VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Tulasikrishna Tammina (1BM22CS310)

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 Tulasikrishna Tammina **(1BM22CS310)**, 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. Lakshmi Neelima
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

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**	
8	**Score of Parentheses - Leetcode**	
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.**	
10	**Delete the middle node of a linked list - Leetcode**	
11	**Odd Even Linked List - Leetcode**	
12	**a) Write a program to traverse a graph using BFS method.** **b) Write a program to check whether given graph is connected or not using DFS method.**	
13	**Delete Node in BST - Leetcode**	
14	**Find bottom left tree value - Leetcode**	
15	**Design and develop a Program in C that uses Hash function H: K -> 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 linear probing.**	

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.

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 <stdbool.h>

#define MAX_SIZE 5

int stack[MAX_SIZE];

int top = -1;

bool isFull() {

    return top == MAX_SIZE - 1;

}

bool isEmpty() {

    return top == -1;

}


void push(int item) {

    if (isFull()) {

        printf("Stack overflow: Cannot push element %d, stack is full.\n", item);

        return;

    }

    stack[++top] = item;

    printf("Element %d pushed onto the stack.\n", item);

}


int pop() {

    if (isEmpty()) {
```

```
        printf("Stack underflow: Cannot pop element, stack is empty.\n");
        return -1;
    }
    printf("Element %d popped from the stack.\n", stack[top]);
    return stack[top--];
}

void display() {
    if (isEmpty()) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Elements in the stack: ");
    for (int i = 0; i <= top; i++) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

int main() {
    int choice, item;

    do {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
            printf("Enter the element to push: ");
            scanf("%d", &item);
            push(item);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
  } while (choice != 4);
    return 0;
}
```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 10
Element 10 pushed onto the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 20
Element 20 pushed onto the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Elements in the stack: 10 20

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Element 20 popped from the stack.
```

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Elements in the stack: 10

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

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<ctype.h>
#define stack_size 20
void push(int *top,char stack[],char ele)
{
   stack[++(*top)]=ele;
}
char pop(int *top,char stack[])
{
   return stack[(*top)--];
}
int prece(char a)
{
   if(a=='^')
   {
      return 3;
   }
   else if(a=='*' || a=='/')
   {
      return 2;
   }
   else if(a=='+'|| a=='-')
   {
      return 1;
```

```
    }
    else
    {
        return 0;
    }
}
void intopo(char infix[],char postfix[])
{
    char ele;
    char stack[stack_size];
    int i=0,j=0,top=-1;
    while(infix[i]!='\0')
    {
        if(isalnum(infix[i]))
        {
            postfix[j]=infix[i];
            j++;
        }
        else if(infix[i]=='(')
        {
            push(&top,stack,infix[i]);
        }
        else if(infix[i]==')')
        {
            while(stack[top]!='(')
            {
                postfix[j]=pop(&top,stack);
                j++;
            }
```

```
        ele=pop(&top,stack);
      } else
      {
        while(prece(stack[top])>=prece(infix[i]))
        {
          postfix[j]=pop(&top,stack);
          j++;
        }
          push(&top,stack,infix[i]);
      }
       i++;
    }
    while(top!=-1)
    {
      postfix[j]=pop(&top,stack);
      j++;
    }
    postfix[j]='\0';
}
void main()
{
    char infix[20],postfix[20];
    printf("enter the infix expression\n");
    scanf("%s",infix);
    intopo(infix,postfix);
    printf("the postfix expression is: %s\n",postfix);
}
```

Output:

```
enter the infix expression
a+b*c/t
the postfix expression is: abc*t/+

Process returned 35 (0x23)   execution time : 7.365 s
Press any key to continue.
```

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<process.h>

#define que_size 3

void enque(int que[],int *front,int *rear,int ele)

{

   if(*rear==que_size-1)

   {

      printf("\n que overflow");

   }

   else

   {

      que[++(*rear)]=ele;

   }

}

int deque(int que[],int *front,int *rear)

{

   int del_ele;

   if((*front)>*rear)

   {

      printf("\nstack underflow");

   }

   else

   {

      del_ele=que[(*front)++];
```

```
    }
    return del_ele;
}
void display(int que[],int *front,int *rear)
{
    if(*front>*rear)
    {
        printf("\nstack is empty");
    }
    else
    {
        for(int i=*front;i<*rear+1;i++)
        {
            printf("\n element is %d",que[i]);
        }
    }
}
void main()
{
    int que[que_size],front=0,rear=-1,ele,ch,del_ele;
    do
    {
        printf("\n enter 1 for enque\n enter 2 for deque\n enter 3 for display\n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            {
                printf("\nenter the element to add:");
```

```
            scanf("%d",&ele);
            enque(que,&front,&rear,ele);
            printf("\nelement added to %d position",rear);
            break;
        }
    case 2:
        {
            del_ele=deque(que,&front,&rear);
            printf("\ndeleted element is %d at %d position",del_ele,front-1);
            break;
        }
    case 3:
        {
            display(que,&front,&rear);
            break;
        }

    default:
        {
            exit(0);
        }
    }
  }while(1);
}
```

Output:

```
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:2

element added to 0 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:3

element added to 1 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:4

element added to 2 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:6

 que overflow
element added to 2 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

deleted element is 2 at 0 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
3

 element is 3
 element is 4
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
```

Write a program 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>
#define MAX_SIZE 5
int queue[MAX_SIZE];
int front = -1, rear = -1;

int isEmpty() {
    return (front == -1 && rear == -1);
}

int isFull() {
    return ((rear + 1) % MAX_SIZE == front);
}

void enqueue(int element) {
    if (isFull()) {
        printf("Queue Overflow\n");
        return;
    } else if (isEmpty()) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
    }
    queue[rear] = element;
    printf("%d enqueued to the queue\n", element);
}
```

```
void dequeue() {
    if (isEmpty()) {
        printf("Queue Underflow\n");
        return;
    } else if (front == rear) {
        printf("%d dequeued from the queue\n", queue[front]);
        front = rear = -1;
    } else {
        printf("%d dequeued from the queue\n", queue[front]);
        front = (front + 1) % MAX_SIZE;
    }
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }
    printf("Elements in the circular queue are:\n");
    int i = front;
    do {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX_SIZE;
    } while (i != (rear + 1) % MAX_SIZE);
    printf("\n");
}

int main() {
```

```
int choice, element;

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 element to enqueue: ");
            scanf("%d", &element);
            enqueue(element);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice\n");
    }
```

```
    } while (choice != 4);

    return 0;
}
```

Output:

```
Circular Queue Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 10
10 enqueued to the queue

Circular Queue Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
20 enqueued to the queue

Circular Queue Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Elements in the circular queue are:
10 20
```

```
Circular Queue Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from the queue

Circular Queue Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Elements in the circular queue are:
20

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

Write A Program to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion and Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct nodes
{
   int data;
   struct nodes *link;
};
struct nodes *deletebg(struct nodes *head)
{
   struct nodes *ptr;
   if (head == NULL)
   {
      printf("create a list");
   }
   else
   {
      ptr = head;
      head = head->link;
      free(ptr);
   }
   return head;
}
struct nodes *deleteeg(struct nodes *head)
```

```
{
    struct nodes *ptr, *prev;
    prev = malloc(sizeof(struct nodes));
    ptr = malloc(sizeof(struct nodes));
    if (head == NULL)
    {
        free(head);
    }
    else
    {
        ptr = head;
        while (ptr->link != NULL)
        {
            prev = ptr;
            ptr = ptr->link;
        }
        prev->link = NULL;
        free(ptr);
    }
    return head;
}
struct nodes *deletetgiven(struct nodes *head, int data)
{
    struct nodes *ptr, *ptr1, *ptr2, *ptr3;
    ptr = malloc(sizeof(struct nodes));
    ptr1 = malloc(sizeof(struct nodes));
    ptr = head;
    while (ptr->link != NULL)
    {
```

```
        ptr1 = ptr;
        ptr = ptr->link;
        if (ptr->data == data)
        {
            ptr2 = ptr->link;
            ptr3 = ptr;
            ptr = ptr1;
            ptr->link = ptr2;
        }
    }
     free(ptr3);
    return head;
}
int main()
{
    struct nodes *head;
    struct nodes *ptr;
    struct nodes *current;
    head = malloc(sizeof(struct nodes));
    head->data = 10;
    head->link = NULL;
    current = malloc(sizeof(struct nodes));
    current->data = 11;
    current->link = NULL;
    head->link = current;
    current = malloc(sizeof(struct nodes));
    current->data = 12;
    current->link = NULL;
    head->link->link = current;
```

```
    current = malloc(sizeof(struct nodes));
    current->data = 13;
    current->link = NULL;
    head->link->link->link = current;
    //head=deletebg(head);
    //head=deleteeg(head);
   // head=deletetgiven(head,11);
    ptr = head;
    while (ptr != NULL)
    {
        printf("%d->", ptr->data);
        ptr = ptr->link;
    }
}
```

Output:

```
C:\Users\syst\Desktop\c\sllins

1. Create linked list
2. Display
3. Insert at the beginning
4. Insert at the end
5. Insert before a given value
6. Insert after a given value
7. Delete from the beginning
8. Delete from the end
9. Delete a specific node
10. Exit
Enter choice: 1

Enter 1. Creating list 2. Exit
1
Enter the value to be inserted: 2

Enter 1. Creating list 2. Exit
2
Enter choice: 3

Enter the value to be inserted: 4
Enter choice: 4

Enter the value to be inserted: 5
Enter choice: 5

Enter the value to be inserted: 7
Enter the value before which the data should be inserted: 4
Enter choice: 6

Enter the value to be inserted: 9
Enter the value after which the data should be inserted: 5
Enter choice: 2
```

```
Linked List: 4  2       5       9
Enter choice: 7

Node deleted from the beginning
Enter choice: 8

Node deleted from the end
Enter choice: 9

Enter value to be deleted: 5

Node with value 5 deleted
Enter choice: 2

Linked List: 2
Enter choice: 10

Exiting program

Process returned 0 (0x0)   execution time : 90.964 s
Press any key to continue.
```

Write A Program 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;
};

struct Node* insertEnd(struct Node* head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    return head;
}
```

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

void sortList(struct Node* head) {
    int swapped, temp;
    struct Node* current;
    struct Node* last = NULL;


    if (head == NULL || head->next == NULL) {
        return;
    }

    do {
        swapped = 0;
        current = head;

        while (current->next != last) {
            if (current->data > current->next->data) {
                temp = current->data;
                current->data = current->next->data;
                current->next->data = temp;
                swapped = 1;
```

```
            }
            current = current->next;
        }
        last = current;
    } while (swapped);
}

struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    return prev;
}

struct Node* concatenateLists(struct Node* list1, struct Node* list2) {
    if (list1 == NULL) {
        return list2;
    }

    struct Node* temp = list1;
    while (temp->next != NULL) {
```

```
        temp = temp->next;
    }

    temp->next = list2;
    return list1;
}

int main() {
    struct Node* list = NULL;
    int choice, value;

    do {
        printf("\n1. Insert\n2. Sort\n3. Reverse\n4. Concatenate\n5. Display\n0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                list = insertEnd(list, value);
                break;
            case 2:
                sortList(list);
                printf("List sorted successfully.\n");
                break;
            case 3:
                list = reverseList(list);
                printf("List reversed successfully.\n");
```

```
            break;
        case 4:
            {
                struct Node* list2 = NULL;
                printf("Enter values for the second list (enter -1 to stop):\n");
                while (1) {
                    scanf("%d", &value);
                    if (value == -1) {
                        break;
                    }
                    list2 = insertEnd(list2, value);
                }
                list = concatenateLists(list, list2);
                printf("Lists concatenated successfully.\n");
            }
            break;
        case 5:
            displayList(list);
            break;
        case 0:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
   } while (choice != 0);

   return 0;
}
```

```
1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 1
Enter the value to insert: 2

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 1
Enter the value to insert: 3

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 2
List sorted successfully.

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 5
2 -> 3 -> NULL

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 3
List reversed successfully.
```

```
1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 5
3 -> 2 -> NULL

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 4
Enter values for the second list (enter -1 to stop):
6
5
-1
Lists concatenated successfully.

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice: 5
3 -> 2 -> 6 -> 5 -> NULL

1. Insert
2. Sort
3. Reverse
4. Concatenate
5. Display
0. Exit
Enter your choice:
```

Write A Program to Implement Single Link List to simulate Stack and Queue Operations.

Linear Queue

```
#include <stdio.h>

#include <stdlib.h>


typedef struct node {

    int data;

    struct node *next;

} node;


node *stack_head = NULL;

void push(int value) {

    node *new_node = (node *)malloc(sizeof(node));

    if (new_node == NULL) {

        printf("Memory allocation failed.\n");

        return;

    }


    new_node->data = value;

    new_node->next = stack_head;

    stack_head = new_node;  // Push to the top of the stack

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

}


int pop() {

    if (stack_head == NULL) {

        printf("Stack underflow.\n");
```

```
        return -1;
    }

    node *temp = stack_head;
    int value = temp->data;
    stack_head = stack_head->next;
    free(temp);
    printf("%d popped from stack\n", value);
    return value;
}

void displayStack() {
    if (stack_head == NULL) {
        printf("Stack is empty.\n");
        return;
    }

    printf("Stack elements:\n");
    node *temp = stack_head;
    while (temp != NULL) {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}

int main() {
int choice, value;

    while (1) {
```

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

    switch (choice) {
      case 1:
        printf("Enter value to push: ");
        scanf("%d", &value);
        push(value);
        break;
      case 2:
        value = pop();
        if (value != -1) {
          printf("%d popped from stack\n", value);
        }
        break;
      case 3:
        displayStack();
      case 4:
        printf("Exiting...\n");
        exit(0);
      default:
        printf("Invalid choice.\n");
    }
  }
```

```
    return 0;
}
```

```
Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 3
3 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 6
6 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 2
6 popped from stack
6 popped from stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 3
Stack elements:
3
Exiting...

Process returned 0 (0x0)   execution time : 18.135 s
Press any key to continue.
```

Circular Queue

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int data;
    struct node *next;
} node;

node *queue_head = NULL;  // Head pointer for queue
node *queue_tail = NULL;  // Tail pointer for efficient enqueue

void enqueue(int value) {
    node *new_node = (node *)malloc(sizeof(node));
    if (new_node == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    new_node->data = value;
    new_node->next = NULL;

    if (queue_head == NULL) {
        queue_head = queue_tail = new_node;  // First node in the queue
    } else {
        queue_tail->next = new_node;
        queue_tail = new_node;  // Update tail pointer
    }
    printf("%d enqueued to queue\n", value);
}
```

```
int dequeue() {
    if (queue_head == NULL) {
       printf("Queue underflow.\n");
        return -1;
    }

    node *temp = queue_head;
    int value = temp->data;
    queue_head = queue_head->next;
    free(temp);

    if (queue_head == NULL) {
        queue_tail = NULL;  // Reset tail pointer if queue becomes empty
    }

    printf("%d dequeued from queue\n", value);
    return value;
}

void displayQueue() {
    if (queue_head == NULL) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements:\n");
    node *temp = queue_head;
    while (temp != NULL) {
```

```
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value;

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

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                value = dequeue();
                if (value != -1) {
                    printf("%d dequeued from queue\n", value);
                }
```

```
            break;
        case 3:
            displayQueue();
            break;
        case 4:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 34
34 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 56
56 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
34 dequeued from queue
34 dequeued from queue

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

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

Process returned 0 (0x0)   execution time : 17.386 s
Press any key to continue.
```

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

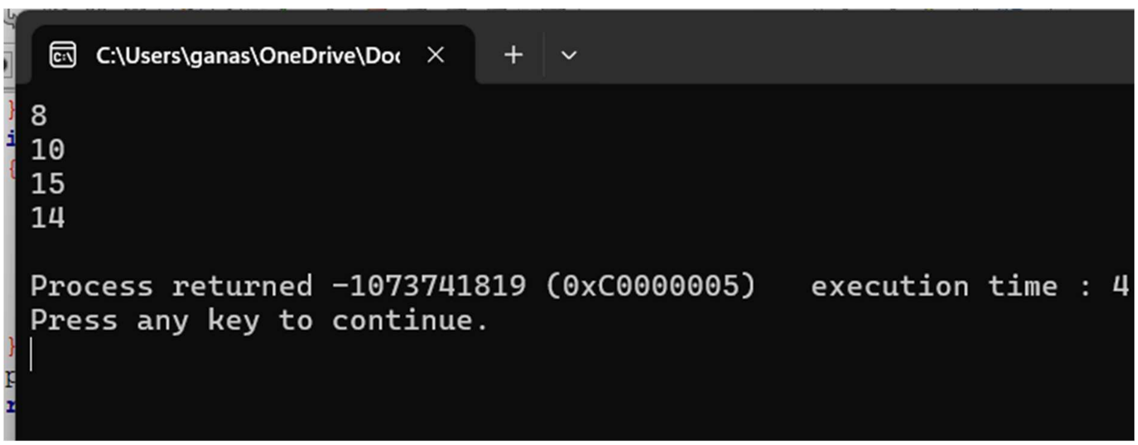
d) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
   int value;
   struct node *prev;
   struct node *next;
} Node;
Node *insertleft(Node *head, int data, int key)
{
   Node *new,*ptr;
   new = malloc(sizeof(Node));
   new->value = data;
   new->prev = NULL;
   new->next = NULL;
   ptr = head;
   if(head==NULL)
   {
      return new;
   }
   while(ptr!=NULL)
   {
      if(ptr->value==key)
```

```
        {
            break;
        }
        ptr=ptr->next;
    }
    if(ptr->value==key)
    {
        new->prev = ptr->prev;
        (ptr->prev)->next = new;
        new->next = ptr;
        ptr->prev = new;
        return head;
    }
    printf("no values");
    return head;
}
Node *deleteval(Node *head,int key)
{
    Node *ptr;
    if(head==NULL)
    {
        printf("list empty");
        return NULL;
    }
    ptr=head;
    while(ptr!=NULL&&ptr->value!=key)
    {
        ptr=ptr->next;
    }
```

```
    if(ptr->value==key)
    {
        (ptr->next)->prev=ptr->prev;
        (ptr->prev)->next=ptr->next;
        free(ptr);
        return head;
    }
    printf("no value");
    return head;
}
int main()
{
    Node *head = malloc(sizeof(Node));
    head->value = 8;
    head->prev = NULL;
    head->next = NULL;
    Node *current = malloc(sizeof(Node));
    current->value = 10;
    current->prev = head;
    current->next = NULL;
    head->next = current;
    Node *current2 = malloc(sizeof(Node));
    current2->value = 14;
    current2->prev = current;
    current2->next = NULL;
    current->next = current2;
    insertleft(head, 15, 14);
    Node *ptr1 = head;
    while (ptr1 != NULL)
```

```
    {
       printf("%d\n", ptr1->value);
       ptr1 = ptr1->next;
    }
    deleteval(head,8);
    Node *ptr = head;
    while (ptr != NULL)
    {
       printf("%d", ptr->value);
       ptr = ptr->next;
    }
}
```



Score of Parentheses (LeetCode)

```
int scoreOfParentheses(char *s) {
    int score = 0;
    int stack[50] = {0}; // Stack to store scores
    int top = -1; // Top of the stack

    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] == '(') {
            stack[++top] = score;
            score = 0;
        } else {
            score = stack[top--] + (score ? score * 2 : 1);
        }
    }

    return score;
}
```

Code

C Auto

```c
int scoreOfParentheses(char *s) {
    int score = 0;
    int stack[50] = {0}; // Stack to store scores
    int top = -1; // Top of the stack

    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] == '(') {
            stack[++top] = score;
            score = 0;
        } else {
            score = stack[top--] + (score ? score * 2 : 1);
        }
    }

    return score;
}
```

Saved to local   Ln 16, Col 2

Testcase | >_ Test Result

Accepted Runtime: 1 ms

• Case 1 • Case 2 • Case 3

Input

s =

"()"

Output

1

Expected

1

Contribute a testcase

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.

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

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

```
    }
    else {
        root->right = insert(root->right, data);
    }
    return root;
}

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

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

void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
```

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

void display(struct Node* root) {
    printf("In-order traversal: ");
    inorder(root);
    printf("\nPre-order traversal: ");
    preorder(root);
    printf("\nPost-order traversal: ");
    postorder(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL; // Initialize an empty binary search tree
    int n, data;
    printf("Enter the number of elements to insert: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }
    display(root);
     return 0;
}
```

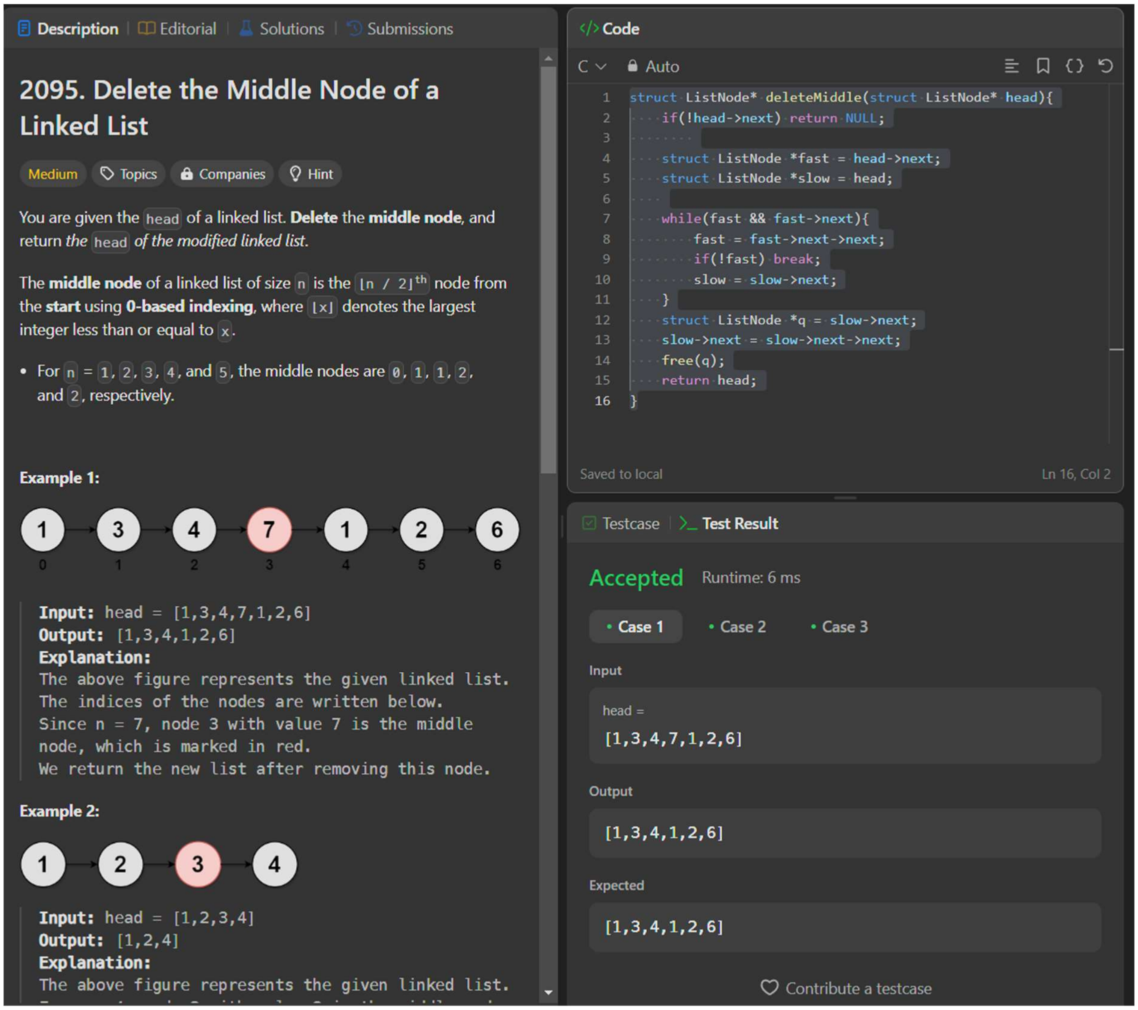
Output:

```
Enter the number of elements to insert: 7
Enter 7 elements:
4
2
6
1
3
5
7
In-order traversal: 1 2 3 4 5 6 7
Pre-order traversal: 4 2 1 3 6 5 7
Post-order traversal: 1 3 2 5 7 6 4
```

Delete the middle node of a linked list (Leetcode)

```
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;
}
```

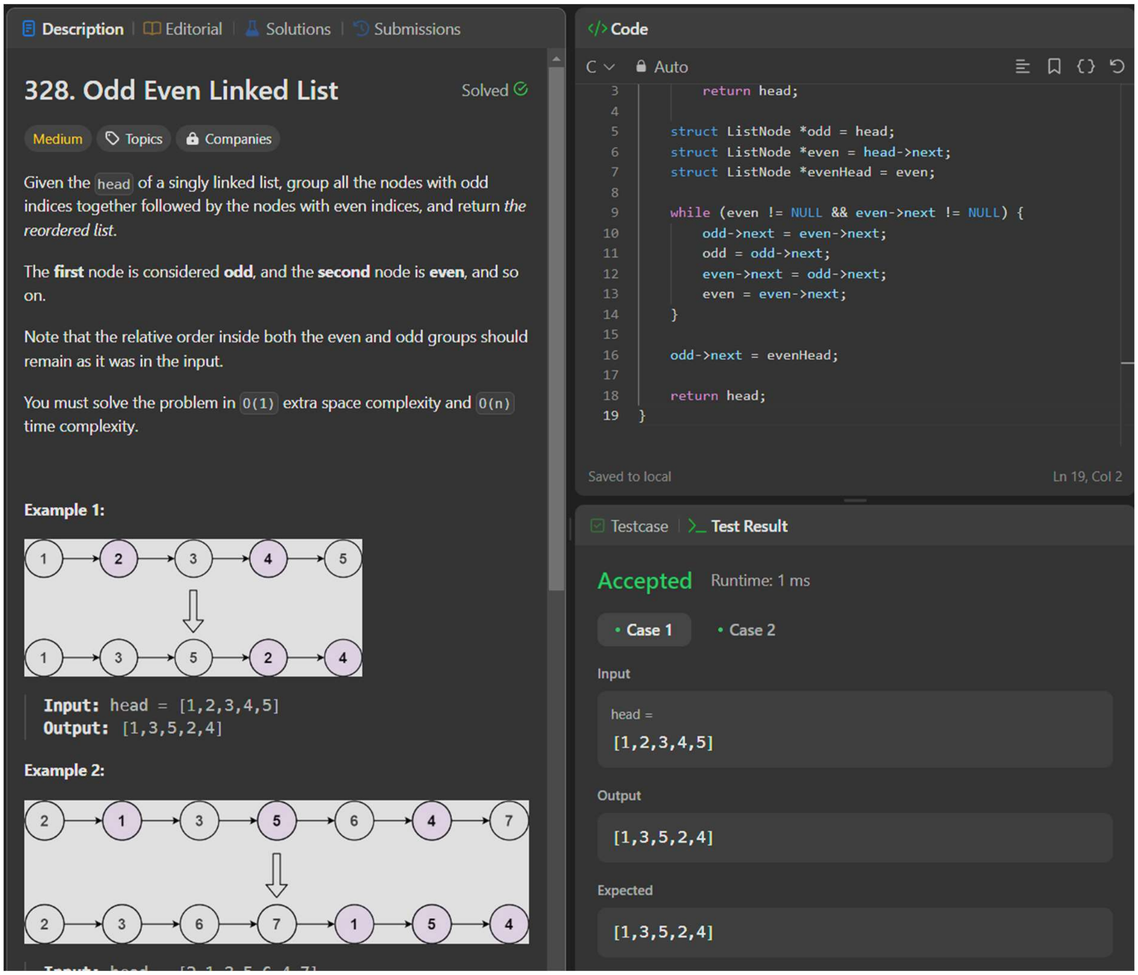
Output:


Description
Editorial
Solutions
Submissions
2095. Delete the Middle Node of a Linked List
Medium
Topics
Companies
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.
Code
C
Auto
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;
}
Saved to local
Ln 16, Col 2
Testcase
Test Result
Accepted
Runtime: 6 ms
Case 1
Case 2
Case 3
Input
head =
[1,3,4,7,1,2,6]
Output
[1,3,4,1,2,6]
Expected
[1,3,4,1,2,6]
Contribute a testcase

Odd Even Linked List (Leetcode)

```
struct ListNode* oddEvenList(struct ListNode* head) {
    if (head == NULL || head->next == NULL || head->next->next == NULL)
        return head;

    struct ListNode *odd = head;
    struct ListNode *even = head->next;
    struct ListNode *evenHead = even;

    while (even != NULL && even->next != NULL) {
        odd->next = even->next;
        odd = odd->next;
        even->next = odd->next;
        even = even->next;
    }

    odd->next = evenHead;

    return head;
}
```

Output:



Description | Editorial | Solutions | Submissions
328. Odd Even Linked List
Solved
Medium
Topics
Companies
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:
Code
C
Auto
return head;
struct ListNode *odd = head;
struct ListNode *even = head->next;
struct ListNode *evenHead = even;
while (even != NULL && even->next != NULL) {
odd->next = even->next;
odd = odd->next;
even->next = odd->next;
even = even->next;
}
odd->next = evenHead;
return head;
}
Saved to local
Ln 19, Col 2
Testcase
Test Result
Accepted
Runtime: 1 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
Output
[1,3,5,2,4]
Expected
[1,3,5,2,4]

Write a program to traverse a graph using BFS method.

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

#define MAX_NODES 100

struct Queue {
    int items[MAX_NODES];
    int front;
    int rear;
};

struct Graph {
    int vertices;
    int** adjMatrix;
};

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

void enqueue(struct Queue* queue, int value) {
    if (queue->rear == MAX_NODES - 1) {
        printf("Queue is full\n");
```

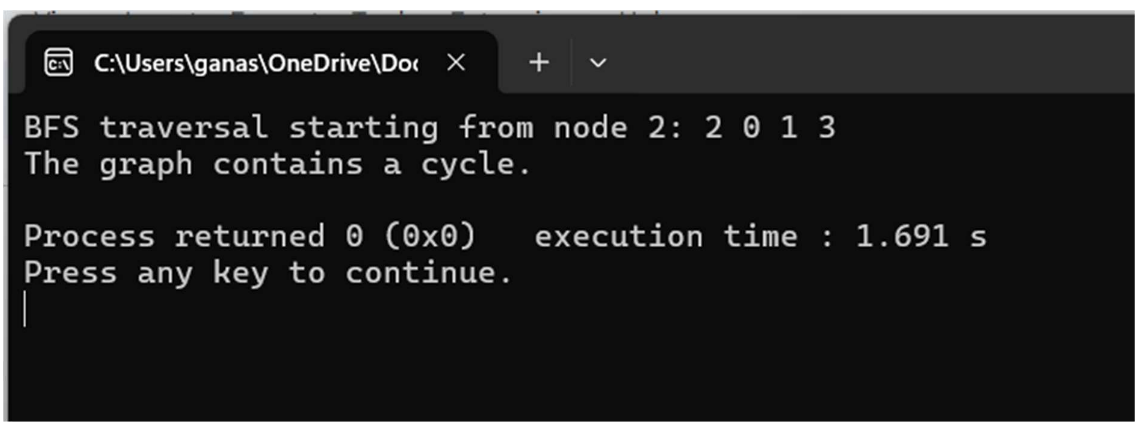
```
    } else {
        if (queue->front == -1) {
            queue->front = 0;
        }
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

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

bool isEmpty(struct Queue* queue) {
    return queue->front == -1;
}

struct Graph* createGraph(int vertices) {
```

```
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;

    graph->adjMatrix = (int**)malloc(vertices * sizeof(int*));
    for (int i = 0; i < vertices; i++) {
        graph->adjMatrix[i] = (int*)malloc(vertices * sizeof(int));
        for (int j = 0; j < vertices; j++) {
            graph->adjMatrix[i][j] = 0;
        }
    }

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1;
}

void BFS(struct Graph* graph, int startNode) {
    struct Queue* queue = createQueue();
    bool visited[MAX_NODES] = {false};

    printf("BFS traversal starting from node %d: ", startNode);

    visited[startNode] = true;
    printf("%d ", startNode);
    enqueue(queue, startNode);
```

```
    while (!isEmpty(queue)) {
        int currentNode = dequeue(queue);

        for (int i = 0; i < graph->vertices; i++) {
            if (graph->adjMatrix[currentNode][i] == 1 && !visited[i]) {
                printf("%d ", i);
                visited[i] = true;
                enqueue(queue, i);
            }
        }
    }

    printf("\n");
}

bool isCyclicUtil(struct Graph* graph, int v, bool visited[], int parent);

bool isCyclic(struct Graph* graph) {
    bool* visited = (bool*)malloc(graph->vertices * sizeof(bool));
    for (int i = 0; i < graph->vertices; i++) {
        visited[i] = false;
    }

    for (int i = 0; i < graph->vertices; i++) {
        if (!visited[i]) {
            if (isCyclicUtil(graph, i, visited, -1)) {
                free(visited);
                return true;
            }
```

```
        }
    }

    free(visited);
    return false;
}

bool isCyclicUtil(struct Graph* graph, int v, bool visited[], int parent) {
    visited[v] = true;

    for (int i = 0; i < graph->vertices; i++) {
        if (graph->adjMatrix[v][i] == 1) {
            if (!visited[i]) {
                if (isCyclicUtil(graph, i, visited, v)) {
                    return true;
                }
            } else if (i != parent) {
                return true;
            }
        }
    }

    return false;
}

int main() {
    struct Graph* graph = createGraph(4);

    addEdge(graph, 0, 1);
```

```
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 0);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 3);

    int startNode = 2;
    BFS(graph, startNode);
    if (isCyclic(graph)) {
        printf("The graph contains a cycle.\n");
    } else {
        printf("The graph does not contain a cycle.\n");
    }

    return 0;
}
```



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

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

int a[20][20], s[20], n;

void dfs(int v) {
    int i;
    printf("%d ", v); // Print the vertex when visiting
    s[v] = 1;
    for (i = 0; i < n; i++) {
        if (a[v][i] && !s[i]) {
            dfs(i);
        }
    }
}

int main() {
    int i, j, count=0;
    printf("\nEnter number of vertices: ");
    scanf("%d", &n);
    for(i=0; i<n; i++) {
        s[i]=0;
        for(j=0; j<n; j++)
            a[i][j]=0;
    }
    printf("Enter the adjacency matrix:\n");
```

```
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%d", &a[i][j]);

    printf("DFS traversal starting from vertex 0: ");
    dfs(0); // Start DFS from vertex 0
    printf("\n");

    for(i=0; i<n; i++) {
        if(s[i])
            count++;
    }
     if(count == n)
        printf("Graph is connected\n");
    else
        printf("Graph is not connected\n");
         return 0;
}
```

Output:

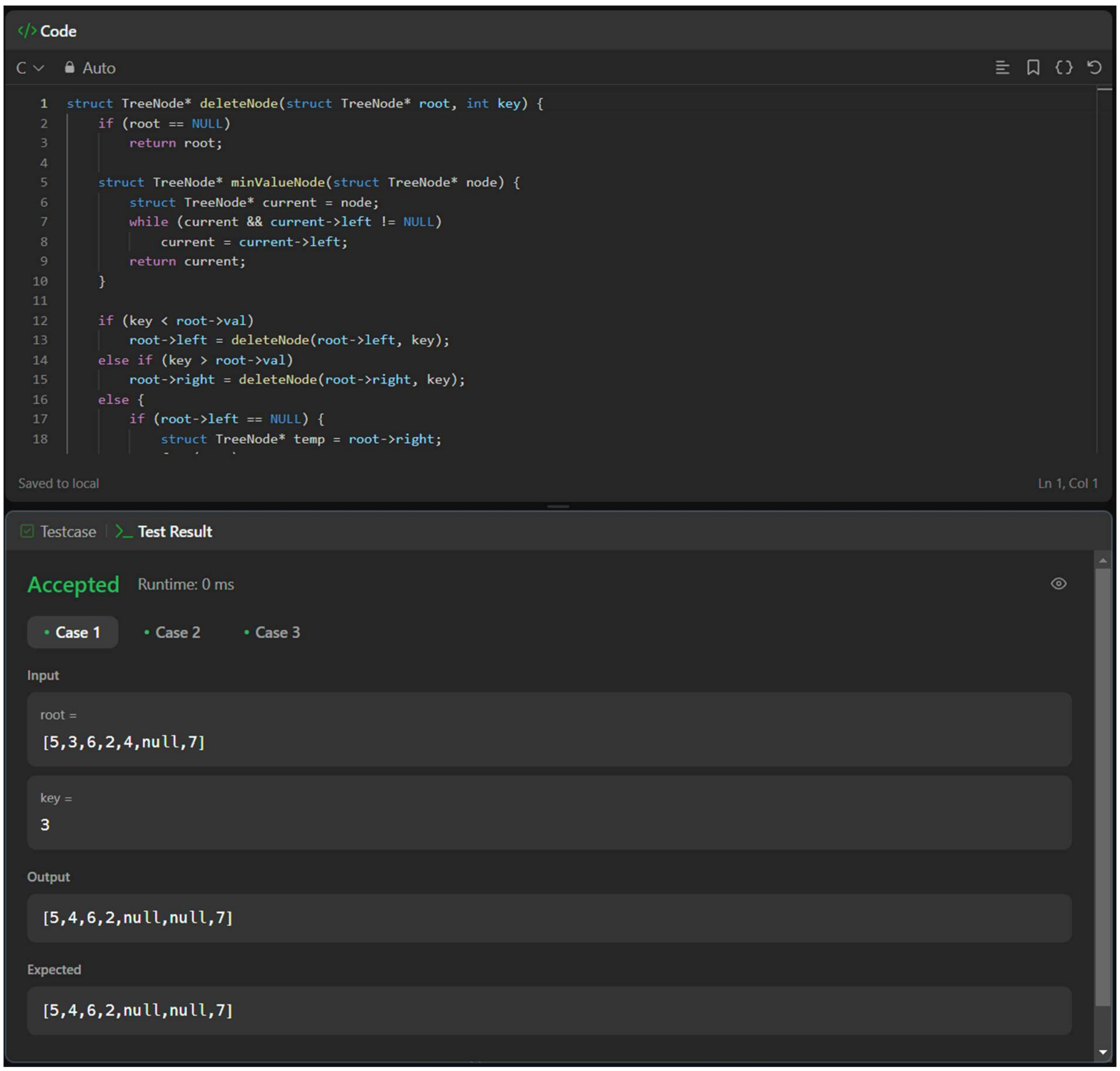
```
/tmp/fydhbR5jIg.o

Enter number of vertices: 3
Enter the adjacency matrix:
54
6
5
4
2
6
2
3
24
DFS traversal starting from vertex 0: 0 1 2
Graph is connected
```

Delete Node in BST (Leetcode)

```
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL)
        return root;

    struct TreeNode* minValueNode(struct TreeNode* node) {
        struct TreeNode* current = node;
        while (current && current->left != NULL)
            current = current->left;
        return current;
    }

    if (key < root->val)
        root->left = deleteNode(root->left, key);
    else if (key > root->val)
        root->right = deleteNode(root->right, key);
    else {
        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 = minValueNode(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
    return root;
}
```

Output:

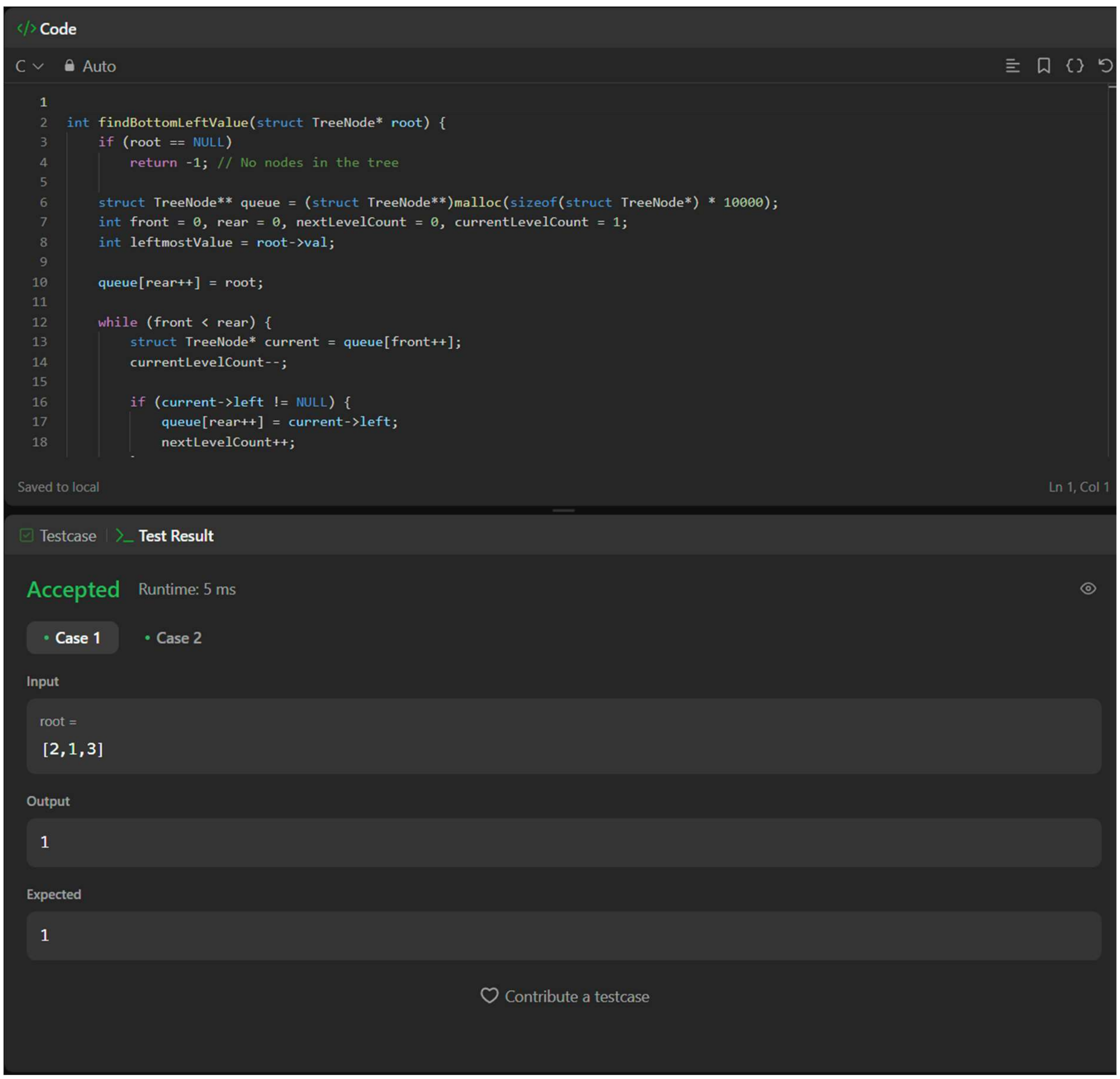


Find bottom left tree value (Leetcode)

```
int findBottomLeftValue(struct TreeNode* root) {
    if (root == NULL)
        return -1; // No nodes in the tree

    struct TreeNode** queue = (struct TreeNode**)malloc(sizeof(struct TreeNode*) * 10000);
    int front = 0, rear = 0, nextLevelCount = 0, currentLevelCount = 1;
    int leftmostValue = root->val;
    queue[rear++] = root;
      while (front < rear) {
        struct TreeNode* current = queue[front++];
        currentLevelCount--;

        if (current->left != NULL) {
            queue[rear++] = current->left;
            nextLevelCount++;
        }
        if (current->right != NULL) {
            queue[rear++] = current->right;
            nextLevelCount++;
        }

        if (currentLevelCount == 0) {
            if (nextLevelCount > 0)
                leftmostValue = queue[front]->val;
            currentLevelCount = nextLevelCount;
```

```
            nextLevelCount = 0;
        }
    }

    free(queue);
    return leftmostValue;
}
```

Output:



Design and develop a Program in C that uses Hash function H: K -> 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 linear probing.

```
#include <stdio.h>

#include <stdlib.h>


#define HT_SIZE 10


typedef struct {

   int key;

   } Employee;


typedef struct {

   int key;

   Employee employee;

} HashEntry;


typedef struct {

   HashEntry *table;

   int size;

} HashTable;


int hashFunction(int key, int size) {

   return key % size;

}


void initializeHashTable(HashTable *ht, int size) {

   ht->size = size;
```

```
    ht->table = (HashEntry *)malloc(size * sizeof(HashEntry));
    for (int i = 0; i < size; i++) {
        ht->table[i].key = -1;
    }
}

void insert(HashTable *ht, int key, Employee employee) {
    int index = hashFunction(key, ht->size);
    while (ht->table[index].key != -1) {
        index = (index + 1) % ht->size;
    }
    ht->table[index].key = key;
    ht->table[index].employee = employee;
}

int search(HashTable *ht, int key) {
    int index = hashFunction(key, ht->size);
    int originalIndex = index;

    while (ht->table[index].key != key && ht->table[index].key != -1) {
        index = (index + 1) % ht->size;
        if (index == originalIndex)
            return -1;
    }

    if (ht->table[index].key == key) {
        return index;
    } else {
        return -1;
```

```
    }
}

int main() {
    HashTable ht;
    initializeHashTable(&ht, HT_SIZE);

    int numEmployees;
    printf("Enter the number of employees: ");
    scanf("%d", &numEmployees);

    for (int i = 0; i < numEmployees; i++) {
        Employee emp;
        printf("Enter key for employee %d: ", i+1);
        scanf("%d", &emp.key);
        insert(&ht, emp.key, emp);
    }
    int searchKey;
    printf("Enter key to search: ");
    scanf("%d", &searchKey);
    int resultIndex = search(&ht, searchKey);
    if (resultIndex != -1) {
        printf("Employee with key %d found at index %d.\n", searchKey, resultIndex);
    } else {
        printf("Employee with key %d not found.\n", searchKey);
    }

    return 0;
}
```

Output:

```
Enter the number of employees: 3
Enter key for employee 1: 101
Enter key for employee 2: 201
Enter key for employee 3: 301
Enter key to search: 201
Employee with key 201 found at index 1.
```