

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SUHAS H A(1BM22CS293)

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
Dec 2023- March 2024

**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 **SUHAS H A(1BM22CS293)**, who is a 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 2023-24. 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
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Operations	
2	Infix to Postfix	
3	Linear and Circular queues	
4	Singly Linked Lists	
5	Reverse,Sort and Concatenation of Linked Lists	
6	Stack and Queue Implementation using Linked Lists	
7	Doubly Linked Lists	
8	Binary Search Trees	
9	Graph Traversal Methods BFS and DFS	
10	LeetCode Programs	

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 1:

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.

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int i,stack[SIZE],top=-1;
void main(){
    int value,choice;
    while(1){
        printf("\n1:Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("\nEnter the value:");
                    scanf("%d",&value);
                    push(value);
                    break;
            case 2:pop();
                    break;
            case 3:display();
                    break;
            case 4:exit(0);
            default:printf("Invalid input\n");
        }
    }
}

void push(int value){
    if(top==SIZE-1)
        printf("Overflow\n");
    else{
        top=top+1;
        stack[top]=value;
        printf("%d inserted\n",value);}
}

void pop(){
    int value;
    if(top==-1)
        printf("Underflow\n");
    else{
        value=stack[top];
        top=top-1;
        printf("%d removed\n",value); }
}

void display()
{
}
```

```

int i;
if(top==1)
    printf("Stack is empty");
else{printf("The stack elements are:");
    for(i=0;i>=0;i--)
        printf("%d",stack[i]); }
}

```

Output:

```

C:\Users\teju3\OneDrive\Desl  x  +  v
Enter the value:23
23 inserted

1:Push
2.Pop
3.Display
4.Exit
Enter your choice:1

Enter the value:56
56 inserted

1:Push
2.Pop
3.Display
4.Exit
Enter your choice:3
The stack elements are:23
1:Push
2.Pop
3.Display
4.Exit
Enter your choice:2
56 removed

1:Push
2.Pop
3.Display
4.Exit
Enter your choice:

```

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

LAB PROGRAM 2:

Write a program 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).

```

#include<stdio.h>

#include<string.h>

int ind=0,top=-1,pos=0,length;

char symbol,temp,infix[20],postfix[20],stack[20];

void infix to postfix();

void push(char symbol);

char pop();

```

```

int pred(char symbol);

void main(){
    printf("Enter the infix expression:");
    scanf("%s",infix);
    infixtopostfix();
    printf("\nInfix expression:%s",infix);
    printf("\nPostfix expression:%s",postfix);
}

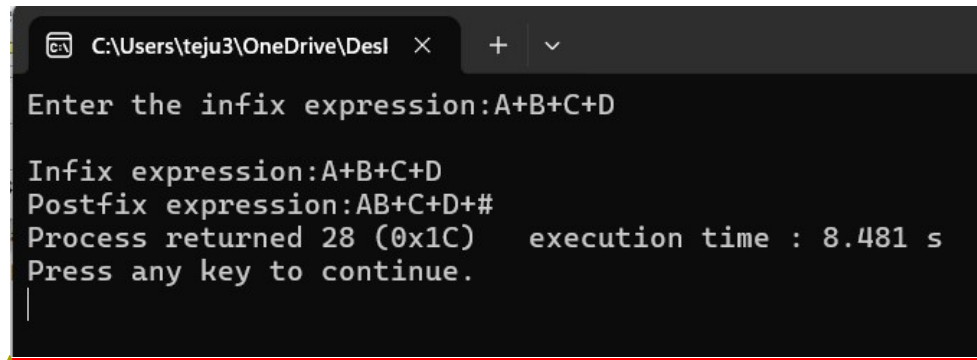
void infixtopostfix(){
    length=strlen(infix);
    push('#');
    while(ind<length){
        symbol=infix[ind];
        switch(symbol){
            case '(':push (symbol);
            break;
            case ')':temp=pop();
                while (temp !=('{){
                    postfix[pos]=temp;
                    pos++;
                    temp=pop(); }
                break;
            case '+':
            case '-':
            case '*':
            case '/':
                while(pred(stack[top])>=pred(symbol)){
                    temp=pop();
                    postfix[pos++]=temp; }

```

```
        push(symbol);
        break;
    default: postfix[pos++] = symbol; }
    ind++;}
while(top > 0) {
    temp = pop();
    postfix[pos++] = temp; }}
void push(char symbol){
    top = top + 1;
    stack[top] = symbol;}
char pop(){
    char symbol;
    symbol = stack[top];
    top = top - 1;
    return (symbol);}
int pred(char symbol){
    int p;

    switch(symbol){
        case '*':
            case '/': p = 2;
            break;
        case '+':
            case '-': p = 1;
            break;
        case '(': p = 0;
            break;
        case '#': p = -1;
```

```
break; }  
return(p);}
```



```
Enter the infix expression:A+B+C+D  
Infix expression:A+B+C+D  
Postfix expression:AB+C+D+#  
Process returned 28 (0x1C) execution time : 8.481 s  
Press any key to continue.  
|
```

Formatted: Font: (Default) Times New Roman, 12 pt

LAB PROGRAM 3:

a) Write a program 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.

```
#include <stdio.h>  
#include <conio.h>  
#define MAX 10  
int queue[MAX];  
int front = -1, rear = -1;  
void insert();  
int delete_element();  
void display();  
int main(){
```



```
int option, val;

do{
printf("\n ***** MAIN MENU *****");
printf("\n 1. Insert an element");
printf("\n 2. Delete an element");
printf("\n 3. Display the queue");
printf("\n 4. EXIT");
printf("\n Enter your option :");
scanf("%d", &option);
switch(option) {
    case 1:insert();
        break;
    case 2:val = delete_element();
        if (val != -1)
            printf("\n The number deleted is : %d", val);
        break;
    case 3:display();
        break; }
}while(option != 4);
getch();
return 0;}

void insert(){
int num;

printf("\n Enter the number to be inserted in the queue : ");
scanf("%d", &num);

if(rear == MAX-1)
printf("\n OVERFLOW");
else if(front == -1 && rear == -1)
front = rear = 0;
```

```
else
rear++;
queue[rear] = num;
printf("%d inserted successfully",num);}

int delete_element(){
int val;
if(front == -1 || front>rear){
printf("\n UNDERFLOW");
return -1;}
else{
val = queue[front];
front++;
if(front > rear)
front = rear = -1;
return val;}
}

void display(){
int i;
printf("\n");
if(front == -1 || front > rear)
printf("\n QUEUE IS EMPTY");
else{
for(i = front;i <= rear;i++)
printf("\t %d", queue[i]);}
}
```

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.

```
#include <stdio.h>

#include <conio.h>

#define MAX 10

int queue[MAX],front=-1,rear=-1;

void insert();

int delete_element();

void display();

int main(){

int option, val;

do{

printf("\n ***** MAIN MENU *****");

printf("\n 1. Insert an element");

printf("\n 2. Delete an element");

printf("\n 3. Display the queue");

printf("\n 4. EXIT");

printf("\n Enter your option : ");

scanf("%d", &option);

switch(option) {

case 1:insert();

break;

case 2:val = delete_element();

if(val!=-1)

printf("\n The number deleted is : %d", val);

break;

case 3:display();
```

```

        break; }
}while(option!=4);
getch();
return 0;}

void insert(){
int num;
printf("\n Enter the number to be inserted in the queue : ");
scanf("%d", &num);
if(front==0 && rear==MAX-1)
printf("\n OVERFLOW");
else if(front== -1 && rear== -1){
front=rear=0;
queue[rear]=num;
printf("Inserted successfully");}
else if(rear==MAX-1 && front!=0){
rear=0;
queue[rear]=num;
printf("Inserted successfully");}
else{
rear++;
queue[rear]=num;
printf("Inserted successfully");} }

int delete_element(){
int val;
if(front== -1 && rear== -1) {
printf("\n UNDERFLOW");
return -1; }
val = queue[front];
if(front==rear)

```

```
front=rear=-1;
else{
    if(front==MAX-1)
front=0;
    else
        front++;}
return val;
printf("Deleted successfully.");}

void display(){
    int i;
    printf("\n");
    if (front ==-1 && rear ==-1)
        printf ("\n QUEUE IS EMPTY");
    else {
        printf("The elements of the queue are:");
        for(i=front;i!=rear;i=(i+1)%MAX)
            printf("\t %d", queue[i]);
        printf("\t %d", queue[i]); }
}
```

```
C:\Users\teju3\OneDrive\Desl  ×  +  v
2 inserted successfully
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :23

***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :3

      2
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :2

The number deleted is : 2
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Display the queue
4. EXIT
Enter your option :|
```

Formatted: Font: (Default) Times New Roman, 12 pt

LAB PROGRAM 4:

Write a program to Implement Singly Linked List with following operations

- Create a linked list.
- Insertion of a node at first position, at any position and at end of list.
- Deletion of first element, specified element and last element in the list.

Display the contents of the linked list.

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
};
```

```
struct node *start = NULL;

struct node *insert_beg(struct node *);

struct node *insert_end(struct node *);

struct node *insert_at_pos(struct node *);

struct node *delete_beg(struct node *);

struct node *delete_end(struct node *);

struct node *delete_at_pos(struct node *);

struct node *display(struct node *);

int main(){

    int option;

    do {

        printf("\n\n *****MAIN MENU *****");

        printf("\n 1: Add a node at the beginning");

        printf("\n 2: Add a node at the end");

        printf("\n 3: Add a node at a specific position");

        printf("\n 4: Delete a node from the beginning");

        printf("\n 5: Delete a node from the end");

        printf("\n 6: Delete a node from a specific position");

        printf("\n 7: Display the list");

        printf("\n 8: EXIT");

        printf("\n\n Enter your option :");

        scanf("%d", &option);

        switch (option) {

            case 1: start = insert_beg(start);

                break;

            case 2: start = insert_end(start);

                break;

            case 3: start = insert_at_pos(start);

                break;
```

```

    case 4: start = delete_beg(start);
        break;
    case 5: start = delete_end(start);
        break;
    case 6: start = delete_at_pos(start);
        break;
    case 7: start = display(start);
        break; }
} while (option != 8);
struct node *temp;
while (start != NULL){
    temp = start;
    start = start->next;
    free(temp); }
return 0;
}

struct node *insert_beg(struct node *start){
    struct node *new_node;
    int num;
    printf("Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
    printf("Inserted at the beginning.\n");
    return start;}

struct node *insert_end(struct node *start){
    struct node *ptr, *new_node;

```



```

int num;

printf("Enter the data: ");
scanf("%d", &num);

new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = num;
new_node->next = NULL;
if (start == NULL) {
    start = new_node;}
else {
    ptr = start;
    while (ptr->next != NULL) {
        ptr = ptr->next;    }
    ptr->next = new_node; }
printf("Inserted at the end.\n");
return start;}

struct node *insert_at_pos(struct node *start){
    struct node *new_node, *ptr, *preptr;
    int pos, num;
    printf("Enter the position to insert at: ");
    scanf("%d", &pos);
    printf("Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    if (pos == 1) {
        new_node->next = start;
        start = new_node;
        printf("Inserted at position %d.\n", pos);
    }
}

```

```

    return start; }
else {
    int i;
    ptr = start;
    for (int i = 1; i < pos && ptr != NULL; i++) {
        preptr = ptr;
        ptr = ptr->next; }
    if (ptr == NULL && pos > i) {
        printf("Invalid position. Node can't be inserted.\n");
        return start; }
    preptr->next = new_node;
    new_node->next = ptr;
    printf("Inserted at position %d.\n", pos);
    return start; }
}

```

```

struct node *delete_beg(struct node *start){
    struct node *ptr;
    ptr = start;
    if (ptr == NULL) {
        printf("Empty list. Can't be deleted.\n");
        return start;}
    else {
        start = start->next;
        free(ptr);
        printf("Deleted at the beginning.\n");
        return start; }
}

struct node *delete_end(struct node *start){

```

```

struct node *ptr, *ptr1;
ptr = start;
if (ptr == NULL) {
    printf("Empty list. Can't be deleted.\n");
    return start; }
else if (ptr->next == NULL){
    free(ptr);
    start = NULL;
    printf("Deleted at the end.\n");
    return start;}
else{
    while (ptr->next != NULL) {
        ptr1 = ptr;
        ptr = ptr->next;
    }
    ptr1->next = NULL;
    free(ptr);
    printf("Deleted at the end.\n");
    return start; }
}

struct node *delete_at_pos(struct node *start){
    struct node *ptr, *preptr;
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);
    if (start == NULL) {
        printf("Empty list. Can't be deleted.\n");
        return start; }
    ptr = start;

```

```

if (pos == 1) {
    start = start->next;
    free(ptr);
    printf("Deleted at position %d.\n", pos);
    return start; }
else {
    for (int i = 1; i < pos && ptr != NULL; i++){
        preptr = ptr;
        ptr = ptr->next; }
    if (ptr == NULL) {
        printf("Invalid position. Node can't be deleted.\n");
        return start;}
    preptr->next = ptr->next;
    free(ptr);
    printf("Deleted at position %d.\n", pos);
    return start; }}

struct node *display(struct node *start){
    struct node *ptr;
    ptr = start;
    if (ptr == NULL){
        printf("Empty list.\n");
        return start;}
    else{
        printf("Linked list elements: ");
        while (ptr != NULL){
            printf("%d\t", ptr->data);
            ptr = ptr->next;}
        printf("\n");
        return start;}}

```

OUTPUT:

```
*****MAIN MENU *****
```

```
1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT
```

```
Enter your option :1
```

```
Enter the data: 10
```

```
Inserted at the beginning.
```

```
*****MAIN MENU *****
```

```
1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT
```

```
Enter your option :2
```

```
Enter the data: 30
```

```
Inserted at the end.
```

```
*****MAIN MENU *****
```

```
1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
```

7: Display the list
8: EXIT

Enter your option :3
Enter the position to insert at: 2
Enter the data: 20
Inserted at position 2.

*****MAIN MENU *****

1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT

Enter your option :7
Linked list elements: 10 20 30

*****MAIN MENU *****

1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT

Enter your option :4
Deleted at the beginning.

*****MAIN MENU *****

1: Add a node at the beginning
2: Add a node at the end

```
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT
```

```
Enter your option :6
Enter the position to delete: 2
Deleted at position 2.
```

```
*****MAIN MENU *****
```

```
1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT
```

```
Enter your option :7
Linked list elements: 20
```

```
*****MAIN MENU *****
```

```
1: Add a node at the beginning
2: Add a node at the end
3: Add a node at a specific position
4: Delete a node from the beginning
5: Delete a node from the end
6: Delete a node from a specific position
7: Display the list
8: EXIT
```

```
Enter your option :8
```

```
Process returned 0 (0x0)   execution time : 67.518 s
```

LAB PROGRAM 5:

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

```
#include <stdio.h>

#include <stdlib.h>

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

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

void sortList(struct Node** head) {
    struct Node *current, *nextNode;
    int temp;
    current = *head;
    while (current != NULL) {
        nextNode = current->next;
        while (nextNode != NULL) {
            if (current->data > nextNode->data) {
                temp = current->data;
```



```

        current->data = nextNode->data;
        nextNode->data = temp; }
        nextNode = nextNode->next;}
    current = current->next;}
}

void reverseList(struct Node** head) {
    struct Node *prev, *current, *nextNode;
    prev = NULL;
    current = *head;
    while (current != NULL) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode; }
    *head = prev;}

void concatenateLists(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
        return; }
    struct Node* temp = *list1;
    while (temp->next != NULL) {
        temp = temp->next; }
    temp->next = list2;}

void main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice;
    int data;

```

```
while(1) {  
    printf("\n1. Insert into List 1\n");  
    printf("2. Insert into List 2\n");  
    printf("3. Sort List 1\n");  
    printf("4. Reverse List 2\n");  
    printf("5. Concatenate Lists\n");  
    printf("6. Print Lists\n");  
    printf("7. Exit\n");  
    printf("Enter your choice: ");  
    scanf("%d", &choice);  
    switch (choice) {  
        case 1: printf("Enter data to insert into List 1: ");  
            scanf("%d", &data);  
            insertAtBeginning(&list1, data);  
            break;  
        case 2: printf("Enter data to insert into List 2: ");  
            scanf("%d", &data);  
            insertAtBeginning(&list2, data);  
            break;  
        case 3: sortList(&list1);  
            printf("List 1 sorted.\n");  
            break;  
        case 4: reverseList(&list1);  
            printf("List 1 reversed.\n");  
            break;  
        case 5: concatenateLists(&list1, list2);  
            printf("Lists concatenated.\n");  
            break;
```

```
case 6:
    printf("List 1: ");
    printList(list1);
    printf("List 2: ");
    printList(list2);
    break;
case 7:
    exit(0);
    break;
default:
    printf("Invalid choice\n");
}
}
}
```

```
C:\Users\teju3\OneDrive\Desl  ×  +  v

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 25

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 89

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 1
Enter data to insert into List 1: 56

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 3
List 1 sorted.
```

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 2
Enter data to insert into List 2: 56
```

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 6
List 1: 25 56 89
List 2: 56 45
```

```
1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 6
List 1: 25 56 89
List 2: 56 45
```

5. Concatenate Lists

6. Print Lists

7. Exit

Enter your choice: 2

Enter data to insert into List 2: 30

1. Insert into List 1

2. Insert into List 2

3. Sort List 1

4. Reverse List 2

5. Concatenate Lists

6. Print Lists

7. Exit

Enter your choice: 6

List 1: 12 10

List 2: 30

1. Insert into List 1

2. Insert into List 2

3. Sort List 1

4. Reverse List 2

5. Concatenate Lists

6. Print Lists

7. Exit

Enter your choice: 5

Lists concatenated.

1. Insert into List 1

2. Insert into List 2

3. Sort List 1

4. Reverse List 2

5. Concatenate Lists

6. Print Lists

7. Exit

Enter your choice: 6

List 1: 12 10 30

List 2: 30

1. Insert into List 1

2. Insert into List 2

```
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 5
Lists concatenated.

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 6
List 1: 12 10 30
List 2: 30

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate Lists
6. Print Lists
7. Exit
Enter your choice: 7

Process returned 0 (0x0)    execution time : 69.851 s
Press any key to continue.
|
```

LAB PROGRAM 6:

Write a program to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h>

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

struct stack *top = NULL;
struct stack *push(struct stack *, int);
struct stack *display(struct stack *);
struct stack *pop(struct stack *);

void main(){
    int val, option;
    while(1){
        printf("\n *****MAIN MENU*****");
        printf("\n 1. PUSH");
        printf("\n 2. POP");
        printf("\n 3. DISPLAY");
        printf("\n 4. EXIT");
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch(option){
            case 1:
                printf("\n Enter the number to be pushed on stack: ");
                scanf("%d", &val);
                top = push(top, val);
                break;
            case 2:
```



```

top = pop(top);
break;
case 3:
top = display(top);
break;
case 4:exit(0);
default:printf("Invalid input"); }}
}

struct stack *push(struct stack *top, int val){
struct stack *ptr;
ptr = (struct stack*)malloc(sizeof(struct stack));
ptr -> data = val;
if(top == NULL){
ptr -> next = NULL;
top = ptr;
printf("The value %d is inserted",val);}
else{
ptr -> next = top;
top = ptr;
printf("The value %d is inserted",val);}
return top;}

struct stack *display(struct stack *top){
struct stack *ptr;
ptr = top;
if(top == NULL)
printf("\n STACK IS EMPTY");
else{
printf("The stack elements are:");
while(ptr != NULL) {

```

```
printf("\n %d", ptr -> data);  
ptr = ptr -> next; } }  
return top;}  
struct stack *pop(struct stack *top){  
struct stack *ptr;  
ptr = top;  
if(top == NULL)  
printf("\n STACK UNDERFLOW");  
else{  
top = top -> next;  
printf("\n The value being deleted is: %d", ptr -> data);  
free(ptr);}  
return top;}  

```

OUTPUT:

C:\Users\teju3\OneDrive\Desl

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 1

Enter the number to be pushed on stack: 89
The value 89 is inserted
*****MAIN MENU*****
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 2

The value being deleted is: 89
*****MAIN MENU*****
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 3
```

```
Enter your option: 3
The stack elements are:
20
10
*****MAIN MENU*****
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 2

The value being deleted is: 20
*****MAIN MENU*****
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 3
The stack elements are:
10
*****MAIN MENU*****
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option: 4

Process returned 0 (0x0)   execution time : 26.630 s
Press any key to continue.
|
```

Queue Implementation:

```
#include <stdio.h>

#include<stdlib.h>

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

struct queue
{
struct node *front;
struct node *rear;
};

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

struct queue *q;

struct queue *insert(struct queue *,int);
struct queue *delete_element(struct queue *);
struct queue *display(struct queue *);

void main(){
int val, option;
q=createQueue(q);
while(1){
printf("\n *****MAIN MENU*****");
printf("\n 1. INSERT");
printf("\n 2. DELETE");
printf("\n 3. DISPLAY");
```

```

printf("\n 4. EXIT");
printf("\n Enter your option : ");
scanf("%d", &option);
switch(option) {
case 1:
printf("\n Enter the number to insert in the queue:");
scanf("%d", &val);
q = insert(q,val);
printf("\nThe value %d is inserted into the queue.\n",val);
break;
case 2:
q = delete_element(q);
break;
case 3:
q = display(q);
break;
case 4:exit(0);
default:printf("Invalid input"); }}
struct queue *insert(struct queue *q,int val){
struct node *ptr;
ptr = (struct node*)malloc(sizeof(struct node));
ptr -> data = val;
if(q -> front == NULL){
q -> front = ptr;
q -> rear = ptr;
q -> front -> next = q -> rear -> next = NULL;}
else{
q -> rear -> next = ptr;
q -> rear = ptr;

```

```
q -> rear -> next = NULL;}

return q;}

struct queue *display(struct queue *q){
struct node *ptr;

ptr = q -> front;

if(ptr == NULL)

printf("\n QUEUE IS EMPTY\n");

else{

printf("\n");

while(ptr!=q -> rear) {

printf("%d\t", ptr -> data);

ptr = ptr -> next; }

printf("%d\t", ptr -> data);}

return q;}

struct queue *delete_element(struct queue *q){

struct node *ptr;

ptr = q -> front;

if(q -> front == NULL)

printf("\n UNDERFLOW\n");

else{

q -> front = q -> front -> next;

printf("\n The value being deleted is : %d\n", ptr -> data);

free(ptr);}

return q;

}
```

OUTPUT:

```
C:\Users\teju3\OneDrive\Desl  ×  +  v
4. EXIT
Enter your option : 1

Enter the number to insert in the queue:99

The value 99 is inserted into the queue.

*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 3

48      99
*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 2

The value being deleted is : 48

*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 3
.

99
*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option :
```

```
Enter your option : 3
10      20
*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 2

The value being deleted is : 10

*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 3

20
*****MAIN MENU*****
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter your option : 4

Process returned 0 (0x0)   execution time : 28.655 s
Press any key to continue.
|
```


LAB PROGRAM 7:

Write a program 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

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* head = NULL;
void createlist() {
    int i, n;
    struct Node* newNode;
    struct Node* temp;
    printf("Enter the number of elements:");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter the element: ");
        scanf("%d", &newNode->data);
        if (head == NULL) {
            head = temp = newNode;
            head->prev = NULL;
            temp->next = NULL;
        }
    }
}
```

```

    } else {
        temp->next = newNode;
        newNode->prev = temp;
        temp = newNode;
        temp->next = NULL; } }

printf("List created successfully.\n");}

void insertLeft(struct Node* temp, int data) {
    struct Node* newNode;
    if (temp == NULL) {
        printf("Target node doesn't exist!\n");
        return; }
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp;
    newNode->prev = temp->prev;
    if (temp->prev != NULL) {
        temp->prev->next = newNode; }
    temp->prev = newNode;
    if (head == temp) {
        head = newNode; }
    printf("Node inserted successfully.\n");}

void deleteNode(int key) {
    struct Node* current = head;
    while (current != NULL) {
        if (current->data == key) {
            if (current->prev != NULL) {
                current->prev->next = current->next; }
            if (current->next != NULL) {
                current->next->prev = current->prev;

```

```

    }
    if (current == head) {
        head = current->next; }
    free(current);
    printf("Node deleted successfully.\n");
    return; }
    current = current->next;}
printf("Node with value %d not found!\n", key);}
void printList() {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("List is empty!\n");
        return; }
    printf("Doubly linked list: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;}
    printf("\n");}
int main() {
    int choice, data, targetValue, deleteValue;
    while(1) {
        printf("\nDoubly Linked List Operations:\n");
        printf("1. Create linked list\n");
        printf("2. Insert left of node\n");
        printf("3. Delete node by value\n");
        printf("4. Print the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

```

```
switch (choice) {
    case 1: createlist();
        break;
    case 2: printf("Enter the value of the node to insert left of: ");
        scanf("%d", &targetValue);
        printf("Enter the element to insert left of the node: ");
        scanf("%d", &data);
        struct Node* temp = head;
        while (temp != NULL) {
            if (temp->data == targetValue) {
                insertLeft(temp, data);
                break; }
            temp = temp->next; }
        break;
    case 3: printf("Enter the value of the node to delete: ");
        scanf("%d", &deleteValue);
        deleteNode(deleteValue);
        break;
    case 4: printList();
        break;
    case 5: exit(0);
        break;
    default:
        printf("Invalid choice!\n");
}
}
return 0;
}
```

OUTPUT:

```
C:\Users\teju3\OneDrive\Desl  ×  +  v
Enter your choice: 2
Enter the value of the node to insert left of: 54
Enter the element to insert left of the node: 99

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 4
Doubly linked list: 12 23 45 56 78

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 74
Node with value 74 not found!

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: |
```

```

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 20
Node deleted successfully.

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 4
Doubly linked list: 10 15 30

Doubly Linked List Operations:
1. Create linked list
2. Insert left of node
3. Delete node by value
4. Print the list
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 60.524 s
Press any key to continue.
|

```

LAB PROGRAM 8:

Write a program

- To construct a binary Search tree.
- To traverse the tree using all the methods i.e., in-order, preorder and post order
- To display the elements in the tree.

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

```
#include <conio.h>
```

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

```
struct node{
```

```
int data;

struct node *left;

struct node *right;

};

struct node *tree=NULL;

struct node *insertElement(struct node *, int);

void preorderTraversal(struct node *);

void inorderTraversal(struct node *);

void postorderTraversal(struct node *);

void main(){

int option, val;

while(1){

printf("\n\n *****MAIN MENU***** \n");

printf("\n 1. Insert Element");

printf("\n 2. Preorder Traversal");

printf("\n 3. Inorder Traversal");

printf("\n 4. Postorder Traversal");

printf("\n 5. Exit");

printf("\n Enter your option : ");

scanf("%d", &option);

switch(option){

case 1:

printf("\n Enter the value of the new node : ");

scanf("%d", &val);

tree = insertElement(tree, val);

break;

case 2:

printf("\n The elements of the tree are : \n");

preorderTraversal(tree);

break;

case 3:
```

```

printf("\n The elements of the tree are : \n");
inorderTraversal(tree);
break;
case 4:
printf("\n The elements of the tree are : \n");
postorderTraversal(tree);
break;
case 5:exit(0);
default:printf("Invalid input");}}}

struct node *insertElement(struct node *tree, int val){
struct node *ptr, *nodeptr, *parentptr;
ptr = (struct node*)malloc(sizeof(struct node));
ptr->data = val;
ptr->left = NULL;
ptr->right = NULL;
if(tree==NULL){
tree=ptr;
tree->left=NULL;
tree->right=NULL;}
else{
parentptr=NULL;
nodeptr=tree;
while(nodeptr!=NULL) {
parentptr=nodeptr;
if(val<nodeptr->data)
nodeptr=nodeptr->left;
else
nodeptr = nodeptr->right; }
if(val<parentptr->data)
parentptr->left = ptr;
else

```



```
parentptr->right = ptr;}
return tree;}

void preorderTraversal(struct node *tree){
if(tree != NULL){
printf("%d\t", tree->data);
preorderTraversal(tree->left);
preorderTraversal(tree->right);}}

void inorderTraversal(struct node *tree){
if(tree != NULL){
inorderTraversal(tree->left);
printf("%d\t", tree->data);
inorderTraversal(tree->right);}}

void postorderTraversal(struct node *tree){
if(tree != NULL){
postorderTraversal(tree->left);
postorderTraversal(tree->right);
printf("%d\t", tree->data);}
}
```

OUTPUT:

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 1

Enter the value of the new node : 7

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 1

Enter the value of the new node : 5

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 1

Enter the value of the new node : 5

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 1

Enter the value of the new node : 8

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 1

Enter the value of the new node : 3

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit

Enter your option : 2

The elements of the tree are :

7 5 3 5 8

```

7       5       3       5       8

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit
Enter your option : 3

The elements of the tree are :
3       5       5       7       8

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit
Enter your option : 4

The elements of the tree are :
3       5       5       8       7

*****MAIN MENU*****

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Exit
Enter your option : 5

Process returned 0 (0x0)   execution time : 54.695 s
Press any key to continue.
|

```

LAB PROGRAM 9:

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

```

#include <stdio.h>

#define MAX 5

void breadth_first_search(int adj[][MAX],int visited[],int start){
    int queue[MAX],rear = -1,front = -1, i;
    queue[++rear] = start;
    visited[start] = 1;
    while(rear != front){
        start = queue[++front];
        if(start == 4)
            printf("%c\t",start+65);
        else
            printf("%c \t",start + 65);
        for(i = 0; i < MAX; i++) {
            if(adj[start][i] == 1 && visited[i] == 0){
                queue[++rear] = i;
                visited[i] = 1; } }
    }
}

int main(){
    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
    printf("\n Enter the adjacency matrix: ");
    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
            scanf("%d", &adj[i][j]);
    breadth_first_search(adj,visited,0);
    return 0;
}

```

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

```

#include <stdio.h>

#define MAX 5

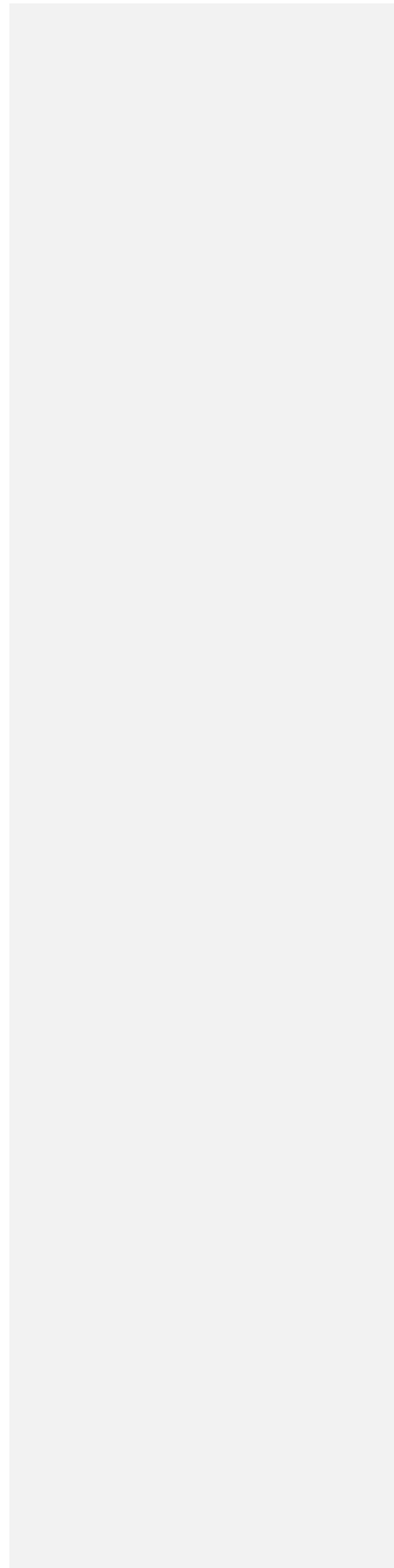
void depth_first_search(int adj[][MAX],int visited[],int start){
    int stack[MAX];
    int top = -1, i;
    printf("%c-",start + 65);
    visited[start] = 1;
    stack[++top] = start;
    while(top!= -1){
        start = stack[top];
        for(i = 0; i < MAX; i++) {
            if(adj[start][i] && visited[i] == 0) {
                stack[++top] = i;
                printf("%c-", i + 65);
                visited[i] = 1;
                break; } }
        if(i == MAX)
            top--;
    }
}

int main(){
    int adj[MAX][MAX];
    int visited[MAX] = {0}, i, j;
    printf("\n Enter the adjacency matrix: ");
    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++)
            scanf("%d", &adj[i][j]);

    printf("DFS Traversal: ");
    depth_first_search(adj,visited,0);
}

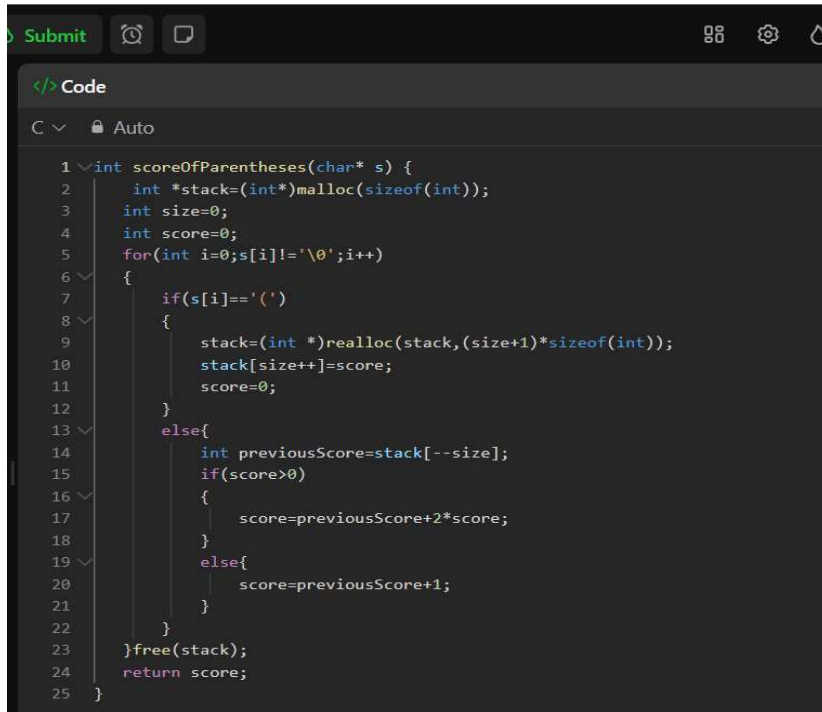
```

```
printf("\n");  
    return 0;  
}
```

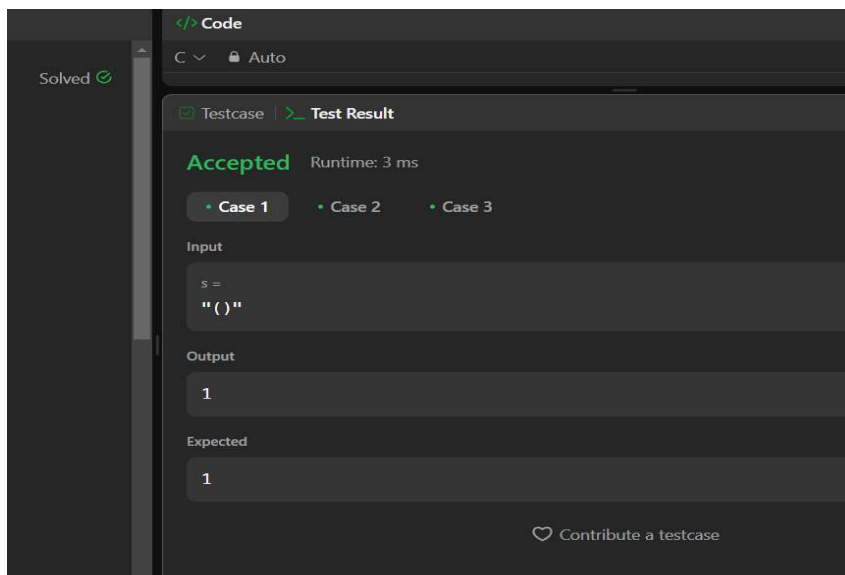


LeetCode Programs:

1.Score of Parentheses(LP:856)



```
1 int scoreOfParentheses(char* s) {
2     int *stack=(int*)malloc(sizeof(int));
3     int size=0;
4     int score=0;
5     for(int i=0;s[i]!='\0';i++)
6     {
7         if(s[i]=='(')
8         {
9             stack=(int *)realloc(stack,(size+1)*sizeof(int));
10            stack[size++]=score;
11            score=0;
12        }
13        else{
14            int previousScore=stack[--size];
15            if(score>0)
16            {
17                score=previousScore+2*score;
18            }
19            else{
20                score=previousScore+1;
21            }
22        }
23    }free(stack);
24    return score;
25 }
```



Solved ✓

Code C Auto

Testcase Test Result

Accepted Runtime: 3 ms

• Case 1 • Case 2 • Case 3

Input

s =
"()"

Output

1

Expected

1

♥ Contribute a testcase

2.Odd Even Linked List(LP:328)

</>Code

C ▾ 🔒 Auto

```
7  */
8  struct ListNode* oddEvenList(struct ListNode* head) {
9      struct ListNode *odd=head;
10     struct ListNode *even=head->next;
11     struct ListNode *evenlist=even;
12     while(odd->next != NULL && even->next != NULL)
13     {
14         odd->next=even->next;
15         odd=odd->next;
16         even->next=odd->next;
17         even=even->next;
18     }
19     odd->next=evenlist;
20     return head;
21 }
```

Saved to local

☒ Testcase | > Test Result

Accepted Runtime: 0 ms

• Case 1

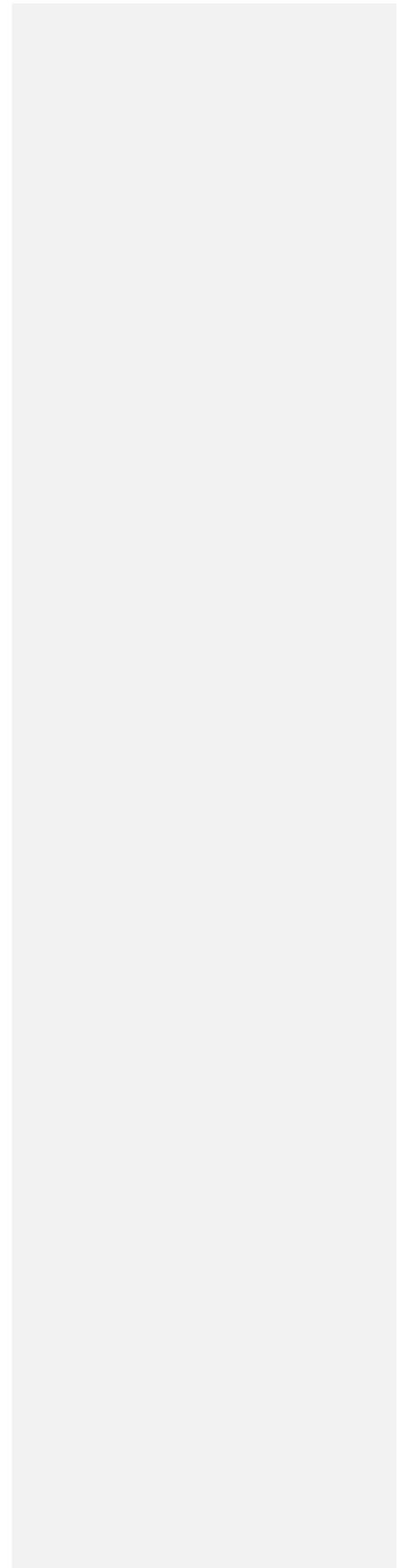
• Case 2

Input

head =
[1,2,3,4,5]

Output

3.Delete middle node of linked list.(LP:2095)



4.Delete a node in BST.(LP:450)

```
</> Code
C  Auto
8  */
9  struct TreeNode *smallest(struct TreeNode *root)
10 {
11     struct TreeNode *cur=root;
12     while(cur->left != NULL)
13         cur=cur->left;
14     return cur;
15 }
16
17 struct TreeNode* deleteNode(struct TreeNode* root, int key) {
18     if(root == NULL)
19         return root;
20
21     if(key<root->val)
22         root->left = deleteNode(root->left,key);
23     else if(key > root->val)
24         root->right = deleteNode(root->right,key);
25     else
26     {
27         if(root->left == NULL)
28         {
29             struct TreeNode *temp =root->right;
30             free(root);
31             return temp;
32         }
33         else if(root->right == NULL)
34         {
35             struct TreeNode *temp=root->left;
```

```
</> Code
C  Auto
33     else if(root->right == NULL)
34     {
35         struct TreeNode *temp=root->left;
36         free(root);
37         return temp;
38     }
39     struct TreeNode *temp= smallest(root->right);
40     root->val=temp->val;
41     root->right = deleteNode(root->right,root->val);
42 }
43 return root;
44 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
3

Output

5. Bottom Left Tree Value.(LP:513)

```
</> Code
C v Auto
8  */
9  int findBottomLeftValue(struct TreeNode* root) {
10     struct TreeNode *queue[100000];
11     int front=0,rear=0;
12     queue[rear++]=root;
13     int leftmostValue=root->val;
14
15     while(front<rear)
16     {
17         int levelSize = rear-front;
18         for(int i=0;i<levelSize;i++)
19         {
20             struct TreeNode *current=queue[front++];
21             if(i==0)
22                 leftmostValue=current->val;
23             if(current->left != NULL)
24                 queue[rear++]=current->left;
25             if(current->right != NULL)
26                 queue[rear++]=current->right;
27         }
28     }
29     return leftmostValue;
30 }
```

```
</> Code
C v Auto
8  */
9  int findBottomLeftValue(struct TreeNode* root) {
10     struct TreeNode *queue[100000];
11     int front=0,rear=0;
12     queue[rear++]=root;
13     int leftmostValue=root->val;
14
15     while(front<rear)
16     {
17         int levelSize = rear-front;
18         for(int i=0;i<levelSize;i++)
19         {
20             struct TreeNode *current=queue[front++];
21             if(i==0)
22                 leftmostValue=current->val;
23             if(current->left != NULL)
24                 queue[rear++]=current->left;
25             if(current->right != NULL)
26                 queue[rear++]=current->right;
27         }
28     }
29     return leftmostValue;
30 }
```

Saved to local

Testcase | Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

root =
[1,2,3,4,null,5,6,null,null,7]

Output

7

Expected

7