VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C

Submitted by

SAMRAAT DABOLAY (1BM22CS236)

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

December-2023 to April-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "**Data Structures using C**" carried out by **SAMRAAT DABOLAY (1BM22CS236),** 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 academic semester December-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.

Name of the Lab-In charge:	Dr. Jyothi S Nayak
(Designation)	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

<table>
<tr><td></td><td>5b) Program - Leetcode platform</td><td></td></tr>
<tr><td>6</td><td>6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.</td><td>43-56</td></tr>
<tr><td>7</td><td>7a) 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
Display the contents of the list

7b) Program - Leetcode platform</td><td>57-68</td></tr>
<tr><td>8</td><td>8a) 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
To display the elements in the tree.

8b) Program - Leetcode platform</td><td>69-76</td></tr>
<tr><td>9</td><td>9a) Write a program to traverse a graph using BFS method.
9b) Write a program to check whether given graph is connected or not using DFS method.</td><td>77-83</td></tr>
<tr><td>10</td><td>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.</td><td>-</td></tr>
</table>

Course Outcome

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyse data structure operations for a given problem.
CO3	CO3 Design and implement operations of linear and nonlinear data structure.
CO4	Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques.

LAB 1

Question

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

Code

```
#include <stdio.h>
#include <stdlib.h>
#define n 4

int stack[n];
int top;

void push()
{
   if(top>=n)
   {
      printf("Stack is full! Overflow error!\n");
   }
   else
   {
      int a;
      printf("Enter element to be inserted: ");
      scanf("%d", &a);
```

```
        top++;
        stack[top] = a;
        printf("Element inserted!\n");
    }
}

void pop()
{
    if(top==-1)
    {
        printf("Stack is empty! Underflow error!\n");
    }
    else
    {
        printf("Element deleted is: %d\n", stack[top]);
        top--;
    }
}

void display()
{
    int i;
    if(top==-1)
    {
        printf("Stack is empty!\n");
```

```
    }
    else
    {
        printf("Elements are: ");
        for(i=n;i>=0;i--)
        {
            printf("%d\n", stack[i]);
        }
    }
}

int main()
{
    int ch;
    top = -1;
    printf("Menu:\n1. Push element\n2. Pop Element\n3. Display Stack\n4.
Exit\n");
    printf("Enter choice: ");
    scanf("%d", &ch);
    while(1)
    {
        switch(ch)
        {
            case 1:
                push();
```

```
            break;
          case 2:
            pop();
            break;
          case 3:
            display();
            break;
          case 4:
            printf("Exiting!");
            exit(0);
          default:
            printf("Please enter valid choice!\n");
        }
        printf("Enter choice: ");
        scanf("%d", &ch);
    }
    return 0;
}
```

Output

```
Menu:
1. Push element
2. Pop Element
3. Display Stack
4. Exit
Enter choice: 1
Enter element to be inserted: 1
Element inserted!
Enter choice: 1
Enter element to be inserted: 2
Element inserted!
Enter choice: 1
Enter element to be inserted: 3
Element inserted!
Enter choice: 1
Enter element to be inserted: 4
Element inserted!
Enter choice: 1
Enter element to be inserted: 5
Element inserted!
Enter choice: 1
Stack is full! Overflow error!
Enter choice: 3
Elements are: 5
4
3
2
1
```

```
Enter choice: 2
Element deleted is: 0
Enter choice: 2
Element deleted is: 4
Enter choice: 2
Element deleted is: 4
Enter choice: 2
Element deleted is: 3
Enter choice: 2
Element deleted is: 2
Enter choice: 2
Element deleted is: 1
Enter choice: 2
Stack is empty! Underflow error!
Enter choice: 3
Stack is empty!
Enter choice: 4
Exiting!
```

LAB 2

Question

a) 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)
b) Demonstration of account creation on LeetCode platform Program - Leetcode platform

Code

```
#include <stdio.h>

#include <ctype.h>

#include <string.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 getpri(char);


int main()

{

   char infix[100], postfix[100];
```

```
    printf("\n Enter any infix expression : ");
    scanf("%s", infix);
    strcpy(postfix, "");
    InfixtoPostfix(infix, postfix);
    printf("\n The corresponding postfix expression is : ");
    puts(postfix);
}

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("\n INCORRECT EXPRESSION");
            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] == '/' ||
source[i] == '%' || source[i] == '^')
    {
        while ((top != -1) && (st[top] != '(') && (getpri(st[top]) > getpri(source[i])))
        {
            target[j] = pop(st);
            j++;
        }
        push(st, source[i]);
```

```
            i++;
        }
        else
        {
            printf("\n INCORRECT ELEMENT IN EXPRESSION");
            exit(1);
        }
    }
    while ((top != -1) && (st[top] != '('))
    {
        target[j] = pop(st);
        j++;
    }
    target[j] = '\0';
}

int getpri(char op)
{
    if (op == '^')
        return 2;
    else if (op == '/' || op == '*' || op == '%')
        return 1;
    else if (op == '+' || op == '-')
        return 0;
}
```

```
void push(char st[], char val)
{
   if (top == MAX - 1)
      printf("\n STACK OVERFLOW");
   else
   {
      top++;
      st[top] = val;
   }
}

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

Output

```
Enter any infix expression : a*b+(e/f^g)

The corresponding postfix expression is : ab*efg^/+
```

```
Enter any infix expression : a*b*c-(d+e/f*(g+h))

The corresponding postfix expression is : abc**defgh+*/+-
```

LAB 3

Question

a) 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

b) 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

Code (a)

```
#include <stdio.h>

#include <stdlib.h>

#define N 5


int q[N];

int front = -1, rear = -1;


void insert(int);

int deleteq();

void display();


int main()

{

   int n, choice;
```

```
printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");

do
{
   printf("\nEnter your option : \n");
   scanf("%d", &choice);
   switch (choice)
   {
      case 1:
         printf("Enter the number to be inserted in the queue : \n");
         scanf("%d", &n);
         insert(n);
         break;
      case 2:
         n = deleteq();
         if (n != -1)
            printf("\n The number deleted is : %d\n", n);
         break;
      case 3:
         display();
         break;
      case 4:
         exit(0);
         break;
      default:
```

```
            printf("Invalid option\n");
            exit(0);
            break;
        }
    } while (choice != 4);
}

void insert(int num)
{
    if (rear == N - 1)
        printf("\n OVERFLOW");
    else if (front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    q[rear] = num;
}

int deleteq()
{
    int val;
    if (front == -1 || front > rear)
    {
        printf("\n UNDERFLOW");
        return -1;
```

```
    }
    else
    {
        val = q[front];
        front++;
        if (front > rear)
            front = rear = -1;
        return val;
    }
}

void display()
{
    int i;
    printf("\n");
    if (front == -1 || front > rear)
        printf("\n QUEUE IS EMPTY");
    else
    {
        for (i = front; i <= rear; i++)
            printf("\t %d", q[i]);
    }
}
```

Output (a)

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

Enter your option :
1
Enter the number to be inserted in the queue :
1

Enter your option :
1
Enter the number to be inserted in the queue :
2

Enter your option :
1
Enter the number to be inserted in the queue :
3

Enter your option :
1
Enter the number to be inserted in the queue :
4

Enter your option :
1
Enter the number to be inserted in the queue :
5

Enter your option :
1
Enter the number to be inserted in the queue :
6

 OVERFLOW
```

```
Enter your option :
3

         1       2       3       4       6
Enter your option :
2

 The number deleted is : 1

Enter your option :
2

 The number deleted is : 2

Enter your option :
2

 The number deleted is : 3

Enter your option :
3

         4       6
Enter your option :
2

 The number deleted is : 4

Enter your option :
2

 The number deleted is : 6

Enter your option :
2

 UNDERFLOW
```

Code (b)

```
#include <stdio.h>

#include <stdlib.h>

#define N 5


int q[N];
```

```
int front = -1, rear = -1;


void insert(int);

int deleteq();

void display();


int main()

{

   int n, choice;

   printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");

   do

   {

      printf("\nEnter your option : \n");

      scanf("%d", &choice);

      switch (choice)

      {

         case 1:

            printf("Enter the number to be inserted in the queue : \n");

            scanf("%d", &n);

            insert(n);

            break;

         case 2:

            n = deleteq();

            if (n != -1)

               printf("\n The number deleted is : %d\n", n);
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("Invalid option\n");
            exit(0);
            break;
      }
   } while (choice != 4);
}

void insert(int num)
{
   if ((front == 0 && rear == N - 1) || (rear == (front - 1)))
      printf("\n OVERFLOW");
   else if (front == -1 && rear == -1)
   {
      front = rear = 0;
      q[rear] = num;
   }
   else if (rear == N - 1 && front != 0)
```

```
    {
        rear = 0;
        q[rear] = num;
    }
    else
    {
        rear++;
        q[rear] = num;
    }
}

int deleteq()
{
    int val;
    if (front == -1 && rear == -1)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    val = q[front];
    if (front == rear)
        front = rear = -1;
    else
    {
        if (front == N - 1)
```

```
        front = 0;
      else
        front++;
    }
    return val;
}

void display()
{
    int i;
    printf("\n");
    if (front == -1 && rear == -1)
      printf("\n QUEUE IS EMPTY");
    else
    {
      if (front < rear)
      {
        for (i = front; i <= rear; i++)
          printf("\t %d", q[i]);
      }
      else
      {
        for (i = front; i < N; i++)
          printf("\t %d", q[i]);
        for (i = 0; i <= rear; i++)
```

```
            printf("\t %d", q[i]);
        }
    }
}
```

Output (b)

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

Enter your option :
1
Enter the number to be inserted in the queue :
1

Enter your option :

1
Enter the number to be inserted in the queue :
2

Enter your option :
1
Enter the number to be inserted in the queue :
3

Enter your option :
1
Enter the number to be inserted in the queue :
4

Enter your option :
1
Enter the number to be inserted in the queue :
5

Enter your option :
1
Enter the number to be inserted in the queue :
6

 OVERFLOW
```

```
Enter your option :
3

         1       2       3       4       5
Enter your option :
2

 The number deleted is : 1

Enter your option :
2

 The number deleted is : 2

Enter your option :
2

 The number deleted is : 3

Enter your option :
3

         4       5
Enter your option :
2

 The number deleted is : 4

Enter your option :
2

 The number deleted is : 5

Enter your option :
2

 UNDERFLOW
```

LAB 4

Question

a) WAP to Implement Singly Linked List with following operations

Create a linked list.

Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

b) Program - Leetcode platform

Code

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

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

void InsertAtBeginning(Node **head_ref, int new_data);
void InsertAtEnd(Node **head_ref, int new_data);
void Insert(Node **head_ref, int new_data, int pos);
void PrintList(Node *next);

void InsertAtBeginning(Node **head_ref, int new_data)
```

```
{
   Node *new_node = (struct Node *)malloc(sizeof(Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   *head_ref = new_node;
}

void InsertAtEnd(Node **head_ref, int new_data)
{
   Node *new_node = (struct Node *)malloc(sizeof(Node));
   Node *last = *head_ref;
   new_node->data = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL)
   {
      *head_ref = new_node;
      return;
   }
   while (last->next != NULL)
      last = last->next;
   last->next = new_node;
}

void Insert(Node **head_ref, int new_data, int pos)
{
```

```
    if (*head_ref == NULL)
    {
        printf("Cannot be NULL\n");
        return;
    }
    Node *temp = *head_ref;
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = new_data;
    newNode->next = NULL;
    while (--pos > 0)
    {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

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

```
int main()
{
   int ch, new, pos;
   Node *head = NULL;
   while (ch != 5)
   {
      printf("Menu\n");
      printf("1.Insert at beginning\n");
      printf("2.Insert at a specific position\n");
      printf("3.Insert at end\n");
      printf("4.Display linked list\n");
      printf("5.Exit\n");

      printf("Enter your choice\n");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
      {
         printf("Enter the data you want to insert at beginning\n");
         scanf("%d", &new);
         InsertAtBeginning(&head, new);
         break;
      }
```

```
case 2:
{
   printf("Enter the data and position at which you want to insert \n");
   scanf("%d%d", &new, &pos);
   Insert(&head, new, pos);
   break;
}
case 3:
{
   printf("Enter the data you want to insert at end\n");
   scanf("%d", &new);
   InsertAtEnd(&head, new);
   break;
}
case 4:
{
   printf("Created linked list is:\n");
   PrintList(head);
   break;
}
case 5:
{
   return 0;
   break;
}
```

```
        case 6:
        {
           printf("Invalid data!");
           break;
        }
        }
    }
    return 0;
}
```

Output

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
1
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
1
Enter the data you want to insert at beginning
2
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
3
Enter the data you want to insert at end
6
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
```

```
Enter the data and position at which you want to insert
3
3
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
2
1
6
3
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to insert
4
4
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
2
1
6
3
```

```
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to insert
4
4
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
2
1
6
3
4
Menu
1.Insert at beginning
2.Insert at a specific position
3.Insert at end
4.Display linked list
5.Exit
Enter your choice
5
```

Leetcode - Valid Parentheses

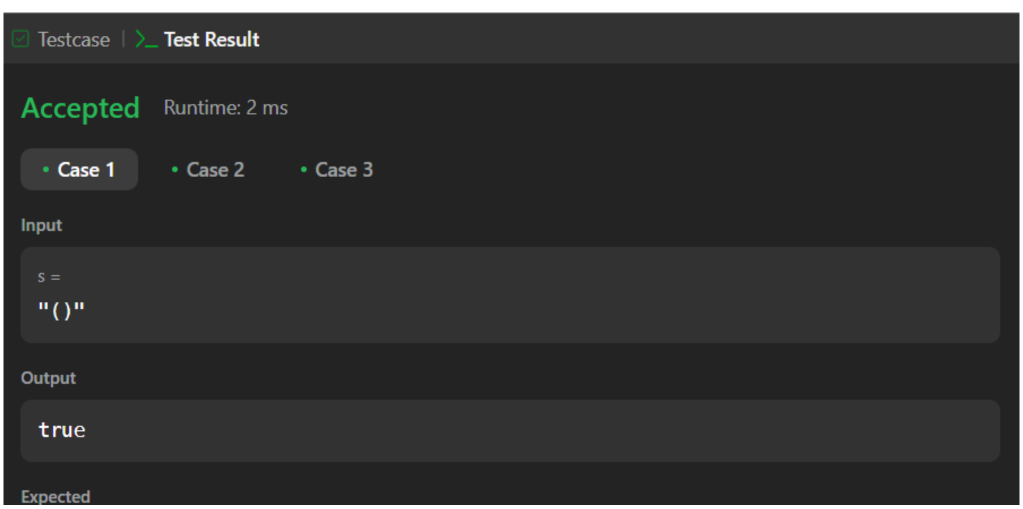
```
struct sNode {
   char data;
   struct sNode* next;
};

// Function to push an item to stack
void push(struct sNode** top_ref, int new_data);

// Function to pop an item from stack
int pop(struct sNode** top_ref);

// Returns 1 if character1 and character2 are matching left
```

```
// and right Brackets
bool isMatchingPair(char character1, char character2)
{
   if (character1 == '(' && character2 == ')')
      return 1;
   else if (character1 == '{' && character2 == '}')
      return 1;
   else if (character1 == '[' && character2 == ']')
      return 1;
   else
      return 0;
}

bool isValid(char* exp)
{
   int i = 0;

   struct sNode* stack = NULL;

   while (exp[i]) {

      if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
         push(&stack, exp[i]);

      if (exp[i] == '}' || exp[i] == ')'
```

```
        || exp[i] == ']') {

        if (stack == NULL)
          return 0;

        // Pop the top element from stack, if it is not
        // a pair bracket of character then there is a
        // mismatch.
        // his happens for expressions like {(})
        else if (!isMatchingPair(pop(&stack), exp[i]))
          return 0;
      }
      i++;
    }

    // If there is something left in expression then there
    // is a starting bracket without a closing
    // bracket
    if (stack == NULL)
      return 1; // balanced
    else
      return 0; // not balanced
}

// Function to push an item to stack
```

```
void push(struct sNode** top_ref, int new_data)
{
   // allocate node
   struct sNode* new_node
      = (struct sNode*)malloc(sizeof(struct sNode));

   if (new_node == NULL) {
      printf("Stack overflow n");
      getchar();
      exit(0);
   }

   // put in the data
   new_node->data = new_data;

   // link the old list of the new node
   new_node->next = (*top_ref);

   // move the head to point to the new node
   (*top_ref) = new_node;
}

// Function to pop an item from stack
int pop(struct sNode** top_ref)
{
```

```
    char res;
    struct sNode* top;

    // If stack is empty then error
    if (*top_ref == NULL) {
        printf("Stack overflow n");
        getchar();
        exit(0);
    }
    else {
        top = *top_ref;
        res = top->data;
        *top_ref = top->next;
        free(top);
        return res;
    }
}
```



LAB 5

Question

a) WAP to Implement Singly Linked List with following operations
   i) Create a linked list.
   ii) Deletion of the first element, specified element and last element in the list.
   iii) Display the contents of the linked list.

b) Program - Leetcode platform

Code

```
#include <stdio.h>

#include <stdlib.h>


typedef struct Node

{

   int data;

   struct Node *next;

} Node;


void InsertAtBeginning(Node **head_ref, int new_data);

void DeleteAtBeginning(Node **head_ref);

void DeleteAtEnd(Node **head_ref);

void Delete(Node **head_ref, int pos);

void PrintList(Node *next);


void InsertAtBeginning(Node **head_ref, int new_data)
```

```
{
   Node *new_node = (struct Node *)malloc(sizeof(Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   *head_ref = new_node;
}

void DeleteAtBeginning(Node **head_ref)
{
   Node *ptr;
   if (*head_ref == NULL)
   {
      printf("\nList is empty");
   }
   else
   {
      ptr = *head_ref;
      *head_ref = ptr->next;
      free(ptr);
      printf("\n Node deleted from the beginning ...");
   }
}

void DeleteAtEnd(Node **head_ref)
{
```

```
    Node *ptr, *ptr1;
    if (*head_ref == NULL)
    {
        printf("\nlist is empty");
    }
    else if ((*head_ref)->next == NULL)
    {
        free(*head_ref);
        *head_ref = NULL;
        printf("\nOnly node of the list deleted ...");
    }
    else
    {
        ptr = *head_ref;
        while (ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\n Deleted Node from the last ...");
    }
}
```

```
void Delete(Node **head_ref, int pos)
{
   Node *temp = *head_ref, *prev;

   if (temp == NULL)
   {
      printf("\nList is empty");
      return;
   }

   if (pos == 1)
   {
      *head_ref = temp->next;
      free(temp);
      printf("\nDeleted node with position %d", pos);
      return;
   }

   for (int i = 0; temp != NULL && i < pos - 1; i++)
   {
      prev = temp;
      temp = temp->next;
   }

   if (temp == NULL)
```

```
    {
        printf("\nPosition out of range");
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("\nDeleted node with position %d", pos);
}

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

int main()
{
    int ch, new, pos;
    Node *head = NULL;
    while (ch != 6)
    {
```

```
printf("\nMenu\n");
printf("1.Create a linked list\n");
printf("2.Delete at beginning\n");
printf("3.Delete at a specific position\n");
printf("4.Delete at end\n");
printf("5.Display linked list\n");
printf("6.Exit\n");
printf("Enter your choice\n");
scanf("%d", &ch);
switch (ch)
{
case 1:
{
   printf("Enter the data you want to insert at beginning\n");
   scanf("%d", &new);
   InsertAtBeginning(&head, new);
   break;
}
case 2:
{
   DeleteAtBeginning(&head);
   break;
}
case 3:
{
```

```
    printf("Enter the position at which you want to delete \n");
    scanf("%d", &pos);
    Delete(&head, pos);
    break;
}
case 4:
{

    DeleteAtEnd(&head);
    break;
}
case 5:
{
    printf("Created linked list is:\n");
    PrintList(head);
    break;
}
case 6:
{
    return 0;
    break;
}
default:
{
    printf("Invalid data!");
```

```
            break;
        }
        }
    }
    return 0;
}
```

Output

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
1
Enter the data you want to insert at beginning
1

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
1
Enter the data you want to insert at beginning
2

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
1
Enter the data you want to insert at beginning
3
```

```
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
1
Enter the data you want to insert at beginning
4

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
5
Created linked list is:
4
3
2
1

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
3
Enter the position at which you want to delete
2

Deleted node with position 2
```

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
5
Created linked list is:
4
2
1

Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
2

 Node deleted from the beginning ...
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
4

 Deleted Node from the last ...
```

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
2

 Node deleted from the beginning ...
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
2

 Node deleted from the beginning ...
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4.Delete at end
5.Display linked list
6.Exit
Enter your choice
2

List is empty
```

Leetcode - Reversing a Linked List

</> Code

C Auto

```c
struct ListNode* reverseList(struct ListNode* head) {
    struct ListNode* temp = head;
    struct ListNode* curr = temp;
    struct ListNode* prev = NULL;
    struct ListNode* nextOne = NULL;

    while(curr != NULL) {
        nextOne = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextOne;
    }
    return prev;
}
```

Saved to local

Ln 17, Col 2

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =

[1,2,3,4,5]

Output

[5,4,3,2,1]

Expected

LAB 6

Question

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

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

Code (a)

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

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

void sortList(struct Node** node);
void create(struct Node** node);
void display(struct Node* node);
void insert(struct Node** node, int value);
void reverse(struct Node** node);
void concat(struct Node** node1, struct Node** node2);

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

```
    struct Node* head2 = NULL;

    printf("Create LL 1 : \n");
    create(&head1);
    printf("Create LL 2 : \n");
    create(&head2);

    printf("Concatenation of two lists is : \n");
    concat(&head1, &head2);
    display(head1);

    printf("Sorting of this list : \n");
    sortList(&head1);
    display(head1);

    printf("Reversing of this list : \n");
    reverse(&head1);
    display(head1);

    // Free memory
    struct Node* temp;
    while (head1 != NULL) {
        temp = head1;
        head1 = head1->next;
        free(temp);
```

```
    }
    while (head2 != NULL) {
        temp = head2;
        head2 = head2->next;
        free(temp);
    }

    return 0;
}

void create(struct Node** node) {
    int ch, val;
    while (1) {
        printf("1. Insert\n2. Exit\n");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Enter the value : ");
                scanf("%d", &val);
                insert(node, val);
                break;
            case 2:
                return;
            default:
```

```
            printf("Invalid choice\n");
        }
    }
}

void insert(struct Node** node, int value) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->val = value;
    new_node->next = *node;
    *node = new_node;
}

void sortList(struct Node** node) {
    struct Node *temp, *i;
    for (temp = *node; temp != NULL; temp = temp->next) {
        for (i = *node; i != NULL; i = i->next) {
            if (i->val > temp->val) {
                int tem = i->val;
                i->val = temp->val;
                temp->val = tem;
            }
        }
    }
}
```

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

void reverse(struct Node** node) {
    struct Node* curr = *node;
    struct Node* prev = NULL;
    struct Node* nextOne = NULL;

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

    *node = prev;
}

void concat(struct Node** node1, struct Node** node2) {
    struct Node* temp1 = *node1;
```

```
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = *node2;
    *node2 = NULL;
}
```

Output (a)

```
Create LL 1 :
1. Insert
2. Exit
1
Enter the value : 1
1. Insert
2. Exit
1
Enter the value : 2
1. Insert
2. Exit
1
Enter the value : 3
1. Insert
2. Exit
2
Create LL 2 :
1. Insert
2. Exit
1
Enter the value : 4
1. Insert
2. Exit
1
Enter the value : 8
1. Insert
2. Exit
1
Enter the value : 2
1. Insert
2. Exit
2
Concatenation of two lists is :
3->2->1->2->8->4->NULL
Sorting of this list :
1->2->2->3->4->8->NULL
Reversing of this list :
8->4->3->2->2->1->NULL
```

Code (b)

Stacks:

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

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

typedef Node* Stack;

Stack head = NULL;

void push(int val) {
   Node *newNode = malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = val;
   newNode->next = head;
   head = newNode;
}
```

```
void pop() {
    if (head == NULL) {
        printf("Stack is Empty\n");
    } else {
        printf("Popped element = %d\n", head->data);
        Node *temp = head;
        head = head->next;
        free(temp);
    }
}

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

void freeStack() {
    Node *temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
```

```
        free(temp);
    }
}

int main() {
    int data, ch;
    printf("Menu:\n 1. Push\n 2. Pop\n 3. Display\n 4. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &ch);
    while (ch != 4) {
        switch (ch) {
            case 1:
                printf("Enter data to be pushed: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                printList();
                break;
            default:
                printf("Invalid choice\n");
        }
```

```
        printf("\nEnter choice: ");
        scanf("%d", &ch);
    }
    freeStack();
    return 0;
}
```

Queues:

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

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

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

void enqueue(int val) {
    struct node *newNode = malloc(sizeof(struct node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = val;
```

```
    newNode->next = NULL;

    if (front == NULL && rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is Empty. Unable to perform dequeue\n");
    } else {
        struct node *temp = front;
        front = front->next;
        free(temp);

        if (front == NULL) {
            rear = NULL;
        }
    }
}

void printList() {
```

```
    struct node *temp = front;
    while (temp) {
        printf("%d->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int data, ch;
    printf("Menu:\n 1. Enqueue\n 2. Dequeue\n 3. Display\n 4. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &ch);
    while (ch != 4) {
        switch (ch) {
            case 1:
                printf("Enter data to be enqueued: ");
                scanf("%d", &data);
                enqueue(data);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                printList();
```

```
            break;
        default:
            printf("Invalid choice\n");
        }
        printf("\nEnter choice: ");
        scanf("%d", &ch);
    }

    return 0;
}
```

Output (b)

Stacks:

```
Menu:
 1. Push
 2. Pop
 3. Display
 4. Exit
Enter choice: 1
Enter data to be pushed: 1

Enter choice: 1
Enter data to be pushed: 2

Enter choice: 3
2->1->NULL

Enter choice: 2
Popped element = 2

Enter choice: 2
Popped element = 1

Enter choice: 2
Stack is Empty

Enter choice: 3
NULL

Enter choice: 4
```

Queues:

```
Menu:
 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter choice: 1
Enter data to be enqueued: 1

Enter choice: 1
Enter data to be enqueued: 2

Enter choice: 3
1->2->NULL

Enter choice: 2

Enter choice: 3
2->NULL

Enter choice: 2

Enter choice: 2
Queue is Empty. Unable to perform dequeue

Enter choice: 3
NULL

Enter choice: 4
```

LAB 7

Question

a) WAP to Implement doubly link list with primitive operations
   i) Create a doubly linked list.
   ii) Insert a new node to the left of the node.
   iii) Delete the node based on a specific value
   iv) Display the contents of the list

b) Program - Leetcode platform

Code

```
#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");

      return NULL;

   }
```

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);

    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}

void insertBeforeNode(struct Node** head, int key, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* newNode = createNode(data);
```

```
    struct Node* current = *head;

    while (current) {
        if (current->data == key) {
            if (current->prev) {
                current->prev->next = newNode;
                newNode->prev = current->prev;
            } else {
                *head = newNode;
            }

            newNode->next = current;
            current->prev = newNode;
            return;
        }
        current = current->next;
    }

    printf("Key not found in the list\n");
}

void deleteNode(struct Node** head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
```

```
}

struct Node* current = *head;
int count = 1;

while (current && count < pos) {
   current = current->next;
   count++;
}

if (current == NULL) {
   printf("Position %d is beyond the length of the list\n", pos);
   return;
}

if (current->prev) {
   current->prev->next = current->next;
} else {
   *head = current->next;
}

if (current->next) {
   current->next->prev = current->prev;
}
```

```
    free(current);
    printf("Node at position %d deleted\n", pos);
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = head;

    while (current) {
        printf("%d-> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

void freeList(struct Node** head) {
    struct Node* current = *head;
    struct Node* nextNode;

    while (current) {
        nextNode = current->next;
```

```
        free(current);
        current = nextNode;
    }

    *head = NULL; // Set head to NULL after freeing all nodes
}

int main() {
    struct Node* head = NULL;
    int ch, newData, pos, key;

    while (1) {
        printf("\nMenu\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert before a node\n");
        printf("3. Delete a node\n");
        printf("4. Display list\n");
        printf("5. Free doubly linked list and exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &newData);
```

```
    insertAtBeginning(&head, newData);
    break;

  case 2:
    printf("Enter the value before which you want to insert: ");
    scanf("%d", &key);
    printf("Enter data to insert: ");
    scanf("%d", &newData);
    insertBeforeNode(&head, key, newData);
    break;

  case 3:
    printf("Enter the position you wish to delete: ");
    scanf("%d", &pos);
    deleteNode(&head, pos);
    break;

  case 4:
    printf("Doubly linked list: ");
    displayList(head);
    break;

  case 5:
    freeList(&head);
    printf("Exiting the program\n");
```

```
            return 0;

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

   return 0;
}
```

Output

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 3
Enter data to insert: 1
List is empty

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 4

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 3

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 5
Enter data to insert: 1
Key not found in the list
```

```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 3
Enter data to insert: 5

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> 4-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 6

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 1
Node at position 1 deleted
```

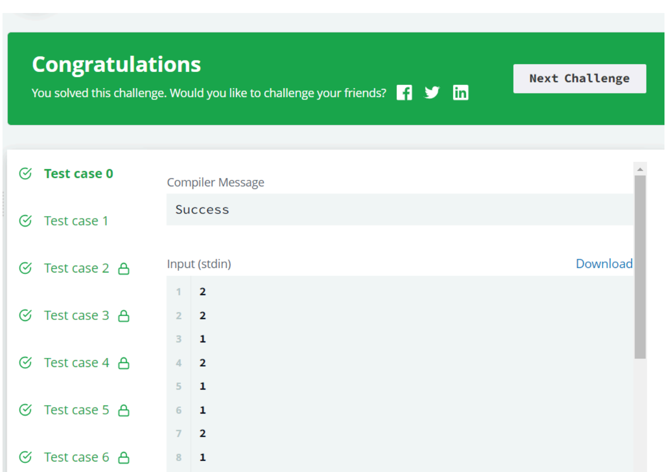
```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> 4-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 3
Node at position 3 deleted

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 5-> 3-> NULL
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 5
Exiting the program
```

Leetcode - Compare Linked Lists

```
#include <bits/stdc++.h>
// Complete the compare_lists function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */
bool compare_lists(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {
 struct SinglyLinkedListNode *t1;
   struct SinglyLinkedListNode *t2;
   t1=head1;t2 = head2;
   if(t1==NULL && t2==NULL)
      return 1;
   if(t1 != NULL && t2 == NULL)
      return 0;
    if(t1 == NULL && t2 != NULL)
      return 0;
   else
   {
      while(t1->next != NULL && t2->next != NULL)
      {
         if(t1->data == t2->data)
         {
            t1 = t1->next;
            t2 = t2->next;
         }
```

```
            else return 0;
        }
        if(t1->next == NULL && t2->next == NULL)
            return 1;
        else return 0;
    }

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    int tests;
    cin >> tests;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int tests_itr = 0; tests_itr < tests; tests_itr++) {
        SinglyLinkedList* llist1 = new SinglyLinkedList();

        int llist1_count;
        cin >> llist1_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist1_count; i++) {
            int llist1_item;
            cin >> llist1_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist1->insert_node(llist1_item);
        }

          SinglyLinkedList* llist2 = new SinglyLinkedList();
```

```
        int llist2_count;
        cin >> llist2_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist2_count; i++) {
            int llist2_item;
            cin >> llist2_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist2->insert_node(llist2_item);
        }

        bool result = compare_lists(llist1->head, llist2->head);

        fout << result << "\n";
    }

    fout.close();

    return 0;
}
```



LAB 8

Question

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

b) Program - Leetcode platform

Code

```
#include <stdio.h>

#include <stdlib.h>


struct node {

  int key;

  struct node *left, *right;

};


// Create a node

struct node *newNode(int item) {

  struct node *temp = (struct node *)malloc(sizeof(struct node));

  if (temp == NULL) {

    printf("Memory allocation failed\n");

    exit(1);

  }
```

```
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Inorder Traversal
void inorder(struct node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d -> ", root->key);
        inorder(root->right);
    }
}

// Preorder Traversal
void preorder(struct node *root) {
    if (root != NULL) {
        printf("%d -> ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}

// Postorder Traversal
void postorder(struct node *root) {
```

```
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d -> ", root->key);
    }
}

// Insert a node
struct node *insert(struct node *node, int key) {
    if (node == NULL)
        return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}

// Free the memory allocated for the tree
void freeTree(struct node *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
```

```
        free(root);
    }
}

// Driver code
int main() {
    struct node *root = NULL;
    root = insert(root, 8);
    insert(root, 3);
    insert(root, 1);
    insert(root, 6);
    insert(root, 7);
    insert(root, 10);
    insert(root, 14);
    insert(root, 4);

    printf("\nInorder traversal: \n");
    inorder(root);

    printf("\nPreorder traversal: \n");
    preorder(root);

    printf("\nPostorder traversal: \n");
    postorder(root);
```

```
    // Free memory
    freeTree(root);


    return 0;
}
```

Output

```
Inorder traversal:
1 -> 3 -> 4 -> 6 -> 7 -> 8 -> 10 -> 14 ->
Preorder traversal:
8 -> 3 -> 1 -> 6 -> 4 -> 7 -> 10 -> 14 ->
Postorder traversal:
1 -> 4 -> 7 -> 6 -> 3 -> 14 -> 10 -> 8 ->
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

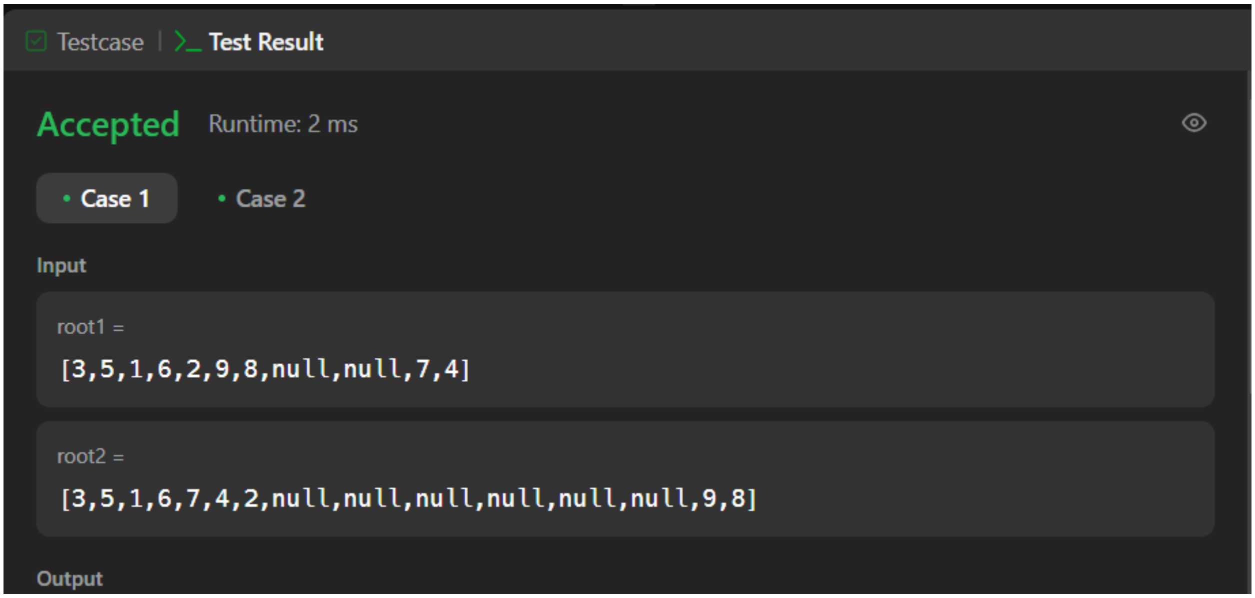
```
Inorder traversal:
1 -> 3 -> 4 -> 7 -> 8 -> 10 -> 16 -> 20 -> 24 -> 41 ->
Preorder traversal:
8 -> 3 -> 1 -> 7 -> 4 -> 16 -> 10 -> 41 -> 20 -> 24 ->
Postorder traversal:
1 -> 4 -> 7 -> 3 -> 10 -> 24 -> 20 -> 41 -> 16 -> 8 ->
Process returned 0 (0x0)   execution time : 0.187 s
Press any key to continue.
```

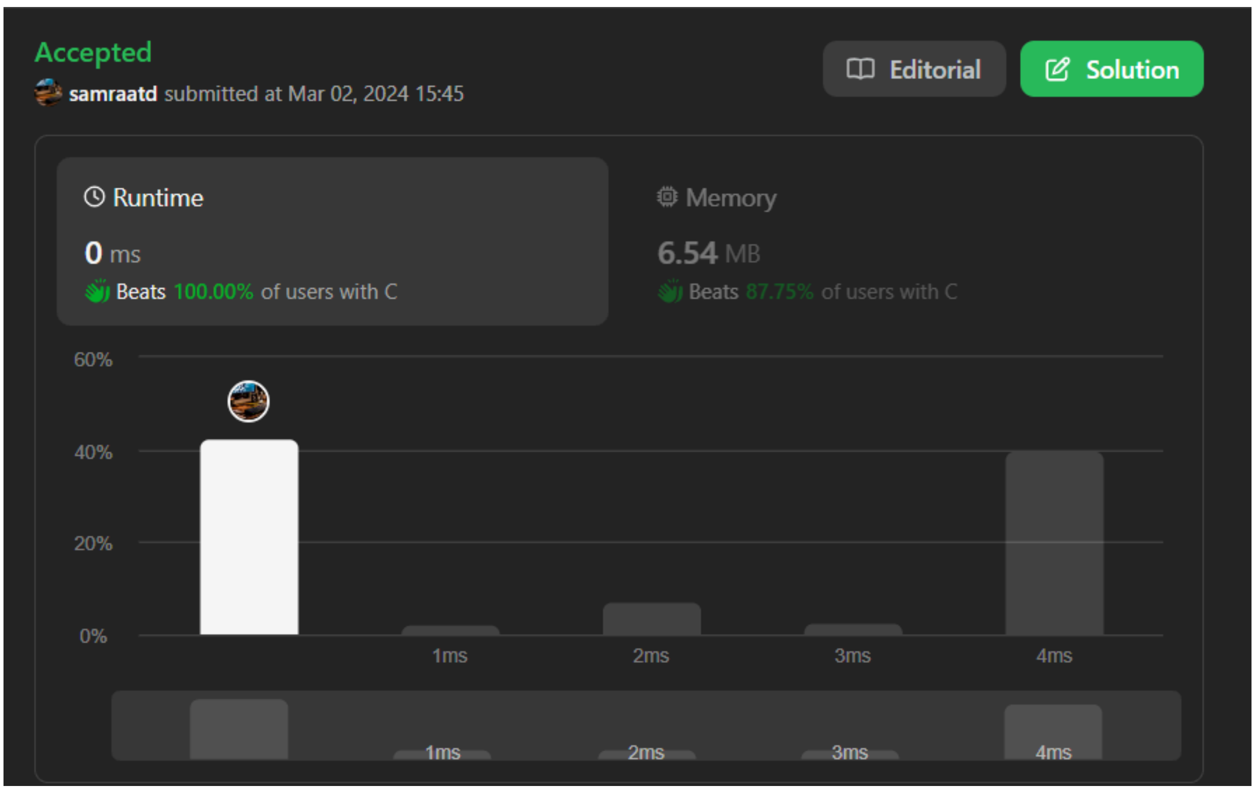
Leetcode

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
```

```
void findLeaves(struct TreeNode* node, int** leafValues, int* size, int* capacity) {
   if (node == NULL) {
      return;
   }

   if (node->left == NULL && node->right == NULL) {
      if (*size >= *capacity) {
         *capacity *= 2;
         *leafValues = (int*) realloc(*leafValues, *capacity * sizeof(int));
      }
      (*leafValues)[(*size)++] = node->val;
   }

   findLeaves(node->left, leafValues, size, capacity);
   findLeaves(node->right, leafValues, size, capacity);
}

bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
   int *leaves1 = (int*) malloc(sizeof(int) * 10);
   int size1 = 0, capacity1 = 10;

   int *leaves2 = (int*) malloc(sizeof(int) * 10);
   int size2 = 0, capacity2 = 10;
```

```
    findLeaves(root1, &leaves1, &size1, &capacity1);
    findLeaves(root2, &leaves2, &size2, &capacity2);

    if (size1 != size2) {
        free(leaves1);
        free(leaves2);
        return false;
    }

    for (int i = 0; i < size1; i++) {
        if (leaves1[i] != leaves2[i]) {
            free(leaves1);
            free(leaves2);
            return false;
        }
    }

    free(leaves1);
    free(leaves2);
    return true;
}
```


Testcase
Test Result
Accepted
Runtime: 2 ms
Case 1
Case 2
Input
root1 =
[3,5,1,6,2,9,8,null,null,7,4]
root2 =
[3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
Output


Accepted
samraatd submitted at Mar 02, 2024 15:45
Editorial
Solution
Runtime
0 ms
Beats 100.00% of users with C
Memory
6.54 MB
Beats 87.75% of users with C
60%
40%
20%
0%
1ms
2ms
3ms
4ms

LAB 9

Question

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

Code (a)

```
#include <stdio.h>


int n, i, j, visited[10], queue[10], front = -1, rear = -1;

int adj[10][10];


void bfs(int v)

{

   for (i = 1; i <= n; i++)

   {

      if (adj[v][i] && !visited[i])

      {

         queue[++rear] = i;

         visited[i] = 1;

      }

   }


   if (front <= rear)

   {

      bfs(queue[++front]);
```

```
    }
}

int main()
{
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
    {
        queue[i] = 0;
        visited[i] = 0;
    }

    printf("Enter graph data in matrix form:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }

    printf("Enter the starting vertex: ");
```

```
    scanf("%d", &v);
    visited[v] = 1;
    bfs(v);

    printf("The nodes which are reachable are:\n");
    for (i = 1; i <= n; i++)
    {
        if (visited[i])
        {
            printf("%d\t", i);
        }
    }

    for (i = 1; i <= n; i++)
    {
        if (!visited[i])
        {
            printf("\nBFS is not possible. Not all nodes are reachable\n");
            break;
        }
    }

    return 0;
}
```

Output (a)

```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 01
0 1 1 0
0
Enter the starting vertex: 2
The node which are reachable are:
1       2       3       4
```

```
Enter the number of vertices: 4
Enter graph data in matrix form:
1 0 0 1
1 1 1 1
0 0 0 0
0 1 1 0
Enter the starting vertex: 3
The nodes which are reachable are:
3
BFS is not possible. Not all nodes are reachable
```

Code (b)

```
#include<stdio.h>


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


void dfs(int v) {
   int i;
   reach[v] = 1;
   for (i = 1; i <= n; i++) {
```

```
      if (a[v][i] && !reach[i]) {
         printf("\n %d->%d", v, i);
         dfs(i);
      }
   }
}

int main() {
   int i, j, count = 0;
   printf("\n Enter number of vertices:");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      reach[i] = 0;
      for (j = 1; j <= n; j++) {
         a[i][j] = 0;
      }
   }
   printf("\n Enter the adjacency matrix:\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         scanf("%d", &a[i][j]);
      }
   }
   dfs(1);
   printf("\n");
```

```
    for (i = 1; i <= n; i++) {
        if (reach[i]) {
            count++;
        }
    }
    if (count == n) {
        printf("\n Graph is connected");
    } else {
        printf("\n Graph is not connected");
    }
    return 0;
}
```

Output (b)

```
 Enter number of vertices:4

 Enter the adjacency matrix:
1 0 0 1
1 1 0 0
0 1 0 1
1 1 1 1

 1->4
 4->2
 4->3

 Graph is connected
```

```
 Enter number of vertices:4

 Enter the adjacency matrix:
1 0 0 0
0 0 0 0
0 0 1 1
0 0 1 1


 Graph is not connected
```