENT

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

struct Node{
    int data;
    struct Node*next;
};
void add(struct Node**head,int n);
void display(struct Node*head);
void deletefrond(struct Node**head);
void deletepos(struct Node**head,int p);
void deletefromback (struct Node**head);

int main(){
    struct Node *head;
    head=NULL;
    int n,da;
    printf("enter the number of elements");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&da);
        add(&head,da);

    }
    display(head);
    deletefrond(&head);
    deletefromback(&head);
    deletepos(&head,2);
display(head);
}
void add(struct Node**head,int n){
    struct Node*temp;

    struct Node*new;
    new= (struct Node*)malloc(sizeof(struct Node));
    new->data=n;
    new->next=NULL;
    if(*head==NULL){
        *head=new;
    }
    else{
        temp=*head;
```

```
        while (temp->next!=NULL)
        {
          temp=temp->next;
        }
        temp->next=new;

    }
}
void display(struct Node*head){
    struct Node *temp;
    temp=head;
    while(temp!=NULL){
        printf("%d",temp->data);
        temp=temp->next;
    }
}
void deletefrond(struct Node**head){
    struct Node*temp;
    temp=*head;
   *head = (*head)->next;
   free(temp);
}
void deletefromback (struct Node**head){
    struct Node*temp;
    struct Node*prev;
    prev=NULL;
    temp=*head;
    while(temp->next!=NULL){
        prev=temp;
        temp=temp->next;
    }
    prev->next=NULL;
    free(temp);

}
void deletepos(struct Node**head,int p){
    struct Node*temp;
    struct Node*prev;
    prev=NULL;
    temp=*head;
    for(int i=0;i<p;i++){
        prev=temp;
        temp=temp->next;
    }
```

```
    prev->next=temp->next;
    free(temp);


}
```

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

struct Node{
    int data;
    struct Node*next;
};
void add(struct Node**head1,int m);
void display(struct Node*head1);
void add1(struct Node**head2,int q);
void concatitante(struct Node*head1,struct Node*head2);


int main(){
    struct Node *head1;
    head1=NULL;
    struct Node*head2;
    head2=NULL;
    int n,m;
    printf("enter the numbers");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&m);
        add(&head1,m);

    }
    display(head1);

    int p,q;
```

```
    printf("enter the numbers");
    scanf("%d",&p);
    for(int i=0;i<p;i++){
        scanf("%d",&q);
        add(&head2,q);

    }
    display(head2);
   printf("\n");
concatitante(head1,head2);
display(head1);

}
void add(struct Node**head,int m){
    struct Node*new=(struct Node*)malloc(sizeof(struct Node));
    struct Node*temp;
    new->data=m;
    new->next=NULL;
    if(*head==NULL){
        *head=new;
    }
    else{
        temp = *head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=new;
    }

}
void add1(struct Node**head,int m){
    struct Node*new=(struct Node*)malloc(sizeof(struct Node));
    struct Node*temp;
    new->data=m;
    new->next=NULL;
    if(*head==NULL){
        *head=new;
    }
    else{
        temp = *head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=new;
```

```
    }

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

}
void concatitante(struct Node*head1,struct Node*head2){
    struct Node*temp;
    temp=head1;
    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=head2;
}
```

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

struct Stack{
    int size;
    int top;
```

```
    int *s;

};
//function prototypes
void create(struct Stack*);
void push(struct Stack*,int);
void display(struct Stack*);
int pop(struct Stack*);
void peek(struct Stack *st);
int peeks(struct Stack*st,int pos);


int main(){
    struct Stack st;
    int emp;
    int emps;

    create(&st);
    push(&st,10);
    push(&st,20);
    push(&st,30);
    push(&st,40);
    display(&st);
    emp=pop(&st);
    printf("the element poped is %d\n",emp);
    display(&st);
    peek(&st);
    int ps=peeks(&st,1);
    printf("the value is %d",ps);

    return 0;
}
void create(struct Stack*st){
    printf("enter size ");
    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");
    }
```

```
    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]);
         printf("\n");
    }

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

// int peek(struct Stack *st){
//     if(st->top==-1){
//         printf("Stack is full");
//         return -1;
//     }
//     else{
//         return st->s[st->top];
//     }
// }

void peek(struct Stack *st){
    int peek;
    printf("enter the index of the peek");
    scanf("%d",&peek);
    for(int i=st->top;i>=0;i--){
        if(peek==i){
            printf("the value is %d",st->s[i]);
        }
```

```
    }
}
int peeks(struct Stack*st,int pos){
    int x=-1;
    if(st->top-pos+1<0){
        printf("invalid position");
    }
    else{
        x=st->s[st->size-(st->top-pos)];
        return x;
    }
    return x;
}
// int peek(struct stack st, int position) {
//     if(st.top - position + 1 < 0 || position > st.top + 1) {
//         printf("Invalid index\n");
//         return -1;
//     }
//     return st.S[st.top - position + 1];
// }
```

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

struct Node {
    int data;
    struct Node* next;
};
void pushoperation(struct Node**head,int n);
int popoperation(struct Node**head);
```

```
void display(struct Node *head);

int main(){
   struct Node*head=NULL;
   int n;
   int v;
   int op;
   while (1)
   {
      printf("\n1 for PUSH\n2 for POP\n Enter your choice");
      scanf("%d",&op);
      switch (op)
      {
      case 1:
            printf("Enter the value to be operated");
            scanf("%d",&n);
            pushoperation(&head,n);

         break;
      case 2:
            v=popoperation(&head);
            printf("the poped value is %d",v);
            break;
      case 3:
            display(head);
            break;

      default:
         break;
      }
   }

}
void pushoperation(struct Node**head,int n){
   struct Node*newnode;
   newnode=(struct Node*)malloc(sizeof(struct Node));
   if(newnode==NULL){
      printf("there in no memory allaocated");
   }
   else{
      newnode->data=n;
      newnode->next=*head;
      *head=newnode;
   }
```

```
}
int popoperation(struct Node**head){
    struct Node*temp;
    int x;
    if(*head==NULL){
        printf("stack is empty");
    }
    else{
        temp=*head;
        *head=(*head)->next;
        x=temp->data;
        free(temp);
        return x;

    }
}

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


}
```

// 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
```

```
 // Linked List Node Structure in C
#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;
void add(AssemblyLine**head,int n);
void insert(AssemblyLine**head,int p);
void display(AssemblyLine *head) ;
AssemblyLine*search(AssemblyLine*head,int n);
void updateStatus(AssemblyLine *head, int lineID,const char *newStatus) ;
void deleteByLineID(AssemblyLine** head, int lineID);

 int main(){
    AssemblyLine *head=NULL;
    int choice,m,p,v;
    int id;
          char state[30];
    while (1)

    {
       printf("1.ADD\n 2.INSERT\n 3.DISPLAY\n 4.SEARCH\n 5.UPDATE\n");


       printf("enter the choice");
       scanf("%d",&choice);
       switch (choice)
       {
       case 1:
          printf("enter the number of assemble lines");
          scanf("%d",&m);
          add(&head,m);

          break;
       case 2:
          printf("enter the position to be inserted 1 for frond 0 for end");
          scanf("%d",&p);
          insert(&head,p);
```

```
        break;
        case 3:
            display(head);
            break;
          case 4:
            printf("enter the item to been searched");
            scanf("%d",&v);
             AssemblyLine* foundLine = search(head,v);
                if (foundLine != NULL) {
                    printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n", foundLine->lineID,
foundLine->lineName, foundLine->capacity, foundLine->status);
                } else {
                    printf("Assembly Line not found.\n");
                }

            break;

            case 5:

            printf("enter the id");
            scanf("%d",&id);
            printf("enter the status");
            scanf("%d",state);
            updateStatus(head,id,state);

            break;
            case 6:
            printf("enter the id");
            scanf("%d",&id);
            deleteByLineID(&head,id);
            break;


        default:
            break;
        }
    }

 }
 void add(AssemblyLine**head,int n){
    AssemblyLine *newNode, *temp;
    *head = NULL;
    for(int i=0;i<n;i++){
```

```
        newNode=(AssemblyLine*)malloc(sizeof(AssemblyLine));
        printf("line ID");
        scanf("%d",&newNode->lineID);
        printf("line name");
        scanf("%s",newNode->lineName);
        printf("capacity");
        scanf("%d",&newNode->capacity);
        printf("status");
        scanf("%s",newNode->status);
        newNode->next=NULL;
        if (*head==NULL)
        {
            *head = newNode;
        }
        else{
            temp=*head;
            while(temp->next!=NULL){
                temp=temp->next;
            }
            temp->next=newNode;
        }

    }
}
void insert(AssemblyLine**head,int p){
    AssemblyLine *newNode;
    AssemblyLine *temp;
    newNode=(AssemblyLine*)malloc(sizeof(AssemblyLine));

    temp=*head;

    printf("Enter Line ID: ");
    scanf("%d", &newNode->lineID);
    printf("Enter Line Name: ");
    scanf("%s", newNode->lineName);
    printf("Enter Capacity: ");
    scanf("%d", &newNode->capacity);
    printf("Enter Status: ");
    scanf("%s", newNode->status);
    newNode->next = NULL;
    if(p==1){
        newNode->next=*head;
        *head=newNode;
    }
```

```
    else if(p==0){
        if(*head==NULL){
            *head= newNode;
        }
        else{
            while(temp->next!=NULL){
                temp=temp->next;
            }

            temp->next=newNode;
        }
    }
    else{
        for(int i=0;i<p-1;i++){
            temp =temp->next;
        }
        if(temp!=NULL){
        newNode->next = temp->next;
        temp->next = newNode;
        }
        else{
            printf("invalid index");
        }
    }

 }

 void display(AssemblyLine *head) {
    AssemblyLine *temp = head;
       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;
    }
}
AssemblyLine* search(AssemblyLine*head,int n){
    AssemblyLine*temp;
    temp=head;
    while(temp!=NULL){
        if(temp->lineID==n){
            return temp;
        }
        temp=temp->next;
    }
```

```
}
void updateStatus(AssemblyLine *head, int lineID,  const char *newStatus) {
    AssemblyLine *temp = head;
    while (temp != NULL) {
        if (temp->lineID == lineID) {
            strcpy(temp->status, newStatus);
            printf("Status updated for Line ID %d.\n", lineID);
            return;
        }
        temp = temp->next;
    }
    printf("Assembly Line not found.\n");
}

void deleteByLineID(AssemblyLine** head, int lineID) {
    AssemblyLine* temp ;

    AssemblyLine* prev;
    temp=*head;
    prev=NULL;
    // from frond
    if (temp != NULL && temp->lineID == lineID) {
        *head = temp->next;
        free(temp);
        printf("Assembly Line with Line ID %d deleted.\n", lineID);
        return;
    }
    //deleting from any point
    while (temp != NULL && temp->lineID != lineID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Assembly Line with Line ID %d not found.\n", lineID);
        return;
    }
        prev->next = temp->next;
    free(temp);
    printf("Assembly Line with Line ID %d deleted.\n", lineID);
}
```