VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**Yashraj Sinha 1BM22CS335**

**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 **Yashraj Sinha (1BM22CS335)**, 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 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.**  **Dr. Jyothi S Nayak**

Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Stacks Using Array | 4 |
| 2 | Infix to Postfix Conversion | 7 |
| 3 | Linear Queue Using Array | 9 |
| 4 | Circular Queue Using Array | 12 |
| 5 | Singly Linked List | 16 |
| 6 | Circular Singly Linked List | 20 |
| 7(a) | Sorting , Reversal and Concatenation Of Linked Lists | 24 |
| 7(b) | Stack & Queue implementation using Linked Lists | 27 |
| 8 | Doubly Liked Lists | 29 |
| 9 | Binary Search Trees | 32 |
| 10 | Hashing | 36 |
| 11 | LeetCode Questions | 41 |

**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:**

1. **Push**
2. **Pop**
3. **Display**

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

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

// Structure to represent the stack

struct Stack {

int arr[MAX\_SIZE];

int top;

};

struct Stack stack={{0,0,0,0,0},-1};

// Function to initialize the stack

// Function to check if the stack is empty

int isEmpty() {

if(stack.top ==-1){

return 1;

}

else{

return 0;

}

}

// Function to check if the stack is full

int isFull() {

return stack.top == MAX\_SIZE - 1;

}

// Function to push an element onto the stack

void push(int value) {

if (isFull(stack)) {

printf("Stack Overflow! Cannot push %d.\n", value);

} else {

stack.arr[++(stack.top)] = value;

printf("Pushed %d onto the stack.\n", value);

}

}

// Function to pop an element from the stack

void pop() {

if (isEmpty(stack)) {

printf("Stack Underflow! Cannot pop from an empty stack.\n");

} else {

printf("Popped %d from the stack.\n", stack.arr[stack.top--]);

}

}

// Function to display the elements of the stack

void display() {

if (isEmpty(stack)) {

printf("Stack is empty.\n");

} else {

printf("Stack elements: ");

for (int i = 0; i <= stack.top; i++) {

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

}

printf("\n");

}

}

int main() {

struct Stack stack;

int choice, value;

do {

printf("\n1. 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 the value to push: ");

scanf("%d", &value);

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

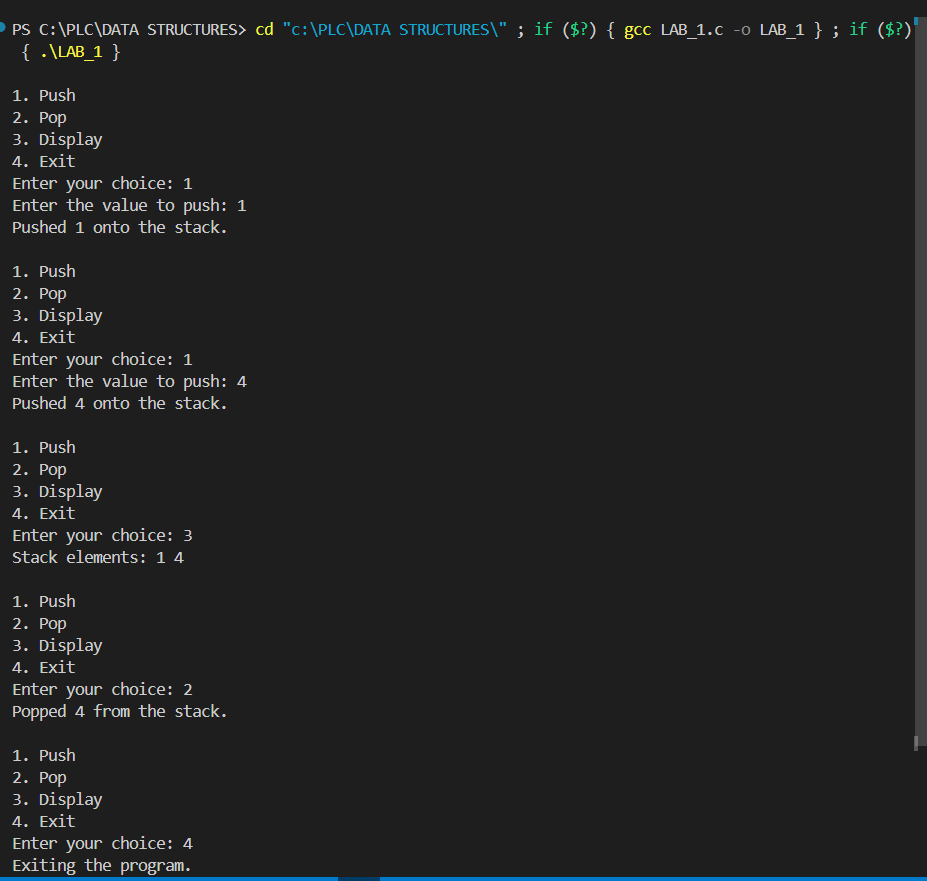
} while (choice != 4);

getchar();

return 0;

}

**Output:**

****

**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),  / (divide) and ^ (power).**

#include<stdio.h>

#include<ctype.h>

#define SIZE 50

char stack[SIZE];

int top=-1;

void push(char x){

top++;

stack[top]=x;

}

char pop(){

char x;

return(stack[top--]);

}

int pr(char symbol){

if(symbol=='^'){

return (3);

}

else if(symbol=='\*' || symbol=='/'){

return(2);

}

else if(symbol=='+' || symbol=='-'){

return(1);

}

else{

return(0);

}

}

void main(){

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("Enter infix expression:");

scanf("%s",infix);

push('#');

while((ch=infix[i++])!='\0'){

if(ch=='(')

push (ch);

else

if(isalnum(ch))

postfix[k++]=ch;

else

if(ch==')')

{

while(stack[top]!='('){

postfix[k++]=pop();

}

elem=pop();

}

else{

while(pr(stack[top])>=pr(ch)){

postfix[k++]=pop();

}

push(ch);

}

}

while(stack[top]!='#'){

postfix[k++]=pop();

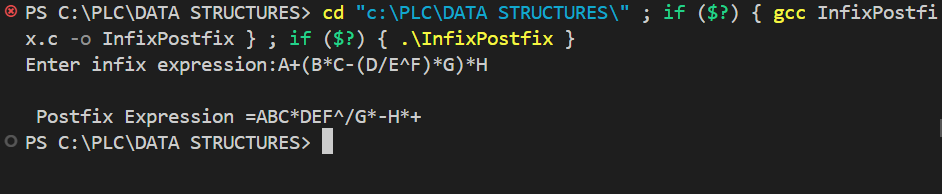
}

postfix[k]='\0';

printf("\n Postfix Expression =%s \n",postfix);

}

**Output:**

****

**Lab Progarm 3:**

**WAP to simulate the working of a queue of integers using an array. Provide the**

**following operations**

**a) Insert**

**b) Delete**

**c) Display**

**The program should print appropriate messages for queue empty and queue overflow**

**Conditions.**

#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

void enqueue();

void dequeue();

void show();

struct queue{

int arr[SIZE];

int top;

int rear;

};

struct queue q={{0,0,0,0,0},-1,-1};

void enqueue(){

int item;

if(q.rear == SIZE-1){

printf("OverFlow \n");

}

else{

if(q.top ==-1 || q.top>=0){

q.top=0;

printf("Enter the element to insert:");

scanf("%d",&item);

printf("\n");

q.rear+=1;

q.arr[q.rear]=item;

}

}

}

void dequeue(){

if(q.top==-1 || q.top>q.rear){

printf("UnderFlow \n");

return;

}

else{

printf("Element deleted:%d \n",q.arr[q.top]);

q.top=q.top+1;

}

}

void show(){

if(q.top == -1){

printf("Empty Queue \n");

}

else{

printf("Queue: \n");

for(int i=q.top;i<=q.rear;i++){

printf("%d ",q.arr[i]);

}

printf("\n");

}

}

int main(){

int ch;

while(1){

printf("1 for Enqueue \n");

printf("2 for Dequeue \n");

printf("3 for Display \n");

printf("4 Exit \n");

printf("Enter your choice \n");

scanf("%d",&ch);

printf("\n");

switch(ch){

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

show();

break;

case 4:

exit(0);

default:

printf("Wrong Choice \n");

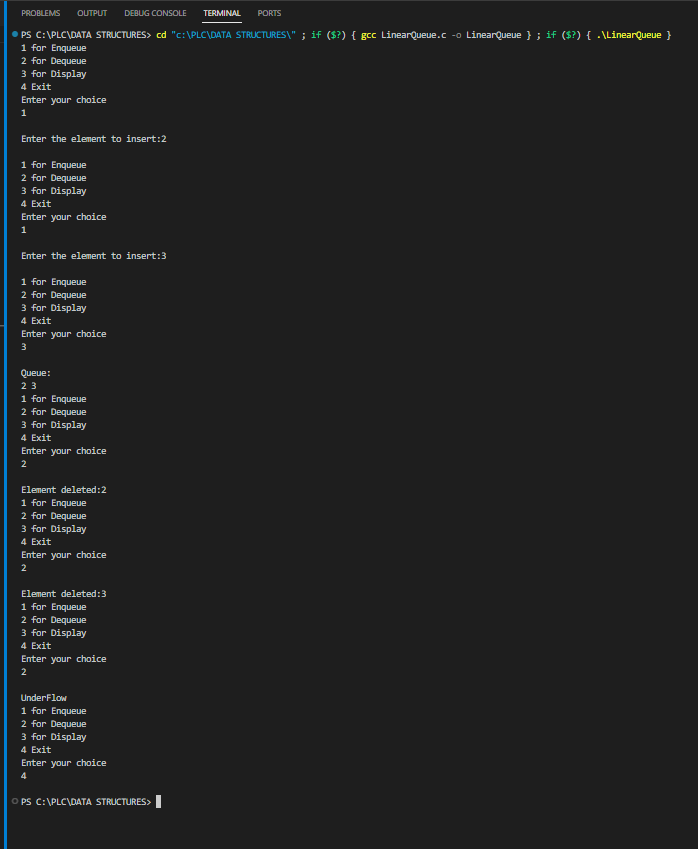
break;

}

}

}

**Output:**

****

**Lab Program 4:**

**WAP to simulate the working of a circular queue of integers using an array. Provide the**

**following operations.**

**a) Insert**

**b) Delete**

**c) Display**

**The program should print appropriate messages for queue empty and queue overflow**

**Conditions.**

#include <stdio.h>

#define MAX\_SIZE 5

// Circular Queue variables

int items[MAX\_SIZE];

int front = -1, rear = -1;

// Function to check if the queue is empty

int isEmpty() {

return (front == -1 && rear == -1);

}

// Function to check if the queue is full

int isFull() {

return ((rear + 1) % MAX\_SIZE == front);

}

// Function to enqueue an element into the circular queue

void enqueue(int value) {

if (isFull()) {

printf("Queue is full. Cannot enqueue %d.\n", value);

return;

}

if (isEmpty()) {

front = 0;

rear = 0;

} else {

rear = (rear + 1) % MAX\_SIZE;

}

items[rear] = value;

printf("%d enqueued to the queue.\n", value);

}

// Function to dequeue an element from the circular queue

int dequeue() {

int dequeuedItem;

if (isEmpty()) {

printf("Queue is empty. Cannot dequeue.\n");

return -1;

}

dequeuedItem = items[front];

if (front == rear) {

// If there was only one element in the queue

front = -1;

rear = -1;

} else {

front = (front + 1) % MAX\_SIZE;

}

printf("%d dequeued from the queue.\n", dequeuedItem);

return dequeuedItem;

}

// Function to display the elements of the circular queue

void display() {

if (isEmpty()) {

printf("Queue is empty.\n");

return;

}

printf("Queue elements: ");

int i = front;

do {

printf("%d ", items[i]);

i = (i + 1) % MAX\_SIZE;

} while (i != (rear + 1) % MAX\_SIZE);

printf("\n");

}

int main() {

int choice, value;

do {

printf("\nCircular Queue Operations:\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 the value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

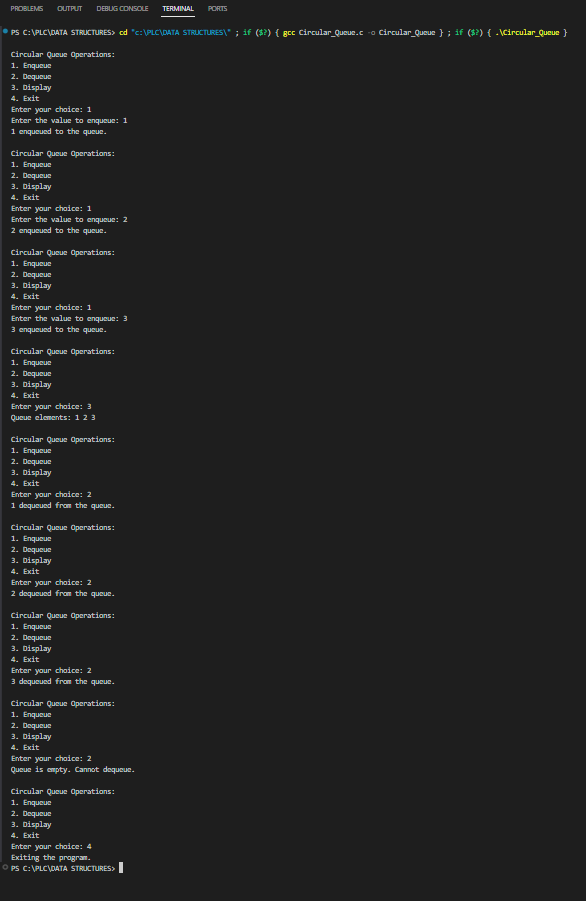
}

} while (choice != 4);

return 0;

}

**Output:**

****

**Lab Program 5:**

**WAP to Implement Singly Linked List with following operations**

**a) Create a linked list.**

**b) Insertion of a node at first position, and at end of list.**

**c) Delete a node at front and at the end of the list.**

**d) Display the contents of the linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

//Create Linked List

struct Node\* createLL(struct Node\* head){

int num;

printf("Enter -1 to stop.\n");

printf("Enter Number:");

scanf("%d",&num);

while(num!=-1){

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

struct Node\* p;

p=head;

if(head==NULL){

newNode->data=num;

newNode->next=NULL;

head=newNode;

}

else{

while(p->next!=NULL){

p=p->next;

}

newNode->data=num;

p->next=newNode;

newNode->next=NULL;

}

printf("Enter Number:");

scanf("%d",&num);

}

return head;

}

//Display Linked List

struct Node\* displayLL(struct Node\* head){

struct Node\*p;

p=head;

printf("Linked List Elements:");

while(p !=NULL){

printf("%d ",p->data);

p=p->next;

}

printf("\n");

return head;

}

//Insert a node at First Position

struct Node\* insertAtBeg(struct Node\* head){

int num;

printf("Enter Number:");

scanf("%d",&num);

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

newNode->data=num;

newNode->next=head;

head=newNode;

return head;

}

//Insert a node at End Position

struct Node\* insertAtEnd(struct Node\* head){

int num;

struct Node \*p;

p=head;

printf("Enter Number:");

scanf("%d",&num);

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

while(p->next!=NULL){

p=p->next;

}

newNode->data=num;

p->next=newNode;

newNode->next=NULL;

return head;

}

//Insert a node at any Position

struct Node\* insertAtPos(struct Node\* head,int pos){

int num,i=0;

struct Node \*p;

p=head;

printf("Enter Number:");

scanf("%d",&num);

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

while(i!=pos-1){

p=p->next;

i++;

}

newNode->data=num;

newNode->next=p->next;

p->next=newNode;

return head;

}

//Delete a node at front

struct Node\* delAtFront(struct Node\* head){

if(head==NULL){

printf("Linked List alredy empty.\n");

return head;

}

else{

struct Node\* p;

p=head->next;

free(head);

head=p;

return head;

}

}

//Delete a node at end

struct Node\* delAtEnd(struct Node\* head){

struct Node \*p,\*preNode;

p=head;

while(p->next!=NULL){

preNode=p;

p=p->next;

}

preNode->next=NULL;

free(p);

return head;

}

//Delete a node at any position

struct Node\* delAtPos(struct Node\* head, int pos){

struct Node\*p,\*preNode;

int i=0;

p=head;

if(pos==0){

head=delAtFront(head);

return head;

}

while(i!=pos){

preNode=p;

p=p->next;

i++;

}

preNode->next=p->next;

free(p);

return head;

}

int main(){

struct Node\* head=NULL;

head=createLL(head);

head=displayLL(head);

head=insertAtBeg(head);

printf("Linked list after insertion at begining.\n");

head=displayLL(head);

head=insertAtEnd(head);

printf("Linked list after insertion at end.\n");

head=displayLL(head);

head=insertAtPos(head,2);

printf("Linked list after insertion at position.\n");

head=displayLL(head);

head=delAtFront(head);

printf("Linked list after deletion at front.\n");

head=displayLL(head);

head=delAtEnd(head);

printf("Linked list after deletion at end.\n");

head=displayLL(head);

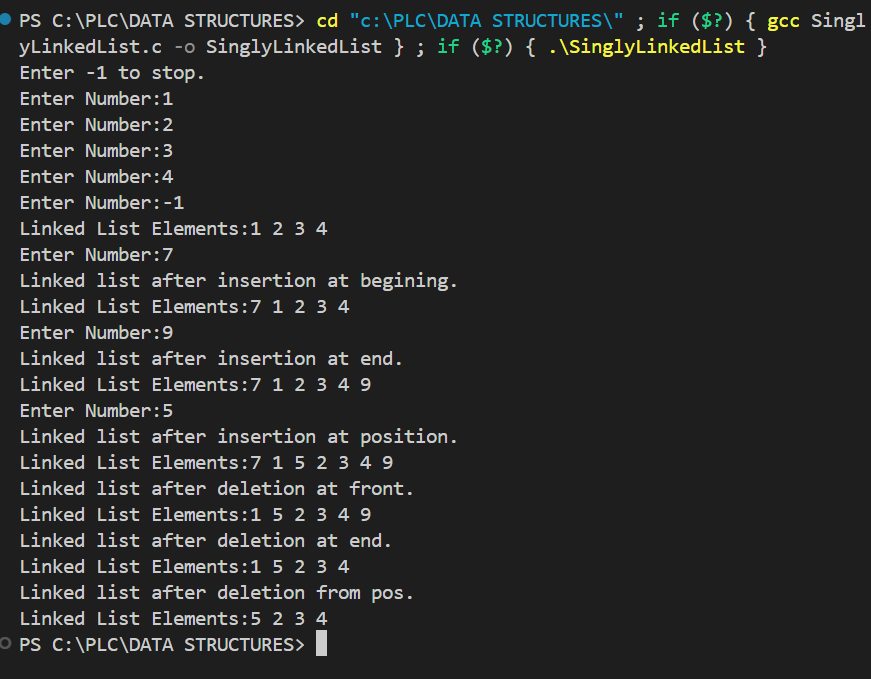
head=delAtPos(head,0);

printf("Linked list after deletion from pos.\n");

head=displayLL(head);

}

**Output:**

****

**Lab Program 6:**

**WAP to Implement Circular Singly Linked List with following operations**

**a) Create a linked list.**

**b) Insertion of a node at first position, and at end of list and at any position**

**c) Delete a node at front and at the end of the list and at any position**

**d) Display the contents of the linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

//Create Circular Linked List

struct Node\* createCircularLL(struct Node\* head){

int data;

struct Node\*p;

printf("Enter -1 to stop.\n");

printf("Enter Number:");

scanf("%d",&data);

while(data!=-1){

struct Node \*newNode=(struct Node\*)malloc(sizeof(struct Node));

newNode->data=data;

if(head==NULL){

newNode->next=newNode;

head=newNode;

}

else{

p=head;

while(p->next!=head){

p=p->next;

}

p->next=newNode;

newNode->next=head;

}

printf("Enter Number:");

scanf("%d",&data);

}

return head;

}

//Display Circular Linked List

struct Node\* displayCircularLL(struct Node\* head){

struct Node\* p;

p=head;

printf("Circular List Elements:");

while(p->next !=head){

printf("%d ",p->data);

p=p->next;

}

printf("%d ",p->data);

printf("\n");

return head;

}

//Insert At Begining

struct Node\* insertFirst(struct Node\* head){

int num;

struct Node \*p;

p=head;

printf("Enter Number:");

scanf("%d",&num);

struct Node \*newNode=(struct Node\*)malloc(sizeof(struct Node));

newNode->data=num;

while(p->next!=head){

p=p->next;

}

p->next=newNode;

newNode->next=head;

head=newNode;

return head;

}

//Insert At End

struct Node\* insertEnd(struct Node\* head){

int num;

struct Node \*p;

p=head;

printf("Enter Number:");

scanf("%d",&num);

struct Node \*newNode=(struct Node\*)malloc(sizeof(struct Node));

newNode->data=num;

while(p->next!=head){

p=p->next;

}

p->next=newNode;

newNode->next=head;

return head;

}

//Insert At Any Position

struct Node\* insertPosition(struct Node\* head, int pos){

int num,i=0;

struct Node \*p;

p=head;

printf("Enter Number:");

scanf("%d",&num);

struct Node \*newNode=(struct Node\*)malloc(sizeof(struct Node));

newNode->data=num;

if(pos==0){

head=insertFirst(head);

return head;

}

else{

while(i!=pos-1){

p=p->next;

i++;

}

newNode->next=p->next;

p->next=newNode;

return head;

}

}

//Delete From Front

struct Node\* DelFromFront(struct Node\* head){

struct Node \*p;

p=head;

while(p->next!=head){

p=p->next;

}

p->next=head->next;

free(head);

head=p->next;

}

//Delete From End

struct Node\* DelFromEnd(struct Node\* head){

struct Node \*p,\*preNode;

p=head;

while(p->next!=head){

preNode=p;

p=p->next;

}

preNode->next=p->next;

free(p);

return head;

}

//Delete From Any Position

struct Node\* DelFromPos(struct Node\* head,int pos){

int i=0;

struct Node\* p,\*preNode;

p=head;

if(pos==0){

head=DelFromFront(head);

return head;

}

else{

while(i!=pos){

preNode=p;

p=p->next;

i++;

}

preNode->next=p->next;

free(p);

return head;

}

}

int main(){

struct Node\* head=NULL;

head=createCircularLL(head);

printf("Linked List Created. \n");

head=displayCircularLL(head);

head=insertFirst(head);

printf("Linked List after insertion at begining.\n");

head=displayCircularLL(head);

head=insertEnd(head);

printf("Linked List after insertion at end.\n");

head=displayCircularLL(head);

head=insertPosition(head, 3);

printf("Linked List after insertion at pos.\n");

head=displayCircularLL(head);

head=DelFromFront(head);

printf("Linked List after deletion from front\n");

head=displayCircularLL(head);

head=DelFromEnd(head);

printf("Linked List after Deletion form end.\n");

head=displayCircularLL(head);

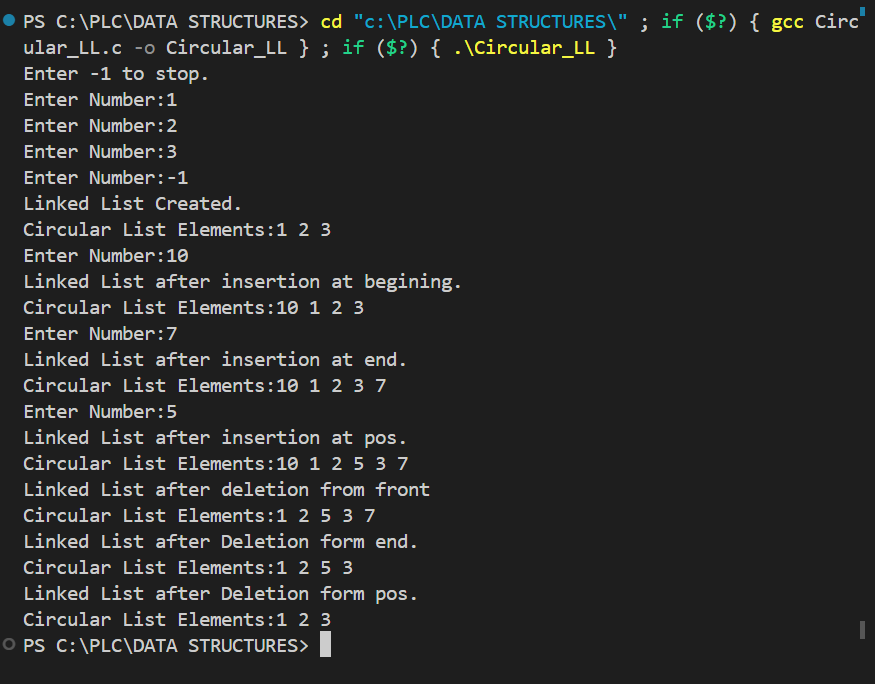
head=DelFromPos(head,2);

printf("Linked List after Deletion form pos.\n");

head=displayCircularLL(head);

}

**Output:**



**Lab Program 7:**

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

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

//Create Linked List

struct Node\* createLL(struct Node\* head){

int num;

printf("Enter -1 to stop.\n");

printf("Enter Number:");

scanf("%d",&num);

while(num!=-1){

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

struct Node\* p;

p=head;

if(head==NULL){

newNode->data=num;

newNode->next=NULL;

head=newNode;

}

else{

while(p->next!=NULL){

p=p->next;

}

newNode->data=num;

p->next=newNode;

newNode->next=NULL;

}

printf("Enter Number:");

scanf("%d",&num);

}

return head;

}

//Display Linked List

struct Node\* displayLL(struct Node\* head){

struct Node\*p;

p=head;

printf("Linked List Elements:");

while(p !=NULL){

printf("%d ",p->data);

p=p->next;

}

printf("\n");

return head;

}

//Sort Linked List

struct Node\* sortLL(struct Node\* head){

struct Node\* ptr,\*trav;

int temp;

ptr=head;

while(ptr->next != NULL){

trav=ptr->next;

while(trav!=NULL){

if(ptr->data > trav->data){

temp=ptr->data;

ptr->data=trav->data;

trav->data=temp;

}

trav=trav->next;

}

ptr=ptr->next;

}

return head;

}

struct Node\* LLRev(struct Node\* head){

struct Node\* temp;

struct Node\* prev=NULL;

struct Node\* cur=head;

while(cur!=NULL){

temp=cur->next;

cur->next=prev;

prev=cur;

cur=temp;

}

head=prev;

return head;

}

struct Node\* ConcatLL(struct Node\* head1,struct Node\* head2){

struct Node\*ptr;

ptr=head1;

while(ptr->next!=NULL){

ptr= ptr->next;

}

ptr->next=head2;

return head1;

}

int main(){

struct Node\* head=NULL;

struct Node\* head1=NULL;

struct Node\* head2=NULL;

head=createLL(head);

head=displayLL(head);

printf("After Sorting \n");

head=sortLL(head);

head=displayLL(head);

printf("After Reversal \n");

head=LLRev(head);

head=displayLL(head);

printf("Enter 1st Linked List : \n");

head1=createLL(head1);

printf("Enter 2nd Linked List : \n");

head2=createLL(head2);

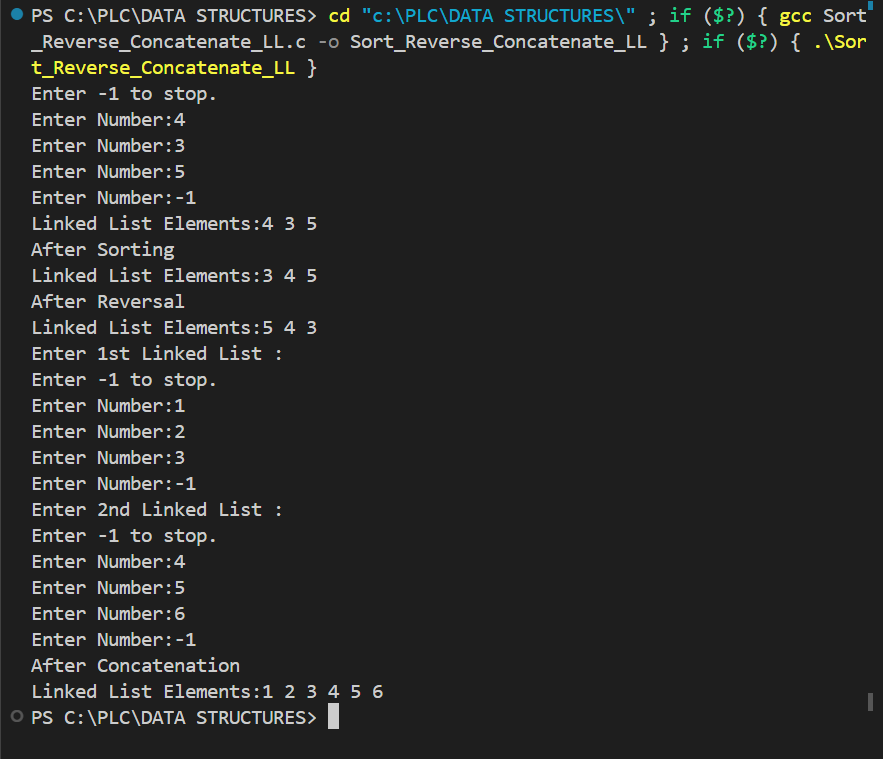
printf("After Concatenation \n");

head1=ConcatLL(head1,head2);

head1=displayLL(head1);

}

**Output:**

****

**7(b) WAP to Implement Single Link List to simulate Stack &Queue Operations.**

**Stack Implementation :-**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\* next;

};

int isEmpty(struct Node\* top){

if(top == NULL){

return 1;

}

return 0;

}

struct Node\* displayLL(struct Node\* top){

if(isEmpty(top)){

printf("No elements to print.\n");

return top;

}

printf("Linked list elements:");

struct Node\* p =top;

while(p!=NULL){

printf("%d ",p->data);

p=p->next;

}

printf("\n");

return top;

}

struct Node\* push(struct Node\* top,int data){

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

if(top==NULL){

newNode->data=data;

top=newNode;

newNode->next=NULL;

return top;

}

else{

newNode->data=data;

newNode->next=top;

top=newNode;

return top;

}

}

struct Node\* pop(struct Node\* top){

if(isEmpty(top)){

printf("Stack is empty.\n");

}

else{

struct Node\* p =top;

top=p->next;

free(p);

return top;

}

}

int peek(struct Node\* top){

struct Node\* p =top;

return p->data;

}

int main(){

struct Node\* top=NULL;

top=push(top,5);

top=push(top,7);

top=push(top,9);

printf("Linked list after push operation \n");

top=displayLL(top);

printf("Linked list after pop operation \n");

top=pop(top);

top=displayLL(top);

int x=peek(top);

printf("Top Element: %d \n",x);

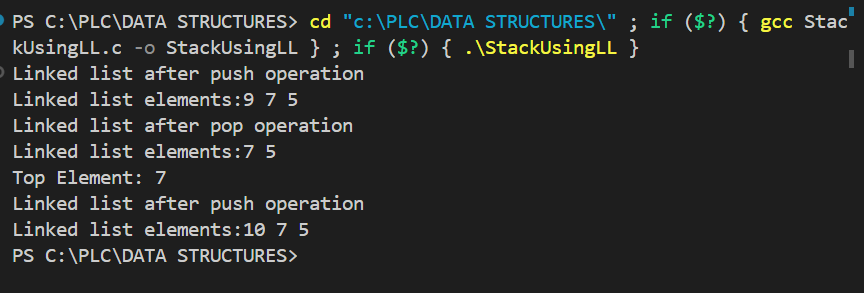
top=push(top,10);

printf("Linked list after push operation \n");

top=displayLL(top);

}

**Output:**

****

**Lab Program 8:**

**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**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\* prev;

struct Node\* next;

};

//Create a DLL

struct Node\* createDLL(struct Node\* head){

int num;

printf("Enter -1 to stop.\n");

printf("Enter Number:");

scanf("%d",&num);

while(num!=-1){

struct Node\* ptr;

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if(head==NULL){

newNode->data=num;

newNode->prev=NULL;

newNode->next=NULL;

head=newNode;

}

else{

ptr=head;

while(ptr->next!=NULL){

ptr=ptr->next;

}

newNode->data=num;

newNode->prev=ptr;

ptr->next=newNode;

newNode->next=NULL;

}

printf("Enter Number:");

scanf("%d",&num);

}

return head;

}

//Display Linked List

struct Node\* displayLL(struct Node\* head){

struct Node\*p;

p=head;

printf("Linked List Elements:");

while(p !=NULL){

printf("%d ",p->data);

p=p->next;

}

printf("\n");

return head;

}

//Insert a new node to the left of the node

struct Node\* insertLeft(struct Node\* head){

struct Node\*ptr;

struct Node\* newNode=(struct Node\*)malloc(sizeof(struct Node));

int n,val;

printf("Enter Number:");

scanf("%d",&n);

printf("Enter the value before which number is to be inserted:");

scanf("%d",&val);

ptr=head;

while(ptr->data!=val){

ptr=ptr->next;

}

newNode->data=n;

newNode->next=ptr;

newNode->prev=ptr->prev;

ptr->prev->next=newNode;

ptr->prev=newNode;

return head;

}

//Delete the node based on a specific value

struct Node\* deleteNode(struct Node\* head){

int val;

struct Node\* ptr;

printf("Enter the value for which node is to be deleted:");

scanf("%d",&val);

ptr=head;

while(ptr->data!=val){

ptr=ptr->next;

}

ptr->prev->next=ptr->next;

ptr->next->prev=ptr->prev;

return head;

}

int main(){

struct Node\* head=NULL;

head=createDLL(head);

head=displayLL(head);

head=insertLeft(head);

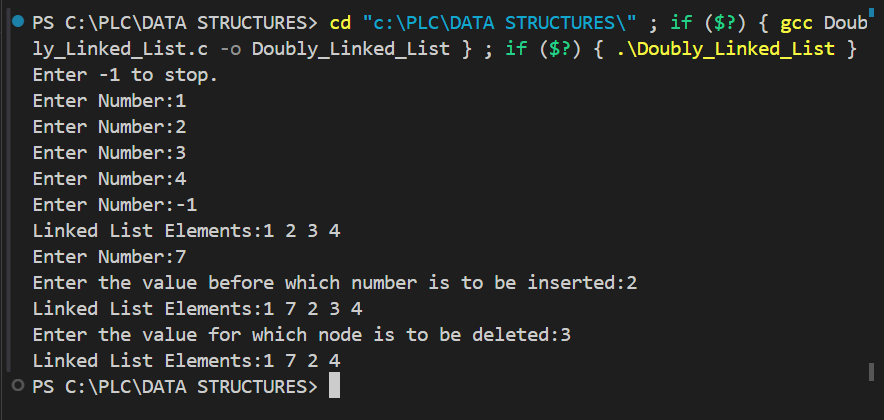
head=displayLL(head);

head=deleteNode(head);

head=displayLL(head);

}

**Output:**

****

**Lab Program 9:**

**Write a program**

**a) To construct a binary Search tree.**

**b) To traverse the tree using all the methods i.e., in-order, preorder and post order**

**c) To display the elements in the tree.**

**Also perform finding the immediate predecessor  and immediate successor in inorder traversal using BST.**

#include <stdio.h>

#include <stdlib.h>

struct node {

struct node\* left;

int data;

struct node\* right;

};

struct node\* CreateNode(int ele) {

struct node\* nn = (struct node\*)malloc(sizeof(struct node));

if (nn == NULL) {

printf("Memory Can't be allocated");

}

else {

nn->data = ele;

nn->left = NULL;

nn->right = NULL;

return nn;

}

}

struct node\* insert(struct node\* root, int data) {

if (root == NULL) {

root = CreateNode(data);

}

else if (data >= root->data) {

root->right = insert(root->right, data);

}

else if (data < root->data) {

root->left = insert(root->left, data);

}

return root;

}

void inordertrav(struct node\* root) {

if (root == NULL) {

return;

}

inordertrav(root->left);

printf("%d ", root->data);

inordertrav(root->right);

}

void postordertrav(struct node\* root) {

if (root == NULL) {

return;

}

postordertrav(root->left);

postordertrav(root->right);

printf("%d ", root->data);

}

void preordertrav(struct node\* root) {

if (root == NULL) {

return;

}

printf("%d ", root->data);

preordertrav(root->left);

preordertrav(root->right);

}

struct node\* findImmediatePredecessor(struct node\* root, int key) {

struct node\* pre = NULL;

while (root) {

if (root->data < key) {

pre = root;

root = root->right;

}

else if (root->data >= key) {

root = root->left;

}

}

return pre;

}

struct node\* findImmediateSuccessor(struct node\* root, int key) {

struct node\* suc = NULL;

while (root) {

if (root->data > key) {

suc = root;

root = root->left;

}

else if (root->data <= key) {

root = root->right;

}

}

return suc;

}

int main() {

struct node\* root = NULL;

int data,key;

root = insert(root, 14);

root = insert(root, 5);

root = insert(root, 44);

root = insert(root, 3);

root = insert(root, 7);

root = insert(root, 100);

root = insert(root, 46);

root = insert(root, 8);

root = insert(root, 10);

root = insert(root, 11);

root = insert(root, 17);

root = insert(root, 25);

root = insert(root, 23);

root = insert(root, 34);

root = insert(root, 16);

root = insert(root, 50);

root = insert(root, 1);

root = insert(root, 6);

root = insert(root, 2);

printf("In Order Traversal: ");

inordertrav(root);

printf("\n");

printf("Post Order Traversal: ");

postordertrav(root);

printf("\n");

printf("Pre Order Traversal: ");

preordertrav(root);

printf("\n");

printf("Enter Key to search:");

scanf("%d",&key);

struct node\* pre = findImmediatePredecessor(root, key);

struct node\* suc = findImmediateSuccessor(root, key);

if (pre)

printf("Immediate Predecessor of %d is %d\n", key, pre->data);

else

printf("No Immediate Predecessor of %d\n", key);

if (suc)

printf("Immediate Successor of %d is %d\n", key, suc->data);

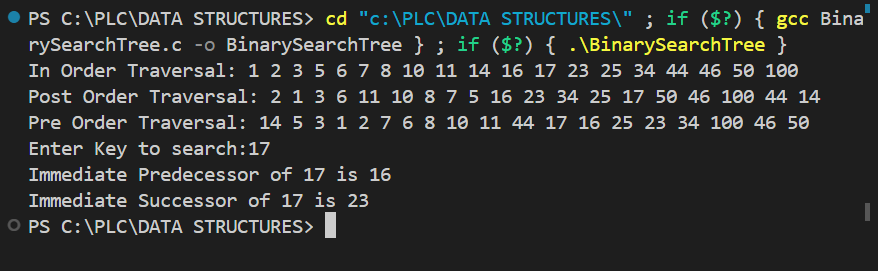
else

printf("No Immediate Successor of %d\n", key);

return 0;

}

**Output:**

****

**Lab Program 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 -&gt; L as H(K)=K mod m**

**(remainder method), and implement hashing technique to map a given key K to the address**

**space L.**

**Resolve the collision (if any) using**

**i. linear probing**

**ii. Quadratic Probing**

**iii. Double Hashing**

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 100

int L[MAX\_SIZE];

int count = 0;

int hash\_lprobe(int key) {

int i = 0;

while (L[(key + i) % MAX\_SIZE] != 0) {

i++;

if (count == MAX\_SIZE) {

printf("Array is full\n");

return -1;

}

}

count++;

return (key + i) % MAX\_SIZE;

}

int hash\_qprobe(int key) {

int i = 0;

while (L[(key + i \* i) % MAX\_SIZE] != 0) {

i++;

if (i == MAX\_SIZE)

return -1;

if (count == MAX\_SIZE) {

printf("Array is full\n");

return -1;

}

}

count++;

return (key + i \* i) % MAX\_SIZE;

}

int double\_hash(int key) {

int i = 0;

while (L[(key % MAX\_SIZE + 97 - key % 97) % MAX\_SIZE] != 0) {

i++;

if (count == MAX\_SIZE) {

printf("Array is full\n");

return -1;

}

}

count++;

return (key % MAX\_SIZE + 97 - key % 97) % MAX\_SIZE;

}

int search\_lp(int key) {

int i = 0;

while (L[(key + i) % MAX\_SIZE] != key) {

if (i == MAX\_SIZE) {

printf("Value doesn't exist\n");

return -1;

}

i++;

}

return (key + i) % MAX\_SIZE;

}

int search\_qp(int key) {

int i = 0;

while (L[(key + i \* i) % MAX\_SIZE] != key) {

if (i == MAX\_SIZE) {

printf("Value doesn't exist\n");

return -1;

}

i++;

}

return (key + i \* i) % MAX\_SIZE;

}

int search\_db(int key) {

int i = 0;

while (L[(key % MAX\_SIZE + 97 - key % 97) % MAX\_SIZE] != key) {

if (i == MAX\_SIZE) {

printf("Value doesn't exist\n");

return -1;

}

i++;

}

return (key % MAX\_SIZE + 97 - key % 97) % MAX\_SIZE;

}

int main() {

printf("1)Insert\n2)Search\n3)Exit\n");

while (1) {

int choice, subChoice, key;

printf("Enter choice:");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the key:");

scanf("%d", &key);

printf("1)Linear\n2)Quadratic\n3)Double Hash\n");

scanf("%d", &subChoice);

if (subChoice == 1)

key = hash\_lprobe(key);

else if (subChoice == 2)

key = hash\_qprobe(key);

else

key = double\_hash(key);

L[key] = key;

if (key != -1)

printf("Value %d inserted\n", key);

break;

case 2:

printf("Enter the key:");

scanf("%d", &key);

printf("1)Linear\n2)Quadratic\n3)Double Hash\n");

scanf("%d", &subChoice);

if (subChoice == 1)

key = search\_lp(key);

else if (subChoice == 2)

key = search\_qp(key);

else

key = search\_db(key);

if (key != -1)

printf("Value %d found at %d\n", L[key], key);

break;

case 3:

exit(0);

default:

printf("Invalid choice\n");

break;

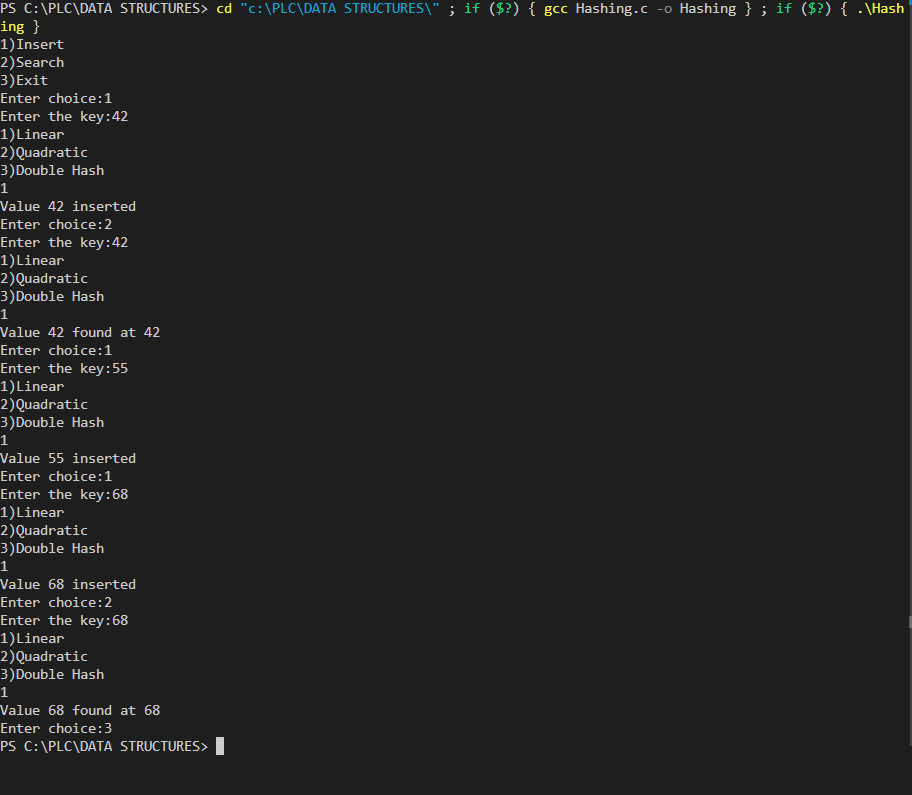
}

}

return 0;

}

**Output:**

****

**LeetCode Questions**

**Question 1:**

856. Score of Parentheses Solved Medium Topics Companies Given a balanced parentheses string s, return the score of the string. The score of a balanced parentheses string is based on the following rule: "()" has score 1. AB has score A + B, where A and B are balanced parentheses strings. (A) has score 2 \* A, where A is a balanced parentheses string. Example 1: Input: s = "()" Output: 1 Example 2: Input: s = "(())" Output: 2 Example 3: Input: s = "()()" Output: 2 Constraints: 2 <= s.length <= 50 s consists of only '(' and ')'. s is a balanced parentheses string.

**Solution :**

#include <stdio.h>

#include <stdlib.h>

int scoreOfParentheses(char \*s) {

int score = 0;

int depth = 0;

for (int i = 0; s[i] != '\0'; i++) {

if (s[i] == '(') {

depth++;

} else {

depth--;

if (s[i - 1] == '(') {

score += 1 << depth;

}

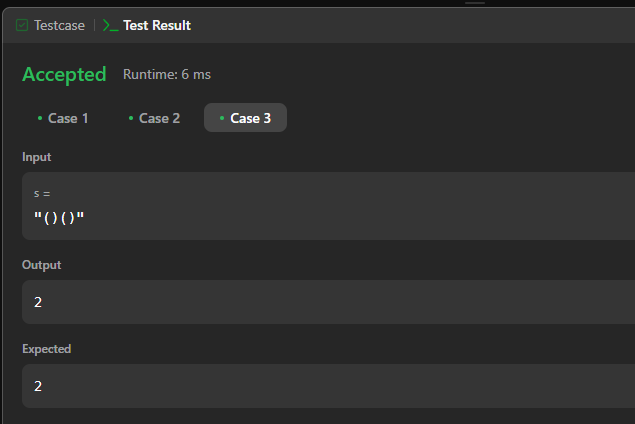
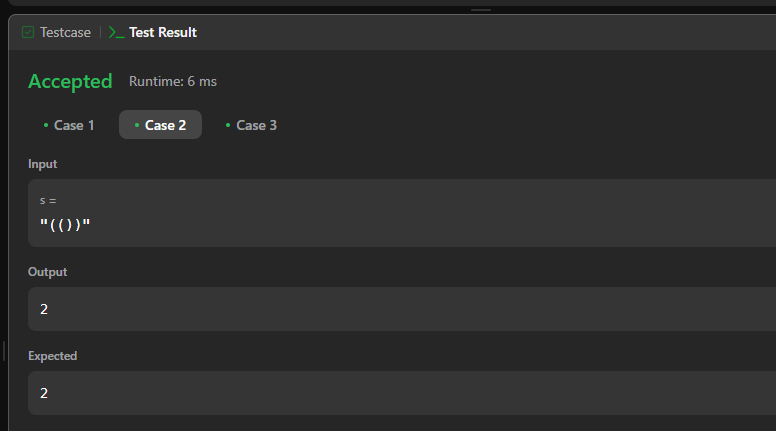
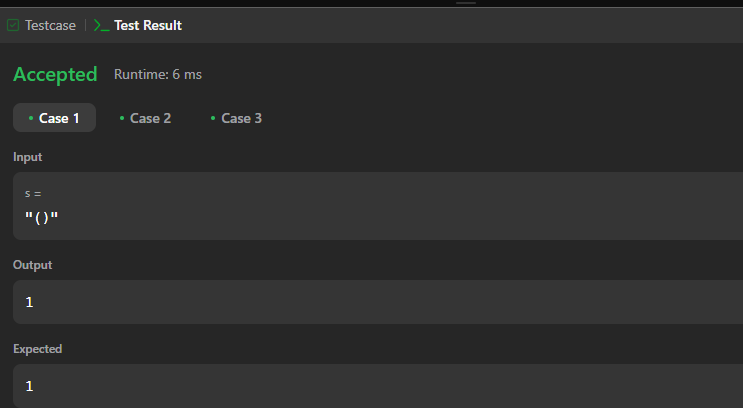
}

}

return score;

}

**Output:**



**Question 2:**

**2095. Delete the Middle Node of a Linked List**

**Hint: You are given the head of a linked list. Delete the middle node, and return the head of the modified linked list.**

**The middle node of a linked list of size n is the ⌊n / 2⌋th node from the start using 0-based indexing, where ⌊x⌋ denotes the largest integer less than or equal to x.**

**For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively.**

**Example 1:**

**Input: head = [1,3,4,7,1,2,6]**

**Output: [1,3,4,1,2,6]**

**Explanation:**

**The above figure represents the given linked list. The indices of the nodes are written below.**

**Since n = 7, node 3 with value 7 is the middle node, which is marked in red.**

**We return the new list after removing this node.**

**Example 2:**

**Input: head = [1,2,3,4]**

**Output: [1,2,4]**

**Explanation:**

**The above figure represents the given linked list.**

**For n = 4, node 2 with value 3 is the middle node, which is marked in red.**

**Example 3:**

**Input: head = [2,1]**

**Output: [2]**

**Explanation:**

**The above figure represents the given linked list.**

**For n = 2, node 1 with value 1 is the middle node, which is marked in red.**

**Node 0 with value 2 is the only node remaining after removing node 1.**

**Constraints:**

**The number of nodes in the list is in the range [1, 105].**

**1 <= Node.val <= 105**

**Solution:**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* deleteMiddle(struct ListNode\* head){

if(!head->next) return NULL;

struct ListNode \*fast = head->next;

struct ListNode \*slow = head;

while(fast && fast->next){

fast = fast->next->next;

if(!fast) break;

slow = slow->next;

}

struct ListNode \*q = slow->next;

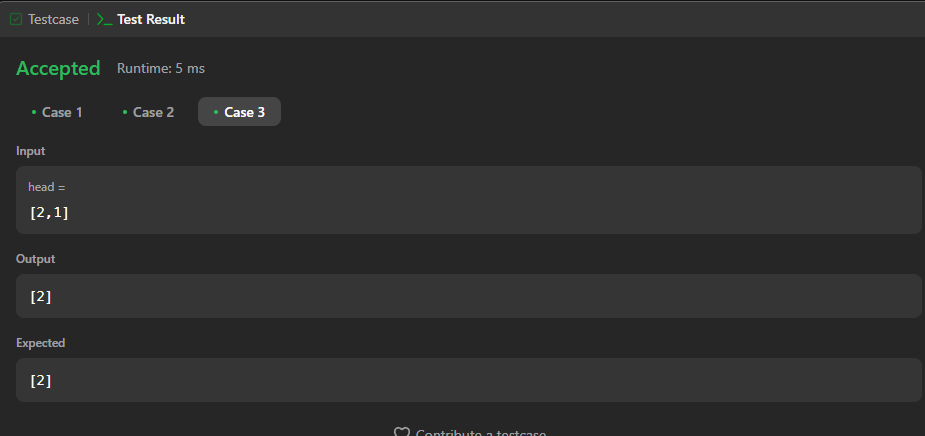
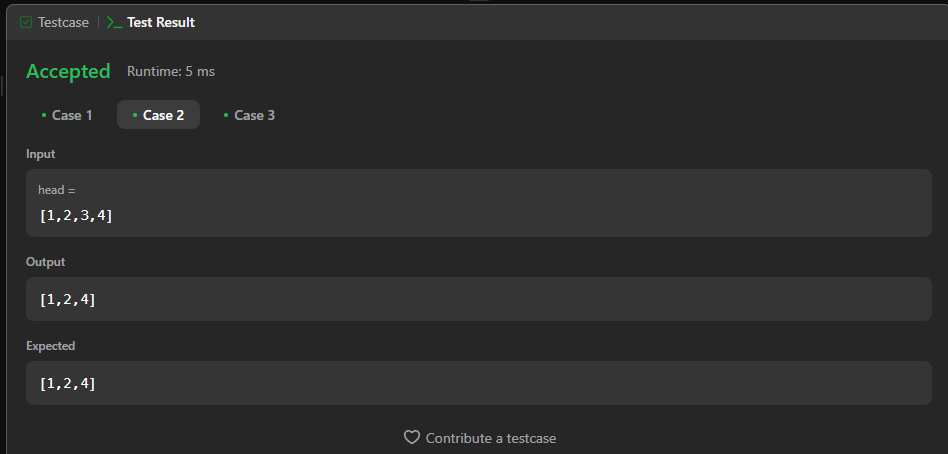
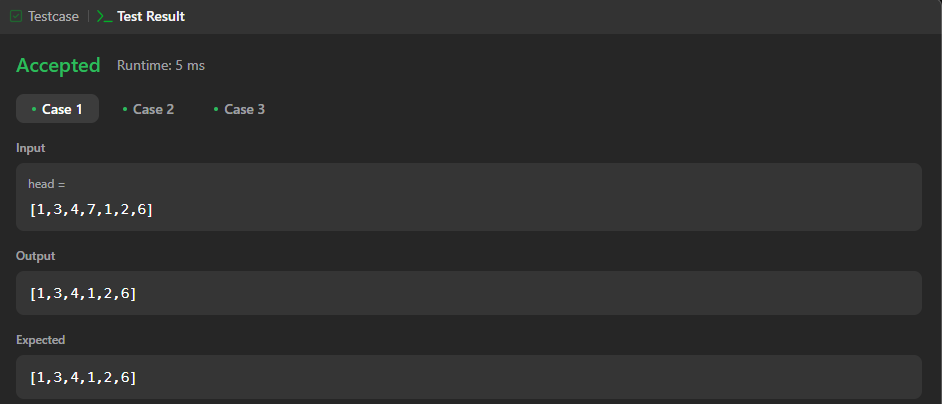
slow->next = slow->next->next;

free(q);

return head;

}

**Output:**



**Question 3:**

**328. Odd Even Linked List**

**Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.**

**The first node is considered odd, and the second node is even, and so on.**

**Note that the relative order inside both the even and odd groups should remain as it was in the input.**

**You must solve the problem in O(1) extra space complexity and O(n) time complexity.**

**Example 1:**

**Input: head = [1,2,3,4,5]**

**Output: [1,3,5,2,4]**

**Example 2:**

**Input: head = [2,1,3,5,6,4,7]**

**Output: [2,3,6,7,1,5,4]**

**Constraints:**

**The number of nodes in the linked list is in the range [0, 104].**

**-106 <= Node.val <= 106**

**Solution:**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* };

\*/

struct ListNode\* oddEvenList(struct ListNode\* head) {

if (head == NULL || head->next == NULL) {

return head;

}

struct ListNode\* odd = (struct ListNode\*)malloc(sizeof(struct ListNode));

odd->val = 0;

odd->next = NULL;

struct ListNode\* odd\_ptr = odd;

struct ListNode\* even = (struct ListNode\*)malloc(sizeof(struct ListNode));

even->val = 0;

even->next = NULL;

struct ListNode\* even\_ptr = even;

int idx = 1;

while (head != NULL) {

if (idx % 2 == 0) {

even\_ptr->next = head;

even\_ptr = even\_ptr->next;

} else {

odd\_ptr->next = head;

odd\_ptr = odd\_ptr->next;

}

head = head->next;

idx++;

}

even\_ptr->next = NULL;

odd\_ptr->next = even->next;

struct ListNode\* newHead = odd->next;

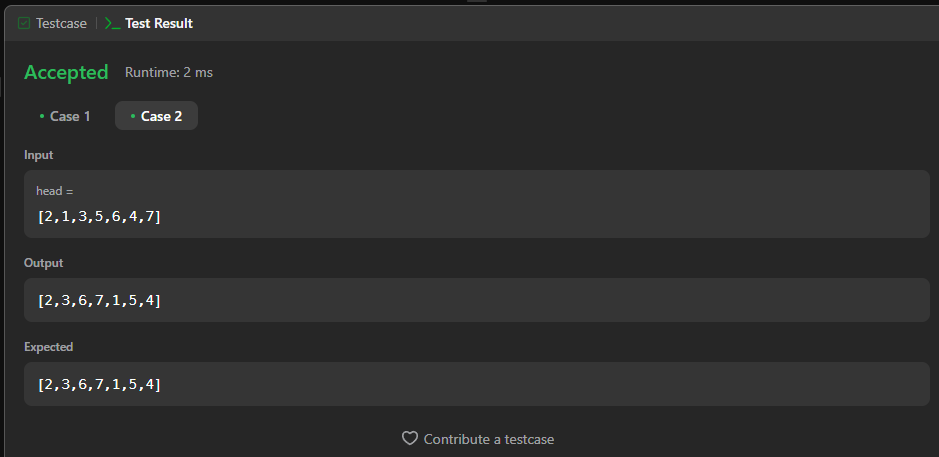
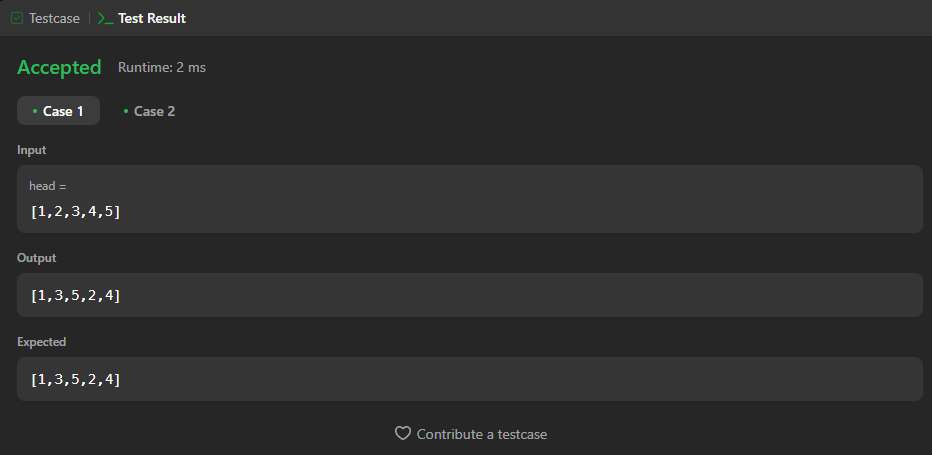
free(odd);

free(even);

return newHead;

}

**Output:**



**Question 4:**

**450. Delete Node in a BST**

**Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.**

**Basically, the deletion can be divided into two stages:**

**Search for a node to remove.**

**If the node is found, delete the node.**

**Example 1:**

**Input: root = [5,3,6,2,4,null,7], key = 3**

**Output: [5,4,6,2,null,null,7]**

**Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.**

**One valid answer is [5,4,6,2,null,null,7], shown in the above BST.**

**Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.**

**Example 2:**

**Input: root = [5,3,6,2,4,null,7], key = 0**

**Output: [5,3,6,2,4,null,7]**

**Explanation: The tree does not contain a node with value = 0.**

**Example 3:**

**Input: root = [], key = 0**

**Output: []**

**Constraints:**

**The number of nodes in the tree is in the range [0, 104].**

**-105 <= Node.val <= 105**

**Each node has a unique value.**

**root is a valid binary search tree.**

**-105 <= key <= 105**

**Solution:**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

struct TreeNode\* smallest(struct TreeNode\* root)

{

struct TreeNode \* cur=root;

while(cur->left!=NULL)

{

cur=cur->left;

}

return cur;

}

struct TreeNode\* deleteNode(struct TreeNode\* root, int key)

{

// base case

if(root==NULL)

{

return root;

}

// If the key to be deleted is smallerthan the root's key,

// then it lies in left subtree

if(key < root->val)

{

root->left = deleteNode(root->left,key);

}

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

else if(key > root->val)

{

root->right=deleteNode(root->right,key);

}

// if key is same as root's key, then This is the node

// to be deleted

else

{

// node with only one child or no child

if (root->left == NULL)

{

struct TreeNode \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

struct TreeNode \*temp = root->left;

free(root);

return temp;

}

struct TreeNode \*temp = smallest(root->right);

root->val = temp->val;

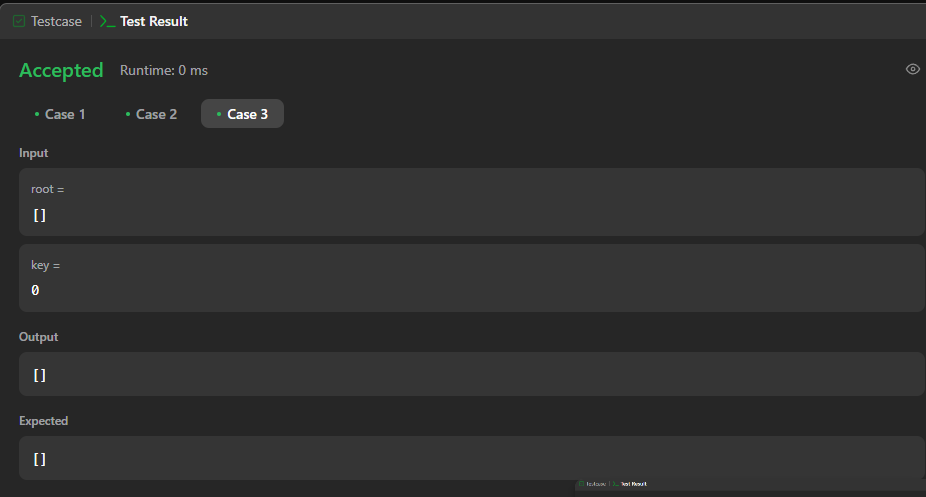
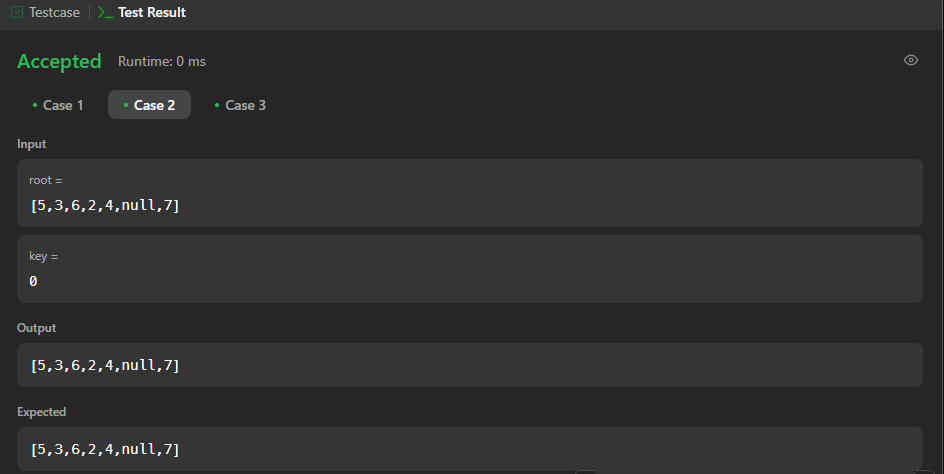
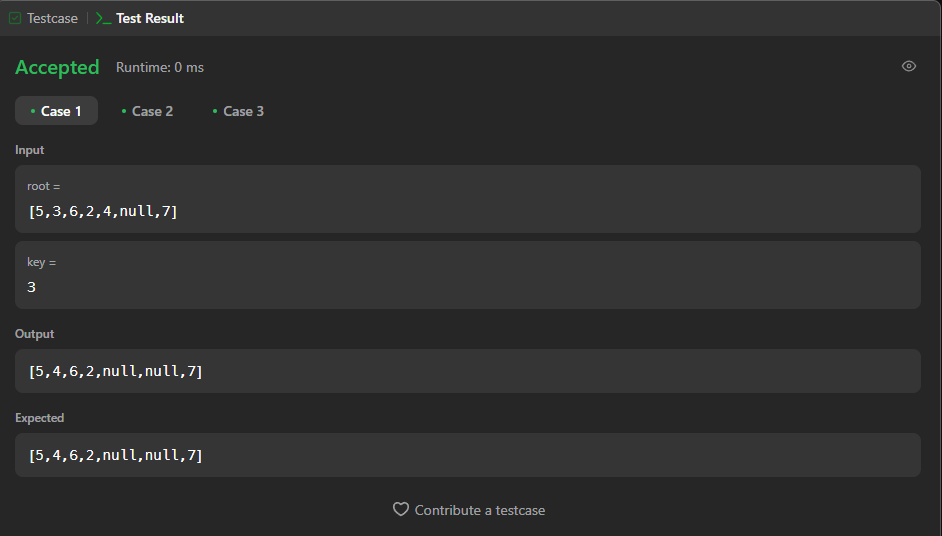
root->right = deleteNode(root->right, root->val);

}

return root;

}

**Output:**



**Question 5:**

**513. Find Bottom Left Tree Value**

**Given the root of a binary tree, return the leftmost value in the last row of the tree.**

**Example 1:**

**Input: root = [2,1,3]**

**Output: 1**

**Example 2:**

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

**Output: 7**

**Constraints:**

**The number of nodes in the tree is in the range [1, 104].**

**-231 <= Node.val <= 231 - 1**

**Solution:**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* struct TreeNode \*left;

\* struct TreeNode \*right;

\* };

\*/

void find(struct TreeNode\* root,int \*maxdepth,int depth,int \*val){

if(!root) return;

if(\*maxdepth < depth){

\*maxdepth=depth;

\*val=root->val;

}

find(root->left,maxdepth,depth+1,val);

find(root->right,maxdepth,depth+1,val);

}

int findBottomLeftValue(struct TreeNode\* root){

int maxdepth=-1;

int val=0;

find(root,&maxdepth,0,&val);

return val;

}

**Output:**

