

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Rakshitha Yathiraj (1BM23CS262)

**in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025**

**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering**



This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by Rakshitha Yathiraj(1BM23CS262), who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - (**23CS3PCDST**)work prescribed for the said degree.

Prof. Lakshmi Neelima M

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Working of Stack using Array	4
2	a) Infix to PostFix b) LeetCode-Majority Element	9
3	a) Working of a queue of integers using an array b) working of a circular queue of integers using an array	13
4	a) Implementation of SinglyLinkedList {Creation, Inersion,Display} b) LeetCode- Game Of two Stack	22
5	a) Implementation of SinglyLinkedList {Creation, Deletion ,Display} b) LeetCode- Palindrome Check	26
6	a) Implementation of SinglyLinkedList {Reverse, Sort, Concatenate} b) Implement Single Link List to simulate Stack &Queue Operations	30
7	a) Implement Double Link List {create, Insert, Delete, Display }	41
8	a) Binary Tree Construction, traverse and Display b) LeetCode - PathSum	46
9	a) BFS b) DFS	49
10	a) Linear Probing	54
11	LeetCode Problems	58

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

LAB Program: 01

Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

c) Display

The program should print appropriate messages for stack overflow, stack underflow

PROGRAM:

```
#include<stdio.h>

//stack operations;

int n;

int stack[20];

int top=-1;

void push(int x){

    if(top<=(n-1)){

        top++;

        stack[top]=x;

    }

    else{

        printf("Stack Overflow \n");

    }

}

void pop(){

    if(top>=0){

        int a = stack[top];

        stack[top]=0;

        printf("Deleted element is %d",a);

        top--;

    }

    else{
```

```

        printf("Stack Underflow");
    }
}
void display(){
    for(int i=0; i<=top; i++){
        printf("%d",stack[i]);
    }
}

void main(){
    int choice,num;
    printf("Enter the size");
    scanf("%d",&n);
    printf("Enter the choice \n");
    printf("1.Push \n 2.POP \n 3.Display \n 4.Exit \n");
    scanf("%d",&choice);
    while(choice<4){
        if(choice==1){
            printf("Enter the number to push");
            scanf("%d",&num);
            push(num);
        }
        else if(choice == 2){
            pop();
        }
        else if(choice ==3){
            display();
        }
        else{
            break;
        }
        printf(" \n Enter the choice \n");
    }
}

```

```

        scanf("%d",&choice);
    }
}

=====

#include<stdio.h>

//stack operations;

int n;

int stack[20];

int top=-1;

void push(int x){
    if(top<=(n-1)){
        top++;
        stack[top]=x;
    }
    else{
        printf("Stack Overflow \n");
    }
}

void pop(){
    if(top>=0){
        int a = stack[top];
        stack[top]=0;
        printf("Deleted element is %d",a);
        top--;
    }
    else{
        printf("Stack Underflow");
    }
}

void display(){
    for(int i=0; i<=top; i++){

```

```

        printf("%d",stack[i]);
    }
}

void main(){
    int choice,num;
    printf("Enter the size");
    scanf("%d",&n);
    printf("Enter the choice \n");
    printf("1.Push \n 2.POP \n 3.Display \n 4.Exit \n");
    scanf("%d",&choice);
    while(choice<4){
        if(choice==1){
            printf("Enter the number to push");
            scanf("%d",&num);
            push(num);
        }
        else if(choice == 2){
            pop();
        }
        else if(choice ==3){
            display();
        }
        else{
            break;
        }
        printf(" \n Enter the choice \n");
        scanf("%d",&choice);
    }
}

```

Output:

```
Enter the size2
Enter the choice
1.Push
2.POP
3.Display
4.Exit
1
Enter the number to push56

Enter the choice
1
Enter the number to push56

Enter the choice
1
Enter the number to push26

Enter the choice
1
Enter the number to push89
Stack Overflow

Enter the choice
3
565626
Enter the choice
2
Deleted element is 26
Enter the choice
2
Deleted element is 56
Enter the choice
2
Deleted element is 56
Enter the choice
2
Stack Underflow
```


LAB Program:02

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

Program:

```
#include<stdio.h>

#include<string.h>

char infix[20];

int pos = -1;

char postfix[20];

char result[20];

void push(char a) {

    pos++;

    postfix[pos] = a;

}

char pop() {

    char temp = postfix[pos];

    postfix[pos] = 0;

    pos--;

    return temp;

}

int precedence(char c) {

    if(c == '+' || c == '-') {

        return 1;

    }
```

```

    } else if(c == '*' || c == '/') {

        return 2;

    } else if(c == '^') {

        return 3;

    }

    return 0;

}

void infixtoPostfix() {

    int resind = 0;

    push('(');

    int index = 0;

    int len = strlen(infix);

    infix[len] = '\0';

    len++;

    while(index < len) {

        if(infix[index] == '(') {

            push('(');

        } else if(infix[index] == ')') {

            while(postfix[pos] != '(') {

                char temp = pop();

                result[resind] = temp;

                resind++;

            }

            pop();

        }
    }
}

```

```

        } else if(infix[index] == '+' || infix[index] == '-' || infix[index] == '*' || infix[index] == '/'
|| infix[index] == '^') {

            while(pos >= 0 && precedence(infix[index]) <= precedence(postfix[pos])) {

                char temp = pop();

                result[resind] = temp;

                resind++;

            }

            push(infix[index]);

        } else {

            result[resind] = infix[index];

            resind++;

        }

        index++;

    }

    result[resind] = '\0';
}

int main() {

    printf("Enter the value infix exp: ");

    scanf("%s", infix);

    infixtoPostfix();

    printf("Postfix expression: ");

    for(int i = 0; i < strlen(result); i++) {

        printf("%c", result[i]);

    }

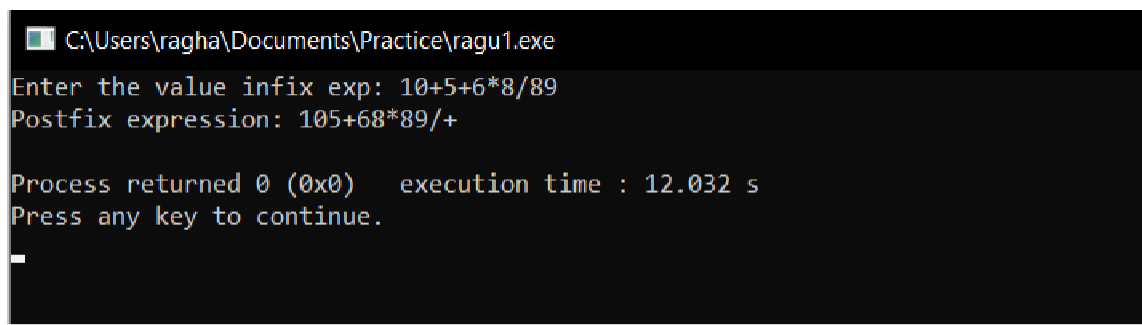
    printf("\n");

    return 0;
}

```

}

Output:



```
C:\Users\ragha\Documents\Practice\ragu1.exe
Enter the value infix exp: 10+5+6*8/89
Postfix expression: 105+68*89/+

Process returned 0 (0x0)   execution time : 12.032 s
Press any key to continue.
-
```

LABProgram:03-a

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

PROGRAM:

```
#include<stdio.h>

#define MAX 4
int queue[MAX];
int rear = -1;
int front = -1;

void insert(int n){
    if(rear == MAX-1){
        printf("Queue Overflow \n");
        return;
    }
    if(rear == -1){
        front = 0;
        rear = 0;
        queue[rear] = n;
    }
    else{
        rear = (rear + 1) % MAX;
        queue[rear] = n;
    }
    printf("Element inserted \n");
}

void deleteel(){
    if(front == -1 || front == rear){
        printf("Queue underflow \n");
        return;
    }
    int temp = queue[front];
    if(front == rear){
        front = -1;
        rear = -1;
    }
    else{
        front = (front + 1) % MAX;
```

```

    }
    printf("Deleted element is = %d \n",temp);
}

void display(){
    if(front == -1){
        printf("Queue is empty \n");
        return;
    }
    if(rear >= front){
        for(int i = front; i <= rear; i++){
            printf("%d ", queue[i]);
        }
    }
    else{
        for(int i = front; i < MAX; i++){
            printf("%d ", queue[i]);
        }
        for(int i = 0; i <= rear; i++){
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}

int main(){
    int k;

    k = 0;
    while(k != 4){
        printf("Enter your choice \n");
        printf("1.Insert \n 2. delete \n 3.Display \n 4.Exit \n");

        scanf("%d",&k);
        if(k == 1){
            printf("Enter the element to be inserted = ");
            int n;
            scanf("%d",&n);
            insert(n);
            continue;
        }
        else if(k == 2){
            deleteel();
            continue;
        }
        else if(k == 3){
            display();
            continue;
        }
    }
}

```

```

    }
    else if(k == 4){
        printf("Thank you");
        break;
    }
}
return 0;
}

```

OUTPUT:

```

OUTPUT
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 56
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 78
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 69
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 45

```

```
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 47
Queue Overflow
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
3
56 78 69 45
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Deleted element is = 56
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Deleted element is = 78
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Deleted element is = 69
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Queue underflow
Enter your choice
1.Insert
2. delete
3.Display
```



```

4.Exit
3
45
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Queue underflow
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
3
45
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
4
Thank you

```

LabProgram3-b

b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

PROGRAM:

```

#include<stdio.h>

#define MAX 4
int queue[MAX];
int rear = -1;
int front = -1;

void insert(int n){
    if((rear + 1) % MAX == front){
        printf("Queue Overflow \n");
        return;
    }
}

```

```

    if(rear == -1){
        front = 0;
        rear = 0;
        queue[rear] = n;
    } else {
        rear = (rear + 1) % MAX;
        queue[rear] = n;
    }
    printf("Element inserted \n");
}

void deleteel(){
    if(front == -1){
        printf("Queue underflow \n");
        return;
    }
    int temp = queue[front];
    if(front == rear){
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX;
    }
    printf("Deleted element is = %d \n", temp);
}

void display(){
    if(front == -1){
        printf("Queue is empty \n");
        return;
    }
    if(rear >= front){
        for(int i = front; i <= rear; i++){
            printf("%d ", queue[i]);
        }
    } else {
        for(int i = front; i < MAX; i++){
            printf("%d ", queue[i]);
        }
        for(int i = 0; i <= rear; i++){
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}

int main(){
    int k;

```

```

k = 0;
while(k != 4){
    printf("Enter your choice \n");
    printf("1.Insert \n 2. delete \n 3.Display \n 4.Exit \n");
    scanf("%d",&k);
    if(k == 1){
        printf("Enter the element to be inserted = ");
        int n;
        scanf("%d",&n);
        insert(n);
        continue;
    }
    else if(k == 2){
        deleteel();
        continue;
    }
    else if(k == 3){
        display();
        continue;
    }
    else if(k == 4){
        printf("Thank you");
        break;
    }
}
return 0;
}

```

OUTPUT:

```

OUTPUT
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 45
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 89

```

```
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 45
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 96
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
3
45 89 45 96
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Deleted element is = 45
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
2
Deleted element is = 89
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
3
45 96
Enter your choice
1.Insert
2. delete
```

```
3.Display
4.Exit
1
Enter the element to be inserted = 89
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
1
Enter the element to be inserted = 56
Element inserted
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
3
45 96 89 56
Enter your choice
1.Insert
2. delete
3.Display
4.Exit
4
Thank you
```

LabProgram 04

WAP to Implement Singly Linked List with following operations a) Createalinkedlist. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

PROGRAM:

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

struct node{
    int data;
    struct node* next;
};
struct node* start = NULL;

//creation of linked list
struct node* create(struct node* start){
    printf("Enter the values for linked list:\n");
    int val;
    scanf("%d",&val);
    while(val!=-1){
        struct node* newnode = (struct node*)malloc(sizeof(struct node));
        newnode->data = val;
        newnode->next = NULL;
        if(start == NULL){
            start = newnode;
        }
        else{
            struct node* temp = start;
            while(temp->next != NULL){
                temp = temp->next;
            }
            temp->next = newnode;
        }
        printf("Enter the next value for linked list (-1 to stop):\n");
        scanf("%d",&val);
    }
    return start;
}

//insertion at the beginning
```

```

struct node* insertatbeg(struct node* start){
    printf("Enter the val to be inserted at the beginning:\n");
    int val;
    scanf("%d",&val);
    struct node*temp = start;
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data= val;
    newnode->next = temp;
    start = newnode;
    return start;
}

//insertion at the end of ll
struct node* insertend(struct node* start){
    int val;
    printf("Enter the val to be inserted at the end:\n");
    scanf("%d",&val);
    struct node*temp = start;
    while(temp->next !=NULL){
        temp = temp->next;
    }
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = NULL;
    temp->next = newnode;
    return start;
}

//insert at kth position of ll
struct node* insertatk(struct node* start,int k){
    int val;
    printf("Enter the val to be inserted at %dth posn \n",k);
    scanf("%d",&val);
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    struct node*temp = start;
    while(k>1){
        k--;
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
    return start;
}

void display(struct node* start)
{

```

```

    struct node* temp = start;
    while(temp != NULL){
        printf("%d-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL \n");
}

//delete head
struct node* deletebeg(struct node*start){
    struct node* temp = start;
    start = start->next;
    free(temp);
    return start;
}

//delete at end
struct node* deleteend(struct node*start){
    struct node* temp = start;
    while(temp->next->next!=NULL){
        temp = temp->next;
    }
    temp->next = NULL;
    return start;
}

//delete at kth posn
struct node* deletekpos(struct node*start , int k){
    struct node* temp = start;
    if(k==1){
        start = start->next;
    }
    else{
        while(k>2){
            k--;
            temp = temp->next;
        }
        temp->next = temp->next->next;
    }
    return start;
}

int main(){
    start = create(start);
    display(start);
    start=insertatbeg(start);
    display(start);
    start = insertend(start);
}

```



```

display(start);
start = insertatk(start,3);
display(start);
start = deletebeg(start);
display(start);
start = deleteend(start);
display(start);
start = deletetekpos(start, 2);
display(start);
}

```

OUTPUT:

```

Enter the values for linked list:
1
Enter the next value for linked list (-1 to stop):
2
Enter the next value for linked list (-1 to stop):
3
Enter the next value for linked list (-1 to stop):
4
Enter the next value for linked list (-1 to stop):
5
Enter the next value for linked list (-1 to stop):
6
Enter the next value for linked list (-1 to stop):
-1
1-> 2-> 3-> 4-> 5-> 6-> NULL
Enter the val to be inserted at the beginning:
45
45-> 1-> 2-> 3-> 4-> 5-> 6-> NULL
Enter the val to be inserted at the end:
96
45-> 1-> 2-> 3-> 4-> 5-> 6-> 96-> NULL
Enter the val to be inserted at 3th posn
5
45-> 1-> 2-> 5-> 3-> 4-> 5-> 6-> 96-> NULL
1-> 2-> 5-> 3-> 4-> 5-> 6-> 96-> NULL
1-> 2-> 5-> 3-> 4-> 5-> 6-> NULL
1-> 5-> 3-> 4-> 5-> 6-> NULL

```

LabProgram:05

WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

Program:

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

struct node{
    int data;
    struct node* next;
};
struct node* start = NULL;

//creation of linked list
struct node* create(struct node* start){
    printf("Enter the values for linked list:\n");
    int val;
    scanf("%d",&val);
    while(val!=-1){
        struct node* newnode = (struct node*)malloc(sizeof(struct node));
        newnode->data = val;
        newnode->next = NULL;
        if(start == NULL){
            start = newnode;
        }
        else{
            struct node* temp = start;
            while(temp->next != NULL){
                temp = temp->next;
            }
            temp->next = newnode;
        }
        printf("Enter the next value for linked list (-1 to stop):\n");
        scanf("%d",&val);
    }
    return start;
}

//insertion at the beginning
struct node* insertatbeg(struct node* start){
```

```

printf("Enter the val to be inserted at the beginning:\n");
int val;
scanf("%d",&val);
struct node*temp = start;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
newnode->data= val;
newnode->next = temp;
start = newnode;
return start;
}

//insertion at the end of ll
struct node* insertend(struct node* start){
    int val;
    printf("Enter the val to be inserted at the end:\n");
    scanf("%d",&val);
    struct node*temp = start;
    while(temp->next !=NULL){
        temp = temp->next;
    }
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = NULL;
    temp->next = newnode;
    return start;
}

//insert at kth position of ll
struct node* insertatk(struct node* start,int k){
    int val;
    printf("Enter the val to be inserted at %dth posn \n",k);
    scanf("%d",&val);
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    struct node*temp = start;
    while(k>1){
        k--;
        temp = temp->next;
    }
    newnode->next = temp->next;
    temp->next = newnode;
    return start;
}

void display(struct node* start)
{
    struct node* temp = start;

```

```

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

//delete head
struct node* deletebeg(struct node*start){
    struct node* temp = start;
    start = start->next;
    free(temp);
    return start;
}

//delete at end
struct node* deleteend(struct node*start){
    struct node* temp = start;
    while(temp->next->next!=NULL){
        temp = temp->next;
    }
    temp->next = NULL;
    return start;
}

//delete at kth posn
struct node* deletekpos(struct node*start , int k){
    struct node* temp = start;
    if(k==1){
        start = start->next;
    }
    else{
        while(k>2){
            k--;
            temp = temp->next;
        }
        temp->next = temp->next->next;
    }
    return start;
}

int main(){
    start = create(start);
    display(start);
    start=insertatbeg(start);
    display(start);
    start = insertend(start);
    display(start);
}

```

```

    start = insertatk(start,3);
    display(start);
    start = deletebeg(start);
    display(start);
    start = deleteend(start);
    display(start);
    start = deletetekpos(start, 2);
    display(start);
}

```

OUTPUT:

```

Enter the values for linked list:
1
Enter the next value for linked list (-1 to stop):
2
Enter the next value for linked list (-1 to stop):
3
Enter the next value for linked list (-1 to stop):
4
Enter the next value for linked list (-1 to stop):
5
Enter the next value for linked list (-1 to stop):
6
Enter the next value for linked list (-1 to stop):
-1
1-> 2-> 3-> 4-> 5-> 6-> NULL
Enter the val to be inserted at the beginning:
45
45-> 1-> 2-> 3-> 4-> 5-> 6-> NULL
Enter the val to be inserted at the end:
96
45-> 1-> 2-> 3-> 4-> 5-> 6-> 96-> NULL
Enter the val to be inserted at 3th posn
5
45-> 1-> 2-> 5-> 3-> 4-> 5-> 6-> 96-> NULL
1-> 2-> 5-> 3-> 4-> 5-> 6-> 96-> NULL
1-> 2-> 5-> 3-> 4-> 5-> 6-> NULL
1-> 5-> 3-> 4-> 5-> 6-> NULL

```

LabProgram-06 -a

a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

PROGRAM:

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

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

struct node* create(struct node* start){
    printf("Enter the values for linked list:\n");
    int val;
    scanf("%d",&val);
    while(val!=-1){
        struct node* newnode = (struct node*)malloc(sizeof(struct node));
        newnode->data = val;
        newnode->next = NULL;
        if(start == NULL){
            start = newnode;
        }
        else{
            struct node* temp = start;
            while(temp->next != NULL){
                temp = temp->next;
            }
            temp->next = newnode;
        }
        printf("Enter the next value for linked list (-1 to stop):\n");
        scanf("%d",&val);
    }
    return start;
}

struct node* reverse(struct node* start){
    struct node* prev = NULL;
    struct node* current = start;
    struct node* next = NULL;
    while(current != NULL){
        next = current->next;
        current->next = prev;
        prev = current;
    }
}
```

```

        current = next;
    }
    return prev;
}

struct node* concatenate(struct node* start1, struct node* start2){
    if(start1 == NULL) return start2;
    if(start2 == NULL) return start1;
    struct node* temp = start1;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = start2;
    return start1;
}

struct node* sort(struct node* start){
    struct node* a = start;
    struct node* b;
    while(a != NULL){
        b = a->next;
        while(b != NULL){
            if(a->data > b->data){
                int temp = a->data;
                a->data = b->data;
                b->data = temp;
            }
            b = b->next;
        }
        a = a->next;
    }
    return start;
}

void display(struct node* start)
{
    struct node* temp = start;
    while(temp != NULL){
        printf("%d-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL \n");
}

int main(){
    struct node* start1 = NULL;
    start1 = create(start1);
    display(start1);
    struct node* start2 = NULL;
    start2 = create(start2);

```

```

    display(start2);
    start1= concatenate(start1,start2);
    display(start1);
    start1 = sort(start1);
    display(start1);
    start1 = reverse(start1);
    display(start1);
}

```

OUTPUT:

```

Enter the values for linked list:
1
Enter the next value for linked list (-1 to stop):
4
Enter the next value for linked list (-1 to stop):
2
Enter the next value for linked list (-1 to stop):
3
Enter the next value for linked list (-1 to stop):
7
Enter the next value for linked list (-1 to stop):
-1
1-> 4-> 2-> 3-> 7-> NULL
Enter the values for linked list:
4
Enter the next value for linked list (-1 to stop):
6
Enter the next value for linked list (-1 to stop):
7
Enter the next value for linked list (-1 to stop):
9
Enter the next value for linked list (-1 to stop):
2
Enter the next value for linked list (-1 to stop):
-1
4-> 6-> 7-> 9-> 2-> NULL
1-> 4-> 2-> 3-> 7-> 4-> 6-> 7-> 9-> 2-> NULL
1-> 2-> 2-> 3-> 4-> 4-> 6-> 7-> 7-> 9-> NULL
9-> 7-> 7-> 6-> 4-> 4-> 3-> 2-> 2-> 1-> NULL

```


LabProgram 06-b

b) WAP to Implement Single Link List to simulate Stack & Queue Operations

PROGRAM:

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

// Define the Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->next = NULL;
    return node;
}

// Function to check if the stack is empty
int isEmpty(struct Node* top) {
    return top == NULL;
}

// Function to push an element onto the stack
void push(struct Node** top, int data) {
    struct Node* node = createNode(data);
    node->next = *top;
    *top = node;
    printf("\nPushed %d onto the stack.", data);
}

// Function to pop an element from the stack
int pop(struct Node** top) {
    if (isEmpty(*top)) {
        printf("Stack underflow\n");
        return -1; // Return -1 to indicate the stack is empty
    }
    struct Node* temp = *top;
    int data = temp->data;
    *top = (*top)->next;
    free(temp);
}
```

```

        return data;
    }

// Function to display the elements in the stack
void display(struct Node* top) {
    if (isEmpty(top)) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    printf("\nStack: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main function with switch-based menu
int main() {
    struct Node* stack = NULL;
    int choice, value;

    while (1) {
        printf("\nStack Operations Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                value = pop(&stack);
                if (value != -1) { // Check for valid pop operation
                    printf("Popped: %d\n", value);
                }
                break;
            case 3:
                display(stack);
                break;
            case 4:

```

```

        printf("Exiting program.\n");
        exit(0);
    default:
        printf("Invalid choice! Please try again.\n");
    }
}

return 0;
}

```

OUTPUT:

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1

Enter value to push: 56

Pushed 56 onto the stack.

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1

Enter value to push: 15

Pushed 15 onto the stack.

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 1

Enter value to push: 96

Pushed 96 onto the stack.

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 96

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 15

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Popped: 56

Stack Operations Menu:

1. Push
2. Pop
3. Display
4. Exit

Enter your choice: 2

Stack underflow

Implementing Queue:

Program:

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

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

struct Queue {
    struct Node* front;
    struct Node* rear;
};

struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->next = NULL;
```

```

        return node;
    }

    struct Queue* createQueue() {
        struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
        queue->front = NULL;
        queue->rear = NULL;
        return queue;
    }

    int isEmpty(struct Queue* queue) {
        return queue->front == NULL;
    }

    void enqueue(struct Queue* queue, int data) {
        struct Node* node = createNode(data);
        if (queue->rear == NULL) {
            queue->front = queue->rear = node;
            return;
        }
        queue->rear->next = node;
        queue->rear = node;
    }

    int dequeue(struct Queue* queue) {
        if (isEmpty(queue)) {
            printf("Queue underflow\n");
            return NULL;
        }
        struct Node* temp = queue->front;
        int data = temp->data;
        queue->front = queue->front->next;
        if (queue->front == NULL) queue->rear = NULL;
        free(temp);
        return data;
    }

    void display(struct Queue* queue) {
        if (isEmpty(queue)) {
            printf("Queue is empty\n");
            return;
        }
        struct Node* temp = queue->front;
        printf("Queue contents:\n");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
    }

```

```

        printf("\n");
    }

int main() {
    struct Queue* queue = createQueue();
    int choice, value;

    while (1) {
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(queue, value);
                printf("Enqueued: %d\n", value);
                break;
            case 2:
                value = dequeue(queue);
                if (value != NULL) {
                    printf("Dequeued: %d\n", value);
                }
                break;
            case 3:
                display(queue);
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}

```

OUTPUT:

```

Queue Operations Menu:
1. Enqueue

```

```
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 56
Enqueued: 56
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 78
Enqueued: 78
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 36
Enqueued: 36
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue contents:
56 78 36
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 56
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
```

Dequeued: 78

Queue Operations Menu:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 2

Dequeued: 36

Queue Operations Menu:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 3

Queue is empty

Queue Operations Menu:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice: 2

Queue underflow

LabProgram 07:

WAP to Implement doubly link list with primitive operations a) Create a doubly linked list.
b) Insert a new node to the left of the node. c) Delete the node based on a specific value
d) Display the contents of the list

PROGRAM:

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

struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *start = NULL;

// creation of linked list

struct node *create(struct node *start)
{
    int data;
    scanf("%d", &data);
    while (data != -1)
    {
        struct node *new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = data;
        if (start == NULL)
        {
            start = new_node;
            new_node->next = NULL;
            new_node->prev = NULL;
        }
        else
        {
            struct node *temp = start;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = new_node;
            new_node->prev = temp;
            new_node->next = NULL;
        }
    }
}
```

```

    }
    scanf("%d", &data);
}
return start;
}

// insertion at the left of the dll
struct node *insert_left(struct node *start, int data, int val)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = data;
    if (start == NULL)
    {
        start = new_node;
        new_node->next = NULL;
        new_node->prev = NULL;
        return start;
    }
    struct node *temp = start;
    while (temp->next != NULL && temp->next->data != val)
    {
        temp = temp->next;
    }
    if (start->data == val)
    {
        temp = start;
        new_node->next = temp;
        temp->prev = new_node;
        start = new_node;
    }
    else if (temp->next == NULL)
    {
        printf("Invalid entry \n");
    }
    else
    {
        new_node->next = temp->next;
        temp->next->prev = new_node;
        temp->next = new_node;
        new_node->prev = temp;
    }

    return start;
}

// deletion of particular node
struct node *delete_node(struct node *start, int val)
{

```

```

    if (start == NULL)
    {
        return start;
    }
    if (start->data == val)
    {
        struct node *temp = start;
        start = start->next;
        if (start != NULL)
        {
            start->prev = NULL;
        }
        free(temp);
        return start;
    }
    struct node *temp = start;
    while (temp->next != NULL && temp->next->data != val)
    {
        temp = temp->next;
    }
    if (temp->next == NULL)
    {
        return start;
    }
    struct node *del = temp->next;
    temp->next = temp->next->next;
    if (temp->next != NULL)
    {
        temp->next->prev = temp;
    }
    free(del);
    return start;
}

// display

void display(struct node *start)
{
    struct node *temp = start;
    while (temp != NULL)
    {
        printf("%d-> ", temp->data);
        temp = temp->next;
    }
    printf(" NULL \n");
}

int main()

```

```

{
    struct node *start = NULL;
    printf("Create the linked list: \n");
    printf("Enter the data to be inserted next press -1 to exit \n");
    start = create(start);
    printf("Linked list created \n");
    display(start);
    printf("Please enter your choice:\n"
           "1. Insert at the left\n"
           "2. Delete Node\n"
           "3. Display\n"
           "4. Exit\n");
    int choice;
    scanf("%d", &choice);
    while (choice != 4)
    {
        switch (choice)
        {
            case 1:
                printf("Enter the data to be inserted: ");
                int data;
                scanf("%d", &data);
                printf("Enter the val before which to be inserted \n");
                int val;
                scanf("%d", &val);
                start = insert_left(start, data, val);
                break;
            case 2:
                printf("Enter the value to be deleted \n");
                scanf("%d", &val);
                start = delete_node(start, val);
                break;
            case 3:
                display(start);
                break;
            default:
                printf("Invalid choice\n");
        }
        scanf("%d",&choice);
    }
}

```

OUTPUT:

```
Create the linked list:
Enter the data to be inserted next press -1 to exit
89
10
20
30
40
-1
Linked list created
89-> 10-> 20-> 30-> 40->  NULL
Please enter your choice:
1. Insert at the left
2. Delete Node
3. Display
4. Exit
1
Enter the data to be inserted: 45
Enter the val before which to be inserted
20
3
89-> 10-> 45-> 20-> 30-> 40->  NULL
2
Enter the value to be deleted
30
3
89-> 10-> 45-> 20-> 40->  NULL
4
```

LabProgram 08

Write a program a) To construct a binary search tree. b) To traverse the tree using all the methods i.e., inorder, preorder and post order c) To display the elements in the tree.

PROGRAM:

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

//Binary tree creation traversal and insertion
struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    if (!newNode) {
        printf("Memory error\n");
        return NULL;
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct node* insert(struct node* root, int data) {
    if (root == NULL) {
        root = createNode(data);
        return root;
    }
    else{
        if (data <= root->data) {
            if (root->left == NULL) {
                root->left = createNode(data);
                return root;
            }
            else{
                root->left = insert(root->left, data);
                return root;
            }
        }
    }
}
```

```

    }
    else{
        if (root->right == NULL) {
            root->right = createNode(data);
            return root;
        }
        else{
            root->right = insert(root->right, data);
            return root;
        }
    }
}

//inorder traversal
void inorder(struct node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf(" %d", root->data);
        inorder(root->right);
    }
}

//postorder traversal
void postorder(struct node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf(" %d", root->data);
    }
}

//preorder traversal
void preorder(struct node* root) {
    if (root != NULL) {
        printf(" %d", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

/// @return
int main(){
    struct node* root = NULL;
    root = insert(root,50);
    root = insert(root, 30);

```

```
root = insert(root, 20);
root = insert(root, 58);
root = insert(root, 42);
printf(" \n Inorder: ");
inorder(root);
printf("\n Postorder: ");
postorder(root);
printf(" \n Preorder: ");
preorder(root);
return 0;}
```

OUTPUT:

```
C:\Users\ragha\1BM23CS258>cd "c:\Users\ragha\1BM23CS258"
Inorder:  20 30 42 50 58
Postorder: 20 42 30 58 50
Preorder: 50 30 20 42 58
```


LabProgram 09-a

a) Write a program to traverse a graph using BFS method.

Program:

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

#define MAX 100

struct Queue {
    int items[MAX];
    int front, rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue is full\n");
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->items[++q->rear] = value;
    }
}

int dequeue(struct Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    } else {
        item = q->items[q->front];
        if (q->front == q->rear) {
```

```

        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}
}

void bfs(int graph[MAX][MAX], int startVertex, int n) {
    int visited[MAX] = {0};
    struct Queue* q = createQueue();

    visited[startVertex] = 1;
    enqueue(q, startVertex);

    printf("BFS Traversal: ");

    while (!isEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);

        for (int i = 1; i <= n; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(q, i);
            }
        }
    }

    printf("\n");
}

int main() {
    int n, startVertex;
    int graph[MAX][MAX];

    printf("Enter the number of vertices : ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);
}

```

```
bfs(graph, startVertex, n);  
  
return 0;  
}
```

Output:

```
Enter the number of vertices : 3  
Enter the adjacency matrix:  
0 1 1 1 0 1 1 1 0  
Enter the starting vertex: 1  
BFS Traversal: 1 2 3
```

LabProgram09-b

- b) Write a program to check whether given graph is connected or not using DFS method.

Program:

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

#define MAX 100 // Maximum number of vertices

// Adjacency matrix to represent the graph
int graph[MAX][MAX];
bool visited[MAX]; // Array to keep track of visited vertices

// Function to perform DFS traversal
void dfs(int vertex, int n) {
    visited[vertex] = true;
    for (int i = 0; i < n; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }
}

// Function to check if the graph is connected
bool isConnected(int n) {
    // Initialize visited array to false
    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    // Perform DFS from the first vertex
    dfs(0, n);

    // Check if all vertices are visited
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return false; // If any vertex is not visited, the graph is not
connected
        }
    }
    return true;
}

int main() {
    int n, edges;
```

```

printf("Enter the number of vertices: ");
scanf("%d", &n);

printf("Enter the number of edges: ");
scanf("%d", &edges);

// Initialize the graph with 0s
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        graph[i][j] = 0;
    }
}

printf("Enter the edges (u v) where u and v are vertices (0-based
index):\n");
for (int i = 0; i < edges; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    graph[u][v] = 1;
    graph[v][u] = 1; // Since the graph is undirected
}

if (isConnected(n)) {
    printf("The graph is connected.\n");
} else {
    printf("The graph is not connected.\n");
}

return 0;
}

```

Output:

```

Enter the number of vertices: 4
Enter the number of edges: 3
Enter the edges (u v) where u and v are vertices (0-based index):
0 1
1 2
2 3

The graph is connected.

```

LabProgram-10

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function $H: K \rightarrow L$ as $H(K) = K \bmod m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing

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

#define MAX 100 // Maximum number of keys in the file
#define EMPTY -1 // Sentinel value to indicate an empty location

// Function to initialize the hash table
void initializeHashTable(int hashTable[], int m) {
    for (int i = 0; i < m; i++) {
        hashTable[i] = EMPTY;
    }
}

// Hash function:  $H(K) = K \bmod m$ 
int hashFunction(int key, int m) {
    return key % m;
}

// Function to insert a key into the hash table using linear probing
void insertKey(int hashTable[], int m, int key) {
    int address = hashFunction(key, m);
    int originalAddress = address;

    // Linear probing to resolve collisions
    while (hashTable[address] != EMPTY) {
        address = (address + 1) % m;
        if (address == originalAddress) { // Table is full
            printf("Hash table is full. Cannot insert key %d\n", key);
            return;
        }
    }
    hashTable[address] = key;
    printf("Key %d inserted at address %d\n", key, address);
}

// Function to search for a key in the hash table
```

```

bool searchKey(int hashTable[], int m, int key) {
    int address = hashFunction(key, m);
    int originalAddress = address;

    // Linear probing to search for the key
    while (hashTable[address] != EMPTY) {
        if (hashTable[address] == key) {
            return true;
        }
        address = (address + 1) % m;
        if (address == originalAddress) { // Table is fully traversed
            break;
        }
    }
    return false;
}

// Function to display the hash table
void displayHashTable(int hashTable[], int m) {
    printf("Hash Table:\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i] == EMPTY) {
            printf("Address %d: EMPTY\n", i);
        } else {
            printf("Address %d: %d\n", i, hashTable[i]);
        }
    }
}

int main() {
    int m, n;

    printf("Enter the number of memory locations (m): ");
    scanf("%d", &m);

    int hashTable[m];
    initializeHashTable(hashTable, m);

    printf("Enter the number of employee keys (n): ");
    scanf("%d", &n);

    int keys[n];
    printf("Enter %d keys (4-digit integers):\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &keys[i]);
    }

    // Insert keys into the hash table

```

```

    for (int i = 0; i < n; i++) {
        insertKey(hashTable, m, keys[i]);
    }

    displayHashTable(hashTable, m);

    int search;
    printf("Enter a key to search: ");
    scanf("%d", &search);

    if (searchKey(hashTable, m, search)) {
        printf("Key %d found in the hash table.\n", search);
    } else {
        printf("Key %d not found in the hash table.\n", search);
    }

    return 0;
}

```

Output:

```

Enter the number of memory locations (m): 10
Enter the number of employee keys (n): 5
Enter 5 keys (4-digit integers):
1234
5678
9101
1123
1456
Enter a key to search: 5678

Key 1234 inserted at address 4
Key 5678 inserted at address 8
Key 9101 inserted at address 1
Key 1123 inserted at address 3
Key 1456 inserted at address 6
Hash Table:
Address 0: EMPTY
Address 1: 9101
Address 2: EMPTY
Address 3: 1123
Address 4: 1234

```



```
Address 5: EMPTY
Address 6: 1456
Address 7: EMPTY
Address 8: 5678
Address 9: EMPTY
Key 5678 found in the hash table.
```

LeetCode Problems

LeetCode 234

Palindrome Linked List

Program:

```
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

bool isPalindrome(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return true;
    }

    char str[100000];
    int len = 0;
    struct ListNode* temp = head;

    while (temp != NULL) {
        str[len++] = temp->val;
        temp = temp->next;
    }

    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - 1 - i]) {
            return false;
        }
    }

    return true;
}
```

Output:

The screenshot shows a LeetCode submission for problem 234, "Palindrome Linked List". The submission is accepted, with a runtime of 2 ms and memory usage of 44.68 MB. The code is in C and implements a recursive solution. The test case shows input head = [1,2,2,1] and output true.

```
struct ListNode {
    int val;
    struct ListNode *next;
};

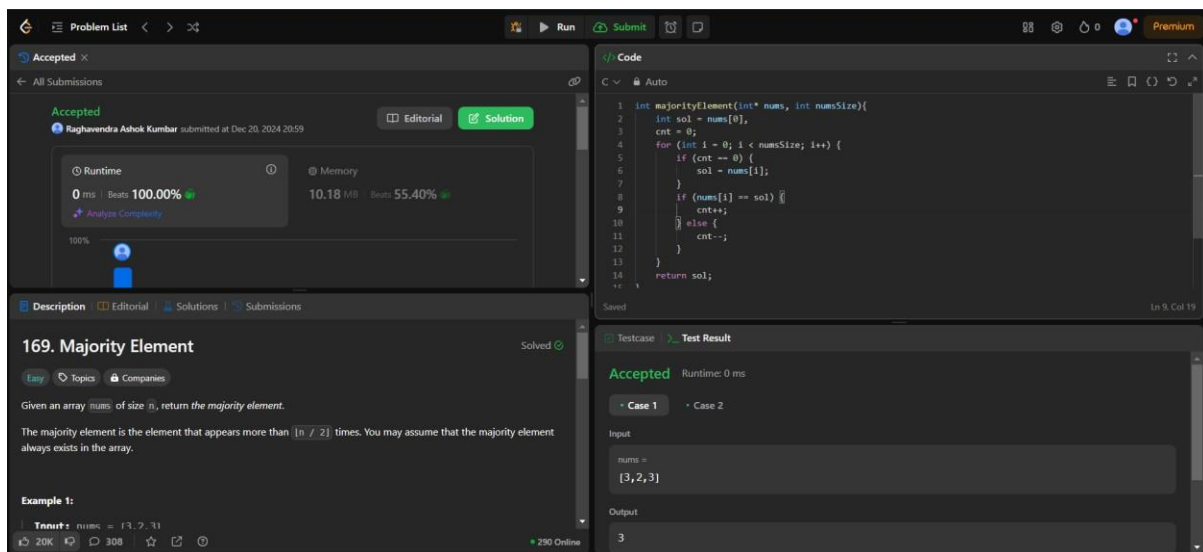
bool isPalindrome(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return true;
    }
    char str[100000];
    int len = 0;
    struct ListNode* temp = head;
    while (temp != NULL) {
        str[len++] = temp->val;
        temp = temp->next;
    }
    return isPalindrome(str, 0, len-1);
}
```

LeetCode 169- Majority Element

Program:

```
int majorityElement(int* nums, int numsSize){
    int sol = nums[0],
    cnt = 0;
    for (int i = 0; i < numsSize; i++) {
        if (cnt == 0) {
            sol = nums[i];
        }
        if (nums[i] == sol) {
            cnt++;
        } else {
            cnt--;
        }
    }
    return sol;
}
```

Output:



The screenshot displays the HackerRank interface for the '169. Majority Element' problem. The problem description states: 'Given an array `nums` of size `n`, return the majority element. The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.' An example is provided: `nums = [3, 2, 3]` returns `3`.

The code editor shows the following C++ solution:

```
1 int majorityElement(int* nums, int numsSize){
2     int sol = nums[0],
3     cnt = 0;
4     for (int i = 0; i < numsSize; i++) {
5         if (cnt == 0) {
6             sol = nums[i];
7         }
8         if (nums[i] == sol) {
9             cnt++;
10        } else {
11            cnt--;
12        }
13    }
14    return sol;
15 }
```

The test results section shows 'Accepted' with a runtime of 0 ms. The input is `nums = [3, 2, 3]` and the output is `3`.

HackerRank- Game Of Two Stacks

Program:

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);
```

```

int twoStacks(int maxSum, int a_count, int* a, int b_count, int*
b) {
    int count = 0, sum = 0, x = 0, y = 0;

    while (x < a_count && (sum + a[x]) <= maxSum) {
        sum += a[x];
        x++;
        count++;
    }

    int maxi = count;

    while (y < b_count) {
        sum += b[y];
        y++;

        while (sum > maxSum && x > 0) {
            x--;
            sum -= a[x];
        }

        if (sum <= maxSum) {
            if(x+y>maxi ){
                maxi = x+y;
            }
        }
    }

    return maxi;
}

```

```

int main() {
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int g = parse_int(ltrim(rtrim(readline())));

    for (int g_itr = 0; g_itr < g; g_itr++) {
        char** first_multiple_input =
split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));
        int m = parse_int(*(first_multiple_input + 1));
    }
}

```

```

    int maxSum = parse_int(*(first_multiple_input + 2));

    char** a_temp = split_string(rtrim(readline()));
    int* a = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) a[i] = parse_int(*(a_temp +
i));

    char** b_temp = split_string(rtrim(readline()));
    int* b = malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) b[i] = parse_int(*(b_temp +
i));

    int result = twoStacks(maxSum, n, a, m, b);
    fprintf(fp_ptr, "%d\n", result);

    free(a);
    free(b);
    free(a_temp);
    free(b_temp);
}

fclose(fp_ptr);

return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length,
stdin);
        if (!line) break;
        data_length += strlen(cursor);
        if (data_length < alloc_length - 1 || data[data_length -
1] == '\n') break;
        alloc_length <= 1;
        data = realloc(data, alloc_length);
        if (!data) return '\0';
    }

    if (data[data_length - 1] == '\n') data[data_length - 1] =
'\0';

```

```

        else data = realloc(data, data_length + 1);
        return data;
    }

char* ltrim(char* str) {
    if (!str || !*str) return str;
    while (*str != '\0' && isspace(*str)) str++;
    return str;
}

char* rtrim(char* str) {
    if (!str || !*str) return str;
    char* end = str + strlen(str) - 1;
    while (end >= str && isspace(*end)) end--;
    *(end + 1) = '\0';
    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");
    int spaces = 0;
    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);
        if (!splits) return splits;
        splits[spaces - 1] = token;
        token = strtok(NULL, " ");
    }
    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);
    if (endptr == str || *endptr != '\0') exit(EXIT_FAILURE);
    return value;
}

```

Output:

Congratulations

You solved this challenge. Would you like to challenge your friends? [f](#) [t](#) [in](#)

[Next Challenge](#)

✔ Test case 0

✔ Test case 1

✔ Test case 2

✔ Test case 3

✔ Test case 4

✔ Test case 5

✔ Test case 6

Compiler Message

Success

Input (stdin) [Download](#)

1	1
2	5 4 10
3	4 2 4 6 1
4	2 1 8 5

Expected Output [Download](#)

1	4
---	---

LeetCode-112 PathSum

Program:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool hasPathSum(struct TreeNode* root, int targetSum) {
    if (!root)
        return false;

    if (!root->left && !root->right)
```



```

        return root->val == targetSum;

        targetSum -= root->val;
        return hasPathSum(root->left, targetSum) || hasPathSum(root->right,
targetSum);
    }
}

```

Output:

The screenshot displays a LeetCode solution for the problem "112. Path Sum". The interface is divided into several sections:

- Accepted Submissions:** Shows the solution was accepted by Raghavendra Ashok Kumbhar on Dec 20, 2024. The runtime is 0 ms (Beats 100.00%) and memory usage is 11.52 MB (Beats 24.95%).
- Description:** The problem statement for "112. Path Sum" is shown, which is an "Easy" problem. It asks to return true if a binary tree has a root-to-leaf path where the sum of node values equals the targetSum. A leaf is defined as a node with no children.
- Example 1:** An example input is provided: root = [5,4,0,11,null,13,4,7,2,null,null,1,1] and targetSum = 22.
- Code:** The C++ code for the solution is displayed, showing a recursive function hasPathSum that checks the left and right subtrees.
- Testcase:** The test result for Case 1 is shown as "Accepted" with a runtime of 0 ms.