VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SPARSHA KADABA (1BM22CS287)

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 SPARSHA KADABA **(1BM22CS287)**, 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

19	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.	64

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.

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.

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

#define max 10

int stack[max];
int top = -1;
int val;

void push(int);
void pop();
void display();

int main()
{
    int choice = 0;

    do
    {
        printf("Enter your choice\n");
        printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            if (top == max - 1)
            {
                printf("Overflow\n");
            }
            else
            {
                printf("Enter value to be pushed\n");
                scanf("%d", &val);
                push(val);
                printf("Push operation completed\n");
```

```
            }
            break;

        case 2:
            if (top == -1)
            {
                printf("Underflow\n");
            }
            else
            {
                pop();
                printf("Pop operation completed\n");
            }
            break;

        case 3:
            if (top == -1)
            {
                printf("Stack is empty\n");
            }
            else
            {
                display();
            }
            break;

        case 4:
            printf("Exit\n");
            exit(0);

        default:
            printf("Incorrect input\n");
        }
    } while (choice != 4);

    return 0;
}

void push(int val)
{
    if (top != max - 1)
    {
        top++;
```

```
        stack[top] = val;
    }
    else
    {
        printf("Overflow\n");
    }
}

void pop()
{
    if (top != -1)
    {
        val = stack[top];
        top--;
    }
    else
    {
        printf("Underflow\n");
    }
}

void display()
{
    if (top != -1)
    {
        for (int i = 0; i <= top; i++)
            printf("%d\t", stack[i]);
    }
    printf("\n");
}
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc one.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
1
Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
2
Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
1
Enter value to be pushed
3
Push operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
3
1       2       3
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
2
Pop operation completed
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
3
1       2
Enter your choice
1.Push
2.Pop
3.Display
4.Exit
4
Exit
PS C:\Users\kadab\OneDrive\Desktop\DS>
```

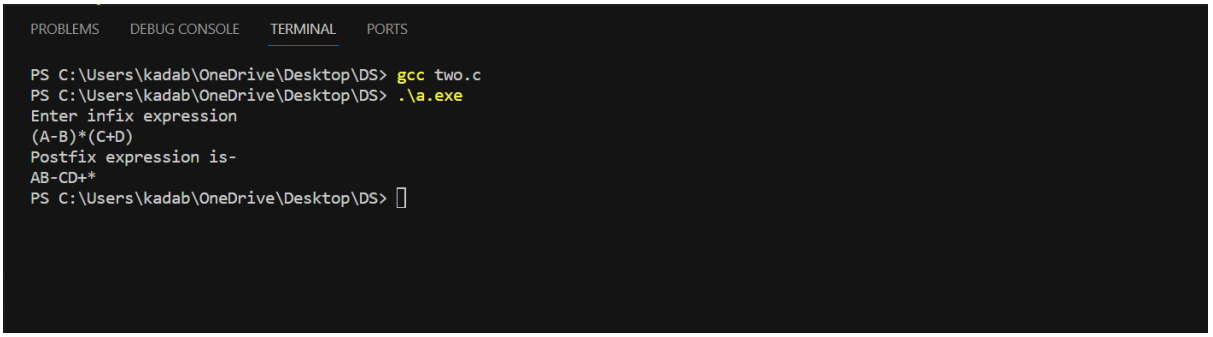
Program 2

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

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
#define max 100

char st[max];
int top=-1;
void push(char st[], char);
char pop(char st[]);
void infixToPostfix(char source[], char target[]);
int getPriority(char);

int main()
{
    char infix[100], postfix[100];
    printf("Enter infix expression\n");
    gets(infix);
    strcpy(postfix,"  ");
    infixToPostfix(infix, postfix);
    printf("Postfix expression is-\n");
    puts(postfix);
    return 0;
}

void infixToPostfix(char source[], char target[])
{
    int i=0, j=0;
    char temp;
    strcpy(target,"  ");
    while (source[i]!='\0')
    {
        if(source[i]=='(')
        {
            push(st,source[i]);
            i++;
        }
```

```
        else if(source[i]==')')
        {
            while((top!=-1)&&(st[top]!='('))
            {
                target[j]=pop(st);
                j++;
            }
            if(top==-1)
            {
                printf("Incorrect expression\n");
                exit(1);
            }
            temp=pop(st);
            i++;
        }
        else if(isdigit(source[i])||isalpha(source[i]))
        {
            target[j]=source[i];
            j++;
            i++;
        }
        else
if(source[i]=='+'||source[i]=='-'||source[i]=='*'||source[i]=='/'||sour
ce[i]=='%')
        {
            while( (top!=-1) && (st[top]!='(') &&
(getPriority(st[top])>getPriority(source[i])) )
            {
                target[j]=pop(st);
                j++;
            }
            push(st,source[i]);
            i++;
        }
        else
        {
            printf("Incorrect element in expression\n");
            exit(1);
        }
    }
    while((top!=-1)&&(st[top]!='('))
    {
        target[j]=pop(st);
```

```
        j++;
    }
    target[j]='\0';
}

int getPriority(char op)
{
    if(op=='/'|| op=='*'||op=='%')
        return 1;
    else if(op=='+'||op=='-')
        return 0;
}

void push(char st[],char val)
{
    if(top==max-1)
    {
        printf("Stack overflow\n");
    }
    else
    {
        top++;
        st[top]=val;
    }
}

char pop(char st[])
{
    char val = ' ';
    if(top == -1)
    {
        printf("Stack underflow\n");
    }
    else
    {
        val = st[top];
        top--;
    }
    return val;
}
```


PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc two.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Enter infix expression
(A-B)*(C+D)
Postfix expression is-
AB-CD+*
PS C:\Users\kadab\OneDrive\Desktop\DS>

Program 3

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define max 100

int q[max], front=1, rear=-1;
void insert();
void delete();
void display();

void main()
{
    while(1)
    {
        printf("Enter your choice\n");
        printf("1.Enqueue\n 2.Dequeue\n 3.Display\n 4.Exit\n");
        int choice=0;
        scanf("%d", & choice);
        switch(choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("Incorrect Input\n");
        }
    }
}
```

```
void insert()
{
    int val=0;
    printf("Enter value to be inserted\n");
    scanf("%d", & val);
    if(rear==max-1)
    printf("Queue Overflow\n");
    if (front==-1||rear==-1)
    {
        front=0;
        rear=0;
    }
    else{
        rear=rear+1;
    }
    q[rear]=val;
}

void delete()
{
    if (front>rear||front==-1)
    {
        printf("Queue Underflow\n");
    }
    else{
        printf("%d has been deleted\n",q[front]);
        front=front+1;
    }
}
void display()
{
    printf("The queue is-\n");
    for(int i=front; i<=rear; i++)
    printf("%d\t\t",q[i]);
    printf("\n");
}
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc three.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
1
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
2
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
1
Enter value to be inserted
3
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
3
The queue is-
1               2               3
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
2
1 has been deleted
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
3
The queue is-
2               3
Enter your choice
1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
4
PS C:\Users\kadab\OneDrive\Desktop\DS>
```

Program 4

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>
#include<conio.h>
#include<stdlib.h>
#define max 20

int q[max], rear=-1, front=-1;
int isFull();
int isEmpty();
void insert (int element);
int delete();
void display();

void main()
{
    int choice, element;
    while(1)
{
    printf("Enter choice\n");
    printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n\n");
    scanf("%d", &choice);
switch(choice)
{
    case 1:
    printf("Enter element to be inserted\n");
    scanf("%d", & element);
    insert(element);
    break;
    case 2:
    element=delete();
    break;
    case 3:
    display();
    break;
    case 4:
    exit(0); break;
    default:
    printf("Incorrect Input\n");
    break;
```

```
}
}
}

int isFull()
{
    if((front==rear+1)||(front==0 && rear==max-1))
        return 1;
    return 0;
}

int isEmpty()
{
    if(front==-1)
        return 1;
    return 0;
}

void insert(int element)
{
    if (isFull())
    {     printf("Overflow\n");
    }
    else
    {
        if(front==-1)
            front=0;
        rear=(rear+1)%max;
        q[rear]=element;
    }
}

int delete()
{
    int value;
    if (isEmpty())
    {
        printf("Underflow\n");
        return -1;
    }
    else{
        value=q[front];
        if(front==rear)
```

```
    {
    front==-1;
    rear==-1;
}
else
{
    front=(front+1)%max;
}
return(value);
}}

void display()
{
    int i;
    if(isEmpty())
        printf("Underflow\n");
    else{
        for(i=front;i!=rear;i=(i+1)%max)
            printf("%d\t",q[i]);
        printf("%d\t",q[i]);
        printf("\n");
        }
        }
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc four.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

1
Enter element to be inserted
1
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

1
Enter element to be inserted
2
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

1
Enter element to be inserted
3
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

3
1       2       3
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

2
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

3
2       3
Enter choice
1.Insert
2.Delete
3.Display
4.Exit

4
PS C:\Users\kadab\OneDrive\Desktop\DS>
```

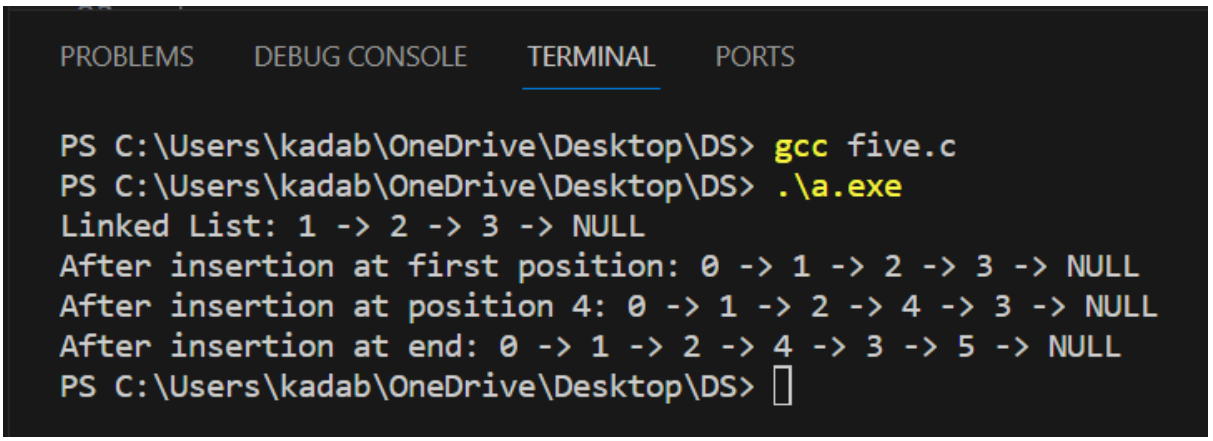
Program 5

Write a program 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. Display the contents of the linked list.

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

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

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

struct Node* insertAtFirst(struct Node* head, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data=newData;
    newNode->next = head;
    return newNode;
}

struct Node* insertAtPosition(struct Node* head, int newData, int
position)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data=newData;

    if (position == 1) {
        newNode->next = head;
        return newNode;
    }

    struct Node* temp = head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Invalid position\n");
        return head;
    }

    newNode->next = temp->next;
    temp->next = newNode;
    return head;
}

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

    if (head == NULL) {
        return newNode;
    }

    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 main() {
    struct Node* head = NULL;

    head = insertAtEnd(head, 1);
    head = insertAtEnd(head, 2);
```

```
    head = insertAtEnd(head, 3);

    printf("Linked List: ");
    displayList(head);

    head = insertAtFirst(head, 0);

    printf("After insertion at first position: ");
    displayList(head);

    head = insertAtPosition(head, 4, 4);

    printf("After insertion at position 4: ");
    displayList(head);

    head = insertAtEnd(head, 5);

    printf("After insertion at end: ");
    displayList(head);
}
```



Program 6

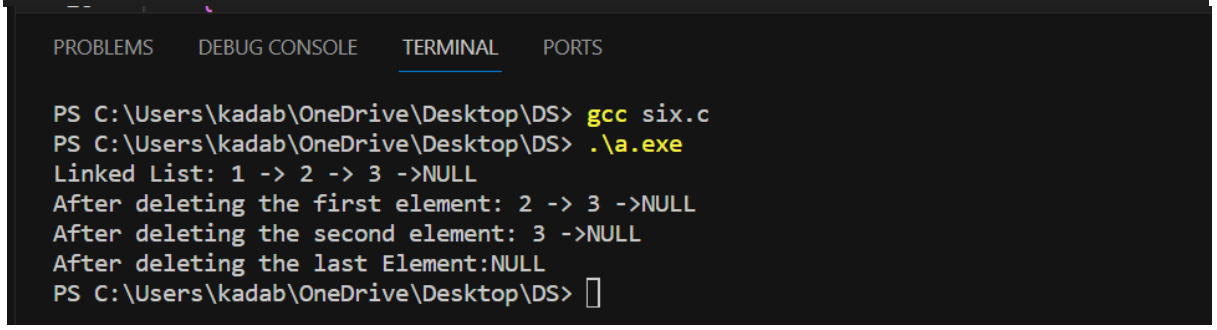
Write a program to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of a node at first position, at any position and at end of list. Display the contents of the linked list.

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

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

struct Node* insertAtEnd(struct Node* head, int newData)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = NULL;
    if (head == NULL)
    {
        return newNode;
    }
    struct Node* temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}

struct Node* deleteFirst(struct Node* head)
{
    if (head == NULL)
    {
        printf("List is Empty! Deletion not Possible");
        return NULL;
    }
    struct Node* newHead = head->next;
    free(head);
    return newHead;
}
```

```c
struct Node* deleteElement(struct Node* head, int target)
{
    if (head == NULL)
    {
        printf("List is Empty, hence cannot Delete \n");
        return NULL;
    }
    if (head->data == target)
    {
        struct Node* newHead = head->next;
        free(head);
        return newHead;
    }

    struct Node* temp = head;
    while (temp->next != NULL && temp->next->data != target)
    {
        temp = temp->next;
    }
    if (temp->next == NULL)
    {
        printf("Element %d not found in the list \n", target);
        return head;
    }
    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
    return head;
}

struct Node* deleteLast(struct Node* head)
{
    if (head == NULL)
    {
        printf("List is Empty, hence cannot Delete \n");
        return NULL;
    }

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

```
    struct Node* temp = head;
    while (temp->next->next != NULL)
    {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    return head;
}

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

int main()
{
    struct Node* head = NULL;
    head = insertAtEnd(head, 1);
    head = insertAtEnd(head, 2);
    head = insertAtEnd(head, 3);

    printf("Linked List:");
    displayList(head);

    head = deleteFirst(head);
    printf("After deleting the first element:");
    displayList(head);

    head = deleteElement(head, 2);
    printf("After deleting the second element:");
    displayList(head);

    head = deleteLast(head);
    printf("After deleting the last Element:");
    displayList(head);
```

```
    return 0;
}
```


PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc six.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Linked List: 1 -> 2 -> 3 ->NULL
After deleting the first element: 2 -> 3 ->NULL
After deleting the second element: 3 ->NULL
After deleting the last Element:NULL
PS C:\Users\kadab\OneDrive\Desktop\DS>

Program 7
Write a program to Implement Single Link List to simulate Stack Operations.

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define CAPACITY 1000

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

int size = 0;

void push(int element);
int pop();
void display();


int main()
{
    int choice, data;
    while(1)
    {
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                printf("Enter data to push into stack: ");
                scanf("%d", &data);
                push(data);
                break;

            case 2:
                data = pop();
```

```
                // If stack is not empty
                if (data != INT_MIN)
                    printf("Data => %d\n", data);
                break;

            case 3:
                display();
                break;

            case 4:
                exit(0);
                break;

            default:
                printf("Invalid choice\n");
        }

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

void push(int element)
{
    if (size >= CAPACITY)
    {
        printf("Stack Overflow\n");
        return;
    }
    struct stack * newNode = (struct stack *) malloc(sizeof(struct
stack));
    newNode->data = element;
    newNode->next = top;
    top = newNode;
    size++;
    printf("Data pushed to stack.\n");
}

int pop()
{
    int data = 0;
    struct stack * topNode;
    if (size <= 0 || !top)
```

```
    {
        printf("Stack is empty.\n");
        return INT_MIN;
    }
    topNode = top;
    data = top->data;
    top = top->next;
    free(topNode);
    size--;
    return data;
}

void display()
{
    if(size<=0||!top)
    {
        printf("Stack is empty\n");
        return;
    }
    struct stack *current=top;
    printf("Stack elements: ");
    while(current!=NULL)
    {
        printf("%d  ",current->data);
        current=current->next;
    }
    printf("\n\n");
}
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
1. Push
2. Pop
3. Display
4. Exit
1
Enter data to push into stack: 1
Data pushed to stack.


1. Push
2. Pop
3. Display
4. Exit
1
Enter data to push into stack: 2
Data pushed to stack.


1. Push
2. Pop
3. Display
4. Exit
1
Enter data to push into stack: 3
Data pushed to stack.


1. Push
2. Pop
3. Display
4. Exit
3
Stack elements: 3  2  1



1. Push
2. Pop
3. Display
4. Exit
2
Data => 3


1. Push
2. Pop
3. Display
4. Exit
3
Stack elements: 2  1



1. Push
2. Pop
3. Display
4. Exit
4
PS C:\Users\kadab\OneDrive\Desktop\DS>
```

Program 8

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

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

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

struct node * front = NULL;
struct node * rear = NULL;

void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr -> next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear -> next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}

int dequeue() {
    if (front == NULL) {
        printf("\nUnderflow\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        return temp_data;
    }
}

void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
```

```
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp -> data);
            temp = temp -> next;
        }
        printf("NULL\n\n");
    }
}

int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (choice != 4) {
        printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", & choice);
        switch (choice) {
            case 1:
                printf("\nEnter the value to insert: ");
                scanf("%d", & value);
                enqueue(value);
                break;
            case 2:
                printf("Popped element is :%d\n", dequeue());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                printf("\nWrong Choice\n");
        }
    }
    return 0;
}
```

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

Enter the value to insert: 1
Node is Inserted

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

Enter the value to insert: 2
Node is Inserted

1.Enqueue
2.Dequeue
3.Display
4.Exit
3
The queue is
1--->2--->NULL

1.Enqueue
2.Dequeue
3.Display
4.Exit
2
Popped element is :1
1.Enqueue
2.Dequeue
3.Display
4.Exit
3
The queue is
2--->NULL

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

Process returned 0 (0x0)   execution time : 15.932 s
Press any key to continue.
```

Program 9

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

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

struct node {
    int info;
    struct node* link;
};

void createList(struct node** start);
void displayList(struct node* start);
void sort(struct node* start);
void reverseLL(struct node** start);
void concat(struct node** start1, struct node* start2);

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

    while (1) {
        printf("1. Create List 1\n");
        printf("2. Create List 2\n");
        printf("3. Display List 1\n");
        printf("4. Display List 2\n");
        printf("5. Sort List 1\n");
        printf("6. Sort List 2\n");
        printf("7. Reverse List 1\n");
        printf("8. Reverse List 2\n");
        printf("9. Concatenate Lists\n");
        printf("0. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&start1);
                break;

            case 2:
                createList(&start2);
                break;
```

```
            case 3:
                displayList(start1);
                break;

            case 4:
                displayList(start2);
                break;

            case 5:
                sort(start1);
                printf("List 1 sorted.\n");
                break;

            case 6:
                sort(start2);
                printf("List 2 sorted.\n");
                break;

            case 7:
                reverseLL(&start1);
                printf("List 1 reversed.\n");
                break;

            case 8:
                reverseLL(&start2);
                printf("List 2 reversed.\n");
                break;

            case 9:
                concat(&start1, start2);
                printf("Lists concatenated.\n");
                break;

            case 0:
                printf("Exiting from the program.\n");
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
```

```
    return 0;
}

void createList(struct node** start) {
    if (*start == NULL) {
        int n, data;
        printf("\nEnter the number of nodes: ");
        scanf("%d", &n);

        struct node* newnode;
        struct node* temp;

        for (int i = 1; i <= n; i++) {
            newnode = malloc(sizeof(struct node));

            printf("\nEnter number to be inserted: ");
            scanf("%d", &data);

            newnode->info = data;
            newnode->link = NULL;

            if (*start == NULL) {
                *start = newnode;
                temp = newnode;
            } else {
                temp->link = newnode;
                temp = temp->link;
            }
        }
        printf("\nThe list is created\n");
    } else {
        printf("\nThe list is already created\n");
    }
}


void displayList(struct node* start) {
    if (start == NULL) {
        printf("List is empty.\n");
    } else {
        struct node* current = start;

        printf("List: ");
```

```
        while (current != NULL) {
            printf("%d -> ", current->info);
            current = current->link;
        }
        printf("NULL\n");
    }
}


void sort(struct node* start) {
    struct node *current, *index;
    int temp;

    if (start == NULL || start->link == NULL) {
        return;
    } else {
        current = start;

        while (current != NULL) {
            index = current->link;

            while (index != NULL) {
                if (current->info > index->info) {
                    temp = current->info;
                    current->info = index->info;
                    index->info = temp;
                }
                index = index->link;
            }

            current = current->link;
        }
    }
}


void reverseLL(struct node** start) {
    struct node *prev, *current, *next;
    current = (*start)->link;
    prev = NULL;

    while (current != NULL) {
        next = current->link;
```

```
        current->link = prev;
        prev = current;
        current = next;
    }

    (*start)->link = prev;
}

void concat(struct node** start1, struct node* start2) {
    if (start2 == NULL) {
        return;  // Nothing to concatenate
    }

    struct node* temp = *start1;

    if (*start1 == NULL) {
        *start1 = start2;
    } else {
        while (temp->link != NULL) {
            temp = temp->link;
        }

        temp->link = start2;
    }
}
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc nine.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
1

Enter the number of nodes: 2

Enter number to be inserted: 1

Enter number to be inserted: 2

The list is created
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
2

Enter the number of nodes: 2

Enter number to be inserted: 5

Enter number to be inserted: 3
```

```
The list is created
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
5
List 1 sorted.
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
6
List 2 sorted.
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
7
```

```
List 1 reversed.
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
8
List 2 reversed.
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
9
Lists concatenated.
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
3
List: 1 -> 2 -> 3 -> 5 -> NULL
```

```
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
4
List: 3 -> 5 -> NULL
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Sort List 2
7. Reverse List 1
8. Reverse List 2
9. Concatenate Lists
0. Exit
0
Exiting from the program.
PS C:\Users\kadab\OneDrive\Desktop\DS>
```

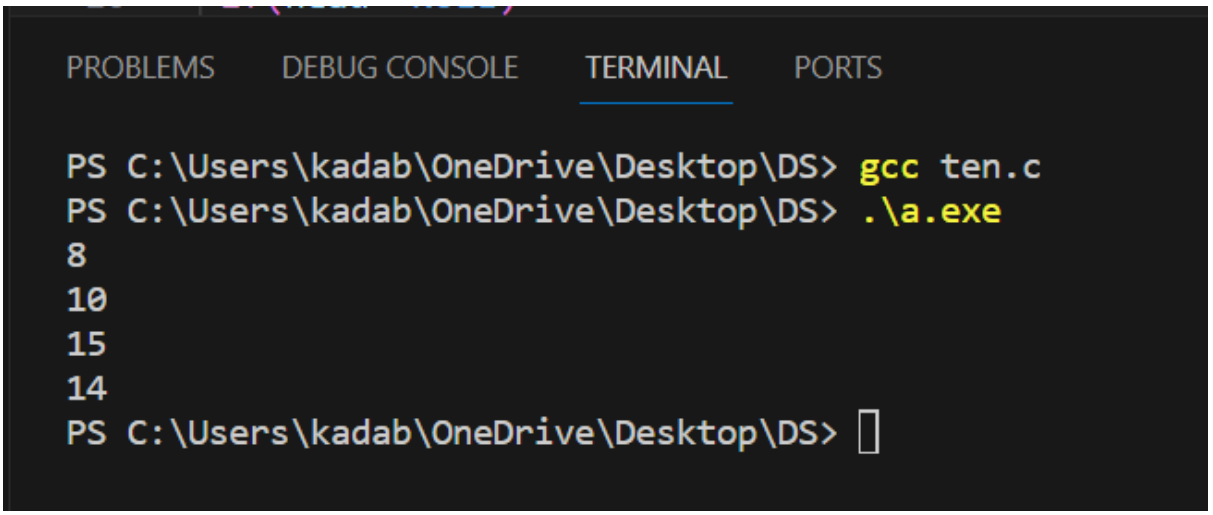
Program 10

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;
}
}
```

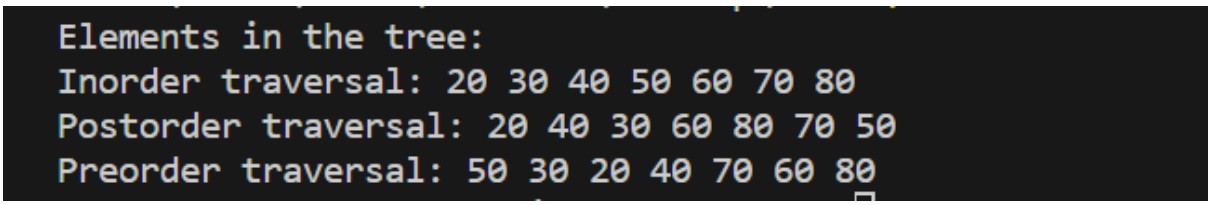


Program 11

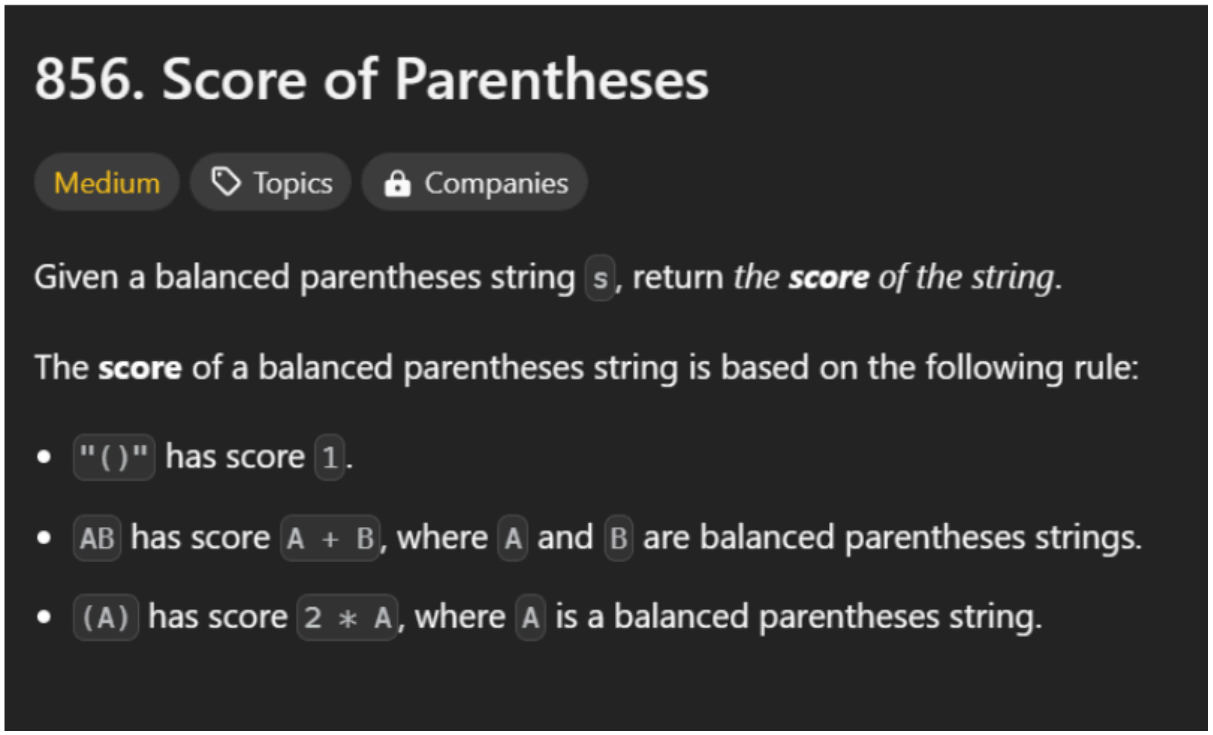
Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree

```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode
{
    int data;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* createNode(int data)
{
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    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);
    } else {
        if (data <= root->data) {
            root->left = insert(root->left, data);
        } else {
            root->right = insert(root->right, data);
        }
        return root;
    }
}
void inorder(struct TreeNode* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void postorder(struct TreeNode* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
```

```
        printf("%d ", root->data);
    }
}
void preorder(struct TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void display(struct TreeNode* root) {
    if (root != NULL) {
        printf("Inorder traversal: ");
        inorder(root);
        printf("\n");

        printf("Postorder traversal: ");
        postorder(root);
        printf("\n");

        printf("Preorder traversal: ");
        preorder(root);
        printf("\n");
    } else {
        printf("Tree is empty.\n");
    }
}

void main() {
    struct TreeNode* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
    printf("Elements in the tree:\n");
    display(root);
}
```


Elements in the tree:
Inorder traversal: 20 30 40 50 60 70 80
Postorder traversal: 20 40 30 60 80 70 50
Preorder traversal: 50 30 20 40 70 60 80

Program 12
Score of Parentheses - Leetcode



856. Score of Parentheses

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.

```c
int scoreOfParentheses(char* s)
{
    int len = strlen(s);
    int score = 0;
    int depth = 0;

    for (int i = 0; i < len; i++)
    {
        if (s[i] == '(')
        {
            depth++;
        }
        else
        {
            depth--;
            if (s[i - 1] == '(')
            {
               score += 1 << depth;
            }
        }
    }
```

```
    return score;
}
```

Program 13
Delete the middle node of a linked list - Leetcode

2095. Delete the Middle Node of a Linked List

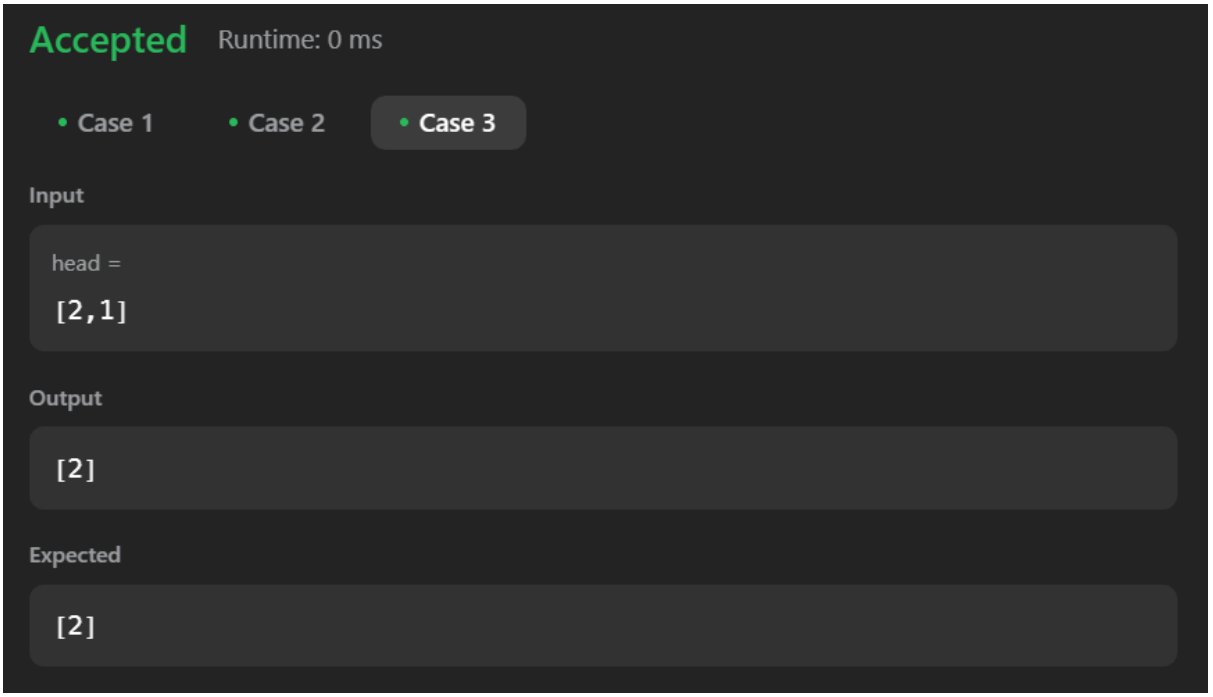
Medium | Topics | Companies | Hint

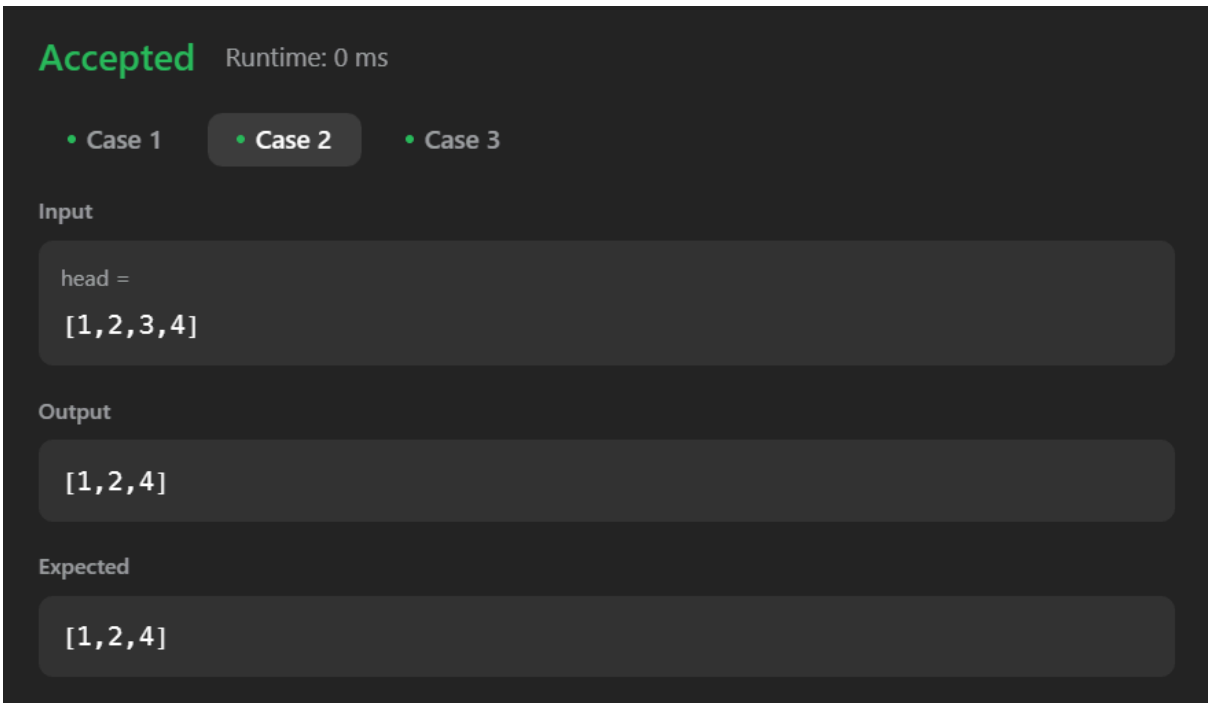
You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.*

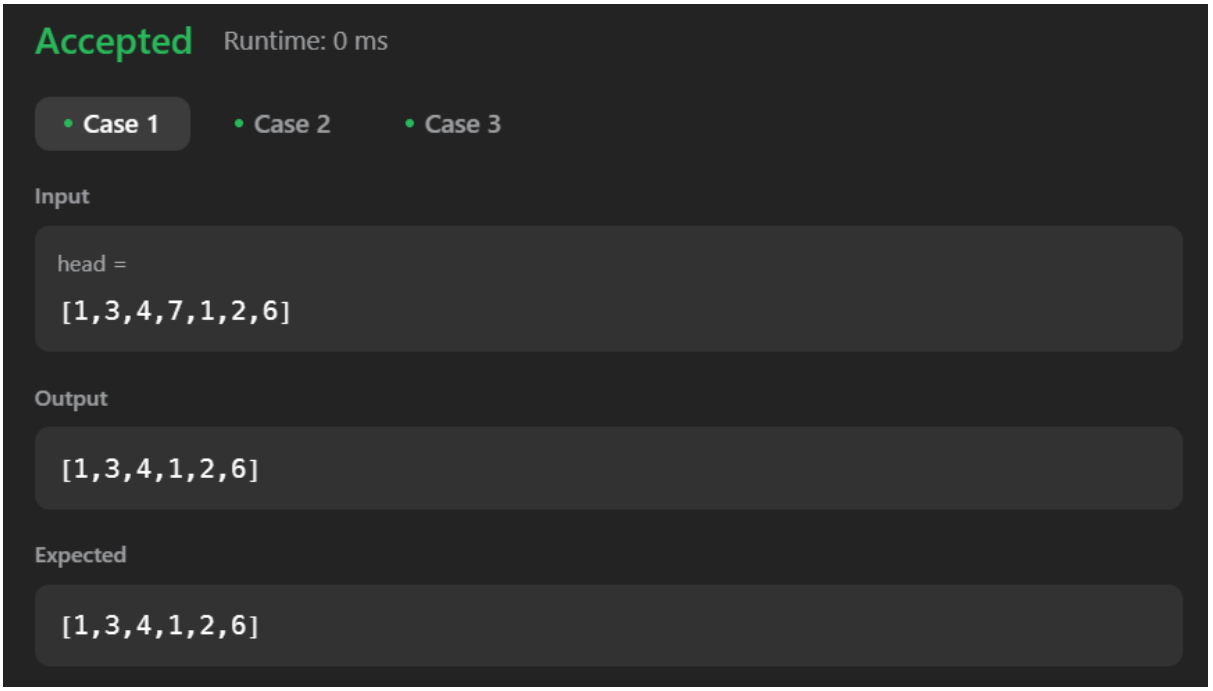
The **middle node** of a linked list of size `n` is the `⌊n / 2⌋th` node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

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

```
struct ListNode* deleteMiddle(struct ListNode* head)
{
    int count=0, middleNode, i=0;
    struct ListNode* temp=head;
    struct ListNode* node=head;
    struct ListNode* prev;
    struct ListNode* next=head;
    while(temp!=NULL)
    {
        count++;
        temp=temp->next;
    }
    middleNode=count/2;
    if(middleNode==0)
    {
        struct ListNode* newHead=head->next;
        free(head);
        return newHead;
    }
    while(i<middleNode)
    {
        prev=node;
        node=node->next;
        next=node->next;
        i++;
    }
    prev->next=next;
    free(node);
    return head;
}
```


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
head =
[2,1]
Output
[2]
Expected
[2]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
head =
[1,2,3,4]
Output
[1,2,4]
Expected
[1,2,4]


Accepted
Runtime: 0 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]

Program 14
Odd Even Linked List - Leetcode

328. Odd Even Linked List

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*.

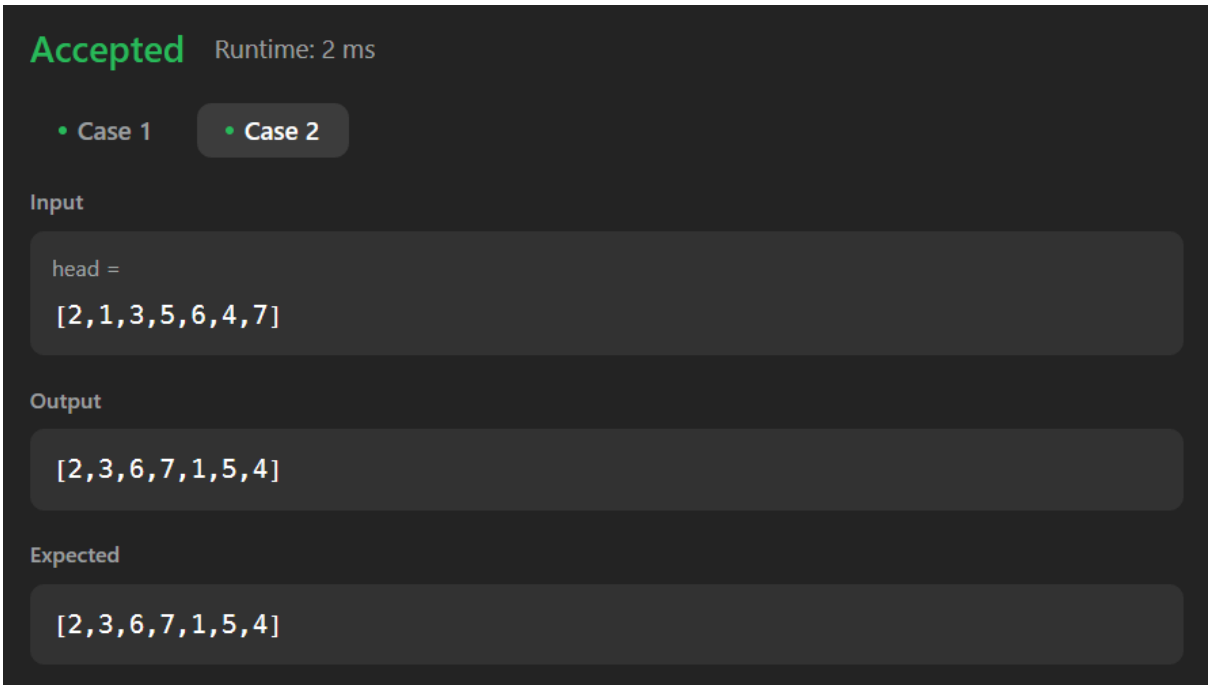
The **first** node is considered **odd**, and the **second** node is **even**, and so on.

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.

```
struct ListNode* oddEvenList(struct ListNode* head)
{
    if(head==NULL||head->next==NULL)
    {
        return head;
    }
    struct ListNode* oddTemp=head;
    struct ListNode* evenTemp=head->next;
    struct ListNode* evenHead=evenTemp;
    while(evenTemp!=NULL && evenTemp->next!=NULL)
    {
        oddTemp->next=evenTemp->next;
        oddTemp=oddTemp->next;
        evenTemp->next=oddTemp->next;
        evenTemp=evenTemp->next;
    }
    oddTemp->next=evenHead;
    return head;
}
```


Accepted
Runtime: 2 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
Output
[1,3,5,2,4]
Expected
[1,3,5,2,4]


Accepted
Runtime: 2 ms
Case 1
Case 2
Input
head =
[2,1,3,5,6,4,7]
Output
[2,3,6,7,1,5,4]
Expected
[2,3,6,7,1,5,4]

Program 15
Write a program to traverse a graph using BFS method.

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

#define MAX_VERTICES 100

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

struct Graph {
    struct Node* adjLists[MAX_VERTICES];
    int visited[MAX_VERTICES];
};

struct Node* createNode(int v) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph() {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    int i;
    for (i = 0; i < MAX_VERTICES; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
```

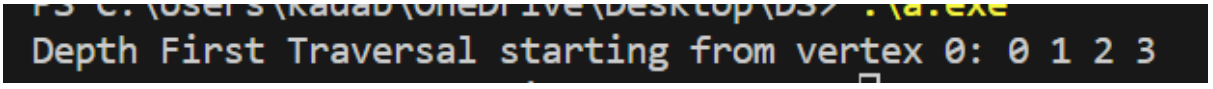
```
}

void BFS(struct Graph* graph, int startVertex) {
    struct Node* temp;
    int queue[MAX_VERTICES];
    int front = 0, rear = 0, v;

    graph->visited[startVertex] = 1;
    queue[rear++] = startVertex;

    while (front < rear) {
        v = queue[front++];
        printf("%d ", v);

        for (temp = graph->adjLists[v]; temp != NULL; temp =
temp->next) {
            if (!graph->visited[temp->vertex]) {
                graph->visited[temp->vertex] = 1;
                queue[rear++] = temp->vertex;
            }
        }
    }
}

void main() {
    struct Graph* graph = createGraph();

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

    printf("Breadth First Traversal starting from vertex 0: ");
    BFS(graph, 0);
}
```

```
Breadth First Traversal starting from vertex 0: 0 2 1 3
```

Program 16
Write a program to traverse a graph using DFS method.

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

#define MAX_VERTICES 100

struct Graph {
    int matrix[MAX_VERTICES][MAX_VERTICES];
    int visited[MAX_VERTICES];
    int numVertices;
};

struct Graph* createGraph(int numVertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;
    int i, j;
    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            graph->matrix[i][j] = 0;
        }
        graph->visited[i] = 0;
    }
    return graph;
}

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

void DFS(struct Graph* graph, int vertex) {
    printf("%d ", vertex);
    graph->visited[vertex] = 1;
    int i;
    for (i = 0; i < graph->numVertices; i++) {
        if (graph->matrix[vertex][i] && !graph->visited[i]) {
            DFS(graph, i);
        }
    }
}

void main() {
```

```c
    int numVertices = 5;
    struct Graph* graph = createGraph(numVertices);

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

    printf("Depth First Traversal starting from vertex 0: ");
    DFS(graph, 0);
}
```



Program 17
Delete Node in BST - Leetcode

450. Delete Node in a BST

Medium | Topics | Companies

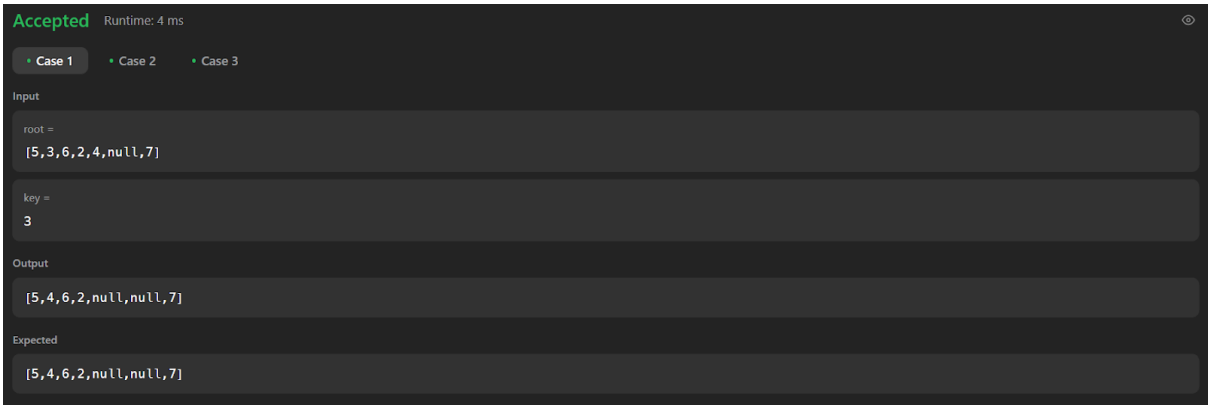
Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST*.

Basically, the deletion can be divided into two stages:

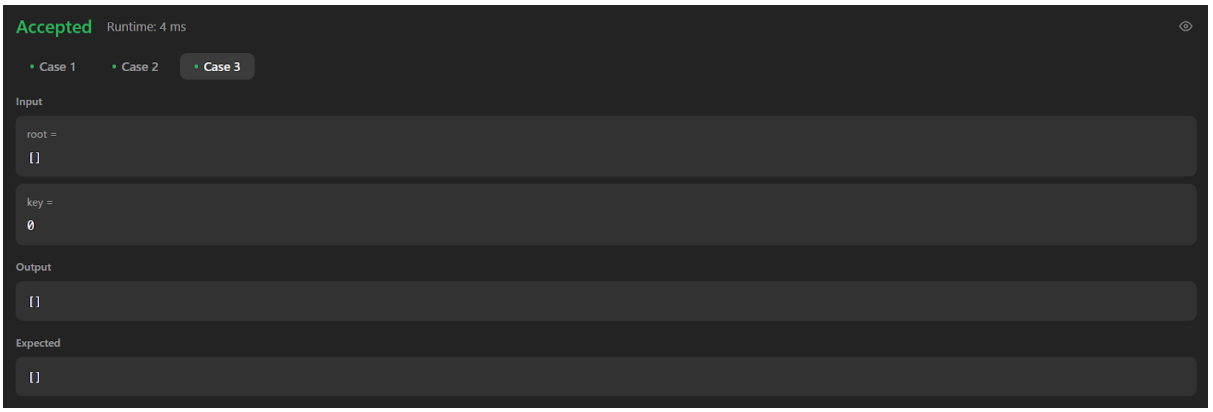
1. Search for a node to remove.
2. If the node is found, delete the node.

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

    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 = root->right;
        while (temp && temp->left != NULL)
            temp = temp->left;
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
```

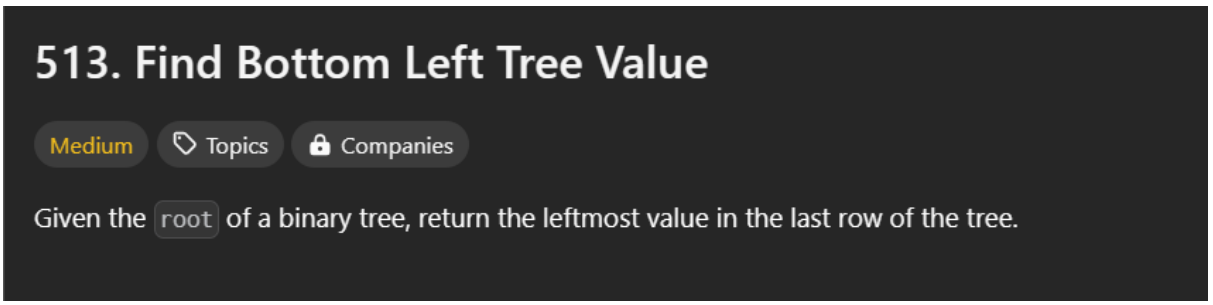
```
    return root;
}
```


Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
root =
[5,3,6,2,4,null,7]
key =
3
Output
[5,4,6,2,null,null,7]
Expected
[5,4,6,2,null,null,7]

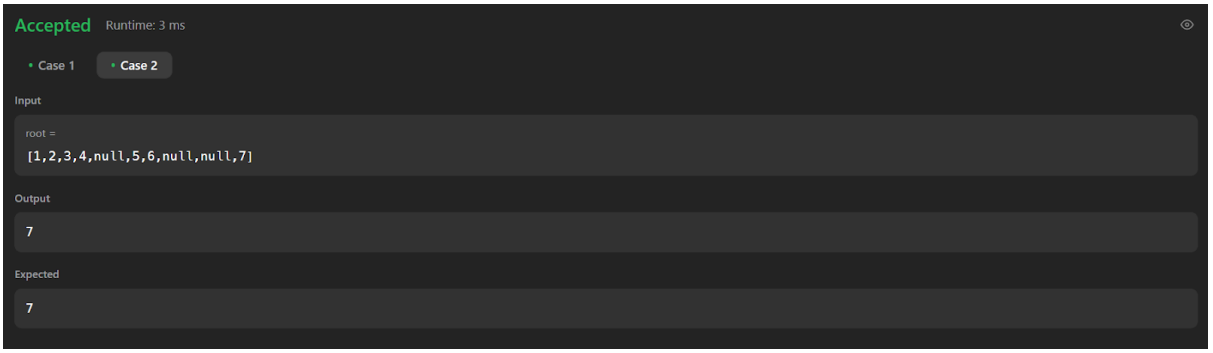

Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
root =
[5,3,6,2,4,null,7]
key =
0
Output
[5,3,6,2,4,null,7]
Expected
[5,3,6,2,4,null,7]


Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
root =
[]
key =
0
Output
[]
Expected
[]

Program 18
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;
}
```


Accepted
Runtime: 3 ms
Case 1
Case 2
Input
root =
[1,2,3,4,null,5,6,null,null,7]
Output
7
Expected
7


Accepted
Runtime: 3 ms
Case 1
Case 2
Input
root =
[2,1,3]
Output
1
Expected
1

Program 19

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;
}
```

```
PS C:\Users\kadab\OneDrive\Desktop\DS> gcc ninteen.c
PS C:\Users\kadab\OneDrive\Desktop\DS> .\a.exe
Enter the number of employees: 2
Enter key for employee 1: 234
Enter key for employee 2: 567
Enter key to search: 234
Employee with key 234 found at index 4.
```