DAY 22

1. #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 main() {
   int A[] = {20, 30, 40, 60, 70};
   create(A, 5);

   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));
    int delvar=DeleteNode(head,3);
    printf("node deltetd =%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);
   }
}

//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;
    if(index<1 || index>nCount(p))
    {
        return -1;
    }
    if(index==1)//here it is index in linkedlist index starts from 1 earlier in insertion we are considering the position to insert hence 0 was used
    {
        x=head->data;//extracting firstnode data
        head=head->next;//head pointing to second node
        free(p);//since p is pointing to first node,deleting first node
        return x;
    }
    else
    {
        p=head;
        for(int i=0;i<index-1 && p;i++ )//index-1 ie jumps
        {
            q=p;//pointing q to p ie first node
            p=p->next;//moving p to the next node , p will be one step a head of q


        }
```

```
        q->next=p->next;
        x=p->data;
        free(p);
        return x;
    }

}
```

```
node deltetd =30
10 -> 20 -> 40 -> 50 -> 60 -> 70 ->
```

2.
```
#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[] = {10,20, 30, 40, 50};
   create(A, 5);
   //forming a loop
    t1=head->next->next;//pointing to 30
    t2=head->next->next->next->next;//pointing to 50
    t2->next=t1;//loop formed ie 50 pointing to 30
  // printf("Original list: ");
   //display(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));
    int delvar;
    delvar=DeleteNode(head,3);
    printf("node deltetd =%d \n",delvar);
    display1(head);
    delvar=DeleteNode(head,4);
    printf("node deltetd =%d \n",delvar);
    display1(head);

 */
    int loop;
    loop=isloop(head);
    printf("loop is %d \n",loop);
    return 0;
}

//function to display using recursion
void display(struct Node*p)
{
    while(p!=NULL)
    {
        printf("%d-->",p->data);
            p=p->next;

    }

}



//function to display using recursion but in reverse order
```

```
//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;
}
```

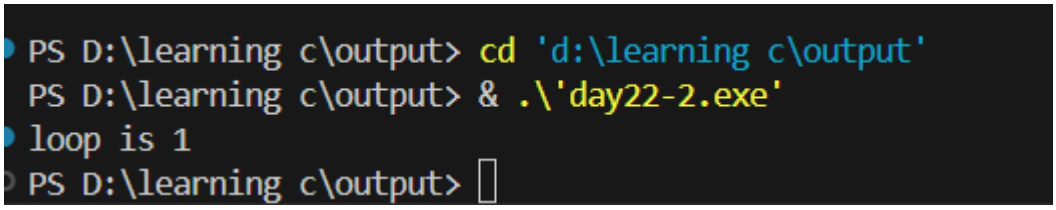
```
//function to find sum using recursion
int rSum(Node *p) {
   int sum = 0;
   if (!p) {
      return 0;
   } else {
      sum += p->data;
      return sum + rSum(p->next);
   }
}

//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 0;
}

//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;
```

```
    if(index<1 || index>nCount(p))
    {
        return -1;
    }
    if(index==1)//here it is index in linkedlist index starts from 1 earlier in insertion we are considering the position to insert hence 0 was used
    {
        x=head->data;//extracting firstnode data
        head=head->next;//head pointing to second node
        free(p);//since p is pointing to first node,deleting first node
        return x;
    }
    else
    {
        p=head;
        for(int i=0;i<index-1 && p;i++ )//index-1 ie jumps
        {
            q=p;//pointing q to p ie first node
            p=p->next;//moving p to the next node , p will be one step a head of q


        }
        q->next=p->next;
        x=p->data;
        free(p);
        return x;
    }

}
int isloop(Node*p)
{
```

```
    Node *q;
    q=head;
    do
    {
        p=p->next;
        q=q->next;
       if(q->next!=NULL)
       {
          q=q->next;


       }
       else
       {
          return 0;
       }
    } while (p != NULL && q != NULL && p != q);
    if(p==q)
    {
       return 1;
    }
    else
    {
       return 0;
    }


}
```



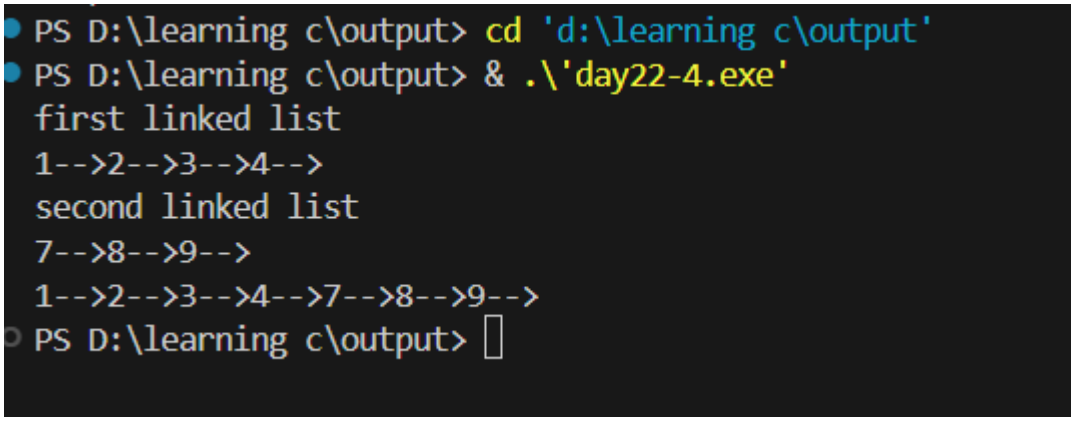
3. //craete two linkedlist and concatenate them

/*reate 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>

struct Node1

{

   int data;

   struct Node1*next;

};

struct Node2

{

   int data;

   struct Node2*next;

};

struct Node1*head;

struct Node2*head1;


void display(struct Node1*);

void display1(struct Node2*);

void concatenate(struct Node1*,struct Node2*);


int main()

{

   head=(struct Node1*)malloc(sizeof(struct Node1));

   struct Node1*sec=(struct Node1*)malloc(sizeof(struct Node1));

   struct Node1*third=(struct Node1*)malloc(sizeof(struct Node1));

   struct Node1*fourth=(struct Node1*)malloc(sizeof(struct Node1));
```

```
    head->data=1;
    head->next=sec;
    sec->data=2;
    sec->next=third;
    third->data=3;
    third->next=fourth;
    fourth->data=4;
    fourth->next=NULL;
    printf("first linked list \n");
    display(head);
    printf("\n");
    head1=(struct Node2*)malloc(sizeof(struct Node2));
    struct Node2*two=(struct Node2*)malloc(sizeof(struct Node2));
    struct Node2*three=(struct Node2*)malloc(sizeof(struct Node2));

    head1->data=7;
    head1->next=two;
    two->data=8;
    two->next=three;
    three->data=9;
    three->next=NULL;
    printf("second linked list \n");
    display1(head1);
    printf("\n");
    concatenate(head,head1);
    display(head);



}
```

```
void display(struct Node1*p)
{
   while(p!=NULL)
   {
      printf("%d-->",p->data);
         p=p->next;

   }

}
void display1(struct Node2*q)
{
   while(q!=NULL)
   {
      printf("%d-->",q->data);
         q=q->next;

   }

}
void concatenate(struct Node1*p,struct Node2*q)
{
   while(p->next!=NULL)
   {
         p=p->next;

   }
   p->next=(struct Node1*)q;
   q=NULL;
}
```



4. **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

```
#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;

// Function prototypes
void createLinkedList(AssemblyLine** head, int n);
void insertAssemblyLine(AssemblyLine** head, int position);
void deleteAssemblyLine(AssemblyLine** head, int lineID);
void searchAssemblyLine(AssemblyLine* head, int lineID);
void updateAssemblyLineStatus(AssemblyLine* head, int lineID);
void displayAssemblyLines(AssemblyLine* head);

// Main function
int main() {
    AssemblyLine* head = NULL;
    int choice, n, lineID, position;
```

```
do {
    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", &choice);

    switch (choice) {
    case 1:
        printf("Enter the number of assembly lines: ");
        scanf("%d", &n);
        createLinkedList(&head, n);
        break;
    case 2:
        printf("Enter the position to insert (0 for start, -1 for end): ");
        scanf("%d", &position);
        insertAssemblyLine(&head, position);
        break;
    case 3:
        printf("Enter the lineID of the assembly line to delete: ");
        scanf("%d", &lineID);
        deleteAssemblyLine(&head, lineID);
        break;
    case 4:
        printf("Enter the lineID to search: ");
```

```
            scanf("%d", &lineID);
            searchAssemblyLine(head, lineID);
            break;
        case 5:
            printf("Enter the lineID to update status: ");
            scanf("%d", &lineID);
            updateAssemblyLineStatus(head, lineID);
            break;
        case 6:
            displayAssemblyLines(head);
            break;
        case 7:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Try again.\n");
        }
    } while (choice != 7);

    return 0;
}

// Function to create a linked list
void createLinkedList(AssemblyLine** head, int n) {
    for (int i = 0; i < n; i++) {
        AssemblyLine* newLine = (AssemblyLine*)malloc(sizeof(AssemblyLine));
        printf("Enter details for line %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &newLine->lineID);
        printf("Name: ");
        scanf("%s", newLine->lineName);
```

```
        printf("Capacity: ");
        scanf("%d", &newLine->capacity);
        printf("Status: ");
        scanf("%s", newLine->status);
        newLine->next = *head;
        *head = newLine;
    }
}

// Function to insert a new assembly line
void insertAssemblyLine(AssemblyLine** head, int position) {
    AssemblyLine* newLine = (AssemblyLine*)malloc(sizeof(AssemblyLine));
    printf("Enter details for new line:\n");
    printf("ID: ");
    scanf("%d", &newLine->lineID);
    printf("Name: ");
    scanf("%s", newLine->lineName);
    printf("Capacity: ");
    scanf("%d", &newLine->capacity);
    printf("Status: ");
    scanf("%s", newLine->status);

    if (position == 0) {
        newLine->next = *head;
        *head = newLine;
    } else {
        AssemblyLine* temp = *head;
        int count = 0;
        while (temp != NULL && count < position - 1) {
            temp = temp->next;
            count++;
```

```
        }
        if (temp != NULL) {
            newLine->next = temp->next;
            temp->next = newLine;
        } else {
            printf("Invalid position!\n");
            free(newLine);
        }
    }
}

// Function to delete an assembly line
void deleteAssemblyLine(AssemblyLine** head, int lineID) {
    AssemblyLine* temp = *head;
    AssemblyLine* prev = NULL;

    while (temp != NULL && temp->lineID != lineID) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Assembly line with ID %d not found!\n", lineID);
        return;
    }
    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }
    free(temp);
    printf("Assembly line with ID %d deleted.\n", lineID);
```

```
}

// Function to search for an assembly line
void searchAssemblyLine(AssemblyLine* head, int lineID) {
    AssemblyLine* temp = head;
    while (temp != NULL) {
        if (temp->lineID == lineID) {
            printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
                   temp->lineID, temp->lineName, temp->capacity, temp->status);
            return;
        }
        temp = temp->next;
    }
    printf("Assembly line with ID %d not found!\n", lineID);
}

// Function to update assembly line status
void updateAssemblyLineStatus(AssemblyLine* head, int lineID) {
    AssemblyLine* temp = head;
    while (temp != NULL) {
        if (temp->lineID == lineID) {
            printf("Enter new status: ");
            scanf("%s", temp->status);
            printf("Status updated successfully.\n");
            return;
        }
        temp = temp->next;
    }
    printf("Assembly line with ID %d not found!\n", lineID);
}
```

```
// Function to display all assembly lines
void displayAssemblyLines(AssemblyLine* head) {
    if (head == NULL) {
        printf("No assembly lines found!\n");
        return;
    }
    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;
    }
}
```

```
PS D:\learning c\output> & .\'day22-6(5th full code ).exe'

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
Enter your choice: 1
Enter the number of assembly lines: 3
Enter details for line 1:
ID: 1
Name: a
Capacity: 8
Status: active
```

Enter details for line 2:

ID: 2

Name: b

Capacity: 6

Status: active

Enter details for line 3:

ID: 3

Name: c

Capacity: active

Status:

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

Enter your choice: 2

Enter the position to insert (0 for start, -1 for end): 0

Enter details for new line:

ID: 0

Name: d

Capacity: 7

Status: active

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

Enter your choice: 3

Enter the lineID of the assembly line to delete: 2

Assembly line with ID 2 deleted.

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

Enter your choice: 4

Enter the lineID to search: 0

Line ID: 0, Name: d, Capacity: 7, Status: active

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

Enter your choice: 5

Enter the lineID to update status: 1

Enter new status: under

Status updated successfully.

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

Enter your choice: 6

Line ID: 0, Name: d, Capacity: 7, Status: active

Line ID: 3, Name: c, Capacity: 35652353, Status: active

Line ID: 1, Name: a, Capacity: 8, Status: under

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

Enter your choice: 7

Exiting...

PS D:\learning c\output>

5. //stack using array

#include<stdio.h>

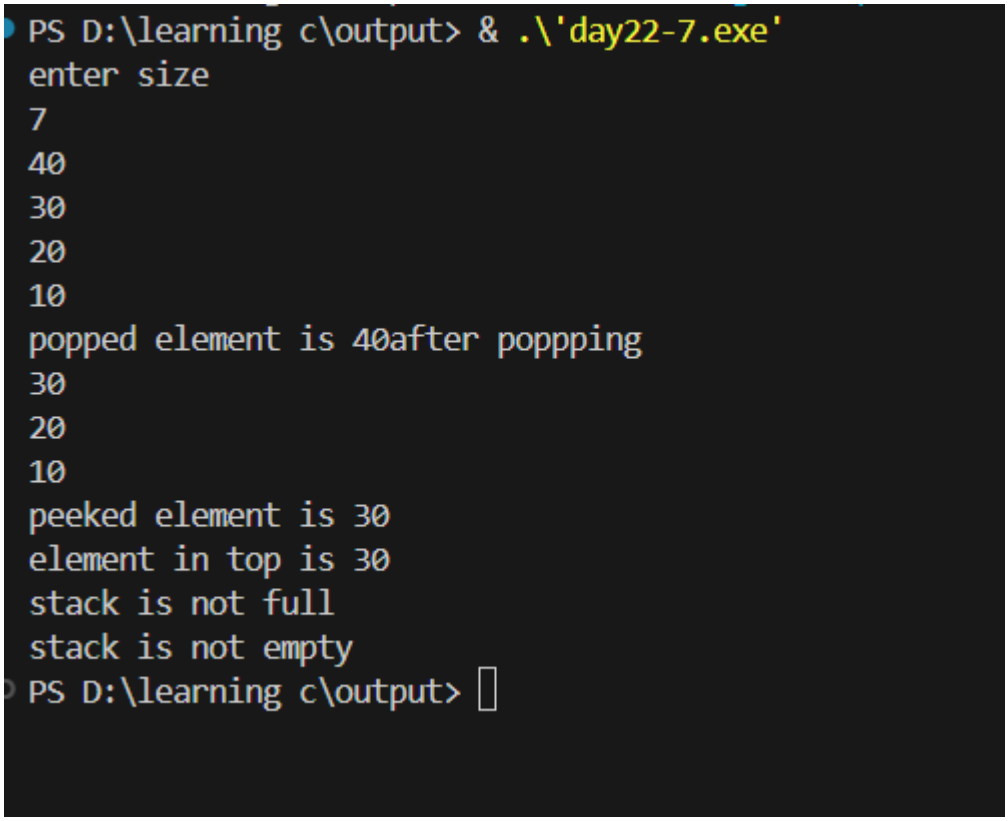
```
#include<stdlib.h>
struct stack
{
   int size;
   int top;
   int *s;
};
void create(struct stack *);
void push(struct stack*,int);
void display(struct stack *);
int pop(struct stack*);
int peek(struct stack*,int);//peaking ata particular index and return the element stored in that index
int stacktop(struct stack*);
int isfull(struct stack*);
int isempty(struct stack*);
int main()
{
   struct stack st;
   int elementpop;
   int elementpeek;
   int stacktopelement;
   int full;
   int empty;
   create(&st);
   push(&st,10);
   push(&st,20);
   push(&st,30);
   push(&st,40);
   display(&st);
   elementpop=pop(&st);
   printf("popped element is %d",elementpop);
```

```
    printf("after poppping \n");
    display(&st);
    elementpeek=peek(&st,1);
    printf("peeked element is %d \n",elementpeek);
    stacktopelement=stacktop(&st);
    printf("element in top is %d \n",stacktopelement);
    full=isfull(&st);
    if(full==1)
    {
        printf("stack is full \n");
    }
    else
    {
        printf("stack is not full \n");

    }
    empty=isempty(&st);
    if(empty==1)
    {
        printf("stack is empty \n");
    }
    else
    {
        printf("stack is not empty \n");
    }
    return 0;

}
void create(struct stack *st)
{
    printf("enter size \n");
```

```
    scanf("%d",&st->size);
    st->top=-1;//since stack is empty
    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]);
        printf("\n");
    }
}
int pop(struct 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(struct stack*st,int x)
{
    int m;
    if(st->top-x+1<0)
    {
       printf("invlaid position \n");
    }
    else
    {
      m=st->s[st->top-x+1];
    }
    return m;
}
int stacktop(struct stack*st)
{   int t;
    if(st->top==-1)
    {
       printf("soory stack empty \n");
    }
    else
    {
      t=st->s[st->top];
    }
```

```
    return t;
}
int isfull(struct stack*st)
{
    if(st->top==st->size-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int isempty(struct stack*st)
{
    if(st->top==-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```



6. //stack using linkedlist

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node*next;
};
struct Node*top=NULL;

void push(int);
void display(struct Node *);
int main()
{
  // top = ( struct Node *)malloc(sizeof(struct Node));
   push(20);
```

```
    push(30);
    push(40);
    push(50);
    display(top);
    printf("\n");
    pop(1);
    printf("after pop operation\n");
    display(top);

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

    }
}
int pop(int x)
{
    struct Node*temp=(struct Node*)malloc(sizeof(struct Node));
    int m=-1;
    if(top==NULL)
    {
```

```
        printf("stack underflow \n");
    }
    else
    {
        temp=top;
        top=top->next;
        m=temp->data;
        free(temp);
    }
    return x;
}
void display(struct Node*p)
{
    while(p!=NULL)
    {
        printf("%d-->",p->data);
            p=p->next;

    }

}
```

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day22-8.exe'
50-->40-->30-->20-->
 after pop operation
40-->30-->20-->
PS D:\learning c\output>
```