```
// Experiment 5: Implement LinkedList:
// You are developing a simple task management system for a personal productivity application.
// The system will manage a list of tasks that a user needs to complete. Each task can be added to the list, marked as completed, or removed from the list.
// The user should also be able to view all current tasks.
// Requirements:
// 1.
// Task Structure: Each task should have the following attributes:
// Task ID (unique identifier)
// Task description (text)
// Status (e.g., "Pending" or "Completed")
// 2.
// Linked List Operations:
// Add Task: Allow the user to add a new task to the end of the list.
// Remove Task: Allow the user to remove a task by its ID.
// Mark Task as Completed: Allow the user to update a task's status to "Completed."
// Display Tasks: Provide a way for the user to view all current tasks in the list.
#include<stdio.h>
#include<stdlib.h>

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

node* createNode(int new_data) {
    node *newnode;
    newnode = (node*)malloc(sizeof(node));
    if (!newnode) {
        printf("Memory is not allocated ");
        exit(0);
    }
    newnode->data = new_data;
    newnode->next = NULL;
    return newnode;
}

void insertNodeFromBeg(node **head, int data_) {
    node* newnode = createNode(data_);
    newnode->next = *head;
    *head = newnode;
}

void insertNodeFromEnd(node **head, int data_) {
    node* newnode = createNode(data_);
    if (*head == NULL) {
        *head = newnode;
```

```
        return;
    }
    node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newnode;
}

void insertNodeATSpecificPos(node **head, int data_, int pos) {
    node* newnode = createNode(data_);
    if (pos == 0) {
        newnode->next = *head;
        *head = newnode;
        return;
    }
    node* temp = *head;
    for (int i = 0; i < pos - 1; i++) {
        temp = temp->next;
        if (temp == NULL) {
            printf("Position out of bounds\n");
            return;
        }
    }
    newnode->next = temp->next;
    temp->next = newnode;
}

void deleteNodeB(node** head) {
    if (*head == NULL) {
        printf("Stack is empty\n");
        return;
    }
    node* del = *head;
    *head = (*head)->next;
    free(del);
}

void deleteNodeE(node** head){
    node* del;
    node* temp;
    del = *head;
    if(del->next==NULL){
        free(head);
        head = NULL;
    }
    else{
        del = *head;
```

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

void deleteNodeSP(node** head ,int pos){
    node* del;
    node* temp;
    if(pos==0){
        printf("List is empty");
    }
    else{
        del = *head;
        for(int i=0;i<pos-1;i++){
            temp = del;
            del = del->next;
            if(del==NULL){
                printf("Invalid position");
                break;
              }

        }
        temp->next = NULL;
        free(del);
    }
}

void push_at_beginning(node **stack, int data) {
    insertNodeFromBeg(stack, data);
}

void push_at_end(node **stack, int data) {
    insertNodeFromEnd(stack, data);
}

void push_at_position(node **stack, int data, int pos) {
    insertNodeATSpecificPos(stack, data, pos);
}

void pop_from_Begining(node **stack) {
    deleteNodeB(stack);
}

void pop_from_End(node **stack) {
```

```
    deleteNodeE(stack);
}

void pop_from_Specific_position(node **stack,int pos) {
    deleteNodeSP(stack,pos);
}


void printStack(node *stack) {
    node *x = stack;
    while (x != NULL){
        printf("%d\t", x->data);
        x = x->next;
    }
    printf("\n");
}

void peek(node *stack) {
    if (stack != NULL) {
        printf("Top: %d\n", stack->data);
    } else {
        printf("Stack is empty\n");
    }
}

int main() {
    node *head = NULL;
    insertNodeFromHead(&head, 10);
    insertNodeFromHead(&head, 20);
    insertNodeFromEnd(&head, 30);
    insertNodeFromEnd(&head, 40);
    insertNodeATSpecificPos(&head, 25, 2);

    printStack(head);

    peek(head);

    pop_from_Begining(&head);
    pop_from_End(&head);
    pop_from_Specific_position(&head,3);

    printStack(head);

    return 0;
}
```