```
// Attempt 1
#include<stdio.h>


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

struct Node* createNode(int data){
struct Node *newnode;
newnode=(struct Node*)malloc(1*sizeof(struct Node));

if(!newnode){
    printf("Memory not allocated");

}

newnode->data=data;
newnode->next=NULL;

return newnode;
}

insertNodeFromHead(struct Node **head,int data){
    struct Node *newnode=createNode(data);

    if (!newnode)
    {
        printf("NODE NOT CREATED");
    }

    if(*head==NULL){
        *head=newnode;
    }
    newnode->next=*head;
    *head=newnode;

}
void deleteNodeFromHead(struct Node** head){
    struct Node* temp=*head;
    *head=(*head)->next;

    free(temp);
}
void push(struct node** stack , int data){
    if(insertNodeFromHead(stack,data)){
        printf("OVERFLOW");
```

```
    }
}

void pop(struct node **stack){
    if(*stack==NULL){
        printf("UNDERFLOW");
    }
    deleteNodeFromHead(stack);

}

void show(struct Node** head){
    struct Node*temp=*head;

    while(temp!=NULL){
        printf("data: %d",temp->data);
        temp=temp->next;
    }
}
int main(){
    struct Node *stack;

    push(&stack,10);
    push(&stack,20);
    show(&stack);
    push(&stack,30);
    pop(&stack);
    show(&stack);

    return 0;

}


// Attempt 2
// #include<stdio.h>
// #include<stdlib.h>

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

// struct Node* createNode(int data) {
//     struct Node *newnode;
//     newnode = (struct Node*)malloc(sizeof(struct Node));

//     if (!newnode) {
```

```
//         printf("Memory not allocated\n");
//         return NULL;
//     }

//     newnode->data = data;
//     newnode->next = NULL;

//     return newnode;
// }

// void insertNodeFromHead(struct Node **head, int data) {
//     struct Node *newnode = createNode(data);

//     if (!newnode) {
//         printf("NODE NOT CREATED\n");
//         return;
//     }

//     if (*head == NULL) {
//         *head = newnode;
//     } else {
//         newnode->next = *head;
//         *head = newnode;
//     }
// }

// void deleteNodeFromHead(struct Node **head) {
//     if (*head == NULL) {
//         printf("Stack is empty, cannot pop\n");
//         return;
//     }

//     struct Node* temp = *head;
//     *head = (*head)->next;

//     free(temp);
// }

// void push(struct Node **stack, int data) {
//     insertNodeFromHead(stack, data);
// }

// void pop(struct Node **stack) {
//     deleteNodeFromHead(stack);
// }

// void displayStack(struct Node *stack) {
//     struct Node *current = stack;
```

```
//     while (current != NULL) {
//         printf("%d -> ", current->data);
//         current = current->next;
//     }
//     printf("NULL\n");
// }

// int main() {
//     struct Node *stack = NULL;

//     push(&stack, 10);
//     push(&stack, 20);
//     push(&stack, 30);

//     printf("Stack after pushing elements: ");
//     displayStack(stack);

//     pop(&stack);

//     printf("Stack after popping an element: ");
//     displayStack(stack);

//     return 0;
// }

// Attempt 3
// #include<stdio.h>
// #include<stdlib.h>

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

// struct Node* createNode(int data) {
//     struct Node *newnode;
//     newnode = (struct Node*)malloc(sizeof(struct Node));

//     if (!newnode) {
//         printf("Memory not allocated\n");
//         return NULL;
//     }

//     newnode->data = data;
//     newnode->next = NULL;

//     return newnode;
// }
```

```
// void insertNodeFromHead(struct Node **head, int data) {
//     struct Node *newnode = createNode(data);

//     if (!newnode) {
//         printf("NODE NOT CREATED\n");
//         return;
//     }

//     if (*head == NULL) {
//         *head = newnode;
//     } else {
//         newnode->next = *head;
//         *head = newnode;
//     }
// }

// void deleteNodeFromHead(struct Node **head) {
//     if (*head == NULL) {
//         printf("Stack is empty, cannot pop\n");
//         return;
//     }

//     struct Node* temp = *head;
//     *head = (*head)->next;

//     free(temp);
// }

// void push(struct Node **stack, int data) {
//     insertNodeFromHead(stack, data);
// }

// void pop(struct Node **stack) {
//     deleteNodeFromHead(stack);
// }

// void displayStack(struct Node *stack) {
//     if (stack != NULL) {
//         printf("%d -> ", stack->data);
//         displayStack(stack->next);
//     } else {
//         printf("NULL\n");
//     }
// }

// int main() {
//     struct Node *stack = NULL;
```

```
//     push(&stack, 10);
//     push(&stack, 20);
//     push(&stack, 30);

//     printf("Stack after pushing elements: ");
//     displayStack(stack);

//     pop(&stack);

//     printf("Stack after popping an element: ");
//     displayStack(stack);

//     return 0;
// }



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

// #define SIZE 10 // Size of the array

// // Structure to represent the two stacks using a single array
// struct TwoStacks {
//     int arr[SIZE]; // The array to hold both stacks
//     int top1;      // Top of Stack 1
//     int top2;      // Top of Stack 2
// };

// // Function to initialize the two stacks
// void initializeStacks(struct TwoStacks *stacks) {
//     stacks->top1 = -1;         // Stack 1 starts from the beginning
//     stacks->top2 = SIZE;       // Stack 2 starts from the end
// }

// // Function to push an element onto Stack 1
// void pushStack1(struct TwoStacks *stacks, int value) {
//     // Check for overflow
//     if (stacks->top1 < stacks->top2 - 1) {
//         stacks->top1++;
//         stacks->arr[stacks->top1] = value;
//     } else {
//         printf("Stack 1 Overflow\n");
//     }
// }

// // Function to push an element onto Stack 2
```

```
// void pushStack2(struct TwoStacks *stacks, int value) {
//     // Check for overflow
//     if (stacks->top1 < stacks->top2 - 1) {
//         stacks->top2--;
//         stacks->arr[stacks->top2] = value;
//     } else {
//         printf("Stack 2 Overflow\n");
//     }
// }

// // Function to pop an element from Stack 1
// int popStack1(struct TwoStacks *stacks) {
//     // Check for underflow
//     if (stacks->top1 >= 0) {
//         int value = stacks->arr[stacks->top1];
//         stacks->top1--;
//         return value;
//     } else {
//         printf("Stack 1 Underflow\n");
//         return -1;
//     }
// }

// // Function to pop an element from Stack 2
// int popStack2(struct TwoStacks *stacks) {
//     // Check for underflow
//     if (stacks->top2 < SIZE) {
//         int value = stacks->arr[stacks->top2];
//         stacks->top2++;
//         return value;
//     } else {
//         printf("Stack 2 Underflow\n");
//         return -1;
//     }
// }

// // Function to display the elements in Stack 1
// void displayStack1(struct TwoStacks *stacks) {
//     printf("Stack 1: ");
//     for (int i = 0; i <= stacks->top1; i++) {
//         printf("%d ", stacks->arr[i]);
//     }
//     printf("\n");
// }

// // Function to display the elements in Stack 2
// void displayStack2(struct TwoStacks *stacks) {
//     printf("Stack 2: ");
```

```
//     for (int i = SIZE - 1; i >= stacks->top2; i--) {
//         printf("%d ", stacks->arr[i]);
//     }
//     printf("\n");
// }

// int main() {
//     struct TwoStacks stacks;
//     initializeStacks(&stacks);

//     // Push elements onto Stack 1
//     pushStack1(&stacks, 10);
//     pushStack1(&stacks, 20);
//     pushStack1(&stacks, 30);

//     // Push elements onto Stack 2
//     pushStack2(&stacks, 40);
//     pushStack2(&stacks, 50);
//     pushStack2(&stacks, 60);

//     // Display the stacks
//     displayStack1(&stacks);
//     displayStack2(&stacks);

//     // Pop elements from both stacks
//     printf("Popped from Stack 1: %d\n", popStack1(&stacks));
//     printf("Popped from Stack 2: %d\n", popStack2(&stacks));

//     // Display the stacks again after popping
//     displayStack1(&stacks);
//     displayStack2(&stacks);

//     return 0;
// }
```