VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Sumit Kumar Mahato(1BF24CS305)

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
August-December 2025

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 Sumit Kumar Mahato(1BF24CS305), 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 2025-2026. 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.

Mrs. Anusha S Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Operations (Push, Pop, Display)	4
2	Infix to Postfix Expression	7
3a	Queue Operations (Insert, Delete, Display)	10
3b	Circular Queue Operations (Insert, Delete, Display)	13
4a	Singly Linked List Insert operations	17
4b	LeetCode: Remove Linked List elements	22
5a	Singly Linked List Delete operations	24
5b	LeetCode: Middle of Linked List	29
6a	Sort, Reverse, Concatenate singly linked list operations	31
6b	Stack and Queue Using Linked List	36
7a	Doubly Linked list operations	41
7b	LeetCode: Linked List cycle	47
8a	Binary Search tree creation and traversals (inorder, preorder, postorder)	48
8b	LeetCode: Merge Two Binary Trees	51
9a	Traverse graph using BFS method	53
9b	Traverse graph using DFS method	55
10	Hashing	57

Course outcomes:

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

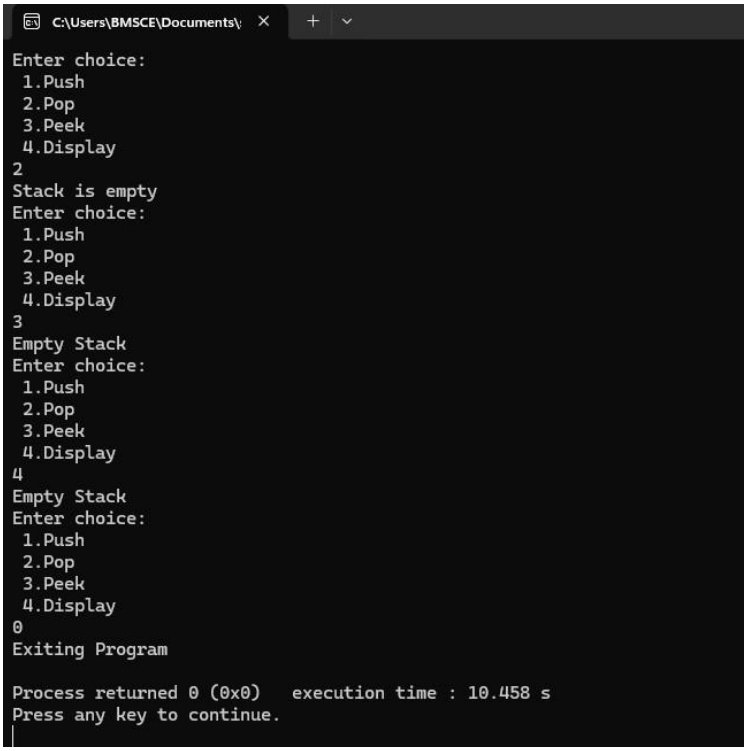
Lab program 1:

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

```
#include<stdio.h>
#define MAX 3

int stack[MAX];
int top = -1;

void push(){
  int x;
  printf("Enter the data:");
  scanf("%d",&x);
  if(top==MAX-1){
     printf("Stack overflow\n");
   }
   else{
     top++;
     stack[top] = x;
   }
}

void pop(){
  int value;
  if(top == -1){
     printf("Stack is empty\n");
  }else{
     value = stack[top];
     top --;
     printf("The popped element is %d \n", value);
  }
}

void peek(){
  if(top == -1){
     printf("Empty Stack\n");
  }else{
     printf("Peek value: %d\n",stack[top]);
  }
}
void display(){
  int i;
  if(top == -1){
```

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



int main() {
    int choice;

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

        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program \n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Output:


C:\Users\BMSCE\Documents\
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
2
Stack is empty
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
3
Empty Stack
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
4
Empty Stack
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
0
Exiting Program
Process returned 0 (0x0) execution time : 10.458 s
Press any key to continue.


C:\Users\BMSCE\Documents\
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
1
Enter the data:5
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
1
Enter the data:6
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
1
Enter the data:7
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
1
Enter the data:8
Stack overflow
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
2
The popped element is 7
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
3
Peek value: 6
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
4
Stack elements: 65
Enter choice:
1.Push
2.Pop
3.Peek
4.Display
0
Exiting Program

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<ctype.h>

#define MAX 50

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

char push(char x){
  stack[++top] = x;
}

char pop(){
  return (stack[top--]);
}

int pr(char ele){
  if(ele == '^'){
     return (3);
  }
  else if(ele == '/' || ele =='*'){
     return 2;
  }
  else if(ele == '+' || ele=='-'){
     return 1;
  }else{
     return 0;
  }
}

void main()
{
  char infix[50], postfix[50], ch, elem;
  int i=0,k=0;
  printf("Enter Infix Expression : ");
  scanf("%s",infix);
  push('#');
  while( (ch = infix[i++]) != '\0')
  {
     if( ch == '(' ) push(ch);
     else
     if(isalnum(ch)) postfix[k++] = ch;
     else
```

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

Output:

```
Enter Infix Expression : A+(B*C-(D/E^F)*G)*H

Postfix Expression = ABC*DEF^/G*-H*+

Process returned 38 (0x26)   execution time : 15.121 s
Press any key to continue.
```

Lab Program 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

```
#include<stdio.h>
#include <stdbool.h>
#define m 1
int front = - 1;
int rear = -1;
int queue[m];

void enque(int x){
  if(rear == m-1){
     printf("Queue is full\n");
  }else if(front==-1 && rear ==-1){
     front=rear=0;
     queue[rear] = x;
  }else {
     rear++;
     queue[rear] = x;
  }
}

int deque(){
  if(front == - 1 && rear ==-1){
     printf("Queue is empty\n");
  }else if(front==0 && rear ==0){
     int ele = queue[front];
     front=rear=-1;
     return ele;
  }else {
     int ele = queue[front];
     front++ ;
     return ele;
  }
}

void display(){
int i;
printf("Elements are:-\n");
for(i=front;i<=rear;i++){
  printf("%d\n",queue[i]);
}
}
```

```
int main(){
int  choice;
int x;
bool exit = false;
do{
  printf("Enter choice:\n");
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
scanf("%d",&choice);
switch(choice){
case 1:
  printf("Enter num to input\n");
  scanf("%d",&x);
  enque(x);
  break;
case 2:
  deque();
  break;
case 3:
  display();
  break;
case 4:
  exit = true;
  break;
default:
  printf("Enter valid input");
  break;

}
}while(exit == false);
}
```

Output:

```
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Queue is empty
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
3
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Elements are:-
3
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
2
Queue is full
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
```

Lab Program 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>
#include <stdbool.h>
#define m 5
int front = - 1;
int rear = -1;
int queue[m];

void enque(int x){
  if((rear+1)%m == front){
     printf("Queue is full\n");
  }else if(front==-1 && rear ==-1){
     front=rear=0;
     queue[rear] = x;
  }else {
     rear=(rear+1)%m;
     queue[rear] = x;
  }
}

int deque(){
  if(front == - 1 && rear ==-1){
     printf("Queue is empty\n");
  }else if(front==0 && rear ==0){
     int ele = queue[front];
     front=rear=-1;
     return ele;
  }else {
     int ele = queue[front];
     front = (front+1)%m;
     return ele;
  }
}

void display(){

if(front==-1 && rear ==-1){
     printf("Queue is empty");
  }
else{
     int i=front;
   printf("Elements are:-\n");
```

```
    while(1){
      printf("%d\n",queue[i]);
      if(i==rear){
         break;
      }
      i = (i+1)%m;
    }
}
}

int main(){
int  choice;
int x;
bool exit = false;
do{
  printf("Enter choice:\n");
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
scanf("%d",&choice);
switch(choice){
case 1:
  printf("Enter num to input\n");
  scanf("%d",&x);
  enque(x);
  break;
case 2:
  deque();
  break;
case 3:
  display();
  break;
case 4:
  exit = true;
  break;
default:
  printf("Enter valid input");
  break;

}
}while(exit == false);
}
```

Output:

```
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
20
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
2
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
30
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
25
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Elements are:-
10
20
```

```
30
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
25
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
3
Elements are:-
10
20
30
25
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
25
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
1
Enter num to input
35
Queue is full
Enter choice:
1.Insert
2.Delete
3.Display
4.Exit
4
```

Lab Program 4a:

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

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

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

struct Node *head = NULL;

struct Node* createANode(int data) {
  struct Node *newNode = (struct Node*) malloc(sizeof(struct Node));
  if (newNode == NULL) {
     printf("Memory allocation failed!\n");
     exit(1);
  }
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}

void createList(int n) {
  struct Node *newNode, *temp = NULL;
  int data, i;

  if (n <= 0) {
     printf("Number of nodes should be greater than 0.\n");
     return;
  }

  for (i = 1; i <= n; i++) {
     printf("Enter data for node %d: ", i);
     scanf("%d", &data);
     newNode = createANode(data);

     if (head == NULL) {
        head = newNode;
     } else {
        temp->next = newNode;
     }
     temp = newNode;
  }
```

```
  printf("\nLinked list created successfully.\n");
}

void insertAtBeginning(int data) {
  struct Node *newNode = createANode(data);
  newNode->next = head;
  head = newNode;
  printf("Inserted at beginning\n");
}

void insertAtEnd(int data) {
  struct Node *newNode = createANode(data);
  if (head == NULL) {
     head = newNode;
  } else {
     struct Node *temp = head;
     while (temp->next != NULL) {
        temp = temp->next;
     }
     temp->next = newNode;
  }
  printf("Inserted at end\n");
}

void insertAtAnyPos(int data, int pos) {
  if (pos <= 0) {
     printf("Enter a valid position\n");
     return;
  }

  if (pos == 1) {
     insertAtBeginning(data);
     return;
  }

  struct Node *newNode = createANode(data);
  struct Node *temp = head;
  int i;

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

  if (temp == NULL) {
     printf("Position out of range\n");
     return;
  }
```

```
  newNode->next = temp->next;
  temp->next = newNode;

  printf("Node inserted at position %d\n", pos);
}

void display() {
  if (head == NULL) {
     printf("Linked list is empty\n");
     return;
  }

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

int main() {
  int data, choice, pos, n;
  bool exit = false;

  printf("--- MENU ---\n");
  printf("1. Create List\n2. Insert At Beginning\n3. Insert At End\n4. Insert At Any Position\n5.
Display\n6. Exit\n");

  while (!exit) {
     printf("\nEnter choice: ");
     scanf("%d", &choice);

     switch (choice) {
        case 1:
           printf("Enter the number of nodes: ");
           scanf("%d", &n);
           createList(n);
           break;
        case 2:
           printf("Enter data to insert at beginning: ");
           scanf("%d", &data);
           insertAtBeginning(data);
           break;
        case 3:
           printf("Enter data to insert at end: ");
```

```
            scanf("%d", &data);
            insertAtEnd(data);
            break;
        case 4:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
            insertAtAnyPos(data, pos);
            break;
        case 5:
            display();
            break;
        case 6:
            exit = true;
            break;
        default:
            printf("Enter valid choice\n");
            break;
    }
  }

  return 0;
}
```

Output:

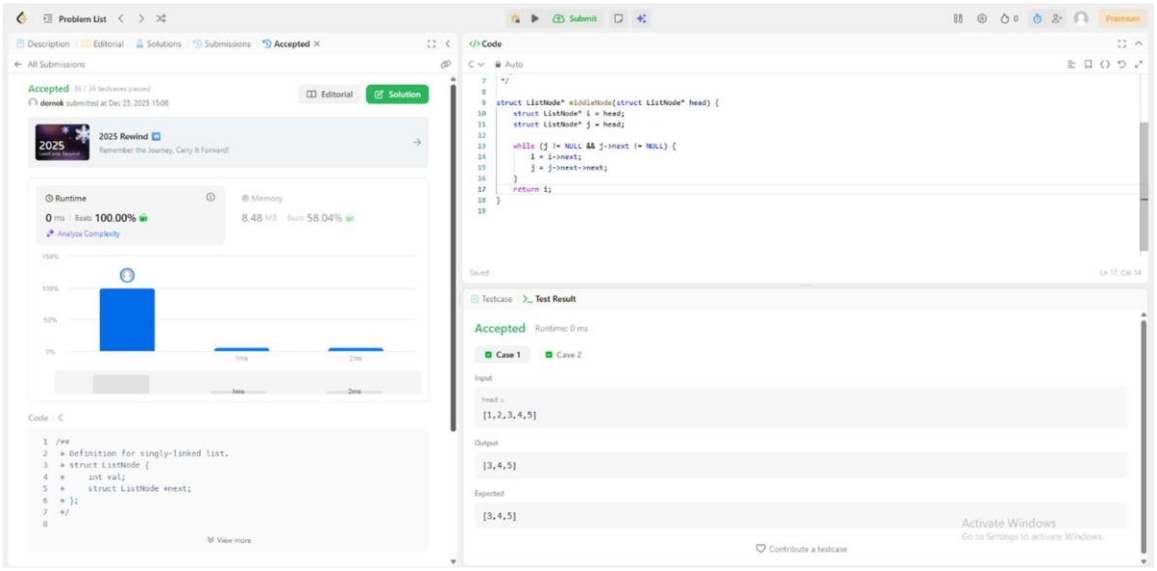
```
--- MENU ---
1. Create List
2. Insert At Beginning
3. Insert At End
4. Insert At Any Position
5. Display
6. Exit

Enter choice: 1
Enter the number of nodes: 3
Enter data for node 1: 100
Enter data for node 2: 200
Enter data for node 3: 300

Linked list created successfully.

Enter choice: 4
Enter data to insert: 250
Enter position: 3
Node inserted at position 3

Enter choice: 3
Enter data to insert at end: 500
Inserted at end

Enter choice: 5
Linked List: 100 -> 200 -> 250 -> 300 -> 500 -> NULL

Enter choice: 6

Process returned 0 (0x0)   execution time : 1105.275 s
Press any key to continue.
```

Lab program 4b:

LeetCode: Middle of the linked list

```
struct ListNode* middleNode(struct ListNode* head) {
  struct ListNode *slow = head;
  struct ListNode *fast = head;

  while (fast != NULL && fast->next != NULL) {
     slow = slow->next;
     fast = fast->next->next;
  }

  return slow;
}
```

Output:


Problem List
Submit
Description
Editorial
Solutions
Submissions
Accepted
All Submissions
Accepted
dornok submitted at Dec 25, 2025 15:06
Editorial
Solution
2025 Rewind
Remember the Journey, Carry It Forward!
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
8.48 MB | Beats 58.04%
Code | C
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
View more
Code
Auto
*/
struct ListNode* middleNode(struct ListNode* head) {
struct ListNode* i = head;
struct ListNode* j = head;
while (j != NULL && j->next != NULL) {
i = i->next;
j = j->next->next;
}
return i;
}
Saved
Ln 17, Col 14
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
head =
[1,2,3,4,5]
Output
[3,4,5]
Expected
[3,4,5]
Contribute a testcase
Activate Windows
Go to Settings to activate Windows.

Lab program 5a:

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>
#include<stdbool.h>

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

struct Node *head = NULL;

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

void createList(int n){
  if (n <= 0) {
     printf("Enter a valid number\n");
     return;
  }

  struct Node *newNode, *temp = NULL;
  int data;

  for (int i = 0; i < n; i++) {
     printf("Enter data for node:\n");
     scanf("%d", &data);

     newNode = createNode(data);

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

  printf("Linked list is created\n");
```

```
}

void deleteFirstEle() {
  if (head == NULL) {
     printf("LL is empty\n");
     return;
  }

  struct Node *temp = head;
  head = head->next;
  printf("%d was deleted \n", temp->data);
  free(temp);
}

void lastEle() {
  if (head == NULL) {
     printf("LL is empty\n");
     return;
  }

  struct Node *temp = head, *prev = NULL;

  if (head->next == NULL) {
     printf("Deleted Ele is %d\n", head->data);
     free(head);
     head = NULL;
     return;
  }

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

  printf("Deleted Ele is %d\n", temp->data);
  prev->next = NULL;
  free(temp);
}

void deleteSpecificEle(int val) {
  struct Node *temp = head, *prev = NULL;

  if (head == NULL) {
     printf("LL is empty\n");
     return;
  }
```

```
  if (head->data == val) {
     temp = head;
     head = head->next;
     printf("Deleted ele %d\n", temp->data);
     free(temp);
     return;
  }

  while (temp != NULL && temp->data != val) {
     prev = temp;
     temp = temp->next;
  }

  if (temp == NULL) {
     printf("Element not found\n");
     return;
  }

  prev->next = temp->next;
  printf("Deleted ele %d\n", temp->data);
  free(temp);
}

void display() {
  if (head == NULL) {
     printf("Linked list is empty\n");
     return;
  }

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

int main() {
  int choice, n, val;
  bool exit = false;

  printf("--- MENU ---\n");
  printf("1. Create List\n2. Delete First Ele\n3. Delete Last Ele\n4. Delete value\n5. Display\n6.
Exit\n");

  while (!exit) {
```

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

    switch (choice) {
      case 1:
        printf("Enter the number of nodes: ");
        scanf("%d", &n);
        createList(n);
        break;
      case 2:
        deleteFirstEle();
        break;
      case 3:
        lastEle();
        break;
      case 4:
        printf("Enter data to remove: ");
        scanf("%d", &val);
        deleteSpecificEle(val);
        break;
      case 5:
        display();
        break;
      case 6:
        exit = true;
        break;
      default:
        printf("Enter valid choice\n");
        break;
    }
  }
  return 0;
}
```

Output:

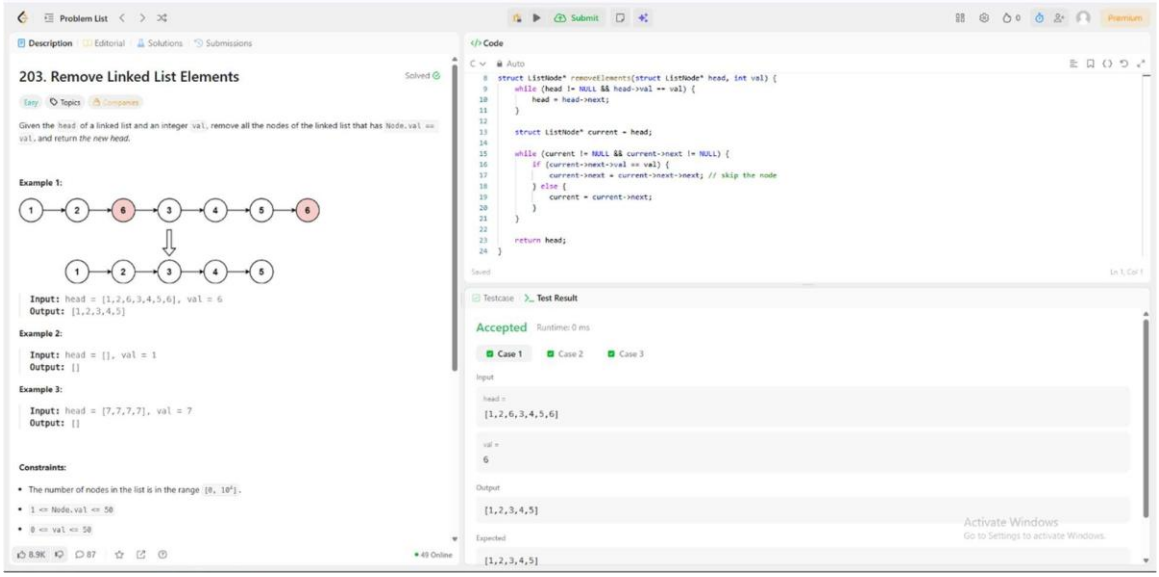
```
--- MENU ---
1. Create List
2. Delete First Ele
3. Delete Last Ele
4. Delete value
5. Display
6. Exit

Enter choice: 1
Enter the number of nodes: 5
Enter data for node:
10
Enter data for node:
20
Enter data for node:
30
Enter data for node:
40
Enter data for node:
50
Linked list is created

Enter choice: 2
10 was deleted

Enter choice: 3
Deleted Ele is 50

Enter choice: 4
Enter data to remove: 40
Deleted ele 40

Enter choice: 5
Linked List: 20 -> 30 -> NULL

Enter choice: 6
```

Lab Program 5b:

LeetCode: Middle of Linked List

```
struct ListNode* removeElements(struct ListNode* head, int val) {
    struct ListNode dummy;
    dummy.next = head;

    struct ListNode *prev = &dummy;
    struct ListNode *curr = head;

    while (curr != NULL) {
        if (curr->val == val) {
            prev->next = curr->next;
            curr = curr->next;
        } else {
            prev = curr;
            curr = curr->next;
        }
    }

    return dummy.next;
}
```

Output:


Problem List
Submit
Premium
Description
Editorial
Solutions
Submissions
203. Remove Linked List Elements
Solved
Easy
Topics
Companies
Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.
Example 1:
Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]
Example 2:
Input: head = [], val = 1
Output: []
Example 3:
Input: head = [7,7,7,7], val = 7
Output: []
Constraints:
The number of nodes in the list is in the range [0, 10^4].
1 <= Node.val <= 50
0 <= val <= 50
8.9K
87
49 Online
Code
Auto
struct ListNode* removeElements(struct ListNode* head, int val) {
while (head != NULL && head->val == val) {
head = head->next;
}
struct ListNode* current = head;
while (current != NULL && current->next != NULL) {
if (current->next->val == val) {
current->next = current->next->next; // skip the node
} else {
current = current->next;
}
}
return head;
}
Saved
Ln 1, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
head =
[1,2,6,3,4,5,6]
val =
6
Output
[1,2,3,4,5]
Expected
[1,2,3,4,5]
Activate Windows
Go to Settings to activate Windows.

Lab Program 6a:

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

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

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

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

void insertEnd(struct node **head, int data) {
  struct node *newNode = createNode(data);

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

  struct node *temp = *head;
  while (temp->next != NULL)
     temp = temp->next;

  temp->next = newNode;
}

void display(struct node *head) {
  if (head == NULL) {
     printf("List is empty\n");
     return;
  }
  while (head != NULL) {
     printf("%d ", head->data);
     head = head->next;
  }
  printf("\n");
}
void sortList(struct node *head) {
  struct node *i, *j;
```

```
  int temp;

  for (i = head; i != NULL && i->next != NULL; i = i->next) {
    for (j = i->next; j != NULL; j = j->next) {
      if (i->data > j->data) {
        temp = i->data;
        i->data = j->data;
        j->data = temp;
      }
    }
  }
}

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

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

struct node* concat(struct node *head1, struct node *head2) {
  if (head1 == NULL) return head2;
  if (head2 == NULL) return head1;

  struct node *temp = head1;
  while (temp->next != NULL)
    temp = temp->next;

  temp->next = head2;
  return head1;
}

int main() {
  struct node *head1 = NULL, *head2 = NULL;
  int choice, value;
  printf("\n----- MENU ------ \n");
    printf("1. Insert into List 1\n");
    printf("2. Insert into List 2\n");
    printf("3. Display List 1\n");
    printf("4. Display List 2\n");
    printf("5. Sort List 1 (FOR loop only)\n");
    printf("6. Reverse List 1\n");
```

```
    printf("7. Concatenate List 1 and List 2\n");
    printf("8. Exit\n");

while (1) {

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

    switch (choice) {
    case 1:
        printf("Enter value: ");
        scanf("%d", &value);
        insertEnd(&head1, value);
        break;

    case 2:
        printf("Enter value: ");
        scanf("%d", &value);
        insertEnd(&head2, value);
        break;

    case 3:
        printf("List 1: ");
        display(head1);
        break;

    case 4:
        printf("List 2: ");
        display(head2);
        break;

    case 5:
        sortList(head1);
        printf("List 1 sorted.\n");
        break;

    case 6:
        head1 = reverseList(head1);
        printf("List 1 reversed.\n");
        break;

    case 7:
        head1 = concat(head1, head2);
        printf("Lists concatenated 8 Now List 1: ");
        display(head1);
        break;
```

```
        case 8:
            exit(0);

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

    return 0;
}
```

Output:

```
----- MENU -----
1. Insert into List 1
2. Insert into List 2
3. Display List 1
4. Display List 2
5. Sort List 1 (FOR loop only)
6. Reverse List 1
7. Concatenate List 1 and List 2
8. Exit
Enter choice: 1
Enter value: 10
Enter choice: 1
Enter value: 20
Enter choice: 1
Enter value: 30
Enter choice: 2
Enter value: 40
Enter choice: 2
Enter value: 50
Enter choice: 2
Enter value: 60
Enter choice: 3
List 1: 10 20 30
Enter choice: 4
List 2: 40 50 60
Enter choice: 5
List 1 sorted.
Enter choice: 3
List 1: 10 20 30
Enter choice: 5
List 1 sorted.
Enter choice: 7
Lists concatenated Now List 1: 10 20 30 40 50 60
Enter choice: 8
```

Lab Program 6b:

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

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

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

struct Node *head = NULL;  // for Stack
struct Node *front = NULL;  // for Queue
struct Node *rear = NULL;   // for Queue

void push(int x) {
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  if (!newNode) {
     printf("Memory allocation failed\n");
     return;
  }
  newNode->value = x;
  newNode->next = head;
  head = newNode;
}

void pop() {
  if (head == NULL) {
     printf("Stack Underflow\n");
     return;
  }
  struct Node *temp = head;
  head = head->next;
  printf("Popped: %d\n", temp->value);
  free(temp);
}

void displayStack() {
  if (head == NULL) {
     printf("Stack is empty\n");
     return;
  }
  struct Node *ptr = head;
  printf("Stack: ");
  while (ptr != NULL) {
```

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

void enqueue(int x) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        return;
    }
    newNode->value = x;
    newNode->next = NULL;

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

void dequeue() {
    if (front == NULL) {
        printf("Queue Underflow\n");
        return;
    }
    struct Node *temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    printf("Dequeued: %d\n", temp->value);
    free(temp);
}

void displayQueue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct Node *ptr = front;
    printf("Queue: ");
    while (ptr != NULL) {
        printf("%d ", ptr->value);
```

```
        ptr = ptr->next;
    }
    printf("\n");
}

int main() {
    int choice, x;
    printf("\n1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue (Queue)\n");
        printf("5. Dequeue (Queue)\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");

    do {

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

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &x);
                push(x);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Enter value to enqueue: ");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 5:
                dequeue();
                break;
            case 6:
                displayQueue();
                break;
            case 7:
                printf("Exiting...\n");
                break;
            default:
```

```
            printf("Invalid choice\n");
        }
    } while (choice != 7);

    return 0;
}
```

Output:

```
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter choice: 1
Enter value to push: 30
Enter choice: 1
Enter value to push: 20
Enter choice: 1
Enter value to push: 40
Enter choice: 2
Popped: 40
Enter choice: 3
Stack: 20 30
Enter choice: 4
Enter value to enqueue: 10
Enter choice: 4
Enter value to enqueue: 20
Enter choice: 5
Dequeued: 10
Enter choice: 6
Queue: 20
Enter choice: 7
Exiting...
```

Lab Program 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 d) Display the contents of the list

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *prev, *next;
};
struct Node *head = NULL;
struct Node* createNode(int data)
{
  struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->prev = newNode->next = NULL;
  return newNode;
}
void createList(int data)
{
  struct Node* newNode = createNode(data);

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

void insertLeft(int value, int newData)
{
  struct Node* temp = head;

  while (temp != NULL && temp->data != value)
     temp = temp->next;

  if (temp == NULL)
     {
     printf("Node with value %d not found!\n", value);
     return;
  }
  struct Node* newNode = createNode(newData);
```

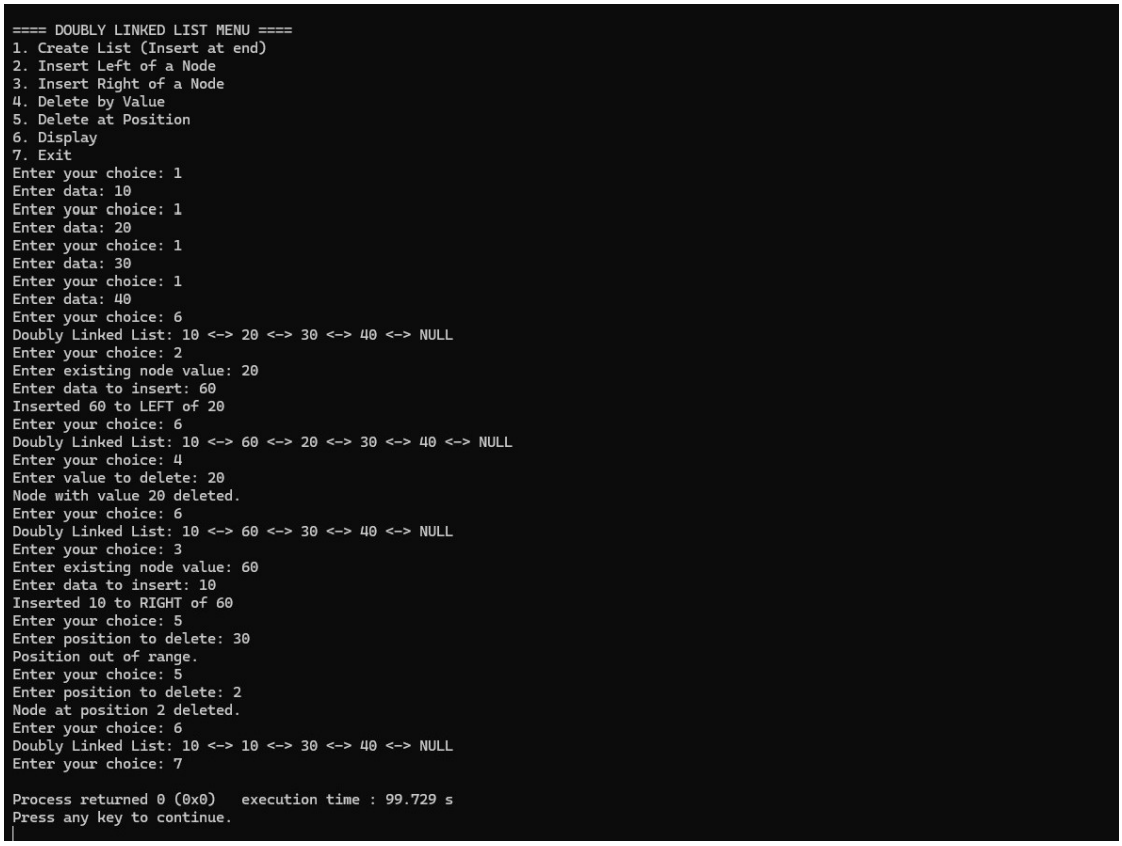
```
  newNode->next = temp;
  newNode->prev = temp->prev;

  if (temp->prev != NULL)
     temp->prev->next = newNode;
  else
     head = newNode;
  temp->prev = newNode;
  printf("Inserted %d to LEFT of %d\n", newData, value);
}

void insertRight(int value, int newData)
{
  struct Node* temp = head;
  while (temp != NULL && temp->data != value)
     temp = temp->next;
  if (temp == NULL){
     printf("Node with value %d not found!\n", value);
     return;
  }
  struct Node* newNode = createNode(newData);
  newNode->prev = temp;
  newNode->next = temp->next;
  if (temp->next != NULL)
     temp->next->prev = newNode;

  temp->next = newNode;

  printf("Inserted %d to RIGHT of %d\n", newData, value);
}

void deleteByValue(int value)
{
  struct Node* temp = head;

  while (temp != NULL && temp->data != value)
     temp = temp->next;

  if (temp == NULL) {
     printf("Node with value %d not found!\n", value);
     return;
  }

  if (temp->prev != NULL)
     temp->prev->next = temp->next;
  else
     head = temp->next;
```

```
  if (temp->next != NULL)
    temp->next->prev = temp->prev;

  free(temp);
  printf("Node with value %d deleted.\n", value);
}

void display() {
  struct Node* temp = head;
  if (temp == NULL) {
    printf("List is empty.\n");
    return;
  }
  printf("Doubly Linked List: ");
  while (temp != NULL) {
    printf("%d <-> ", temp->data);
    temp = temp->next;
  }
  printf("NULL\n");
}
void deleteAtPosition(int position) {
  if (head == NULL || position <= 0) {
    printf("Invalid position.\n");
    return;
  }

  struct Node* temp = head;

  for (int i = 1; temp != NULL && i < position; i++)
    temp = temp->next;

  if (temp == NULL) {
    printf("Position out of range.\n");
    return;
  }

  if (temp->prev != NULL)
    temp->prev->next = temp->next;
  else
    head = temp->next;

  if (temp->next != NULL)
    temp->next->prev = temp->prev;

  printf("Node at position %d deleted.\n", position);
  free(temp);
```

```
}

int main() {
  int choice, data, value, pos;
    printf("\n==== DOUBLY LINKED LIST MENU ====\n");
    printf("1. Create List (Insert at end)\n");
    printf("2. Insert Left of a Node\n");
    printf("3. Insert Right of a Node\n");
    printf("4. Delete by Value\n");
    printf("5. Delete at Position\n");
    printf("6. Display\n");
    printf("7. Exit\n");

  while (1) {
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter data: ");
        scanf("%d", &data);
        createList(data);
        break;

      case 2:
        printf("Enter existing node value: ");
        scanf("%d", &value);
        printf("Enter data to insert: ");
        scanf("%d", &data);
        insertLeft(value, data);
        break;

      case 3:
        printf("Enter existing node value: ");
        scanf("%d", &value);
        printf("Enter data to insert: ");
        scanf("%d", &data);
        insertRight(value, data);
        break;

      case 4:
        printf("Enter value to delete: ");
        scanf("%d", &value);
        deleteByValue(value);
        break;

      case 5:
```

```
            printf("Enter position to delete: ");
            scanf("%d", &pos);
            deleteAtPosition(pos);
            break;

        case 6:
            display();
            break;

        case 7:
            exit(0);

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

Output:



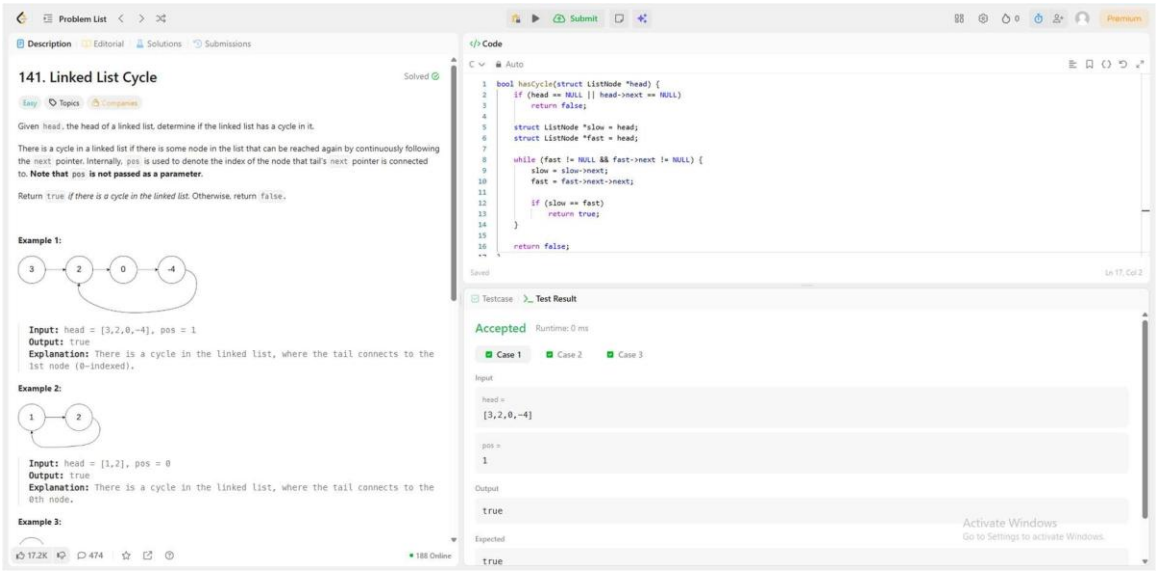
```
==== DOUBLY LINKED LIST MENU ====
1. Create List (Insert at end)
2. Insert Left of a Node
3. Insert Right of a Node
4. Delete by Value
5. Delete at Position
6. Display
7. Exit
Enter your choice: 1
Enter data: 10
Enter your choice: 1
Enter data: 20
Enter your choice: 1
Enter data: 30
Enter your choice: 1
Enter data: 40
Enter your choice: 6
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
Enter your choice: 2
Enter existing node value: 20
Enter data to insert: 60
Inserted 60 to LEFT of 20
Enter your choice: 6
Doubly Linked List: 10 <-> 60 <-> 20 <-> 30 <-> 40 <-> NULL
Enter your choice: 4
Enter value to delete: 20
Node with value 20 deleted.
Enter your choice: 6
Doubly Linked List: 10 <-> 60 <-> 30 <-> 40 <-> NULL
Enter your choice: 3
Enter existing node value: 60
Enter data to insert: 10
Inserted 10 to RIGHT of 60
Enter your choice: 5
Enter position to delete: 30
Position out of range.
Enter your choice: 5
Enter position to delete: 2
Node at position 2 deleted.
Enter your choice: 6
Doubly Linked List: 10 <-> 10 <-> 30 <-> 40 <-> NULL
Enter your choice: 7

Process returned 0 (0x0)   execution time : 99.729 s
Press any key to continue.
```

Lab Program 7b:

LeetCode: Linked List Cycle

```
#include <stdbool.h>

bool hasCycle(struct ListNode *head) {
  struct ListNode *slow = head;
  struct ListNode *fast = head;

  while (fast != NULL && fast->next != NULL) {
    slow = slow->next;
    fast = fast->next->next;

    if (slow == fast) {
      return true;
    }
  }

  return false;
}
```

Output:



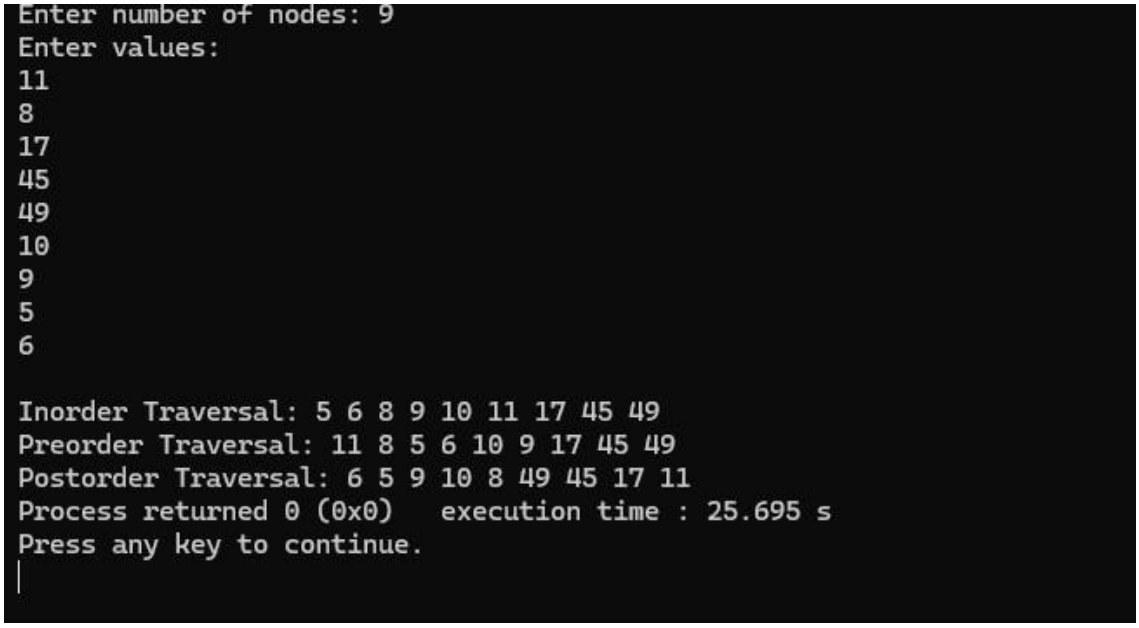
Lab Program 8a:

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 c) To display the elements in the tree.

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

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

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

struct Node* insert(struct Node* root, int value) {
  if (root == NULL)
     return createNode(value);

  if (value < root->data)
     root->left = insert(root->left, value);
  else
     root->right = insert(root->right, value);

  return root;
}

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

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

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

int main() {
  struct Node* root = NULL;
  int n, value;

  printf("Enter number of nodes: ");
  scanf("%d", &n);

  printf("Enter values:\n");
  for(int i = 0; i < n; i++) {
     scanf("%d", &value);
     root = insert(root, value);
  }

  printf("\nInorder Traversal: ");
  inorder(root);

  printf("\nPreorder Traversal: ");
  preorder(root);

  printf("\nPostorder Traversal: ");
  postorder(root);

  return 0;
}
```

Output:

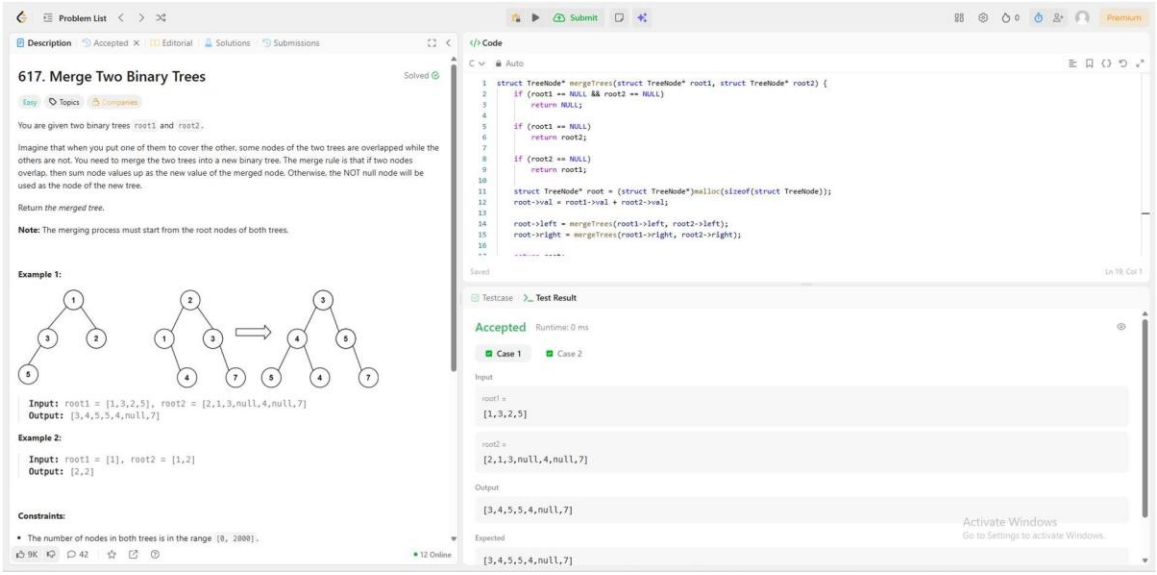

Enter number of nodes: 9
Enter values:
11
8
17
45
49
10
9
5
6
Inorder Traversal: 5 6 8 9 10 11 17 45 49
Preorder Traversal: 11 8 5 6 10 9 17 45 49
Postorder Traversal: 6 5 9 10 8 49 45 17 11
Process returned 0 (0x0) execution time : 25.695 s
Press any key to continue.

Lab 8b:

LeetCode: Merge Two Binary Trees

```
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
if(root1==NULL && root2==NULL)
 return NULL;
 if(root1==NULL)
 return root2;
 if (root2==NULL)
 return root1;
 struct TreeNode* root =(struct TreeNode*)malloc(sizeof(struct TreeNode));
root->val=root1->val+root2->val;
 root->left=mergeTrees(root1->left,root2->left);
 root->right=mergeTrees(root1->right,root2->right);
 return root;
}
```

Output:


Problem List
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
617. Merge Two Binary Trees
Solved
Easy
Topics
Companies
You are given two binary trees root1 and root2.
Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.
Return the merged tree.
Note: The merging process must start from the root nodes of both trees.
Example 1:
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]
Example 2:
Input: root1 = [1], root2 = [1,2]
Output: [2,2]
Constraints:
The number of nodes in both trees is in the range [0, 2000].
9K
42
12 Online
Code
Auto
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
if (root1 == NULL && root2 == NULL)
return NULL;
if (root1 == NULL)
return root2;
if (root2 == NULL)
return root1;
struct TreeNode* root = (struct TreeNode*)malloc(sizeof(struct TreeNode));
root->val = root1->val + root2->val;
root->left = mergeTrees(root1->left, root2->left);
root->right = mergeTrees(root1->right, root2->right);
Saved
Ln 19, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root1 =
[1,3,2,5]
root2 =
[2,1,3,null,4,null,7]
Output
[3,4,5,5,4,null,7]
Expected
[3,4,5,5,4,null,7]
Activate Windows
Go to Settings to activate Windows.

Lab 9a:

Write a program to traverse a graph using BFS method.

```
#include<stdio.h>

int graph [20][20],visited[20],n;

void BFS(int start){
  int queue[20],front =0,rear = 0;

  visited[start] = 1;
  queue[rear++]= start;

  while(front < rear) {
     int node = queue[front++];
     printf("%d ",node);
     for (int i=0;i<n;i++){
        if(graph[node][i]==1 && !visited[i]){
           visited[i]=1;
           queue[rear++] = i;
        }
     }
  }
}

int main(){
  int start;
  printf("Enter number of vertices: ");
  scanf("%d", &n);
  printf("Enter adjacency matrix:\n");
  for(int i=0;i<n;i++){
     for(int j =0;j<n;j++){
        scanf("%d",&graph[i][j]);
     }
  }
  for(int i =0;i<n;i++){
     visited[i] = 0;
  }
  printf("Enter starting vertex: ");
  scanf("%d",&start);

  printf("BFS Traversal: ");

  BFS(start);
  return 0;
}
```

Output:

```
Enter number of vertices: 5
Enter adjacency matrix:
0 1 0 1 0
1 0 1 0 1
0 0 1 0 0
1 0 0 0 1
0 1 0 1 0
Enter starting vertex: 0
BFS Traversal: 0 1 3 2 4
Process returned 0 (0x0)   execution time : 54.092 s
Press any key to continue.
```

Lab 9b:

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

```
#include <stdio.h>
#define MAX 10

int visited[MAX];
int adj[MAX][MAX];
int n;
void DFS(int v){
  visited [v] = 1;
  printf("%d",v);
  for(int i=0; i<n;i++){
     if(adj[v][i]==1 && !visited[i]){
        DFS(i);
     }
  }
}

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

  printf("Enter adjacency matrix:\n");
  for(int i=0;i<n;i++){
     for(int j =0;j<n;j++){
        scanf("%d",&adj[i][j]);
     }
  }

  for(int i =0;i<n;i++){
     visited[i] =0;
  }
  printf("DFS Traversal starting from vertex 0:\n");
  DFS(0);

  return 0;
}
```

Output:

```
Enter number of vertices: 5
Enter adjacency matrix:
0 1 0 1 0
1 0 1 0 1
0 0 1 0 0
1 0 0 0 1
0 1 0 1 0
DFS Traversal starting from vertex 0:
01243
Process returned 0 (0x0)   execution time : 28.406 s
Press any key to continue.
```

Lab 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 MAX 20

int hashTable[MAX];
int m;

/* Function to insert key using Linear Probing */
void insert(int key)
{
  int index = key % m;

  if (hashTable[index] == -1)
  {
     hashTable[index] = key;
  }
  else
  {
     int i = 1;
     while (hashTable[(index + i) % m] != -1)
     {
        i++;
     }
     hashTable[(index + i) % m] = key;
  }
}

/* Function to display hash table */
void display()
{
  printf("\nHash Table:\n");
  for (int i = 0; i < m; i++)
  {
     if (hashTable[i] != -1)
        printf("Address %d : %d\n", i, hashTable[i]);
     else
```

```
            printf("Address %d : Empty\n", i);
    }
}

int main()
{
    int n, key;

    printf("Enter size of hash table (m): ");
    scanf("%d", &m);

    printf("Enter number of employee records: ");
    scanf("%d", &n);

    /* Initialize hash table */
    for (int i = 0; i < m; i++)
        hashTable[i] = -1;

    printf("Enter %d employee keys (4-digit):\n", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &key);
        insert(key);
    }

    display();

    return 0;
}
```

Output:

```
Enter size of hash table (m):  10
Enter number of employee records: 5
Enter 5 employee keys (4-digit):
1234
2314
8900
7890
6785

Hash Table:
Address 0 : 8900
Address 1 : 7890
Address 2 : Empty
Address 3 : Empty
Address 4 : 1234
Address 5 : 2314
Address 6 : 6785
Address 7 : Empty
Address 8 : Empty
Address 9 : Empty

Process returned 0 (0x0)   execution time : 22.347 s
Press any key to continue.
```