VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RAHUL N RAJU (1BM22CS215)

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 **RAHUL N RAJU (1BM22CS215)**, who is a 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. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
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.

```
//stack implementation

#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int top = -1;
int a[SIZE];
void push();
void pop();
void show();

void main()
{
    int ch;
    while (1)
    {
        printf("Operations on the stack:\n");
        printf("1.Push the element 2.Pop the element 3.Show 4.End\n");
        printf("Enter the choice: ");
        scanf("%d",&ch);

        switch (ch)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            show();
            break;
        case 4:
            exit(0);

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

void push()
```

```
{
    int x;
    if (top == SIZE - 1)
    {
        printf("Overflow\n");
    }
    else
    {
        printf("Enter the element to be added :\n ");
        scanf("%d", &x);
        top = top + 1;
        a[top] = x;
    }
}
void pop()
{
    if (top == -1)
    {
        printf("Underflow\n");
    }
    else
    {
        printf("Popped element: %d\n", a[top]);
        top = top - 1;
    }
}
void show()
{
    if (top == -1)
    {
        printf("Underflow\n");
    }
    else
    {
        printf("Elements in the stack are: \n");
        for (int i = top; i >= 0; i--)
            printf("%d\n", a[i]);
    }
}
```

OUTPUT :

```
C:\Users\Admin\Desktop\1bm22cs215\stack.exe
Operations on the stack:
1.Push the element 2.Pop the element 3.Show 4.End
Enter the choice: 1
Enter the element to be added :
 3
Operations on the stack:
1.Push the element 2.Pop the element 3.Show 4.End
Enter the choice: 1
Enter the element to be added :
 2
Operations on the stack:
1.Push the element 2.Pop the element 3.Show 4.End
Enter the choice: 2
Popped element: 2
Operations on the stack:
1.Push the element 2.Pop the element 3.Show 4.End
Enter the choice: 3
Elements in the stack are:
3
Operations on the stack:
1.Push the element 2.Pop the element 3.Show 4.End
Enter the choice: 4

Process returned 0 (0x0)   execution time : 14.749 s
Press any key to continue.
```

Lab program 2:

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

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char);
char pop();
int precedence(char);
void infixToPostfix(char infix[], char postfix[]);

void push(char item) {
    if (top == SIZE - 1) {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
    } else {
        top++;
        stack[top] = item;
    }
}

char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    } else {
        char popped = stack[top];
        top--;
        return popped;
    }
}

int precedence(char symbol) {
    if (symbol == '^') {
        return 3;
    } else if (symbol == '*' || symbol == '/') {
        return 2;
    } else if (symbol == '+' || symbol == '-') {
```

```c
        return 1;
    } else {
        return -1;
    }
}

void infixToPostfix(char infix[], char postfix[]) {
    int i = 0, j = 0;
    char symbol, temp;

    push('#');

    while ((symbol = infix[i++]) != '\0') {
        if (symbol == '(') {
            push(symbol);
        } else if (isalnum(symbol)) {
            postfix[j++] = symbol;
        } else if (symbol == ')') {
            while (stack[top] != '(') {
                postfix[j++] = pop();
            }
            temp = pop(); // Remove '(' from the stack
        } else {
            while (precedence(stack[top]) >= precedence(symbol)) {
                postfix[j++] = pop();
            }
            push(symbol);
        }
    }

    while (stack[top] != '#') {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
}

int main() {
    char infix[SIZE], postfix[SIZE];

    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("The postfix expression is: %s\n", postfix);

    return 0;
}
```

OUTPUT :

```
Enter a valid parenthesized infix expression: a*b+c*d-e
The postfix expression is: ab*cd*+e-


...Program finished with exit code 0
Press ENTER to exit console.
```

Lab program 3:

3a) WAP 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

```c
#include <stdio.h>
#define MAX 6

int Q[MAX];
int front = -1, rear = -1;

void insert(int element) {
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        Q[rear] = element;
    }
}

void delete() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return;
    } else {
        printf("Deleted element: %d\n", Q[front]);
        front++;
```

```
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    } else {
        printf("Elements in the queue: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", Q[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, element;

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

        switch (choice) {
            case 1:
                printf("Enter element to insert: ");
                scanf("%d", &element);
                insert(element);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please enter a valid choice.\n");
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT :

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter element to insert: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter element to insert: 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted element: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Elements in the queue: 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
```

3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#define SIZE 5

int items[SIZE];
int front = -1, rear = -1;

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

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

void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted -> %d", element);
  }
  printf("\n");
}

int deQueue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
    else {
      front = (front + 1) % SIZE;
    }
    printf("\n Deleted element -> %d \n", element);
    return (element);
```

```c
  }
  printf("\n");
}

void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
    printf("\n Rear -> %d \n", rear);
  }
  printf("\n");
}

void main()
{
    int option, val;
    int ele;
    do
    {
        printf("1.insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("enter your option : \n");
        scanf("%d",&option);
        switch(option)
        {
            case 1:
                    printf("enter the element : ");
                    scanf("%d", &ele);
                    enQueue(ele);
                    break;
            case 2: val = deQueue();
                    if(val != -1)
                        printf("the number deleted is : %d \n",val);
                    break;
            case 3:display();
                    break;
        }
    }while(option!=4);
}
```

OUTPUT :

```
1.insert
2.Delete
3.Display
4.Exit
enter your option :
1
enter the element : 1

 Inserted -> 1
1.insert
2.Delete
3.Display
4.Exit
enter your option :
1
enter the element : 2

 Inserted -> 2
1.insert
2.Delete
3.Display
4.Exit
enter your option :
2

 Deleted element -> 1
the number deleted is : 1
1.insert
2.Delete
3.Display
4.Exit
enter your option :
3

 Front -> 1
 Items -> 2
 Rear -> 1

1.insert
2.Delete
3.Display
4.Exit
```

Lab program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

```
// singly linked list

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

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

void insert(struct Node **head, int data)
{
    struct Node *newnode = (struct Node *)malloc (sizeof(struct Node));

    newnode ->
    data = data;
    newnode ->
    next= *head;

    *head = newnode;
}

void display (struct Node *node)
{
  printf ("\nLinked List: ");
  while (node != NULL)
    {
      printf ("%d ", node->data);
      node = node->next;
    }
  printf("\n");
}

void main()
{
    struct Node *head = NULL;
    insert(&head, 100);
    insert(&head, 80);
    insert(&head, 60);
```

```
    insert(&head, 40);
    display(head);
}
```

OUTPUT:

```
Linked List: 40 60 80 100


...Program finished with exit code 0
Press ENTER to exit console.
```

Program - Leetcode platform – MinStack

```
typedef struct {
    int str[8000];
    int top;
    int min[1000];
    int mincnt;
} MinStack;


MinStack* minStackCreate() {
    MinStack *Min;
    Min=(MinStack*)malloc(sizeof(MinStack));
    Min->top=-1;
    Min->mincnt=0;
    return Min;
}

void minStackPush(MinStack* obj, int x) {
    obj->top++;
    obj->str[obj->top]=x;

    printf("mincnt=%d push:%d\n",obj->mincnt,x);
    if( obj->mincnt==0 || x<=obj->min[obj->mincnt-1] )
    {
        obj->min[obj->mincnt++]=x;
        printf("%d*",x);
    }
    printf("\n===end===\n\n");
}

void minStackPop(MinStack* obj) {
    if(obj->top==-1)
        return ;
```

```c
    if(obj->mincnt==0)
        ;//////////////////////

    else if( obj->str[obj->top]==obj->min[obj->mincnt-1] )
        obj->mincnt--;

    obj->top--;
}

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

int minStackGetMin(MinStack* obj) {
    return obj->min[obj->mincnt-1];
}

void minStackFree(MinStack* obj) {
    free(obj);
}

/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);

 * minStackPop(obj);

 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);

 * minStackFree(obj);
*/
```

OUTPUT :

Accepted Runtime: 3 ms

• Case 1

Input

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

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

Stdout

```
mincnt=0 push:-2
-2*
===end===

mincnt=1 push:0

===end===
```

View more

Output

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

Expected

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

Lab program 5:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

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

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

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

void insertAtEnd(struct Node** head,int value)
{

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

//function to delete the first element from the linked list
```

```c
void deleteFirst(struct Node** head)
{
    if(*head != NULL)
    {
        struct Node* temp = *head;
        *head = (*head) -> next;
        free(temp);
    }
}

//to delete a specified element
void deleteEle(struct Node** head,int value)
{
    struct Node* current  = *head;
    struct Node* prev = NULL;

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

    if(current == NULL)
    {
        printf("empty");
    }
    if(prev == NULL)
    {
        *head = current -> next;
    }
    else
    {
        prev -> next = current -> next;
    }
    free(current);
}

//to delete the last element
void deleteLast(struct Node** head)
{
    if(*head == NULL)
    {
        printf("empty");
    }
    struct Node* temp = *head;
    struct Node* prev = NULL;
    while(temp -> next != NULL)
    {
        prev = temp;
        temp = temp -> next;
```

```
    }
    if(prev == NULL)
    {
        *head = NULL;
    }
    else
    {
        prev -> next = NULL;
    }
    free(temp);
}

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

//main function
void main()
{
    struct Node* head = NULL;

    insertAtEnd(&head,1);
    insertAtEnd(&head,2);
    insertAtEnd(&head,3);
    insertAtEnd(&head,4);

    printf("initial linked list : ");
    display(head);

    deleteFirst(&head);
    printf("\nAfter deleting the first element : ");
    display(head);

    deleteEle(&head,2);
    printf("\nafter deleting the specified element : ");
    display(head);

    deleteLast(&head);
    printf("\nafter deleting the last element : ");
    display(head);

}
```

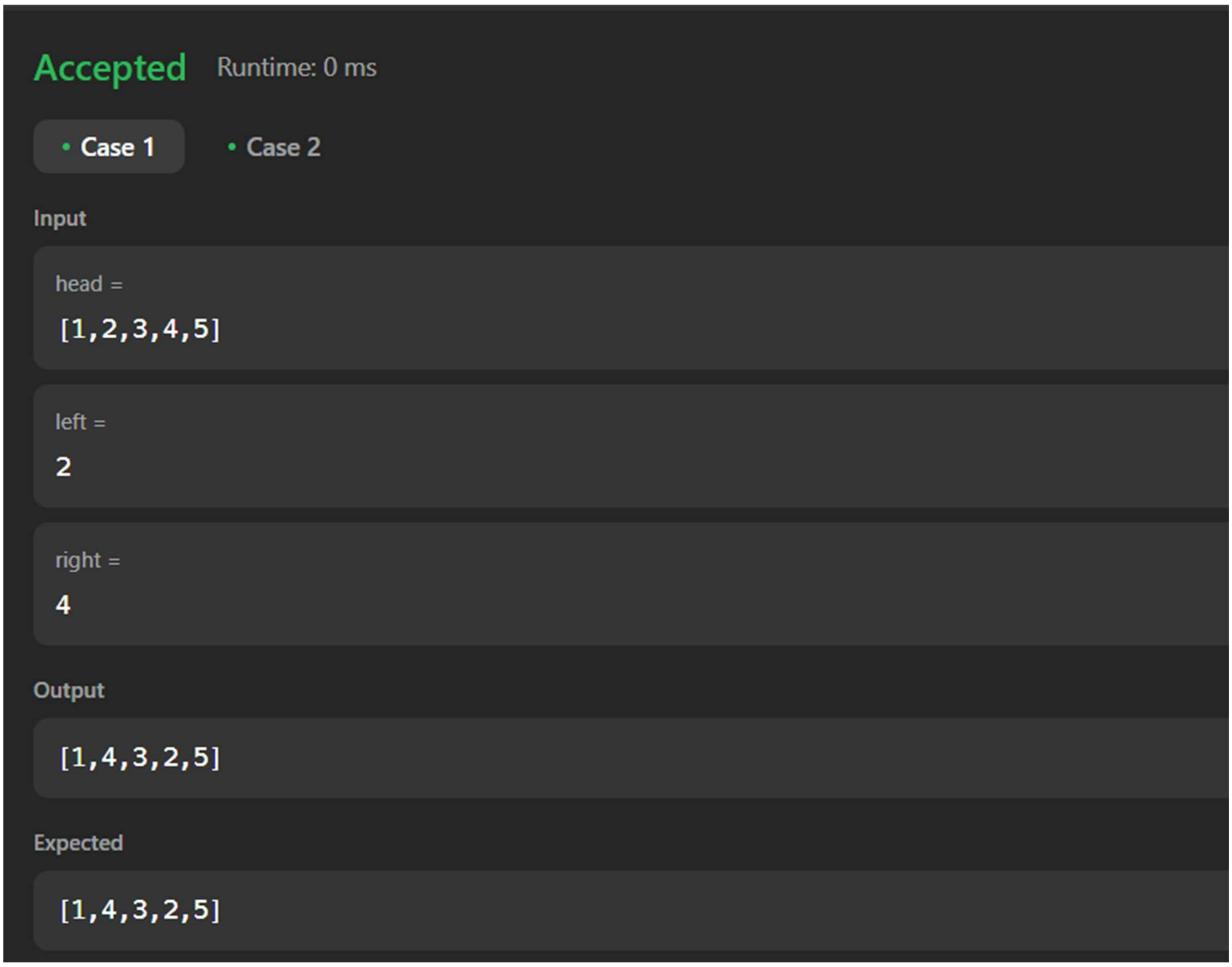
```
C:\Users\Admin\Desktop\1BM22CS215\DeleteLinkList.exe
initial linked list : 1 -> 2 -> 3 -> 4 -> NULL

After deleting the first element : 2 -> 3 -> 4 -> NULL

after deleting the specified element : 3 -> 4 -> NULL

after deleting the last element : 3 -> NULL

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Program - Leetcode platform – Reverse Linked List 2

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (head == NULL) return NULL;

    if (left == right) return head;

    struct ListNode* prev = NULL;
    struct ListNode* curr = head;

    int index = 1;
    while (index < left)
    {
        prev = curr;
        curr = curr->next;
        index++;
    }

    struct ListNode* leftMinusOneNode = prev;
```

```
    struct ListNode* leftNode = curr;
    struct ListNode* next = NULL;

    while (left <= right)
    {
        next = curr->next;

        curr->next = prev;

        prev = curr;
        curr = next;
        left++;
    }

    if (leftMinusOneNode == NULL) // means head changes
        head = prev;
    else
        leftMinusOneNode->next = prev;   // [   1   (2  <-  3   <-  4)  5   ]  //
when input -> 5 2 4
                                         //  1 is leftMinusOneNode and 4 is prev
                                         //  so 1's next is 4
    leftNode->next = curr;               //  and 2 is leftNode, so makes 2's next
is 5

    return head;
}
```

OUTPUT :


Accepted
Runtime: 0 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]

Lab program 6:

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```c
//linked list operation

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

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

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

Node *sortList(Node *head)
{
    if(head == NULL || head -> next == NULL)
        return head;
    int swapped;
    Node *temp;
    Node *end = NULL;
    do
    {
        swapped = 0;
        temp = head;
        while(temp -> next != end)
        {
```

```
            if(temp -> data > temp -> next ->data)
            {
                int tempData = temp -> data;
                temp -> data = temp -> next -> data;
                temp -> next -> data =tempData;
                swapped = -1;
            }
            temp = temp -> next;
        }
        end = temp;
    }while(swapped);
    return head;
}

Node *reverseList(Node *head)
{
    Node *prev = NULL;
    Node *current = head;
    Node *nextNode = NULL;
    while(current != NULL)
    {
        nextNode = current -> next;
        current -> next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

Node *concatLists(Node *list1 , Node *list2)
{
    if(list1 == list2)
        return list2;
    Node *temp = list1;
    while(temp -> next != NULL)
    {
        temp = temp -> next;
    }
    temp -> next = list2;
    return list1;
}

void main()
{
    Node *list1 = createNode(3);
    list1 -> next = createNode(1);
    list1 -> next -> next = createNode(4);

    Node *list2 = createNode(2);
    list2 -> next = createNode(5);
```

```
    printf("original list 1 : ");
    display(list1);
    printf("original list 2 : ");
    display(list2);

    list1 = sortList(list1);
    printf("sorted list 1 : ");
    display(list1);

    list1 = reverseList(list1);
    printf("reversed list 1 : ");
    display(list1);

    Node *concatenated = concatLists(list1,list2);
    printf("concatenated list : ");
    display(concatenated);
}
```

C:\Users\Admin\Desktop\1BM22CS215\link_list_operation.exe

```
original list 1 : 3 -> 1 -> 4 -> NULL
original list 2 : 2 -> 5 -> NULL
sorted list 1 : 1 -> 3 -> 4 -> NULL
reversed list 1 : 4 -> 3 -> 1 -> NULL
concatenated list : 4 -> 3 -> 1 -> 2 -> 5 -> NULL

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

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

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

typedef struct Node Node;

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

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

typedef struct {
    Node *top;
}LinkedList;

void push(LinkedList *stack, int value) {
    Node *newNode = createNode(value);
    newNode -> next = stack -> top;
    stack -> top = newNode;
}

int pop(LinkedList *stack) {
    if(stack -> top == NULL) {
        printf("stack is empty \n");
        return -1;
    }
```

```c
    int poppedValue = stack -> top -> data;
    Node *temp = stack -> top;
    stack -> top = stack -> top -> next;
    free(temp);
    return poppedValue;
}

void main() {
    LinkedList stack;
    stack.top = NULL;
    printf("stack operations : \n");
     push(&stack, 10);
     push(&stack, 20);
     push(&stack, 25);
     push(&stack, 30);
     display(stack.top);
    printf("popped value : %d\n", pop(&stack));
    printf("popped value : %d\n", pop(&stack));
     display(stack.top);
}
```

C:\Users\Admin\Desktop\1BM22CS215\stack_using_linked.exe

```
stack operations :
30 -> 25 -> 20 -> 10 -> NULL
popped value : 30
popped value : 25
20 -> 10 -> NULL

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

```c
//queue implementation using linked list

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

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

typedef struct Node Node;

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

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

typedef struct {
    Node *front;
    Node *rear;
}LinkedList;

void enqueue(LinkedList *queue , int value)
{
    Node *newNode = createNode(value);
    if(queue -> front ==NULL)
    {
        queue -> front = newNode;
        queue -> rear = newNode;
    }
    else
    {
        queue -> rear -> next = newNode;
```

```
        queue -> rear = newNode;
    }
}

int dequeue(LinkedList *queue)
{
    if(queue -> front == NULL)
    {
        printf("queue is empty : \n");
        return -1;
    }
    int dequeuedvalue = queue -> front -> data;
    Node *temp = queue -> front;
    queue -> front = queue -> front -> next;
    free(temp);
    return dequeuedvalue;
}

void main()
{
    LinkedList queue;
    queue.front = NULL;
    queue.rear = NULL;

    printf("\n queue operations : \n");
    enqueue(&queue,40);
    enqueue(&queue,50);
    enqueue(&queue,60);
    display(queue.front);
    printf("dequeued from queue : %d\n",dequeue(&queue));
    printf("dequeued from queue : %d\n",dequeue(&queue));
    display(queue.front);
}
```

```
C:\Users\Admin\Desktop\1BM22CS215\queue_linked_list.exe

 queue operations :
40 -> 50 -> 60 -> NULL
dequeued from queue : 40
dequeued from queue : 50
60 -> NULL

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Lab program 7:

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

```c
//doubly linked list operations

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

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

struct Node *createNode(int data)
{

    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if(newNode == NULL)
    {

        printf("memory allocation failed \n");
        exit(1);
    }
    newNode -> data = data;
```

```c
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
};

void insertNode(struct Node *head,struct Node *forget,int data)
{
    struct Node *newNode = createNode(data);
    if(forget -> prev != NULL)
    {
        forget -> prev -> next = newNode;
        newNode -> prev = forget -> prev;
    }
    else
    {
        head = newNode;
    }
    newNode -> next = forget;
    forget -> prev = newNode;
}

void deleteNode(struct Node *head,int value)
{
    struct Node *current = head;
    while(current != NULL)
    {
        if(current -> data ==value)
        {
            if(current -> prev != NULL)
            {
                current -> prev -> next = current -> next;
            }
            else
            {
                head = current -> next;
            }
            if(current -> next != NULL)
            {
                current -> next -> prev = current -> prev;
            }
            free(current);
            return;
        }
        current = current -> next;
    }
    printf("node with value %d not found \n",value);
}

void display(struct Node *head)
{
```

```c
    printf("doubly linked list : \n");
    while(head != NULL)
    {
        printf("%d <-> ",head -> data);
        head = head -> next;
    }
    printf("NULL\n");
}

void main()
{
    struct Node *head = NULL;
    head = createNode(1);
    head -> next = createNode(2);
    head -> next -> prev = head;
    head -> next -> next = createNode(3);
    head -> next -> next -> prev = head -> next;

    display(head);

    insertNode(head,head -> next,10);
    printf("after insertion : \n");
    display(head);

    deleteNode(head,2);
    printf("after deletion : \n");
    display(head);
    return 0;
}
```

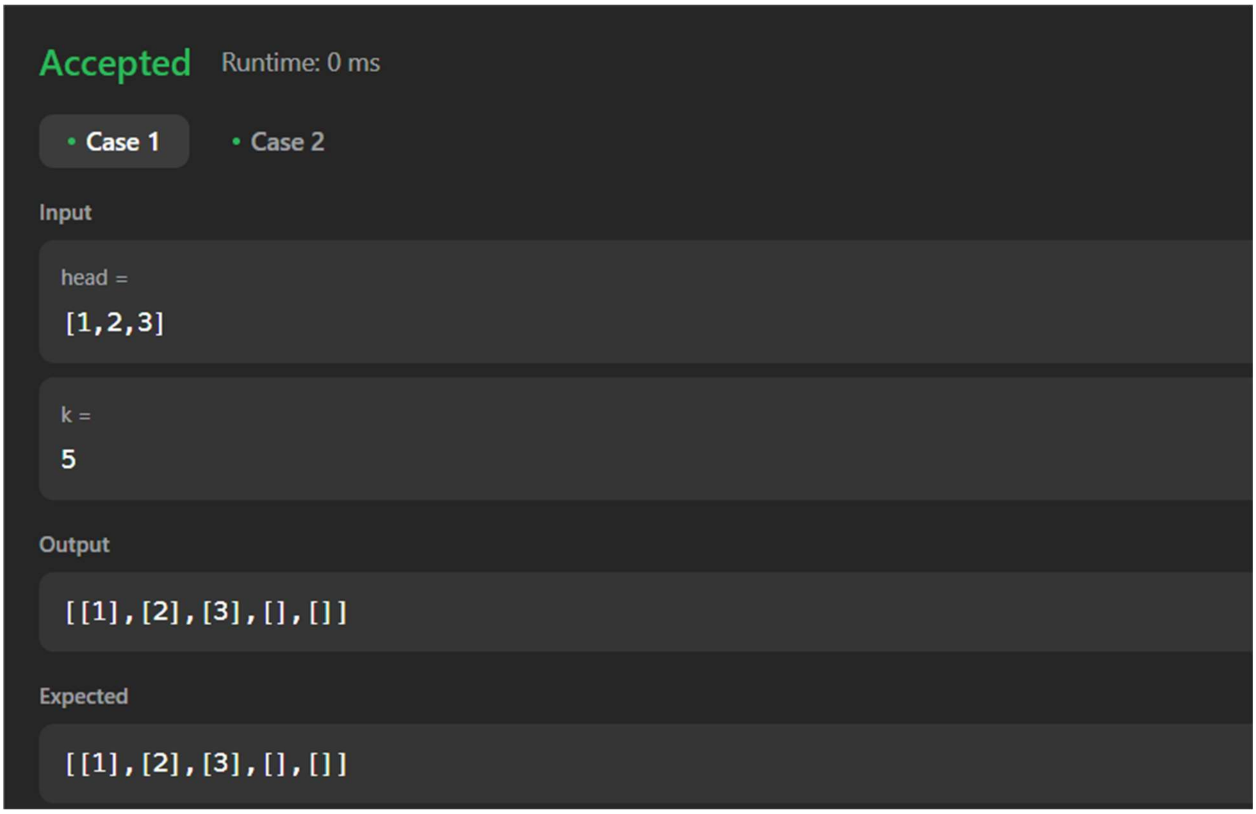
```
doubly linked list :
1 <-> 2 <-> 3 <-> NULL
after insertion :
doubly linked list :
1 <-> 10 <-> 2 <-> 3 <-> NULL
after deletion :
doubly linked list :
1 <-> 10 <-> 3 <-> NULL


...Program finished with exit code 5
Press ENTER to exit console.
```

Program - Leetcode platform – Split Linked list in parts

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
 int getLength(struct ListNode* head) {
    int length = 0;
    while (head != NULL) {
        length++;
        head = head->next;
    }
    return length;
}


struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize)
{
    int length = getLength(head);
    int partSize = length / k;
    int remainder = length % k;

    struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct
ListNode*));
    *returnSize = k;

    for (int i = 0; i < k; i++) {
        int currentPartSize = partSize + (i < remainder ? 1 : 0);

        if (currentPartSize == 0) {
            result[i] = NULL;
        } else {
            result[i] = head;
            for (int j = 0; j < currentPartSize - 1; j++) {
                head = head->next;
            }

            struct ListNode* temp = head->next;
            head->next = NULL;
            head = temp;
        }
    }
```

```
    return result;
}
```

OUTPUT :



Lab program 8:

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

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

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

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

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

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

void preorder(struct Node* temp)
{
    if(temp == NULL)
```

```
        return;
    printf("%d ",temp -> data);
    preorder(temp -> left);
    preorder(temp -> right);
}

void postorder(struct Node* temp)
{
    if(temp == NULL)
        return;
    postorder(temp -> left);
    postorder(temp -> right);
    printf("%d ",temp -> data);
}

void main()
{
    struct Node* root = NULL;
    int data,choice;
    root = insert(root,20);
    root = insert(root,10);
    root = insert(root,5);
    root = insert(root,15);
    root = insert(root,40);
    root = insert(root,30);
    root = insert(root,50);
    printf("\n the inorder traversal is : \n");
    inorder(root);
    printf("\n");
    printf("\n the preorder traversal is : \n");
    preorder(root);
    printf("\n");
    printf("\n the postorder traversal is : \n");
    postorder(root);
    printf("\n");
}
```

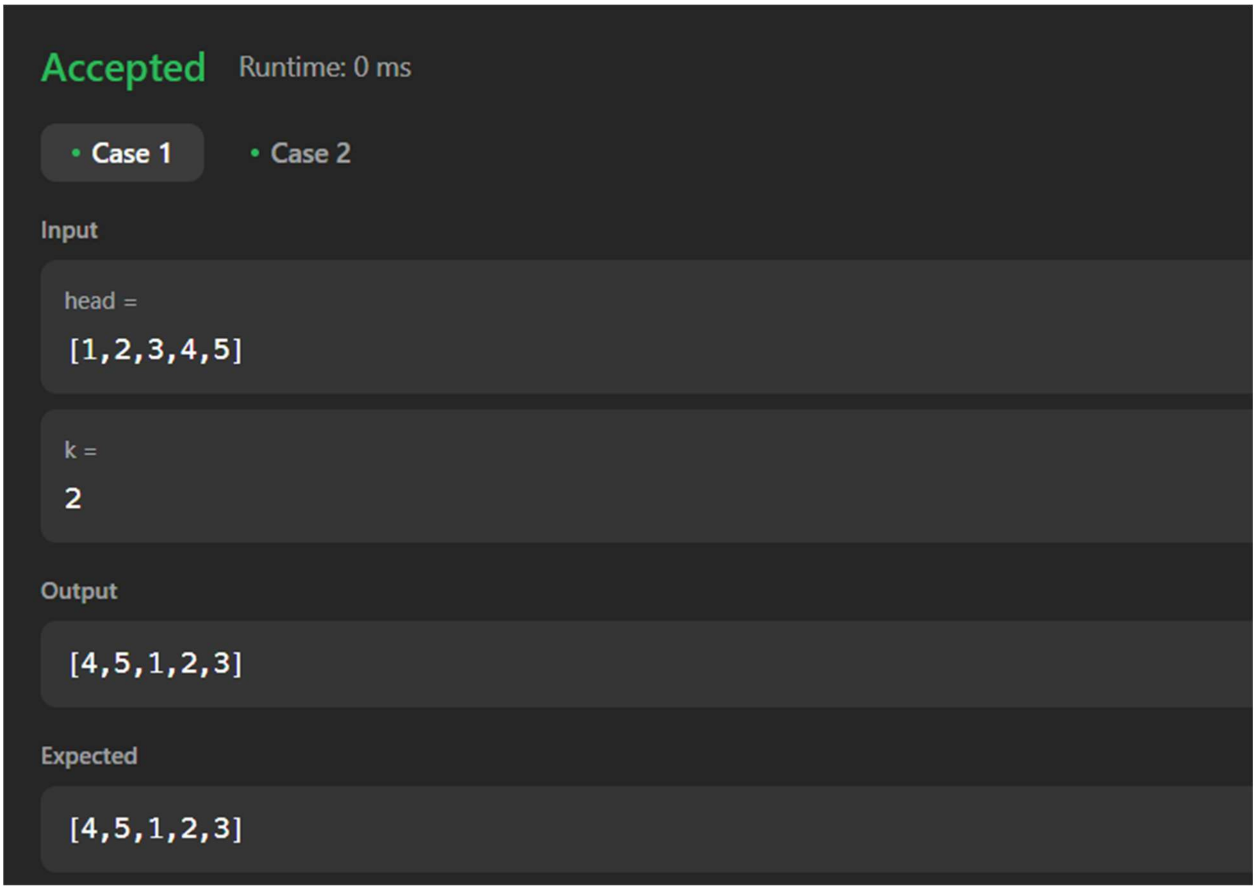
```
C:\Users\RAHUL\OneDrive\Documents\BST.exe

 the inorder traversal is :
5 10 15 20 30 40 50

 the preorder traversal is :
20 10 5 15 40 30 50

 the postorder traversal is :
5 15 10 30 50 40 20

Process returned 10 (0xA)   execution time : 0.295 s
Press any key to continue.
```

Program - Leetcode platform – Rotate List

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
 int GetLength(struct ListNode* head)
{
    if (head == NULL)
        return 0;

    return 1 + GetLength(head->next);
}
struct ListNode* rotateRight(struct ListNode* head, int k){
        if (head == NULL || k == 0)
        return head;
    int length = GetLength(head);

    if (length == 1)
        return head;
    for(int i=0;i<k%length;i++)
    {
        struct ListNode *p=head;
        while(p->next->next!=NULL)
        {
            p=p->next;
        }
```

```c
        struct ListNode *a=(struct ListNode *)malloc(sizeof(struct ListNode));
        a->val=p->next->val;
        a->next=head;
        head=a;
        p->next=NULL;
    }
    return head;

}
```

OUTPUT :



Lab program 9:

9a) Write a program to traverse a graph using BFS method.

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

#define MAX_SIZE 100

// Queue implementation
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

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

// Check if the queue is empty
int isEmpty(struct Queue* queue) {
    if (queue->rear == -1)
        return 1;
    else
        return 0;
}

// Check if the queue is full
int isFull(struct Queue* queue) {
    if (queue->rear == MAX_SIZE - 1)
        return 1;
    else
        return 0;
}

// Add an item to the queue
void enqueue(struct Queue* queue, int value) {
    if (isFull(queue))
        printf("\nQueue is Full!!");
    else {
        if (queue->front == -1)
            queue->front = 0;
        queue->rear++;
        queue->items[queue->rear] = value;
```

```c
    }
}

// Remove an item from the queue
int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty");
        item = -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

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

// Create a graph with 'vertices' number of vertices
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;

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

// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1; // Uncomment if the graph is undirected
}

// Breadth First Search traversal
void BFS(struct Graph* graph, int startVertex) {
    int visited[MAX_SIZE] = {0}; // Initialize all vertices as not visited
    struct Queue* queue = createQueue();
```

```c
    visited[startVertex] = 1;
    enqueue(queue, startVertex);

    printf("Breadth First Search Traversal: ");

    while (!isEmpty(queue)) {
        int currentVertex = dequeue(queue);
        printf("%d ", currentVertex);

        for (int i = 0; i < graph->vertices; i++) {
            if (graph->adjMatrix[currentVertex][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                enqueue(queue, i);
            }
        }
    }
    printf("\n");
}

int main() {
    int vertices, edges, src, dest;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++) {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }

    int startVertex;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    BFS(graph, startVertex);

    return 0;
}
```

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destination): 3 4
Enter edge 6 (source destination): 3 4
Enter the starting vertex for BFS: 0
Breadth First Search Traversal: 0 1 2 3 4


...Program finished with exit code 0
Press ENTER to exit console.
```

9b) Write a program to check whether given graph is connected or not using DFS method.

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

#define MAX_SIZE 100

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

// Create a graph with 'vertices' number of vertices
struct Graph* createGraph(int vertices)
{
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;

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

```c
// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest)
{
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1; // Uncomment if the graph is undirected
}

// Depth First Search traversal
void DFS(struct Graph* graph, int startVertex, int visited[])
{
    visited[startVertex] = 1;

    for (int i = 0; i < graph->vertices; i++)
    {
        if (graph->adjMatrix[startVertex][i] == 1 && visited[i] == 0)
            DFS(graph, i, visited);
    }
}

// Check if the graph is connected
int isConnected(struct Graph* graph)
{
    int* visited = (int*)malloc(graph->vertices * sizeof(int));

    for (int i = 0; i < graph->vertices; i++)
        visited[i] = 0;

    DFS(graph, 0, visited);

    for (int i = 0; i < graph->vertices; i++)
    {
        if (visited[i] == 0)
            return 0; // Graph is not connected
    }
    return 1; // Graph is connected
}

int main()
{
    int vertices, edges, src, dest;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    struct Graph* graph = createGraph(vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++)
```

```
    {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d%d", &src, &dest);
        addEdge(graph, src, dest);
    }

    if (isConnected(graph))
        printf("The graph is connected.\n");
    else
        printf("The graph is not connected.\n");

    return 0;
}
```

```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 0 1
Enter edge 3 (source destination): 1 2
Enter edge 4 (source destination): 1 2
The graph is not connected.


...Program finished with exit code 0
Press ENTER to exit console.
```

Hackerrank program

```
#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;
}
```

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 linear probing.

```
#include <stdio.h>

#define SIZE 10

int hashFunction(int key)
{
    return key%SIZE;
}
void insertValue(int hashTable[],int key)
{
    int i=0;
    int hkey = hashFunction(key);
    int index;

do
{
    index = (hkey + i)% SIZE;
    if(hashTable[index] == -1)
    {
        hashTable[index] = key;
        printf("inserted key %d at index %d\n",key,index);
        return;
    }
    i++;
}while(i<SIZE);
printf("unable to insert key %d Hash table is full\n",key);
}

int searchValue(int hashTable[],int key)
{
    int i = 0;
    int hkey = hashFunction(key);
    int index;
    do
```

```
    {
        index = (hkey + i)%SIZE;
        if(hashTable[index] == key)
        {
            printf("key %d found at index %d\n",key,index);
            return index;
        }
        i++;
    }while(i < SIZE);
    printf("key %d not found in hash table\n",key);
    return -1;
}

void main ()
{
    int hashTable[SIZE];
    for(int i = 0;i<SIZE;i++)
    {
        hashTable[i] = -1;
    }
    insertValue(hashTable,1234);
    insertValue(hashTable,5678);
    insertValue(hashTable,9876);
    searchValue(hashTable,5678);
    searchValue(hashTable,1111);
    return 0;
}
```

```
inserted key 1234 at index 4
inserted key 5678 at index 8
inserted key 9876 at index 6
key 5678 found at index 8
key 1111 not found in hash table


...Program finished with exit code 0
Press ENTER to exit console.
```