**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“Jnana Sangama”, Belgaum -590014, Karnataka.**



**LAB REPORT On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

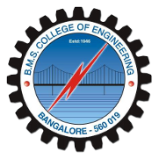
**Tutari Likitha (1BM22CS312 )**

**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 Tutari Likitha **(1BM22CS312)**, 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 202324. 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** | **Dr. Jyothi S Nayak** |
| Assistant Professor | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |
| **Index** |  |

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Stacks | 4-5 |
| 2 | Stack Infix exp | 6-8 |
| 3 | Queues | 8-10 |
| 4 | Circular Queue | 10-12 |
| 5 | Singly Linked List-Insertion & Deletion | 12-17 |
| 6 | Singly Linked List-sort,reverse,concatenate | 17-19 |
| 7 | 1. Stack implementation using SLL. 2. Queue implementation using SLL. | 20-21 |
| 8 | Doubly Linked List | 22-24 |
| 9 | Binary search tree | 24-26 |
| 10 | BFS & DFS | 26-30 |
| 11 | Leet Codes | 31-36 |

**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>

#define max 5

int stack[max];

int top = -1;

void push(int val) {

if (top == max - 1) {

printf("Stack Overflow");

} else {

top++;

stack[top] = val;

printf("Successfully Added");

}

}

void pop() {

if (top == -1) {

printf("Stack Underflow");

} else {

printf("%d has been removed from stack", stack[top]);

top--;

}

}

void display() {

if (top == -1) {

printf("Stack Underflow");

} else {

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

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

}

}

}

void main() {

int choice, boolean = 1, val;

while (boolean) {

printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &val);

push(val);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

boolean = 0;

break;

default:

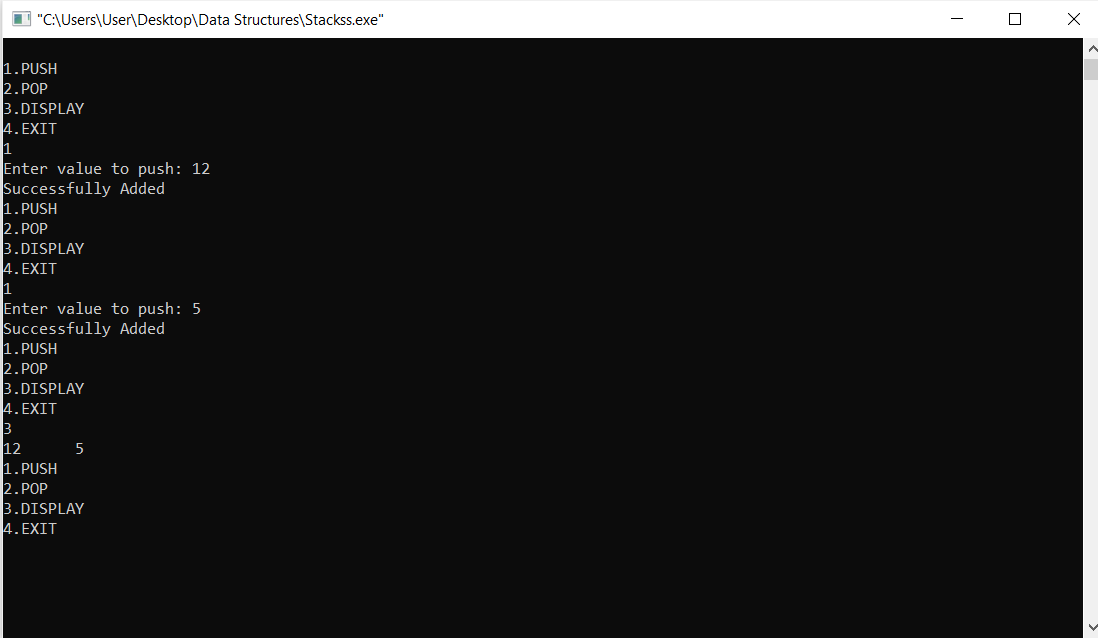
printf("Invalid Input");

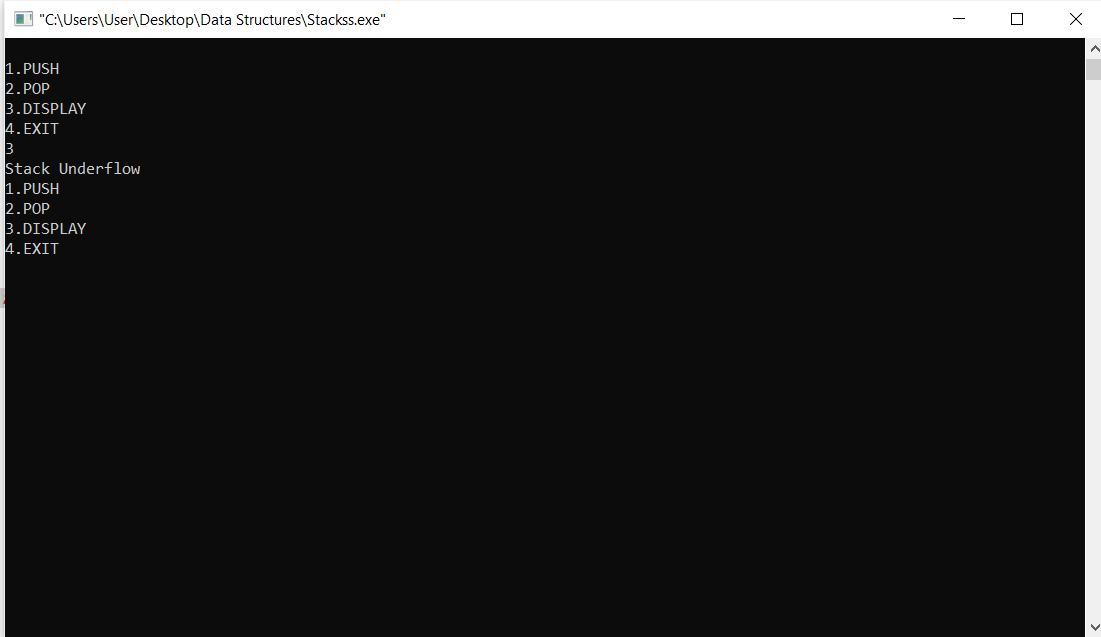
}

}

}

**Output:**





**Lab program 2:**

**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)**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define max 100

char stack[max], postfix[max], infix[max];

int top = -1;

void push(char symbol);

char pop();

int isEmpty();

int precedence(char symbol);

void display();

void intopos();

int main() {

printf("Enter Infix Expression: ");

gets(infix);

intopos();

display();

return 0;

}

void intopos() {

int i = 0, j = 0;

char symbol, next;

for (i = 0; i < strlen(infix); i++) {

symbol = infix[i];

switch (symbol) {

case '(':

push(symbol);

break;

case ')':

while ((next = pop()) != '(') {

postfix[j++] = next;

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

while (!isEmpty() && precedence(stack[top]) >= precedence(symbol)) {

postfix[j++] = pop();

}

push(symbol);

break;

default:

postfix[j++] = symbol;

break;

}

}

while (!isEmpty()) {

postfix[j++] = pop();

}

postfix[j] = '\0'; // Null-terminate the postfix expression

}

void push(char symbol) {

if (top == max - 1) {

printf("Stack Overflow");

return;

}

stack[++top] = symbol;

}

char pop() {

if (isEmpty()) {

printf("Stack Underflow");

return '\0';

}

return stack[top--];

}

int isEmpty() {

return top == -1;

}

int precedence(char symbol) {

switch (symbol) {

case '^':

return 3;

case '\*':

case '/':

return 2;

case '+':

case '-':

return 1;

default:

return 0;

}

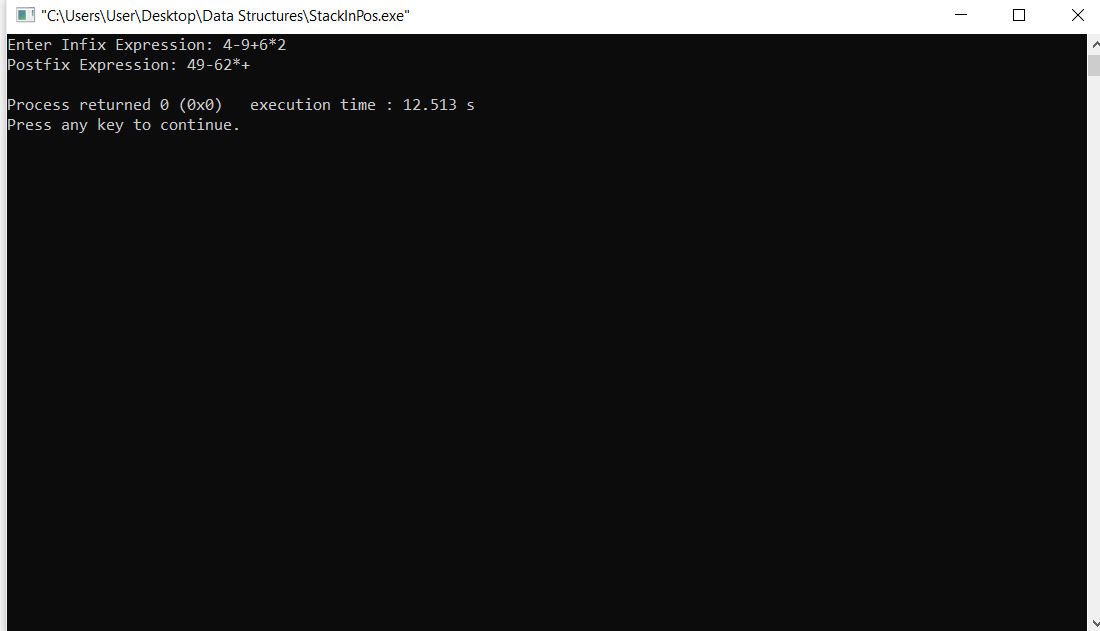
}

void display() {

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

}

**Output:**



**Lab program 3:**

**write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate message for overflow and underflow condition**.

#include<stdio.h>

#define max 5

int front = -1, rear = -1, q[max];

void enqueue(int value) {

if (rear == -1 && front == -1) {

front = rear = 0;

q[rear] = value;

} else if (rear == max - 1) {

printf("Overflow");

} else {

q[++rear] = value;

}

}

void dequeue() {

if (front == -1) {

printf("Underflow");

} else {

printf("%d ", q[front]);

if (front == rear) {

front = rear = -1;

} else {

front++;

}

}

}

void display() {

if (front == -1) {

printf("Underflow");

} else {

for (int i = front; i <= rear; i++) {

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

}

}

}

int main() {

int boolean = 1, choice, value;

while (boolean) {

printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

boolean = 0;

break;

default:

printf("Invalid input");

break;

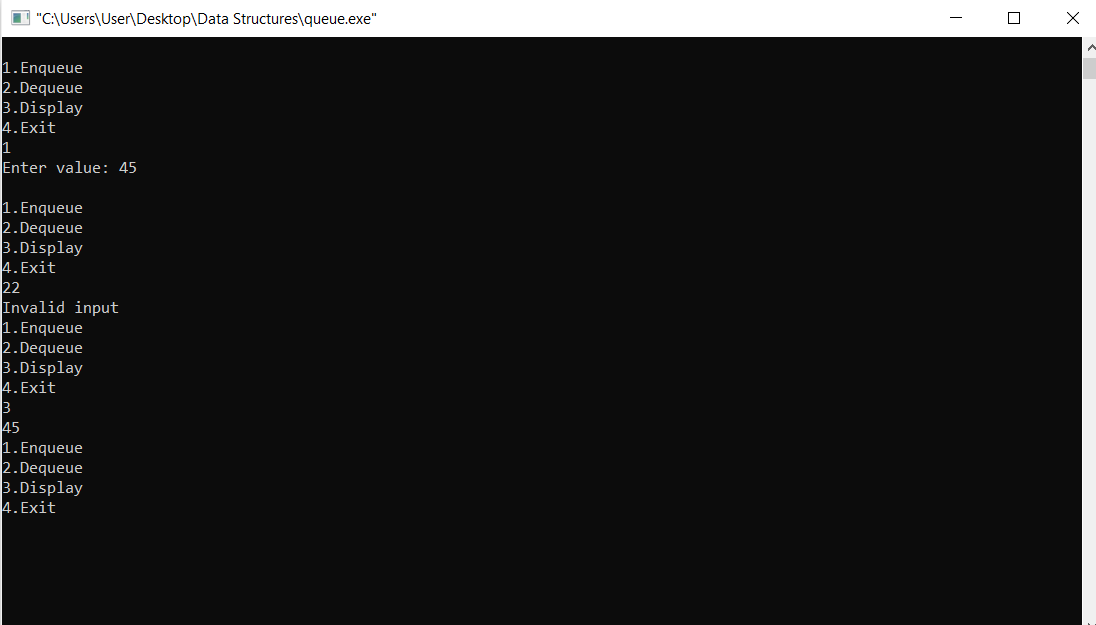
}

}

return 0;

}

**Output:**



**Lab program 4:**

**write a program to simulate the working of a circular queue using an array. Provide the following operations: insert, delete& display. The program should print appropriate message for queue empty and queue overflow conditions.**

#include<stdio.h>

#define max 5

int front = -1, rear = -1, q[max];

void enqueue(int value) {

if ((rear + 1) % max == front) {

printf("Overflow\n");

} else if (front == -1 && rear == -1) {

front = rear = 0;

q[rear] = value;

} else {

rear = (rear + 1) % max;

q[rear] = value;

}

}

void dequeue() {

if (front == -1) {

printf("Underflow\n");

} else if (front == rear) {

printf("%d\n", q[front]);

front = rear = -1;

} else {

printf("%d\n", q[front]);

front = (front + 1) % max;

}

}

void display() {

if (front == -1) {

printf("Underflow\n");

} else {

int i = front;

do {

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

i = (i + 1) % max;

} while (i != (rear + 1) % max);

printf("\n");

}

}

int main() {

int boolean = 1, choice, value;

while (boolean) {

printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

boolean = 0;

break;

default:

printf("Invalid input\n");

break;

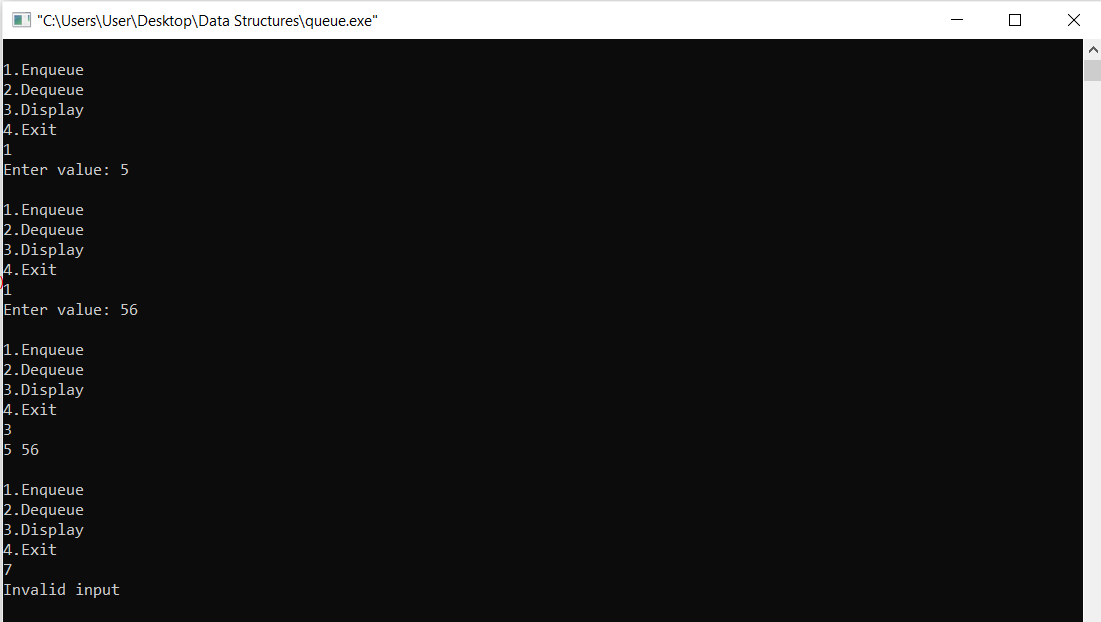
}

}

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, at any position and at end of list.  
c)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;

};

void print(struct node\* head){

if(head == NULL){

printf("List is Empty\n");

} else{

struct node \*ptr = head;

while(ptr != NULL){

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

ptr = ptr->next;

}

}

}

void insertBeg(struct node \*\*head){

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

printf("Enter data: ");

scanf("%d", &newnode->data);

newnode->next = \*head;

\*head = newnode;

}

void insertEnd(struct node \*head){

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

printf("Enter data: ");

scanf("%d", &newnode->data);

newnode->next = NULL;

if(head == NULL){

head = newnode;

} else{

struct node \*ptr = head;

while(ptr->next != NULL){

ptr = ptr->next;

}

ptr->next = newnode;

}

}

void insertpos(struct node \*\*head, int pos){

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

printf("Enter pos: ");

scanf("%d", &pos);

int count = 0;

struct node \*ptr = \*head;

while(ptr != NULL){

count++;

ptr = ptr->next;

}

if(pos > count){

printf("Invalid pos\n");

} else{

ptr = \*head;

struct node \*prev = NULL;

for(int i = 1; i < pos; i++){

prev = ptr;

ptr = ptr->next;

}

printf("Enter data: ");

scanf("%d", &newnode->data);

newnode->next = ptr;

if(prev == NULL){

\*head = newnode;

} else{

prev->next = newnode;

}

}

}

void deletebeg(struct node \*\*head){

if(\*head == NULL){

printf("List is Empty\n");

} else{

struct node \*ptr = \*head;

\*head = (\*head)->next;

free(ptr);

}

}

void delend(struct node \*head){

if(head == NULL){

printf("List is Empty\n");

} else if(head->next == NULL){

free(head);

head = NULL;

} else{

struct node \*ptr = head;

struct node \*prev = NULL;

while(ptr->next != NULL){

prev = ptr;

ptr = ptr->next;

}

prev->next = NULL;

free(ptr);

}

}

void delatpos(struct node \*\*head, int pos){

printf("Enter pos: ");

scanf("%d", &pos);

if(\*head == NULL){

printf("List is Empty\n");

} else{

struct node \*ptr = \*head;

struct node \*prev = NULL;

for(int i = 1; i < pos && ptr != NULL; i++){

prev = ptr;

ptr = ptr->next;

}

if(ptr == NULL){

printf("Invalid pos\n");

} else if(prev == NULL){

\*head = ptr->next;

free(ptr);

} else{

prev->next = ptr->next;

free(ptr);

}

}

}

int main(){

struct node \*head = NULL;

int choice, pos;

while(1){

printf("\n1. Insert at Beginning\n");

printf("2. Insert at End\n");

printf("3. Insert at Position\n");

printf("4. Delete from Beginning\n");

printf("5. Delete from End\n");

printf("6. Delete from Position\n");

printf("7. Print List\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice){

case 1:

insertBeg(&head);

break;

case 2:

insertEnd(head);

break;

case 3:

insertpos(&head, pos);

break;

case 4:

deletebeg(&head);

break;

case 5:

delend(head);

break;

case 6:

delatpos(&head, pos);

break;

case 7:

print(head);

break;

case 8:

exit(0);

default:

printf("Invalid Choice\n");

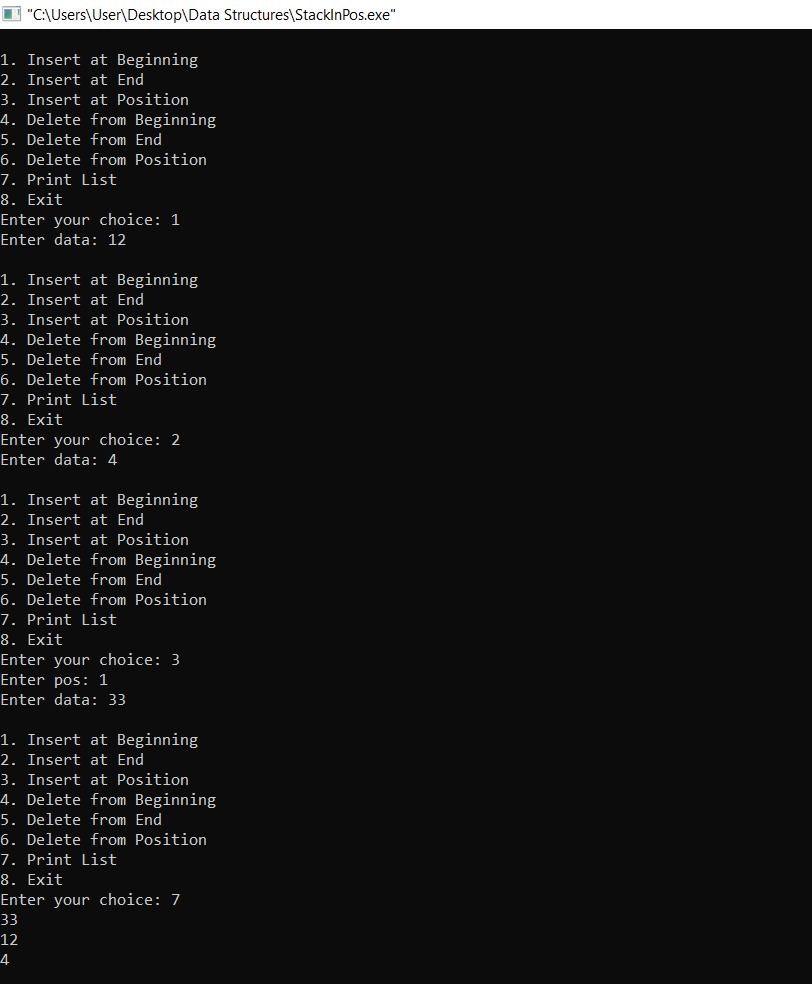
}

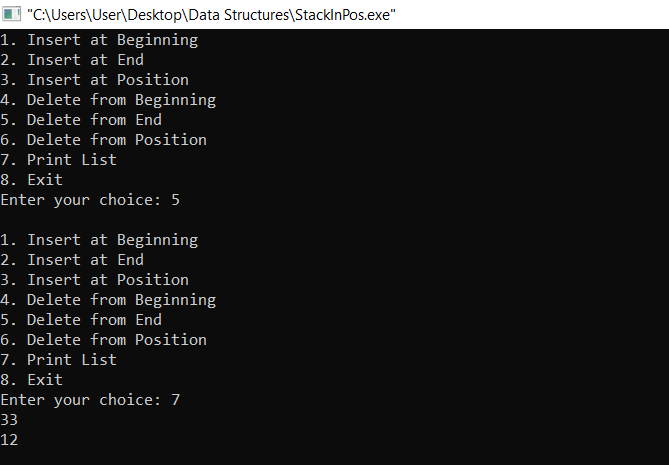
}

return 0;

}

**Output:**





**Lab program 6:**

**sll- sort, reverse, concatenation**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

void printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

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

struct Node \*prev = NULL, \*curr = head, \*next = NULL;

while (curr != NULL) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

return prev;

}

void sortList(struct Node\*\* head) {

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

return;

}

int swapped;

struct Node\* ptr1;

struct Node\* lptr = NULL;

do {

swapped = 0;

ptr1 = \*head;

while (ptr1->next != lptr) {

if (ptr1->data > ptr1->next->data) {

int temp = ptr1->data;

ptr1->data = ptr1->next->data;

ptr1->next->data = temp;

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

} while (swapped);

}

void concatenateLists(struct Node\*\* list1, struct Node\* list2) {

if (\*list1 == NULL) {

\*list1 = list2;

} else {

struct Node\* temp = \*list1;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = list2;

}

}

int main() {

struct Node\* head1 = NULL;

struct Node\* head2 = NULL;

insertAtEnd(&head1, 10);

insertAtEnd(&head1, 30);

insertAtEnd(&head1, 20);

printf("First Linked List: ");

printList(head1);

insertAtEnd(&head2, 5);

insertAtEnd(&head2, 15);

insertAtEnd(&head2, 25);

printf("Second Linked List: ");

printList(head2);

sortList(&head1);

printf("Sorted First Linked List: ");

printList(head1);

head2 = reverseList(head2);

printf("Reversed Second Linked List: ");

printList(head2);

concatenateLists(&head1, head2);

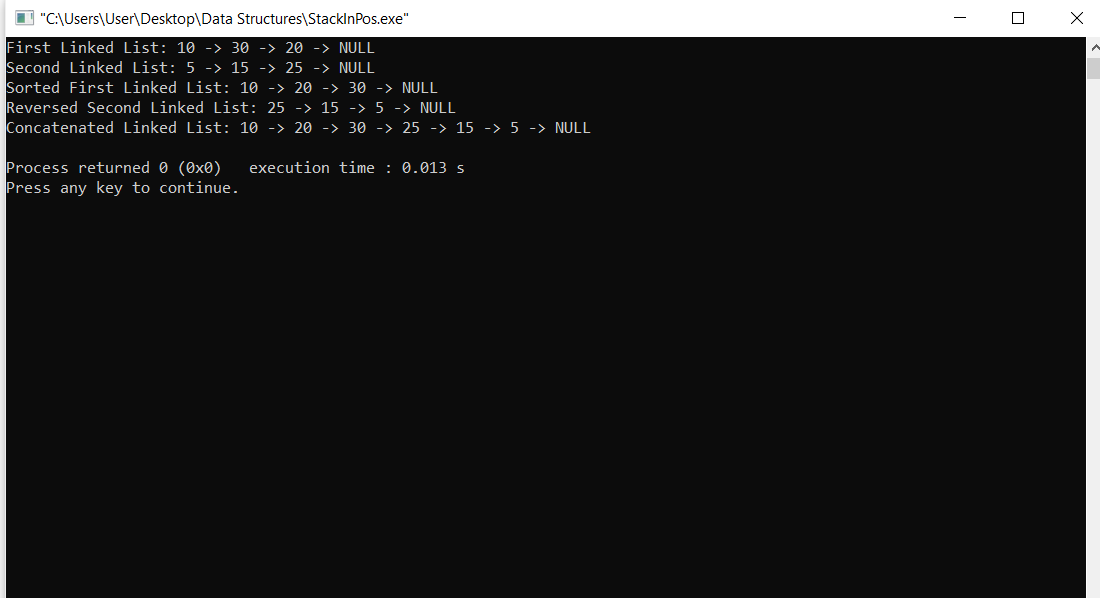
printf("Concatenated Linked List: ");

printList(head1);

return 0;

}

**Output:**



**Lab program 7:**

**a)Stack implementation using single linked list**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

newNode->next = \*top;

\*top = newNode;

printf("%d pushed to stack\n", data);

}

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

if (\*top == NULL) {

printf("Stack is empty\n");

exit(1);

}

struct Node\* temp = \*top;

int data = temp->data;

\*top = (\*top)->next;

free(temp);

return data;

}

void display(struct Node\* top) {

if (top == NULL) {

printf("Stack is empty\n");

return;

}

printf("Stack elements:\n");

while (top != NULL) {

printf("%d\n", top->data);

top = top->next;

}

}

int main() {

struct Node\* top = NULL;

int choice, data;

while (1) {

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 data to push: ");

scanf("%d", &data);

push(&top, data);

break;

case 2:

if (top != NULL)

printf("Popped element: %d\n", pop(&top));

break;

case 3:

display(top);

break;

case 4:

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

exit(0);

default:

printf("Invalid choice\n");

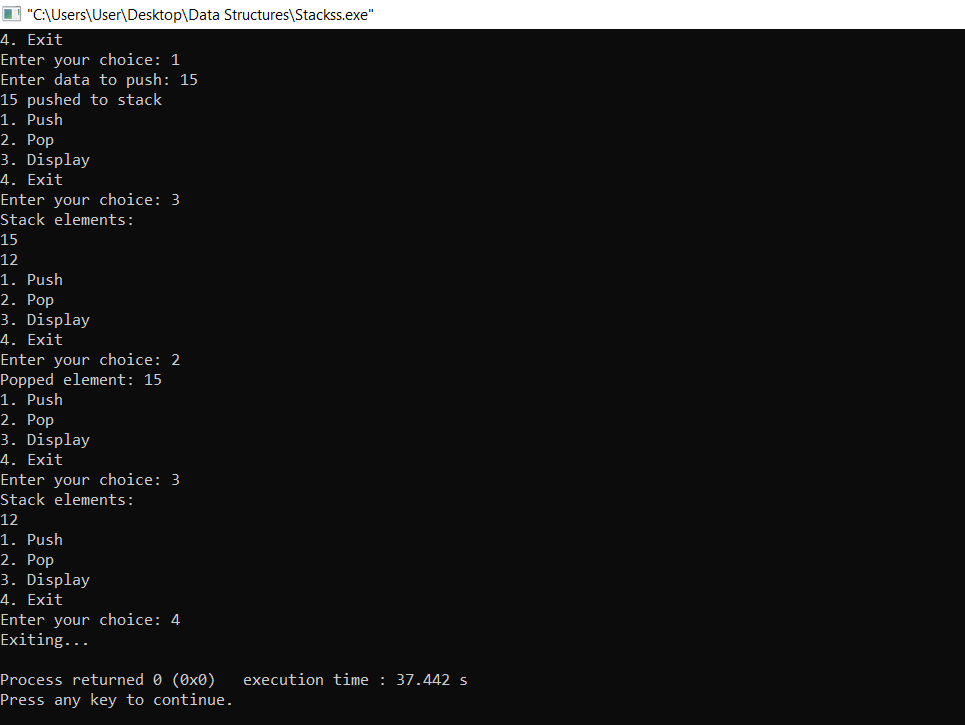
}

}

return 0;

}

**Output:**



**Lab program 7:**

**b)Queue implementation using single linked list**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Queue {

struct Node \*front, \*rear;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

if (queue == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

queue->front = queue->rear = NULL;

return queue;

}

int isEmpty(struct Queue\* queue) {

return (queue->front == NULL);

}

void enqueue(struct Queue\* queue, int data) {

struct Node\* newNode = createNode(data);

if (queue->rear == NULL) {

queue->front = queue->rear = newNode;

return;

}

queue->rear->next = newNode;

queue->rear = newNode;

}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

exit(1);

}

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) {

struct Node\* temp = queue->front;

if (isEmpty(queue)) {

printf("Queue is empty\n");

return;

}

printf("Queue elements:\n");

while (temp != NULL) {

printf("%d\n", temp->data);

temp = temp->next;

}

}

int main() {

struct Queue\* queue = createQueue();

int choice, data;

while (1) {

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 data to enqueue: ");

scanf("%d", &data);

enqueue(queue, data);

break;

case 2:

if (!isEmpty(queue))

printf("Dequeued element: %d\n", dequeue(queue));

break;

case 3:

display(queue);

break;

case 4:

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

exit(0);

default:

printf("Invalid choice\n");

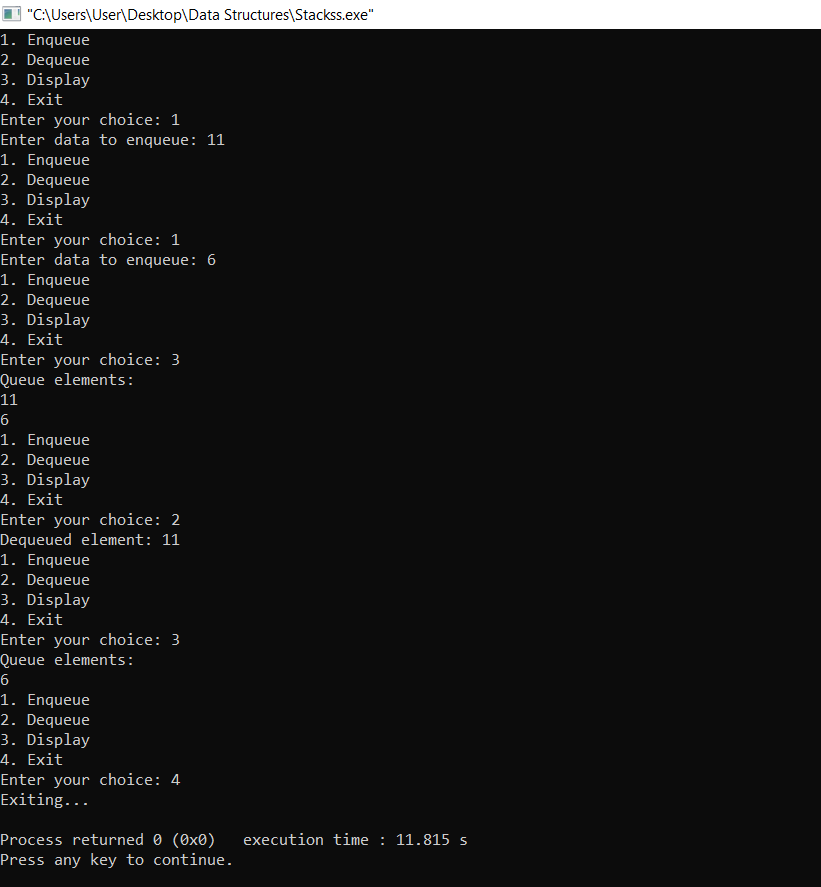
}

}

return 0;

}

**Output:**



**Lab program 8:**

**Doubly Linked List.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

void insertNode(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

void deleteNode(struct Node\*\* head, struct Node\* del) {

if (\*head == NULL || del == NULL) {

return;

}

if (\*head == del) {

\*head = del->next;

}

if (del->next != NULL) {

del->next->prev = del->prev;

}

if (del->prev != NULL) {

del->prev->next = del->next;

}

free(del);

}

void display(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("\n");

}

int main() {

struct Node\* head = NULL;

insertNode(&head, 10);

insertNode(&head, 20);

insertNode(&head, 30);

insertNode(&head, 40);

display(head);

struct Node\* nodeToDelete = head->next;

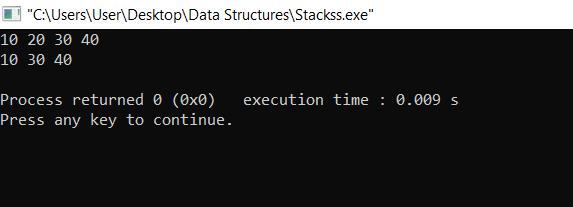
deleteNode(&head, nodeToDelete);

display(head);

return 0;

}

**Output:**



**Lab program 9:**

**Write a program.   
    a. To construct Binary Search tree  
    b. Traverse the tree using inorder , postorder, preorder.  
    c. Display the elements in the tree.**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

};

struct TreeNode\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

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

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

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

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

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

}

return root;

}

void inorderTraversal(struct TreeNode\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

void preorderTraversal(struct TreeNode\* root) {

if (root == NULL) {

return;

}

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct TreeNode\* root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

void display(struct TreeNode\* root) {

printf("Binary Search Tree Elements:\n");

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\nPreorder Traversal: ");

preorderTraversal(root);

printf("\nPostorder Traversal: ");

postorderTraversal(root);

printf("\n");

}

int main() {

struct TreeNode\* root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 70);

insert(root, 20);

insert(root, 40);

insert(root, 60);

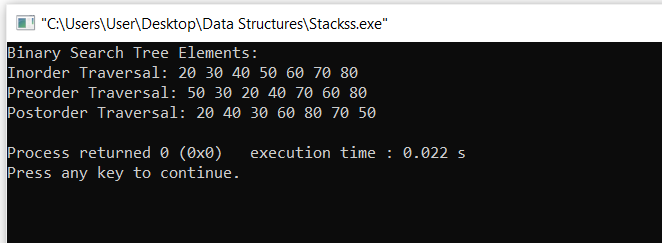
insert(root, 80);

display(root);

return 0;

}

**Output:**



**Lab program 10:**

**1.BFS  
2.DFS**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

struct AdjListNode {

int dest;

struct AdjListNode\* next;

};

struct AdjList {

struct AdjListNode\* head;

};

struct Graph {

int V;

struct AdjList\* array;

};

struct AdjListNode\* newAdjListNode(int dest) {

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

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int V) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->V = V;

graph->array = (struct AdjList\*)malloc(V \* sizeof(struct AdjList));

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

}

void BFS(struct Graph\* graph, int start) {

bool\* visited = (bool\*)malloc(graph->V \* sizeof(bool));

for (int i = 0; i < graph->V; ++i)

visited[i] = false;

int queue[MAX\_VERTICES];

int front = 0, rear = -1;

visited[start] = true;

queue[++rear] = start;

while (front <= rear) {

int current = queue[front++];

printf("%d ", current);

struct AdjListNode\* temp = graph->array[current].head;

while (temp != NULL) {

int adj = temp->dest;

if (!visited[adj]) {

visited[adj] = true;

queue[++rear] = adj;

}

temp = temp->next;

}

}

free(visited);

}

void DFSUtil(struct Graph\* graph, int current, bool visited[]) {

visited[current] = true;

printf("%d ", current);

struct AdjListNode\* temp = graph->array[current].head;

while (temp != NULL) {

int adj = temp->dest;

if (!visited[adj])

DFSUtil(graph, adj, visited);

temp = temp->next;

}

}

void DFS(struct Graph\* graph, int start) {

bool\* visited = (bool\*)malloc(graph->V \* sizeof(bool));

for (int i = 0; i < graph->V; ++i)

visited[i] = false;

DFSUtil(graph, start, visited);

free(visited);

}

int main() {

int V = 5;

struct Graph\* graph = createGraph(V);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

printf("Breadth First Search (BFS): ");

BFS(graph, 0);

printf("\n");

printf("Depth First Search (DFS): ");

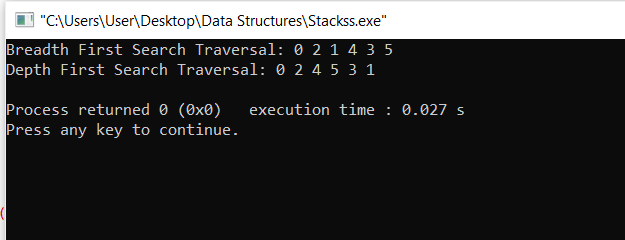
DFS(graph, 0);

printf("\n");

return 0;

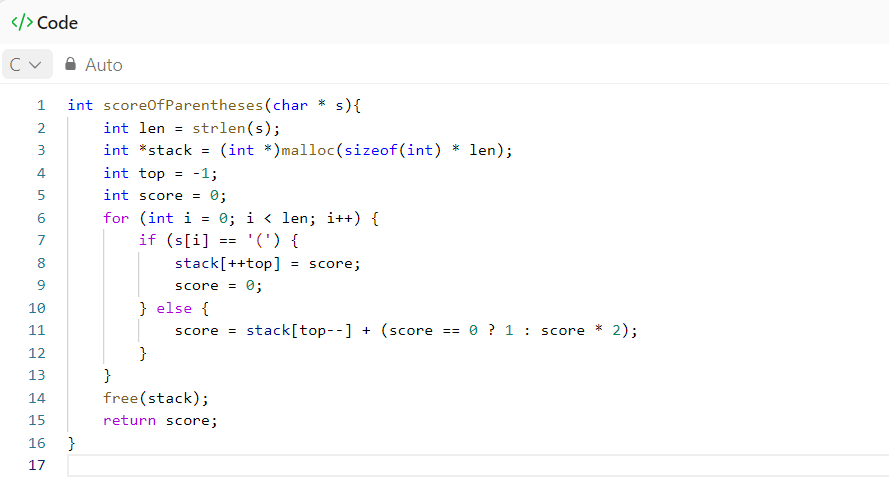
}

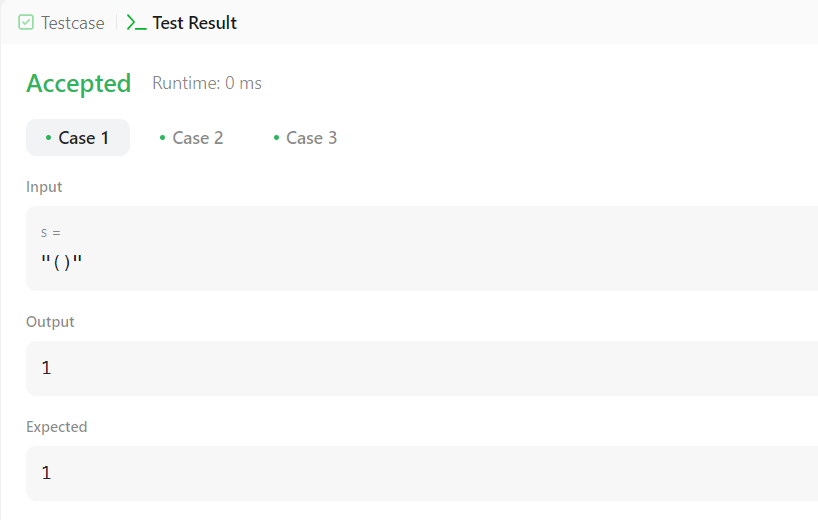
**Output:**



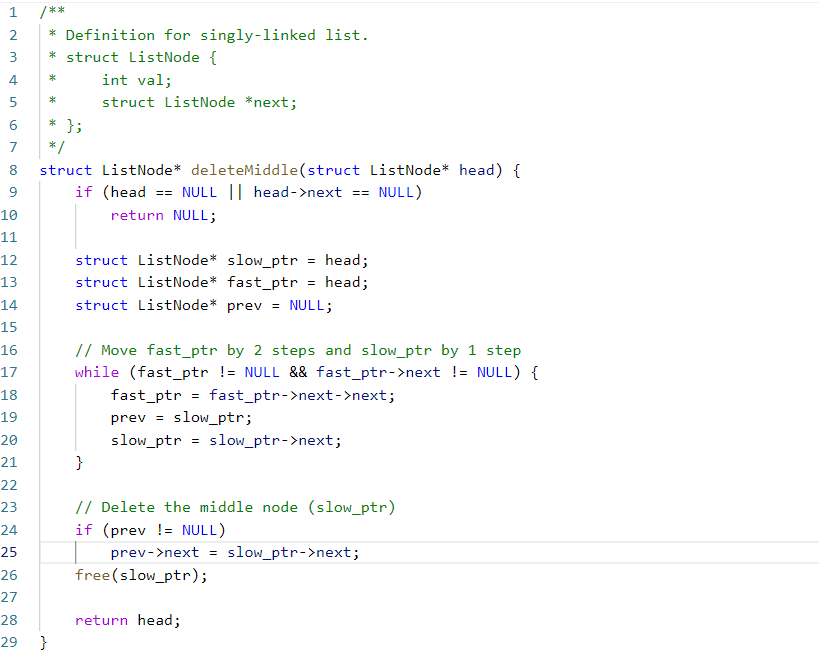
**Leet codes:**

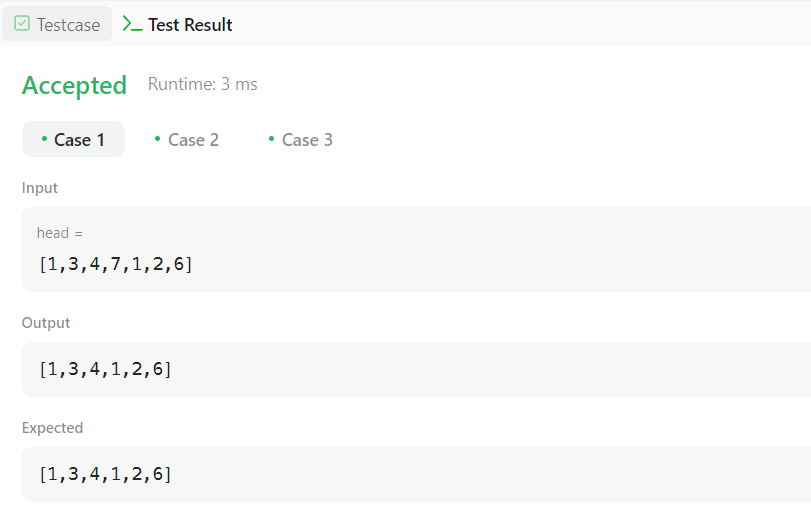
**1.Score of Parenthesis**





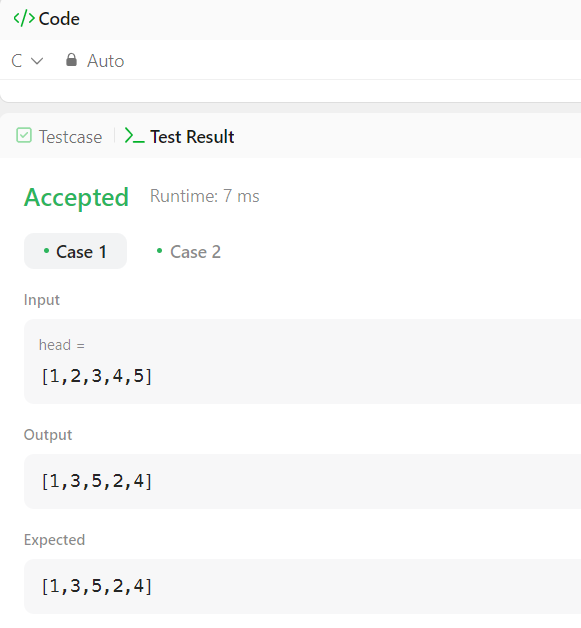
**2. Delete middle node**





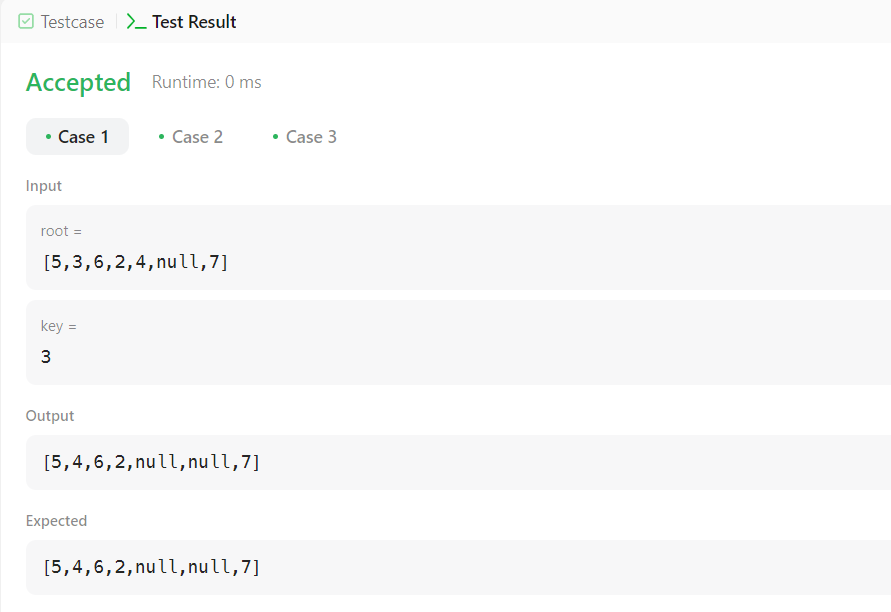
**3. Odd Even of Singly Linked List**





**4. Min value node in a BST**





**5. Find bottom left value in a BST**

