VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ROHIT RAMCHNADRA GANDHI
(1BM23CS417)

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 **ROHIT RAMCHANDRA GANDHI (1BM23CS417)**, 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.

M Lakshmi Neelima	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

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

#define MAX_SIZE 5

// Structure to represent the stack
struct Stack {
   int arr[MAX_SIZE];
   int top;
};
struct Stack stack={{0,0,0,0,0},-1};
// Function to initialize the stack



// Function to check if the stack is empty
int isEmpty() {
   if(stack.top ==-1){
      return 1;
   }
   else{
      return 0;
   }
}

// Function to check if the stack is full
int isFull() {
   return stack.top == MAX_SIZE - 1;
}

// Function to push an element onto the stack
void push(int value) {
   if (isFull(stack)) {
      printf("Stack Overflow! Cannot push %d.\n", value);
   } else {
      stack.arr[++(stack.top)] = value;
      printf("Pushed %d onto the stack.\n", value);
   }

}

// Function to pop an element from the stack
```

```
void pop() {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop from an empty stack.\n");
    } else {
        printf("Popped %d from the stack.\n", stack.arr[stack.top--]);
    }

}

// Function to display the elements of the
stack void display() {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= stack.top; i++) {
            printf("%d ", stack.arr[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;

    int choice,
    value; do {
        printf("\n1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
               push(value)
                ; break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program.\n");
```

```
                break
            ;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);
    getchar();
    return 0;
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc LAB_1.c -o LAB_1 } ; if ($?)
 { .\LAB_1 }

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 1
Pushed 1 onto the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 4
Pushed 4 onto the stack.

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

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 4 from the stack.

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

Lab Program 2:

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

```
#include<stdio.h>
#include<ctype.h
> #define SIZE
50 char
stack[SIZE]; int
top=-1;
void push(char x){
   top++;
   stack[top]=x;
}
char
   pop(){
   char x;
   return(stack[top--]);
}
      int pr(char
       symbol){
  if(symbol=='^'){
      return (3);
   }
   else if(symbol=='*' || symbol=='/'){
      return(2);
   }
   else if(symbol=='+' || symbol=='-'){
      return(1);
   }
   else{
      return(0);
   }
}
void main(){
   char infix[50],postfix[50],ch,elem;
   int i=0,k=0;
   printf("Enter infix expression:");
   scanf("%s",infix);
   push('#');
   while((ch=infix[i++])!='\0')
   {
      if(ch=='(')
      push (ch);
      else
      if(isalnum(ch))
      postfix[k++]=ch
      ; else
      if(ch==')')
      {
```

```
while(stack[top]!='('){
    postfix[k++]=pop()
    ;
```

```
            elem=pop();
        }
        else{
            while(pr(stack[top])>=pr(ch))
                { postfix[k++]=pop();
            }
            push(ch);
        }
    }
    while(stack[top]!='#')
        {
        postfix[k++]=pop()
        ;
    }
    postfix[k]='\0';
    printf("\n Postfix Expression =%s \n",postfix);

}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc InfixPostfi
x.c -o InfixPostfix } ; if ($?) { .\InfixPostfix }
Enter infix expression:A+(B*C-(D/E^F)*G)*H

 Postfix Expression =ABC*DEF^/G*-H*+
PS C:\PLC\DATA STRUCTURES>
```

Lab Progarm 3:

WAP to simulate the working of a queue of integers using an array. Provide the following operations
a) Insert
b) Delete
c) Display
The program should print appropriate messages for queue empty and queue overflow Conditions.

```
#include<stdio.h>
#include<stdlib.h
> #define SIZE 5
void enqueue();
void dequeue();
void show();
struct queue{
    int arr[SIZE];
    int top;
    int rear;
};
struct queue q={{0,0,0,0,0},-1,-1};
void enqueue(){
    int item;
    if(q.rear == SIZE-1){
        printf("OverFlow \n");
    }
    else{
        if(q.top ==-1 || q.top>=0){
            q.top=0;
            printf("Enter the element to insert:");
            scanf("%d",&item);
            printf("\n");
            q.rear+=1;
            q.arr[q.rear]=item
            ;
        }
    }
}
void dequeue(){
    if(q.top==-1 || q.top>q.rear){
        printf("UnderFlow \n");
        return;
    }
    else{
        printf("Element deleted:%d \n",q.arr[q.top]);
        q.top=q.top+1;
    }
}
void show(){
    if(q.top == -1){
```

```
        printf("Empty Queue \n");
    }
    else{
        printf("Queue: \n");
        for(int i=q.top;i<=q.rear;i++){
            printf("%d ",q.arr[i]);
        }
        printf("\n");
    }
}
int main(){
    int ch;
    while(1)
    {
        printf("1 for Enqueue \n");
        printf("2 for Dequeue \n");
        printf("3 for Display \n");
        printf("4 Exit \n");
        printf("Enter your choice \n");
        scanf("%d",&ch);
        printf("\n");
        switch(ch){
            case 1:
            enqueue()
            ; break;
            case 2:
            dequeue()
            ; break;
            case 3:
            show();
            break;
            case 4:
            exit(0);
            default:
            printf("Wrong Choice \n");
            break;
        }
    }
}
```

Output:

```
Enter the element to insert:2

1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
1

Enter the element to insert:3

1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
3

Queue:
2 3
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

Element deleted:2
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

Element deleted:3
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

UnderFlow
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
4

PS C:\PLC\DATA STRUCTURES>
```

Lab Program 4:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.
a) Insert
b) Delete
c) Display
The program should print appropriate messages for queue empty and queue overflow Conditions.

```
#include <stdio.h>

#define MAX_SIZE 5

// Circular Queue variables
int items[MAX_SIZE];
int front = -1, rear = -1;

// Function to check if the queue is
empty int isEmpty() {
    return (front == -1 && rear == -1);
}

// Function to check if the queue is full
int isFull() {
    return ((rear + 1) % MAX_SIZE == front);
}

// Function to enqueue an element into the circular queue
void enqueue(int value) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue %d.\n", value);
        return;
    }

    if (isEmpty()) {
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
    }

    items[rear] = value;
    printf("%d enqueued to the queue.\n", value);
}

// Function to dequeue an element from the circular queue
int dequeue() {
    int dequeuedItem;

    if (isEmpty()) {
```

```
printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    dequeuedItem = items[front];
    if (front == rear) {
        // If there was only one element in the queue
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }

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

// Function to display the elements of the circular queue
void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    int i = front;
    do {
        printf("%d ", items[i]);
        i = (i + 1) % MAX_SIZE;
    } while (i != (rear + 1) % MAX_SIZE);
    printf("\n");
}

int main() {
    int choice, value;

    do {
        printf("\nCircular Queue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
```

```
            break;

        case 2:
            dequeue();
            break;

        case 3:
            display();
            break;

        case 4:
            printf("Exiting the program.\n");
            break;

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

    } while (choice != 4);

    return 0;
}
```

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Circular_Queue.c -o Circular_Queue } ; if ($?) { .\Circular_Queue }

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

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

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

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

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

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

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

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue is empty. Cannot dequeue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 5:

WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, and at end of list.
c) Delete a node at front and at the end of the list.
d) Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h
> struct Node{
    int data;
    struct Node *next;
};
//Create Linked List
struct Node* createLL(struct Node* head){
    int num;

    printf("Enter -1 to stop.\n");
    printf("Enter Number:");
    scanf("%d",&num);
    while(num!=-1){
        struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
        struct Node* p;
        p=head;
        if(head==NULL)
        {
            newNode->data=num;
            newNode->next=NULL
            ; head=newNode;
        }
        else{

            while(p->next!=NULL)
                { p=p->next;
            }
            newNode->data=num
            ; p->next=newNode;
            newNode->next=NULL;
        }
    printf("Enter Number:");
    scanf("%d",&num);
    }
    return head;
}
//Display Linked List
struct Node* displayLL(struct Node* head){
    struct Node*p;
    p=head;
    printf("Linked List Elements:");
    while(p !=NULL){
```

```
 printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
    return head;
}
//Insert a node at First Position
struct Node* insertAtBeg(struct Node* head){
    int num;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct
    Node)); newNode->data=num;
    newNode->next=head
    ; head=newNode;
    return head;
}
//Insert a node at End Position
struct Node* insertAtEnd(struct Node* head){
    int num;
    struct Node
    *p; p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct
    Node)); while(p->next!=NULL){
        p=p->next;
    }
    newNode->data=num
    ; p->next=newNode;
    newNode->next=NULL
    ; return head;
}
//Insert a node at any Position
struct Node* insertAtPos(struct Node* head,int pos){
    int num,i=0;
    struct Node
    *p; p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct
    Node)); while(i!=pos-1){
        p=p->next
        ; i++;
    }
    newNode->data=num;
    newNode->next=p->next
    ; p->next=newNode;
    return head;
}
//Delete a node at front
```

```
struct Node* delAtFront(struct Node* head){
    if(head==NULL){
        printf("Linked List alredy empty.\n");
        return head;
    }
    else{
    struct Node*
    p;
    p=head->next;
    free(head);
    head=p;
    return head;
    }
}
//Delete a node at end
struct Node* delAtEnd(struct Node* head){
    struct Node *p,*preNode;
    p=head;
    while(p->next!=NULL)
        { preNode=p;
        p=p->next;
    }
    preNode->next=NULL
    ; free(p);
    return head;
}
//Delete a node at any position
struct Node* delAtPos(struct Node* head, int pos){
    struct Node*p,*preNode;
    int i=0;
    p=head;
    if(pos==0){
        head=delAtFront(head)
        ; return head;
    }
    while(i!=pos)
        {
        preNode=p;
        p=p->next;
        i++;
    }
    preNode->next=p->next
    ; free(p);
    return head;
}
int main(){
    struct Node* head=NULL;
    head=createLL(head);
    head=displayLL(head);

    head=insertAtBeg(head);
    printf("Linked list after insertion at begining.\n");
```

```
    head=insertAtEnd(head);
    printf("Linked list after insertion at end.\n");
    head=displayLL(head);

    head=insertAtPos(head,2);
    printf("Linked list after insertion at position.\n");
    head=displayLL(head);

    head=delAtFront(head);
    printf("Linked list after deletion at front.\n");
    head=displayLL(head);

    head=delAtEnd(head);
    printf("Linked list after deletion at end.\n");
    head=displayLL(head);

    head=delAtPos(head,0);
    printf("Linked list after deletion from pos.\n");
    head=displayLL(head);
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Singl
yLinkedList.c -o SinglyLinkedList } ; if ($?) { .\SinglyLinkedList }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:4
Enter Number:-1
Linked List Elements:1 2 3 4
Enter Number:7
Linked list after insertion at begining.
Linked List Elements:7 1 2 3 4
Enter Number:9
Linked list after insertion at end.
Linked List Elements:7 1 2 3 4 9
Enter Number:5
Linked list after insertion at position.
Linked List Elements:7 1 5 2 3 4 9
Linked list after deletion at front.
Linked List Elements:1 5 2 3 4 9
Linked list after deletion at end.
Linked List Elements:1 5 2 3 4
Linked list after deletion from pos.
Linked List Elements:5 2 3 4
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 6:

WAP to Implement Circular Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, and at end of list and at any position
c) Delete a node at front and at the end of the list and at any position
d) Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h
> struct Node{
   int data;
   struct Node *next;
};
//Create Circular Linked List
struct Node* createCircularLL(struct Node* head){
   int data;
   struct Node*p;
   printf("Enter -1 to stop.\n");
   printf("Enter Number:");
   scanf("%d",&data);
   while(data!=-1){

      struct Node *newNode=(struct Node*)malloc(sizeof(struct Node));
      newNode->data=data;
      if(head==NULL){
         newNode->next=newNode;
         head=newNode;
      }
      else{
         p=head;
         while(p->next!=head)
            { p=p->next;
         }
         p->next=newNode;
         newNode->next=head
         ;

      }
   printf("Enter Number:");
   scanf("%d",&data);
   }
   return head;
}
//Display Circular Linked List
struct Node* displayCircularLL(struct Node* head){
   struct Node* p;
   p=head;
   printf("Circular List Elements:");
   while(p->next !=head){
      printf("%d ",p->data);
      p=p->next;
   }
```

```
printf("%d ",p->data);
    printf("\n");
    return head;
}
//Insert At Begining
struct Node* insertFirst(struct Node* head){
    int num;
    struct Node
    *p; p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node *newNode=(struct Node*)malloc(sizeof(struct
    Node)); newNode->data=num;
    while(p->next!=head)
        { p=p->next;
    }
    p->next=newNode;
    newNode->next=head
    ; head=newNode;
    return head;
}
//Insert At End
struct Node* insertEnd(struct Node* head){
    int num;
    struct Node
    *p; p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node *newNode=(struct Node*)malloc(sizeof(struct
    Node)); newNode->data=num;
    while(p->next!=head)
        { p=p->next;
    }
    p->next=newNode;
    newNode->next=head
    ; return head;
}
//Insert At Any Position
struct Node* insertPosition(struct Node* head, int pos){
    int num,i=0;
    struct Node
    *p; p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node *newNode=(struct Node*)malloc(sizeof(struct
    Node)); newNode->data=num;
    if(pos==0){
        head=insertFirst(head);
        return head;
    }
    else{
```

```
 while(i!=pos-1){
           p=p->next
           ; i++;
       }
    newNode->next=p->next
    ; p->next=newNode;
    return head;
    }

}
//Delete From Front
struct Node* DelFromFront(struct Node* head){
    struct Node *p;
    p=head;
    while(p->next!=head)
        { p=p->next;
    }
    p->next=head->next
    ; free(head);
    head=p->next;
}
//Delete From End
struct Node* DelFromEnd(struct Node* head){
    struct Node *p,*preNode;
    p=head;
    while(p->next!=head){
        preNode=p;
        p=p->next;
    }
    preNode->next=p->next
    ; free(p);
    return head;
}
//Delete From Any Position
struct Node* DelFromPos(struct Node* head,int
    pos){ int i=0;
    struct Node* p,*preNode;
    p=head;
    if(pos==0){
        head=DelFromFront(head)
        ; return head;
    }
    else{
        while(i!=pos)
            {
            preNode=p;
            p=p->next;
            i++;
        }
    preNode->next=p->next
    ; free(p);
    return head;
```

```
}
}
int main(){
    struct Node* head=NULL;
    head=createCircularLL(head);
    printf("Linked List Created. \n");
    head=displayCircularLL(head);

    head=insertFirst(head);
    printf("Linked List after insertion at begining.\n");
    head=displayCircularLL(head);

    head=insertEnd(head);
    printf("Linked List after insertion at end.\n");
    head=displayCircularLL(head);

    head=insertPosition(head, 3);
    printf("Linked List after insertion at pos.\n");
    head=displayCircularLL(head);

    head=DelFromFront(head);
    printf("Linked List after deletion from
    front\n"); head=displayCircularLL(head);

    head=DelFromEnd(head);
    printf("Linked List after Deletion form end.\n");
    head=displayCircularLL(head);

    head=DelFromPos(head,2);
    printf("Linked List after Deletion form pos.\n");
    head=displayCircularLL(head);
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Circ
ular_LL.c -o Circular_LL } ; if ($?) { .\Circular_LL }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:-1
Linked List Created.
Circular List Elements:1 2 3
Enter Number:10
Linked List after insertion at begining.
Circular List Elements:10 1 2 3
Enter Number:7
Linked List after insertion at end.
Circular List Elements:10 1 2 3 7
Enter Number:5
Linked List after insertion at pos.
Circular List Elements:10 1 2 5 3 7
Linked List after deletion from front
Circular List Elements:1 2 5 3 7
Linked List after Deletion form end.
Circular List Elements:1 2 5 3
Linked List after Deletion form pos.
Circular List Elements:1 2 3
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 7:

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

```
#include<stdio.h>
#include<stdlib.h
> struct Node{
   int data;
   struct Node *next;
};
//Create Linked List
struct Node* createLL(struct Node* head){
   int num;

   printf("Enter -1 to stop.\n");
   printf("Enter Number:");
   scanf("%d",&num);
   while(num!=-1){
      struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
      struct Node* p;
      p=head;
      if(head==NULL)
      {
         newNode->data=num;
         newNode->next=NULL
         ; head=newNode;
      }
      else{

         while(p->next!=NULL)
            { p=p->next;
         }
         newNode->data=num
         ; p->next=newNode;
         newNode->next=NULL;
      }
   printf("Enter Number:");
   scanf("%d",&num);
   }
   return head;
}
//Display Linked List
struct Node* displayLL(struct Node* head){
   struct Node*p;
   p=head;
   printf("Linked List Elements:");
   while(p !=NULL){
      printf("%d ",p->data);
      p=p->next;

}
```

```
 printf("\n");
    return head;
}
//Sort Linked List
struct Node* sortLL(struct Node* head){
    struct Node* ptr,*trav;
    int temp;
    ptr=head;
    while(ptr->next != NULL){
        trav=ptr->next;
        while(trav!=NULL){
            if(ptr->data > trav->data){
                temp=ptr->data;
                ptr->data=trav->data
                ; trav->data=temp;
            }
            trav=trav->next;
        }
        ptr=ptr->next;
    }
    return head;
}
struct Node* LLRev(struct Node* head){
    struct Node* temp;
    struct Node*
    prev=NULL; struct
    Node* cur=head;
    while(cur!=NULL){
        temp=cur->next
        ;
        cur->next=prev;
        prev=cur;
        cur=temp;
    }
    head=prev;
    return head;
}
struct Node* ConcatLL(struct Node* head1,struct Node*
    head2){ struct Node*ptr;
    ptr=head1;
    while(ptr->next!=NULL)
        { ptr= ptr->next;
    }
    ptr->next=head2
    ; return head1;

}
int main(){
    struct Node* head=NULL;
    struct Node*
    head1=NULL; struct
    Node* head2=NULL;
```

```
head=displayLL(head);
    printf("After Sorting \n");
    head=sortLL(head);
    head=displayLL(head);
    printf("After Reversal \n");
    head=LLRev(head);
    head=displayLL(head);
    printf("Enter 1st Linked List : \n");
    head1=createLL(head1);
    printf("Enter 2nd Linked List : \n");
    head2=createLL(head2);
    printf("After Concatenation \n");
    head1=ConcatLL(head1,head2);
    head1=displayLL(head1);
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Sort
_Reverse_Concatenate_LL.c -o Sort_Reverse_Concatenate_LL } ; if ($?) { .\Sor
t_Reverse_Concatenate_LL }
Enter -1 to stop.
Enter Number:4
Enter Number:3
Enter Number:5
Enter Number:-1
Linked List Elements:4 3 5
After Sorting
Linked List Elements:3 4 5
After Reversal
Linked List Elements:5 4 3
Enter 1st Linked List :
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:-1
Enter 2nd Linked List :
Enter -1 to stop.
Enter Number:4
Enter Number:5
Enter Number:6
Enter Number:-1
After Concatenation
Linked List Elements:1 2 3 4 5 6
PS C:\PLC\DATA STRUCTURES>
```

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

Stack Implementation :-

```
#include<stdio.h>
#include<stdlib.h
> struct Node{
    int data;
    struct Node* next;
};
int isEmpty(struct Node* top){
    if(top == NULL){
        return 1;
    }
    return 0;
}
struct Node* displayLL(struct Node* top){
    if(isEmpty(top)){
        printf("No elements to print.\n");
        return top;
    }
    printf("Linked list elements:");
    struct Node* p =top;
    while(p!=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n")
    ; return top;
}
struct Node* push(struct Node* top,int data){
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    if(top==NULL){
        newNode->data=data;
        top=newNode;
        newNode->next=NULL
        ; return top;
    }
    else{
        newNode->data=data
        ;
        newNode->next=top;
        top=newNode;
        return top;
    }
}
struct Node* pop(struct Node* top){
    if(isEmpty(top)){
        printf("Stack is empty.\n");
    }
    else{
        struct Node* p =top;
```

```
top=p->next;
        free(p);
        return top;
    }
}
int peek(struct Node* top){
    struct Node* p =top;
    return p->data;
}
int main(){
    struct Node* top=NULL;
    top=push(top,5);
    top=push(top,7);
    top=push(top,9);
    printf("Linked list after push operation \n");
    top=displayLL(top);
    printf("Linked list after pop operation \n");
    top=pop(top);
    top=displayLL(top)
    ; int x=peek(top);
    printf("Top Element: %d \n",x);
    top=push(top,10);
    printf("Linked list after push operation \n");
    top=displayLL(top);
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc StackUsingLL.c -o StackUsingLL } ; if ($?) { .\StackUsingLL }
Linked list after push operation
Linked list elements:9 7 5
Linked list after pop operation
Linked list elements:7 5
Top Element: 7
Linked list after push operation
Linked list elements:10 7 5
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 8:

WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h
> struct Node{
    int data;
    struct Node* prev;
    struct Node* next;
};
//Create a DLL
struct Node* createDLL(struct Node* head){
    int num;
    printf("Enter -1 to stop.\n");
    printf("Enter Number:");
    scanf("%d",&num);
    while(num!=-1)
    { struct Node*
    ptr;
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if(head==NULL){
        newNode->data=num;
        newNode->prev=NULL
        ;
        newNode->next=NULL
        ; head=newNode;
    }
    else{
        ptr=head;
        while(ptr->next!=NULL)
            { ptr=ptr->next;
        }
        newNode->data=num
        ;
        newNode->prev=ptr;
        ptr->next=newNode;
        newNode->next=NULL;
    }
    printf("Enter Number:");
    scanf("%d",&num);
    }
    return head;

}
//Display Linked List
struct Node* displayLL(struct Node* head){
    struct Node*p;
```

```
while(p !=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
    return head;
}

//Insert a new node to the left of the node
struct Node* insertLeft(struct Node*
head){
    struct Node*ptr;
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    int n,val;
    printf("Enter Number:");
    scanf("%d",&n);
    printf("Enter the value before which number is to be inserted:");
    scanf("%d",&val);
    ptr=head;
    while(ptr->data!=val)
        { ptr=ptr->next;
    }
    newNode->data=n;
    newNode->next=ptr;
    newNode->prev=ptr->prev
    ;
    ptr->prev->next=newNode
    ; ptr->prev=newNode;
    return head;

}
//Delete the node based on a specific value
struct Node* deleteNode(struct Node* head){
    int val;
    struct Node* ptr;
    printf("Enter the value for which node is to be deleted:");
    scanf("%d",&val);
    ptr=head;
    while(ptr->data!=val)
        { ptr=ptr->next;
    }
    ptr->prev->next=ptr->next
    ;
    ptr->next->prev=ptr->prev
    ; return head;
}
int main(){
    struct Node* head=NULL;
    head=createDLL(head);
    head=displayLL(head);
    head=insertLeft(head);
    head=displayLL(head);
```

```
    head=deleteNode(head);
    head=displayLL(head);
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Doub
ly_Linked_List.c -o Doubly_Linked_List } ; if ($?) { .\Doubly_Linked_List }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:4
Enter Number:-1
Linked List Elements:1 2 3 4
Enter Number:7
Enter the value before which number is to be inserted:2
Linked List Elements:1 7 2 3 4
Enter the value for which node is to be deleted:3
Linked List Elements:1 7 2 4
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 9:

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

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

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

struct node {
    struct node* left;
    int data;
    struct node* right;
};

struct node* CreateNode(int ele) {
    struct node* nn = (struct node*)malloc(sizeof(struct node));
    if (nn == NULL) {
        printf("Memory Can't be allocated");
    }
    else {
        nn->data = ele;
        nn->left = NULL;
        nn->right = NULL;
        return nn;
    }
}


struct node* insert(struct node* root, int data) {
    if (root == NULL) {
        root = CreateNode(data);
    }
    else if (data >= root->data) {
        root->right = insert(root->right, data);
    }
    else if (data < root->data) {
        root->left = insert(root->left, data);
    }
    return root;
}
void inordertrav(struct node* root) {
    if (root == NULL) {
        return;
    }
    inordertrav(root->left);
```

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

void postordertrav(struct node* root) {
    if (root == NULL) {
        return;
    }
    postordertrav(root->left);
    postordertrav(root->right)
    ; printf("%d ",
    root->data);
}
void preordertrav(struct node* root) {
    if (root == NULL) {
        return;
    }
    printf("%d ", root->data);
    preordertrav(root->left);
    preordertrav(root->right)
    ;
}
struct node* findImmediatePredecessor(struct node* root, int key) {
    struct node* pre = NULL;
    while (root) {
        if (root->data < key) {
            pre = root;
            root = root->right;
        }
        else if (root->data >= key) {
            root = root->left;
        }
    }
    return pre;
}
struct node* findImmediateSuccessor(struct node* root, int key) {
    struct node* suc = NULL;
    while (root) {
        if (root->data > key) {
            suc = root;
            root = root->left;
        }
        else if (root->data <= key) {
            root = root->right;
        }
    }
    return suc;
}

int main() {
```

```
    root = insert(root, 14);
    root = insert(root, 5);
    root = insert(root, 44);
    root = insert(root, 3);
    root = insert(root, 7);
    root = insert(root, 100);
    root = insert(root, 46);
    root = insert(root, 8);
    root = insert(root, 10);
    root = insert(root, 11);
    root = insert(root, 17);
    root = insert(root, 25);
    root = insert(root, 23);
    root = insert(root, 34);
    root = insert(root, 16);
    root = insert(root, 50);
    root = insert(root, 1);
    root = insert(root, 6);
    root = insert(root, 2);

    printf("In Order Traversal: ");
    inordertrav(root);
    printf("\n");

    printf("Post Order Traversal: ");
    postordertrav(root);
    printf("\n");

    printf("Pre Order Traversal: ");
    preordertrav(root);
    printf("\n");

    printf("Enter Key to search:");
    scanf("%d",&key);
    struct node* pre = findImmediatePredecessor(root, key);
    struct node* suc = findImmediateSuccessor(root, key);

    if (pre)
        printf("Immediate Predecessor of %d is %d\n", key, pre->data);
    else
        printf("No Immediate Predecessor of %d\n", key);

    if (suc)
        printf("Immediate Successor of %d is %d\n", key, suc->data);
    else
        printf("No Immediate Successor of %d\n", key);

    return 0;
}
```

Output:

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc BinarySearchTree.c -o BinarySearchTree } ; if ($?) { .\BinarySearchTree }
In Order Traversal: 1 2 3 5 6 7 8 10 11 14 16 17 23 25 34 44 46 50 100
Post Order Traversal: 2 1 3 6 11 10 8 7 5 16 23 34 25 17 50 46 100 44 14
Pre Order Traversal: 14 5 3 1 2 7 6 8 10 11 44 17 16 25 23 34 100 46 50
Enter Key to search:17
Immediate Predecessor of 17 is 16
Immediate Successor of 17 is 23
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 10 :

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.
Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.
Let the keys in K and addresses in L are integers.
Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m
(remainder method), and implement hashing technique to map a given key K to the address
space L.
Resolve the collision (if any) using
i. linear probing
ii. Quadratic Probing
iii. Double Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 100
int L[MAX_SIZE];
int count = 0;
int hash_lprobe(int key) {
    int i = 0;
    while (L[(key + i) % MAX_SIZE] != 0) {
        i++;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key + i) % MAX_SIZE;
}
int hash_qprobe(int key) {
    int i = 0;
    while (L[(key + i * i) % MAX_SIZE] != 0)
        { i++;
        if (i == MAX_SIZE)
            return -1;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key + i * i) % MAX_SIZE;
}
int double_hash(int key) {
    int i = 0;
```

```
    while (L[(key % MAX_SIZE + 97 - key % 97) % MAX_SIZE] != 0) {
        i++;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key % MAX_SIZE + 97 - key % 97) % MAX_SIZE;
}
int search_lp(int key) {
    int i = 0;
    while (L[(key + i) % MAX_SIZE] != key) {
        if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key + i) % MAX_SIZE;
}
int search_qp(int key) {
    int i = 0;
    while (L[(key + i * i) % MAX_SIZE] != key)
        { if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key + i * i) % MAX_SIZE;
}

int search_db(int key) {
    int i = 0;
    while (L[(key % MAX_SIZE + 97 - key % 97) % MAX_SIZE] != key) {
        if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key % MAX_SIZE + 97 - key % 97) % MAX_SIZE;
}

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

    while (1) {
```

```
    printf("Enter choice:");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the key:");
            scanf("%d", &key);
            printf("1)Linear\n2)Quadratic\n3)Double Hash\n");
            scanf("%d", &subChoice);

            if (subChoice == 1)
                key = hash_lprobe(key);
            else if (subChoice == 2)
                key = hash_qprobe(key);
            else
                key = double_hash(key);

            L[key] =
            key; if (key
            != -1)
                printf("Value %d inserted\n", key);
            break;

        case 2:
            printf("Enter the key:");
            scanf("%d", &key);
            printf("1)Linear\n2)Quadratic\n3)Double Hash\n");
            scanf("%d", &subChoice);

            if (subChoice == 1)
                                            key =
                 search_lp(key); else if
              (subChoice == 2) key =
                       search_qp(key);
            else
                key = search_db(key);

            if (key != -1)
                printf("Value %d found at %d\n", L[key], key);
            break;

        case 3:
            exit(0);

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

```c
    return 0;
}
```

Output

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Hashing.c -o Hashing } ; if ($?) { .\Hashing }
1)Insert
2)Search
3)Exit
Enter choice:1
Enter the key:42
1)Linear
2)Quadratic
3)Double Hash
1
Value 42 inserted
Enter choice:2
Enter the key:42
1)Linear
2)Quadratic
3)Double Hash
1
Value 42 found at 42
Enter choice:1
Enter the key:55
1)Linear
2)Quadratic
3)Double Hash
1
Value 55 inserted
Enter choice:1
Enter the key:68
1)Linear
2)Quadratic
3)Double Hash
1
Value 68 inserted
Enter choice:2
Enter the key:68
1)Linear
2)Quadratic
3)Double Hash
1
Value 68 found at 68
Enter choice:3
PS C:\PLC\DATA STRUCTURES>
```

LEETCODE 1:

Demonstration of stack program

```
typedef struct {
int *stack;
int *minStack;
int top
} MinStack;


MinStack* minStackCreate() {
MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
stack->stack = (int*)malloc(sizeof(int) * 10000);
stack->minStack = (int*)malloc(sizeof(int) * 10000);
stack->top = -1;
return stack;
}

void minStackPush(MinStack* obj, int val) {
obj->top++;
obj->stack[obj->top] = val;
if (obj->top == 0 || val <= obj->minStack[obj->top - 1]) {
obj->minStack[obj->top] = val;
} else {
obj->minStack[obj->top] = obj->minStack[obj->top - 1];
}
}

void minStackPop(MinStack* obj) {
obj->top--;
}

int minStackTop(MinStack* obj) {
return obj->stack[obj->top];
}

int minStackGetMin(MinStack* obj) {
return obj->minStack[obj->top];
}

void minStackFree(MinStack* obj) {
free(obj->stack);
free(obj->minStack);
free(obj);
```

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1

Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

```
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

```
[null,null,null,null,-3,null,0,-2]
```

LEETCODE 2:

Demonstration of LeetCode program on Singly linked list

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int
right) {
    if (head == NULL|| left == right)
    {
        return head;
    }
    struct ListNode *dummy = (struct ListNode*)malloc(sizeof(struct
ListNode));
    dummy->next=head;
    struct ListNode *prev=dummy;
    for(int i=1; i<left; i++)
    {
        prev=prev->next;
    }
    struct ListNode* current=prev->next;
    struct ListNode* next=NULL;
    struct ListNode* tail=current;
    for(int i=left; i<=right; i++){
        struct ListNode* temp=current->next;
        current->next=next;
        next=current;
        current=temp;
    }
    prev->next=next;
    tail->next=current;
    struct ListNode* result=dummy->next;
    free(dummy);
    return result;
}
```

Testcase | >_ Test Result

Accepted Runtime: 6 ms

• Case 1 • Case 2

Input

head =

[1,2,3,4,5]

left =

2

right =

4

Output

[1,4,3,2,5]

Expected

[1,4,3,2,5]

LEETCODE 3:

Demonstration of LeetCode program on Singly linked list

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize)
{
    // Calculate the length of the linked list int
    length = 0;
    struct ListNode* current = head;
    while (current != NULL) {
        length++;
        current = current->next;
    }

    // Calculate the size of each part and the number of nodes that will have
an extra node
    int partSize = length / k; int
    extraNodes = length % k;

    // Initialize the array to store the heads of the parts
    struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct
ListNode*));

    // Split the linked list into parts
    current = head;
    for (int i = 0; i < k; i++) {
   // Determine the size of the current part
   int currentPartSize = partSize + (i < extraNodes ? 1 : 0);

   // Store the head of the current part in the result array result[i]
   = current;

   // Move to the end of the current part
   for (int j = 0; j < currentPartSize - 1 && current != NULL; j++) { current =
   current->next;
   }

   // If there are more nodes, break the link between parts if
   (current != NULL) {
   struct ListNode* nextNode = current->next;
   current->next = NULL;
   current = nextNode;
   }
```

Testcase | >_ Test Result

Accepted Runtime: 6 ms

• Case 1 • Case 2

Input

head =

[1,2,3,4,5]

left =

2

right =

4

Output

[1,4,3,2,5]

Expected

[1,4,3,2,5]

LEETCODE 4 :

```
struct ListNode* rotateRight(struct ListNode* head, int k) {
    if (head == NULL || head->next == NULL || k == 0) // If the list
is empty or has only one node or k is zero, no need to rotate
        return head;

    int length = 1;
    struct ListNode *tail = head;

    while (tail->next != NULL) { // Finding the length of the list
and the tail node
        length++;
        tail = tail->next;
    }

    k = k % length; // Adjusting k if it's greater than the length
of the list
    if (k == 0) // If k becomes 0 after adjustment, no rotation is
needed
        return head;

    struct ListNode *new_tail = head;
    for (int i = 0; i < length - k - 1; i++) { // Finding the new
tail node after rotation
        new_tail = new_tail->next;
    }

    struct ListNode *new_head = new_tail->next; // New head after
rotation
    new_tail->next = NULL; // Breaking the link between the new tail
and the next node to form a new list

    tail->next = head; // Connecting the original tail to the
original head to form a circular list
    return new_head;
}
```

Testcase | >_ Test Result

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

head =

```
[1,2,3,4,5]
```

k =

```
2
```

Output

```
[4,5,1,2,3]
```

Expected

```
[4,5,1,2,3]
```

Hacker Rank 1:

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

struct node
{
    int id;
    int depth;
    struct node *left, *right;
};

void
inorder(struct node* tree)
{
    if(tree == NULL)
        return;

    inorder(tree->left);
    printf("%d ",tree->id);
    inorder((tree->right));

}


int
main(void)
{
    int no_of_nodes, i = 0;
    int l,r, max_depth,k;
    struct node* temp = NULL;
    scanf("%d",&no_of_nodes);
    struct node* tree = (struct node *) calloc(no_of_nodes , sizeof(struct
node));
    tree[0].depth = 1;
    while(i <  no_of_nodes )
    {
        tree[i].id = i+1;
        scanf("%d %d",&l,&r);
        if(l ==  -1)
            tree[i].left = NULL;
        else
             {
                 tree[i].left = &tree[l-1];
                 tree[i].left->depth = tree[i].depth + 1;
                 max_depth = tree[i].left->depth;
             }
```

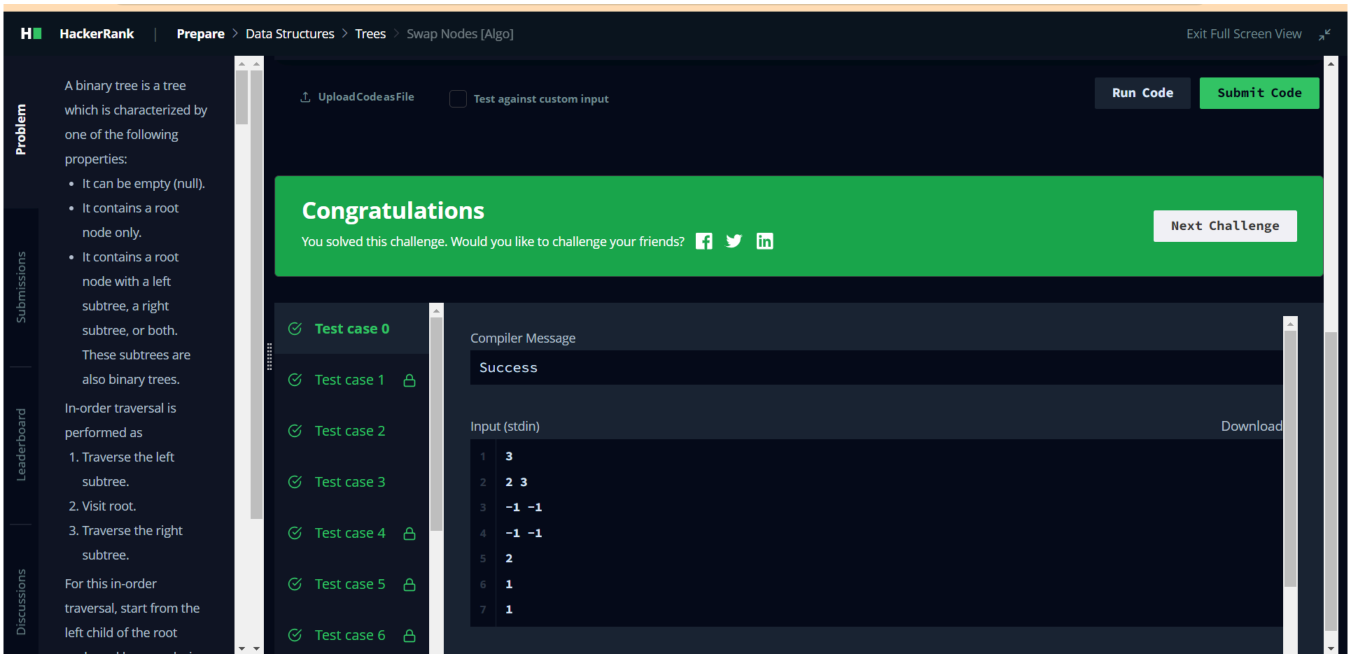
```
            if(r ==  -1)
                tree[i].right = NULL;
            else
                 {
                     tree[i].right = &tree[r-1];
                     tree[i].right->depth = tree[i].depth + 1;
                     max_depth = tree[i].right->depth+2;
                 }

        i++;
        }

        scanf("%d", &i);
        while(i--)
        {
            scanf("%d",&l);
            r = l;
            while(l <= max_depth)
            {
                for(k = 0;k < no_of_nodes; ++k)
                {
                    if(tree[k].depth == l)
                    {
                        temp = tree[k].left;
                        tree[k].left = tree[k].right;
                        tree[k].right = temp;
                    }
                }
                l = l + r;
            }
            inorder(tree);
            printf("\n");
        }



        return 0;
    }
```


HackerRank
Prepare > Data Structures > Trees > Swap Nodes [Algo]
Exit Full Screen View
Problem
Submissions
Leaderboard
Discussions
A binary tree is a tree which is characterized by one of the following properties:
• It can be empty (null).
• It contains a root node only.
• It contains a root node with a left subtree, a right subtree, or both. These subtrees are also binary trees.
In-order traversal is performed as
1. Traverse the left subtree.
2. Visit root.
3. Traverse the right subtree.
For this in-order traversal, start from the left child of the root
Upload Code as File
Test against custom input
Run Code
Submit Code
Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge
Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
Compiler Message
Success
Input (stdin)
Download
3
2 3
-1 -1
-1 -1
2
1
1